Database Schema

Below is the complete SQL schema used for the application. Comments have been added to explain the purpose of each table and significant columns, particularly regarding the "Soft Delete" functionality (status columns) and the complex many-to-many relationships. More information about the Soft Delete functionality as mentioned below.

1. Degree Table

- Represents a **degree program** at the university (e.g., “Computer Science – MS”, “Data Science – BS”). Each degree is uniquely identified by a combination of its name and level.

```
CREATE TABLE Degree (
    degree_name VARCHAR(50),
    degree_level VARCHAR(10),      -- e.g., 'BS', 'MS', 'Cert'
    description TEXT,
    status VARCHAR(20) DEFAULT 'Active', -- 'Active' or 'Inactive' for Soft Delete
    PRIMARY KEY (degree_name, degree_level)
);
```

2. Course Table

- Stores the **catalog of courses** offered by the department (e.g., “CS 7330 – Databases”). Each course has a unique course code.

```
CREATE TABLE Course (
    course_code VARCHAR(10),       -- Unique ID, e.g., 'CS5330'
    course_name VARCHAR(100),
    status VARCHAR(20) DEFAULT 'Active', -- Preserves history even if course is discontinued
    PRIMARY KEY (course_code)
);
```

3. Instructor Table

- Stores basic information about **faculty/instructors** who teach course sections that are evaluated.

```
CREATE TABLE Instructor (
    instructor_id VARCHAR(10),     -- Unique ID, e.g., 'I-101'
    first_name VARCHAR(50),
    middle_name VARCHAR(50),
    last_name VARCHAR(50),
```

```
    email_id VARCHAR(100),
    phone_number VARCHAR(20),
    status VARCHAR(20) DEFAULT 'Active', -- Allows archiving instructors who leave without losing history
    PRIMARY KEY (instructor_id)
);
```

4. Section Table

- Represents a specific offering of a course in a specific semester.
- A course can have multiple sections (e.g., 001, 002) in the same term.

```
CREATE TABLE Section (
    course_code VARCHAR(10),
    section_num INT,              -- Stored as INT (e.g., 1), displayed as '001' in app
    semester VARCHAR(10),         -- 'Fall', 'Spring', 'Summer'
    year_offered INT,
    num_enrollments INT,          -- Total students enrolled (Validation: Must be > 0)
    PRIMARY KEY (course_code, section_num, semester, year_offered),
    FOREIGN KEY (course_code) REFERENCES Course(course_code)
);
```

5. Teaches Table (Relationship)

- Links an Instructor to a specific Section.

```
CREATE TABLE Teaches (
    course_code VARCHAR(10),
    section_num INT,
    semester VARCHAR(10),
    year_offered INT,
```

```
    instructor_id VARCHAR(10),
    PRIMARY KEY (course_code, section_num, semester, year_offered, instructor_id),
    FOREIGN KEY (course_code, section_num, semester, year_offered)
        REFERENCES Section(course_code, section_num, semester, year_offered) ON DELETE CASCADE,
    FOREIGN KEY (instructor_id) REFERENCES Instructor(instructor_id)
);
```

6. Objective Table

- Stores the learning goals (e.g., "SQL Mastery").

```
CREATE TABLE Objective (
    obj_code VARCHAR(20),          -- Unique code, e.g., 'OBJ-SQL'
    title VARCHAR(120),
    description TEXT,
    PRIMARY KEY (obj_code)
);
```

7. Degree_Course Table (Mapping)

- Maps courses to degrees and defines if they are Core or Elective.

```
CREATE TABLE Degree_Course (
    degree_name VARCHAR(50),
    degree_level VARCHAR(10),
    course_code VARCHAR(10),
    is_core BOOLEAN,               -- TRUE = Core, FALSE = Elective
    PRIMARY KEY (degree_name, degree_level, course_code),
    FOREIGN KEY (degree_name, degree_level) REFERENCES Degree(degree_name, degree_level),
    FOREIGN KEY (course_code) REFERENCES Course(course_code)
);
```

