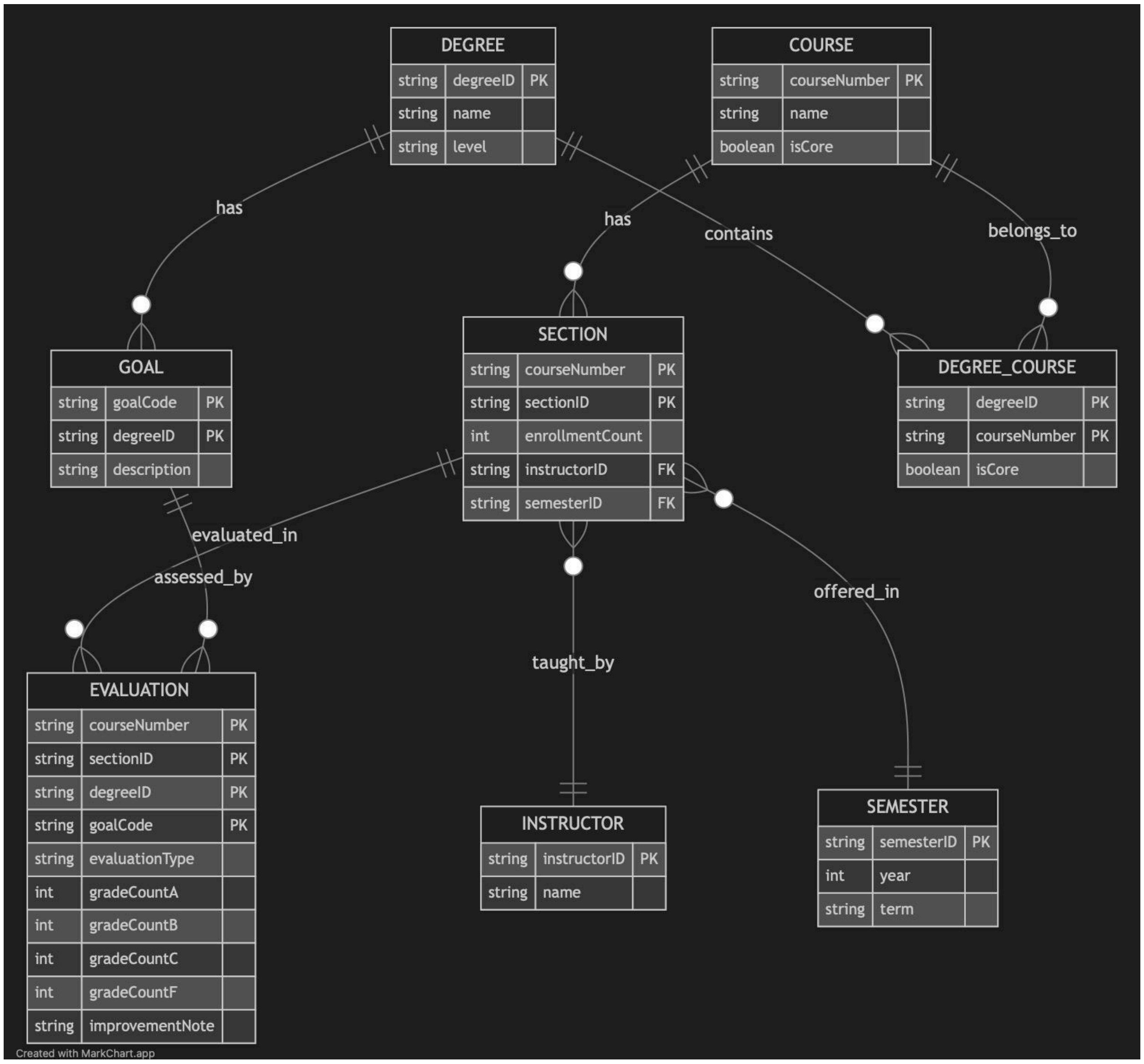
Group 4:
Hunter Ryan
Keaton Harvey
Elijah Posiulai
Daymien Zavala

