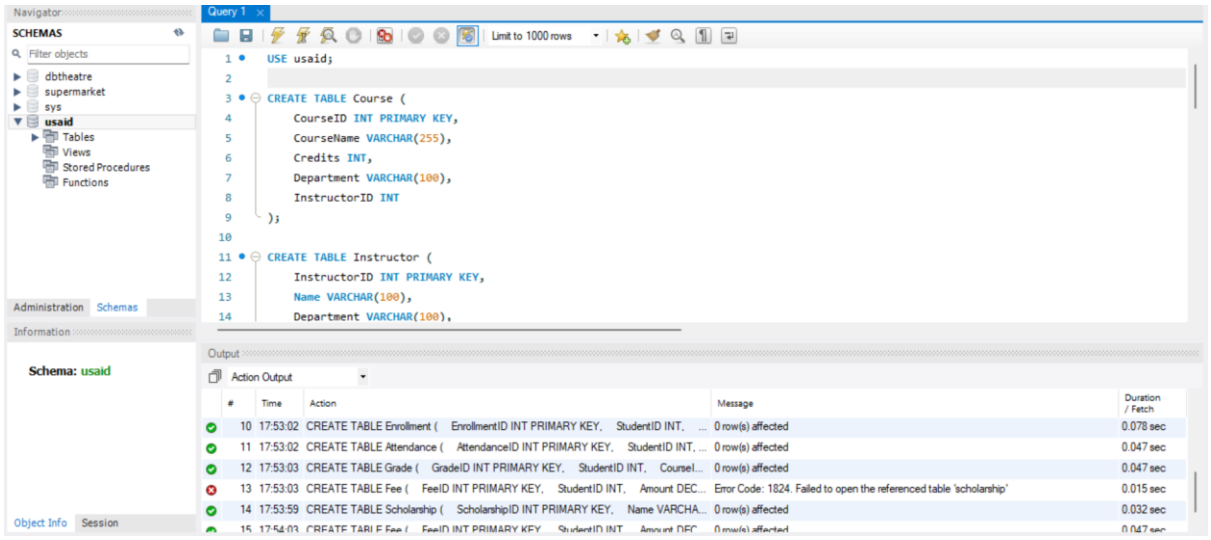
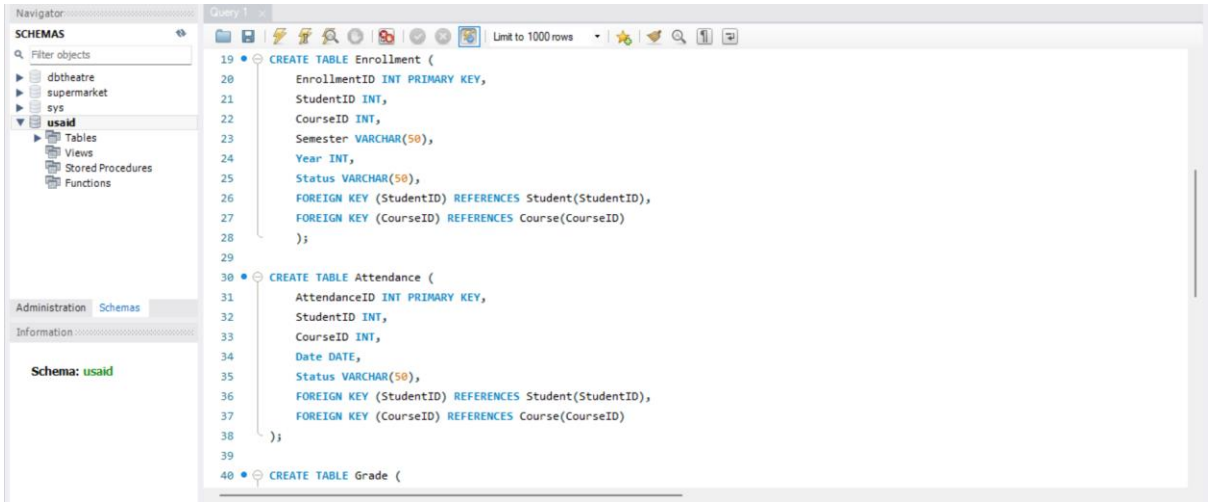
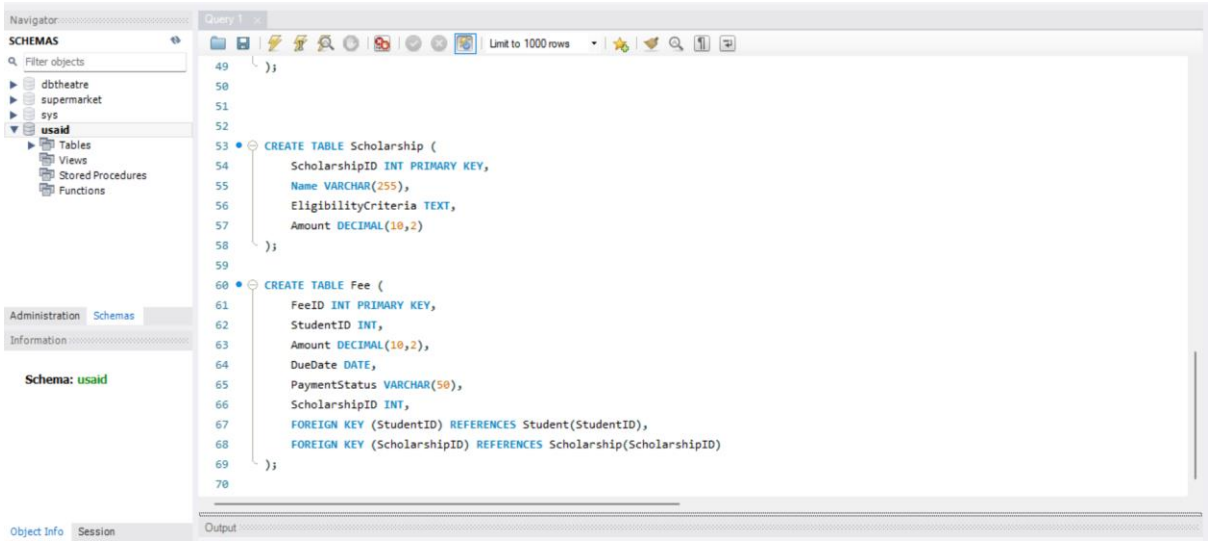
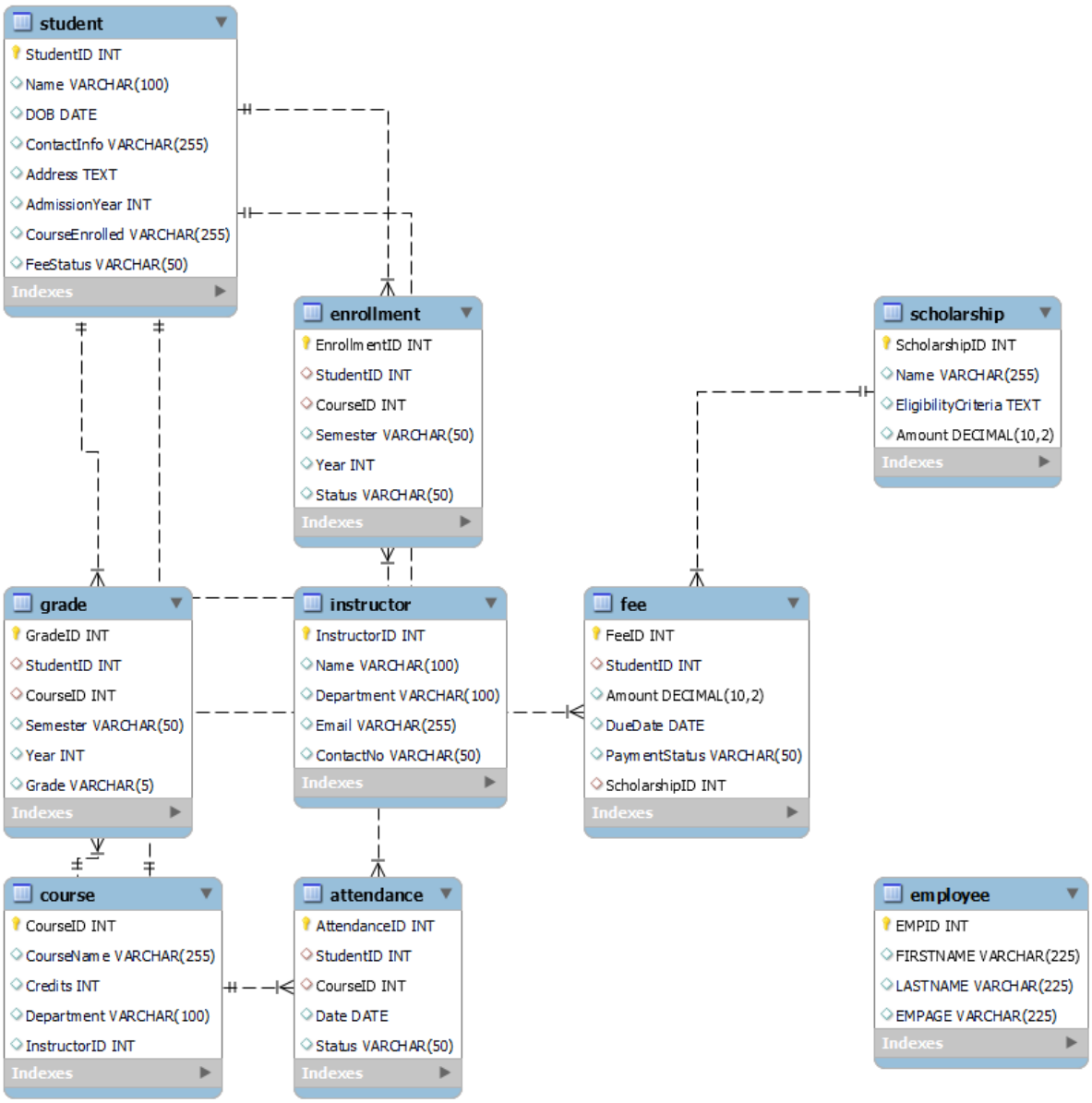
Step 1 (20%); Using your Relational Schema Diagram developed in Part 2,







PHYSICAL MODEL EER DIAGRAM



Step 2 (5%): Forward Engineer

SQL CODE AFTER FORWARD ENGINEERING

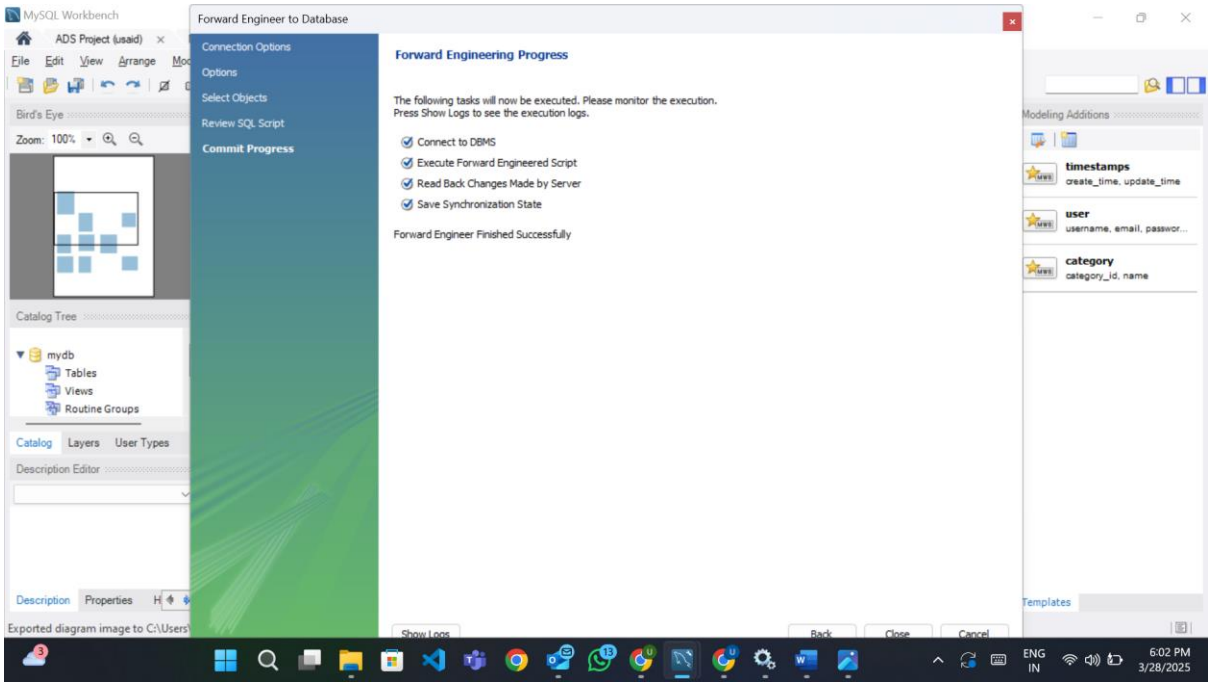
```
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_
DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- -----------------------------------------------------
-- Schema mydb
-- -----------------------------------------------------
-- -----------------------------------------------------
-- Schema usaid
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema usaid
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `usaid` DEFAULT CHARACTER SET utf8mb4 COLLATE
utf8mb4_0900_ai_ci ;
USE `usaid` ;

-- -----------------------------------------------------
-- Table `usaid`.`student`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `usaid`.`student` (
  `StudentID` INT NOT NULL,
  `Name` VARCHAR(100) NULL DEFAULT NULL,
  `DOB` DATE NULL DEFAULT NULL,
  `ContactInfo` VARCHAR(255) NULL DEFAULT NULL,
  `Address` TEXT NULL DEFAULT NULL,
  `AdmissionYear` INT NULL DEFAULT NULL,
  `CourseEnrolled` VARCHAR(255) NULL DEFAULT NULL,
  `FeeStatus` VARCHAR(50) NULL DEFAULT NULL,
  PRIMARY KEY (`StudentID`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
```

