Khanh Tra Nguyen Tran 24379393

Management Information System for Math, Statistics & Data Science, and Computer Science (MSCS) Department

Khanh Tra Nguyen Tran 24379393

Section A: Description of Database Application and ER Model

1. Application Description:

For this project, I implemented an information management system for the Math, Statistics and Data Science, and Computer Science Department at St. Olaf College, my home university. The database stores and manages student, faculty, course, degree path, major, grade, and course registration information related to the department. The project uses MySQL and follows relational database design principles.

The **Student** table contains information on students who take courses the department offers. Most also declare at least one major in Maths, Computer Science, or Statistics and Data Science. Each tuple in this table consists of a student's unique ID number, first name, last name, class years (the year they graduate), and three boolean values indicating whether the student majors in Mathematics, Statistics & Data Science, or Computer Science (true if they declare the corresponding major and false if they don't). Each student is automatically assigned an ID based on the order of their entry into the database (using *AUTO_INCREMENT*).

The **Faculty** table contains information about faculty members who teach courses related to the three majors mentioned above or work for the department (academic administrative assistant, cluster managers, …) Each tuple consists of a faculty member's ID number, prefix (for faculty members who have obtained a Doctorate), first name, and last name. Faculty members receive an auto-incremented FacultyID upon entry.

The **Major** table defines academic majors managed by the department (Computer Science, Math, Statistics & Data Science), storing the major code, major name, the number of courses required to complete the major, and the number of students currently pursuing it. Each major is directed by a faculty member referenced by their FacultyID.

The **Course** table contains the details of academic courses offered by the department. Each tuple in the database consists of courses' course IDs (unique ID for each course), course names, capacity, number of sections each course provides, and a boolean value indicating whether the course fulfills OLE Core requirements, which are general education requirements for students at my college.

The **CourseRegistration** table logs sections for courses offered in a specific class year and semester (three semesters, fall, spring, or winter, in an academic year). Each entry is uniquely identified by a combination of CourseID, SectionID, ClassYear, and Semester since, in a semester, some courses may have more than one section. It also records the LecturerID (the faculty member teaching the section), referenced by their FacultyID, the number of spots in the section filled, and the number of overridden requests (students' requests to gain permission to enroll when the number of filled spots has reached the maximum capacity or the prerequisite is not satisfied).

The **LetterGrade** table helps to convert the standard letter-to-GPA conversion scale. Each unique letter grade (primary key) will be equivalent to the numerical grade.

Letter Grade	4.0 Grade
A+/A	4.0
A-	3.7
B+	3.3
B	3.0
B-	2.7
C+	2.3
C	2.0
C-	1.7
D+	1.3
D	1.0
F	0.0

The **Grade** table records grades the students receive from the courses they have taken so far. It includes the student ID, the course ID, and the grade they have received from that course.

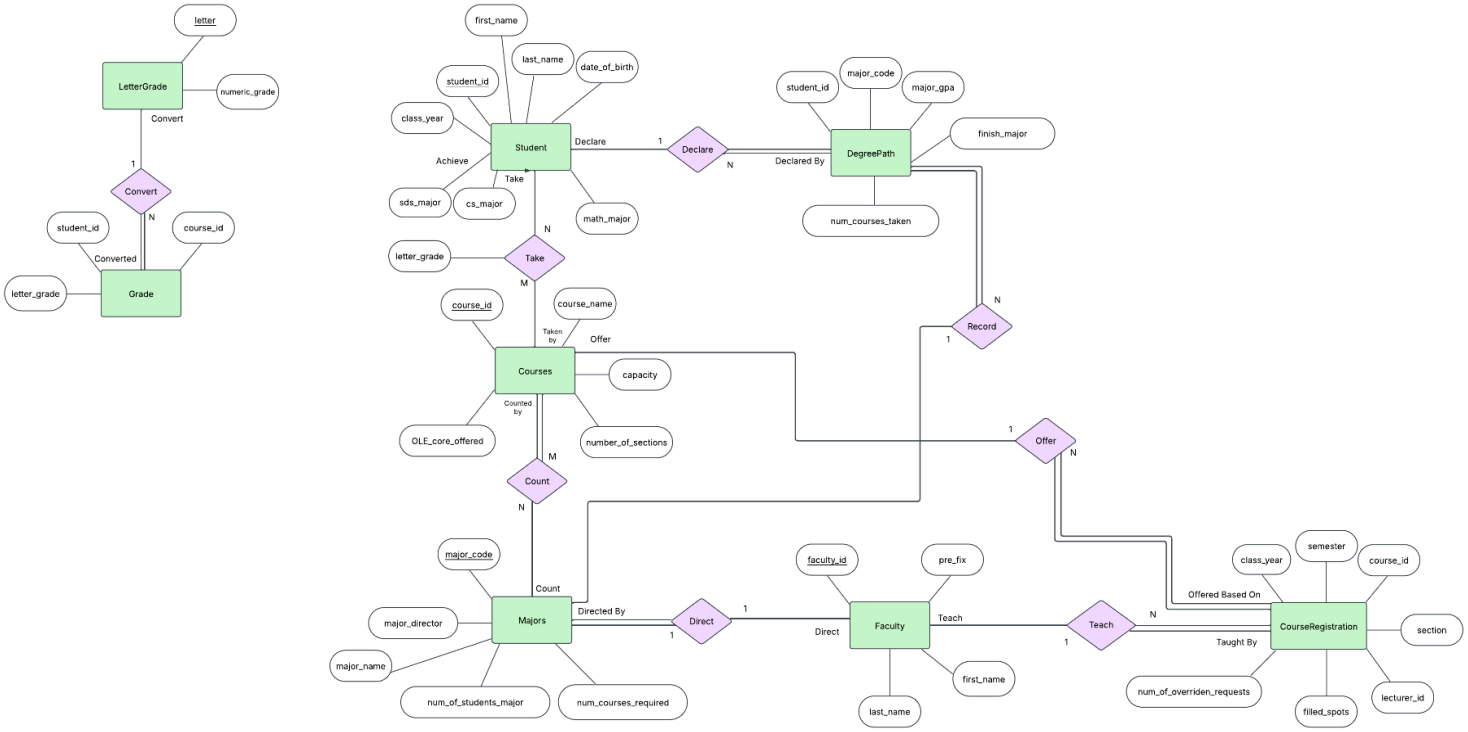
The **CourseCountedToMajor** table shows courses that are counted to fulfill a major. It includes the course ID and the major code.

The **DegreePath** table records each student's progress toward completing a major. It includes the Student ID, MajorCode, the number of courses taken toward the major, the number of required courses to complete the corresponding major, a boolean (**FinishMajor**) indicating whether the major is completed, and the MajorGPA (calculated on a 4.0 scale). The table enforces a foreign key constraint linking MajorCode to the **Major** table.