ER Diagram



1. Entities and Attributes:
    - Degree: degreeID (PK), name, level
    - Course: courseNumber (PK), name, isCore

    - Semester: semesterID (PK), year, term
    - Goal: Composite PK of goalCode and degreeID, Description
    - Instructor: InstructorID (PK), Name
    - Section: Composite PK of courseNumber and sectionID, enrollmentCount, Composite FK of instructorID and semesterID
    - **Relationship: Degree_Course: Composite PK of courseNumber, degreeID, isCore**
    - Evaluation: Composite PK of courseNumber and sectionID and degreeID and goalCode, evaluationType, gradeCountA, gradeCountB, gradeCountC, gradeCountF, improvementNote

2. Relationships:

Degree - Course (via Degree_Course join table)

- Relationship: Contains / Belongs To
- Type: Many-to-Many
- Explanation:
    - Each Degree can include multiple Courses, and each Course can be part of multiple Degrees. This many-to-many relationship is managed by the Degree_Course join table.
    - Degree_Course includes the isCore attribute, which indicates whether a specific course is a core requirement for the degree. This allows flexibility in defining the curriculum requirements for each degree.
- Cardinality: Many-to-Many, represented by a composite primary key in Degree_Course (degreeID and courseNumber).

Degree - Goal

- Relationship: Has
- Type: One-to-Many
- Explanation:
    - Each Degree has a unique set of Goals associated with it. These goals represent the learning objectives or competencies that students are expected to achieve within the degree program.
    - The Goal entity has a composite primary key (goalCode and degreeID), ensuring that each goal is unique within the context of the degree. The description attribute provides details about each goal.
- Cardinality: One-to-Many, as each degree has multiple goals, but each goal is specific to one degree.

Course - Section

- Relationship: Has
- Type: One-to-Many
- Explanation:
    - Each Course "has" multiple Sections associated with it. A section represents a specific instance of the course offered during a particular semester, often with unique details such as the instructor, enrollment count, and evaluation data.
    - The Section entity has a composite primary key (courseNumber, sectionID), where courseNumber identifies the course and sectionID distinguishes the specific section within that course.
    - This relationship allows the tracking of different offerings of the same course across semesters and instructors, making it possible to see which sections belong to which course.
- Cardinality: One-to-Many, as each course can have multiple sections, but each section belongs to only one course.

Section - Instructor

- Relationship: Taught By
- Type: Many-to-One
- Explanation:
    - Each Section is taught by one Instructor, but an instructor may teach multiple sections. This relationship allows tracking which instructor is responsible for each section.
    - The instructorID in Section is a foreign key referencing the Instructor entity, ensuring that each section has an assigned instructor.
- Cardinality: Many-to-One, as multiple sections can be taught by the same instructor, but each section has only one instructor.

Section - Semester

- Relationship: Offered In
- Type: Many-to-One
- Explanation:
    - Each Section is offered during a specific Semester (e.g., Fall 2024). This allows for tracking when each section of a course is available.
    - The semesterID in Section is a foreign key referencing the Semester entity, linking each section to the semester it's being taught.
- Cardinality: Many-to-One, as each semester can offer multiple sections, but each section is tied to one semester.

Section - Evaluation (via Evaluation join table)

- Relationship: evaluated_in
- Type: Many-to-Many (contextualized by degree, course, and goal)
- Explanation:
  - Each Section is evaluated based on the goals associated with the degree(s) it supports. The Evaluation table captures performance data for each goal within the context of a specific Degree and Course Section.
  - This relationship enables tracking of how students in each section perform on specific goals for each degree the course serves. It is essential for assessing whether the section meets the degree-specific goals.
  - Evaluation records aggregated data such as the type of assessment (e.g., Homework, Project) and the number of students achieving each grade level (A, B, C, F) for each goal. An optional improvementNote field allows instructors to provide feedback on potential improvements related to that goal.
- Cardinality: Many-to-Many, as each section may have multiple evaluations (one for each degree's goal), and each goal within a degree can be evaluated in multiple sections of a course.

Attributes in Evaluation:

- evaluationType (e.g., Homework, Project, Exam): Specifies the type of assessment.
- gradeCountA, gradeCountB, gradeCountC, gradeCountF: Tracks the number of students receiving each grade for the goal.
- improvementNote: An optional note for improvement suggestions related to that goal.