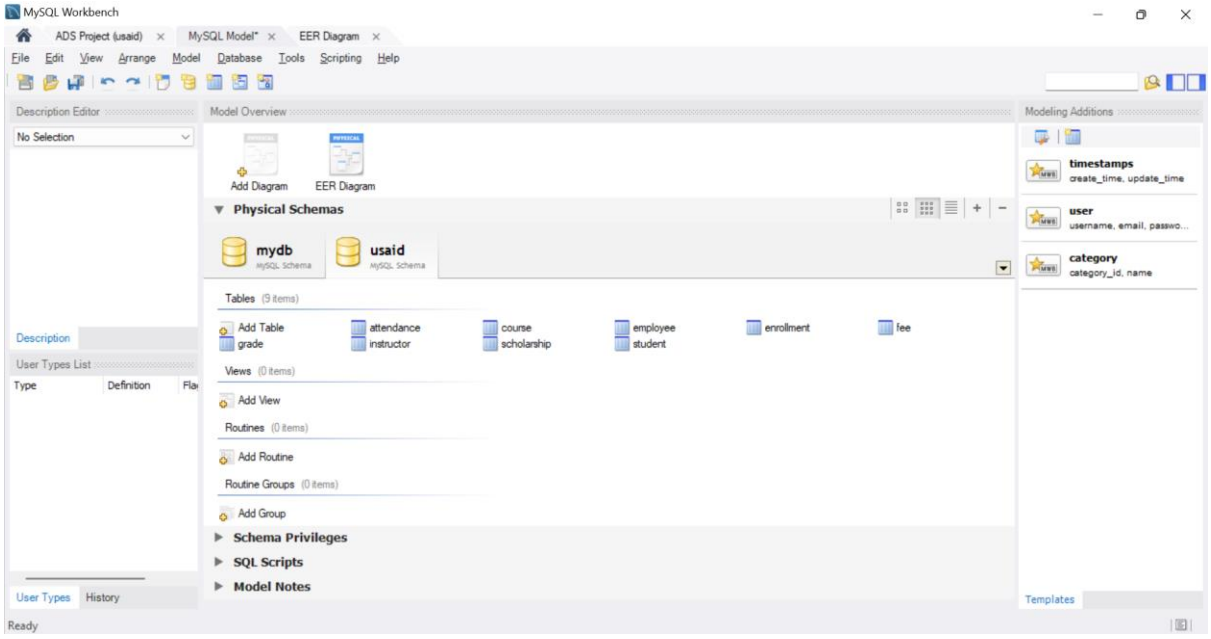
```
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `usaid`.`course`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `usaid`.`course` (
  `CourseID` INT NOT NULL,
  `CourseName` VARCHAR(255) NULL DEFAULT NULL,
  `Credits` INT NULL DEFAULT NULL,
  `Department` VARCHAR(100) NULL DEFAULT NULL,
  `InstructorID` INT NULL DEFAULT NULL,
  PRIMARY KEY (`CourseID`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `usaid`.`attendance`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `usaid`.`attendance` (
  `AttendanceID` INT NOT NULL,
  `StudentID` INT NULL DEFAULT NULL,
  `CourseID` INT NULL DEFAULT NULL,
  `Date` DATE NULL DEFAULT NULL,
  `Status` VARCHAR(50) NULL DEFAULT NULL,
  PRIMARY KEY (`AttendanceID`),
  INDEX `StudentID` (`StudentID` ASC) VISIBLE,
  INDEX `CourseID` (`CourseID` ASC) VISIBLE,
  CONSTRAINT `attendance_ibfk_1`
    FOREIGN KEY (`StudentID`)
    REFERENCES `usaid`.`student` (`StudentID`),
  CONSTRAINT `attendance_ibfk_2`
    FOREIGN KEY (`CourseID`)
    REFERENCES `usaid`.`course` (`CourseID`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```

```
-- -----------------------------------------------------
-- Table `usaid`.`employee`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `usaid`.`employee` (
  `EMPID` INT NOT NULL,
  `FIRSTNAME` VARCHAR(225) NULL DEFAULT NULL,
  `LASTNAME` VARCHAR(225) NULL DEFAULT NULL,
  `EMPAGE` VARCHAR(225) NULL DEFAULT NULL,
  PRIMARY KEY (`EMPID`),
  UNIQUE INDEX `EMPAGE` (`EMPAGE` ASC) VISIBLE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `usaid`.`enrollment`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `usaid`.`enrollment` (
  `EnrollmentID` INT NOT NULL,
  `StudentID` INT NULL DEFAULT NULL,
  `CourseID` INT NULL DEFAULT NULL,
  `Semester` VARCHAR(50) NULL DEFAULT NULL,
  `Year` INT NULL DEFAULT NULL,
  `Status` VARCHAR(50) NULL DEFAULT NULL,
  PRIMARY KEY (`EnrollmentID`),
  INDEX `StudentID` (`StudentID` ASC) VISIBLE,
  INDEX `CourseID` (`CourseID` ASC) VISIBLE,
  CONSTRAINT `enrollment_ibfk_1`
    FOREIGN KEY (`StudentID`)
    REFERENCES `usaid`.`student` (`StudentID`),
  CONSTRAINT `enrollment_ibfk_2`
    FOREIGN KEY (`CourseID`)
    REFERENCES `usaid`.`course` (`CourseID`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `usaid`.`scholarship`
```

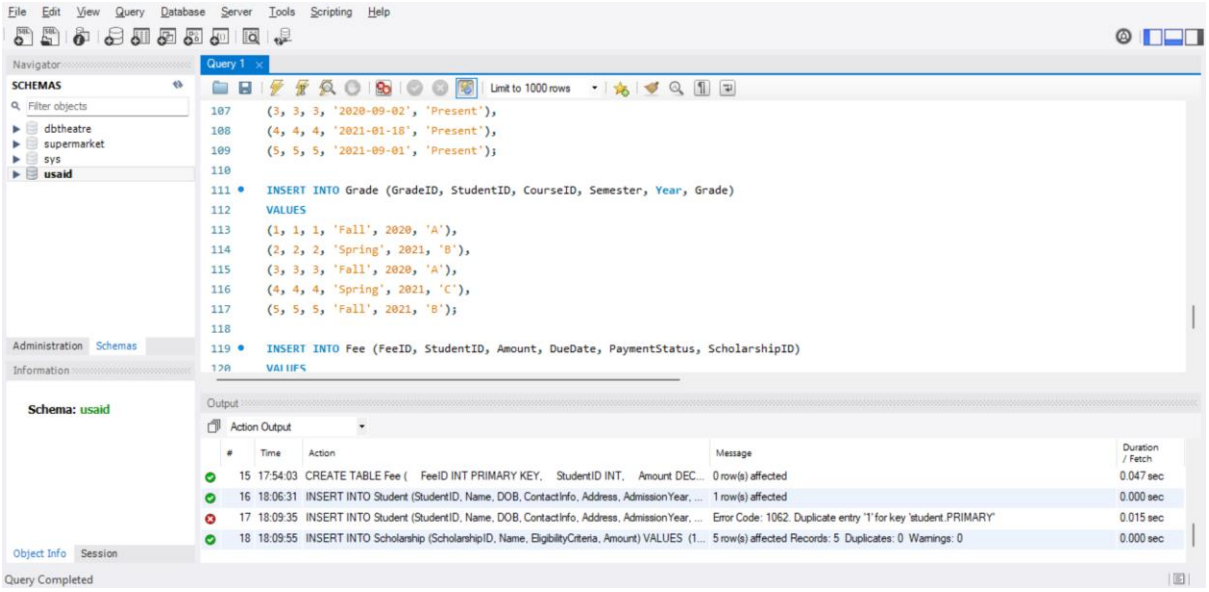
```
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `usaid`.`scholarship` (
  `ScholarshipID` INT NOT NULL,
  `Name` VARCHAR(255) NULL DEFAULT NULL,
  `EligibilityCriteria` TEXT NULL DEFAULT NULL,
  `Amount` DECIMAL(10,2) NULL DEFAULT NULL,
  PRIMARY KEY (`ScholarshipID`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `usaid`.`fee`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `usaid`.`fee` (
  `FeeID` INT NOT NULL,
  `StudentID` INT NULL DEFAULT NULL,
  `Amount` DECIMAL(10,2) NULL DEFAULT NULL,
  `DueDate` DATE NULL DEFAULT NULL,
  `PaymentStatus` VARCHAR(50) NULL DEFAULT NULL,
  `ScholarshipID` INT NULL DEFAULT NULL,
  PRIMARY KEY (`FeeID`),
  INDEX `StudentID` (`StudentID` ASC) VISIBLE,
  INDEX `ScholarshipID` (`ScholarshipID` ASC) VISIBLE,
  CONSTRAINT `fee_ibfk_1`
    FOREIGN KEY (`StudentID`)
    REFERENCES `usaid`.`student` (`StudentID`),
  CONSTRAINT `fee_ibfk_2`
    FOREIGN KEY (`ScholarshipID`)
    REFERENCES `usaid`.`scholarship` (`ScholarshipID`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `usaid`.`grade`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `usaid`.`grade` (
  `GradeID` INT NOT NULL,
```

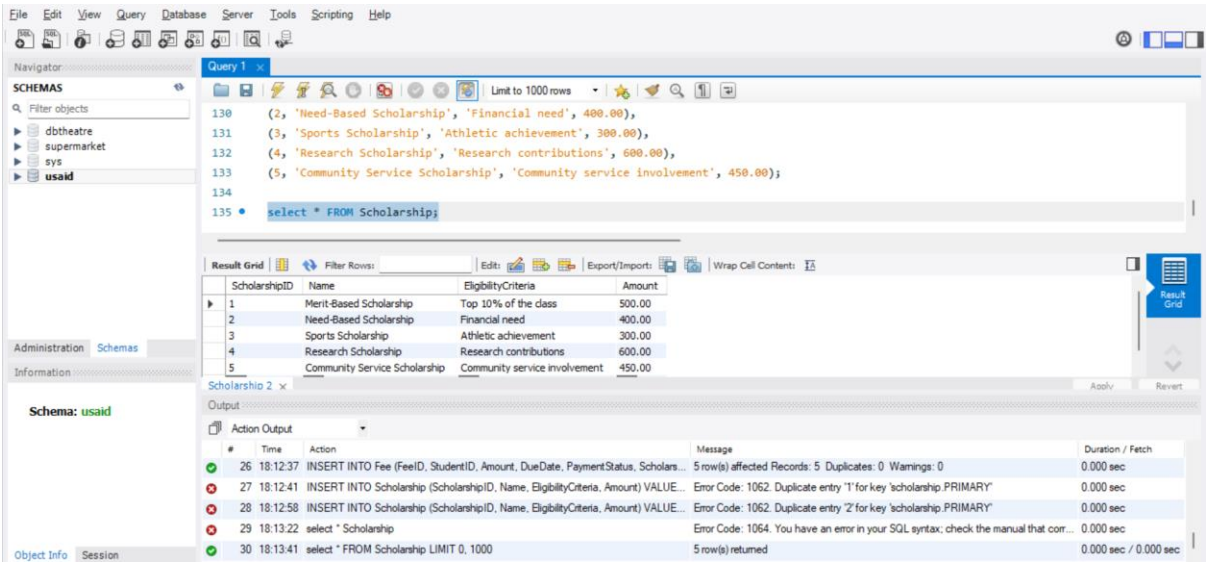
```
  `StudentID` INT NULL DEFAULT NULL,
  `CourseID` INT NULL DEFAULT NULL,
  `Semester` VARCHAR(50) NULL DEFAULT NULL,
  `Year` INT NULL DEFAULT NULL,
  `Grade` VARCHAR(5) NULL DEFAULT NULL,
  PRIMARY KEY (`GradeID`),
  INDEX `StudentID` (`StudentID` ASC) VISIBLE,
  INDEX `CourseID` (`CourseID` ASC) VISIBLE,
  CONSTRAINT `grade_ibfk_1`
    FOREIGN KEY (`StudentID`)
    REFERENCES `usaid`.`student` (`StudentID`),
  CONSTRAINT `grade_ibfk_2`
    FOREIGN KEY (`CourseID`)
    REFERENCES `usaid`.`course` (`CourseID`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `usaid`.`instructor`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `usaid`.`instructor` (
  `InstructorID` INT NOT NULL,
  `Name` VARCHAR(100) NULL DEFAULT NULL,
  `Department` VARCHAR(100) NULL DEFAULT NULL,
  `Email` VARCHAR(255) NULL DEFAULT NULL,
  `ContactNo` VARCHAR(50) NULL DEFAULT NULL,
  PRIMARY KEY (`InstructorID`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```



