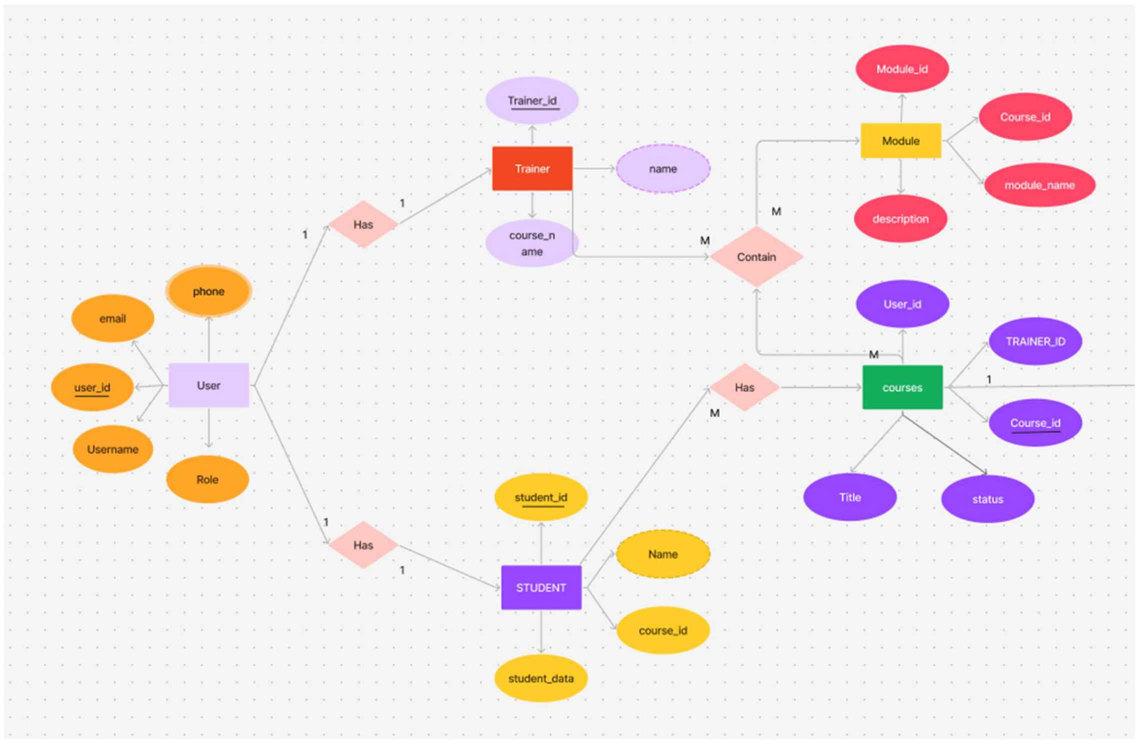
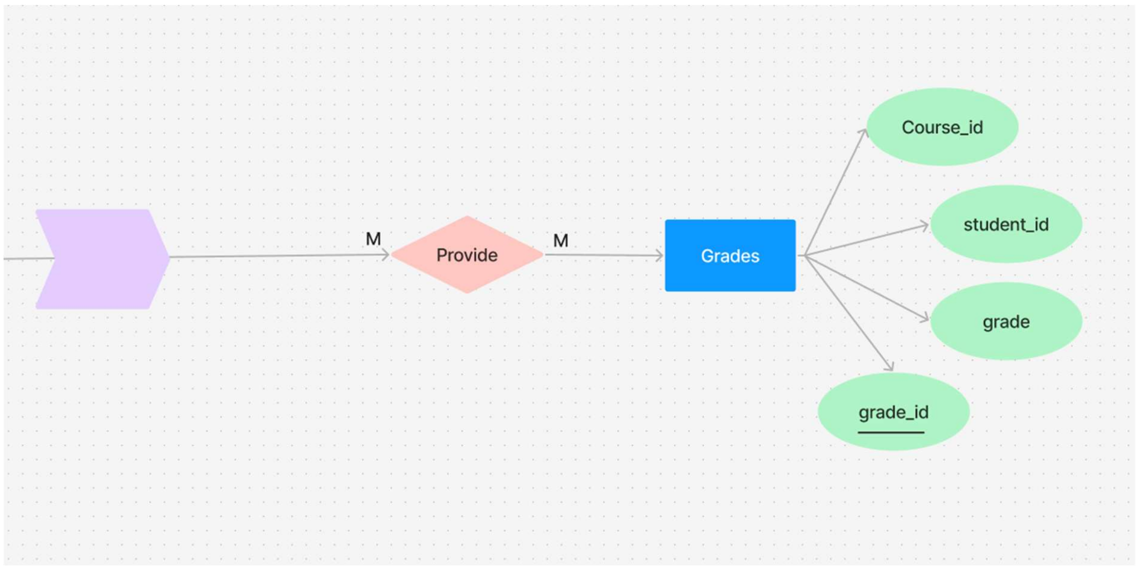
Case Study on Edu Tech Website

This case study explores the creation and management of an education-based SQL database named **EDUTECH**. The database contains multiple entities like **USERS**, **TRAINER**, **COURSES**, **STUDENT**, **MODULE**, and **GRADES**, which are used to manage an educational platform's operations.

Entity Relationship Diagram (E.R):

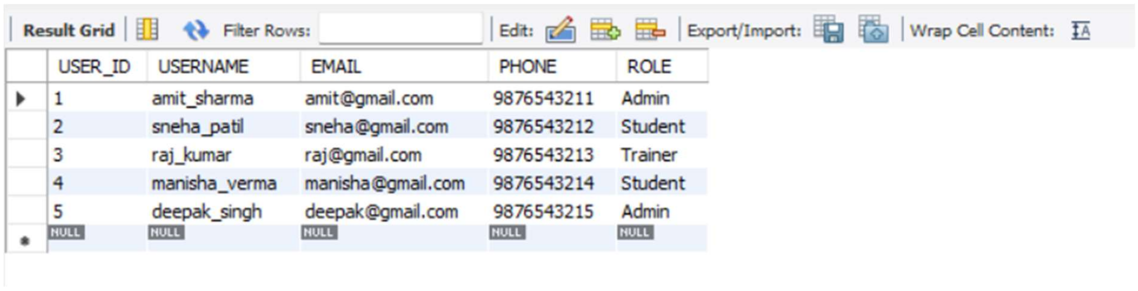




1. USERS Table:

The USERS table stores the basic information of users on the platform, including their username, email, phone number, and role (e.g., Admin, Student, Trainer).