Semester - Course (indirectly through Section)

- Relationship: Offered In Semester
- Type: Many-to-Many (indirect)
- Explanation:
  - This is an indirect many-to-many relationship, where Semester and Course are connected through Section. Each Course can be offered in multiple semesters, and each Semester can offer multiple courses.
  - By using Section as the intermediary, you can track when and where each course is available each semester.

Database Schema

```
-- Table: Degree
CREATE TABLE Degree (
    degreeID INT PRIMARY KEY,
    name VARCHAR(255),
    level VARCHAR(50)
);

-- Table: Course
CREATE TABLE Course (
    courseNumber VARCHAR(50) PRIMARY KEY,
    name VARCHAR(255)
);

-- Table: Semester
CREATE TABLE Semester (
    semesterID INT PRIMARY KEY,
    year INT,
    term VARCHAR(50)
);

-- Table: Goal
CREATE TABLE Goal (
    goalCode VARCHAR(50),
    degreeID INT,
    description TEXT,
    PRIMARY KEY (goalCode, degreeID),
    FOREIGN KEY (degreeID) REFERENCES Degree(degreeID)
);

-- Table: Instructor
CREATE TABLE Instructor (
    instructorID INT PRIMARY KEY,
    name VARCHAR(255)
);

-- Table: Section
CREATE TABLE Section (
    courseNumber VARCHAR(50),
    sectionID INT,
    instructorID INT,
    semesterID INT,
    enrollmentCount INT,
    PRIMARY KEY (courseNumber, sectionID, semesterID),
    FOREIGN KEY (courseNumber,semesterID) REFERENCES
OfferedCourse(courseNumber,semesterID),
    FOREIGN KEY (instructorID) REFERENCES Instructor(instructorID),
    FOREIGN KEY (semesterID) REFERENCES Semester(semesterID)
);

-- Table: Degree_Course (Join table between Degree and Course)
CREATE TABLE Degree_Course (
    degreeID INT,
    courseNumber VARCHAR(50),
    isCore BOOLEAN,
    PRIMARY KEY (degreeID, courseNumber),
    FOREIGN KEY (degreeID) REFERENCES Degree(degreeID),
    FOREIGN KEY (courseNumber) REFERENCES Course(courseNumber)
);

-- Table: Evaluation
CREATE TABLE Evaluation (
    courseNumber VARCHAR(50),
    sectionID INT,
    degreeID INT,
    semesterID VARCHAR(10),
    goalCode VARCHAR(50),
    evaluationType VARCHAR(50),
    gradeCountA INT,
    gradeCountB INT,
    gradeCountC INT,
    gradeCountF INT,
    improvementNote TEXT,
   PRIMARY KEY (courseNumber, sectionID, semesterID, degreeID, goalCode),
    FOREIGN KEY (courseNumber, sectionID, semesterID)
        REFERENCES Section(courseNumber, sectionID, semesterID)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (degreeID, goalCode)
        REFERENCES DegreeGoal(degreeID, goalCode)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (evaluationTypeID)
        REFERENCES EvaluationType(evaluationTypeID)
        ON DELETE RESTRICT
        ON UPDATE CASCADE,
    CONSTRAINT check_grade_counts
        CHECK (gradeCountA >= 0 AND gradeCountB >= 0 AND gradeCountC >= 0 AND
gradeCountF >= 0)
);
```

When needing to create virtual environment again:

- conda deactivate
- python3 -m venv venv
- source venv/bin/activate
- pip install mysql-connector-python

When wanting to access database and tabs and clear the whole database tables

- mysql -u cs5330 -p
- USE university;
    - SELECT * FROM Degree;
    - SELECT * FROM Course;
    - SELECT * FROM Instructor;
    - SELECT * FROM Goal;
    - SELECT * FROM Semester;
    - SELECT * FROM Section;
    - SELECT * FROM Course_Degree;
    - SELECT * FROM Evaluation;
- To delete just replace the name with what table you want
    - DELETE FROM Degree;
- To Delete the Database to start over or start up sql again
    - mysql -u root -p
    - DROP DATABASE university;
    - exit

Questions that could be asked about code:

How does your schema ensure data integrity?

- Answer: Data integrity is ensured using primary keys for unique identification, foreign keys for maintaining relationships, and constraints like `NOT NULL` and `UNIQUE`
-

What relationships exist between your tables?

