DATABASE MANAGEMENT SYSTEMS

STUDENT MANAGEMENT SYSTEM

Project Designed and Curated by:

Register Number	Name of the Student
RA2111026050016	Challa Aashlesha

LIST OF EXPERIMENTS & SCHEDULE

Exp. No.	Title
1	Data Definition Language (DDL) commands in RDBMS
2	Data Manipulation Language (DML) and DataControl Language (DCL)
3	High level language extensions with cursors
4	High level language extension with Triggers
5	Procedures and Functions
6	Embedded SQL
7	Database design using E-R modeland Normalization
8	Design and implementation of Studentmanagement system

EX.NO:1 IMPLEMENTATION OF DDL COMMANDS

1. Create command:

   The Create Table Command: - it defines each column of the table uniquely. Each column has minimum of three attributes, a name, data type and size.

Syntax:

Create table <table name> (<col1> <datatype>(<size>), <col2> <datatype><size>));

```
mysql> use student;
Database changed
mysql> CREATE TABLE attendence (
    ->   aid int(11) NOT NULL,
    ->   rollno varchar(20) NOT NULL,
    ->   attendance int(100) NOT NULL
    -> );
Query OK, 0 rows affected, 2 warnings (0.07 sec)
```

2. Modifying the structure of the tables:
   a. Add new columns:

      Syntax:

      Alter table <table name> add (<new col><datatype(size), <new col>datatype(size));

```
mysql> ALTER TABLE student
    ->   ADD PRIMARY KEY (id);
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

b. Dropping a column from the table:

Syntax:

Alter table <table name> drop column <col>;

```
mysql> alter table student
    -> drop column number;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

c. Modifying existing columns:

Syntax:

Alter table <table name> modify(<col><new data type> (<new size>));

```
mysql> ALTER TABLE student
    -> MODIFY gender VARCHAR(3);
Query OK, 0 rows affected (0.07 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

d. Rename the table:

Syntax:

Rename <old table> to <new table>;

```
mysql> ALTER TABLE student
    -> RENAME TO new_student;
Query OK, 0 rows affected (0.02 sec)
```

e. Truncate table

Syntax:

Truncate table <table name>;

```
mysql> TRUNCATE TABLE new_student;
Query OK, 0 rows affected (0.04 sec)
```

f. Destroying tables

Syntax:

Drop table <table name>;

```
mysql> DROP TABLE new_student;
Query OK, 0 rows affected (0.03 sec)
```

Result:

Thus, the DDL Commands for the Student management system has been successfully executed.

EX.NO:2 IMPLEMENTATION OF DML AND DCL COMMANDS

- **Implementation of DML Commands:**

1. Insert Command:

   Syntax:

   INSERT INTO table name (column1, column2, column3, …)
   VALUES (value1, value2, value3, …);

```
mysql> INSERT INTO new_student (id, rollno, sname, sem, gender, branch, email, number, address)
    -> VALUES
    -> (1, '2023001', 'John Doe', 3, 'Male', 'Computer Science', 'john.doe@example.com', '1234567890', '123 Main St, City'),
    -> (2, '2023002', 'Jane Smith', 2, 'Female', 'Electrical Engineering', 'jane.smith@example.com', '9876543210', '456 Elm St, Town');
Query OK, 2 rows affected (0.02 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

2. Select Command:

   Syntax:

   SELECT column1, column2, …
   FROM table name
   WHERE condition;

```
mysql> select * from new_student;
+----+---------+------------+-----+--------+----------------------+-----------------------+------------+-------------------+
| id | rollno  | sname      | sem | gender | branch               | email                 | number     | address           |
+----+---------+------------+-----+--------+----------------------+-----------------------+------------+-------------------+
|  1 | 2023001 | John Doe   |   3 | Male   | Computer Science     | john.doe@example.com  | 1234567890 | 123 Main St, City |
|  2 | 2023002 | Jane Smith |   2 | Female | Electrical Engineering | jane.smith@example.com | 9876543210 | 456 Elm St, Town  |
+----+---------+------------+-----+--------+----------------------+-----------------------+------------+-------------------+
2 rows in set (0.00 sec)
```

3. Update Command:

   Syntax:

   UPDATE table name
   SET column1 = value1, column2 = value2, …
   WHERE condition;

```
mysql> UPDATE  new_student
    -> SET sname = 'John Deo ', sem = 3
    -> WHERE rollno = '2023001';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from new_student;
+----+---------+------------+-----+--------+----------------------+-----------------------+------------+-------------------+
| id | rollno  | sname      | sem | gender | branch               | email                 | number     | address           |
+----+---------+------------+-----+--------+----------------------+-----------------------+------------+-------------------+
|  1 | 2023001 | John Deo   |   3 | Male   | Computer Science     | john.doe@example.com  | 1234567890 | 123 Main St, City |
|  2 | 2023002 | Jane Smith |   2 | Female | Electrical Engineering | jane.smith@example.com | 9876543210 | 456 Elm St, Town  |
+----+---------+------------+-----+--------+----------------------+-----------------------+------------+-------------------+
2 rows in set (0.00 sec)
```

4. Delete Command:

   Syntax:

   DELETE FROM table name

   WHERE condition;

```
mysql> DELETE FROM new_student
    -> WHERE rollno = '2023001';
Query OK, 1 row affected (0.01 sec)

mysql> select * from new_student;
+----+---------+------------+-----+--------+-----------------------+-----------------------+------------+------------------+
| id | rollno  | sname      | sem | gender | branch                | email                 | number     | address          |
+----+---------+------------+-----+--------+-----------------------+-----------------------+------------+------------------+
|  2 | 2023002 | Jane Smith |   2 | Female | Electrical Engineering | jane.smith@example.com | 9876543210 | 456 Elm St, Town |
+----+---------+------------+-----+--------+-----------------------+-----------------------+------------+------------------+
1 row in set (0.00 sec)
```

