

Name: Yun Winner

Class: ITE (M2)

Course: WCT II

SQL queries with explanations:

1. Retrieve all students who enrolled in a specific course.

The screenshot shows a SQL IDE interface. On the left, the 'SCHEMAS' pane displays a tree view with 'wct_homework' selected. The main editor shows the following SQL query:

```
1 SELECT s.student_id, s.first_name, s.last_name, s.email
2 FROM students s
3 JOIN enrollments e ON s.student_id = e.student_id
4 JOIN courses c ON e.course_id = c.course_id
5 WHERE c.course_code = 'CS101';
6
```

Below the query, the 'Result Grid' shows the following data:

student_id	first_name	last_name	email
1	Alice	Johnson	alice@example.com

How it works:

- Joins students, enrollments, and courses tables to find students enrolled in a specific course.
- Filters results based on course_code (replace 'CS101' with any course you want).

2. Find all faculty members in a particular department.

The screenshot shows a SQL IDE interface. On the left, the 'SCHEMAS' pane displays a tree view with 'wct_homework' selected. The main editor shows the following SQL query:

```
1 SELECT f.faculty_id, f.first_name, f.last_name, f.email
2 FROM faculty f
3 JOIN departments d ON f.department_id = d.department_id
4 WHERE d.department_name = 'Computer Science';
5
```

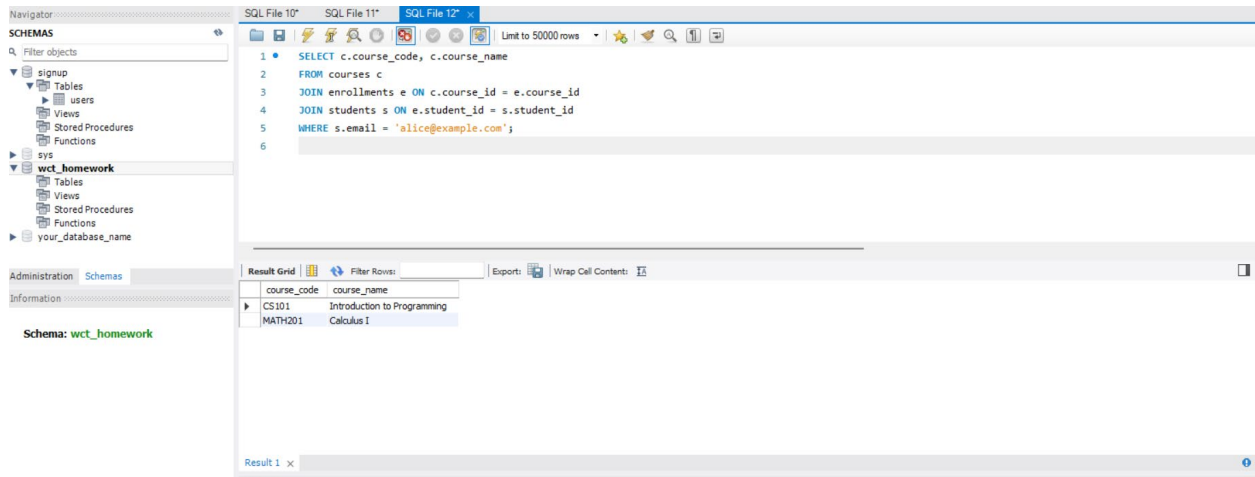
Below the query, the 'Result Grid' shows the following data:

faculty_id	first_name	last_name	email
1	John	Doe	john.doe@example.com

How it works:

- Joins faculty and departments tables to match faculty members with their departments.
- Filters results based on the department name (replace 'Computer Science' with any department).

