

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

The screenshot shows the MySQL Workbench interface. On the left, the Navigator pane displays the schema structure under 'usaid'. The 'Tables' node is expanded, showing 'Course' and 'Instructor'. The 'Information' pane at the bottom shows the schema for 'usaid'. The main area, titled 'Query 1', contains the SQL code for creating the 'Course' and 'Instructor' tables. The 'Output' pane at the bottom shows the execution log with 0 row(s) affected for each create statement, and the 'Object Info' and 'Session' tabs are visible at the bottom.

```
USE usaid;
CREATE TABLE Course (
    CourseID INT PRIMARY KEY,
    CourseName VARCHAR(255),
    Credits INT,
    Department VARCHAR(100),
    InstructorID INT
);
CREATE TABLE Instructor (
    InstructorID INT PRIMARY KEY,
    Name VARCHAR(100),
    Department VARCHAR(100),
);
```

The screenshot shows the SSMS interface. The Object Explorer on the left displays the database structure under the 'usaid' schema, including tables like 'Enrollment', 'Attendance', and 'Grade'. The 'Information' node is also visible. The 'Administration' tab is selected. The 'Schemas' tab is highlighted in green, indicating the current schema. The 'Query 1' window on the right contains the T-SQL code for creating three tables: Enrollment, Attendance, and Grade. The code uses INT PRIMARY KEY for primary keys and VARCHAR(50) for character fields. FOREIGN KEY constraints link StudentID and CourseID between tables.

```
CREATE TABLE Enrollment (
    EnrollmentID INT PRIMARY KEY,
    StudentID INT,
    CourseID INT,
    Semester VARCHAR(50),
    Year INT,
    Status VARCHAR(50),
    FOREIGN KEY (StudentID) REFERENCES Student(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Course(CourseID)
);

CREATE TABLE Attendance (
    AttendanceID INT PRIMARY KEY,
    StudentID INT,
    CourseID INT,
    Date DATE,
    Status VARCHAR(50),
    FOREIGN KEY (StudentID) REFERENCES Student(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Course(CourseID)
);

CREATE TABLE Grade (
```

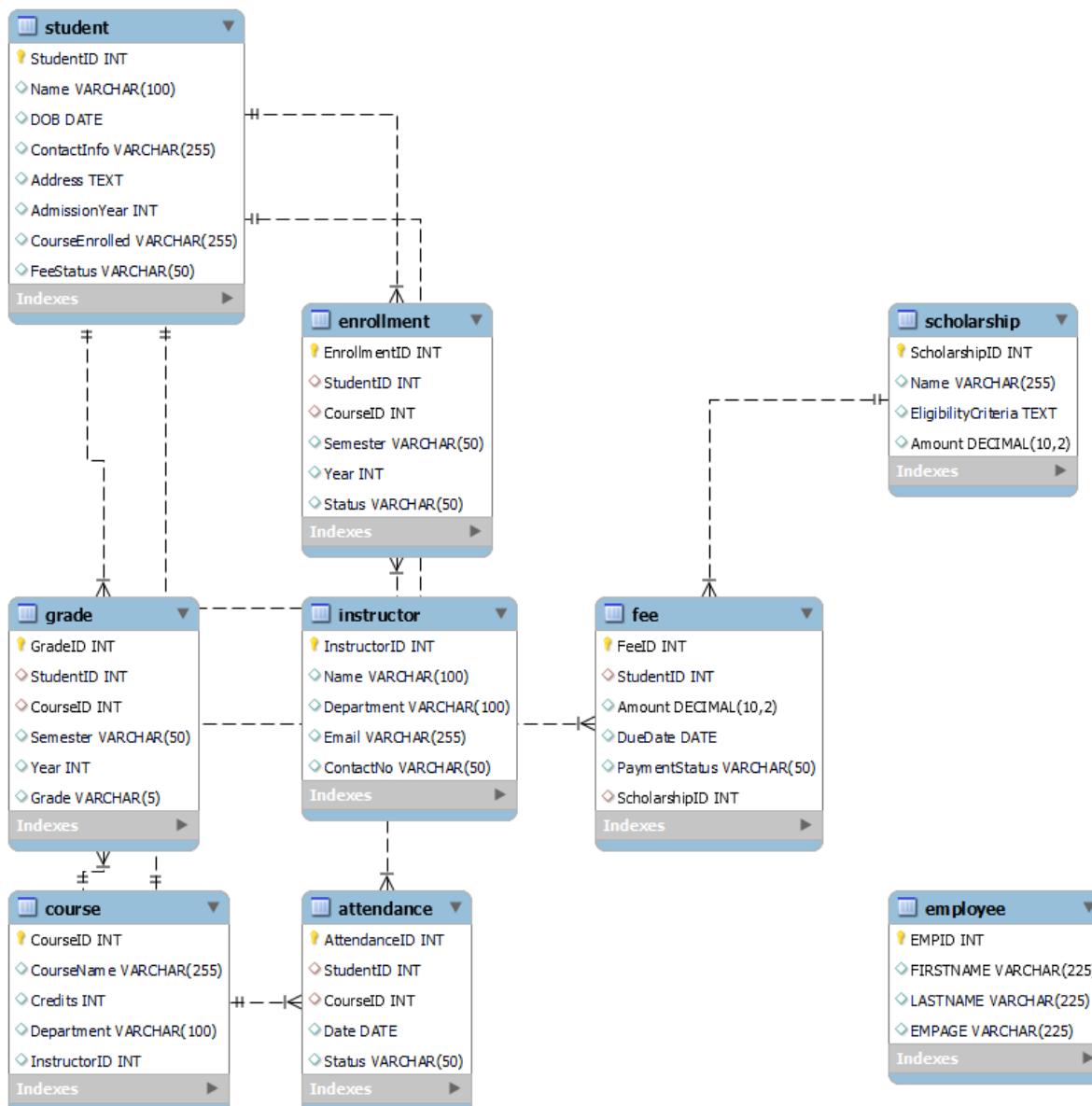
Schemas: usaid

```

49  );
50
51
52
53 • CREATE TABLE Scholarship (
54     ScholarshipID INT PRIMARY KEY,
55     Name VARCHAR(255),
56     EligibilityCriteria TEXT,
57     Amount DECIMAL(10,2)
58 );
59
60 • CREATE TABLE Fee (
61     FeeID INT PRIMARY KEY,
62     StudentID INT,
63     Amount DECIMAL(10,2),
64     DueDate DATE,
65     PaymentStatus VARCHAR(50),
66     ScholarshipID INT,
67     FOREIGN KEY (StudentID) REFERENCES Student(StudentID),
68     FOREIGN KEY (ScholarshipID) REFERENCES Scholarship(ScholarshipID)
69 );
70