SELECT * FROM USERS;

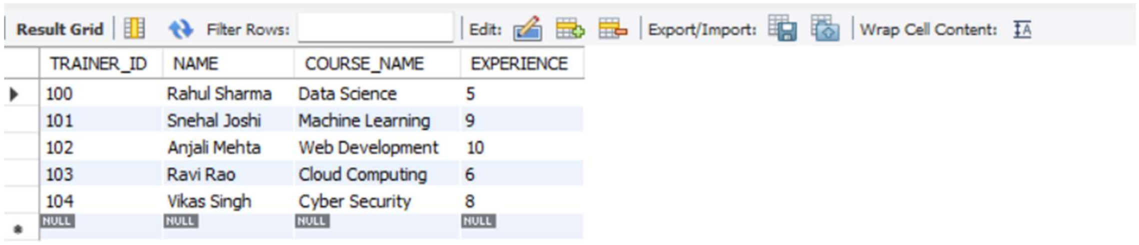


USER_ID	USERNAME	EMAIL	PHONE	ROLE
1	amit_sharma	amit@gmail.com	9876543211	Admin
2	sneha_patil	sneha@gmail.com	9876543212	Student
3	raj_kumar	raj@gmail.com	9876543213	Trainer
4	manisha_verma	manisha@gmail.com	9876543214	Student
5	deepak_singh	deepak@gmail.com	9876543215	Admin
NULL	NULL	NULL	NULL	NULL

2. TRAINER Table:

The TRAINER table holds the trainer's details including their name, the course they teach, and their experience in years.

SELECT * FROM TRAINER;

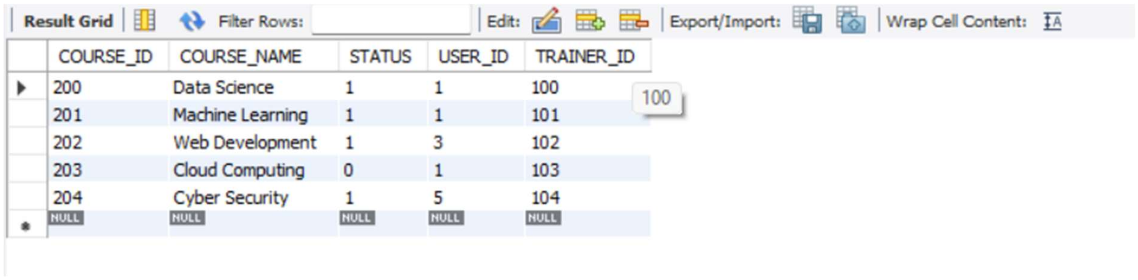


TRAINER_ID	NAME	COURSE_NAME	EXPERIENCE
100	Rahul Sharma	Data Science	5
101	Snehal Joshi	Machine Learning	9
102	Anjali Mehta	Web Development	10
103	Ravi Rao	Cloud Computing	6
104	Vikas Singh	Cyber Security	8
NULL	NULL	NULL	NULL

3. COURSES Table:

The COURSES table stores information about the available courses, including their status (active/inactive), trainer, and related users.

SELECT * FROM COURSES;

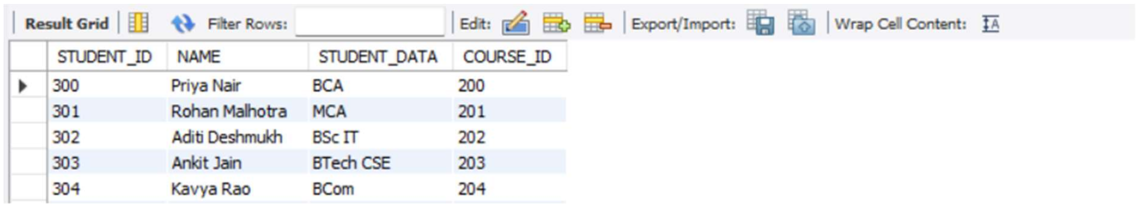


COURSE_ID	COURSE_NAME	STATUS	USER_ID	TRAINER_ID
200	Data Science	1	1	100
201	Machine Learning	1	1	101
202	Web Development	1	3	102
203	Cloud Computing	0	1	103
204	Cyber Security	1	5	104
NULL	NULL	NULL	NULL	NULL

4. STUDENT Table:

The STUDENT table contains student information, such as their name, educational background, and the course they are enrolled in.

SELECT * FROM STUDENT;



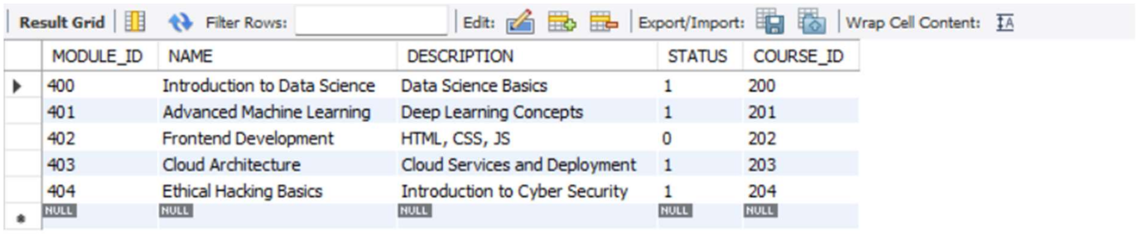
Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

STUDENT_ID	NAME	STUDENT_DATA	COURSE_ID
300	Priya Nair	BCA	200
301	Rohan Malhotra	MCA	201
302	Aditi Deshmukh	BSc IT	202
303	Ankit Jain	BTech CSE	203
304	Kavya Rao	BCom	204

5. MODULE Table:

The MODULE table records individual course modules, including their name, description, and status.