- **Answer:** Relationships include:
  - **Degree** and **Course_Degree**: One-to-many.
  - **Course** and **Section**: One-to-many.
  - **Section** and **Instructor**: Many-to-one.

What is the purpose of the `Course_Degree` table?

- **Answer:** It acts as a join table to associate courses with degrees and specify if a course is core or elective for that degree.

How do you handle duplicate entries?

- **Answer:** Constraints like `PRIMARY KEY` and `UNIQUE` are applied to prevent duplicate entries, and checks are performed before insertion.

Why do you have separate tables for courses and sections?

- **Answer:** Courses store general course information, while sections represent specific offerings of a course in a particular semester with instructor and enrollment details.

How does your schema handle prerequisites?

- **Answer:** Relationships and validation checks in functions ensure that prerequisites like valid IDs or existing associations are satisfied before insertion.

What data types are used in your schema and why?

- **Answer:**
  - **VARCHAR** for textual data like names.
  - **INT** for numeric IDs.
  - **ENUM** for predefined values like terms (e.g., `Spring`, `Fall`).

Explanation of our code:

Our code prevents duplicates and ensures uniqueness by defining primary keys and unique constraints in your schema. For example:

1. Primary Keys:
   - Each table (e.g., `Course`, `Degree`, `Instructor`) has a primary key (like `courseNumber`, `degreeID`, or `instructorID`) which ensures that no two records can have the same value for this column.
   - Example: You can't have two courses with the same `courseNumber` like `CS101`.
2. Unique Constraints:
   - Some tables enforce uniqueness on a combination of columns. For instance, the combination of `degreeID` and `level` in the `Degree` table is unique.
   - Example: A `degreeID` of `101` with a `level` of `BS` cannot exist twice, even if other attributes differ.

Examples of Handle Functions

1. `handle_add_degree()`

- **Purpose**: Facilitates adding a degree.
- **Steps**:
  1. Collects `degree_id`, `degree_name`, and `degree_level` from the GUI.
  2. Calls `add_degree()` from the backend.
  3. Displays success or error messages based on the result.

Key Components

1. **Tabs for Functionality**
   The GUI uses **Notebook widgets** to organize functionality into tabs like:

Input Validation
The GUI ensures input validation through:

- `messagebox.showerror` for displaying error messages.
- Direct validation logic in backend functions (e.g., regex checks for formatting).

Buttons to Trigger Actions
Buttons in each tab call respective handler functions, such as:

- `handle_add_degree` for adding degrees.
- `handle_add_course` for adding courses.

Code for Demo:

In the terminal:

Delete the Database and start over before the demo:

1. mysql -u cs5330 -p
2. DROP DATABASE university;
3. exit;
4. python Database.py

In the GUI:

Add Degree Tab:

1. Degree 1:
    a. Degree ID: D1001
    b. Name: Computer Science
    c. Level: BS
2. Degree 2:
    a. Degree ID: D2001
    b. Name: Mathematics
    c. Level: BS

Add Course Tab:

1. Course 1:
    a. Course Number: CSCI1000
    b. Course Name: Introduction to Computing
2. Course 2:
    a. Course Number: MATH2000
    b. Course Name: Calculus I

Add Instructor Tab:

1. Instructor 1:
    a. Instructor ID: 00000001
    b. Instructor Name: John Doe
2. Instructor 2:
    a. Instructor ID: 00000002
    b. Instructor Name: Jane Smith

Add Goal Tab:

1. Goal 1:
    a. Goal Code: C001
    b. Degree ID: D1001
    c. Description: Demonstrate foundational computing principles
2. Goal 2:
    a. Goal Code: M001
    b. Degree ID: D2001
    c. Description: Apply fundamental calculus techniques

Add Semester Tab:

1. Semester 1:
    a. Year: 2024
    b. Term: Spring
2. Semester 2:
    a. Year: 2024
    b. Term: Fall

Switch to ryan

Add Section Tab:

1. Section 1:
    a. Course Number: CSCI1000
    b. Section ID: 001
    c. Year: 2024
    d. Term: Spring
    e. Instructor ID: 00000001
    f. Enrollment Count: 30
2. Section 2:
    a. Course Number: MATH2000
    b. Section ID: 001
    c. Year: 2024
    d. Term: Fall
    e. Instructor ID: 00000002
    f. Enrollment Count: 40