```

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`

```

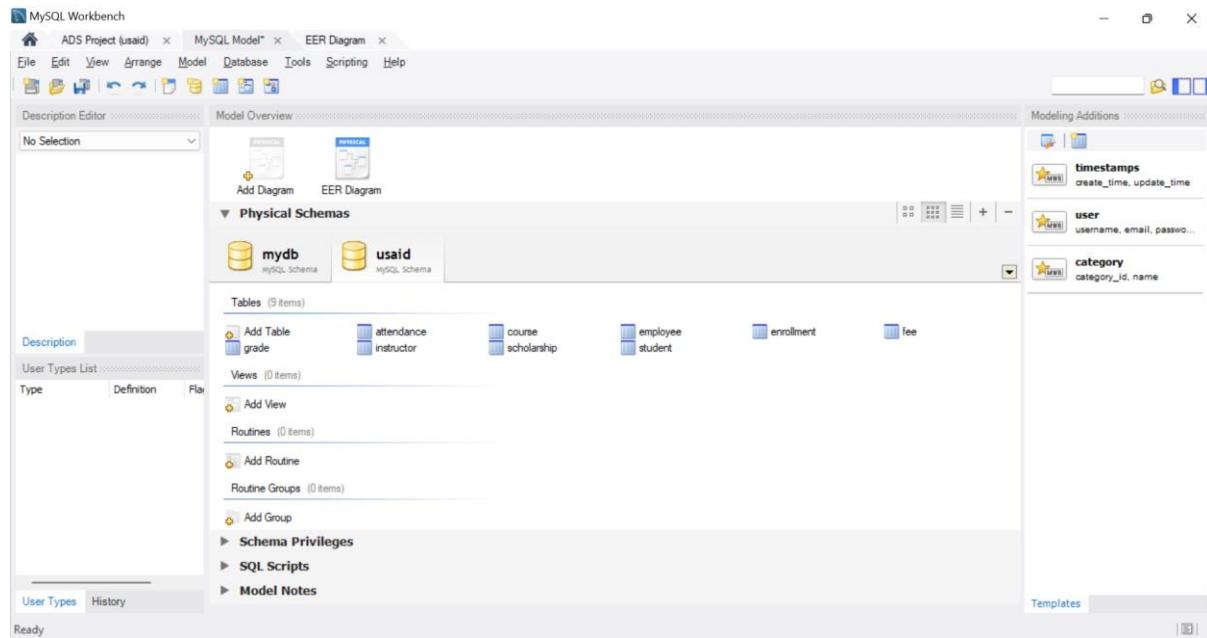
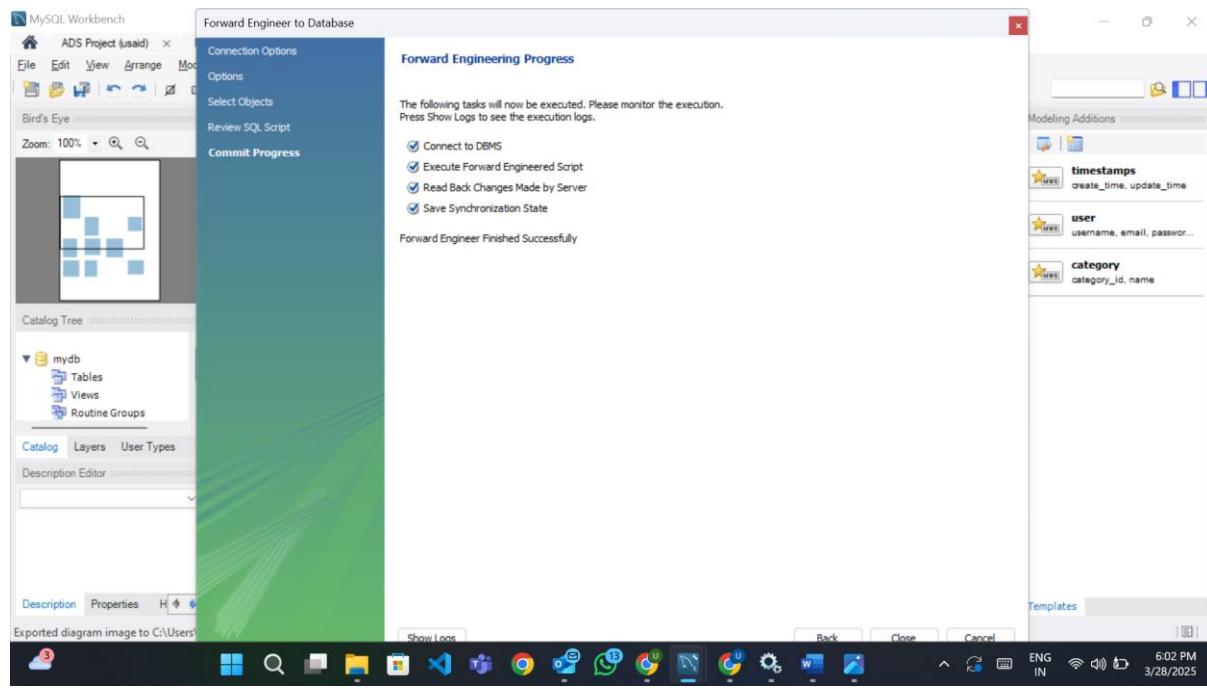
CREATE TABLE IF NOT EXISTS `usaid`.`instructor` (
`InstructorID` INT NOT NULL,
`Name` VARCHAR(100) NULL DEFAULT NULL,
`Department` VARCHAR(100) NULL DEFAULT NULL,
`Email` VARCHAR(255) NULL DEFAULT NULL,
`ContactNo` VARCHAR(50) NULL DEFAULT NULL,
PRIMARY KEY (`InstructorID`))
ENGINE = InnoDB
DEFAULT CHARACTER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

The screenshot shows the SSMS interface with the following details:

- File Edit View Query Database Server Tools Scripting Help**
- Navigator** pane shows SCHEMAS: **usaid**.
- Query 1** pane contains the following SQL code:


```

107 (3, 3, 3, '2020-09-02', 'Present'),
108 (4, 4, 4, '2021-01-18', 'Present'),
109 (5, 5, 5, '2021-09-01', 'Present');

110
111 • INSERT INTO Grade (GradeID, StudentID, CourseID, Semester, Year, Grade)
VALUES
(1, 1, 1, 'Fall', 2020, 'A'),
(2, 2, 2, 'Spring', 2021, 'B'),
(3, 3, 3, 'Fall', 2020, 'A'),
(4, 4, 4, 'Spring', 2021, 'C'),
(5, 5, 5, 'Fall', 2021, 'B');

118
119 • INSERT INTO Fee (FeeID, StudentID, Amount, DueDate, PaymentStatus, ScholarshipID)
VALUES
      
```
- Output** pane shows the execution log with the following entries:

#	Time	Action	Message	Duration / Fetch
15	17:54:03	CREATE TABLE Fee (FeeID INT PRIMARY KEY, StudentID INT, Amount DEC...)	0 row(s) affected	0.047 sec
16	18:06:31	INSERT INTO Student (StudentID, Name, DOB, ContactInfo, Address, AdmissionYear, ...)	1 row(s) affected	0.000 sec
17	18:09:35	INSERT INTO Student (StudentID, Name, DOB, ContactInfo, Address, AdmissionYear, ...)	Error Code: 1062. Duplicate entry '1' for key 'student.PRIMARY'	0.015 sec
18	18:09:55	INSERT INTO Scholarship (ScholarshipID, Name, EligibilityCriteria, Amount) VALUES (1, 'Need-Based Scholarship', 'Financial need', 400.00), (2, 'Sports Scholarship', 'Athletic achievement', 300.00), (3, 'Research Scholarship', 'Research contributions', 600.00), (4, 'Community Service Scholarship', 'Community service involvement', 450.00);	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0	0.000 sec
- Object Info Session** pane.
- Query Completed** message.

The screenshot shows the SSMS interface with the following details:

- File Edit View Query Database Server Tools Scripting Help**
- Navigator** pane shows SCHEMAS: **usaid**.
- Query 1** pane contains the following SQL code:


```

130 (2, 'Need-Based Scholarship', 'Financial need', 400.00),
131 (3, 'Sports Scholarship', 'Athletic achievement', 300.00),
132 (4, 'Research Scholarship', 'Research contributions', 600.00),
133 (5, 'Community Service Scholarship', 'Community service involvement', 450.00);

134
135 • select * FROM Scholarship;
      
```
- Result Grid** pane displays the data from the Scholarship table:

ScholarshipID	Name	EligibilityCriteria	Amount
1	Merit-Based Scholarship	Top 10% of the class	500.00
2	Need-Based Scholarship	Financial need	400.00
3	Sports Scholarship	Athletic achievement	300.00
4	Research Scholarship	Research contributions	600.00
5	Community Service Scholarship	Community service involvement	450.00
- Output** pane shows the execution log with the following entries:

#	Time	Action	Message	Duration / Fetch
26	18:12:37	INSERT INTO Fee (FeeID, StudentID, Amount, DueDate, PaymentStatus, ScholarshipID)	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0	0.000 sec
27	18:12:41	INSERT INTO Scholarship (ScholarshipID, Name, EligibilityCriteria, Amount) VALUES (1, 'Need-Based Scholarship', 'Financial need', 400.00), (2, 'Sports Scholarship', 'Athletic achievement', 300.00), (3, 'Research Scholarship', 'Research contributions', 600.00), (4, 'Community Service Scholarship', 'Community service involvement', 450.00);	Error Code: 1062. Duplicate entry '1' for key 'scholarship.PRIMARY'	0.000 sec
28	18:12:58	INSERT INTO Scholarship (ScholarshipID, Name, EligibilityCriteria, Amount) VALUES (1, 'Need-Based Scholarship', 'Financial need', 400.00), (2, 'Sports Scholarship', 'Athletic achievement', 300.00), (3, 'Research Scholarship', 'Research contributions', 600.00), (4, 'Community Service Scholarship', 'Community service involvement', 450.00);	Error Code: 1062. Duplicate entry '2' for key 'scholarship.PRIMARY'	0.000 sec
29	18:13:22	select * FROM Scholarship	Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'select * FROM Scholarship' at line 1	0.000 sec
30	18:13:41	select * FROM Scholarship LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
- Object Info Session** pane.

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