SELECT * FROM MODULE;



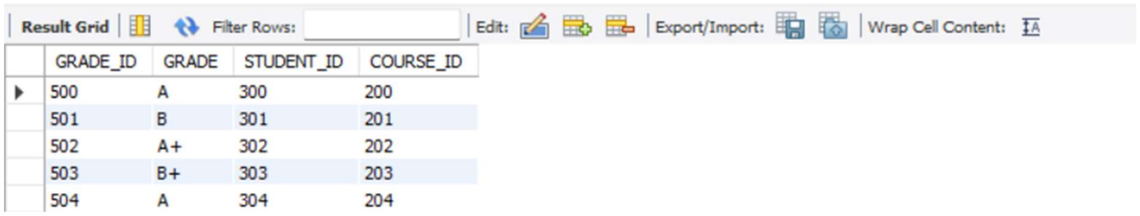
Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

MODULE_ID	NAME	DESCRIPTION	STATUS	COURSE_ID
400	Introduction to Data Science	Data Science Basics	1	200
401	Advanced Machine Learning	Deep Learning Concepts	1	201
402	Frontend Development	HTML, CSS, JS	0	202
403	Cloud Architecture	Cloud Services and Deployment	1	203
404	Ethical Hacking Basics	Introduction to Cyber Security	1	204
NULL	NULL	NULL	NULL	NULL

6. GRADES Table:

The GRADES table stores student grades for each course.

SELECT * FROM GRADES;



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

GRADE_ID	GRADE	STUDENT_ID	COURSE_ID
500	A	300	200
501	B	301	201
502	A+	302	202
503	B+	303	203
504	A	304	204

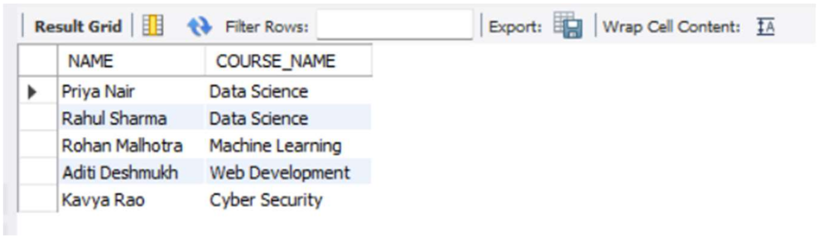
7. **List all students who are enrolled in active courses.**

    SELECT STUDENT.NAME, COURSES.COURSE_NAME

    FROM STUDENT

    JOIN COURSES ON STUDENT.COURSE_ID = COURSES.COURSE_ID

WHERE COURSES.STATUS = TRUE;



NAME	COURSE_NAME
Priya Nair	Data Science
Rahul Sharma	Data Science
Rohan Malhotra	Machine Learning
Aditi Deshmukh	Web Development
Kavya Rao	Cyber Security

8. **Retrieve the trainers who are assigned to more than or equal to one course.**

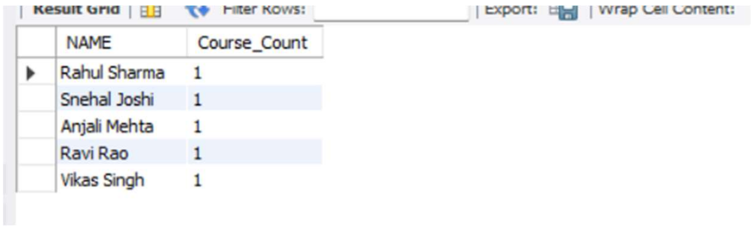
SELECT TRAINER.NAME, COUNT (COURSES.COURSE_ID) AS Course_Count

FROM TRAINER

JOIN COURSES ON TRAINER.TRAINER_ID = COURSES.TRAINER_ID

GROUP BY TRAINER.NAME

HAVING Course_Count >= 1;



NAME	Course_Count
Rahul Sharma	1
Snehal Joshi	1
Anjali Mehta	1
Ravi Rao	1
Vikas Singh	1

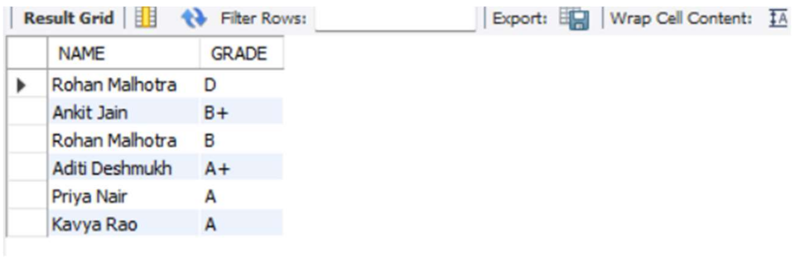
9. **Display all students along with the grade they received, sorted by the grade in descending order.**

SELECT STUDENT.NAME, GRADES.GRADE

FROM STUDENT

JOIN GRADES ON STUDENT.STUDENT_ID = GRADES.STUDENT_ID

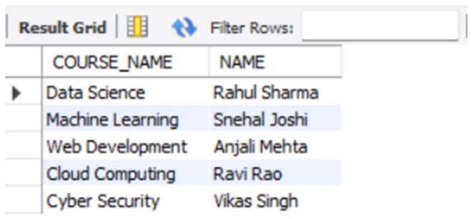
ORDER BY GRADES.GRADE DESC;



NAME	GRADE
Rohan Malhotra	D
Ankit Jain	B+
Rohan Malhotra	B
Aditi Deshmukh	A+
Priya Nair	A
Kavya Rao	A

10. Find the course names along with the trainer names for all courses.

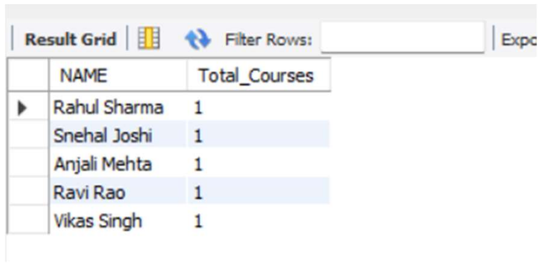
```
SELECT COURSES.COURSE_NAME, TRAINER.NAME
FROM COURSES
JOIN TRAINER ON COURSES.TRAINER_ID = TRAINER.TRAINER_ID;
```