- GitHub link: https://github.com/FayNguyen03/MSCS-Information-Management.git

2. Entity–relationship model:

- Clear version: https://github.com/FayNguyen03/MSCS-Information-Management/blob/8208fa1dff702e132040ee4290857b396ba234e6/ER%20Model.png (download version)



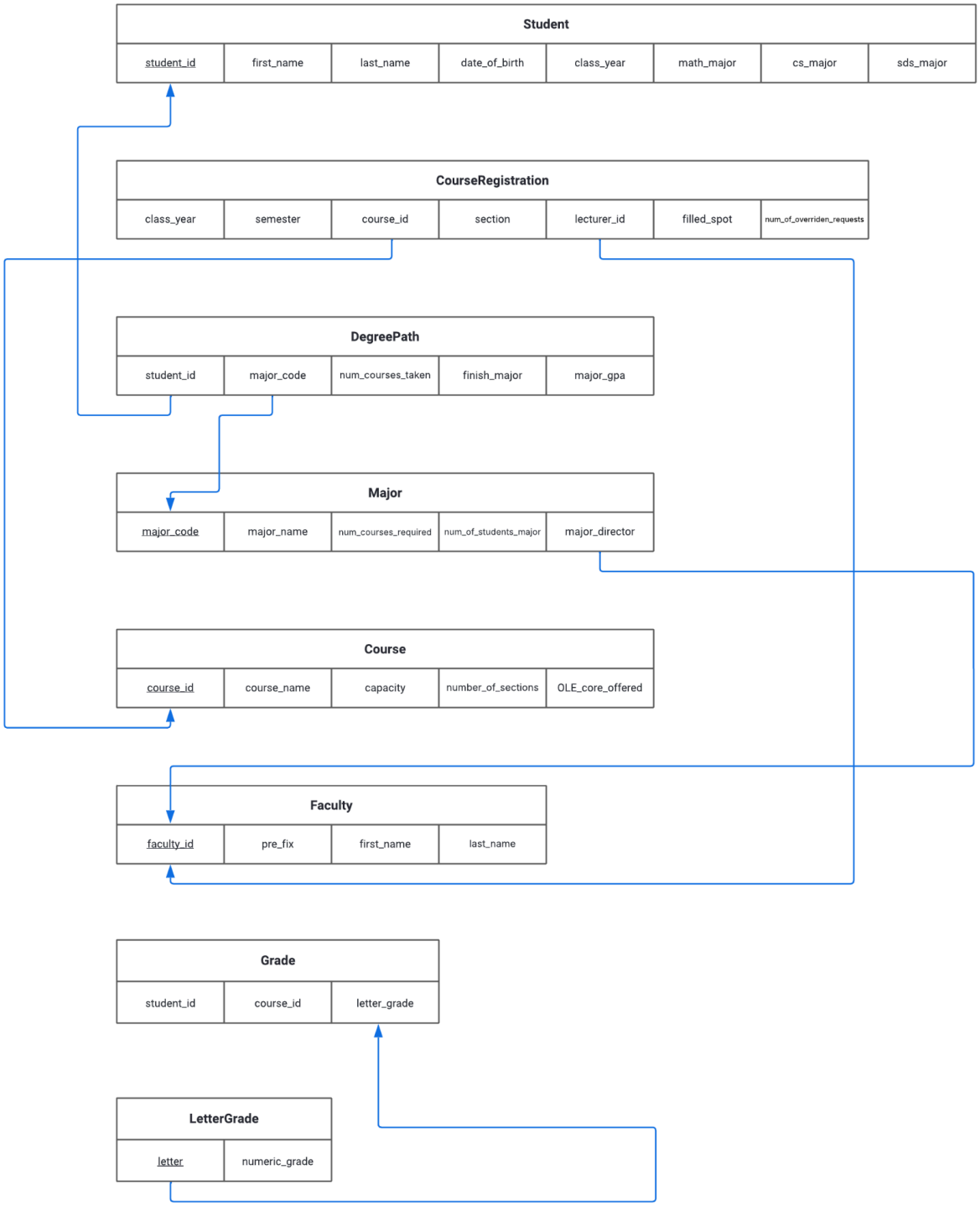
ASSUMPTION:

- Each course section listed in **CourseRegistration** must be taught by one faculty member, while a faculty member may or may not teach any course (researcher, staff, …). However, a faculty member can teach multiple sections per semester.
- Each course section listed in **CourseRegistration** must be offered based on one course listed on the **Course** table, but each course in the **Course** list may have multiple sections offered in a semester or may not be offered in a specific semester.
- Each major must be directed by one faculty member and each assigned member only directs one major. However, not every faculty member directs a major.
- Each major may count many courses that can fulfill the major requirements, and each course can be counted by many majors. Every course must be included in the requirement of at least one major.
- An entry in the **DegreePath** must record a student's progress toward completing one major and each major may have multiple entries in the **DegreePath**.
- A student may take multiple courses, and a course may be taken by many students; for each course a student takes, they receive a letter grade.
- One student can have multiple degree paths (or none if they have not declared any MSCS major), each corresponding to a different major they have declared. Each degree path must belong to one and only one student.

- Each letter grade recorded in the **Grade** table must be converted into a numerical grade by using the **LetterGrade** table, while a letter grade in the **LetterGrade** table may convert some grades from the **Grade** table (some letter grades may not be used)

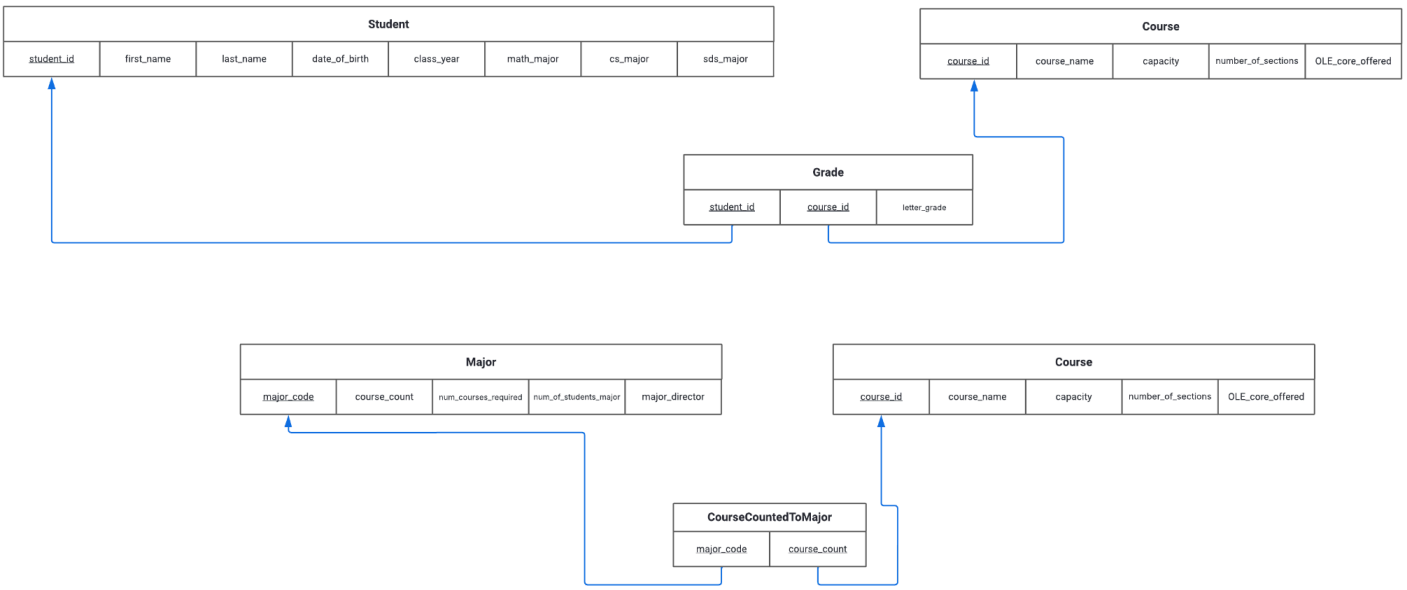
3. Relationship Schema:

(Primary keys are underlined)



- Relationships in the ER Model included in the schema above:
  - 1-to-1 **Direct** relationship between **Faculty** and **Major** (N-side)

- 1-to-many **Teach** relationship between **Faculty** and **CourseRegistration** (N-side)
- 1-to-many **Offer** relationship between **Course** and **CourseRegistration** (N-side)
- 1-to-many **Declare** relationship between **Student** and **DegreePath** (N-side)
- 1-to-many **Convert** relationship between **LetterGrade** and **Grade** (N-side) (relation created from the man
- 1-to-many **Record** relationship between **Major** and **DegreePath** (N-side)