8. Degree_Objective Table (Mapping)

- Defines which **learning objectives** are associated with each **degree program**. Different degrees can share or have distinct objectives.

```
CREATE TABLE Degree_Objective (
    degree_name VARCHAR(50),
    degree_level VARCHAR(10),
    obj_code VARCHAR(20),
    PRIMARY KEY (degree_name, degree_level, obj_code),
    FOREIGN KEY (degree_name, degree_level) REFERENCES Degree(degree_name, degree_level),
    FOREIGN KEY (obj_code) REFERENCES Objective(obj_code)
);
```

9. Course_Objective Table (Mapping)

- Represents a **"diamond" relationship** linking:
    - Degree
    - Course
    - Objective
- This allows the same course to satisfy **different objectives for different degrees**.
- Example:
  The course CS 7330 might support a "Database Design" objective for the CS degree, but a different objective for a Data Engineering degree.

```
CREATE TABLE Course_Objective (
    degree_name VARCHAR(50),
    degree_level VARCHAR(10),
    course_code VARCHAR(10),
    obj_code VARCHAR(20),
    PRIMARY KEY (degree_name, degree_level, course_code, obj_code),
    FOREIGN KEY (degree_name, degree_level) REFERENCES Degree(degree_name, degree_level),
    FOREIGN KEY (course_code) REFERENCES Course(course_code),
    FOREIGN KEY (obj_code) REFERENCES Objective(obj_code)
);
```

10. Evaluation Table

- Stores **aggregated evaluation results** for a given:

    - degree + level
    - course section
    - learning objective
- For that combination, we track how many students earned grades A, B, C, and F, along with optional notes on improvement.

```
CREATE TABLE Evaluation (
    course_code VARCHAR(10),
    section_num INT,
    semester VARCHAR(10),
    year_offered INT,
    degree_name VARCHAR(50),        -- Context: Evaluations are tied to the Degree
    degree_level VARCHAR(10),
    obj_code VARCHAR(20),
    count_A INT,
    count_B INT,
    count_C INT,
    count_F INT,
    improvement TEXT,               -- Instructor's feedback/notes
    PRIMARY KEY (course_code, section_num, semester, year_offered, degree_name, degree_level, obj_code),
    FOREIGN KEY (degree_name, degree_level, course_code, obj_code)
        REFERENCES Course_Objective(degree_name, degree_level, course_code, obj_code)
);
```

11. Evaluation_Method Table

- Stores the **assessment methods** used to evaluate a particular objective in a given section (e.g., “Quiz”, “Project”, “Final Exam”).
- This is a separate table to allow multiple methods per objective (1-to-Many).

```
CREATE TABLE Evaluation_Method (
    course_code VARCHAR(10),
    section_num INT,
```

```
    semester VARCHAR(10),
    year_offered INT,
    degree_name VARCHAR(50),
    degree_level VARCHAR(10),
    obj_code VARCHAR(20),
    method_name VARCHAR(50),        -- e.g., 'Homework', 'Final Exam', 'Lab Demo'
    PRIMARY KEY (course_code, section_num, semester, year_offered, degree_name, degree_level, obj_code, method_name),
    FOREIGN KEY (course_code, section_num, semester, year_offered, degree_name, degree_level, obj_code)
        REFERENCES Evaluation(course_code, section_num, semester, year_offered, degree_name, degree_level, obj_code) ON DELETE CASCADE
);
```

Feature Highlight: Soft Delete Implementation

1. The Problem with Standard Deletion ("Hard Delete"): In a university context, permanently deleting records creates data integrity issues. For example, if a professor retires and their record is deleted, all historical reports for courses they taught in previous years would break or show missing data.

2. The Solution: "Soft Delete" (Archiving) We implemented a Soft Delete strategy using a status column.

- Mechanism: Major tables (Instructor, Course, Degree) include a status column defaulting to 'Active'.
- Archiving: Clicking "Archive" runs an UPDATE command (setting status to 'Inactive') rather than a DELETE command.
- Benefits:
    - Data Preservation: Historical reports remain accurate.
    - User Interface: Dropdown menus filter out Inactive records for *future* assignments but allow them for *historical* data entry.
    - Restoration: Records can be reactivated if needed.