Associate Course to Degree Tab:

1. Associate Course to Degree Tab 1:
    a. Course Number: CSCI1000
    b. Degree ID: D1001
    c. Is Core: 1
2. Associate Course to Degree Tab 2:
    a. Course Number: MATH2000
    b. Degree ID: D2001
    c. Is Core: 0

Associate Course with Goal Tab:

1. Associate Course with Goal Tab 1:
    a. Course Number: CSCI1000
    b. Degree ID: D1001
    c. Goal Code: C001
2. Associate Course with Goal Tab 2:
    a. Course Number: MATH2000
    b. Degree ID: D2001
    c. Goal Code: M001

Add Course to Semester Tab:

1. Add Course to Semester Tab 1:
    a. Year: 2024
    b. Term: Spring
    c. Course Number: CSCI1000
    d. Section ID: 002
    e. Instructor ID: 00000001
    f. Enrollment Count: 25
2. Add Course to Semester Tab 2:
    a. Year: 2024
    b. Term: Fall
    c. Course Number: MATH2000
    d. Section ID: 002
    e. Instructor ID: 00000002
    f. Enrollment Count: 35

Queries & Evaluations Tabs:

Enter/Update Evaluations:

1. Semester 1:
    a. Year: 2024
    b. Term: Spring
    c. Instructor: 00000001
    d. Two Sections:
        i. CSCI 1000-001:
            1. DegreeID: D1001
            2. GoalCode: C001
            3. Eval Type: Mid-term
            4. Grade A: 10
            5. Grade B: 10
            6. Grade C: 10
            7. Grade F: 0
            8. Improvement Note: More practical coding assignments
        ii. CSCI 1000-002
            1. DegreeID: D1001
            2. GoalCode: C001
            3. Eval Type: Final Exam
            4. Grade A: 5
            5. Grade B: 15
            6. Grade C: 5
            7. Grade F: 0
            8. Improvement Note: Students need review of data structures
2. Semester 2:
    a. Year: 2024
    b. Term: Fall
    c. Instructor: 00000002
    d. Two Sections:
        i. MATH 2000-001

            1. DegreeID: D2001
            2. GoalCode: M001
            3. Eval Type: Quiz
            4. Grade A: 15
            5. Grade B: 20
            6. Grade C: 5
            7. Grade F: 0
            8. Improvement Note: Needs more examples on series expansions
        ii. MATH 2000-002
            1. DegreeID: D2001
            2. GoalCode: M001
            3. Eval Type: Homework
            4. Grade A: 10
            5. Grade B: 15
            6. Grade C: 10
            7. Grade F: 0
            8. Improvement Note: Good progress overall

Evaluation Status by Semester:

1. Evaluation Status by Semester 1:
    a. Year: 2024
    b. Term: Spring
2. Evaluation Status by Semester 2:
    a. Year: 2024
    b. Term: Fall

Sections Above Percentage:

1. Sections Above Percentage 1:
    a. Year: 2024
    b. Term: Spring
    c. Percentage: 50
2. Sections Above Percentage 2:
    a. Year: 2024
    b. Term: Fall
    c. Percentage: 50

Additional Queries:

1. Additional Queries 1:
    a. Degree ID: D1001
        i. Expect to see CSCI1000 listed as a core course.
2. Additional Queries 2:
    a. Degree ID: D2001
        i. Expect to see MATH2000 listed as a NOT core course.

Show Improvement Note:

1. Show Improvement Note 1:

a. Course Number: CSCI1000
b. Section ID: 001
c. Section ID: 002

2. Show Improvement Note 2:
   a. Course Number: MATH2000
   b. Section ID: 001
   c. Section ID: 002

Back in the terminal:

Show the tables of the Database:

1. mysql -u cs5330 -p
   a. pw5330
2. USE university;
3. SHOW TABLES;
4. **(show the content of each table)**
   a. SELECT * FROM Degree;
   b. SELECT * FROM Course;
   c. SELECT * FROM Instructor;
   d. SELECT * FROM Goal;
   e. SELECT * FROM Semester;
   f. SELECT * FROM Section;
   g. SELECT * FROM Course_Degree;
   h. SELECT * FROM Evaluation