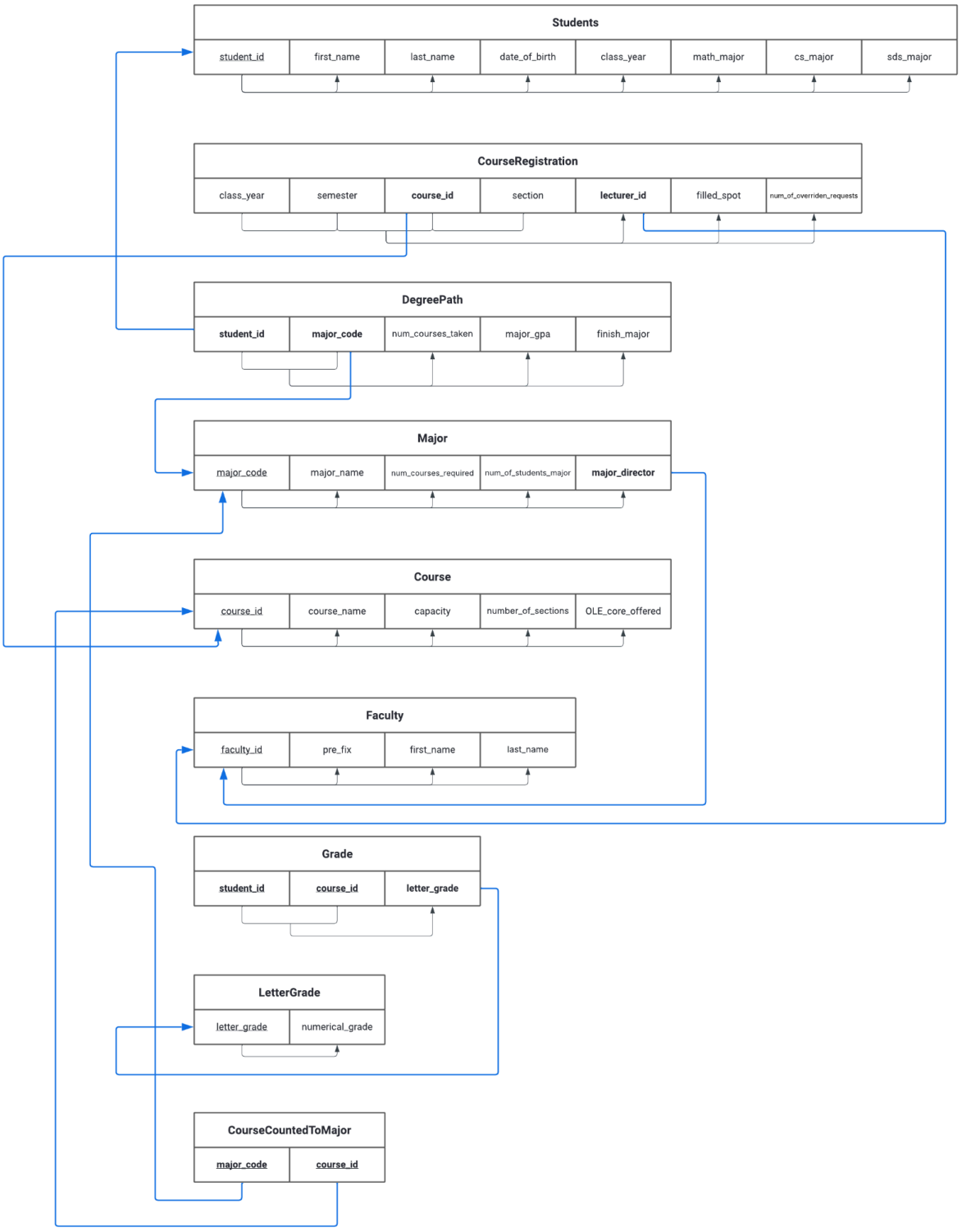


- Relationships in the ER Model included in the schema above:

  - Many-to-many **Take** relationship between **Student** and **Major**
  - Many-to-many **Count** relationship between **Major** and **Course**

4. Functional Dependency Diagram:

(Primary keys are underlined, foreign keys are bold)



Section B: Explanation of data and SQL Code:

Khanh Tra Nguyen Tran 24379393

1. Create tables:

```
CREATE TABLE Student (
  student_id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
  first_name VARCHAR(50) NOT NULL,
  last_name VARCHAR(50) NOT NULL,
  date_of_birth DATE,
  class_year YEAR NOT NULL,
  math_major BOOLEAN NOT NULL,
  cs_major BOOLEAN NOT NULL,
  sds_major BOOLEAN NOT NULL
);
```

The **Student** table stores core personal and academic information about each student. Each student has a unique integer for the auto-incremented student ID. The first name and last name have the type VARCHAR with a maximum length of 50. The date of birth is recorded with type DATE, while the class year has a type of YEAR. It also includes three boolean flags indicating whether the student is declared a Math, Computer Science (CS), or Statistics and Data Science (SDS) major.

- **Primary Key:** student_id
- **Constraints:** All fields are mandatory except date of birth.

```
CREATE TABLE Faculty (
  faculty_id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
  pre_fix VARCHAR(3),
  first_name VARCHAR(50) NOT NULL,
  last_name VARCHAR(50) NOT NULL
 );
```

The **Faculty** table stores core personal information about each faculty member working for the MSCS department. Each faculty member has a unique integer for the auto-incremented faculty ID. The first name and last name have the type VARCHAR with a maximum length of 50. Some faculty members may have a prefix if they obtained a higher education degree

- **Primary Key:** faculty_id
- **Constraints:** All fields are mandatory except prefix.

```
CREATE TABLE Course (
   course_id VARCHAR(10)  PRIMARY KEY,
   course_name VARCHAR(100) NOT NULL UNIQUE,
   capacity INT NOT NULL,
   number_of_sections INT NOT NULL,
   OLE_core_offered BOOLEAN
 );
```

The **Course** table stores information about each course offered by the MSCS department. The department offers classes for Maths, Computer Science, Statistics and Data Science majors. Each course has a unique course ID, which is a string. The name of the course is stored with the type VARCHAR with a maximum length of 100. The capacity (maximum number of students who can register for this course) and number of sections (multiple sections can be offered for a course in a semester) are stored as integers. There is a Boolean flag indicating whether the course offers the OLE core.

- **Primary Key:** course_id
- **Constraints:** All fields are mandatory except OLE_core_offered.

```
CREATE TABLE CourseRegistration (
   class_year YEAR NOT NULL,
   semester VARCHAR(1) NOT NULL CHECK (semester IN ('F', 'S', 'W')),
   course_id VARCHAR(10)NOT NULL,
   section VARCHAR(1) ,
   lecturer_id INT UNSIGNED,
   filled_spot INT UNSIGNED,
   num_of_overriden_requests INT UNSIGNED,
   UNIQUE (class_year, semester, course_id, section),
   FOREIGN KEY (course_id) REFERENCES Course(course_id),
   FOREIGN KEY (lecturer_id) REFERENCES Faculty(faculty_id)
);
```

The **CourseRegistration** table captures details about course offerings per semester and section. It links courses, faculty, and scheduling data, including enrollment and override request counts. It records the academic year the course section is offered for (type YEAR), a single character ('F', 'S', or 'W') indicating Fall, Spring, or Winter term, a course ID referencing the course being offered, a section label (one character: e.g. 'A', 'B') distinguishing multiple offerings of the same course, the faculty ID of the faculty member assigned to teach this course section, an integer indicating the umber of students who have successfully registered the section, and an integer indicating number of override (capacity-exceeding or prerequisite-overriding) enrollment requests pending.