- **Implementation of DCL Commands:**

1. Grant Command:

   Syntax:

   GRANT privileges

   ON database_name.table_name

   TO 'username '@'hostname' IDENTIFIED BY 'password';

```
mysql> GRANT INSERT, SELECT, UPDATE, DELETE ON new_student.* TO 'new_student'@'localhost';
Query OK, 0 rows affected (0.02 sec)
```

2. Revoke Command:

   Syntax:

   REVOKE privileges

   ON database_name.table_name

   FROM 'username '@'hostname';

```
mysql> REVOKE INSERT, SELECT, UPDATE, DELETE ON new_student.* FROM 'new_student'@'localhost';
Query OK, 0 rows affected (0.01 sec)
```

- **Implementation of TCL Commands:**

1. Commit Command:

   Syntax:

   COMMIT;

```
mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)
```

2. Rollback Command;

Syntax:

ROLLBACK;

```
mysql> GRANT INSERT, SELECT, UPDATE, DELETE ON new_student.* TO 'new_student'@'localhost';
Query OK, 0 rows affected (0.02 sec)

mysql> REVOKE INSERT, SELECT, UPDATE, DELETE ON new_student.* FROM 'new_student'@'localhost';
Query OK, 0 rows affected (0.01 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)

mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)
```

3. Savepoint Command:

Syntax:

SAVEPOINT savepoint_name;

```
mysql> -- Set a savepoint before updating another field
mysql> SAVEPOINT before_fee_update;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> -- Update another field (for example, update the fee)
mysql> UPDATE student
    -> SET Fee = 3500.00
    -> WHERE SName = 'John Doe';
ERROR 1054 (42S22): Unknown column 'Fee' in 'field list'
mysql>
mysql> -- If something goes wrong, rollback to the savepoint
mysql> ROLLBACK TO SAVEPOINT before_fee_update;
Query OK, 0 rows affected (0.00 sec)
```

Result:

Thus, the DML, DCL, TCL commands on the Student management system has been successfully executed.

EX.NO:3 IMPLEMENTATION OF CURSORS

PL/SQL Cursor Implementation Program:

```
DELIMITER //

CREATE PROCEDURE display_student_data()
BEGIN
   -- Declare variables to store column values fetched by the cursor
   DECLARE v_student_name VARCHAR(255);
   DECLARE v_student_id INT;
   DECLARE v_course VARCHAR(255);
   DECLARE v_department VARCHAR(255);
   DECLARE v_semester INT;
   DECLARE v_email VARCHAR(255);
   DECLARE v_phone_number VARCHAR(15);
   -- Declare a variable to handle the NOT FOUND condition
   DECLARE no_more_rows BOOLEAN DEFAULT FALSE;
   -- Declare a cursor for selecting data from the student table
   DECLARE student_cursor CURSOR FOR
      SELECT StudentName, StudentID, Course, Department, Semester, Email,
PhoneNumber
      FROM student_entry;
   -- Declare exit handler for cursor
   DECLARE CONTINUE HANDLER FOR NOT FOUND SET no_more_rows = TRUE;

   -- Open the cursor
   OPEN student_cursor;

   -- Fetch data from the cursor into variables and display
   read_loop: LOOP
      FETCH student_cursor INTO v_student_name, v_student_id, v_course,
v_department, v_semester, v_email, v_phone_number;
      IF no_more_rows THEN
         LEAVE read_loop;
      END IF;
      -- Display fetched data
      SELECT CONCAT('Student Name: ', v_student_name, ', Student ID: ',
v_student_id, ', Course: ', v_course,
                    ', Department: ', v_department, ', Semester: ', v_semester, ', Email: ',
```

```
v_email,
                    ', Phone Number: ', v_phone_number) AS Output;
    END LOOP;

    -- Close the cursor
    CLOSE student_cursor;
END //

DELIMITER ;
```

Output:

```
mysql> DELIMITER //
mysql>
mysql> CREATE PROCEDURE display_student_data()
    -> BEGIN
    ->     -- Declare variables to store column values fetched by the cursor
    ->     DECLARE v_student_name VARCHAR(255);
    ->     DECLARE v_student_id INT;
    ->     DECLARE v_course VARCHAR(255);
    ->     DECLARE v_department VARCHAR(255);
    ->     DECLARE v_semester INT;
    ->     DECLARE v_email VARCHAR(255);
    ->     DECLARE v_phone_number VARCHAR(15);
    ->     -- Declare a variable to handle the NOT FOUND condition
    ->     DECLARE no_more_rows BOOLEAN DEFAULT FALSE;
    ->     -- Declare a cursor for selecting data from the student table
    ->     DECLARE student_cursor CURSOR FOR
    ->         SELECT StudentName, StudentID, Course, Department, Semester, Email, PhoneNumber
    ->         FROM student_entry;
    ->     -- Declare exit handler for cursor
    ->     DECLARE CONTINUE HANDLER FOR NOT FOUND SET no_more_rows = TRUE;
    ->
    ->     -- Open the cursor
    ->     OPEN student_cursor;
    ->
    ->     -- Fetch data from the cursor into variables and display
    ->     read_loop: LOOP
    ->         FETCH student_cursor INTO v_student_name, v_student_id, v_course, v_department, v_semester, v_email, v_phone_number;
    ->         IF no_more_rows THEN
    ->             LEAVE read_loop;
    ->         END IF;
    ->         -- Display fetched data
    ->         SELECT CONCAT('Student Name: ', v_student_name, ', Student ID: ', v_student_id, ', Course: ', v_course,
    ->                       ', Department: ', v_department, ', Semester: ', v_semester, ', Email: ', v_email,
    ->                       ', Phone Number: ', v_phone_number) AS Output;
    ->     END LOOP;
    ->
    ->     -- Close the cursor
    ->     CLOSE student_cursor;
    -> END //
Query OK, 0 rows affected (0.02 sec)
```

