AMERICAN UNIVERSITY OF BEIRUT

EECE 433: Database Systems

Project Phase 3 – SQL Queries

UHO University Registration System

Name: Hadi El Nawfal

ID: 202200853

Name: Omar Ramadan

ID: 202204622

Course Number: EECE 433

Course Name: Database Systems

Semester: Spring 2024-2025

This course is a practical introduction to concepts of Database design and Database system implementation: SQL, Data Modeling, Database Storage and Indexing, Database Design Theory, Database Application Development, Normalization, an introduction to NoSQL, and an introduction to Data Warehouses.

Date of Submission: Sunday April 27th, 2025

Contents

Table of Figures 4
Chapter 1 Project Proposal Requirements 6
System Overview 6
Justification 6
Preliminary Requirements 7
Project Objectives 7
Methodology 9
Chapter 2 Requirements Analysis and Conceptual Design 10
Entities 10
Relationships 11
Granular Entities 13
Granular Relationships 18
Chapter 3 Relational Schema 23
Comprehensive Relational Schema 23
Granular Entities and Tables 24
Granular One-to-One Relationships 29
Granular One-to-Many Relationships 29
Granular Many-to-Many Relationships 31
Granular Unary Relationships 34
Chapter 4 SQL Queries 36
Table Creations 36
Data Definition Language (DDL) 37
Testing Queries: 45
Data Insertion: 47
Testing Queries: 53
Indexing Strategy 59
Foreign Key Indexes 61
Performance-Oriented Indexes 61
Validation of Index Creation 62
Testing Queries 62
Views, Stored Procedures and Triggers 66
Testing Queries: 70
Advanced Topics 73
Idempotent Script Design: 86
Logical Organization of Database Objects 86

Conclusion 87
Appendix.....89

Table of Figures

Figure 1 – Full ER Diagram of UHO registration system 12
Figure 2 – Entity Person and its corresponding attributes 13
Figure 3 – Entity Student and its corresponding additional attributes. 13
Figure 4 – Entity Professor and its corresponding attributes. 13
Figure 5 – Entity Person as a supertype for entities Student and Professor. 14
Figure 6 – Entity Department and its corresponding attributes. 14
Figure 7 – Entity Faculty and its corresponding attributes. 15
Figure 8 – Entity Campus and its corresponding attributes. 15
Figure 9 – Entity Building and its corresponding attributes. 15
Figure 10 – Entity Course and its corresponding attributes. 15
Figure 11 – Entity Club and its corresponding attributes. 16
Figure 12 – Entity Scholarship and its corresponding attributes. 16
Figure 13 – Entity Transcript and its corresponding attributes. 16
Figure 14 – Weak entity Enrollment and its corresponding attributes. 16
Figure 15 – Weak entity Exam and its corresponding attributes. 17
Figure 16 – Weak entity Room and its corresponding attributes 17
Figure 17 – M:N relationship between Student and Course through the weak Enrollment entity 18
Figure 18 – 1:1 relationship between Transcript Student 18
Figure 19 – M:N relationship between Club and Student 18
Figure 20 – M:N relationship between Scholarship and Student 19
Figure 21 – unary relationship where a student mentors other students 19
Figure 22 – M:N relationship between Professor and Course 19
Figure 23 – unary relationship where a Professor supervises other Professors 20
Figure 24 – 1:M identifying relationship between Course and Exam 20
Figure 25 – 1:M relationship between Course and Department 20
Figure 26 – unary relationship where a Course is a prerequisite of other courses 21
Figure 27 – 1:M relationship between Faculty and Department 21
Figure 28 – 1:1 relationship between Faculty and Building 21
Figure 29 – 1:M relationship between Campus and Building 22
Figure 30 – 1:M identifying relationship between Room and Building 22
Figure 31 – Full Relational Schema 23
Figure 32 – ERD schema for entity Person with its corresponding relational table. 24
Figure 33 – ERD schema for entity Student with its corresponding relational table. 24
Figure 34 – ERD schema for entity Professor with its corresponding relational table. 25
Figure 35 – ERD schema for entity Person as a supertype for entities Student and Professor with the correponding relational tables and the relationships occurring in between 25
Figure 36 – ERD schema for entity Department with its corresponding relational table. 26
Figure 37 – ERD schema for entity Faculty with its corresponding relational table. 26
Figure 38 – ERD schema for entity Campus with its corresponding relational table. 26
Figure 39 – ERD schema for entity Building with its corresponding relational table. 27
Figure 40 – ERD schema for entity Course with its corresponding relational table. 27
Figure 41 – ERD schema for entity Club with its corresponding relational table. 27
Figure 42 – ERD schema for entity Scholarship with its corresponding relational table. 27
Figure 43 – ERD schema for entity Transcript with its corresponding relational table. 28
Figure 44 – ERD schema for weak entity Enrollment with its corresponding relational table. 28