- **Constraints:** class year, semester, and course ID are mandatory. A composite unique constraint ensures no duplicate section for the same course in the same semester/year.
- **Identifier:** (class_year, semester, course_id, section)

```
CREATE TABLE Grade (
   student_id INT UNSIGNED,
   course_id VARCHAR(10),
   letter_grade CHAR(2),
   FOREIGN KEY (student_id) REFERENCES Student(student_id),
   FOREIGN KEY (course_id) REFERENCES Course(course_id),
```

```
   FOREIGN KEY (letter_grade) REFERENCES LetterGrade(letter),
   PRIMARY KEY(student_id, course_id)
);
```

The **Grade** table records the final letter grades that students receive for specific courses. The student ID references the **Student** table, while the course ID references the **Course** table, establishing foreign key relationships that ensure only valid students and courses can be recorded. The letter_grade column stores the grade the student received in the American standard grading scale and references the **LetterGrade** table, which defines the set of valid letter grades and possibly their corresponding grade points. This structure ensures accurate and consistent tracking of academic performance across students and courses.

- **Primary Key:** (student_id, course_id) constraint to prevent duplicate entries for each course a student takes.

```
CREATE TABLE Major (
   major_code VARCHAR(3) PRIMARY KEY,
   major_name VARCHAR(50) NOT NULL,
   num_courses_required INT UNSIGNED NOT NULL,
   num_of_students_major INT UNSIGNED DEFAULT 0,
   major_director INT UNSIGNED,
  FOREIGN KEY (major_director) REFERENCES Faculty(faculty_id)
);
```

The **Major** table stores information about academic majors offered by the department. Each major is uniquely identified by a short code (major_code), which serves as the primary key. The major_name column holds the full name of the major and is required for each entry. The num_courses_required field indicates the total number of courses a student must complete to fulfill the major requirements and must be a non-negative integer. The num_of_students_major column tracks how many students are currently pursuing the major and defaults to 0. Additionally, the major_director field references the faculty_id of the professor overseeing the major, creating a foreign key relationship with the Faculty table. This structure allows for efficient tracking of major requirements, enrollment, and faculty oversight.

- **Primary Key:** major_code
- **Constraints:** The major code column is restricted to specific values ('M', 'CS', or 'SDS') using a CHECK constraint.

```
CREATE TABLE DegreePath (
   student_id INT UNSIGNED,
   major_code VARCHAR(3) CHECK (major_code IN ('M', 'CS', 'SDS')),
   num_courses_taken INT UNSIGNED DEFAULT 0,
   finish_major BOOLEAN DEFAULT false,
   major_gpa DOUBLE(3, 2),
   FOREIGN KEY (major_code) REFERENCES Major(major_code),
   FOREIGN KEY (student_id) REFERENCES Student(student_id),
   UNIQUE (student_id, major_code)
);
```

The **DegreePath** table tracks a student's progress toward completing a specific major. Each entry uniquely represents the relationship between a student and a declared major, identified by the combination of student ID and major code. The major code references the **Major** table. Similarly, student ID references the **Student** table. The table also records the number of courses the student has taken toward the major, whether they have completed the major, and their major GPA. Default values are set for course count (0) and completion status (false).

- **Constraints:** The major code column is restricted to specific values ('M', 'CS', or 'SDS') using a CHECK constraint. The UNIQUE constraint on (student_id, major_code) ensures that a student cannot be associated with the same major more than once.

```
CREATE TABLE LetterGrade (
   letter VARCHAR(2) PRIMARY KEY,
   numeric_grade DOUBLE(3, 2)
);
```

The **LetterGrade** table has two attributes, providing the conversion table for the grading system.

- **Primary Key:** letter

```
CREATE TABLE CourseCountedToMajor (
   major_code VARCHAR(3),
   course_id VARCHAR(10),
   FOREIGN KEY (major_code) REFERENCES Major(major_code) ON DELETE ,
   FOREIGN KEY (course_id) REFERENCES Course(course_id),
   PRIMARY KEY(major_code, course_id)
);
```

The **CourseCountedToMajor** records the courses students can take to fulfill their major. This is the relationship we have from the many-to-many between **Major** and **Course** entities.

- **Primary Key:** (major_code, course_id)

2. Altering and populating tables:

For tables that have auto-incremented IDs, we don’t need to specify the IDs during insertion.

Table	Data populated
Faculty	`INSERT INTO Faculty (pre_fix, first_name, last_name) VALUES ('Dr.', 'Ryota', 'Matsuura');` `INSERT INTO Faculty (pre_fix, first_name, last_name) VALUES ('Dr.', 'Olaf', 'Hall-Holt');`

Khanh Tra Nguyen Tran 24379393

	`INSERT INTO Faculty (pre_fix, first_name, last_name) VALUES ('Dr.', 'Melissa', 'Lynn');`
	`INSERT INTO Faculty (pre_fix, first_name, last_name) VALUES ('Dr.', 'Paul', 'Roback');`
	`INSERT INTO Faculty (pre_fix, first_name, last_name) VALUES ('Dr.', 'Jaime', 'Davila');`
	`INSERT INTO Faculty (pre_fix, first_name, last_name) VALUES ('Dr.', 'Matthew', 'Wright');`
	`INSERT INTO Faculty (pre_fix, first_name, last_name) VALUES ('', 'Kim', 'Mandery');`
	`INSERT INTO Faculty (pre_fix, first_name, last_name) VALUES ('Dr.', 'Will', 'Leeson');`
	`INSERT INTO Faculty (pre_fix, first_name, last_name) VALUES ('Dr.', 'Jane', 'Willows');`
	`INSERT INTO Faculty (pre_fix, first_name, last_name) VALUES ('Dr.', 'Matt', 'Prinkles');`
	`INSERT INTO Faculty (pre_fix, first_name, last_name) VALUES ('Dr', 'Alice', 'Smith');`
	`INSERT INTO Faculty (pre_fix, first_name, last_name) VALUES ('Dr.', 'Bob', 'Johnson');`
	`INSERT INTO Faculty (pre_fix, first_name, last_name) VALUES ('Dr.', 'Ian', 'Taylor');`
	`INSERT INTO Faculty (pre_fix, first_name, last_name) VALUES ('Dr.', 'Jack', 'Anderson');`

Major	`INSERT INTO Major VALUES ('M', 'Math', 8, 0, 6);`
	`INSERT INTO Major VALUES ('CS', 'Computer Science', 7, 0, 1);`
	`INSERT INTO Major VALUES ('SDS', 'Statistics and Data Science', 6, 0, 4);`

Student	`INSERT INTO Student (first_name, last_name, date_of_birth, class_year, math_major, cs_major, sds_major) VALUES ('Fayellete', 'Tran', '2003-11-15', 2026, 1, 1, 1);`
	`INSERT INTO Student (first_name, last_name, date_of_birth, class_year, math_major, cs_major, sds_major) VALUES ('Khae', 'Trinh', '2001-03-01', 2023, 0, 0, 1);`
	`INSERT INTO Student (first_name, last_name, date_of_birth, class_year, math_major, cs_major, sds_major) VALUES ('Quinn', 'Jane', '2003-06-01', 2030, 1, 0, 0);`
	`INSERT INTO Student (first_name, last_name, date_of_birth, class_year, math_major, cs_major, sds_major) VALUES ('Anna', 'Ally', '2004-03-18', 2027, 1, 0, 1);`
	`INSERT INTO Student (first_name, last_name, date_of_birth, class_year, math_major, cs_major, sds_major) VALUES ('Mindy', 'Cian', '2006-06-01', 2028, 0, 1, 1);`