Result:

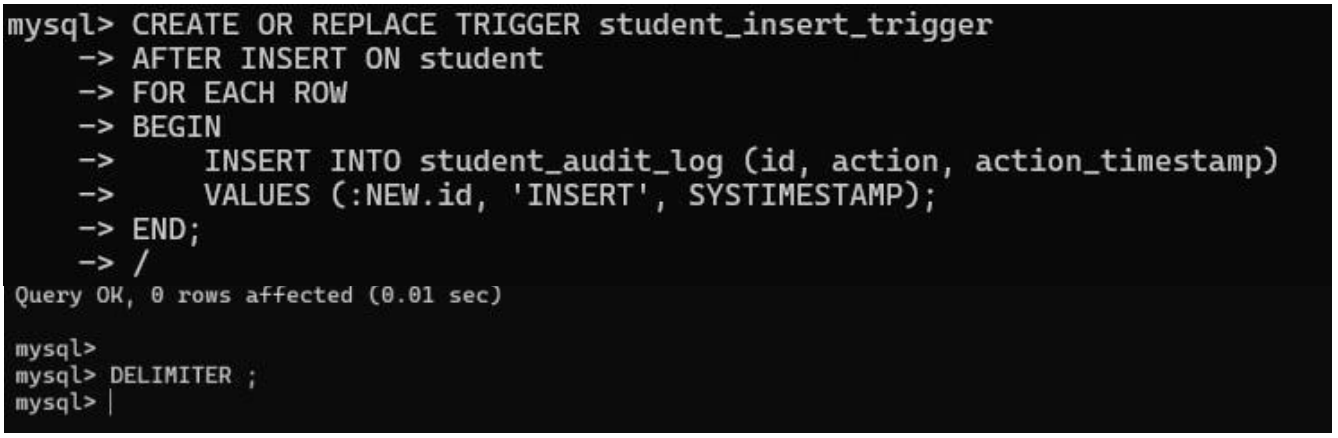
Thus, PL/SQL cursor implementation on Student management system has been successfully implemented.

EX.NO:4 IMPLEMENTATION OF TRIGGERS

PL/SQL Trigger Implementation Program:

```
 DELIMITER //
CREATE OR REPLACE TRIGGER student_insert_trigger
AFTER INSERT ON student
FOR EACH ROW
BEGIN
   INSERT INTO student_audit_log (id, action, action_timestamp)
   VALUES (:NEW.id, 'INSERT', SYSTIMESTAMP);
END;
/
 DELIMITER ;
```

Output:



Result:

Thus, PL/SQL Trigger program on Student management system has been successfully implemented.

EX.NO:5 IMPLEMENTATION OF PROCEDURES & FUNCTIONS

- **Implementation of Procedures:**

1. Create a procedure for insertion of new record.

```
DELIMITER //
CREATE PROCEDURE insert_student_entry (
   IN p_student_name VARCHAR(255),
   IN p_student_id INT,
   IN p_course VARCHAR(255),
   IN p_department VARCHAR(255),
   IN p_semester INT,
   IN p_email VARCHAR(255),
   IN p_phone_number VARCHAR(15)
)
BEGIN
   INSERT INTO student_entry (StudentName,
StudentID, Course, Department, Semester, Email,
PhoneNumber, RegistrationDate)
   VALUES (p_student_name, p_student_id, p_course,
p_department, p_semester, p_email, p_phone_number,
CURRENT_DATE());
END;
//
DELIMITER;
```

Output:

```
mysql> DELIMITER //
mysql>
mysql> CREATE PROCEDURE insert_student_entry (
    ->     IN p_student_name VARCHAR(255),
    ->     IN p_student_id INT,
    ->     IN p_course VARCHAR(255),
    ->     IN p_department VARCHAR(255),
    ->     IN p_semester INT,
    ->     IN p_email VARCHAR(255),
    ->     IN p_phone_number VARCHAR(15)
    -> )
    -> BEGIN
    ->     INSERT INTO student_entry (StudentName, StudentID, Course, Department, Semester, Email, PhoneNumber, RegistrationDate)
    ->     VALUES (p_student_name, p_student_id, p_course, p_department, p_semester, p_email, p_phone_number, CURRENT_DATE());
    -> END;
    -> //
Query OK, 0 rows affected (0.02 sec)

mysql>
mysql> DELIMITER ;
```

2. Create a procedure for updating a record

```
DELIMITER //
CREATE PROCEDURE
update_fee_by_student_name (
IN p_student_name VARCHAR(255),
   IN p_new_fee DECIMAL(10, 2)
)
BEGIN
UPDATE student_entry SET Fee =
p_new_fee
WHERE StudentName = p_student_name;
END;
//
DELIMITER ;
```

Output:

```
mysql> DELIMITER //
mysql>
mysql> CREATE PROCEDURE update_fee_by_student_name (
    ->     IN p_student_name VARCHAR(255),
    ->     IN p_new_fee DECIMAL(10, 2)
    -> )
    -> BEGIN
    ->     UPDATE student_entry SET Fee = p_new_fee
    ->     WHERE StudentName = p_student_name;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)
```

3. Create a procedure for deleting a record:

```
DELIMITER //

CREATE PROCEDURE
delete_student_entry_by_student_name (

  IN p_student_name VARCHAR(255)

)

BEGIN

  DELETE FROM student_entry WHERE
StudentName = p_student_name;

END;

//

DELIMITER;
```