COURSE_NAME	NAME
Data Science	Rahul Sharma
Machine Learning	Snehal Joshi
Web Development	Anjali Mehta
Cloud Computing	Ravi Rao
Cyber Security	Vikas Singh

11. Count the number of courses each trainer is teaching.

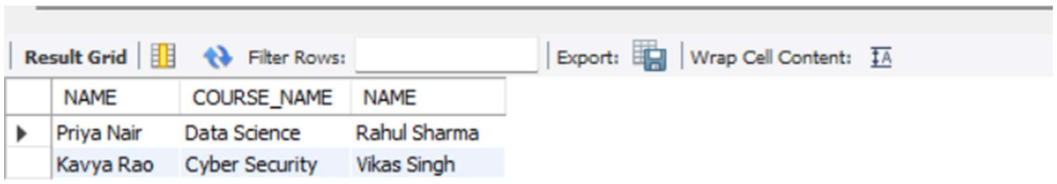
```
SELECT TRAINER.NAME, COUNT(COURSES.COURSE_ID) AS Total_Courses
FROM TRAINER
JOIN COURSES ON TRAINER.TRAINER_ID = COURSES.TRAINER_ID
GROUP BY TRAINER.NAME;
```



NAME	Total_Courses
Rahul Sharma	1
Snehal Joshi	1
Anjali Mehta	1
Ravi Rao	1
Vikas Singh	1

12. Find the course and trainer for students who scored an "A" grade.

```
SELECT STUDENT.NAME, COURSES.COURSE_NAME, TRAINER.NAME
FROM STUDENT
JOIN GRADES ON STUDENT.STUDENT_ID = GRADES.STUDENT_ID
JOIN COURSES ON GRADES.COURSE_ID = COURSES.COURSE_ID
JOIN TRAINER ON COURSES.TRAINER_ID = TRAINER.TRAINER_ID
WHERE GRADES.GRADE = 'A';
```


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

NAME	COURSE_NAME	NAME
Priya Nair	Data Science	Rahul Sharma
Kavya Rao	Cyber Security	Vikas Singh

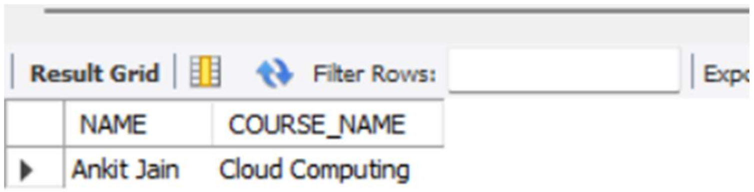
13. List the names of students and the courses that have the status 'FALSE'.

SELECT STUDENT.NAME, COURSES.COURSE_NAME

FROM STUDENT

JOIN COURSES ON STUDENT.COURSE_ID = COURSES.COURSE_ID

WHERE COURSES.STATUS = FALSE;


Result Grid | Filter Rows: | Expo

NAME	COURSE_NAME
Ankit Jain	Cloud Computing

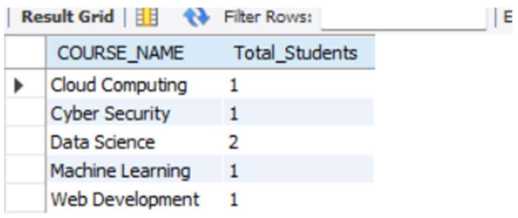
14. Display the total number of students enrolled in each course.

SELECT COURSES.COURSE_NAME, COUNT(STUDENT.STUDENT_ID) AS Total_Students

FROM COURSES

JOIN STUDENT ON COURSES.COURSE_ID = STUDENT.COURSE_ID

GROUP BY COURSES.COURSE_NAME;


Result Grid | Filter Rows: | E

COURSE_NAME	Total_Students
Cloud Computing	1
Cyber Security	1
Data Science	2
Machine Learning	1
Web Development	1

15. Create a view to list all active courses and their trainers.

CREATE VIEW ActiveCoursesView AS

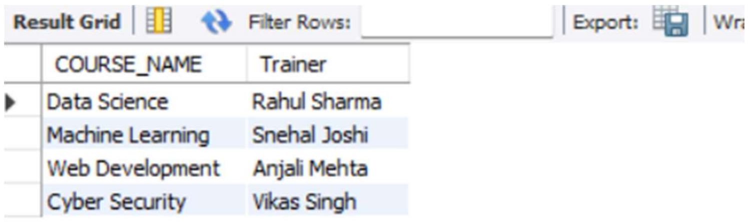
SELECT COURSES.COURSE_NAME, TRAINER.NAME AS Trainer

FROM COURSES

JOIN TRAINER ON COURSES.TRAINER_ID = TRAINER.TRAINER_ID

WHERE COURSES.STATUS = TRUE;

SELECT * FROM ActiveCoursesView; //------- to retrieve the view



COURSE_NAME	Trainer
Data Science	Rahul Sharma
Machine Learning	Snehal Joshi
Web Development	Anjali Mehta
Cyber Security	Vikas Singh

16. List all courses that have more than 2 modules.

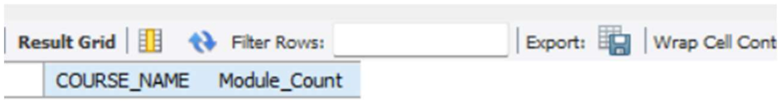
SELECT COURSES.COURSE_NAME, COUNT(MODULE.MODULE_ID) AS Module_Count

FROM COURSES

JOIN MODULE ON COURSES.COURSE_ID = MODULE.COURSE_ID

GROUP BY COURSES.COURSE_NAME

HAVING Module_Count > 2;



COURSE_NAME	Module_Count

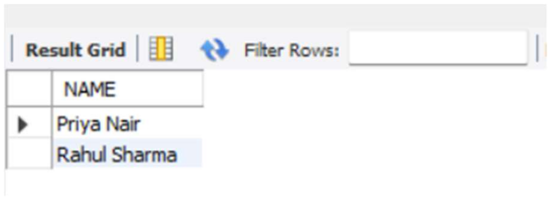
17. Find the student(s) enrolled in the course 'Data Science'.