Khanh Tra Nguyen Tran 24379393

Course

```
INSERT INTO Course VALUES ('M120', 'Calculus I', 30, 6, false);
INSERT INTO Course VALUES ('M126', 'Calculus II', 25, 6, false);
INSERT INTO Course VALUES ('M128', 'Honors Calculus II', 25, 6, false);
INSERT INTO Course VALUES ('M220', 'Elementary Linear Algebra', 20, 2,
false);
INSERT INTO Course VALUES ('M226', 'Multivariable Calculus', 25, 1,
false);
INSERT INTO Course VALUES ('M232', 'Differential Equations I', 20, 2,
false);
INSERT INTO Course VALUES ('M234', 'Discrete Math', 24, 1, false);
INSERT INTO Course VALUES ('M242', 'Modern Computational Math', 24, 2,
false);
INSERT INTO Course VALUES ('M244', 'Real Analysis', 18, 2, false);
INSERT INTO Course VALUES ('M252', 'Abstract Algebra', 24, 2, false);
INSERT INTO Course VALUES ('M262', 'Probability Theory', 18, 2, false);
INSERT INTO Course VALUES ('M320', 'Advanced Linear Algebra', 20, 1,
false);
INSERT INTO Course VALUES ('M332', 'Differential Equations II', 20, 1,
false);

INSERT INTO Course VALUES ('CS121', 'Introduction to Computer Science',
30, 3, true);
INSERT INTO Course VALUES ('CS221', 'Data Structures in C++', 30, 2,
false);
INSERT INTO Course VALUES ('CS241', 'Hardware Design', 25, 2, false);
INSERT INTO Course VALUES ('CS251', 'Software Design', 25, 2, false);
INSERT INTO Course VALUES ('CS263', 'Ethics for Software Design', 20,
2, false);
INSERT INTO Course VALUES ('CS276', 'Programming Language', 25, 1,
false);
INSERT INTO Course VALUES ('CS353', 'Algorithm Analysis', 25, 2,
false);
INSERT INTO Course VALUES ('CS379', 'Introduction to Artificial
Intelligence', 30, 1, false);

INSERT INTO Course VALUES ('SDS164', 'Introduction to Data Science',
30, 2, true);
INSERT INTO Course VALUES ('SDS172', 'Introduction to Statistics ', 30,
5, true);
INSERT INTO Course VALUES ('SDS250', 'Principle of Data Visualization',
30, 1, false);
INSERT INTO Course VALUES ('SDS264', 'Advanced Data Science', 30, 1,
false);
INSERT INTO Course VALUES ('SDS272', 'Statistical Modelling', 30, 2,
```

	false); INSERT INTO Course VALUES ('SDS322', 'Statistical Theory', 30, 1, false); INSERT INTO Course VALUES ('SDS341', 'Algorithms for Decision Making', 30, 1, false);
CourseRegistration	INSERT INTO CourseRegistration (class_year, semester, course_id, section, lecturer_id, filled_spot, num_of_overriden_requests) VALUES (2024, 'F', 'M120', 'A', 9, 25, 10); INSERT INTO CourseRegistration (class_year, semester, course_id, section, lecturer_id, filled_spot, num_of_overriden_requests) VALUES (2024, 'F', 'M120', 'B', 10, 20, 0); INSERT INTO CourseRegistration (class_year, semester, course_id, section, lecturer_id, filled_spot, num_of_overriden_requests) VALUES (2024, 'F', 'M120', 'C', 9, 15, 0); INSERT INTO CourseRegistration (class_year, semester, course_id, section, lecturer_id, filled_spot, num_of_overriden_requests) VALUES (2024, 'F', 'M120', 'D', 10, 24, 0); INSERT INTO CourseRegistration (class_year, semester, course_id, section, lecturer_id, filled_spot, num_of_overriden_requests) VALUES (2024, 'F', 'M120', 'E', 9, 20, 0); INSERT INTO CourseRegistration (class_year, semester, course_id, section, lecturer_id, filled_spot, num_of_overriden_requests) VALUES (2024, 'F', 'M120', 'F', 9, 26, 5); INSERT INTO CourseRegistration (class_year, semester, course_id, section, lecturer_id, filled_spot, num_of_overriden_requests) VALUES (2024, 'F', 'M126', 'A', 10, 25, 0); INSERT INTO CourseRegistration (class_year, semester, course_id, section, lecturer_id, filled_spot, num_of_overriden_requests) VALUES (2024, 'F', 'M126', 'B', 9, 25, 2); INSERT INTO CourseRegistration (class_year, semester, course_id, section, lecturer_id, filled_spot, num_of_overriden_requests) VALUES (2024, 'F', 'M126', 'C', 10, 25, 2); INSERT INTO CourseRegistration (class_year, semester, course_id, section, lecturer_id, filled_spot, num_of_overriden_requests) VALUES (2024, 'F', 'M126', 'D', 9, 20, 0); INSERT INTO CourseRegistration (class_year, semester, course_id, section, lecturer_id, filled_spot, num_of_overriden_requests) VALUES (2024, 'F', 'M126', 'E', 10, 21, 0); INSERT INTO CourseRegistration (class_year, semester, course_id, section, lecturer_id, filled_spot, num_of_overriden_requests) VALUES (2024, 'F', 'M126', 'F', 9, 15, 0); …

Khanh Tra Nguyen Tran 24379393

Grade

```
INSERT INTO Grade VALUES(1, 'CS221', 'A+'), (1, 'CS251', 'A'), (1, 'CS353', 'A+'), (1, 'CS241', 'A'), (1, 'M120', 'A+'), (1, 'M128', 'A'), (1, 'M220', 'A'), (1, 'M234', 'A'), (1, 'M252', 'A'), (1, 'M262', 'A+'), (1, 'M242', 'A'), (1, 'SDS164', 'A+'), (1, 'SDS172', 'A+'), (1, 'SDS264', 'A'), (1, 'SDS341', 'A');

INSERT INTO Grade VALUES(2, 'SDS164', 'A'), (2, 'SDS172', 'A-'), (2, 'SDS250', 'B+'), (2, 'SDS264', 'A'), (2, 'SDS272', 'A-'), (2, 'SDS322', 'A'), (2, 'SDS341', 'A'), (2, 'M126', 'A+'), (2, 'M220', 'A'), (2, 'M262', 'A');

INSERT INTO Grade VALUES(3, 'CS121', 'A-'), (3, 'CS221', 'A-'), (3, 'CS251', 'A-'), (3, 'CS353', 'A'), (3, 'CS241', 'A+'), (3, 'SDS341', 'A'), (3, 'M120', 'A+'), (3, 'M126', 'A-'), (3, 'M220', 'A'), (3, 'M234', 'A');

INSERT INTO Grade VALUES(4, 'CS121', 'A'), (4, 'CS221', 'B'), (4, 'CS251', 'B+'), (4, 'CS241', 'A-'), (4, 'M120', 'A-'), (4, 'M126', 'B+'), (4, 'M220', 'A-'), (4, 'M234', 'B'), (4, 'SDS164', 'A'), (4, 'SDS172', 'A-');

INSERT INTO Grade VALUES(5, 'CS121', 'A-'), (5, 'CS221', 'A+'), (5, 'M126', 'A'), (5, 'M220', 'A+'), (5, 'SDS172', 'A');
```