Output:

```
mysql> DELIMITER //
mysql>
mysql> CREATE PROCEDURE delete_student_entry_by_student_name (
    ->     IN p_student_name VARCHAR(255)
    -> )
    -> BEGIN
    ->     DELETE FROM student_entry WHERE StudentName = p_student_name;
    -> END;
    -> //
Query OK, 0 rows affected (0.02 sec)
```

- **Implementation of Functions:**

1. A function to calculate the total fees based on the semester and base fee

```
DELIMITER //

CREATE FUNCTION calculate_total_fee(
   p_semester INT,
   p_base_fee DECIMAL(10, 2)
)
RETURNS DECIMAL(10, 2) DETERMINISTIC
BEGIN
   DECLARE total_fee DECIMAL(10, 2);

   -- Calculate the total fee based on the semester and base fee
   SET total_fee = p_semester * p_base_fee;

   RETURN total_fee;
END;
//

DELIMITER ;
```

Output:

```
mysql> DELIMITER //
mysql>
mysql> CREATE FUNCTION calculate_total_fee(
    ->     p_semester INT,
    ->     p_base_fee DECIMAL(10, 2)
    -> )
    -> RETURNS DECIMAL(10, 2) DETERMINISTIC
    -> BEGIN
    ->     DECLARE total_fee DECIMAL(10, 2);
    ->
    ->     -- Calculate the total fee based on the semester and base fee
    ->     SET total_fee = p_semester * p_base_fee;
    ->
    ->     RETURN total_fee;
    -> END;
    -> //
Query OK, 0 rows affected (0.02 sec)
```

2. A function to retrieve the course for the given student name:

```
DELIMITER //

CREATE FUNCTION get_course_by_student_name(
  p_student_name VARCHAR(255)
)
RETURNS VARCHAR(255) READS SQL DATA
BEGIN
  DECLARE student_course VARCHAR(255);

  -- Retrieve the course for the given student name
  SELECT Course INTO student_course
  FROM student_entry
  WHERE StudentName = p_student_name
  LIMIT 1;

  RETURN student_course;
END;
//

DELIMITER ;
```

Output:

```
mysql> DELIMITER //
mysql>
mysql> CREATE FUNCTION get_course_by_student_name(
    ->     p_student_name VARCHAR(255)
    -> )
    -> RETURNS VARCHAR(255) READS SQL DATA
    -> BEGIN
    ->     DECLARE student_course VARCHAR(255);
    ->
    ->     -- Retrieve the course for the given student name
    ->     SELECT Course INTO student_course
    ->     FROM student_entry
    ->     WHERE StudentName = p_student_name
    ->     LIMIT 1;
    ->
    ->     RETURN student_course;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> DELIMITER ;
```

3. A function to calculate the average fee for the given department:

```
DELIMITER //

CREATE FUNCTION calculate_average_fee_by_department(

  p_department VARCHAR(255)

)

RETURNS DECIMAL(10, 2) READS SQL DATA

BEGIN

  DECLARE avg_fee DECIMAL(10, 2);

  -- Calculate the average fee for the given department

  SELECT AVG(Fee) INTO avg_fee

  FROM student_entry

  WHERE Department = p_department;
```

```
    RETURN avg_fee;

END;

//

DELIMITER ;
```

Output:

```
mysql>
mysql> CREATE FUNCTION calculate_average_fee_by_department(
    ->     p_department VARCHAR(255)
    -> )
    -> RETURNS DECIMAL(10, 2) READS SQL DATA
    -> BEGIN
    ->     DECLARE avg_fee DECIMAL(10, 2);
    ->
    ->     -- Calculate the average fee for the given department
    ->     SELECT AVG(Fee) INTO avg_fee
    ->     FROM student_entry
    ->     WHERE Department = p_department;
    ->
    ->     RETURN avg_fee;
    -> END;
    -> //
Query OK, 0 rows affected (0.02 sec)
```

Result:

Thus, procedures and functions on the Student management system has been successfully implemented;

EX.NO:6 IMPLEMENTATION OF EMBEDDED SQL

1. Trigger to update the Fee on semesters:

```
DELIMITER //

CREATE TRIGGER update_fee_on_sem_update AFTER UPDATE ON
student

FOR EACH ROW

BEGIN

  IF OLD.Sem <> NEW.Sem THEN

    UPDATE student_entry

    SET Fee = NEW.Sem * 1000 -- Adjust the fee calculation as needed

    WHERE SName = NEW.SName;

  END IF;

END;

//

DELIMITER ;
```

Output:

```
mysql> DELIMITER //
mysql>
mysql> CREATE TRIGGER update_fee_on_sem_update AFTER UPDATE ON student
    -> FOR EACH ROW
    -> BEGIN
    ->     IF OLD.Sem <> NEW.Sem THEN
    ->         UPDATE student_entry
    ->         SET Fee = NEW.Sem * 1000 -- Adjust the fee calculation as needed
    ->         WHERE SName = NEW.SName;
    ->     END IF;
    -> END;
    -> //
Query OK, 0 rows affected (0.02 sec)
```

2. Trigger to insert a log record when a new student is inserted:

```
DELIMITER //

CREATE TRIGGER insert_log_on_student_insert

AFTER INSERT ON student FOR EACH ROW

BEGIN

   INSERT INTO student_log (Action, ID, RollNo, SName,

    Sem,Gender, Branch, Email, Number, Address)

   VALUES ('INSERT',NEW.ID, NEW.RollNo, NEW.SName,

    NEW.Sem, NEW.Gender, NEW.Branch, NEW.Email,NEW.Address,

    NOW());

END;

//

DELIMITER ;
```

Output:

```
mysql> CREATE TRIGGER insert_log_on_student_insert
    -> AFTER INSERT ON student FOR EACH ROW
    -> BEGIN
    ->     INSERT INTO student_log (Action, ID, RollNo, SName, Sem,Gender, Branch, Email, Number, Address)
    ->     VALUES ('INSERT',NEW.ID, NEW.RollNo, NEW.SName,  NEW.Sem, NEW.Gender, NEW.Branch, NEW.Email,NEW.Address, NOW());
    -> END;
    -> //
Query OK, 0 rows affected (0.02 sec)
```