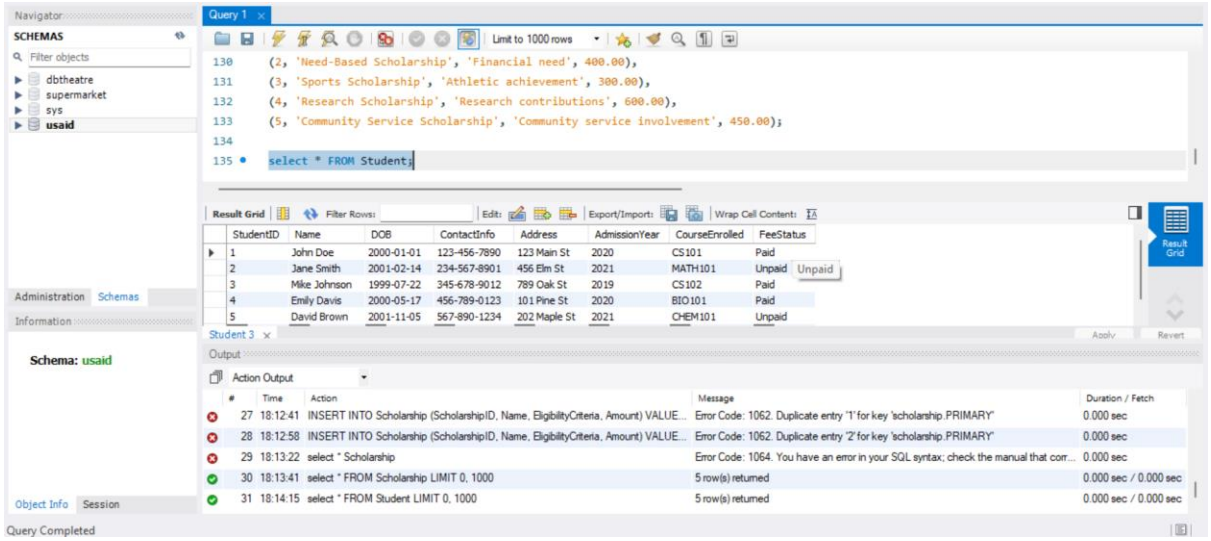


Step 3 (25%): Populate each of the table with at least 5 records per table



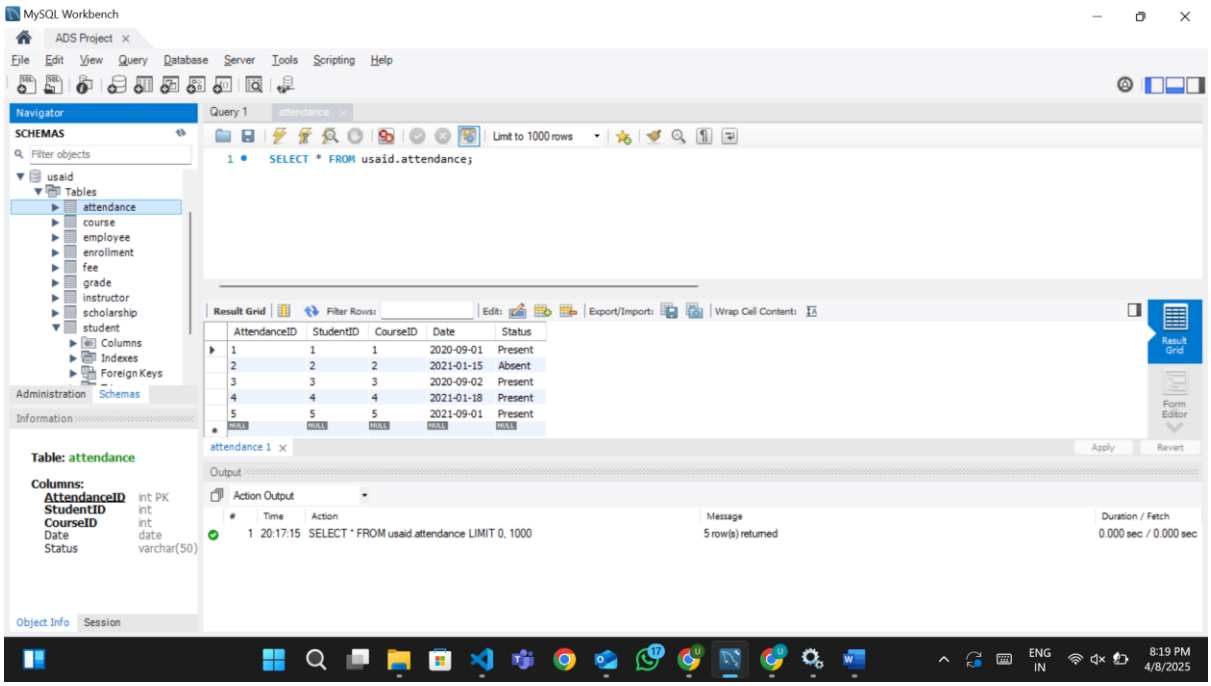


I HAVE DONE THIS FOR ALL TABLE
SCREENSHOT IS FOR THE REFERENCE

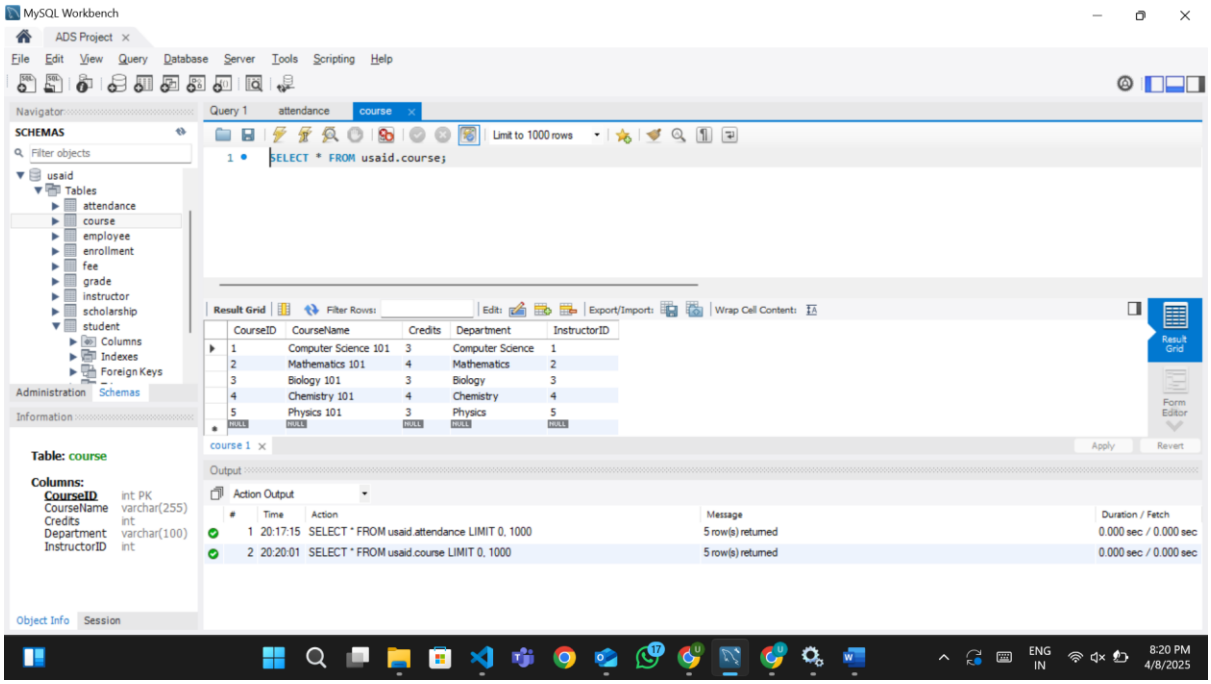


I HAVE REVISED THIS FILE BY SHOWING UP ALL THE TABLES, I HAVE CREATED THEM ALREADY THE FIRST TIME NOW I WILL JUST SHOW THEM HERE AS SCREENSHOTS

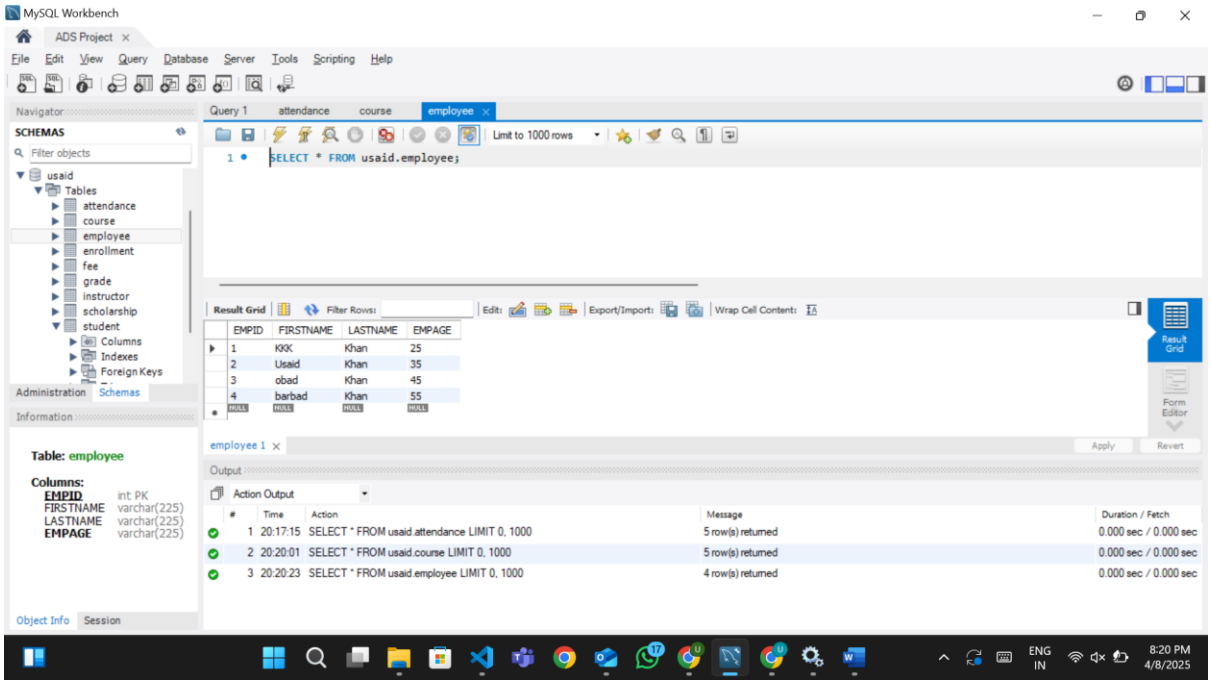
ATTENDANCE:



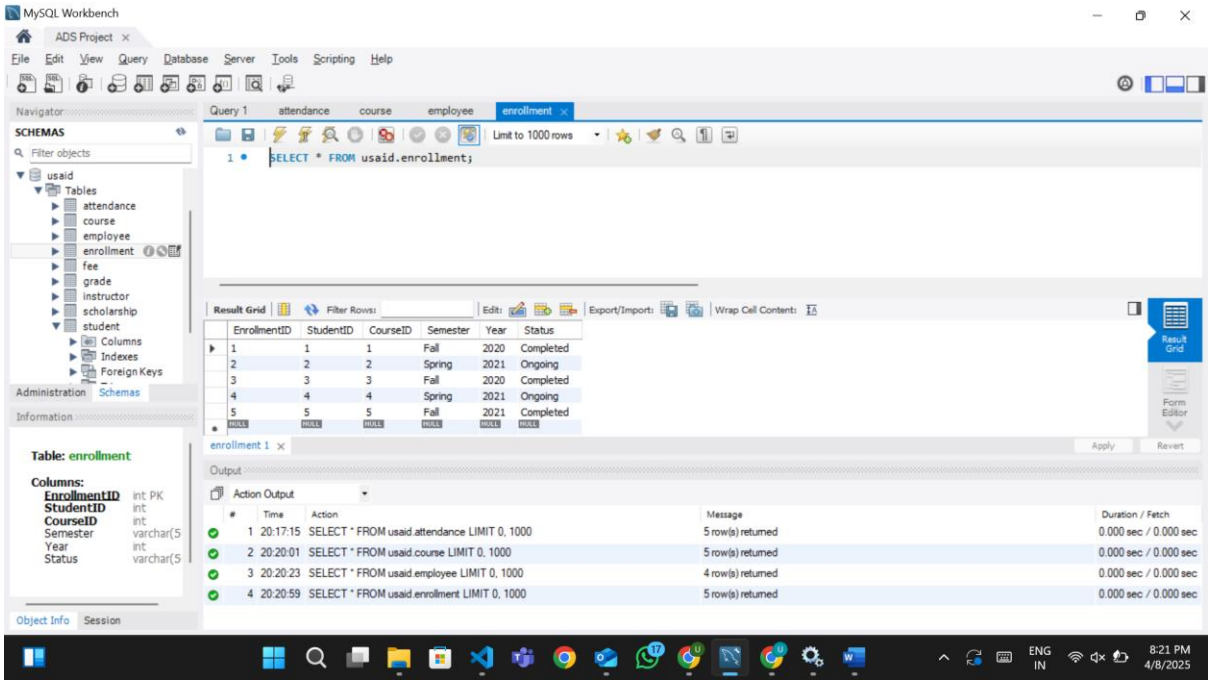
COURSE:


MySQL Workbench
ADS Project
File Edit View Query Database Server Tools Scripting Help
Navigator
SCHEMAS
Filter objects
usaid
Tables
attendance
course
employee
enrollment
fee
grade
instructor
scholarship
student
Columns
Indexes
Foreign Keys
Administration Schemas
Information
Table: course
Columns:
CourseID int PK
CourseName varchar(255)
Credits int
Department varchar(100)
InstructorID int
Object Info Session
Query 1 attendance course
Limit to 1000 rows
1 SELECT * FROM usaid.course;
Result Grid Filter Rows: Edit: Export/Import: Wrap Cell Content:
CourseID CourseName Credits Department InstructorID
1 Computer Science 101 3 Computer Science 1
2 Mathematics 101 4 Mathematics 2
3 Biology 101 3 Biology 3
4 Chemistry 101 4 Chemistry 4
5 Physics 101 3 Physics 5
course 1
Apply Revert
Output
Action Output
Time Action Message Duration / Fetch
1 20:17:15 SELECT * FROM usaid.attendance LIMIT 0, 1000 5 row(s) returned 0.000 sec / 0.000 sec
2 20:20:01 SELECT * FROM usaid.course LIMIT 0, 1000 5 row(s) returned 0.000 sec / 0.000 sec
ENG IN 8:20 PM 4/8/2025