CourseCountedToMajor

```
INSERT INTO CourseCountedToMajor VALUES ('M', 'M120');
INSERT INTO CourseCountedToMajor VALUES ('M', 'M126');
INSERT INTO CourseCountedToMajor VALUES ('M', 'M128');
INSERT INTO CourseCountedToMajor VALUES ('M', 'M220');
INSERT INTO CourseCountedToMajor VALUES ('CS', 'M220');
INSERT INTO CourseCountedToMajor VALUES ('M', 'M226');
INSERT INTO CourseCountedToMajor VALUES ('M', 'M232');
INSERT INTO CourseCountedToMajor VALUES ('M', 'M234');
INSERT INTO CourseCountedToMajor VALUES ('CS', 'M234');
INSERT INTO CourseCountedToMajor VALUES ('M', 'M242');
INSERT INTO CourseCountedToMajor VALUES ('M', 'M244');
INSERT INTO CourseCountedToMajor VALUES ('M', 'M252');
INSERT INTO CourseCountedToMajor VALUES ('M', 'M262');
INSERT INTO CourseCountedToMajor VALUES ('SDS', 'M220');
INSERT INTO CourseCountedToMajor VALUES ('M', 'M320');
INSERT INTO CourseCountedToMajor VALUES ('M', 'M332');

INSERT INTO CourseCountedToMajor VALUES ('CS','CS121');
INSERT INTO CourseCountedToMajor VALUES ('CS','CS221');
INSERT INTO CourseCountedToMajor VALUES ('CS','CS241');
```

	`INSERT INTO CourseCountedToMajor VALUES ('CS','CS251');`
	`INSERT INTO CourseCountedToMajor VALUES ('CS','CS263');`
	`INSERT INTO CourseCountedToMajor VALUES ('CS','CS276');`
	`INSERT INTO CourseCountedToMajor VALUES ('CS','CS353');`
	`INSERT INTO CourseCountedToMajor VALUES ('M','CS353');`
	`INSERT INTO CourseCountedToMajor VALUES ('CS','CS379');`
	
	`INSERT INTO CourseCountedToMajor VALUES ('SDS','SDS164');`
	`INSERT INTO CourseCountedToMajor VALUES ('SDS','SDS172');`
	`INSERT INTO CourseCountedToMajor VALUES ('SDS','SDS250');`
	`INSERT INTO CourseCountedToMajor VALUES ('SDS','SDS264');`
	`INSERT INTO CourseCountedToMajor VALUES ('SDS','SDS272');`
	`INSERT INTO CourseCountedToMajor VALUES ('M','SDS264');`
	`INSERT INTO CourseCountedToMajor VALUES ('M','SDS272');`
	`INSERT INTO CourseCountedToMajor VALUES ('SDS','SDS322');`
	`INSERT INTO CourseCountedToMajor VALUES ('M','SDS322');`
	`INSERT INTO CourseCountedToMajor VALUES ('SDS','SDS341');`
	`INSERT INTO CourseCountedToMajor VALUES ('M','SDS341');`

LetterGrade	`INSERT INTO LetterGrade VALUES ('A+', 4.0), ('A', 4.0), ('A-', 3.7), ('B+', 3.3), ('B', 3.0), ('B-', 2.7), ('C+', 2.3), ('C', 2.0), ('C-', 1.7), ('D+', 1.3), ('D', 1.0), ('F', 0.0);`

3. Triggers:

Trigger	Description

`CREATE TRIGGER MajorCountAdd`
`AFTER INSERT ON Student`
`FOR EACH ROW`
`BEGIN`
`  -- Update Math Major Count`
`  IF NEW.math_major = 1 THEN`
`  UPDATE Major`
`  SET num_of_students_major = num_of_students_major + 1`
`  WHERE major_code = 'M';`
 	
`  END IF;`
`  -- Update CS Major Count`
`  IF NEW.cs_major = 1 THEN`
	This is triggered every time we insert a new student entry into the **Student** table. The trigger automatically increments the number of students declaring the corresponding major(s) in the **Major** table.

```
        UPDATE Major
        SET num_of_students_major =
num_of_students_major + 1
        WHERE major_code = 'CS';
    END IF;

    -- Update SDS Major Count
    IF NEW.sds_major = 1 THEN
        UPDATE Major
        SET num_of_students_major =
num_of_students_major + 1
        WHERE major_code = 'SDS';
    END IF;
END;
```

```
CREATE TRIGGER MajorCountDelete
AFTER DELETE ON Student
FOR EACH ROW
BEGIN
    -- Update Math Major Count
    IF OLD.math_major = 1 THEN
        UPDATE Major
        SET num_of_students_major =
num_of_students_major - 1
        WHERE major_code = 'M';
    END IF;

    -- Update CS Major Count
    IF OLD.cs_major = 1 THEN
        UPDATE Major
        SET num_of_students_major =
num_of_students_major - 1
        WHERE major_code = 'CS';
    END IF;

    -- Update SDS Major Count
    IF OLD.sds_major = 1 THEN
        UPDATE Major
        SET num_of_students_major =
num_of_students_major - 1
        WHERE major_code = 'SDS';
    END IF;
END;
```

This is triggered every time we delete an existing student entry in the **Student** table. The trigger automatically decrements the number of students declaring the corresponding major(s) in the **Major** table.

```
CREATE TRIGGER MajorCountUpdate
AFTER UPDATE ON Student
FOR EACH ROW
BEGIN
   -- Update Math Major Count
   IF NEW.math_major = 1 AND OLD.math_major =
0 THEN
       UPDATE Major
       SET num_of_students_major =
num_of_students_major + 1
       WHERE major_code = 'M';
   ELSEIF NEW.math_major = 0 AND
OLD.math_major = 1 THEN
       UPDATE Major
       SET num_of_students_major =
num_of_students_major - 1
       WHERE major_code = 'M';
   END IF;

   -- Update CS Major Count
   IF NEW.cs_major = 1 AND OLD.cs_major = 0
THEN
       UPDATE Major
       SET num_of_students_major =
num_of_students_major + 1
       WHERE major_code = 'CS';
   ELSEIF NEW.cs_major = 0 AND OLD.cs_major =
1 THEN
       UPDATE Major
       SET num_of_students_major =
num_of_students_major - 1
       WHERE major_code = 'CS';
   END IF;

   -- Update SDS Major Count
   IF NEW.sds_major = 1 AND OLD.sds_major = 0
THEN
       UPDATE Major
       SET num_of_students_major =
num_of_students_major + 1
       WHERE major_code = 'SDS';
   ELSEIF NEW.sds_major = 0 AND OLD.sds_major
= 1 THEN
       UPDATE Major
```

This is triggered every time we update an existing student entry in the **Student** table by changing whether the student declares or drops a major(s). The trigger automatically increments or decrements (accordingly) the number of students declaring the corresponding major(s) in the **Major** table.

`SET num_of_students_major = num_of_students_major - 1`
`WHERE major_code = 'SDS';`
`END IF;`
 	
`END;`	

`CREATE TRIGGER DegreePathAdd`
`AFTER Insert ON Student`
`FOR EACH ROW`
`BEGIN`
`-- Update Math Major Count`
`IF NEW.math_major = 1 THEN`
`INSERT INTO DegreePath`
`VALUES (NEW.student_id, 'M', 0, false, 0.0);`
`END IF;`
 	
`-- Update CS Major Count`
`IF NEW.cs_major = 1 THEN`
`INSERT INTO DegreePath`
`VALUES (NEW.student_id, 'CS', 0, false, 0.0);`
`END IF;`
 	
`-- Update SDS Major Count`
`IF NEW.sds_major = 1 THEN`
`INSERT INTO DegreePath`
`VALUES (NEW.student_id, 'SDS', 0, false, 0.0);`
`END IF;`
`END;`
	This is triggered every time we add a new student entry in the **Student** table. The trigger automatically adds a new entry into the **DegreePath** according to the majors the student has declared. If they declare a major before taking any courses in that major, the entry will record that the student has taken no courses and the major GPA is 0.0.

`CREATE TRIGGER DegreePathDelete`
`AFTER DELETE ON Student`
`FOR EACH ROW`
`BEGIN`
`DELETE FROM DegreePath WHERE student_id = OLD.student_id;`
`END;`
	This is triggered every time we delete an existing student entry in the **Student** table. The trigger automatically drops the existing entries related to their student in the **DegreePath**.