3. Trigger to update a log record when a student is updated:

```
    DELIMITER //

    CREATE TRIGGER update_log_on_student_update

AFTER UPDATE ON student
```

```
    FOR EACH ROW
    BEGIN
      UPDATE student_log
      SET Action = 'UPDATE',
        ID = NEW.ID,
        RollNo = NEW.RollNo,
        SName = NEW.SName,
        Sem = NEW.Sem,
        Gender = NEW.Gender,
        Branch = NEW.Branch,
        Email = NEW.Email,
        Number  = NEW.Number,
        Address = NEW.Address -- Corrected column assignment
      WHERE SName = NEW.SName;
    END;
    //
DELIMITER ;
```

Output:

```
mysql> CREATE TRIGGER update_log_on_student_update
    -> AFTER UPDATE ON student
    -> FOR EACH ROW
    -> BEGIN
    ->     UPDATE student_log
    ->     SET Action = 'UPDATE',
    ->         ID = NEW.ID,
    ->         RollNo = NEW.RollNo,
    ->         SName = NEW.SName,
    ->         Sem = NEW.Sem,
    ->         Gender = NEW.Gender,
    ->         Branch = NEW.Branch,
    ->         Email = NEW.Email,
    ->         Number = NEW.Number,
    ->         Address = NEW.Address -- Corrected column assignment
    ->     WHERE SName = NEW.SName;
    -> END;
    -> //
Query OK, 0 rows affected (0.02 sec)
```

4. Trigger to delete a log record when a student is deleted:

```
DELIMITER //

CREATE TRIGGER delete_log_on_student_delete

BEFORE DELETE ON student

FOR EACH ROW

BEGIN

   DELETE FROM student_log WHERE SName = OLD.SName;

END;

//

DELIMITER ;
```

Output:

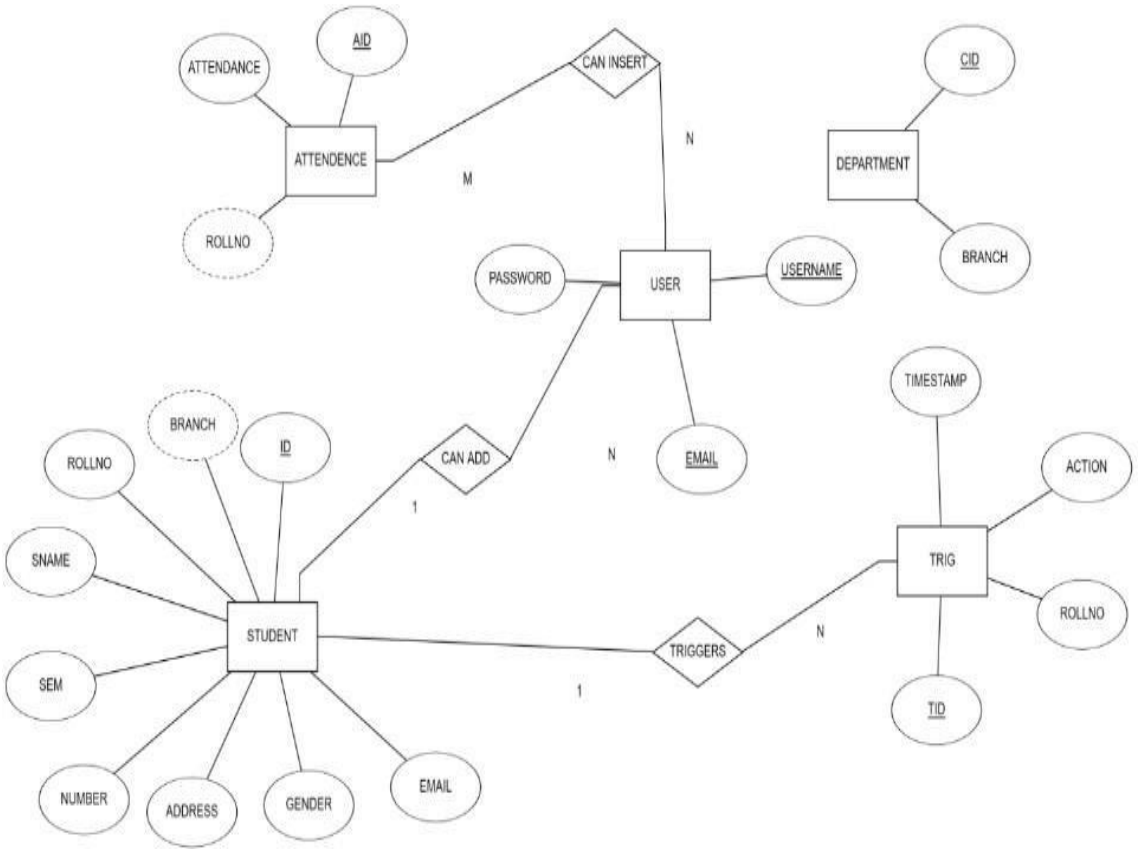
```
mysql> DELIMITER //
mysql>
mysql> CREATE TRIGGER delete_log_on_student_delete
    -> BEFORE DELETE ON student
    -> FOR EACH ROW
    -> BEGIN
    ->     DELETE FROM student_log WHERE SName = OLD.SName;
    -> END;
    -> //
Query OK, 0 rows affected (0.02 sec)
```

Result:

Thus, Embedded SQl functions and queries has been executed on the student management system.