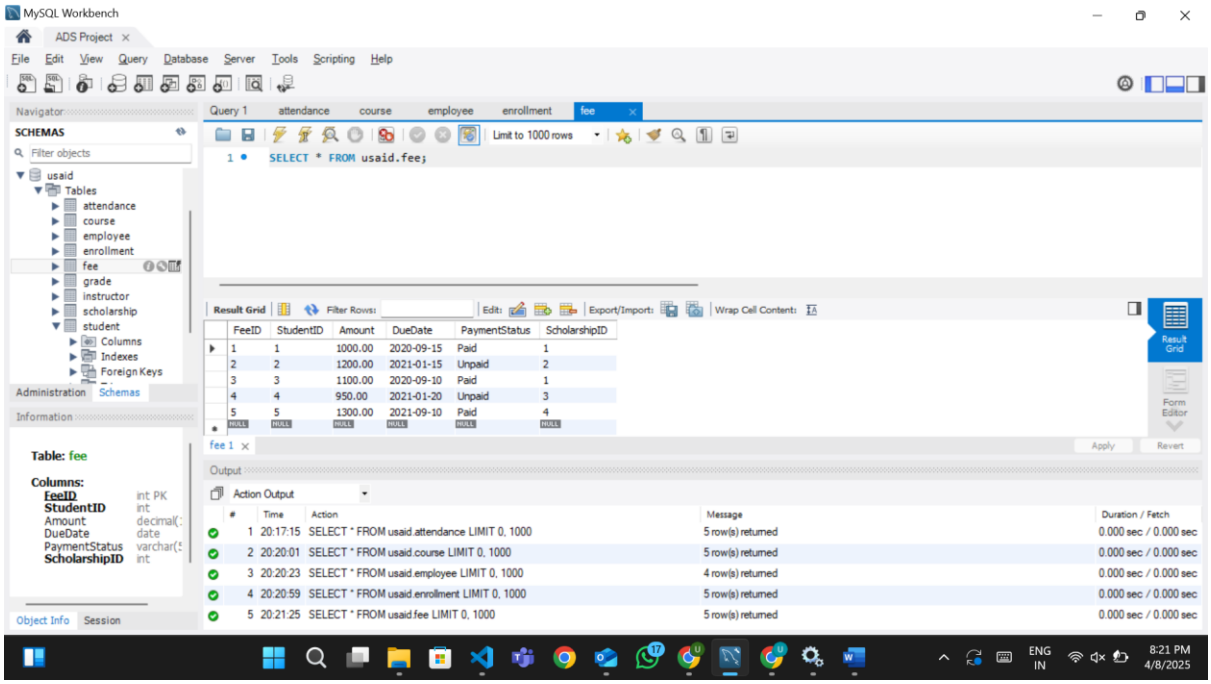
EMPLOYEE:


MySQL Workbench
ADS Project
File Edit View Query Database Server Tools Scripting Help
Navigator
SCHEMAS
Filter objects
usaid
Tables
attendance
course
employee
enrollment
fee
grade
instructor
scholarship
student
Columns
Indexes
Foreign Keys
Administration Schemas
Information
Table: employee
Columns:
EMPID int PK
FIRSTNAME varchar(225)
LASTNAME varchar(225)
EMPAGE varchar(225)
Object Info Session
Query 1 attendance course employee
Limit to 1000 rows
1 SELECT * FROM usaid.employee;
Result Grid Filter Rows: Edit: Export/Import: Wrap Cell Content:
EMPID FIRSTNAME LASTNAME EMPAGE
1 KKK Khan 25
2 Usaid Khan 35
3 obad Khan 45
4 barbad Khan 55
employee 1
Apply Revert
Output
Action Output
Time Action Message Duration / Fetch
1 20:17:15 SELECT * FROM usaid.attendance LIMIT 0, 1000 5 row(s) returned 0.000 sec / 0.000 sec
2 20:20:01 SELECT * FROM usaid.course LIMIT 0, 1000 5 row(s) returned 0.000 sec / 0.000 sec
3 20:20:23 SELECT * FROM usaid.employee LIMIT 0, 1000 4 row(s) returned 0.000 sec / 0.000 sec
ENG IN 8:20 PM 4/8/2025

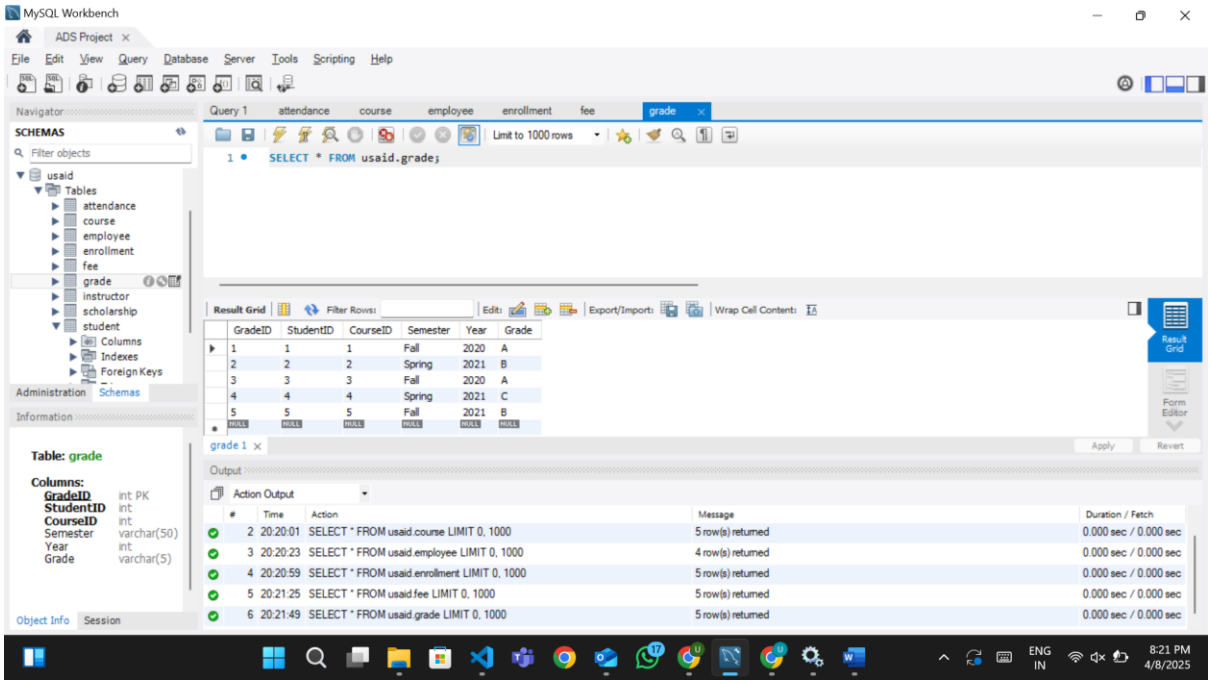
ENROLLMENT:



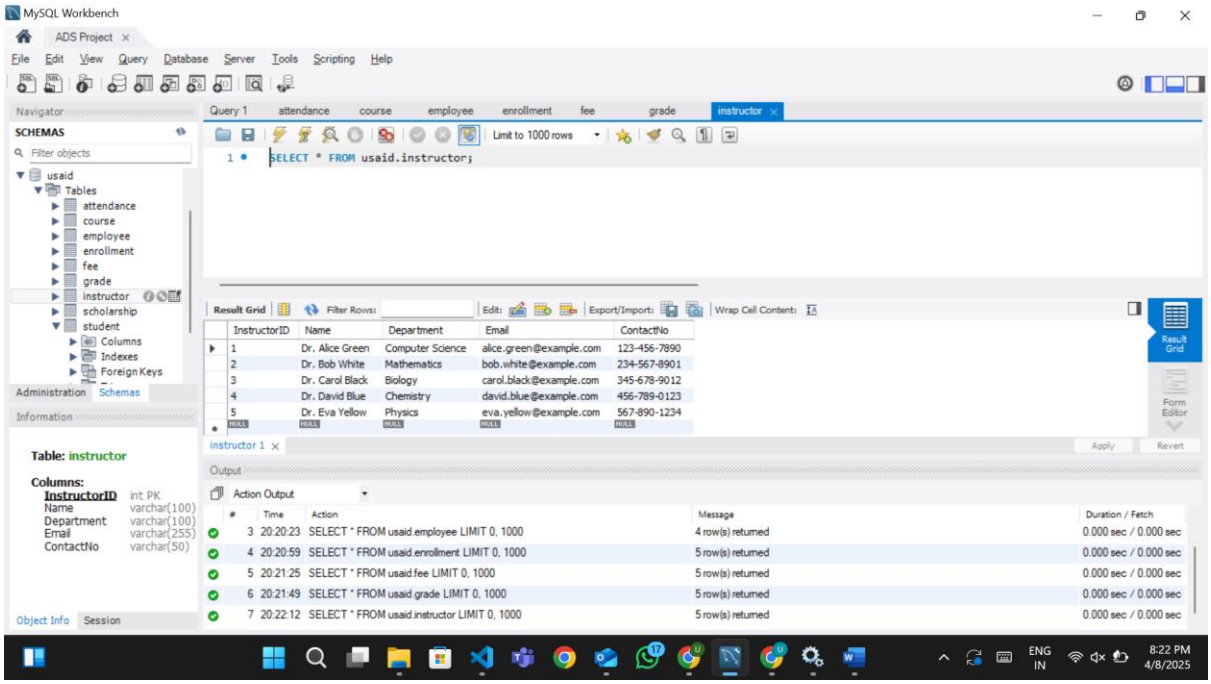
FEE:



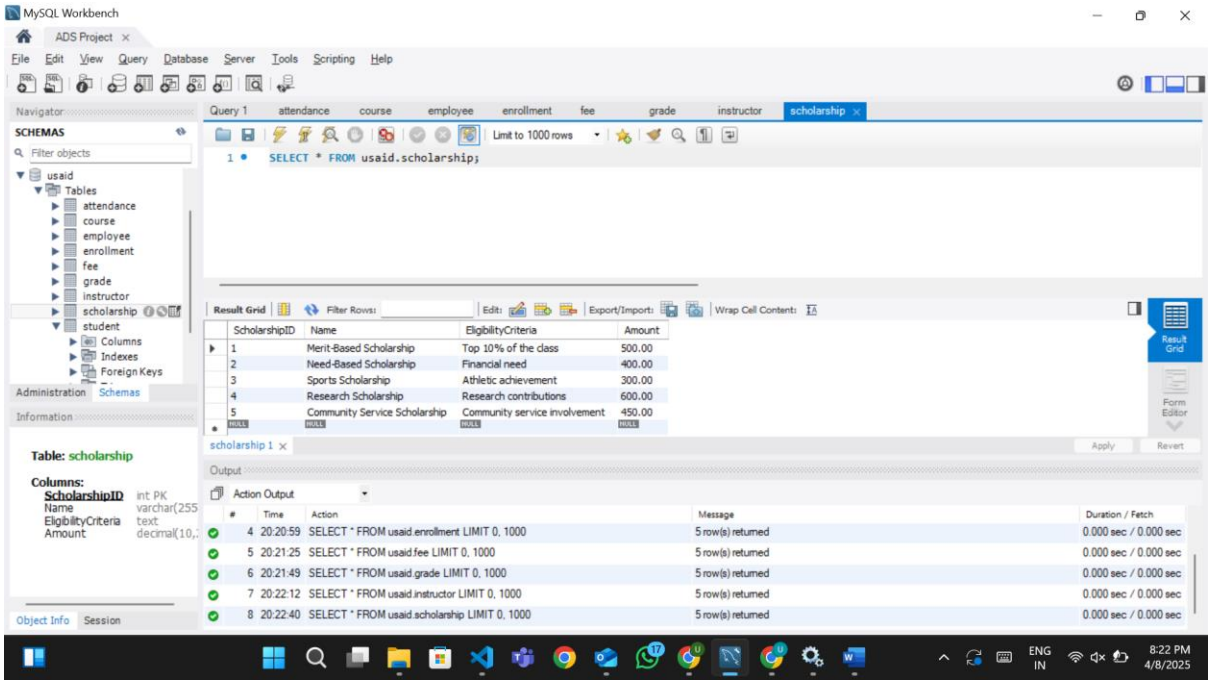
GRADE:



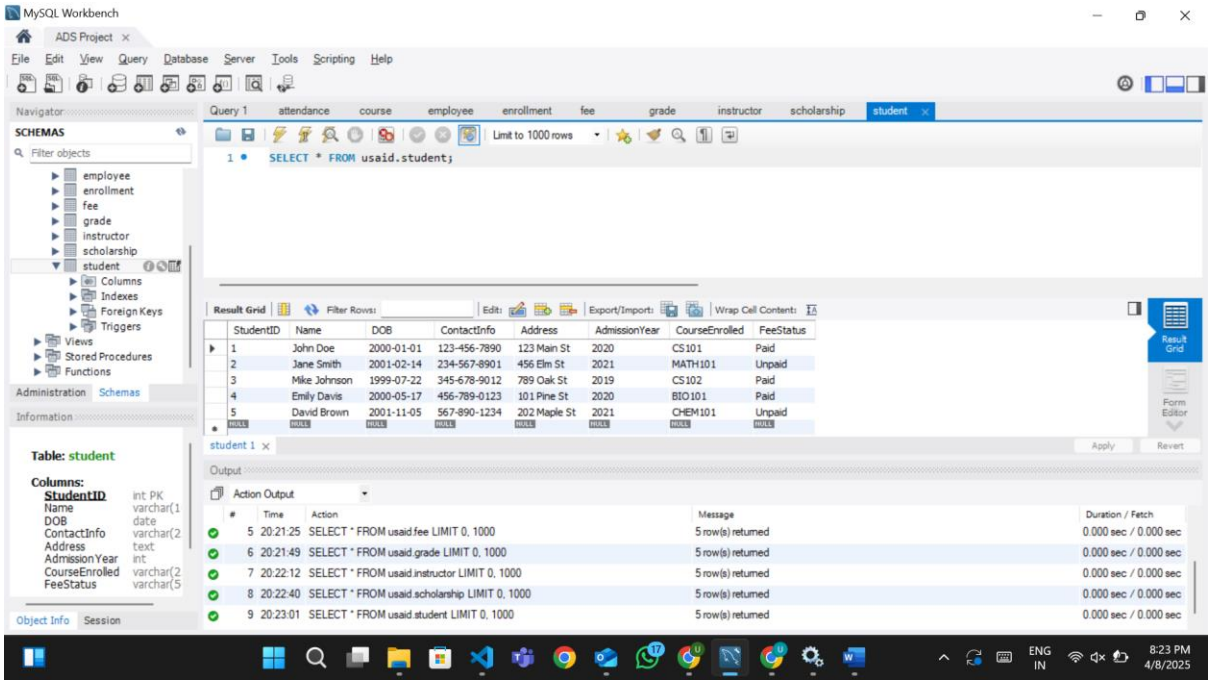
INSTRUCTOR:



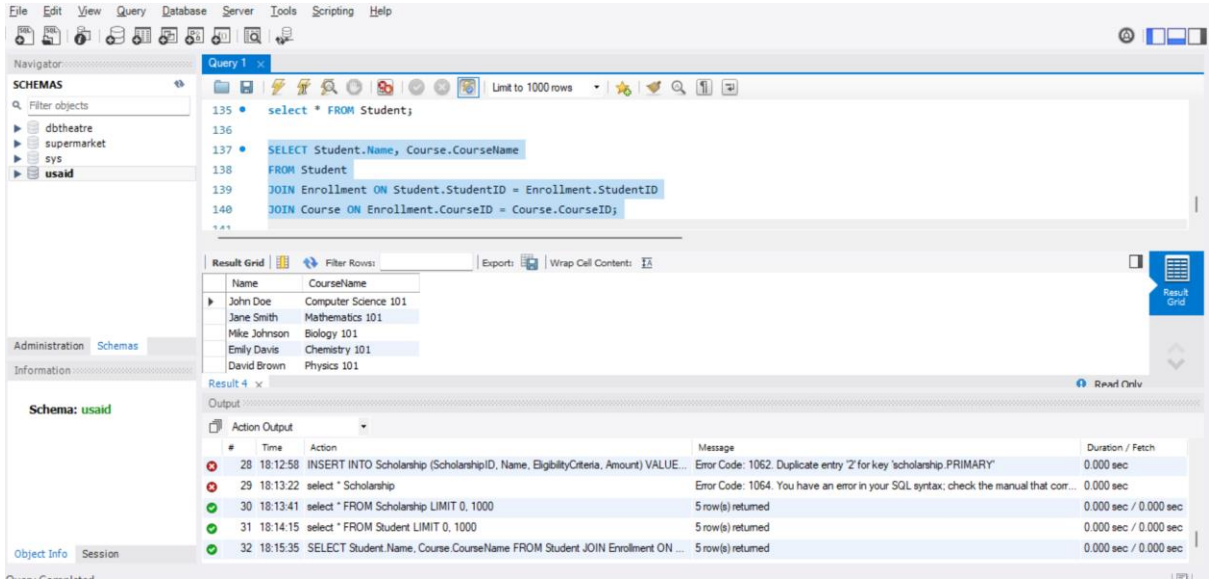
SCHOLARSHIP:

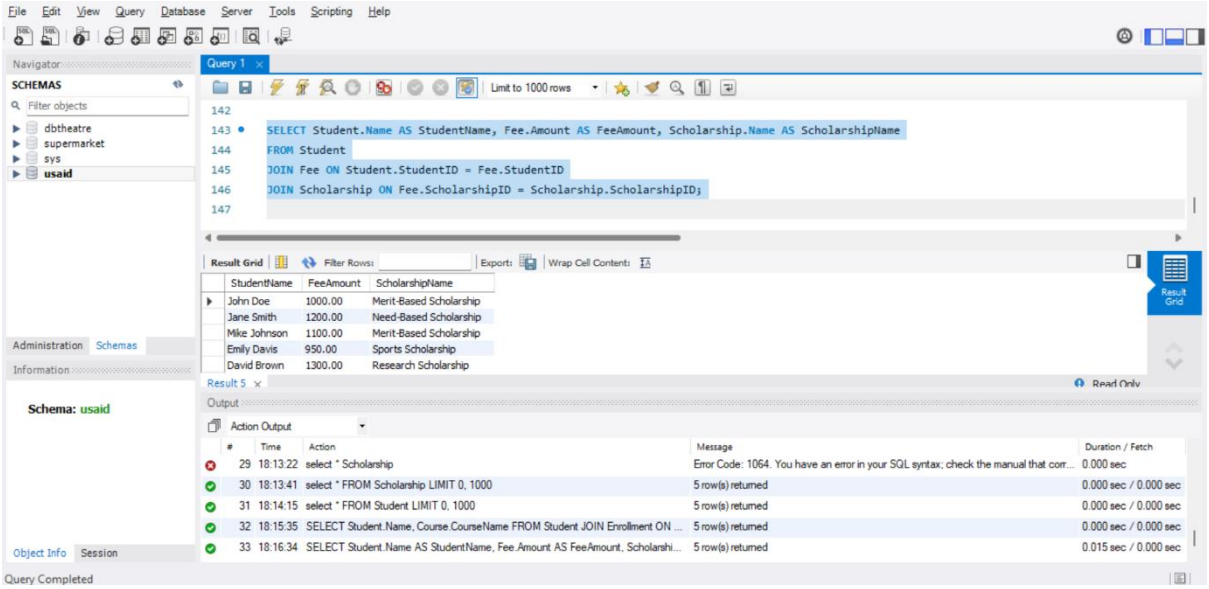


Student:



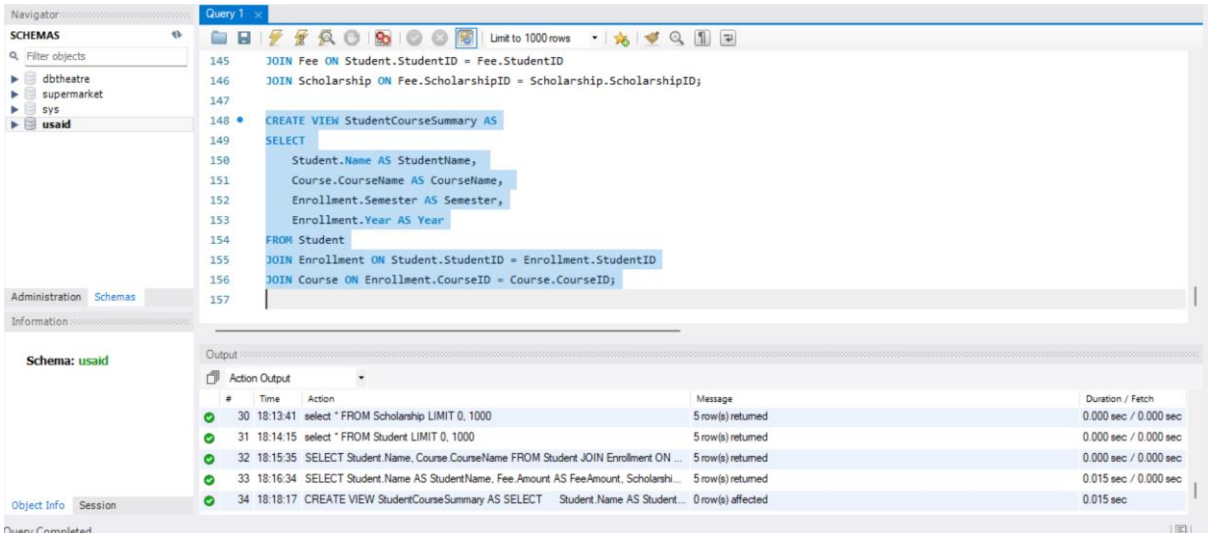
Step 4 (10%): Perform at least 2 or more JOIN queries



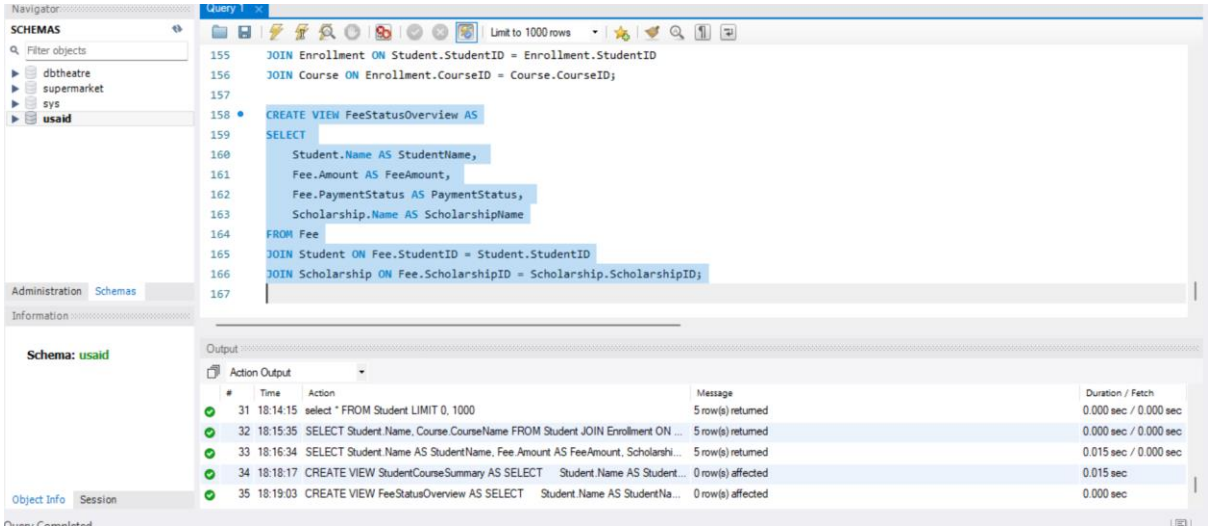


STEP 5:

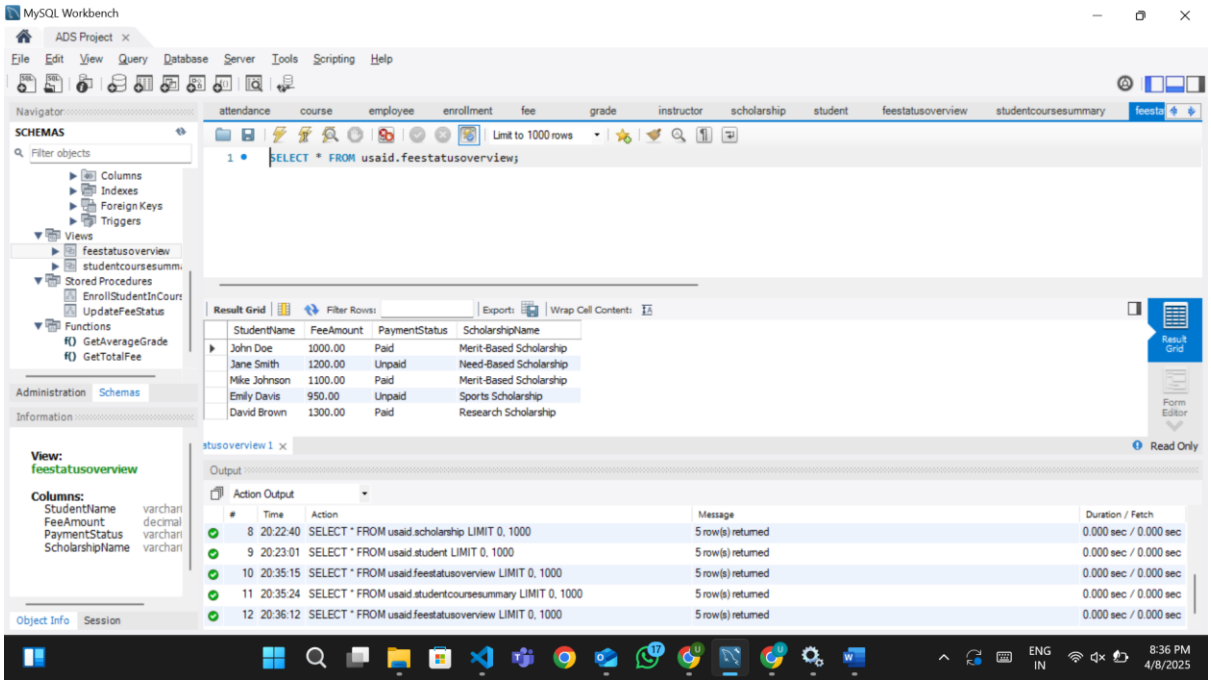
VIEW STUDENT COURSE SUMMARY

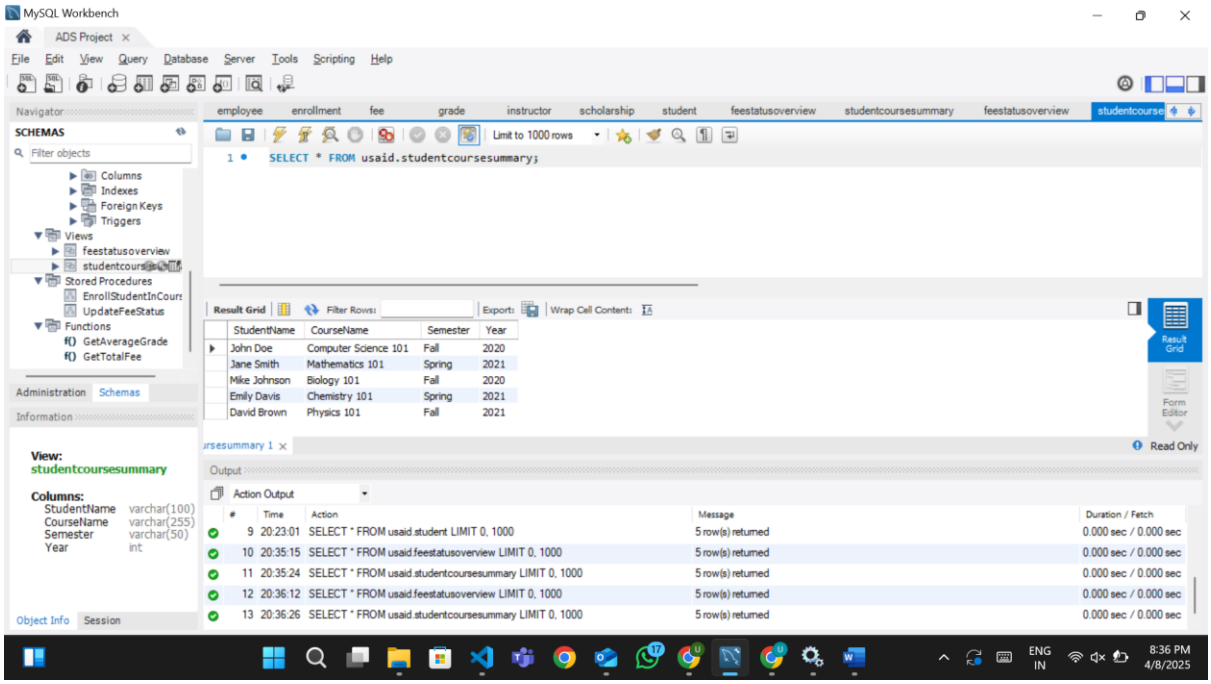


VIEW FEESTATUSOVERVIEW



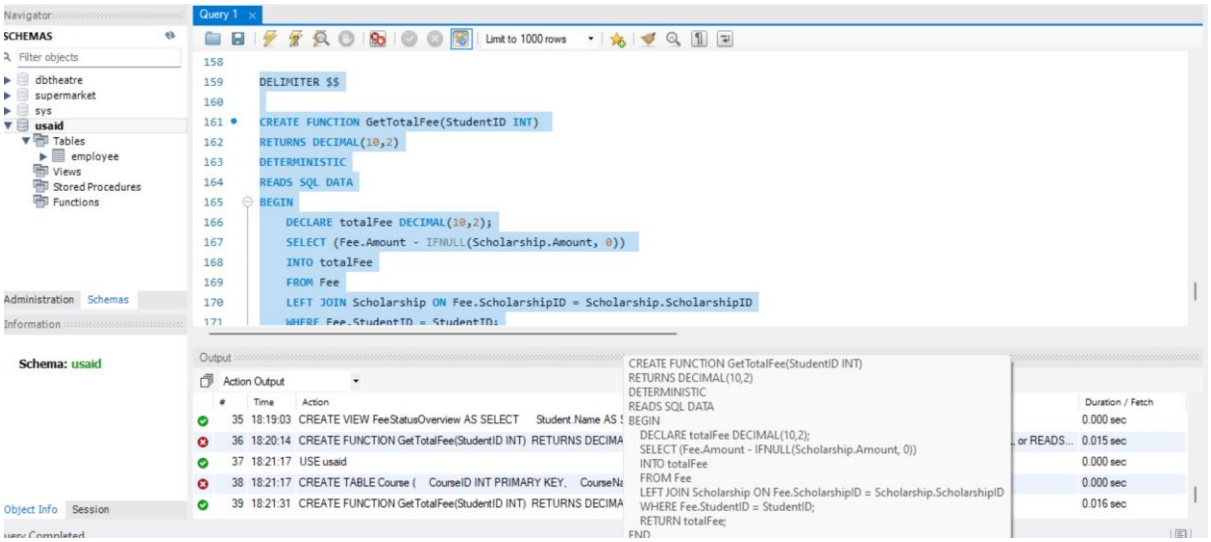
TESTED VIEWS:



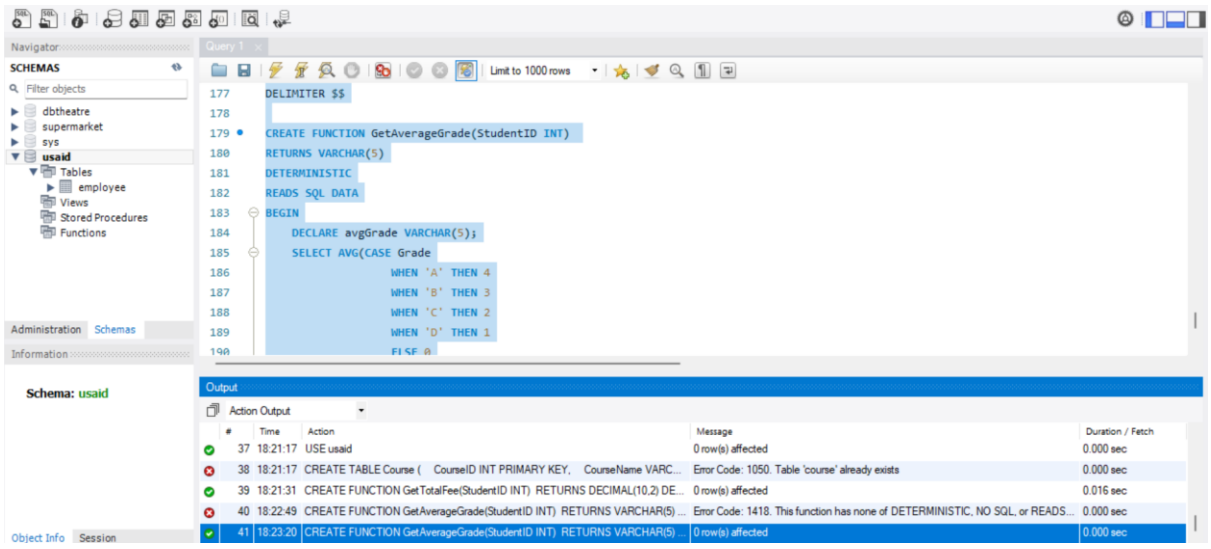


FUNCTIONS

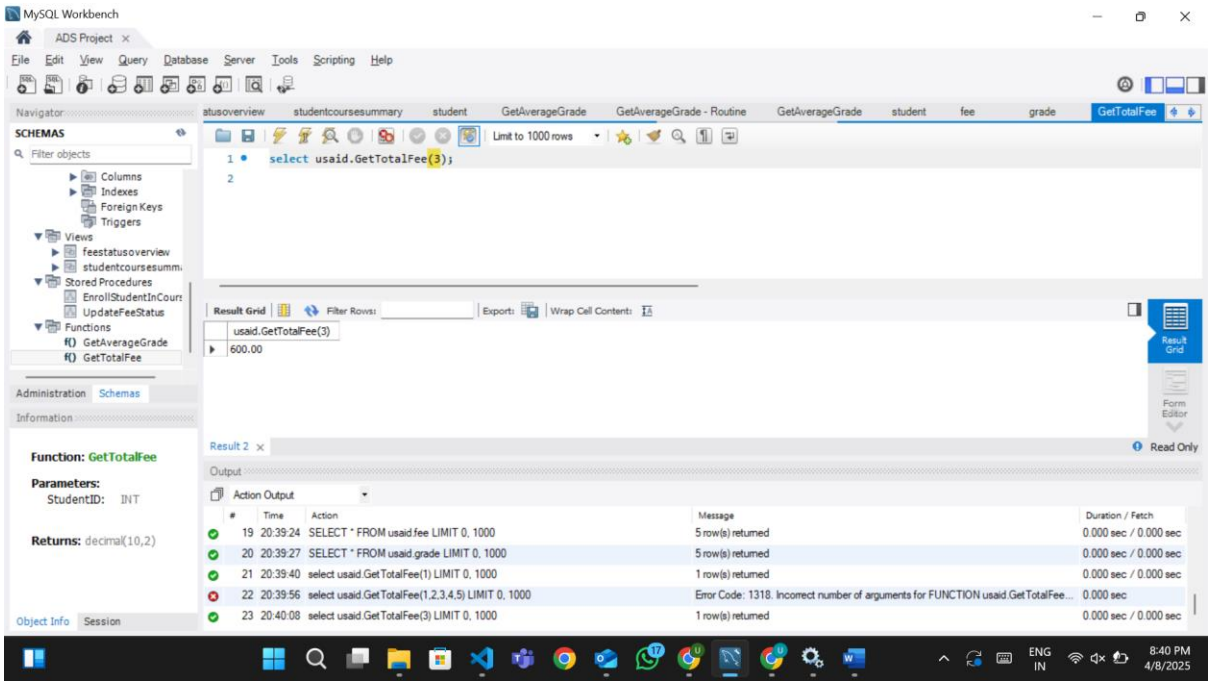
Function 1: Get Total Fee for a Student

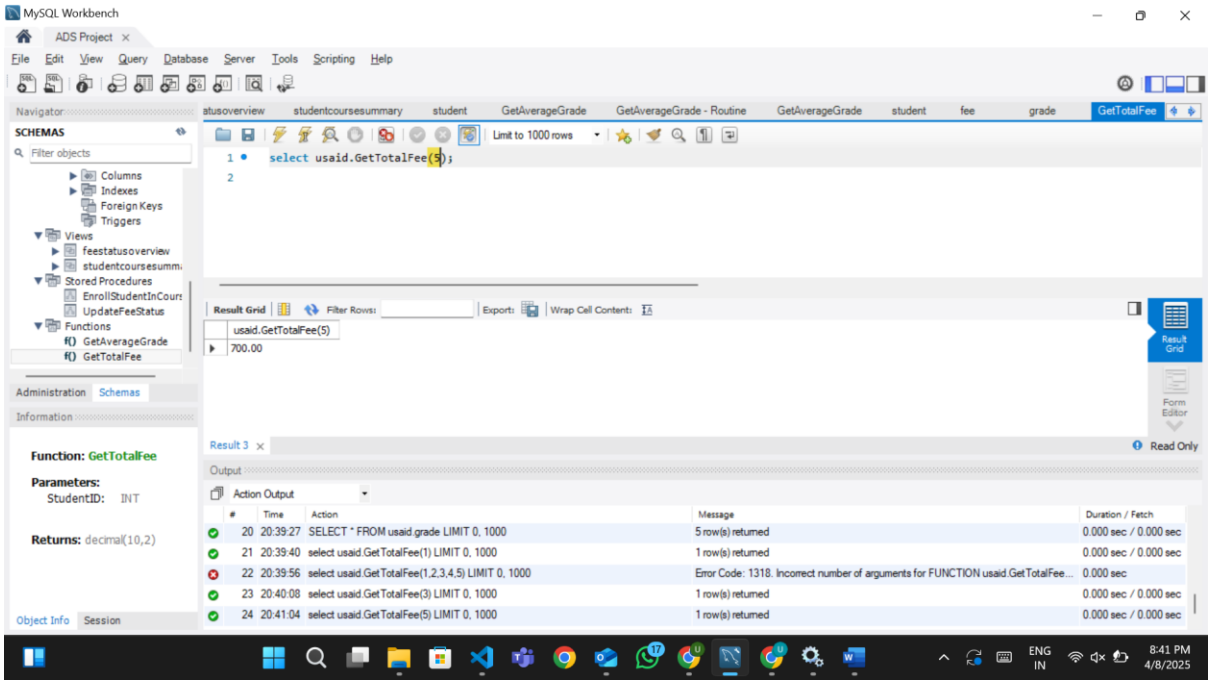


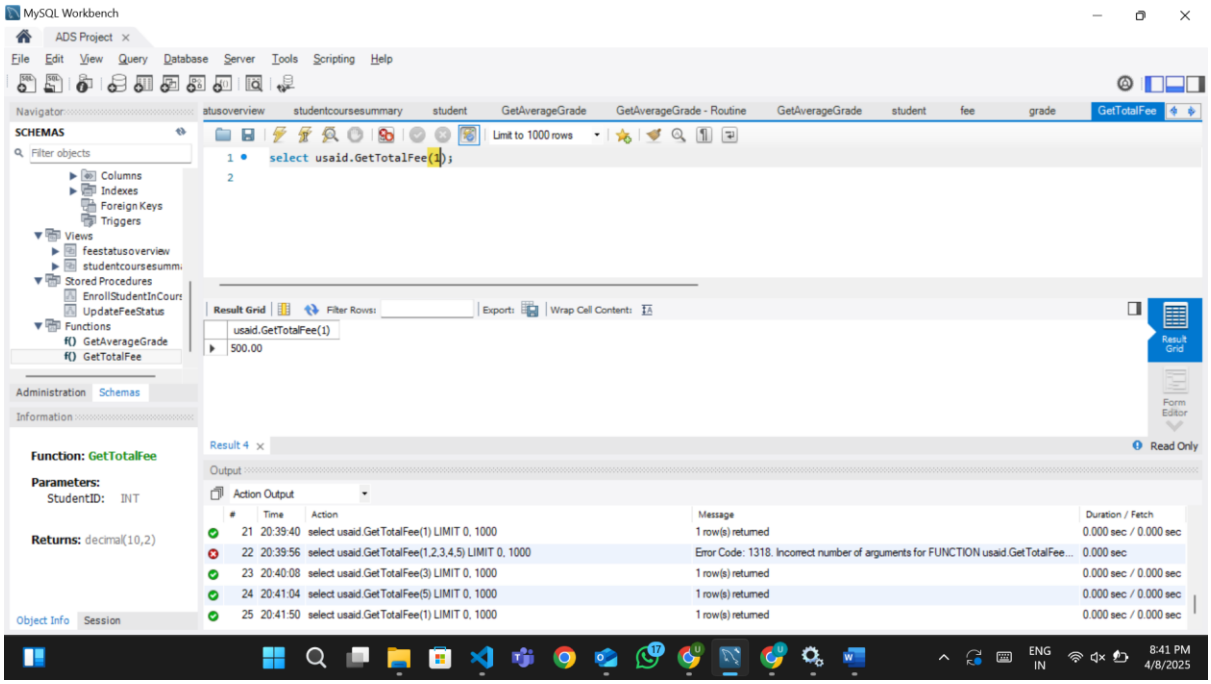
Function 2: Get Average Grade for a Student