`CREATE TRIGGER DegreePathUpdate`
`AFTER UPDATE ON Student`
`FOR EACH ROW`
	This is triggered every time we update an existing student entry in the **Student** table. The trigger automatically drops the existing entries related to their student in the **DegreePath**. If the current student declares a new major,

```
BEGIN
   -- Declare all variables at the beginning
(MySQL requires this)
   DECLARE MathCourseTaken INT DEFAULT 0;
   DECLARE CSCourseTaken INT DEFAULT 0;
   DECLARE SDSCourseTaken INT DEFAULT 0;

   DECLARE MathGPA DECIMAL(4,2) DEFAULT 0.0;
   DECLARE CSGPA DECIMAL(4,2) DEFAULT 0.0;
   DECLARE SDSGPA DECIMAL(4,2) DEFAULT 0.0;

   DECLARE MathReq INT DEFAULT 0;
   DECLARE CSReq INT DEFAULT 0;
   DECLARE SDSReq INT DEFAULT 0;

   -- MATH MAJOR LOGIC
   -- MATH major
   IF NEW.math_major = 1 AND OLD.math_major =
0 THEN
       SELECT COUNT(*) INTO MathCourseTaken
       FROM Grade G
       JOIN CourseCountedToMajor CCM ON
G.course_id = CCM.course_id
       WHERE G.student_id = NEW.student_id AND
CCM.major_code = 'M';

       IF MathCourseTaken = 0 THEN
           INSERT INTO DegreePath VALUES
(NEW.student_id, 'M', 0, FALSE, 0.0);
       ELSE
           SELECT
SUM(LG.numeric_grade)/COUNT(*) INTO MathGPA
           FROM Grade G
           JOIN CourseCountedToMajor CCM ON
G.course_id = CCM.course_id
           JOIN LetterGrade LG ON
G.letter_grade = LG.letter
           WHERE G.student_id = NEW.student_id
AND CCM.major_code = 'M';

           SELECT num_courses_required INTO
MathReq FROM Major WHERE major_code = 'M';

           INSERT INTO DegreePath
```

the trigger automatically adds a new entry into the **DegreePath** according to the majors the student has declared. If they declare a major before taking any courses in that major, the entry will record that the student has taken no courses and the major GPA is 0.0. If the current student drops a new major, the trigger automatically deletes the existing entry in the **DegreePath** according to the majors the student has dropped.

```
            VALUES (NEW.student_id, 'M',
MathCourseTaken, (MathCourseTaken >= MathReq),
MathGPA);
        END IF;


    ELSEIF NEW.math_major = 0 AND
OLD.math_major = 1 THEN
        DELETE FROM DegreePath WHERE student_id
= OLD.student_id AND major_code = 'M';
    END IF;

    -- CS MAJOR LOGIC
    IF NEW.cs_major = 1 AND OLD.cs_major = 0
THEN
        SELECT COUNT(*) INTO CSCourseTaken
        FROM Grade G
        JOIN CourseCountedToMajor CCM ON
G.course_id = CCM.course_id
        WHERE G.student_id = NEW.student_id AND
CCM.major_code = 'CS';

        IF CSCourseTaken = 0 THEN
            INSERT INTO DegreePath VALUES
(NEW.student_id, 'CS', 0, FALSE, 0.0);
        ELSE
            SELECT
SUM(LG.numeric_grade)/COUNT(*) INTO CSGPA
            FROM Grade G
            JOIN CourseCountedToMajor CCM ON
G.course_id = CCM.course_id
            JOIN LetterGrade LG ON
G.letter_grade = LG.letter
            WHERE G.student_id = NEW.student_id
AND CCM.major_code = 'CS';

            SELECT num_courses_required INTO
CSReq FROM Major WHERE major_code = 'CS';

            INSERT INTO DegreePath
            VALUES (NEW.student_id, 'CS',
CSCourseTaken, (CSCourseTaken >= CSReq),
CSGPA);
        END IF;
```

```
   ELSEIF NEW.cs_major = 0 AND OLD.cs_major =
1 THEN
       DELETE FROM DegreePath WHERE student_id
= OLD.student_id AND major_code = 'CS';
   END IF;

   -- SDS MAJOR LOGIC
   IF NEW.sds_major = 1 AND OLD.sds_major = 0
THEN
       SELECT COUNT(*) INTO SDSCourseTaken
       FROM Grade G
       JOIN CourseCountedToMajor CCM ON
G.course_id = CCM.course_id
       WHERE G.student_id = NEW.student_id AND
CCM.major_code = 'SDS';

       IF SDSCourseTaken = 0 THEN
           INSERT INTO DegreePath VALUES
(NEW.student_id, 'SDS', 0, FALSE, 0.0);
       ELSE
           SELECT
SUM(LG.numeric_grade)/COUNT(*) INTO SDSGPA
           FROM Grade G
           JOIN CourseCountedToMajor CCM ON
G.course_id = CCM.course_id
           JOIN LetterGrade LG ON
G.letter_grade = LG.letter
           WHERE G.student_id = NEW.student_id
AND CCM.major_code = 'SDS';

           SELECT num_courses_required INTO
SDSReq FROM Major WHERE major_code = 'SDS';

           INSERT INTO DegreePath
           VALUES (NEW.student_id, 'SDS',
SDSCourseTaken, (SDSCourseTaken >= SDSReq),
SDSGPA);
       END IF;

   ELSEIF NEW.sds_major = 0 AND OLD.sds_major
= 1 THEN
       DELETE FROM DegreePath WHERE student_id
= OLD.student_id AND major_code = 'SDS';
```

```
   END IF;

END;
//
```

```
CREATE TRIGGER GradeAdd
AFTER INSERT ON Grade
FOR EACH ROW
BEGIN
   DECLARE GradeNumeric DECIMAL(3,2);

   -- Convert letter grade to numeric
   SELECT numeric_grade INTO GradeNumeric
   FROM LetterGrade
   WHERE letter = NEW.letter_grade;

   -- Update all DegreePath rows where the
course is counted toward a major
   UPDATE DegreePath DP
   JOIN CourseCountedToMajor CCM ON
CCM.major_code = DP.major_code
   SET DP.major_gpa = (DP.num_courses_taken *
DP.major_gpa + GradeNumeric) /
(DP.num_courses_taken + 1),
       DP.num_courses_taken =
DP.num_courses_taken + 1
   WHERE DP.student_id = NEW.student_id
     AND CCM.course_id = NEW.course_id;

END;
```

This is triggered every time we add a grade entry in the **Grade** table. This indicates the grade a student receives after taking a course offered by the MSCS department. If the student has declared the major related to the course before, the trigger will automatically increment the number of courses they have taken for the major and update the major GPA correspondingly.

```
CREATE TRIGGER GradeDelete
AFTER DELETE ON Grade
FOR EACH ROW
BEGIN
   DECLARE GradeNumeric DECIMAL(3,2);

   -- Convert letter grade to numeric
   SELECT numeric_grade INTO GradeNumeric
   FROM LetterGrade
   WHERE letter = OLD.letter_grade;

   -- Update DegreePath for all majors where
the deleted course counts
   UPDATE DegreePath DP
```