```

130 (2, 'Need-Based Scholarship', 'Financial need', 400.00),
131 (3, 'Sports Scholarship', 'Athletic achievement', 300.00),
132 (4, 'Research Scholarship', 'Research contributions', 600.00),
133 (5, 'Community Service Scholarship', 'Community service involvement', 450.00);
134
135 • select * FROM Student;

```

StudentID	Name	DOB	ContactInfo	Address	AdmissionYear	CourseEnrolled	FeeStatus
1	John Doe	2000-01-01	123-456-7890	123 Main St	2020	CS101	Paid
2	Jane Smith	2001-02-14	234-567-8901	456 Elm St	2021	MATH101	Unpaid
3	Mike Johnson	1999-07-22	345-678-9012	789 Oak St	2019	CS102	Paid
4	Emily Davis	2000-05-17	456-789-0123	101 Pine St	2020	BIO101	Paid
5	David Brown	2001-11-05	567-890-1234	202 Maple St	2021	CHEM101	Unpaid

Schema: usaid

Action Output

#	Time	Action	Message	Duration / Fetch
27	18:12:41	INSERT INTO Scholarship (ScholarshipID, Name, EligibilityCriteria, Amount) VALUE...	Error Code: 1062. Duplicate entry '1' for key 'scholarship.PRIMARY'	0.000 sec
28	18:12:58	INSERT INTO Scholarship (ScholarshipID, Name, EligibilityCriteria, Amount) VALUE...	Error Code: 1062. Duplicate entry '2' for key 'scholarship.PRIMARY'	0.000 sec
29	18:13:22	select * Scholarship	Error Code: 1064. You have an error in your SQL syntax; check the manual that cor...	0.000 sec
30	18:13:41	select * FROM Scholarship LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
31	18:14:15	select * FROM Student LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec

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:

```

1 • SELECT * FROM usaid.attendance;

```

AttendanceID	StudentID	CourseID	Date	Status
1	1	1	2020-09-01	Present
2	2	2	2021-01-15	Absent
3	3	3	2020-09-02	Present
4	4	4	2021-01-18	Present
5	5	5	2021-09-01	Present
NULL	NULL	NULL	NULL	NULL

Table: attendance

Columns:

AttendanceID	int PK
StudentID	int
CourseID	int
Date	date
Status	varchar(50)

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

COURSE:

DATABASE DESIGN PART3 – Physical design and working

USAIDKHAN

The screenshot shows the MySQL Workbench interface with the 'course' table selected in the Navigator. The table structure is as follows:

```

Table: course
Columns:
CourseID int PK
CourseName varchar(255)
Credits int
Department varchar(100)
InstructorID int

```

The Result Grid displays the following data:

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
NULL	NULL	NULL	NULL	NULL

The Output pane shows the execution history:

#	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

EMPLOYEE:

The screenshot shows the MySQL Workbench interface with the 'employee' table selected in the Navigator. The table structure is as follows:

```

Table: employee
Columns:
EMPID int PK
FIRSTNAME varchar(225)
LASTNAME varchar(225)
EMPAGE varchar(225)

```

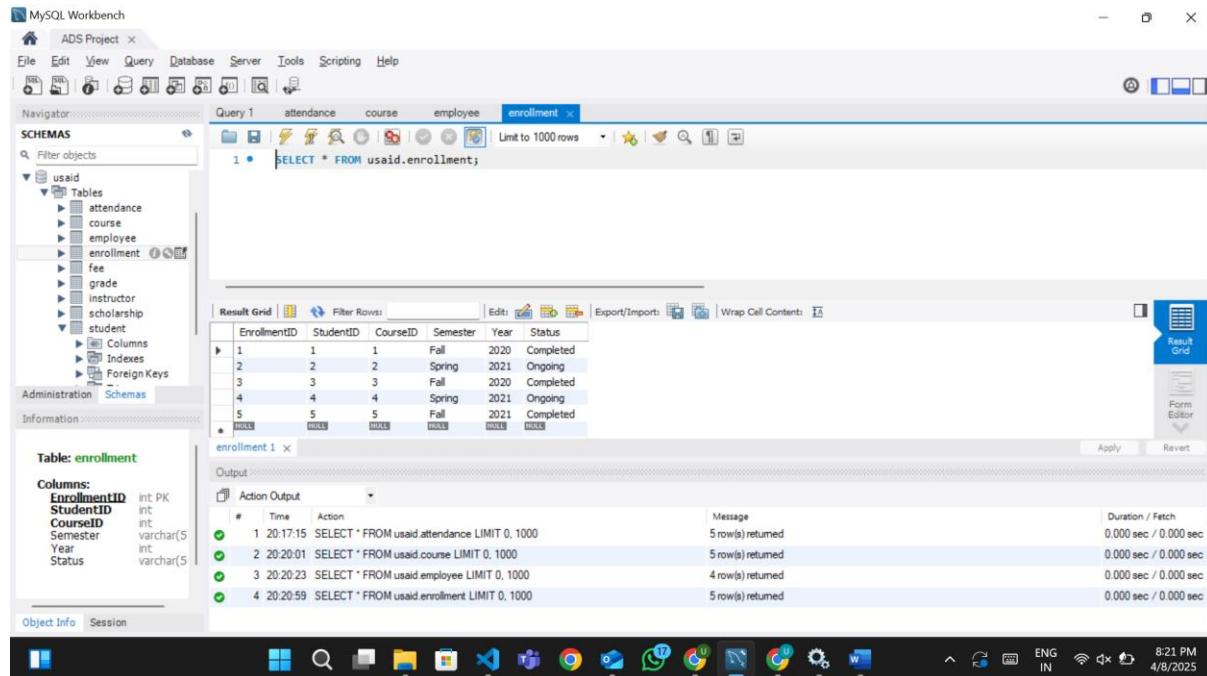
The Result Grid displays the following data:

EMPID	FIRSTNAME	LASTNAME	EMPAGE
1	KKK	Khan	25
2	Usaid	Khan	35
3	obad	Khan	45
4	barbad	Khan	55
NULL	NULL	NULL	NULL

The Output pane shows the execution history:

#	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

ENROLLMENT:



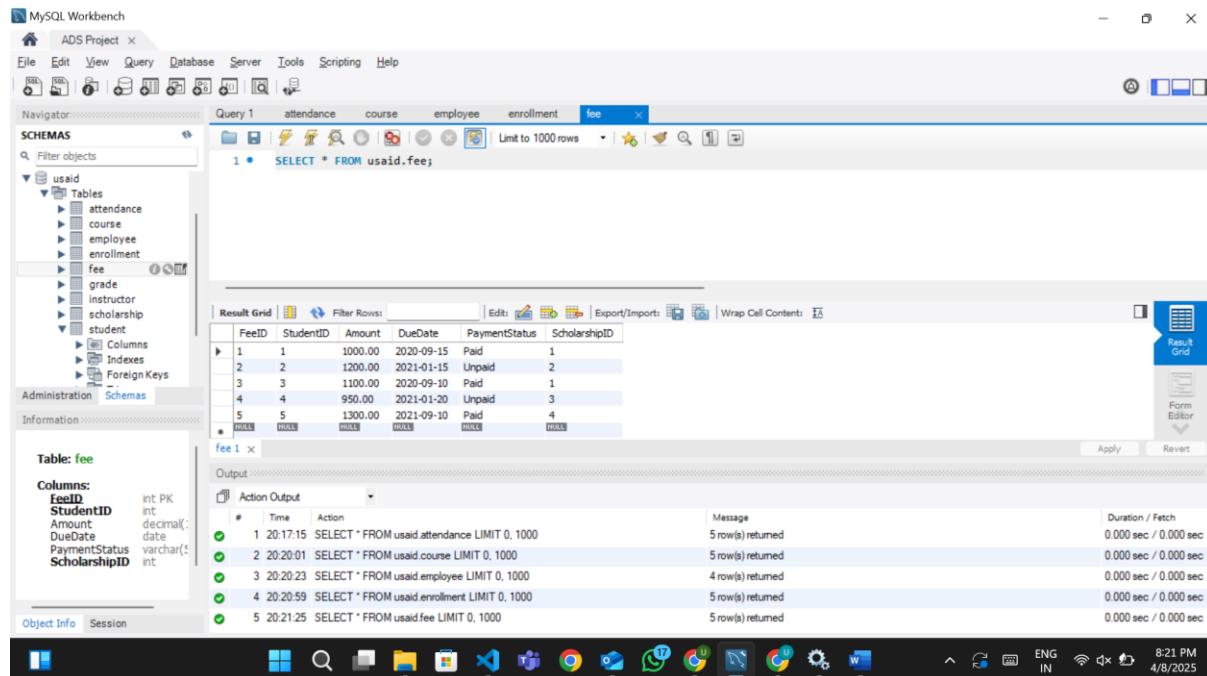
The screenshot shows the MySQL Workbench interface with the 'ADS Project' selected. The 'enrollment' table is currently selected in the query editor. The results grid displays the following data:

EnrollmentID	StudentID	CourseID	Semester	Year	Status
1	1	1	Fall	2020	Completed
2	2	2	Spring	2021	Ongoing
3	3	3	Fall	2020	Completed
4	4	4	Spring	2021	Ongoing
5	5	5	Fall	2021	Completed

The output pane shows the following log entries:

#	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
4	20:20:59	SELECT * FROM usaid.enrollment LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec

FEE:



The screenshot shows the MySQL Workbench interface with the 'ADS Project' selected. The 'fee' table is currently selected in the query editor. The results grid displays the following data:

FeeID	StudentID	Amount	DueDate	PaymentStatus	ScholarshipID
1	1	1000.00	2020-09-15	Paid	1
2	2	1200.00	2021-01-15	Unpaid	2
3	3	1100.00	2020-09-10	Paid	1
4	4	950.00	2021-01-20	Unpaid	3
5	5	1300.00	2021-09-10	Paid	4

The output pane shows the following log entries:

#	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
4	20:20:59	SELECT * FROM usaid.enrollment LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
5	20:21:25	SELECT * FROM usaid.fee LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec

GRADE:

The screenshot shows the MySQL Workbench interface with the 'ADS Project' selected. In the Navigator pane, under the 'usaid' schema, the 'grade' table is selected. The 'Tables' section lists other tables like attendance, course, employee, enrollment, fee, and student. The 'grade' table has columns GradeID (int PK), StudentID (int), CourseID (int), Semester (varchar(50)), Year (int), and Grade (varchar(5)). A query window titled 'Query 1' displays the following SQL statement and its results:

```
1 • SELECT * FROM usaid.grade;
```

Result Grid

GradeID	StudentID	CourseID	Semester	Year	Grade
1	1	1	Fall	2020	A
2	2	2	Spring	2021	B
3	3	3	Fall	2020	A
4	4	4	Spring	2021	C
5	5	5	Fall	2021	B
NULL	NULL	NULL	NULL	NULL	NULL

Action Output

#	Time	Action	Message	Duration / Fetch
1	20:20:01	SELECT * FROM usaid.course LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
2	20:20:23	SELECT * FROM usaid.employee LIMIT 0, 1000	4 row(s) returned	0.000 sec / 0.000 sec
3	20:20:59	SELECT * FROM usaid.enrollment LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
4	20:21:25	SELECT * FROM usaid.fee LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
5	20:21:49	SELECT * FROM usaid.grade LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
6	20:21:49	SELECT * FROM usaid.instructor LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec

INSTRUCTOR:

The screenshot shows the MySQL Workbench interface with the 'ADS Project' selected. In the Navigator pane, under the 'usaid' schema, the 'instructor' table is selected. The 'Tables' section lists other tables like attendance, course, employee, enrollment, fee, and grade. The 'instructor' table has columns InstructorID (int PK), Name (varchar(100)), Department (varchar(100)), Email (varchar(255)), and ContactNo (varchar(50)). A query window titled 'Query 1' displays the following SQL statement and its results:

```
1 • SELECT * FROM usaid.instructor;
```

Result Grid

InstructorID	Name	Department	Email	ContactNo
1	Dr. Alice Green	Computer Science	alice.green@example.com	123-456-7890
2	Dr. Bob White	Mathematics	bob.white@example.com	234-567-8901
3	Dr. Carol Black	Biology	carol.black@example.com	345-678-9012
4	Dr. David Blue	Chemistry	david.blue@example.com	456-789-0123
5	Dr. Eva Yellow	Physics	eva.yellow@example.com	567-890-1234
NULL	NULL	NULL	NULL	NULL

Action Output

#	Time	Action	Message	Duration / Fetch
1	20:20:23	SELECT * FROM usaid.employee LIMIT 0, 1000	4 row(s) returned	0.000 sec / 0.000 sec
2	20:20:59	SELECT * FROM usaid.enrollment LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
3	20:21:25	SELECT * FROM usaid.fee LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
4	20:21:49	SELECT * FROM usaid.grade LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
5	20:22:12	SELECT * FROM usaid.instructor LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec

SCHOLARSHIP:

The screenshot shows the MySQL Workbench interface with the 'ADS Project' selected. In the Navigator pane, under the 'usaid' schema, the 'scholarship' table is selected. The table has the following structure:

```

Table: scholarship
Columns:
  ScholarshipID int PK
  Name varchar(255)
  EligibilityCriteria text
  Amount decimal(10,2)
  
```

In the Query Editor, the query `SELECT * FROM usaid.scholarship;` is run, resulting in the following data:

ScholarshipID	Name	EligibilityCriteria	Amount
1	Merit-Based Scholarship	Top 10% of the class	500.00
2	Need-Based Scholarship	Financial need	400.00
3	Sports Scholarship	Athletic achievement	300.00
4	Research Scholarship	Research contributions	600.00
5	Community Service Scholarship	Community service involvement	450.00
NULL	NULL	NULL	NULL

The Output pane shows the execution history:

#	Time	Action	Message	Duration / Fetch
4	20:20:59	SELECT * FROM usaid.enrollment LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
5	20:21:25	SELECT * FROM usaid.fee LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
6	20:21:49	SELECT * FROM usaid.grade LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
7	20:22:12	SELECT * FROM usaid.instructor LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
8	20:22:40	SELECT * FROM usaid.scholarship LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec

Student:

The screenshot shows the MySQL Workbench interface with the 'ADS Project' selected. In the Navigator pane, under the 'usaid' schema, the 'student' table is selected. The table has the following structure:

```

Table: student
Columns:
  StudentID int PK
  Name varchar(100)
  DOB date
  ContactInfo varchar(20)
  Address text
  AdmissionYear int
  CourseEnrolled varchar(2)
  FeeStatus varchar(5)
  
```

In the Query Editor, the query `SELECT * FROM usaid.student;` is run, resulting in the following data:

StudentID	Name	DOB	ContactInfo	Address	AdmissionYear	CourseEnrolled	FeeStatus
1	John Doe	2000-01-01	123-456-7890	123 Main St	2020	CS101	Paid
2	Jane Smith	2001-02-14	345-567-8901	456 Elm St	2021	MATH101	Unpaid
3	Mike Johnson	1999-07-22	345-678-9012	789 Oak St	2019	CS102	Paid
4	Emily Davis	2000-05-17	456-789-0123	101 Pine St	2020	BIO101	Paid
5	David Brown	2001-11-05	567-890-1234	202 Maple St	2021	CHEM101	Unpaid
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

The Output pane shows the execution history:

#	Time	Action	Message	Duration / Fetch
5	20:21:25	SELECT * FROM usaid.fee LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
6	20:21:49	SELECT * FROM usaid.grade LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
7	20:22:12	SELECT * FROM usaid.instructor LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
8	20:22:40	SELECT * FROM usaid.scholarship LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
9	20:23:01	SELECT * FROM usaid.student LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec

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

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

- Filter objects
- dbtheatre
- supermarket
- sys
- usaid**

Query 1

```
135 • select * FROM Student;
136
137 • SELECT Student.Name, Course.CourseName
138   FROM Student
139   JOIN Enrollment ON Student.StudentID = Enrollment.StudentID
140   JOIN Course ON Enrollment.CourseID = Course.CourseID;
```

Result Grid | Filter Rows: Export: Wrap Cell Content: Result Grid

Name	CourseName
John Doe	Computer Science 101
Jane Smith	Mathematics 101
Mike Johnson	Biology 101
Emily Davis	Chemistry 101
David Brown	Physics 101

Result 4 x

Action Output

#	Time	Action	Message	Duration / Fetch
28	18:12:58	INSERT INTO Scholarship (ScholarshipID, Name, EligibilityCriteria, Amount) VALUE...	Error Code: 1062. Duplicate entry '2' for key 'scholarship.PRIMARY'	0.000 sec
29	18:13:22	select * Scholarship	Error Code: 1064. You have an error in your SQL syntax; check the manual that corr...	0.000 sec
30	18:13:41	select * FROM Scholarship LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
31	18:14:15	select * FROM Student LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
32	18:15:35	SELECT Student.Name, Course.CourseName FROM Student JOIN Enrollment ON ...	5 row(s) returned	0.000 sec / 0.000 sec

Schema: usaid

Object Info Session

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

- Filter objects
- dbtheatre
- supermarket
- sys
- usaid**

Query 1