SELECT STUDENT.NAME

FROM STUDENT

JOIN COURSES ON STUDENT.COURSE_ID = COURSES.COURSE_ID

WHERE COURSES.COURSE_NAME = 'Data Science';



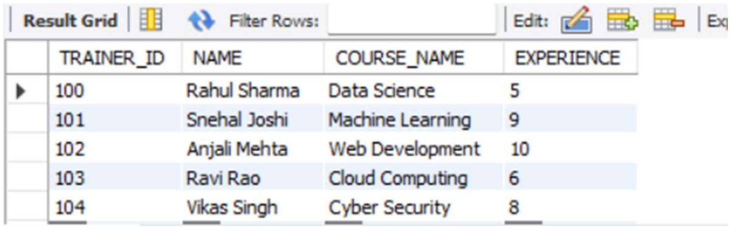
NAME
Priya Nair
Rahul Sharma

18. List all trainers who have not taught any courses.

```
SELECT NAME
FROM TRAINER
WHERE TRAINER_ID NOT IN (SELECT DISTINCT TRAINER_ID FROM COURSES);
```

19. Write a stored procedure to update the experience of a trainer-by-trainer ID.

```
DELIMITER //
CREATE PROCEDURE UpdateTrainerExperience(IN trainerId INT, IN newExperience INT)
BEGIN
  UPDATE TRAINER
  SET EXPERIENCE = newExperience
  WHERE TRAINER_ID = trainerId;
END //
DELIMITER ;

-- Call the procedure
CALL UpdateTrainerExperience(101, 9);
```

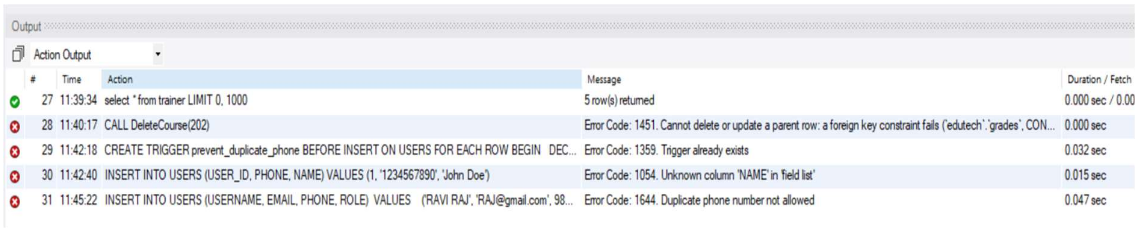


TRAINER_ID	NAME	COURSE_NAME	EXPERIENCE
100	Rahul Sharma	Data Science	5
101	Snehal Joshi	Machine Learning	9
102	Anjali Mehta	Web Development	10
103	Ravi Rao	Cloud Computing	6
104	Vikas Singh	Cyber Security	8

20. Create a trigger that prevents the insertion of duplicate phone numbers in the USERS table.

```
DELIMITER //
CREATE TRIGGER prevent_duplicate_phone
BEFORE INSERT ON USERS
FOR EACH ROW
BEGIN
```

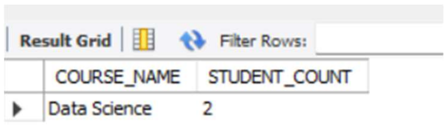
```
    DECLARE phoneExists INT;
    SELECT COUNT(*) INTO phoneExists FROM USERS WHERE PHONE = NEW.PHONE;
    IF phoneExists > 0 THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Duplicate phone number not allowed';
    END IF;
  END//
  DELIMITER ;


  INSERT INTO USERS (USERNAME, EMAIL, PHONE, ROLE)
  VALUES
    ('RAVI RAJ', 'RAJ@gmail.com', 9876543211, 'dean');
```



```
Output
Action Output
#   Time      Action                                                                          Message                                                                                              Duration / Fetch
27  11:39:34  select * from trainer LIMIT 0, 1000                                             5 row(s) returned                                                                                    0.000 sec / 0.00
28  11:40:17  CALL DeleteCourse(202)                                                          Error Code: 1451. Cannot delete or update a parent row: a foreign key constraint fails ('edutech'.'grades', CON...  0.000 sec
29  11:42:18  CREATE TRIGGER prevent_duplicate_phone BEFORE INSERT ON USERS FOR EACH ROW BEGIN  DEC...  Error Code: 1359. Trigger already exists                                                    0.032 sec
30  11:42:40  INSERT INTO USERS (USER_ID, PHONE, NAME) VALUES (1, '1234567890', 'John Doe')   Error Code: 1054. Unknown column 'NAME' in 'field list'                                               0.015 sec
31  11:45:22  INSERT INTO USERS (USERNAME, EMAIL, PHONE, ROLE)  VALUES   ('RAVI RAJ', 'RAJ@gmail.com', 98...  Error Code: 1644. Duplicate phone number not allowed                                  0.047 sec
```

21. Find the course(s) with the most students enrolled.

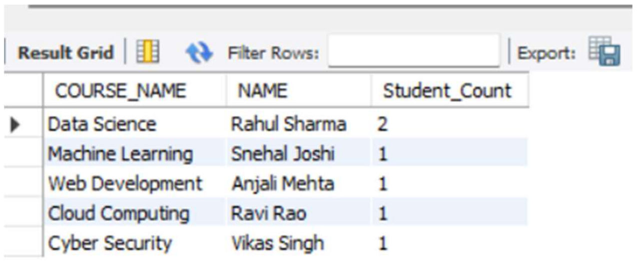
```
SELECT C.COURSE_NAME, COUNT(S.STUDENT_ID) AS STUDENT_COUNT
FROM COURSES C
JOIN STUDENT S ON C.COURSE_ID = S.COURSE_ID
GROUP BY C.COURSE_ID, C.COURSE_NAME
ORDER BY STUDENT_COUNT DESC
LIMIT 1;
```



COURSE_NAME	STUDENT_COUNT
Data Science	2

22. Display the total number of students for each course along with their respective trainers.

```
SELECT COURSES.COURSE_NAME, TRAINER.NAME, COUNT(STUDENT.STUDENT_ID) AS Student_Count

FROM COURSES

JOIN TRAINER ON COURSES.TRAINER_ID = TRAINER.TRAINER_ID

JOIN STUDENT ON COURSES.COURSE_ID = STUDENT.COURSE_ID

GROUP BY COURSES.COURSE_NAME, TRAINER.NAME;
```