EX.NO:7 IMPLEMENTATION OF ER DIAGRAM

The following is the Entity Relationship Diagram for Student management system:



Result:

Thus, the implementation of ER diagram for the student management system has been successfully executed.

EX.NO:8 IMPLEMENTATION OF STUDENT MANAGEMENT SYSTEM

BACKEND PYTHON WITH MYSQL CODE

```
from flask import
Flask,render_template,request,session,redirect,url_for,flash
from flask_sqlalchemy  import SQLAlchemy
from flask_login import UserMixin
from werkzeug.security import generate_password_hash,check_password_hash
from flask_login import
login_user,logout_user,login_manager,LoginManager from
flask_login import login_required,current_user
import json

# MY db
connection
local_server=
True
app = Flask(_name_)
app.secret_key='kusumachandash
wini'


# this is for getting unique
user access
login_manager=LoginManage
r(app)
login_manager.login_view='lo
gin'

@login_manager.user
_loaderdef
```

```
load_user(user_id):
    return  User.query.get(int(user_id))



#
app.config['SQLALCHEMY_DATABASE_URL']='mysql://username:password@localhost/d
atabas_table_ name'
app.config['SQLALCHEMY_DATABASE_URI']='mysql://root:@localhost/students'
db=SQLAlchemy(app)

# here we will create db models
that is tablesclass
Test(db.Model):
    id=db.Column(db.Integer,primary_key=True)
    name=db.Column(db.String(100))
    email=db.Column(db.String(100))
    class Department(db.Model):
    cid=db.Column(db.Integer,primary_key=True)
    branch=db.Column(db.String(100))

class Attendence(db.Model):
    aid=db.Column(db.Integer,primary_key=True)
    rollno=db.Column(db.String(100))
    attendance=db.Column(db.Integer())

class Trig(db.Model):
    tid=db.Column(db.Integer,primary_key=True)
    rollno=db.Column(db.String(100))
    action=db.Column(db.String(100))
    timestamp=db.Column(db.String(100))
```

```
class User(UserMixin,db.Model):
    id=db.Column(db.Integer,primary_key=True)
    username=db.Column(db.String(50))
    email=db.Column(db.String(50),unique=True)
    password=db.Column(db.String(1000))

class Student(db.Model):
    id=db.Column(db.Integer,primary_key=True)
    rollno=db.Column(db.String(50))
    sname=db.Column(db.String(50))
    sem=db.Column(db.Integer)
    gender=db.Column(db.String(50))
    branch=db.Column(db.String(50))
    email=db.Column(db.String(50))
    number=db.Column(db.String(12))
    address=db.Column(db.String(100))
@app.ro
ute('/')
def
index():
    return  render_template('index.html')

@app.route('/studen
tdetails') def
studentdetails():
    query=db.engine.execute(f"SELECT  * FROM
    `student`") return
    render_template('studentdetails.html',query=
    query)

@app.route('/tri
ggers') def
triggers():
```

```
    query=db.engine.execute(f"SELECT  * FROM
    `trig`") return
    render_template('triggers.html',query=query
    )

@app.route('/department',methods=['PO
ST','GET']) def department():
    if request.method=="POST":
        dept=request.form.get('dept')
        query=Department.query.filter_by(branch=
        dept).first() if query:
            flash("Department Already
            Exist","warning") return
            redirect('/department')
        dep=Department(branch=de
        pt) db.session.add(dep)
        db.session.commit()
        flash("Department
        Addes","success")
    return  render_template('department.html')

@app.route('/addattendance',methods=['PO
ST','GET']) def addattendance():
    query=db.engine.execute(f"SELECT  * FROM
    `student`") if request.method=="POST":
        rollno=request.form.get('rollno')
        attend=request.form.get('attend')
        print(attend,rollno)
        atte=Attendence(rollno=rollno,attendan
        ce=attend) db.session.add(atte)
        db.session.commit()
        flash("Attendance  added","warning")
    return  render_template('attendance.html',query=query)
```

```
@app.route('/search',methods=['PO
ST','GET']) def search():
    if request.method=="POST":
        rollno=request.form.get('roll')
        bio=Student.query.filter_by(rollno=rollno).f
        irst()
        attend=Attendence.query.filter_by(rollno=r
        ollno).first()
        return

    render_template('search.html',bio=bio,attend=attend

    )return  render_template('search.html')

@app.route("/delete/<string:id>",methods=['POS
T','GET']) @login_required
def delete(id):
    db.engine.execute(f"DELETE  FROM `student` WHERE
    `student`.`id`={id}") flash("Slot Deleted Successful","danger")
    return redirect('/studentdetails')


@app.route("/edit/<string:id>",methods=['POST'
,'GET']) @login_required
def edit(id):
    dept=db.engine.execute("SELECT * FROM `department`")
    posts=Student.query.filter_by(id=id).first()
    if request.method=="POST":
        rollno=request.form.get('rollno')
        sname=request.form.get('sname')
        sem=request.form.get('sem')
        gender=request.form.get('gender')
```

```
        branch=request.form.get('branch')
        email=request.form.get('email')
        num=request.form.get('num')
        address=request.form.get('address')
        query=db.engine.execute(f"UPDATE
        `student` SET
`rollno`='{rollno}',`sname`='{sname}',`sem`='{sem}',`gender`='{gender}',`branch`='{branch}',`
email`='{em ail}',`number`='{num}',`address`='{address}'")
        flash("Slot is
        Updates","success")
        return
        redirect('/studentdetail
        s')

    return    render_template('edit.html',posts=posts,dept=dept)


@app.route('/signup',methods=['PO
ST','GET']) def signup():
    if request.method == "POST":
        username=request.form.get('userna
        me')
        email=request.form.get('email')
        password=request.form.get('passwo
        rd')
        user=User.query.filter_by(email=em
        ail).first()if user:
            flash("Email Already
            Exist","warning") return
            render_template('/signup.ht
            ml')
        encpassword=generate_password_hash(password)
```

```
        new_user=db.engine.execute(f"INSERT INTO `user` (`username`,`email`,`password`)
VALUES ('{username}','{email}','{encpassword}')")

        # this is method 2 to save data in db
        #
        newuser=User(username=username,email=email,password=enc
        password)# db.session.add(newuser)
        # db.session.commit()
        flash("Signup Succes Please
        Login","success") return
        render_template('login.html')



    return  render_template('signup.html')

@app.route('/login',methods=['POS
T','GET']) def login():
    if request.method == "POST":
        email=request.form.get('email')
        password=request.form.get('password')
        user=User.query.filter_by(email=email).first
        ()
    if user and
        check_password_hash(user.password,passw
        ord): login_user(user) flash("Login
        Success","primary") return
        redirect(url_for('index')) else:
        flash("invalid credentials","danger") return
        render_template('login.html') return
        render_template('login.html')
        @app.route('/logout')  @login_required def
```

```
    logout(): logout_user() flash("Logout
    SuccessFul","warning") return
    redirect(url_for('login'))
    @app.route('/addstudent',methods=['POST
    ','GET']) @login_required def
    addstudent():
    dept=db.engine.execute("SELECT * FROM
    `department`") if
    request.method=="POST":
    rollno=request.form.get('rollno')
    sname=request.form.get('sname')
    sem=request.form.get('sem')
    gender=request.form.get('gender')
    branch=request.form.get('branch')
    email=request.form.get('email')
    num=request.form.get('num')
    address=request.form.get('address')
    query=db.engine.execute(f"INSERT   INTO
    `student`
    (`rollno`,`sname`,`sem`,`gender`,`branch
    `,`email`,`number`,`address`) VALUES
    ('{rollno}','{sname}','{sem}','{gender}','{bra
    nch}','{email}','{num}','{address}')")
    flash("Booking Confirmed","info")
    return
    render_template('student.html',dept=dept) @app.route('/test') def test():
    try:
    Test.query.all()
    return 'My database is
  Connected' except:
    return 'My db is not Connected'

app.run(debug=True)
```

