Mini Project 03

Course Enrollment Explorer (JOIN, UNION, and Subqueries)

CSC-361 Database System Design

Dr. Qixin Deng

Due by: May 3rd, 2026

Overview. In this mini project, you will extend the course demo database application by implementing three important and realistic SQL features:

1. **JOIN** to combine data from multiple related tables
2. **UNION** to merge results from multiple queries
3. **Subqueries** to answer more complex questions using nested SQL

Goal: practice the full workflow

$$\textit{HTML Form} \rightarrow \textit{FastAPI Route} \rightarrow \textit{Database Function} \rightarrow \textit{SQL}.$$

You must finish this project Independently!

Database Setup

This project assumes your database contains at least the following three tables:

- `Student(student_id, name, major, gpa, credits, city)`
- `Course(course_id, title, department, credits)`
- `Enrollment(student_id, course_id, semester, grade)`

You may reuse the `Student` and `Enrollment` tables from class and add the `Course` table if needed.

Part A: Enrollment Viewer (JOIN)

In this part, you must implement an **Enrollment Viewer** that displays enrollment information by combining data from multiple tables using **JOIN**.

Why this part matters

The purpose of this part is not simply to build a search bar. The main goal is to help you understand **why JOIN is necessary in relational databases**.

A single table cannot answer many realistic questions. For example:

- the `Student` table knows the student's name and major

- the `Course` table knows the course title and department
- the `Enrollment` table knows which student takes which course in which semester

If you want to display all of this information together, you must combine these tables correctly. Optional filters are included only to make the joined query more useful and interactive in a web application.

Example scenario

Suppose you want to answer the following question:

> Show all students enrolled in `Computer Science` courses in `Fall 2025`.

This query requires information from all three tables:

- `Student` provides the student name and major
- `Enrollment` provides the semester and grade
- `Course` provides the course title and department

A typical SQL query could be:

```
SELECT s.student_id, s.name, s.major,
c.course_id, c.title, c.department,
e.semester, e.grade
FROM Student s
JOIN Enrollment e ON s.student_id = e.student_id
JOIN Course c ON e.course_id = c.course_id
WHERE c.department = 'Computer Science'
AND e.semester = 'Fall 2025'
```

Web UI Requirements

- Add a form above the result table.
- The form must use HTTP `GET`.
- The form must contain:
  - a text input named `keyword`
  - a dropdown selector named `mode`

The dropdown must support at least the following choices:

Mode	Meaning
Student Name	Filter joined results by student name
Major	Filter joined results by student major
Department	Filter joined results by course department
Semester	Filter joined results by semester

If the keyword is empty, show all joined enrollment records.

JOIN Requirements

Your result table must display information such as:

- Student ID
- Student Name
- Major
- Course ID
- Course Title
- Department
- Semester
- Grade

This information must be produced using SQL **JOIN**, not Python post-processing.

Part B: Special Student List (UNION)

In this part, you must implement a feature that combines the results of two different SQL queries using **UNION**.

Why this part matters

This part helps you understand that sometimes one query result is not enough. In SQL, it is common to define two meaningful groups separately and then merge them into one final result.

This is different from simply filtering one table with a single condition. You will learn how SQL can combine complete result sets and automatically remove duplicates.

Example scenario

Suppose you want to create a scholarship candidate list containing students who satisfy **either** of the following conditions:

- GPA is at least `3.70`
- completed credits are at least `90`

A student who satisfies both conditions should appear only once in the final output.

A typical SQL query could be:

```
SELECT student_id, name, major, gpa, credits
FROM Student
WHERE gpa >= 3.70
UNION
SELECT student_id, name, major, gpa, credits
FROM Student
WHERE credits >= 90
```

Web UI Requirements

- Add a second section, form, or button for this feature.
- This section must display a result table containing:
  - `student_id`
  - `name`
  - `major`
  - `gpa`
  - `credits`

You may either hard-code the thresholds or allow the user to enter them.

Recommended thresholds:

- GPA threshold: `3.70`
- Credits threshold: `90`

UNION Requirements

Create a special student list that includes students satisfying **either** of the following conditions:

1. students with high GPA
2. students with high completed credits

A student who satisfies both conditions should appear only once in the final result.

You must use SQL `UNION`. Do **not** use Python to run two queries and manually merge the results.

You may also sort the final result:

```
ORDER BY name
```

Part C: Reports with Subqueries

In this part, you must implement at least **two reports** that require **subqueries**.

Why this part matters

Subqueries are important because some SQL questions depend on the result of another query.

Instead of reading all rows into Python and computing the answer manually, you will let SQL solve the problem using nested queries. This is a key idea in relational thinking: one query can produce a value or a set of values, and another query can use that result directly.

Example scenarios

Below are several strong examples of subqueries.

Example 1: Students Above Average GPA

Display students whose GPA is greater than the average GPA of all students.

A typical SQL query could be:

```
SELECT student_id, name, major, gpa
FROM Student
WHERE gpa > (SELECT AVG(gpa) FROM Student)
```

This is a good example because the outer query depends on a value computed by the inner query.

Example 2: Students Not Enrolled in Any Course

Display students who are not enrolled in any course.

A typical SQL query could be:

```
SELECT student_id, name, major
FROM Student
WHERE student_id NOT IN (SELECT student_id FROM Enrollment)
```

This is a classic example because the inner query finds students who appear in `Enrollment`, and the outer query finds students who do not.

Example 3: Students Enrolled in at Least One Mathematics Course

Display students who are enrolled in at least one course from the Mathematics department.

A typical SQL query could be:

```
SELECT student_id, name, major
FROM Student
WHERE student_id IN (
```

```
SELECT e.student_id
FROM Enrollment e
JOIN Course c ON e.course_id = c.course_id
WHERE c.department = 'Mathematics')
```

This example is especially valuable because it combines a subquery with a JOIN inside the subquery.

Web UI Requirements

- Add a "Reports" section to the webpage.
- The section may use buttons, a dropdown, or multiple links.
- You must support at least **two** subquery-based reports.
- Each report should display its results in a separate table or panel.

Subquery Requirements

You may choose any two of the example reports above, or implement all of them.

- At least two reports are required.
- The main logic must be implemented in SQL.
- Do not compute the answer in Python after reading all rows.

Suggested Page Layout

You may organize the webpage into three sections:

1. **Enrollment Viewer** for JOIN queries
2. **Special Student List** for UNION queries
3. **Reports** for subquery-based queries

This will make it easier to demonstrate all three SQL ideas in a single project.

Submission Requirements

Submit a single `.zip` file containing the following items:

1. **Source Code** (your full project folder).
2. **Report (PDF)** explaining all modifications you made.
   - Explain what files you modified.
   - Explain how you implemented JOIN.
   - Explain how you implemented UNION.
   - Explain how you implemented subqueries.
   - Provide sufficient explanation to demonstrate your design idea.
   - Submitting code means you fully understand what you submitted.
3. **Short Demo Video** (screen recording).
   - Record the entire screen.
   - Show the project open in your IDE.
   - Manually run the project.
   - Show at least **two JOIN examples**.
   - Show at least **one UNION example**.
   - Show at least **two subquery examples**.