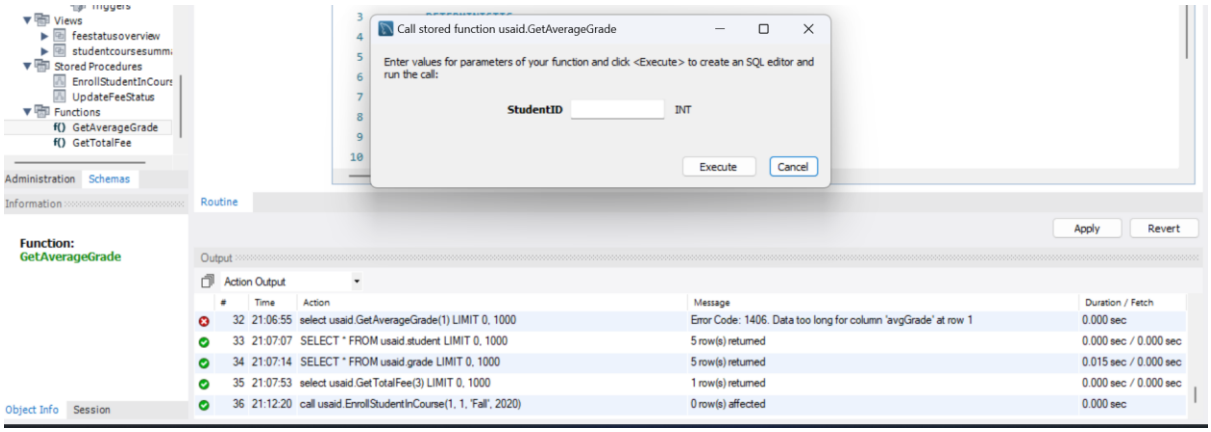
TESTED FUNCTIONS: Get total Fee



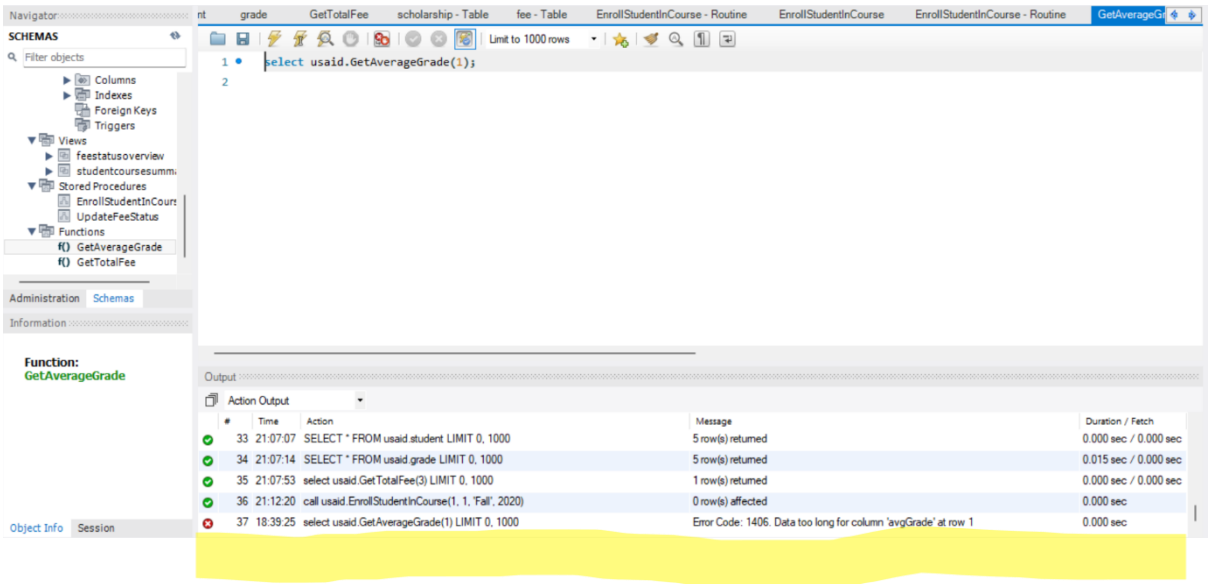




Get Average Grade of Students

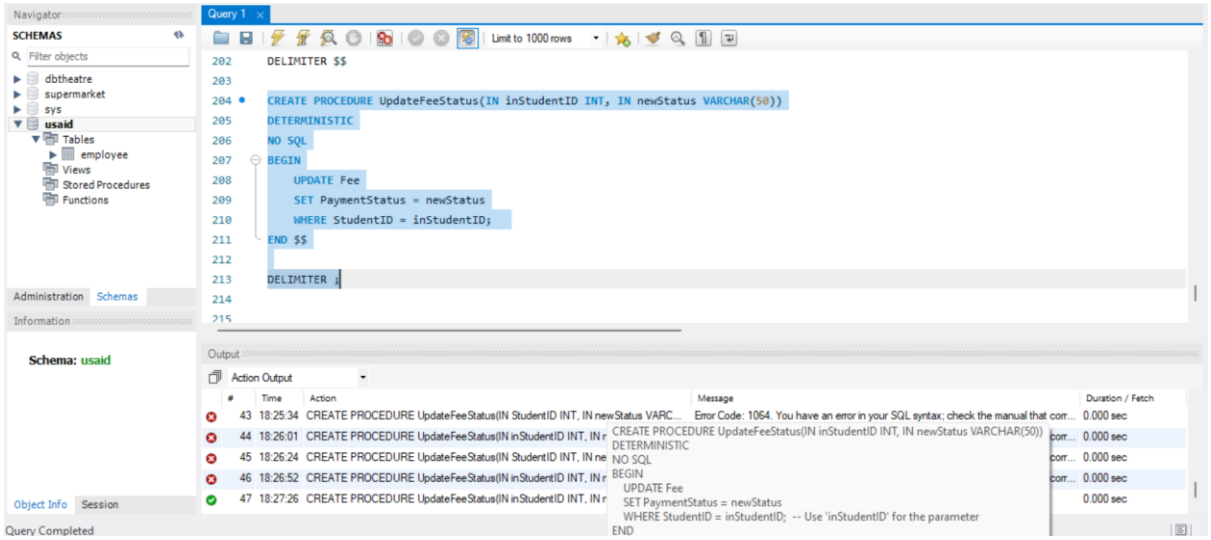


Here it could not return the grades as the grade are in Alphabets like A,B,C. So, it returns this as shown below

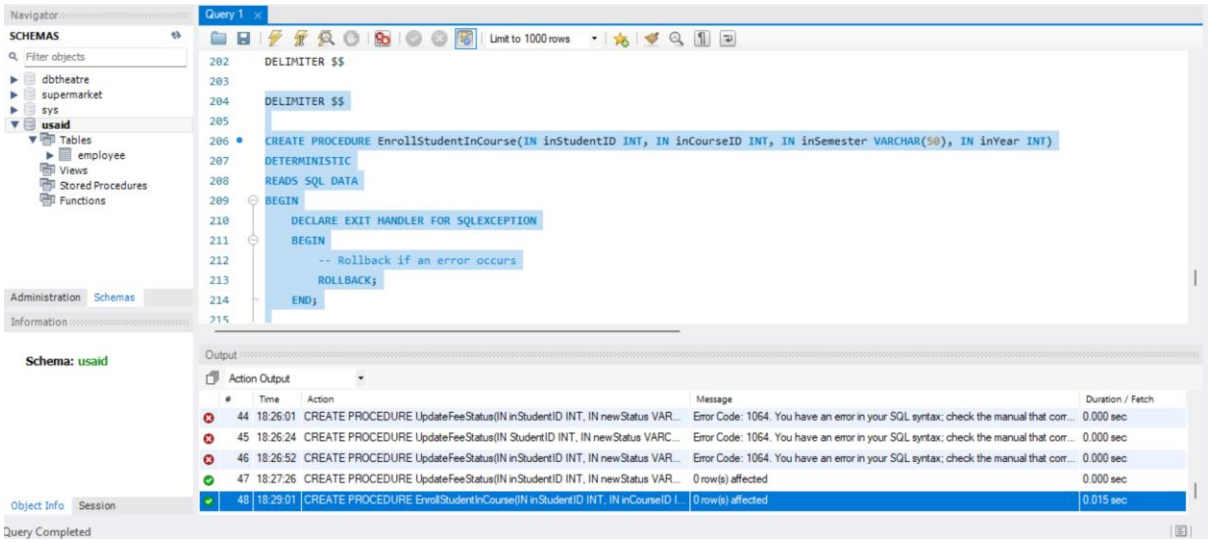


STORRED PROCEDURE:

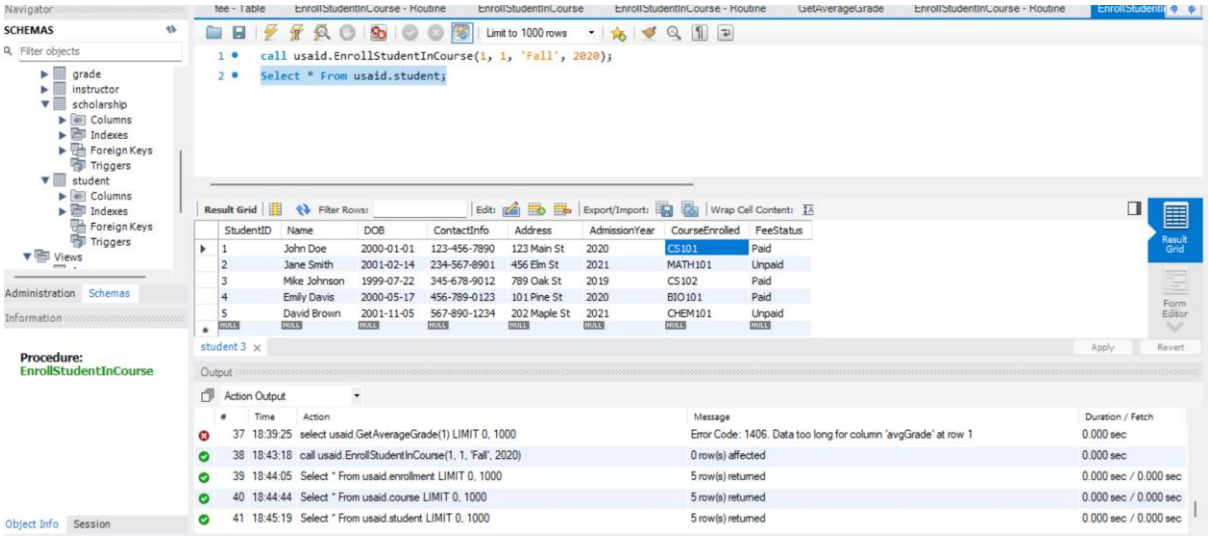
1. **Stored Procedure 1: Update Fee Status**



2. Stored Procedure: EnrollStudentInCourse (with Transaction)

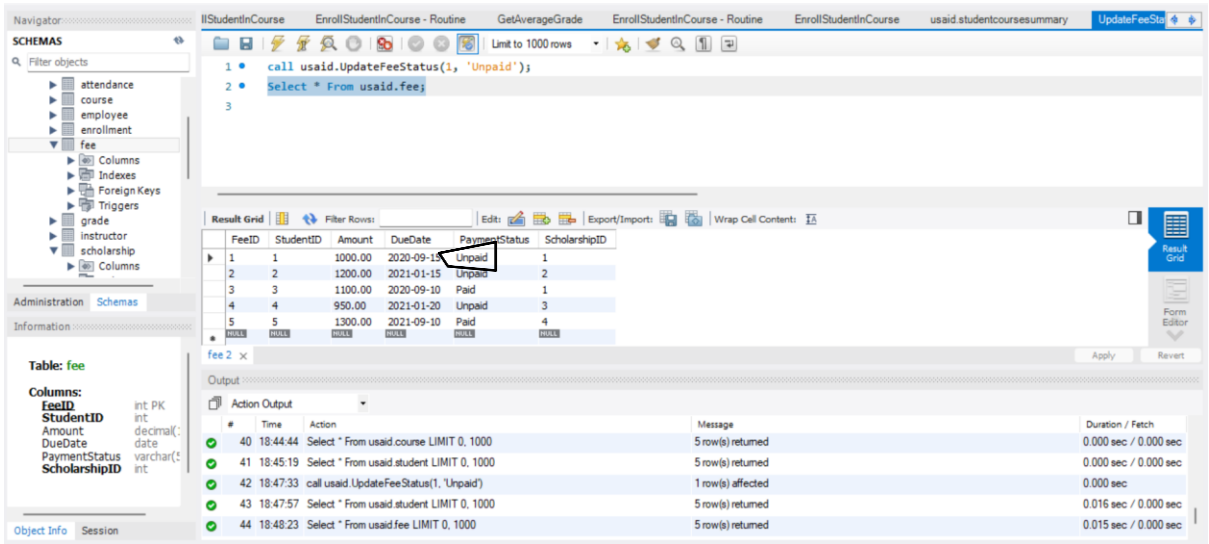


TESTING STORED PROCEDURE: **EnrollStudentInCourse**



After running the Stored Procedure, Student John Doe(StudentID) got enrolled with CS101 as shown above

TEST UpdateFeeStatus:



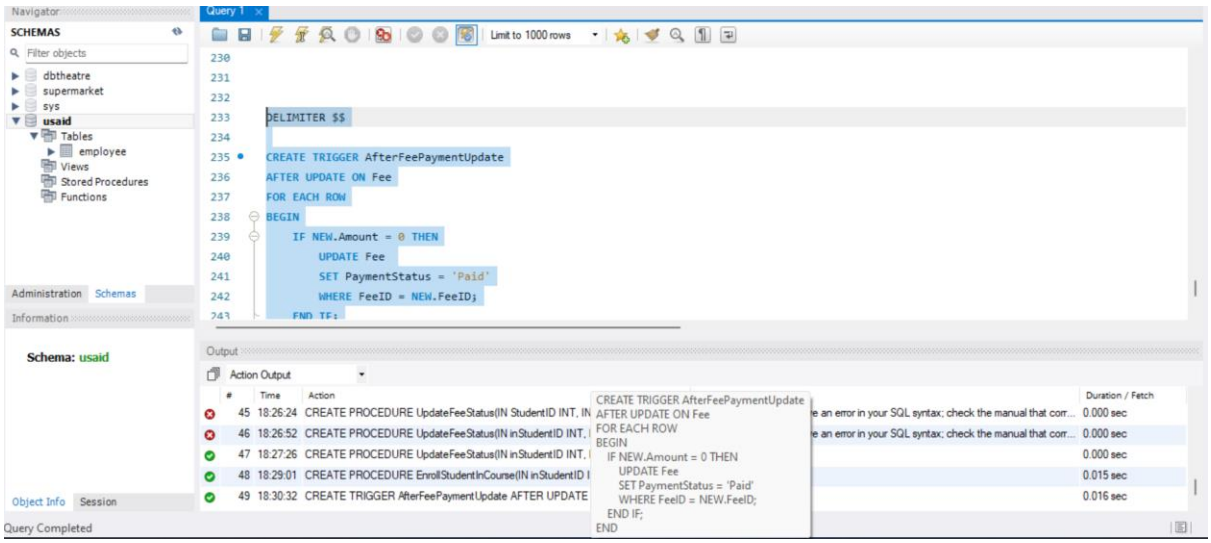
As we can see the StudentId (1) Fee status got updates as unpaid, before it was paid.

