Education Platform: Documentation

I chose to build the **Education Management System** database because it represents a platform **I am very familiar with**. I used to study at a traditional university in Thailand and I had seen firsthand how these systems work every day from viewing schedules to registering for courses. Even though the registration process is a bit different at the Yrkeshögskola in Sweden, I found that familiarity incredibly helpful for visualizing **how the data needs to be linked together**. Since you have also used this platform as a teaching method in the class example many times, it made the initial design process much easier to follow and allowed me to focus less on *what* the system does, and more on *how to make the database work well*.

The database represents an **Education Management System** for a university. Its primary function is to manage and track **Students**, **Teachers**, **Courses**, **Enrollments**, and **Scheduling** of those courses.

- **User Roles:**

    1. **Student (Student):** Registers for courses and views their personal schedule.
    2. **Teacher (Lärare):** Teaches courses and potentially records grades (optional extension).
    3. **Administrator/Staff (Personal):** Manages all records, creates courses, assigns teachers, and sets up the schedule.

- **Type of Information Handled:** Student personal data, course details (name, credits), teacher assignments, enrollment records (which student is in which course), and time/location scheduling details.

The process when I start to work on SQL files

I start 5 table in schema.sql, which are divide 3 kinds of table (parent table ,child table and associate table)

1. Parent tables are TABLE Teachers and TABLE Students.

2.Child tables are TABLE Courses and TABLE Schedules.

3. The associate table is TABLE Enrollments for connecting between Students and Courses.

Overview of the Database

Here is the initial design mapping Ir example objects to database tables:

Table	Represent	Primary Key (PK)	Foreign Keys (FKs)
Students	Individuals taking the courses.	Student_ID	-
Teachers	Who instructed the courses.	Teacher_ID	-
Courses	The subjects offered	Course_ID	TeacherID (to **Teachers**)
Enrollments	The link shows which student is registered for which course.	Enrollments_ID	StudentID (to **Students**), CourseCode (to **Courses**)
Schedules	The specific time and place for a course occurrence.	Schedules_ID	CourseCode (to **Courses**)

Relationships and Motivation

I separate the data into different tables to achieve **data integrity** and **minimize redundancy**

A: One-to-Many (1:M) Relationships

1:M relationship exists when one record in the parent table can be linked to multiple records in the child table. The link is established by placing the **Primary Key (PK)** of the 'One' side into the 'Many' side as a **Foreign Key (FK)**.

Relationship	Parent Table (1-Side)	Child Table (M-Side)	Description
Teacher to Course	**Teachers** (TeacherID: PK)	**Courses** (TeacherID: FK)	One teacher can teach multiple courses, but each course is assigned to only one primary teacher. Separating them avoids

			repeating the teacher's name and contact info in every course record.
Course to Schedule	**Courses** (CourseCode : PK)	**Schedules** (CourseCode :FK)	One course (e.g., "Database Design 101") can be scheduled multiple times (e.g., Monday 9 AM, Wednesday 1 PM) in the **Schedules** table. This prevents repeating course details for every scheduled session.

B: Many-to-Many (M:M) Relationships

A M:M relationship exists when one record in the first table can be linked to multiple records in the second table, and vice versa. M:M relationships **cannot** be implemented directly and require a third table—an **Associative Table** (or **Linking Table**).

Relationship	**Tables Linked**	**Associative Table (M:M Resolver)**	**Description**
Student to **Course**	**Students** and **Courses**	**Enrollments** (StudentID : FK, CourseCode : FK)	One student can enroll in multiple courses, and one course can be taken by multiple students. The **Enrollments** table tracks *when* the student registered and facilitates storing grade data for that specific pairing.

Motivation for Table Separation

The primary motivation for this separation is avoiding data duplication. Essentially, I broke the data into pieces mainly because **I do not want to waste time with unnecessary extra work**, it's all about making the data clean and manageable.

For example, without separate tables:

- **Redundancy (Wasted Space):** If Professor Ugglas teaches 10 courses, his name, address, and contact information would be repeated 10 times in a single Course table, wasting space.
- **The Update Disaster (Inconsistency):** If Professor Ugglas moves or changes his phone number, **I** would have to update 10 rows. If **I** miss one, the data becomes inconsistent, and the system starts telling a lie about his contact info.
- **Startup/Deletion Issues:** I couldn't record a new Teacher until they are assigned a course (an insertion anomaly), or **I** might accidentally lose a Course's time slot if the teacher quits (a deletion anomaly).

By using Foreign Keys, I ensure data is recorded only once (e.g., in the Teachers table) and simply referenced everywhere else.

Normalization (3NF) Justification

I designed my database to meet the Third Normal Form (3NF) rules. The main goal of 3NF is simple: I want to guarantee the data is always reliable, accurate, and consistent. It prevents me from having to do a ton of messy cleanup work later.

Here's how my design satisfies 3NF:

I. Eliminated Redundancy by Separating Data:

- The Problem: If I kept the Teacher's phone number in the Courses table, that contact information would be repeated every time that teacher taught a new course.
- My Solution: I created a separate Teachers table. I now store the teacher's details only once. The Courses table just uses the TeacherID (As Foreign Key) to point to the correct teacher record. This prevents data from being duplicated.