FRONTEND PYTHON WITH MYSQL CODE

```
<!DOCTYPE html>
<html lang="en">

<head>
 <meta charset="utf-8">
 <meta content="width=device-width, initial-scale=1.0" name="viewport">

 <title>{% block title %}
 {% endblock title %}</title>
 <meta content="" name="description">
 <meta content="" name="keywords">

{% block style %}
{% endblock style %}
 <link
href="https://fonts.googleapis.com/css?family=Open+Sans:300,300i,400,400i,70
0,700i|Raleway:3 00,400,500,700,800" rel="stylesheet">

 <!-- Vendor CSS Files -->
 <link href="static/assets/vendor/bootstrap/css/bootstrap.min.css" rel="stylesheet">
 <link href="static/assets/vendor/venobox/venobox.css" rel="stylesheet">
 <link href="static/assets/vendor/font-awesome/css/font-awesome.min.css"
 rel="stylesheet">
 <link href="static/assets/vendor/owl.carousel/assets/owl.carousel.min.css"
 rel="stylesheet">
 <link href="static/assets/vendor/aos/aos.css" rel="stylesheet">

 <!-- Template Main CSS File -->
 <link href="static/assets/css/style.css" rel="stylesheet">
```

```
</head>

<body>
 <header id="header">
  <div class="container">

   <div id="logo" class="pull-left">

    <a href="/" class="scrollto">S.M.S</a>
   </div>

   <nav id="nav-menu-container">
    <ul class="nav-menu">
     <li class="{% block home %}
     {% endblock home %}"><a href="/">Home</a></li>

<li><a href="/addstudent">Students</a></li>
<li><a href="/addattendance">Attendance</a></li>
<li><a href="/department">Department</a></li>
<li><a href="/triggers">Records</a></li>
<li><a href="/studentdetails">Student Details</a></li>
<li><a href="/search">Search</a></li>


     <li><a href="/about">About</a></li>



  {% if current_user.is_authenticated  %}
     <li class="buy-tickets"><a href="">Welcome</a></li>
      <li class="buy-tickets"><a href="/logout">Logout</a></li>
     {% else %}
```

```
        <li class="buy-tickets"><a href="/signup">Signin</a></li>

        {% endif %}
      </ul>
    </nav><!-- #nav-menu-container -->
  </div>
 </header><!-- End Header -->

 <!-- ======= Intro Section ======= -->
<section id="intro">
  <div class="intro-container" data-aos="zoom-in" data-aos-delay="100">
    <h1 class="mb-4 pb-0">STUDENT MANAGEMENT SYSTEM </span> </h1>
    <p class="mb-4 pb-0">DBMS Mini Project Using Flask & MYSQL</p>

    <a href="" class="about-btn scrollto">View More</a>
  </div>
 </section><!-- End Intro Section -->
 <main id="main">


 {% block body %}

{% 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}}


</div>
```

```
 {% endfor %}
 {% endif %}
 {% endwith %}
 {% endblock body %}

 <a href="#" class="back-to-top"><i class="fa fa-angle-up"></i></a>
 <!-- Vendor JS Files -->
 <script src="static/assets/vendor/jquery/jquery.min.js"></script>
 <script src="static/assets/vendor/bootstrap/js/bootstrap.bundle.min.js"></script>
 <script src="static/assets/vendor/jquery.easing/jquery.easing.min.js"></script>
 <script src="static/assets/vendor/php-email-form/validate.js"></script>
 <script src="static/assets/vendor/venobox/venobox.min.js"></script>
 <script src="static/assets/vendor/owl.carousel/owl.carousel.min.js"></script>
 <script  src="static/assets/vendor/superfish/superfish.min.js"></script>
 <script  src="static/assets/vendor/hoverIntent/hoverIntent.js"></script>
 <script src="static/assets/vendor/aos/aos.js"></script>


 <!-- Template Main JS File -->
 <script src="static/assets/js/main.js"></script>

</body>

</html>
```