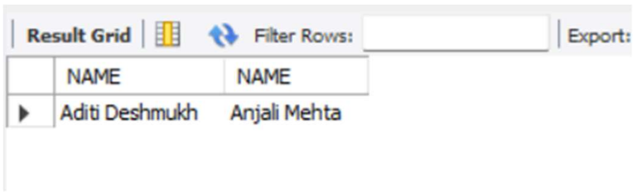
COURSE_NAME	NAME	Student_Count
Data Science	Rahul Sharma	2
Machine Learning	Snehal Joshi	1
Web Development	Anjali Mehta	1
Cloud Computing	Ravi Rao	1
Cyber Security	Vikas Singh	1

23. Retrieve all students who have grades lower than the average grade for their course.

```
SELECT STUDENT.NAME, GRADES.GRADE, COURSES.COURSE_NAME

FROM STUDENT

JOIN GRADES ON STUDENT.STUDENT_ID = GRADES.STUDENT_ID

JOIN COURSES ON GRADES.COURSE_ID = COURSES.COURSE_ID

WHERE GRADES.GRADE < (SELECT AVG(GRADES.GRADE)

                      FROM GRADES

                      WHERE COURSE_ID = COURSES.COURSE_ID);
```

24. Find the students who are enrolled in courses taught by 'Anjali Mehta'.

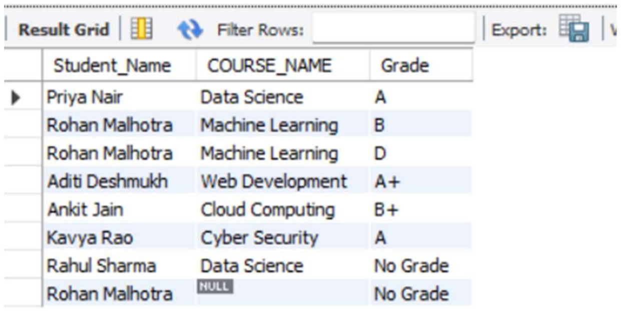
```
SELECT STUDENT.NAME,TRAINER.NAME

FROM STUDENT

JOIN COURSES ON STUDENT.COURSE_ID = COURSES.COURSE_ID

JOIN TRAINER ON COURSES.TRAINER_ID = TRAINER.TRAINER_ID

WHERE TRAINER.NAME = 'Anjali Mehta';
```



NAME	NAME
Aditi Deshmukh	Anjali Mehta

25. Show the names of students along with the name of their courses and their grades, including students without grades.

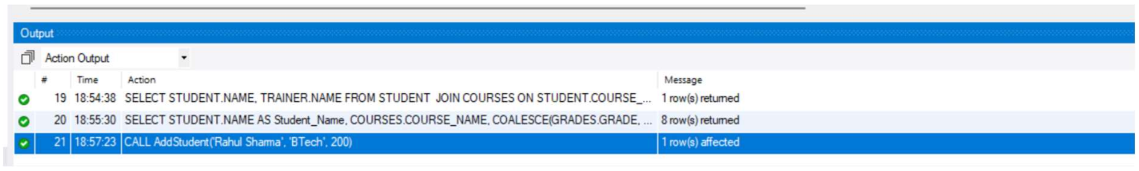
```
SELECT STUDENT.NAME AS Student_Name, COURSES.COURSE_NAME, COALESCE(GRADES.GRADE, 'No Grade') AS Grade

FROM STUDENT

LEFT JOIN COURSES ON STUDENT.COURSE_ID = COURSES.COURSE_ID

LEFT JOIN GRADES ON STUDENT.STUDENT_ID = GRADES.STUDENT_ID;
```



Student_Name	COURSE_NAME	Grade
Priya Nair	Data Science	A
Rohan Malhotra	Machine Learning	B
Rohan Malhotra	Machine Learning	D
Aditi Deshmukh	Web Development	A+
Ankit Jain	Cloud Computing	B+
Kavya Rao	Cyber Security	A
Rahul Sharma	Data Science	No Grade
Rohan Malhotra	NULL	No Grade

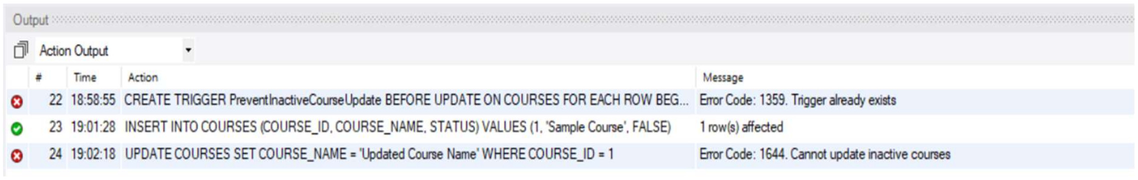
26. Write a stored procedure to add a new student to the database.

```
DELIMITER //

CREATE PROCEDURE AddStudent(IN studentName VARCHAR(30), IN studentData VARCHAR(20), IN courseId INT)

BEGIN

INSERT INTO STUDENT (NAME, STUDENT_DATA, COURSE_ID)

VALUES (studentName, studentData, courseId);

END //

DELIMITER ;

CALL AddStudent('Rahul Sharma', 'BTech', 200);
```


```
Output
Action Output
#   Time      Action                                                                          Message
19  18:54:38  SELECT STUDENT.NAME, TRAINER.NAME FROM STUDENT  JOIN COURSES ON STUDENT.COURSE_...  1 row(s) returned
20  18:55:30  SELECT STUDENT.NAME AS Student_Name, COURSES.COURSE_NAME, COALESCE(GRADES.GRADE, ...  8 row(s) returned
21  18:57:23  CALL AddStudent('Rahul Sharma', 'BTech', 200)                                   1 row(s) affected
```