TRIGGER:

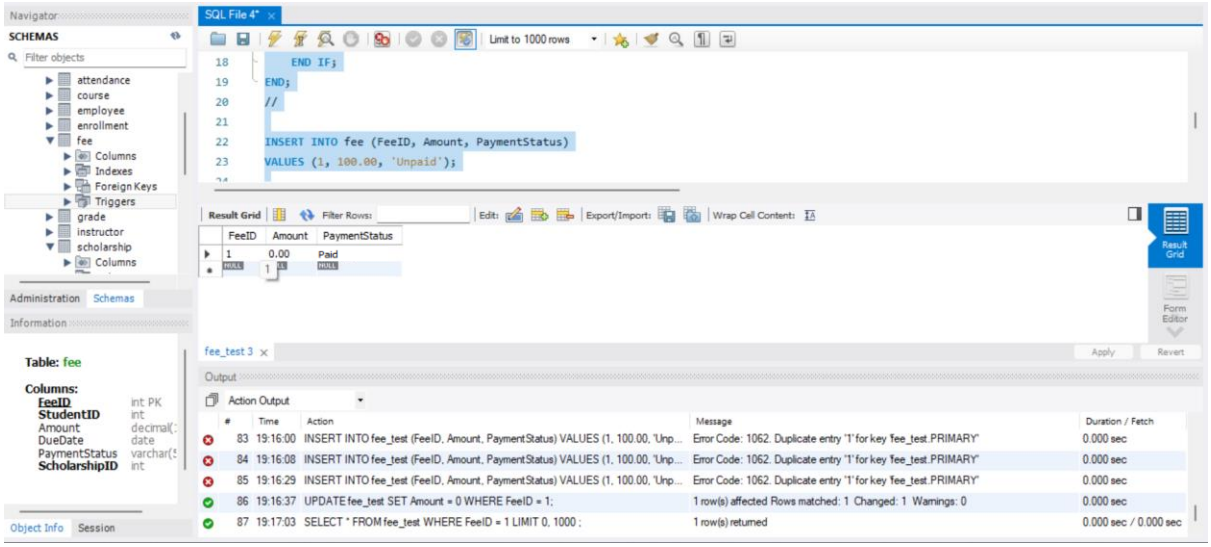
Trigger 1: Automatically Update Fee Status After Payment

```
CREATE DEFINER=`root`@`localhost` TRIGGER `AfterFeePaymentUpdate` AFTER UPDATE ON `fee` FOR EACH ROW BEGIN
    IF NEW.Amount = 0 THEN
        UPDATE Fee
        SET PaymentStatus = 'Paid'
        WHERE FeeID = NEW.FeeID;
    END IF;
END
DROP TABLE IF EXISTS fee_test;

CREATE TABLE fee_test (
    FeeID INT PRIMARY KEY,
    Amount DECIMAL(10,2),
    PaymentStatus VARCHAR(20)
);
DROP TRIGGER IF EXISTS BeforeFeePaymentUpdate;

DELIMITER //

CREATE TRIGGER BeforeFeePaymentUpdate
BEFORE UPDATE ON fee
FOR EACH ROW
BEGIN
    IF NEW.Amount = 0 THEN
        SET NEW.PaymentStatus = 'Paid';
    END IF;
```

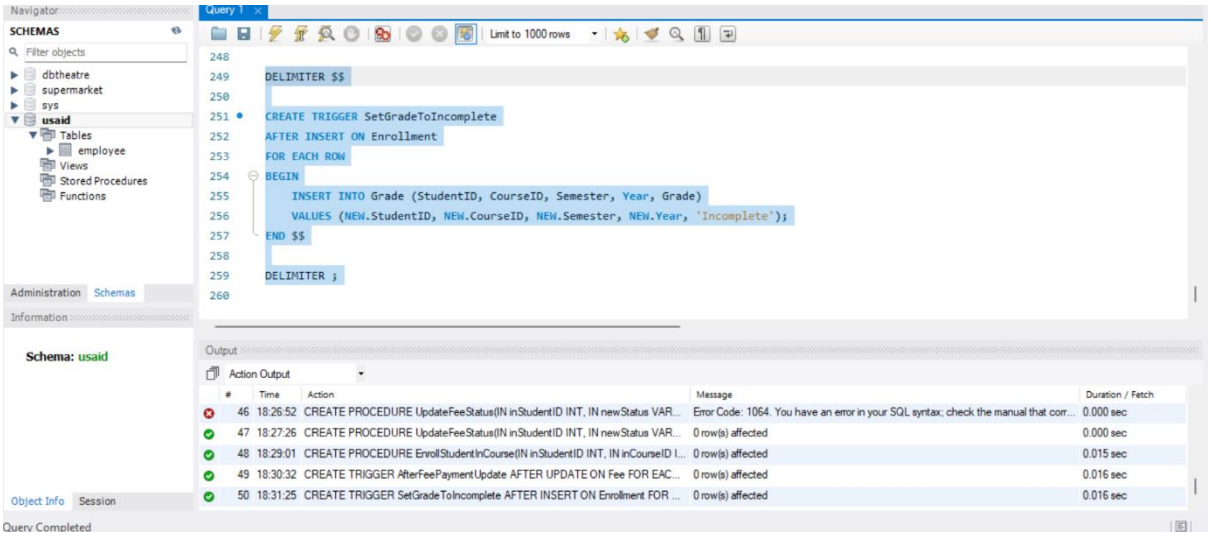
```
END;
//


INSERT INTO fee (FeeID, Amount, PaymentStatus)
VALUES (1, 100.00, 'Unpaid');


DELIMITER


UPDATE fee
SET Amount = 0
WHERE FeeID = 1;



SELECT * FROM fee_test WHERE FeeID = 1;
```



Test: As you can see it got updated to paid in status



Trigger 2: Update Grade After Enrollment



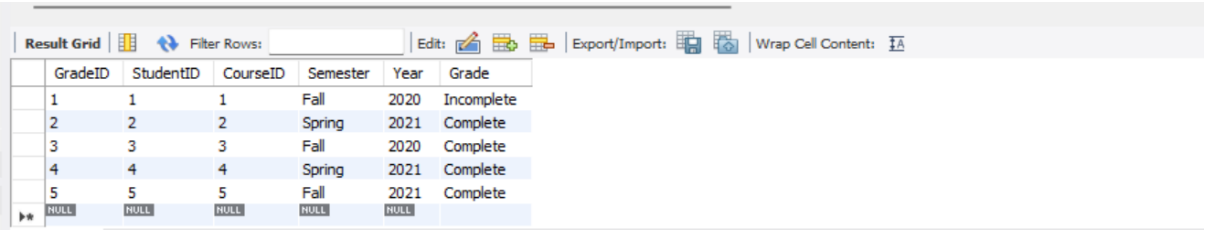
CREATE DEFINER=`root`@`localhost` TRIGGER `SetGradeToIncomplete` AFTER INSERT ON `enrollment` FOR EACH ROW BEGIN

INSERT INTO Grade (StudentID, CourseID, Semester, Year, Grade)

VALUES (NEW.StudentID, NEW.CourseID, NEW.Semester, NEW.Year, 'Incomplete');

END

Test: As we can see the student ID change to incomplete



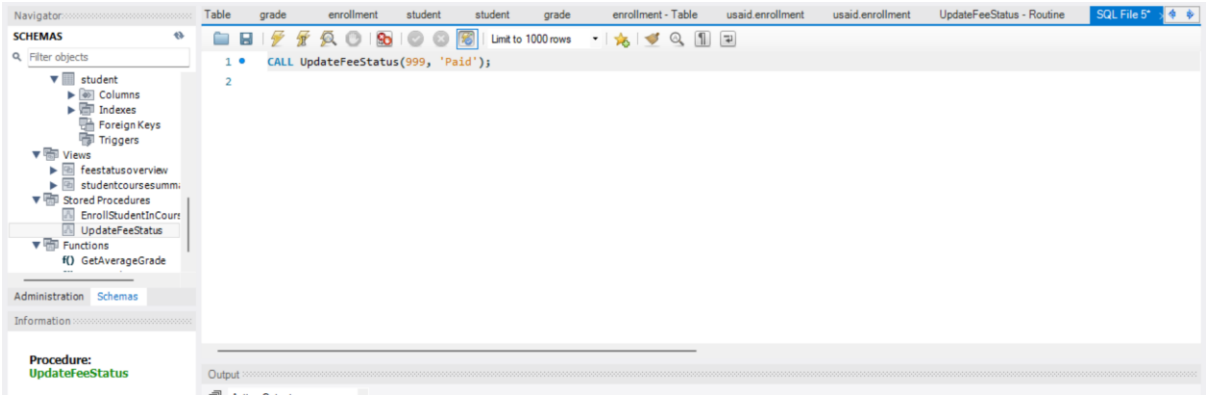
Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

GradeID	StudentID	CourseID	Semester	Year	Grade
1	1	1	Fall	2020	Incomplete
2	2	2	Spring	2021	Complete
3	3	3	Fall	2020	Complete
4	4	4	Spring	2021	Complete
5	5	5	Fall	2021	Complete
NULL	NULL	NULL	NULL	NULL	

5. Transaction:

A transaction is included in **Stored Procedure 2** (Enroll Student in Course), where we ensure that both the **Enrollment** and **Attendance** are recorded in a single transaction. If an error occurs at any point, the transaction is rolled back to maintain data consistency.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `UpdateFeeStatus`(IN inStudentID INT, IN newStatus VARCHAR(50))
    NO SQL
    DETERMINISTIC
BEGIN
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
    BEGIN
        -- Rollback transaction if any error occurs
        ROLLBACK;
    END;

    START TRANSACTION;

    -- Update Fee status
    UPDATE Fee
    SET PaymentStatus = newStatus
    WHERE StudentID = inStudentID;

    -- Simulate an error for testing (optional)
    -- SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Simulated Error';

    COMMIT;
END;
```



In this case, the update fails, and the rollback is triggered.

Reflective Report: Physical Design Implementation in MySQL Workbench

Project Overview:

The goal of this phase was to implement the physical database design for our academic management system using MySQL Workbench. This included transforming the relational schema into an EER diagram, generating SQL scripts through forward engineering, populating tables, and implementing SQL operations such as joins, views, stored procedures, functions, and triggers.

My Contributions:

I undertook the **entire responsibility** of this phase, from conceptualization to implementation. My tasks included:

- **Creating the EER Diagram:** Using the relational schema from Part 2, I designed the EER model that visually represented the database structure and relationships.
- **Forward Engineering:** I generated SQL code from the model using MySQL Workbench's forward engineering feature and created all tables in the usaid schema.
- **Data Population:** I manually inserted sample data—**at least 5 records per table**—into all tables including student, course, attendance, employee, enrollment, fee, grade, instructor, and scholarship.
- **SQL Operations Implementation:**
    - **JOIN Queries:** I wrote and tested multiple join queries to retrieve meaningful data across tables.
    - **Views:** Created and tested views like StudentCourseSummary and FeeStatusOverview.
    - **Functions:** Developed stored functions such as GetTotalFeeForStudent and GetAverageGradeForStudent.
    - **Stored Procedures:** Implemented procedures like UpdateFeeStatus and EnrollStudentInCourse, incorporating transactional logic.
    - **Triggers:** Developed triggers to automatically update fee status and student grades after relevant actions.

Each component was tested rigorously, and results were documented through screenshots and outputs for validation.

Challenges Faced:

1. **Server Connection Error:** Initially, I encountered an issue with MySQL Workbench where the server status showed as **“Stopped”**, and the application failed to connect to localhost. The error message was:

"Could not connect, server may not be running. Unable to connect to localhost."

Solution: I had to manually start the MySQL service using the Services panel in Windows. Once restarted, MySQL Workbench could establish a connection and function properly.

2. **Forward Engineering Conflicts:** When forward engineering, I had to ensure foreign key constraints and data types matched accurately, especially when linking tables like fee, attendance, and enrollment. Mismatches caused errors that I resolved by modifying the model and regenerating the SQL.
3. **Populating Foreign Key Tables:** Populating data in tables that involved foreign key relationships required careful ordering. I inserted data into parent tables like student, course, and scholarship first to avoid constraint violations.
4. **Syntax Errors in Stored Procedures and Triggers:** Initially, I faced minor syntax issues while defining stored procedures and triggers, especially related to delimiters and transaction blocks. I resolved these through debugging and MySQL documentation reference.

Reflection:

This experience significantly enhanced my understanding of physical database design, data integrity enforcement, and advanced SQL operations. By managing every aspect of the implementation—from schema creation to automation through triggers—I gained practical insights into database architecture, debugging, and best practices in SQL-based development. Despite the initial hurdles, this project gave me a strong foundation in MySQL Workbench and real-world database implementation.