2.Studen

ts.html

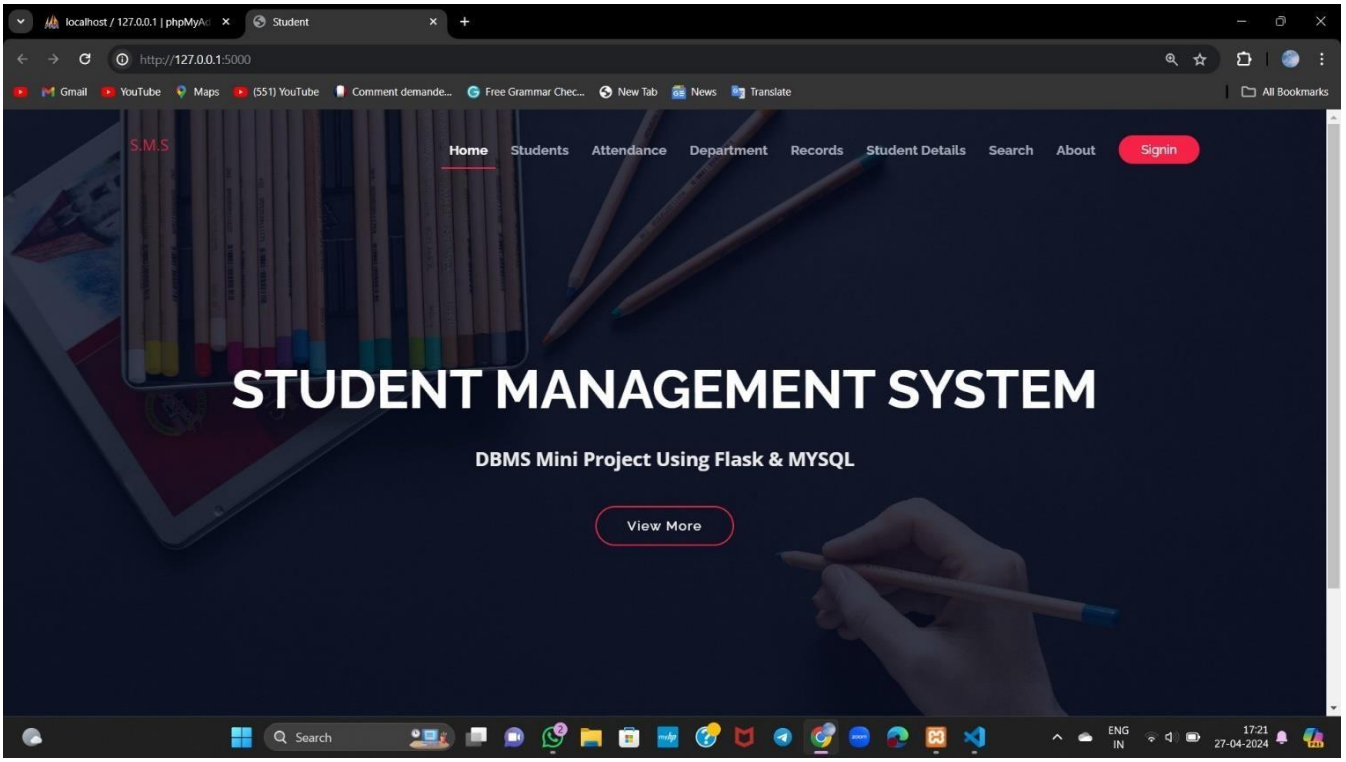
```
{% extends 'base.html' %}
{% block
title %}
Add
Students
```

```
{% endblock title %}

{% block body %}
 <h3 class="text-center"><span>Add Student Details</span> </h3>

{% 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}}



</div>
 {% endfor %}
 {% endif %}
 {% endwith %}
<br>
<div class="container">

<div class="row">

<div class="col-md-4"></div>
<div class="col-md-4">

<form action="/addstudent" method="post">
<div class="form-group">

<label for="rollno">Roll Number</label>
<input type="text" class="form-control" name="rollno" id="rollno">
</div>
<br>
```

```
<div class="form-group">

<label for="sname">Student Name</label>
<input type="text" class="form-control" name="sname" id="sname">
</div>
<br>
<div class="form-group">

<label for="sem">Sem</label>
<input type="number" class="form-control" name="sem" id="sem">
</div>
<br>
<div class="form-group">
<select  class="form-control" id="gender" name="gender"  required>
     <option selected>Select Gender</option>

     <option value="male">Male</option>
     <option value="female">Female</option>

   </select>
 </div>
<br>

<div class="form-group">
<select class="form-control" id="branch" name="branch"  required>
     <option selected>Select Branch</option>
     {% for d in dept  %}
     <option value="{{d.branch}}">{{d.branch}}</option>
     {% endfor %}
   </select>
 </div>
```

```
<br>
<div class="form-group">

<label for="email">Email</label>
<input type="email" class="form-control" name="email" id="email">
</div>
<br>
<div class="form-group">
<label for="num">Phone Number</label>
<input type="number" class="form-control" name="num" id="num">
</div>
<br>

<div class="form-group">
<label for="address">Address</label>
<textarea class="form-control" name="address" id="address"></textarea>
</div>
<br>
<button type="submit" class="btn btn-danger btn-sm btn-block">Add Record</button>
</form>
<br>
<br>
</div>
<div class="col-md-4"></div>
</div></div>
{% endblock body %}
```

Output:



Result:

Thus, the implementation of the student management system has been successfully executed