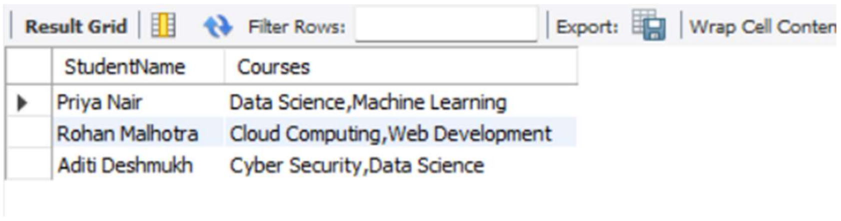
27. Create an Exception handling mechanism to prevent updates on courses that are inactive.

```
DELIMITER //
CREATE TRIGGER PreventInactiveCourseUpdate
BEFORE UPDATE ON COURSES
FOR EACH ROW
BEGIN
  IF NEW.STATUS = FALSE THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Cannot update inactive courses';
  END IF;
END //
DELIMITER ;
INSERT INTO COURSES (COURSE_ID, COURSE_NAME, STATUS)
VALUES (1, 'Sample Course', FALSE);
UPDATE COURSES
SET COURSE_NAME = 'Updated Course Name'
WHERE COURSE_ID = 1;
```


```
Output
Action Output
#   Time      Action                                                                              Message
22  18:58:55  CREATE TRIGGER PreventInactiveCourseUpdate BEFORE UPDATE ON COURSES FOR EACH ROW BEG...  Error Code: 1359. Trigger already exists
23  19:01:28  INSERT INTO COURSES (COURSE_ID, COURSE_NAME, STATUS) VALUES (1, 'Sample Course', FALSE)  1 row(s) affected
24  19:02:18  UPDATE COURSES SET COURSE_NAME = 'Updated Course Name' WHERE COURSE_ID = 1              Error Code: 1644. Cannot update inactive courses
```

28. Write a query to retrieve students who are taking multiple courses, and list the names of those courses.

```
SELECT S.NAME AS StudentName, GROUP_CONCAT(C.COURSE_NAME) AS Courses

FROM STUDENT S

JOIN STUDENT_COURSES SC ON S.STUDENT_ID = SC.STUDENT_ID

JOIN COURSES C ON SC.COURSE_ID = C.COURSE_ID

GROUP BY S.STUDENT_ID

HAVING COUNT (SC.COURSE_ID) > 1;
```

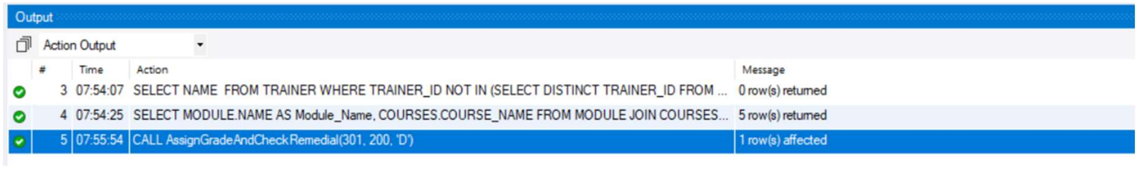


StudentName	Courses
Priya Nair	Data Science,Machine Learning
Rohan Malhotra	Cloud Computing,Web Development
Aditi Deshmukh	Cyber Security,Data Science

29. Create a stored procedure that assigns a grade to a student. If the grade is below 'C', automatically enroll the student in a remedial course.

```
DELIMITER //


CREATE PROCEDURE AssignGradeAndCheckRemedial(

 IN studentId INT,

 IN courseId INT,

 IN grade VARCHAR(10)

)

BEGIN

   -- Assign the grade to the student

   INSERT INTO GRADES (GRADE, STUDENT_ID, COURSE_ID)

   VALUES (grade, studentId, courseId);


   -- Check if grade is below 'C', if so enroll in remedial course

   IF grade IN ('D', 'E', 'F') THEN

      INSERT INTO STUDENT (NAME, STUDENT_DATA, COURSE_ID)
```

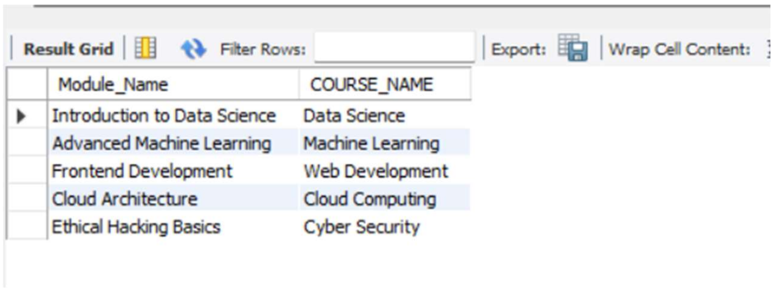
```
    SELECT NAME, STUDENT_DATA, (SELECT COURSE_ID FROM COURSES WHERE COURSE_NAME = 'Remedial Course')
    FROM STUDENT WHERE STUDENT_ID = studentId;
  END IF;
END //

DELIMITER ;

-- Call the procedure
CALL AssignGradeAndCheckRemedial(301, 200, 'D');
```



```
Output
Action Output
#  Time      Action                                                                              Message
3  07:54:07  SELECT NAME  FROM TRAINER WHERE TRAINER_ID NOT IN (SELECT DISTINCT TRAINER_ID FROM ...  0 row(s) returned
4  07:54:25  SELECT MODULE.NAME AS Module_Name, COURSES.COURSE_NAME FROM MODULE JOIN COURSES...     5 row(s) returned
5  07:55:54  CALL AssignGradeAndCheckRemedial(301, 200, 'D')                                         1 row(s) affected
```

30. Get the list of all modules along with the corresponding course name.

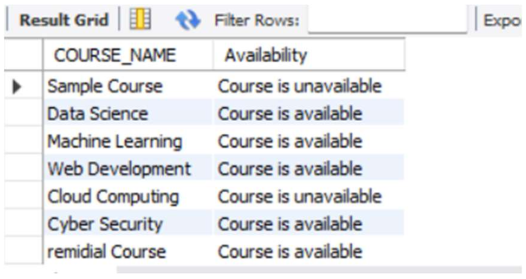
```
SELECT MODULE.NAME AS Module_Name, COURSES.COURSE_NAME
FROM MODULE
JOIN COURSES ON MODULE.COURSE_ID = COURSES.COURSE_ID;
```



Module_Name	COURSE_NAME
Introduction to Data Science	Data Science
Advanced Machine Learning	Machine Learning
Frontend Development	Web Development
Cloud Architecture	Cloud Computing
Ethical Hacking Basics	Cyber Security

31. Write a query to check if a course is active (where STATUS = TRUE). If active, display "Course is available," otherwise display "Course is unavailable."