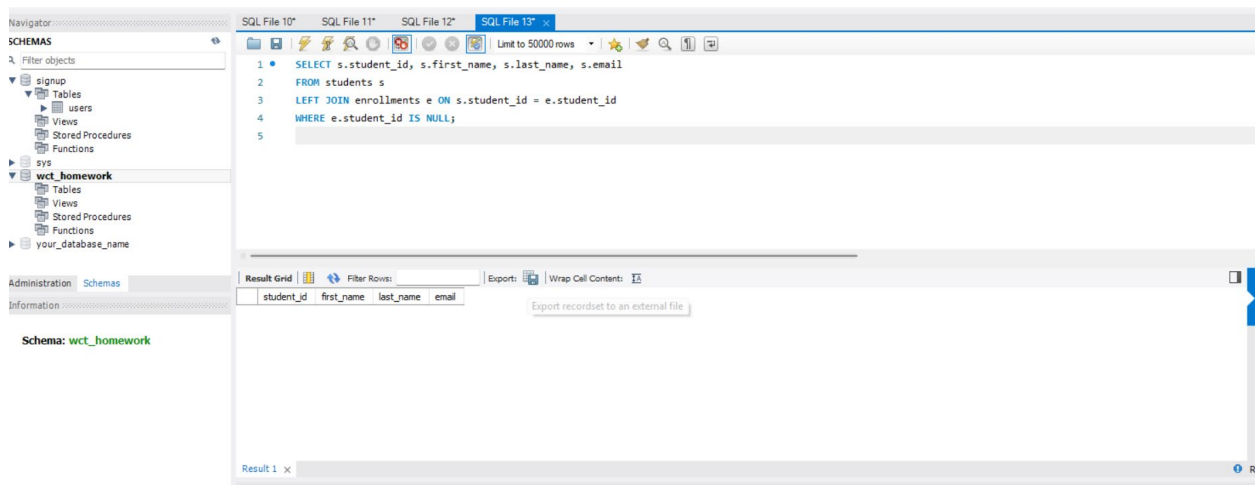
3. List all courses a particular student is enrolled in.



How it works:

- Joins students, enrollments, and courses tables to find courses associated with a student.
- Filters results by the student's email (replace 'alice@example.com' with any student's email).

4. Retrieve students who have not enrolled in any course.



How it works:

- Uses a LEFT JOIN between students and enrollments, which keeps all students in the result.
- Filters out students who have a matching enrollment entry (i.e., they have enrolled in at least one course).
- If e.student_id IS NULL, it means the student has no enrollments.

5. Find the average grade of students in a specific course.

The screenshot shows a SQL IDE interface with a query editor and a results grid. The query editor contains the following SQL code:

```
1 SELECT c.course_code, c.course_name, AVG(CASE
2   WHEN e.grade REGEXP '^[A-F][+-]?[0-9]?$' THEN
3     CASE e.grade
4       WHEN 'A+' THEN 4.3
5       WHEN 'A' THEN 4.0
6       WHEN 'A-' THEN 3.7
7       WHEN 'B+' THEN 3.3
8       WHEN 'B' THEN 3.0
9       WHEN 'B-' THEN 2.7
10      WHEN 'C+' THEN 2.3
11      WHEN 'C' THEN 2.0
12      WHEN 'C-' THEN 1.7
13      WHEN 'D+' THEN 1.3
14      WHEN 'D' THEN 1.0
15      WHEN 'F' THEN 0.0
```

The results grid shows the following data:

course_code	course_name	average_gpa
CS101	Introduction to Programming	4.00000

The output pane shows the following messages:

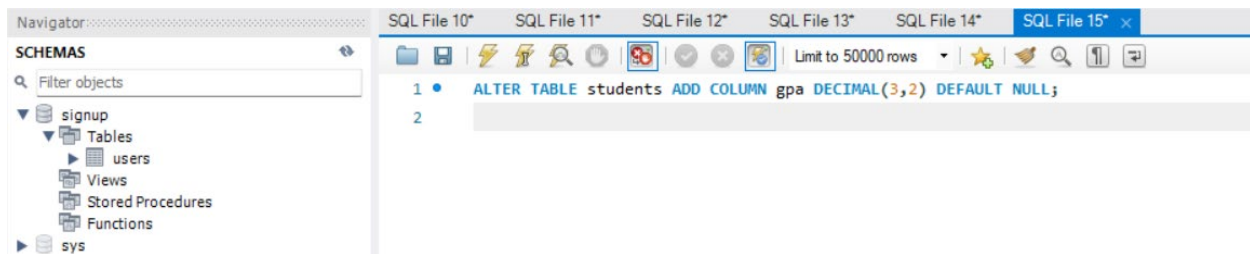
```
41 21:06:57 SELECT s.student_id, s.first_name, s.last_name, s.email FROM students s LEFT JOIN enrollments e ON... 0 row(s) returned
42 21:08:47 SELECT c.course_code, c.course_name, AVG(CASE WHEN e.grade REGEXP '^[A-F][+-]?[0-9]?$' THEN... 1 row(s) returned
```

How it works:

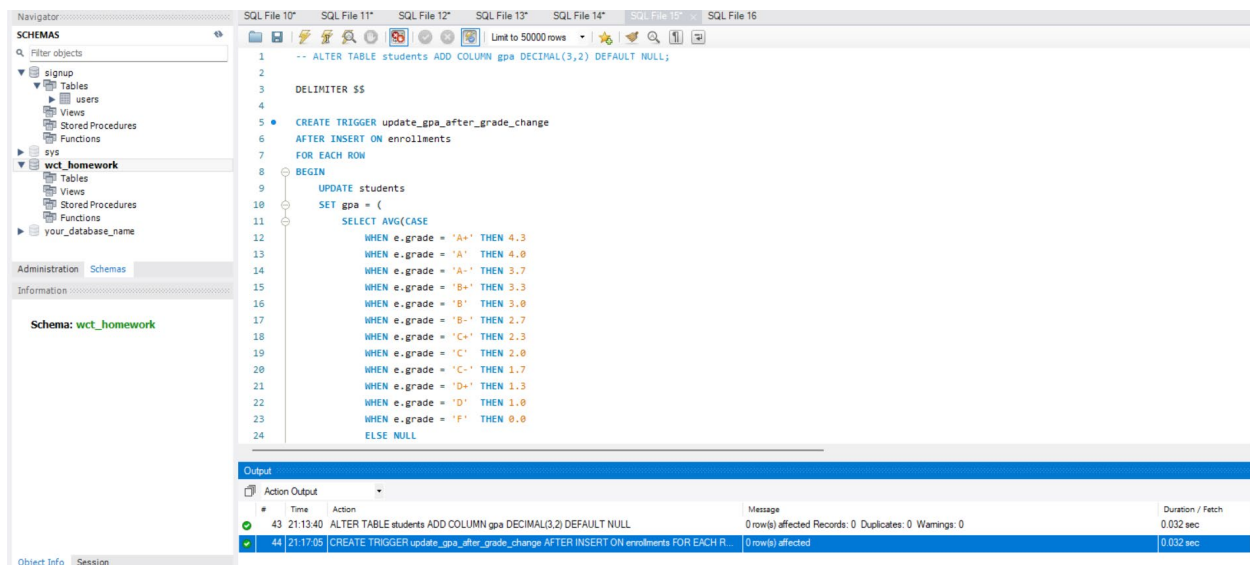
- Joins courses and enrollments to get the grades of students in a specific course.
- Converts letter grades (A, B+, etc.) to a GPA scale (A+ = 4.3, F = 0).
- Computes the average GPA of students in that course.

Bonus:

- Implement a trigger to update a student's GPA when a grade is updated.**
 - Step 1 :** Add a gpa Column to students Table By



b. Step 2 : Create the Trigger



How it works:

- Converts letter grades (A, B+, etc.) into GPA values (4.0 scale).
- Calculates the average GPA for a student based on all their grades.
- Updates a new gpa column in the students table.

II. Design a stored procedure to enroll a student in a course.

The screenshot shows the SQL Server Enterprise Manager interface. On the left, the 'Schemas' pane shows the 'wct_homework' schema. The main pane displays the SQL code for creating a stored procedure named 'enroll_student'. The code is as follows:

```

CREATE PROCEDURE enroll_student(IN p_student_id INT, IN p_course_id INT)
BEGIN
    DECLARE enrollment_exists INT;

    -- Check if the student is already enrolled in the course
    SELECT COUNT(*) INTO enrollment_exists
    FROM enrollments
    WHERE student_id = p_student_id AND course_id = p_course_id;

    -- If student is not enrolled, insert new enrollment
    IF enrollment_exists = 0 THEN
        INSERT INTO enrollments (student_id, course_id, enrollment_date)
        VALUES (p_student_id, p_course_id, CURDATE());
        SELECT CONCAT('Student ', p_student_id, ' successfully enrolled in Course ', p_course_id) AS Message;
    ELSE
        SELECT 'Student is already enrolled in this course!' AS Message;
    END IF;
END $$
DELIMITER ;

```

Below the code, the 'Output' pane shows the execution results. The first row shows the creation of the stored procedure 'enroll_student' at 21:20:38, which took 0.016 seconds to execute. The second row shows the execution of the stored procedure 'enroll_student(1, 2)' at 21:17:05, which took 0.032 seconds to execute and returned the message 'Student is already enrolled in this course!'.

How it works:

- Checks if the student is already enrolled in the course.
- If not enrolled, inserts the record into enrollments.
- Returns a success message or an error message if the student is already enrolled.

➤ Testing the Stored Procedure

The screenshot shows the SQL Server Enterprise Manager interface. The main pane displays the SQL code for calling the stored procedure 'enroll_student(1, 2)'. The code is as follows:

```

CALL enroll_student(1, 2);

```

Below the code, the 'Result Grid' pane shows the execution results. The first row shows the message 'Student is already enrolled in this course!'.