This is triggered every time we delete a grade entry in the **Grade** table. This indicates the grade a student receives after taking a course offered by the MSCS department. If the student has declared the major related to the course before, the trigger will automatically decrement the number of courses they have taken for the major and update the major GPA correspondingly.

```
   JOIN CourseCountedToMajor CCM ON
DP.major_code = CCM.major_code
   SET
       major_gpa = CASE
           WHEN DP.num_courses_taken = 1 THEN
0.0
           ELSE (DP.num_courses_taken *
DP.major_gpa - GradeNumeric) /
(DP.num_courses_taken - 1)
       END,
       num_courses_taken =
DP.num_courses_taken - 1
   WHERE DP.student_id = OLD.student_id
     AND CCM.course_id = OLD.course_id;

END;
```

```
CREATE TRIGGER GradeUpdate
AFTER UPDATE ON Grade
FOR EACH ROW
BEGIN
   DECLARE OldGradeNumeric DECIMAL(3,2);
   DECLARE NewGradeNumeric DECIMAL(3,2);

   -- Convert both old and new letter grades
   SELECT numeric_grade INTO OldGradeNumeric
   FROM LetterGrade
   WHERE letter = OLD.letter_grade;

   SELECT numeric_grade INTO NewGradeNumeric
   FROM LetterGrade
   WHERE letter = NEW.letter_grade;

   -- Update GPA for majors that count this
course
   UPDATE DegreePath DP
   JOIN CourseCountedToMajor CCM ON
DP.major_code = CCM.major_code
   SET major_gpa =
       (DP.num_courses_taken * DP.major_gpa -
OldGradeNumeric + NewGradeNumeric)
       / DP.num_courses_taken
   WHERE DP.student_id = NEW.student_id
     AND CCM.course_id = OLD.course_id;
```

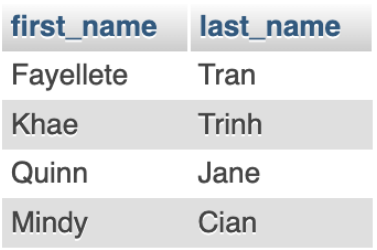
This is triggered every time we update a grade entry in the **Grade** table. This indicates the grade a student receives after taking a course offered by the MSCS department. Usually, they update the letter grade and update the major GPA.

`END;`	

`CREATE TRIGGER MajorCompleted AFTER UPDATE ON DegreePath`
`FOR EACH ROW BEGIN`
`  DECLARE NumCoursesRequired INT;`
`  SELECT num_courses_required INTO NumCoursesRequired`
`  FROM Major`
`  WHERE major_code = NEW.major_code;`
`  IF (NEW.num_courses_taken = NumCoursesRequired AND NEW.major_gpa >= 2.5) THEN`
`  SET NEW.finish_major = true;`
`  ELSE`
`  SET NEW.finish_major = false;`
`  END IF;`
`END`
	This is triggered every time we update the grade entry in the **Grade** table. If the student has taken enough courses required by the major and received a decent major GPA (>= 2.5/4.0), the trigger automatically finishes the major.

4. Views:

Views	Description	Example

`DROP VIEW IF EXISTS dean_list;`
`CREATE VIEW dean_list AS`
`SELECT DISTINCT s.first_name, s.last_name,`
`FROM Student AS s JOIN DegreePath as d ON s.student_id = d.student_id`
`WHERE d.major_gpa >= 3.75;`
	This view prints out the first names and last names of students nominated to the Dean's List (who achieve at least 3.75/4.0 for at least one major GPA).	

`DROP VIEW IF EXISTS graduate_ready;`
`CREATE VIEW graduate_ready AS`
`WITH helper AS(`
`  SELECT s.student_id,`
`  CASE`
`  WHEN (AVG(d.major_gpa) >= 3.9) THEN 'Summa Cum Laude'`
`  WHEN (AVG(d.major_gpa) >= 3.75) THEN 'Magna Cum Laude'`
`  WHEN`
	This view prints out the student IDs, first names, last names, and the honor of students who are ready to graduate (finish all the declared majors with decent major GPAs). Besides, based on the average major GPAs, the students can receive a corresponding honor (Summa Cum Laude: >= 3.9/4.0; Magna Cum Laude: 3.75/4.0; Cum Laude: 3.6/4.0).	

```
(AVG(d.major_gpa) >= 3.6) THEN
'Cum Laude'
          ELSE NULL
      END AS honor,
      CASE
          WHEN (SUM(CASE
WHEN(d.finish_major = true) THEN
1 ELSE 0 END) = (s.math_major +
s.cs_major + s.sds_major)) THEN
true
          ELSE FALSE
      END AS graduate_ready
   FROM Student AS s JOIN
DegreePath as d
   ON s.student_id =
d.student_id
   GROUP BY s.student_id)
SELECT s.first_name,
s.last_name, s.class_year,
h.honor
FROM Student AS s JOIN helper as
h ON s.student_id = h.student_id
WHERE h.graduate_ready = TRUE;
```

```
DROP VIEW IF EXISTS
faculty_classes;
CREATE VIEW faculty_classes AS
SELECT f.faculty_id,
f.first_name, f.last_name,
COUNT(*) AS number_classes_teach
FROM Faculty AS f JOIN
CourseRegistration as c
ON f.faculty_id = c.lecturer_id
GROUP BY f.faculty_id;
```

This view prints out the faculty members' IDs, first names, last names, and the number of course sections they have taught.

faculty_id	first_name	last_name	number_classes_teach
1	Ryota	Matsuura	3
2	Olaf	Hall-Holt	4
3	Melissa	Lynn	3
4	Paul	Roback	6
5	Jaime	Davila	1
6	Matthew	Wright	3
7	Kim	Mandery	1
8	Will	Leeson	2
9	Jane	Willows	9
10	Matt	Prinkles	8
11	Alice	Smith	3
12	Bob	Johnson	3
13	Ian	Taylor	3
14	Jack	Anderson	6

```
DROP VIEW IF EXISTS directors;
CREATE VIEW directors AS
SELECT f.faculty_id,
f.first_name, f.last_name,
m.major_name AS major
FROM Major AS m JOIN Faculty as
f
ON m.major_director =
f.faculty_id;
```

This view prints out the director of each major's faculty ID, first name, last name, and the name of the major they direct.

faculty_id	first_name	last_name	major
1	Ryota	Matsuura	Computer Science
6	Matthew	Wright	Math
4	Paul	Roback	Statistics and Data Science

5. Security Command - role-based access control:

```
DROP ROLE IF EXISTS 'mscs_director';

CREATE ROLE 'mscs_director';

GRANT ALL PRIVILEGES ON knguyent_2425_db TO 'mscs_director' WITH GRANT OPTION;

DROP ROLE IF EXISTS 'major_director';

CREATE ROLE 'major_director';

GRANT SELECT ON knguyent_2425_db.Major TO 'major_director' ;

GRANT SELECT ON knguyent_2425_db.Student TO 'major_director';

GRANT SELECT ON knguyent_2425_db.DegreePath TO 'major_director';

GRANT SELECT ON knguyent_2425_db.CourseRegistration TO 'major_director';

DROP ROLE IF EXISTS 'faculty_member';

CREATE ROLE 'faculty_member';

GRANT ALL PRIVILEGES ON knguyent_2425_db.Grade TO 'faculty_member';

GRANT SELECT ON knguyent_2425_db.Student TO 'faculty_member';

GRANT SELECT ON knguyent_2425_db.CourseRegistration TO 'faculty_member';

DROP ROLE IF EXISTS 'student';

CREATE ROLE 'student';

GRANT SELECT ON knguyent_2425_db.CourseRegistration TO 'student';
```

I use SQL to set up role-based security in the **knguyent_2425_db** database by creating and assigning permissions to specific user roles. Each role is tailored to a type of user within the academic system, with privileges granted based on their responsibilities:

- **mscs_director:** having full access (ALL PRIVILEGES) to the entire database. This role represents a superuser or overall program director with unrestricted control. They are also be able to grant all privileges to anyone they want.
- **major_director:** viewing data (SELECT) related to majors, students, degree paths, and course registrations. This role is designed for academic major directors who need read-only access to evaluate student progress.
- **faculty_member:** reading from **Student** and **CourseRegistration** tables and having full control over the **Grade** table—allowing them to assign, update, or delete grades for students.
- **student:** having read-only access to the **CourseRegistration** table so they can view available course offerings and sections.