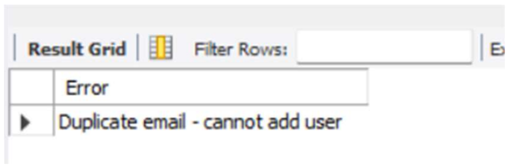
```
SELECT COURSE_NAME,
  CASE
    WHEN STATUS = TRUE THEN 'Course is available'
    ELSE 'Course is unavailable'
  END AS Availability
FROM COURSES;
```


Result Grid | Filter Rows: | Expo

COURSE_NAME	Availability
Sample Course	Course is unavailable
Data Science	Course is available
Machine Learning	Course is available
Web Development	Course is available
Cloud Computing	Course is unavailable
Cyber Security	Course is available
remidial Course	Course is available

32. Write a query that attempts to insert a new user with an existing email. Handle the error by printing "Duplicate email - cannot add user.

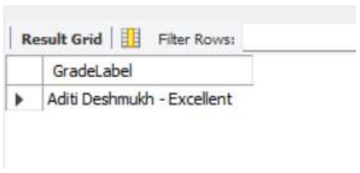
```
DELIMITER //
CREATE PROCEDURE InsertUserWithCheck()
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    SELECT 'Duplicate email - cannot add user' AS Error;
  END;
  INSERT INTO USERS (USERNAME, EMAIL, PHONE, ROLE)
  VALUES ('john_doe', 'amit@gmail.com', 9876543216, 'Student');
 END //
 DELIMITER ;

CALL InsertUserWithCheck();
```


Result Grid | Filter Rows: | E

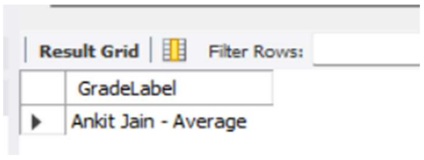
Error
Duplicate email - cannot add user

33. Write a stored procedure that assigns labels to students based on their grade.

```
DELIMITER //

CREATE PROCEDURE CheckStudentGrades()

BEGIN

  DECLARE done INT DEFAULT FALSE;

  DECLARE studentName VARCHAR(30);

  DECLARE grade VARCHAR(10);

  DECLARE cur CURSOR FOR

    SELECT S.NAME, G.GRADE

    FROM STUDENT S
    JOIN GRADES G ON S.STUDENT_ID = G.STUDENT_ID;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;


  OPEN cur;
  loop_grades: LOOP
    FETCH cur INTO studentName, grade;
    IF done THEN
      LEAVE loop_grades;
    END IF;


    IF grade = 'A+' THEN
      SELECT CONCAT(studentName, ' - Excellent') AS GradeLabel;
    ELSEIF grade = 'A' THEN
      SELECT CONCAT(studentName, ' - Good') AS GradeLabel;
    ELSEIF grade = 'B+' THEN
      SELECT CONCAT(studentName, ' - Average') AS GradeLabel;
    ELSE
      SELECT CONCAT(studentName, ' - Needs Improvement') AS GradeLabel;
    END IF;
  END LOOP;
  CLOSE cur;
END //
DELIMITER ;
CALL CheckStudentGrades();
```



GradeLabel
Aditi Deshmukh - Excellent

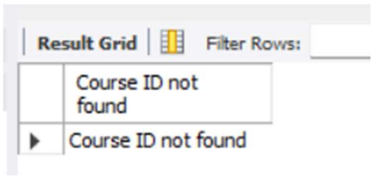


GradeLabel
Ankit Jain - Average

34. Write a procedure to insert a student. If the COURSE_ID doesn’t exist, display "Course ID not found.

```
DELIMITER //
```

```
CREATE PROCEDURE InsertStudentWithCourseCheck()

BEGIN

  DECLARE courseExists INT;

  SET courseExists = (SELECT COUNT(*) FROM COURSES WHERE COURSE_ID = 999);

  IF courseExists = 0 THEN

    SELECT 'Course ID not found';

  ELSE

    INSERT INTO STUDENT (NAME, STUDENT_DATA, COURSE_ID)

    VALUES ('John Doe', 'BCA', 999);

  END IF;

END //

DELIMITER ;

CALL InsertStudentWithCourseCheck();
```

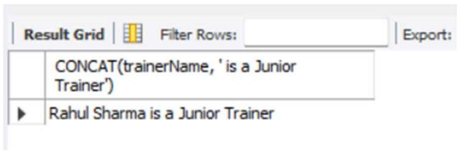


```
Course ID not found
Course ID not found
```

35. **Write a loop that goes through each trainer's experience in the TRAINER table. If experience is greater than 5 years, print "Senior Trainer"; otherwise, print "Junior Trainer.**

```
DELIMITER //
CREATE PROCEDURE CheckTrainerExperience()
BEGIN
  DECLARE done INT DEFAULT FALSE;
  DECLARE trainerName VARCHAR(20);
  DECLARE experience INT;
  DECLARE cur CURSOR FOR SELECT NAME, EXPERIENCE FROM TRAINER;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

  OPEN cur;
  loop_trainers: LOOP
    FETCH cur INTO trainerName, experience;
    IF done THEN
      LEAVE loop_trainers;
    END IF;
    IF experience > 5 THEN
      SELECT CONCAT(trainerName, ' is a Senior Trainer');
    ELSE
```

```
            SELECT CONCAT(trainerName, ' is a Junior Trainer');
        END IF;
    END LOOP;
    CLOSE cur;
END //
DELIMITER ;

CALL CheckTrainerExperience();
```



```
CONCAT(trainerName, ' is a Junior Trainer')
Rahul Sharma is a Junior Trainer
```