```
142
143 • SELECT Student.Name AS StudentName, Fee.Amount AS FeeAmount, Scholarship.Name AS ScholarshipName
144   FROM Student
145   JOIN Fee ON Student.StudentID = Fee.StudentID
146   JOIN Scholarship ON Fee.ScholarshipID = Scholarship.ScholarshipID;
```

Result Grid | Filter Rows: Export: Wrap Cell Content: Result Grid

StudentName	FeeAmount	ScholarshipName
John Doe	1000.00	Merit-Based Scholarship
Jane Smith	1200.00	Need-Based Scholarship
Mike Johnson	1100.00	Merit-Based Scholarship
Emily Davis	950.00	Sports Scholarship
David Brown	1300.00	Research Scholarship

Result 5 x

Action Output

#	Time	Action	Message	Duration / Fetch
29	18:13:22	select * Scholarship	Error Code: 1064. You have an error in your SQL syntax; check the manual that corr...	0.000 sec
30	18:13:41	select * FROM Scholarship LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
31	18:14:15	select * FROM Student LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
32	18:15:35	SELECT Student.Name, Course.CourseName FROM Student JOIN Enrollment ON ...	5 row(s) returned	0.000 sec / 0.000 sec
33	18:16:34	SELECT Student.Name AS StudentName, Fee.Amount AS FeeAmount, Scholarshi...	5 row(s) returned	0.015 sec / 0.000 sec

Schema: usaid

Object Info Session

Query Completed

STEP 5:

VIEW STUDENT COURSE SUMMARY

The screenshot shows the Oracle SQL Developer interface. In the top-left corner, the 'Navigator' pane displays the 'SCHEMAS' section with 'usaid' selected. The main area, 'Query 1', contains the following SQL code:

```

145 JOIN Fee ON Student.StudentID = Fee.StudentID
146 JOIN Scholarship ON Fee.ScholarshipID = Scholarship.ScholarshipID;
147
148 • CREATE VIEW StudentCourseSummary AS
149     SELECT
150         Student.Name AS StudentName,
151         Course.CourseName AS CourseName,
152         Enrollment.Semester AS Semester,
153         Enrollment.Year AS Year
154     FROM Student
155     JOIN Enrollment ON Student.StudentID = Enrollment.StudentID
156     JOIN Course ON Enrollment.CourseID = Course.CourseID;
157

```

Below the code, the 'Output' pane shows the execution log:

#	Time	Action	Message	Duration / Fetch
30	18:13:41	select * FROM Scholarship LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
31	18:14:15	select * FROM Student LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
32	18:15:35	SELECT Student.Name, Course.CourseName FROM Student JOIN Enrollment ON ...	5 row(s) returned	0.000 sec / 0.000 sec
33	18:16:34	SELECT Student.Name AS StudentName, Fee.Amount AS FeeAmount, Scholarshi... 5 row(s) returned	0.015 sec / 0.000 sec	
34	18:18:17	CREATE VIEW StudentCourseSummary AS SELECT Student.Name AS Student... 0 row(s) affected	0.015 sec	

VIEW FEESTATUSOVERVIEW

The screenshot shows the Oracle SQL Developer interface. In the top-left corner, the 'Navigator' pane displays the 'SCHEMAS' section with 'usaid' selected. The main area, 'Query 1', contains the following SQL code:

```

155 JOIN Enrollment ON Student.StudentID = Enrollment.StudentID
156 JOIN Course ON Enrollment.CourseID = Course.CourseID;
157
158 • CREATE VIEW FeeStatusOverview AS
159     SELECT
160         Student.Name AS StudentName,
161         Fee.Amount AS FeeAmount,
162         Fee.PaymentStatus AS PaymentStatus,
163         Scholarship.Name AS ScholarshipName
164     FROM Fee
165     JOIN Student ON Fee.StudentID = Student.StudentID
166     JOIN Scholarship ON Fee.ScholarshipID = Scholarship.ScholarshipID;
167

```

Below the code, the 'Output' pane shows the execution log:

#	Time	Action	Message	Duration / Fetch
31	18:14:15	select * FROM Student LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
32	18:15:35	SELECT Student.Name, Course.CourseName FROM Student JOIN Enrollment ON ...	5 row(s) returned	0.000 sec / 0.000 sec
33	18:16:34	SELECT Student.Name AS StudentName, Fee.Amount AS FeeAmount, Scholarshi... 5 row(s) returned	0.015 sec / 0.000 sec	
34	18:18:17	CREATE VIEW FeeStatusOverview AS SELECT Student.Name AS Student... 0 row(s) affected	0.015 sec	
35	18:19:03	CREATE VIEW FeeStatusOverview AS SELECT Student.Name AS StudentNa... 0 row(s) affected	0.000 sec	

TESTED VIEWS:

MySQL Workbench

ADS Project

File Edit View Query Database Server Tools Scripting Help

Navigator

Schemas

Filter objects

Views feestatusoverview studentcoursesummary Stored Procedures EnrollStudentInCourse UpdateFeeStatus Functions GetAverageGrade GetTotalFee

Administration Schemas Information

View: feestatusoverview

Columns:

StudentName	FeeAmount	PaymentStatus	ScholarshipName
John Doe	1000.00	Paid	Merit-Based Scholarship
Jane Smith	1200.00	Unpaid	Need-Based Scholarship
Mike Johnson	1100.00	Paid	Merit-Based Scholarship
Emily Davis	950.00	Unpaid	Sports Scholarship
David Brown	1300.00	Paid	Research Scholarship

Result Grid | Filter Rows! | Export | Wrap Cell Content:

Output

Action Output

#	Time	Action	Message	Duration / Fetch
8	20:22:40	SELECT * FROM usaid.feestatusoverview LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
9	20:23:01	SELECT * FROM usaid.student LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
10	20:35:15	SELECT * FROM usaid.feestatusoverview LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
11	20:35:24	SELECT * FROM usaid.studentcoursesummary LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
12	20:36:12	SELECT * FROM usaid.feestatusoverview LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
13	20:36:26	SELECT * FROM usaid.feestatusoverview LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec

Result Grid Form Editor Read Only

Object Info Session

8:36 PM IN 4/8/2025

MySQL Workbench

ADS Project

File Edit View Query Database Server Tools Scripting Help

Navigator

Schemas

Filter objects

Views feestatusoverview studentcoursesummary Stored Procedures EnrollStudentInCourse UpdateFeeStatus Functions GetAverageGrade GetTotalFee

Administration Schemas Information

View: studentcoursesummary

Columns:

StudentName	CourseName	Semester	Year
John Doe	Computer Science 101	Fall	2020
Jane Smith	Mathematics 101	Spring	2021
Mike Johnson	Biology 101	Fall	2020
Emily Davis	Chemistry 101	Spring	2021
David Brown	Physics 101	Fall	2021

Result Grid | Filter Rows! | Export | Wrap Cell Content:

Output

Action Output

#	Time	Action	Message	Duration / Fetch
9	20:23:01	SELECT * FROM usaid.student LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
10	20:35:15	SELECT * FROM usaid.feestatusoverview LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
11	20:35:24	SELECT * FROM usaid.studentcoursesummary LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
12	20:36:12	SELECT * FROM usaid.feestatusoverview LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
13	20:36:26	SELECT * FROM usaid.studentcoursesummary LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec

Result Grid Form Editor Read Only

Object Info Session

8:36 PM IN 4/8/2025

FUNCTIONS

Function 1: Get Total Fee for a Student

The screenshot shows the SSMS interface with the following details:

- Schemas:** The left pane shows the database structure under the schema `usaid`, which contains tables like `dbtheatre`, `supermarket`, and `sys`, as well as views, stored procedures, and functions.
- Query Editor:** The main window displays a T-SQL script for creating a function named `GetTotalFee`. The script uses a cursor to iterate through the `Fee` table and calculate the total fee for each student by subtracting scholarship amounts from fee amounts. It includes error handling for null values in the scholarship amount.
- Output Window:** Below the editor, the output window shows the history of recent database operations, including the creation of the `GetTotalFee` function and other objects.
- Object Info and Session:** At the bottom, there are tabs for "Object Info" and "Session".

Function 2: Get Average Grade for a Student

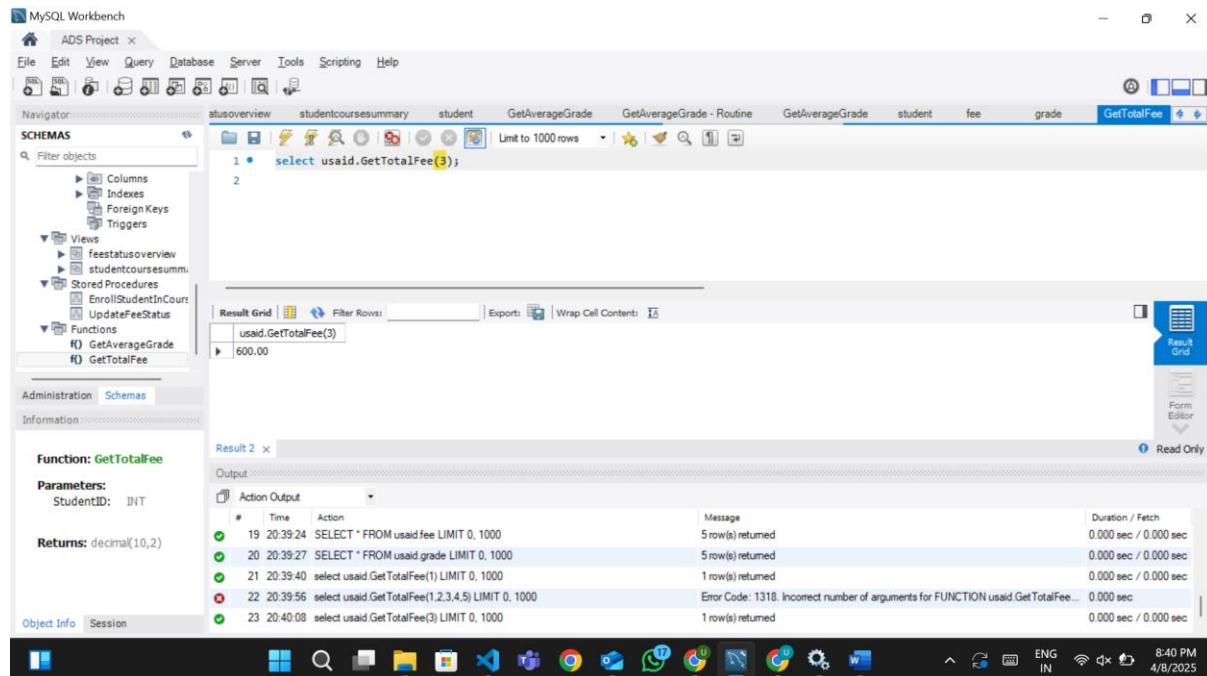
The screenshot shows the Oracle SQL Developer interface. The left sidebar displays the Navigator with the schema 'usaid' selected, showing tables like 'employee', views, stored procedures, and functions. The main area is titled 'Query 1' and contains the following SQL code:

```
177 DELIMITER $$  
178  
179 • CREATE FUNCTION GetAverageGrade(StudentID INT)  
180     RETURN VARCHAR(5);  
181     DETERMINISTIC  
182     READS SQL DATA  
183 BEGIN  
184     DECLARE avgGrade VARCHAR(5);  
185     SELECT AVG(CASE Grade  
186             WHEN 'A' THEN 4  
187             WHEN 'B' THEN 3  
188             WHEN 'C' THEN 2  
189             WHEN 'D' THEN 1  
190         ELSE 0  
191     END) INTO avgGrade;  
192     RETURN avgGrade;  
193 END;  
194 $$
```

The bottom section, 'Output', shows the execution log:

Action Output	#	Time	Action	Message	Duration / Fetch
37	18:21:17	USE usaid		0 row(s) affected	0.000 sec
38	18:21:17	CREATE TABLE Course (CourseID INT PRIMARY KEY, CourseName VARCHAR(50))		Error Code: 1050. Table 'course' already exists	0.000 sec
39	18:21:31	CREATE FUNCTION GetTotalFee(StudentID INT) RETURNS DECIMAL(10,2)		0 row(s) affected	0.016 sec
40	18:22:49	CREATE FUNCTION GetAverageGrade(StudentID INT) RETURNS VARCHAR(5)		Error Code: 1418. This function has none of DETERMINISTIC, NO SQL, or READS SQL DATA specified.	0.000 sec
41	18:23:20	CREATE FUNCTION GetAverageGrade(StudentID INT) RETURNS VARCHAR(5)		0 row(s) affected	0.000 sec

TESTED FUNCTIONS: Get total Fee



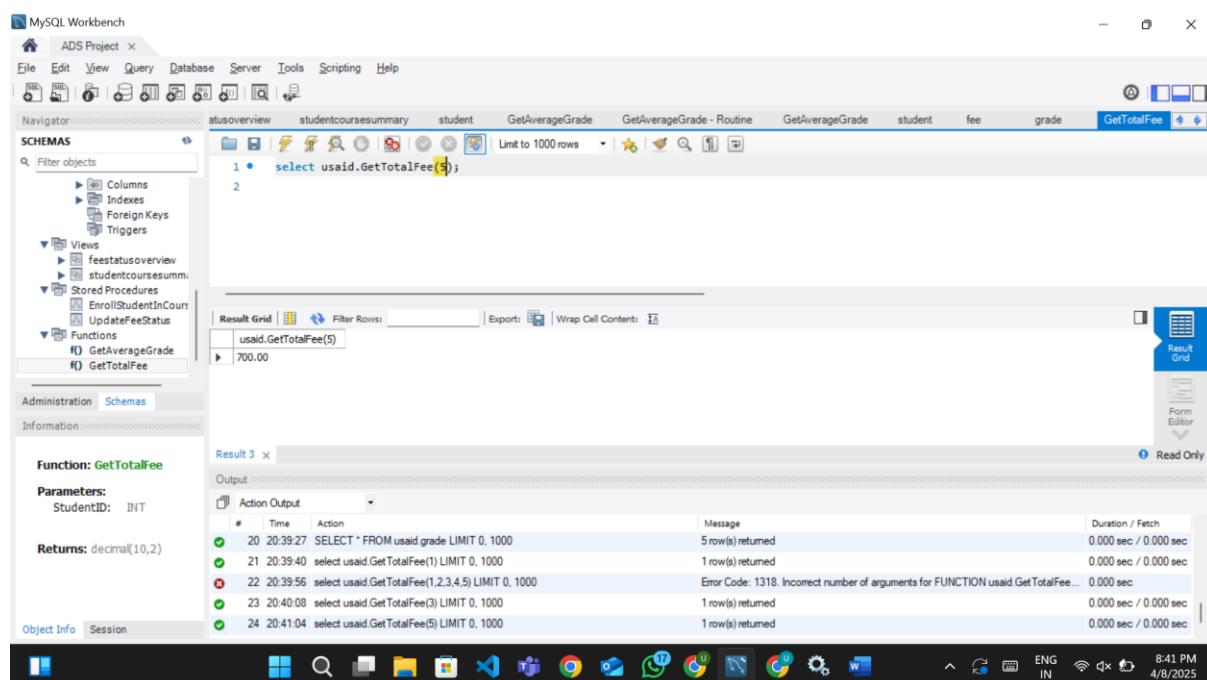
```
MySQL Workbench - ADS Project
```

Result Grid

1	usaid.GetTotalFee(3);
	600.00

Output

#	Time	Action	Message	Duration / Fetch
19	20:39:24	SELECT * FROM usaid.fee LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
20	20:39:27	SELECT * FROM usaid.grade LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
21	20:39:40	select usaid.GetTotalFee() LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
22	20:39:56	select usaid.GetTotalFee(1,2,3,4,5) LIMIT 0, 1000	Error Code: 1318. Incorrect number of arguments for FUNCTION usaid.GetTotalFee...	0.000 sec
23	20:40:08	select usaid.GetTotalFee(3) LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec



```
MySQL Workbench - ADS Project
```

Result Grid

1	usaid.GetTotalFee(5);
	700.00

Output

#	Time	Action	Message	Duration / Fetch
20	20:39:27	SELECT * FROM usaid.grade LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
21	20:39:40	select usaid.GetTotalFee() LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
22	20:39:56	select usaid.GetTotalFee(1,2,3,4,5) LIMIT 0, 1000	Error Code: 1318. Incorrect number of arguments for FUNCTION usaid.GetTotalFee...	0.000 sec
23	20:40:08	select usaid.GetTotalFee(3) LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
24	20:41:04	select usaid.GetTotalFee(5) LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec

The screenshot shows the MySQL Workbench interface. In the top navigation bar, 'ADS Project' is selected. The left sidebar under 'Schemas' shows various database objects like Views, Stored Procedures, and Functions. The 'Functions' section contains two entries: 'GetAverageGrade' and 'GetTotalFee'. The 'GetTotalFee' entry is selected, and its details are shown in the center pane. The 'Parameters' section shows 'StudentID: INT' and the 'Returns' section shows 'decimal(10,2)'. Below this, the 'Object Info' and 'Session' tabs are visible.

In the main query editor area, the following SQL code is written:

```
1 • select usaid.GetTotalFee(1);
2
```

The result grid shows the output of the function call:

Result Grid	Filter Rows:	Exports:	Wrap Cell Content:
usaid.GetTotalFee(1)			
500.00			

Below the result grid, the log shows the following activity:

#	Time	Action	Message	Duration / Fetch
21	20:39:40	select usaid.GetTotalFee(1) LIMIT 0, 1000	1row(s) returned	0.000 sec / 0.000 sec
22	20:39:56	select usaid.GetTotalFee(1,2,3,4,5) LIMIT 0, 1000	Error Code: 1318. Incorrect number of arguments for FUNCTION usaid.GetTotalFee...	0.000 sec
23	20:40:08	select usaid.GetTotalFee(3) LIMIT 0, 1000	1row(s) returned	0.000 sec / 0.000 sec
24	20:41:04	select usaid.GetTotalFee(5) LIMIT 0, 1000	1row(s) returned	0.000 sec / 0.000 sec
25	20:41:50	select usaid.GetTotalFee(1) LIMIT 0, 1000	1row(s) returned	0.000 sec / 0.000 sec

The status bar at the bottom right indicates '8:41 PM 4/8/2025'.

Get Average Grade of Students

The screenshot shows the MySQL Workbench interface. The left sidebar under 'Schemas' shows various database objects. The 'Functions' section contains two entries: 'GetAverageGrade' and 'GetTotalFee'. The 'GetAverageGrade' entry is selected, and its details are shown in the center pane. The 'Function' section shows 'GetAverageGrade' and the 'Parameters' section shows 'StudentID' with a type of 'INT'.

In the main query editor area, a dialog box titled 'Call stored function usaid.GetAverageGrade' is open. It prompts the user to enter values for parameters and provides a SQL editor for running the call. The parameter 'StudentID' is set to 'INT'.

The result grid shows the output of the function call:

Result Grid	Filter Rows:	Exports:	Wrap Cell Content:
usaid.GetAverageGrade			
100.00			

Below the result grid, the log shows the following activity:

#	Time	Action	Message	Duration / Fetch
32	21:06:55	select usaid.GetAverageGrade(1) LIMIT 0, 1000	Error Code: 1406. Data too long for column 'avgGrade' at row 1	0.000 sec
33	21:07:07	SELECT * FROM usaid.student LIMIT 0, 1000	5row(s) returned	0.000 sec / 0.000 sec
34	21:07:14	SELECT * FROM usaid.grade LIMIT 0, 1000	5row(s) returned	0.015 sec / 0.000 sec
35	21:07:53	select usaid.GetTotalFee(3) LIMIT 0, 1000	1row(s) returned	0.000 sec / 0.000 sec
36	21:12:20	call usaid.EnrollStudentInCourse(1, 1, 'Fall', 2020)	0row(s) affected	0.000 sec

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

The screenshot shows the MySQL Workbench interface. The left sidebar under 'Schemas' shows various database objects. The 'Functions' section contains two entries: 'GetAverageGrade' and 'GetTotalFee'. The 'GetAverageGrade' entry is selected, and its details are shown in the center pane.

In the main query editor area, the following SQL code is written:

```
1 • select usaid.GetAverageGrade(1);
2
```

The result grid shows the output of the function call:

Result Grid	Filter Rows:	Exports:	Wrap Cell Content:
usaid.GetAverageGrade(1)			
100.00			

Below the result grid, the log shows the following activity:

#	Time	Action	Message	Duration / Fetch
33	21:07:07	SELECT * FROM usaid.student LIMIT 0, 1000	5row(s) returned	0.000 sec / 0.000 sec
34	21:07:14	SELECT * FROM usaid.grade LIMIT 0, 1000	5row(s) returned	0.015 sec / 0.000 sec
35	21:07:53	select usaid.GetTotalFee(3) LIMIT 0, 1000	1row(s) returned	0.000 sec / 0.000 sec
36	21:12:20	call usaid.EnrollStudentInCourse(1, 1, 'Fall', 2020)	0row(s) affected	0.000 sec
37	18:39:25	select usaid.GetAverageGrade(1) LIMIT 0, 1000	Error Code: 1406. Data too long for column 'avgGrade' at row 1	0.000 sec

A yellow box highlights the error message in the log: 'Error Code: 1406. Data too long for column 'avgGrade' at row 1'.

STORRED PROCEDURE:

1. Stored Procedure 1: Update Fee Status

The screenshot shows the MySQL Workbench interface with the 'Query 1' tab active. The code in the query editor is as follows:

```

202 DELIMITER $$

203

204 • CREATE PROCEDURE UpdateFeeStatus(IN inStudentID INT, IN newStatus VARCHAR(50))
205   DETERMINISTIC
206   NO SQL
207   BEGIN
208     UPDATE Fee
209     SET PaymentStatus = newStatus
210     WHERE StudentID = inStudentID;
211   END $$

212

213 DELIMITER ;

```

The 'Output' pane shows the execution results:

#	Time	Action	Message	Duration / Fetch
43	18:25:34	CREATE PROCEDURE UpdateFeeStatus(IN StudentID INT, IN newStatus VARCHAR(50))	Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'CREATE PROCEDURE UpdateFeeStatus(IN inStudentID INT, IN newStatus VARCHAR(50))' at line 1	0.000 sec
44	18:26:01	CREATE PROCEDURE UpdateFeeStatus(IN inStudentID INT, IN newStatus VARCHAR(50))	DETERMINISTIC	0.000 sec
45	18:26:24	CREATE PROCEDURE UpdateFeeStatus(IN StudentID INT, IN ne	NO SQL	0.000 sec
46	18:26:52	CREATE PROCEDURE UpdateFeeStatus(IN inStudentID INT, IN r	BEGIN	0.000 sec
47	18:27:26	CREATE PROCEDURE UpdateFeeStatus(IN inStudentID INT, IN r	UPDATE Fee	0.000 sec
			SET PaymentStatus = newStatus	
			WHERE StudentID = inStudentID; -- Use 'inStudentID' for the parameter	
			END	

The 'Object Info' and 'Session' tabs are visible at the bottom.

2. Stored Procedure: EnrollStudentInCourse (with Transaction)

The screenshot shows the MySQL Workbench interface with the 'Query 1' tab active. The code in the query editor is as follows:

```

202 DELIMITER $$

203

204 • CREATE PROCEDURE EnrollStudentInCourse(IN inStudentID INT, IN inCourseID INT, IN inSemester VARCHAR(50), IN inYear INT)
205   DETERMINISTIC
206   READS SQL DATA
207   BEGIN
208     DECLARE EXIT HANDLER FOR SQLEXCEPTION
209     BEGIN
210       -- Rollback if an error occurs
211       ROLLBACK;
212     END;
213   END;
214

215

```

The 'Output' pane shows the execution results:

#	Time	Action	Message	Duration / Fetch
44	18:26:01	CREATE PROCEDURE UpdateFeeStatus(IN StudentID INT, IN newStatus VARCHAR(50))	Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'CREATE PROCEDURE UpdateFeeStatus(IN inStudentID INT, IN newStatus VARCHAR(50))' at line 1	0.000 sec
45	18:26:24	CREATE PROCEDURE UpdateFeeStatus(IN StudentID INT, IN newStatus VARCHAR(50))	Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'DETERMINISTIC' at line 2	0.000 sec
46	18:26:52	CREATE PROCEDURE UpdateFeeStatus(IN inStudentID INT, IN newStatus VARCHAR(50))	Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'READS SQL DATA' at line 3	0.000 sec
47	18:27:26	CREATE PROCEDURE UpdateFeeStatus(IN inStudentID INT, IN newStatus VARCHAR(50))	Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'BEGIN' at line 4	0.000 sec
			DECLARE EXIT HANDLER FOR SQLEXCEPTION	
			BEGIN	
			-- Rollback if an error occurs	
			ROLLBACK;	
			END;	
48	18:29:01	CREATE PROCEDURE EnrollStudentInCourse(IN inStudentID INT, IN inCourseID INT, IN inSemester VARCHAR(50), IN inYear INT)	0 row(s) affected	0.015 sec

The 'Object Info' and 'Session' tabs are visible at the bottom.

TESTING STORED PROCEDURE: EnrollStudentInCourse

The screenshot shows the Oracle SQL Developer interface. In the top navigation bar, the path is 'fee -> table -> EnrollStudentInCourse - Routine'. The code editor contains the following SQL:

```

1 • call usaid.EnrollStudentInCourse(1, 1, 'Fall', 2020);
2 • Select * From usaid.student;

```

The Result Grid displays student data:

StudentID	Name	DOB	ContactInfo	Address	AdmissionYear	CourseEnrolled	FeeStatus
1	John Doe	2000-01-01	123-456-7890	123 Main St	2020	CS101	Paid
2	Jane Smith	2001-02-14	234-567-8901	456 Elm St	2021	MATH101	Unpaid
3	Mike Johnson	1999-07-22	345-678-9012	789 Oak St	2019	CS102	Paid
4	Emily Davis	2000-05-17	456-789-0123	101 Pine St	2020	BIO101	Paid
5	David Brown	2001-11-05	567-890-1234	202 Maple St	2021	CHEM101	Unpaid
•	NULL	NULL	NULL	NULL	NULL	NULL	NULL

The Output pane shows the execution log:

#	Time	Action	Message	Duration / Fetch
37	18:39:25	select usaid.GetAverageGrade() LIMIT 0, 1000	Error Code: 1406. Data too long for column 'avgGrade' at row 1	0.000 sec
38	18:43:18	call usaid.EnrollStudentInCourse(1, 1, 'Fall', 2020)	0 row(s) affected	0.000 sec
39	18:44:03	Select * From usaid.student LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
40	18:44:44	Select * From usaid.course LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
41	18:45:19	Select * From usaid.student LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec

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

TEST UpdateFeeStatus:

The screenshot shows the Oracle SQL Developer interface. In the top navigation bar, the path is 'fee -> table -> EnrollStudentInCourse - Routine'. The code editor contains the following SQL:

```

1 • call usaid.UpdateFeeStatus(1, 'Unpaid');
2 • Select * From usaid.fee;
3

```

The Result Grid displays fee data:

FeeID	StudentID	Amount	DueDate	PaymentStatus	ScholarshipID
1	1	1000.00	2020-09-15	Unpaid	1
2	2	1200.00	2021-01-15	Unpaid	2
3	3	1100.00	2020-09-10	Paid	1
4	4	950.00	2021-01-20	Unpaid	3
5	5	1300.00	2021-09-10	Paid	4
•	NULL	NULL	NULL	NULL	NULL

The Output pane shows the execution log:

#	Time	Action	Message	Duration / Fetch
40	18:44:44	Select * From usaid.course LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
41	18:45:19	Select * From usaid.student LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
42	18:47:33	call usaid.UpdateFeeStatus(1, 'Unpaid')	1 row(s) affected	0.000 sec
43	18:47:57	Select * From usaid.student LIMIT 0, 1000	5 row(s) returned	0.016 sec / 0.000 sec
44	18:48:23	Select * From usaid.fee LIMIT 0, 1000	5 row(s) returned	0.015 sec / 0.000 sec

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;
```

The screenshot shows the MySQL Workbench interface. The left pane displays the Navigator with the schema 'usaid' selected, showing tables like dbtheatre, supermarket, and sys, along with their respective employee, views, stored procedures, and functions. The right pane is the Query Editor titled 'Query 1'. The code being run is:

```

230
231
232
233 DELIMITER $$ 
234
235 • CREATE TRIGGER AfterFeePaymentUpdate
236 AFTER UPDATE ON Fee
237 FOR EACH ROW
238 BEGIN
239     IF NEW.Amount = 0 THEN
240         UPDATE Fee
241             SET PaymentStatus = 'Paid'
242             WHERE FeeID = NEW.FeeID;
243 END$$
  
```

The code editor highlights the trigger definition and the IF condition. Below the code, the 'Output' pane shows the execution log:

Action	Time	Action	Duration / Fetch
CREATE PROCEDURE UpdateFeeStatus(IN StudentID INT, IN FeeID INT, IN PaymentStatus VARCHAR(50))	45 18:26:24	CREATE PROCEDURE UpdateFeeStatus(IN StudentID INT, IN FeeID INT, IN PaymentStatus VARCHAR(50))	0.000 sec
CREATE PROCEDURE UpdateFeeStatus(IN inStudentID INT, IN inFeeID INT, IN inPaymentStatus VARCHAR(50))	46 18:26:52	CREATE PROCEDURE UpdateFeeStatus(IN inStudentID INT, IN inFeeID INT, IN inPaymentStatus VARCHAR(50))	0.000 sec
CREATE PROCEDURE EnrollStudentInCourse(IN inStudentID INT, IN inCourseID INT)	47 18:27:26	CREATE PROCEDURE EnrollStudentInCourse(IN inStudentID INT, IN inCourseID INT)	0.000 sec
CREATE PROCEDURE EnrollStudentInCourse(IN inStudentID INT, IN inCourseID INT)	48 18:29:01	CREATE PROCEDURE EnrollStudentInCourse(IN inStudentID INT, IN inCourseID INT)	0.015 sec
CREATE TRIGGER AfterFeePaymentUpdate AFTER UPDATE	49 18:30:32	CREATE TRIGGER AfterFeePaymentUpdate AFTER UPDATE	0.016 sec

The log shows several errors related to syntax issues in the CREATE PROCEDURE statements, which are likely artifacts from previous executions or incomplete code.

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

```

18    END IF;
19  END;
20  //
21
22  INSERT INTO fee (FeeID, Amount, PaymentStatus)
23  VALUES (1, 100.00, 'Unpaid');

```

FeeID	Amount	PaymentStatus
1	100.00	Unpaid

Output:

#	Time	Action	Message	Duration / Fetch
83	19:16:00	INSERT INTO fee_test (FeeID, Amount, PaymentStatus) VALUES (1, 100.00, 'Unp...)	Error Code: 1062. Duplicate entry '1' for key 'fee_test.PRIMARY'	0.000 sec
84	19:16:08	INSERT INTO fee_test (FeeID, Amount, PaymentStatus) VALUES (1, 100.00, 'Unp...)	Error Code: 1062. Duplicate entry '1' for key 'fee_test.PRIMARY'	0.000 sec
85	19:16:29	INSERT INTO fee_test (FeeID, Amount, PaymentStatus) VALUES (1, 100.00, 'Unp...)	Error Code: 1062. Duplicate entry '1' for key 'fee_test.PRIMARY'	0.000 sec
86	19:16:37	UPDATEfee_test SET Amount = 0 WHERE FeeID = 1;	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.000 sec
87	19:17:03	SELECT * FROMfee_test WHERE FeeID = 1 LIMIT 0, 1000;	1 row(s) returned	0.000 sec / 0.000 sec

Trigger 2: Update Grade After Enrollment

```

248
249  DELIMITER $$;
250
251  CREATE TRIGGER SetGradeToIncomplete
252  AFTER INSERT ON Enrollment
253  FOR EACH ROW
254  BEGIN
255      INSERT INTO Grade (StudentID, CourseID, Semester, Year, Grade)
256      VALUES (NEW.StudentID, NEW.CourseID, NEW.Semester, NEW.Year, 'Incomplete');
257  END $$;
258
259  DELIMITER ;

```

Output:

#	Time	Action	Message	Duration / Fetch
46	18:26:52	CREATE PROCEDURE UpdateFeeStatus(IN StudentID INT, IN newStatus VARBINARY(10))	Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'CREATE PROCEDURE UpdateFeeStatus(IN StudentID INT, IN newStatus VARBINARY(10))' at line 1	0.000 sec
47	18:27:26	CREATE PROCEDURE UpdateFeeStatus(IN StudentID INT, IN newStatus VARBINARY(10))	0 row(s) affected	0.000 sec
48	18:29:01	CREATE PROCEDURE EnrollStudentInCourse(IN StudentID INT, IN inCourseID INT)	0 row(s) affected	0.015 sec
49	18:30:32	CREATE TRIGGER AfterFeePaymentUpdate AFTER UPDATE ON Fee FOR EACH ROW	0 row(s) affected	0.016 sec
50	18:31:25	CREATE TRIGGER SetGradeToIncomplete AFTER INSERT ON Enrollment FOR EACH ROW	0 row(s) affected	0.016 sec

```

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

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	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;
```

The screenshot shows a database management interface. On the left, the 'Schemas' tree is expanded, showing the 'student' schema with its tables ('grade', 'enrollment', 'student'), views ('feestatusoverview', 'studentcoursesummary'), stored procedures ('EnrollStudentInCourt', 'UpdateFeeStatus'), and functions ('GetAverageGrade'). The 'UpdateFeeStatus' procedure is selected. In the center, the SQL editor window displays the following code:

```
1 • CALL UpdateFeeStatus(999, 'Paid');  
2
```

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.