II. Ensured All Columns Depend Only on the Primary Key:

- The main rule of 3NF is simple: every column must depend directly and only on the Primary Key (PK) of its table.
- Example: I made sure that a non-key column (like Teacher's Phone) is not accidentally dependent on another non-key column (like Teacher's Name) within the same table.
- My Tables: All columns in the Courses table (like CourseName and Credits) rely only on the CourseCode (the PK). They do not depend on the TeacherID (the FK), which keeps the relationship clean and follows the 3NF rule.

Choice of Data Types

Appropriate data types are chosen to ensure data integrity, optimize storage, and enforce constraints.

Column Example	Data Type	Motivation
StudentID, TeacherID, CourseCode	**INT** or **VARCHAR(10)**	Use **INT** for simple numerical keys, or **VARCHAR** if the IDs contain letters (e.g., 'CS101'). PKs must be fast to search.
StudentName, CourseName	**VARCHAR(100)**	**VARCHAR** is used for text that varies in length (efficient storage). (100) is sufficient to accommodate full names and course titles.
EnrollmentDate	**DATE**	Used to store only the date (YYYY-MM-DD) when the student registered, optimizing storage over TIMESTAMP when time is not needed.
StartTime in Schedules	**TIMESTAMP**	Used to accurately record both the **date and time** for a specific class meeting.

Security Aspects (Data Integrity)

Data integrity is the system's ability to maintain data accuracy and consistency.

- **Primary Keys (PK):** Guarantees **Entity Integrity** (every record is uniquely identifiable).
- **Foreign Keys (FK):** Guarantees **Referential Integrity**. This prevents assigning a course to a TeacherID that doesn't exist in the Teachers table.
- **NOT NULL Constraints:** Applied to essential columns like Name and Email to ensure that critical data is never missing.
- **UNIQUE Constraints:** Applied to StudentEmail (not shown but recommended) to prevent two students from having the same email address.

And I also applied this constraint to all Foreign Keys (like StudentID and CourseCode in the Enrollments table) to ensure every record linking a student to a course is complete and valid.

Indexing (Performance Optimization)

To ensure the database can handle quick lookups, I have chosen to index the **LastName** column in the Students table.

Motivation: This column is indexed because administrators or teachers will frequently search for students based on their last name. An index on LastName dramatically reduces the search time in large tables, which makes the system more responsive during reporting and lookups.

Views

I created two views to simplify query writing, enhance security, and facilitate report generation, which will be included in the schema.sql file.

1. **Report View (CourseEnrollmentCount):** I created this view to quickly see how popular each course is without having to write a complex GROUP BY and COUNT query every time. It simulates a "top list" or report on course enrollment levels.
2. **Simplified View (StudentCourseDetails):** This view simplifies day-to-day queries by combining the student's name, the course name, and the grade from three different tables (Students, Enrollments, Courses). This helps administrators search for a student's full details quickly without writing a three-table JOIN.

Future Problems: What My Database *Can't* Handle Yet

If the university grows, my current, simple schema would immediately run into some limitations because the design does not have all the functions to make everything work in real use yet.

🛠️ Summary of Problems and Solutions

1. Database and Table Creation Errors

Problem Faced	Error Code / Message	Cause of Problem	My Solution (The Fix)
Database Existed	Error Code: 1007. Can't create database... database exists	I was attempting to create the database (CREATE DATABASE) when it had already been created earlier in the session.	I confirmed the database already existed and instructed you to proceed by using the command **USE SchoolProgram_project** to select it.
Table Existed	Error Code: 1050. Table 'teachers' already exists	I attempted to run a CREATE TABLE command for a table that was already defined.	I confirmed the table structure was correct and instructed you to skip the redundant CREATE TABLE commands.
Missing Table	Error Code: 1146. Table 'schoolprogram_project.enrollments' doesn't exist	During cleanup (TRUNCATE TABLE), I referenced a table (Enrollments) before it was successfully created or after it was accidentally dropped.	I confirmed the table creation order and ensured all tables were present before moving to the data insertion phase.

2. Data Insertion and Setup Errors

Problem Faced	Error Code / Message	Cause of Problem	My Solution (The Fix)

Foreign Key Constraint	Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails	I tried to insert data into a child table (like Courses) referencing a parent table (Teachers) before the parent data was inserted.	I corrected the execution order by making sure the data for **Teachers, Students, and Courses** was inserted **before** the data for Schedules and **Enrollments**.

3. Reporting and Query Errors

Problem Faced	Error Code / Message	Cause of Problem	My Solution (The Fix)
Query Returns Nothing	(No error code, but no result table appeared)	The SQL client was likely confused and trying to run the whole script or just comments instead of the specific query.	I instructed you to **highlight the specific SELECT statement** from SELECT down to the semicolon ; and use the "Execute Selection" command.

Conclusion and Reflection

Overall, this project was a **huge step forward for me**, having only studied databases for a month. I feel pretty good about the result, even though, of course, I wish I could make it better. I started with the clear goal of building a fully functional Education Management System and achieved it by creating five correctly linked tables and populating them with real data. I learned a huge amount about troubleshooting and debugging through this process, and even though I don't feel like I understand the whole thing yet, I definitely want to learn and improve more about databases, as I think it's a very essential skill for an IT career.