Figure 45 – ERD schema for weak entity Exam with its corresponding relational table. 28
Figure 46 – ERD schema for weak entity Room with its corresponding relational table.....29
Figure 47 – ERD schema for 1:1 relationship between entities Transcript and Student with the corresponding relational tables involved. 29
Figure 49 – ERD schema for 1:M relationship between entities Course and Exam with the corresponding relational tables involved. 29
Figure 50 – ERD schema for 1:M relationship between entities Course and Department with the corresponding relational tables involved. 30
Figure 52 – ERD schema for 1:M relationship between entities Campus and Building with the corresponding relational tables involved. 30
Figure 53 – ERD schema for 1:M relationship between entities Faculty and Department with the corresponding relational tables involved. 31
Figure 54 – ERD schema for M:N relationship between Student and Course through the weak Enrollment entity with the corresponding relational tables involved. 31
Figure 55 – ERD schema for M:N relationship between entities Club and Student with the corresponding relational tables involved. 32
Figure 57 – ERD schema for M:N relationship between entities Professor and Course with the corresponding relational tables involved. 33
Figure 58 – ERD schema for unary relationship between entity Student and itself with the corresponding relational tables involved. 34
Figure 59 – ERD schema for unary relationship between entity Professor and itself with the corresponding relational tables involved. 34
Figure 60 – ERD schema for unary relationship between entity Course and itself with the corresponding relational tables involved……..33

Chapter 1 Project Proposal Requirements

System Overview

UHO University Registration System is an easy-to-use online platform that helps students sign up for classes and manage their schedules. It lets students quickly register for courses and see when and where they take place. Professors use the system to update course details and monitor student progress. The system scope includes storing important information such as student records, course locations, and exam dates. It links together departments, buildings, clubs, and other university resources. This makes the whole registration process smoother and reduces mistakes. Overall, it supports a well-organized academic life at UHO.

Justification

We have selected the UHO University Registration System as an application for the Database Systems course because it aligns with both our current academic learning objectives as current students and its relevance in real word applicability. This system offers a multifaceted environment that enables us to a wide array of database concepts while engaging with a domain we are familiar with.

To further elaborate, the UHO University Registration System is relevant to our course's concepts by embodying the core principles of database design that we have covered during the semester: it involves not only entity-relationship modeling, normalization, the use of derived and multivalued attributes but also uses the management of complex relationships such as many-to-many and one-to-one mappings, while also necessitating the implementation of constraints, cardinalities, and integrity rules, which are critical topics in database theory. In addition, the real-world applicability of this project topic stems from academic institutions relying on robust registration systems to manage student enrollment, course offerings, and other academic records, and thus designing such a system simulates the challenges faced by database administrators in the education sector. Moreover, the system presents notable design challenges in terms of complexity and scope, such as handling course prerequisites (unary relationships), managing course capacities (cardinality constraints), and ensuring accurate GPA calculation (derived attributes), while also requiring careful modeling of weak entities (e.g. enrollment records) and the handling of multivalued attributes (e.g. student contact information, course material, etc.). All of these factors push us to think critically and apply the course material in an efficient manner.

Finally, as current students ourselves, we interact with university registration systems regularly and understand the critical role they play in our academic journey, and this direct experience has given us valuable insight into both the strengths and weaknesses of such systems, fueling our aspiration to design a solution that is practical, intuitive and user-friendly, and thus motivating us to deliver a well-crafted database design of a system we heavily rely on.

Preliminary Requirements

The UHO University Registration System must adhere to a variety of business logic rules and constraints to ensure smooth academic operations and data consistency. While the full set of requirements is most likely more comprehensive than the following rules, these are basic rules that should be respected in any variant implementation of a university registration system:

- Student Enrollment Rules: students can register for multiple courses each semester but must not exceed a maximum credit limit (e.g. 18 credits). Course registration is only allowed during official enrollment periods, and students cannot enroll in courses for which they have not completed the prerequisites.
- Course Capacity Constraints: Each course has a defined maximum capacity. Once this is reached, no additional students can enroll in the course. A minimum number of enrolled students may be required for a course to remain active (e.g. at least 20% of class capacity).
- Instructor Assignment Rules: Instructors may teach multiple courses, but their teaching load per semester is capped (e.g. a maximum of 4 courses). An instructor cannot be assigned to two courses that are scheduled at overlapping times.
- Class Scheduling Constraints: Courses must be scheduled in available classrooms without time conflicts. Each classroom has a fixed capacity and must accommodate the enrolled students (i.e. the classroom's capacity must be greater than or equal to that of the scheduled course). Time slots are pre-defined and cannot overlap for the same instructor or classroom.
- Grade and GPA Calculation: Student letter grades must be recorded for each completed course, which are then mapped to a specific grade value over 4.3 (i.e. A+ is 4.3, A is 4.0, A- is 3.7, etc.). GPA is a derived attribute calculated based on the weighted average of grades and credit hours. A pass/fail grading metric is disregarded from the GPA calculation.
- Advisor Assignment Rules: Each student is assigned to one academic advisor, but an advisor may supervise multiple students.
- Data Integrity Constraints: Unique identifiers are required for students (StudentID), instructors (InstructorID), and courses (CourseID), and all mandatory fields (e.g. CourseName, StudentName, etc.) must be provided during data entry.
- Prerequisite Enforcement: Students must have successfully completed all prerequisite courses before enrolling in advanced courses. A course can have multiple prerequisites, and these must be checked automatically during registration.

Project Objectives

The primary objective of this project is to design and develop a robust, efficient, and scalable database system that accurately models the core processes involved in a typical university registration system. Hence, through this project, we aim to achieve the following chronological goals:

- Accurate Data Representation: Develop a comprehensive Entity-Relationship (ER) diagram that represents all relevant entities (e.g., students, courses, instructors, etc.) and their relationships, while also ensuring data consistency and integrity. We also aim to provide a granular representation by depicting each entity alone with its attributes, as well as each relationship alone with its corresponding entities through separate and smaller-scaled ER diagrams.
- Coherent Schematic Visualization: After setting up a complete and logically correct ER diagram, we aim to create the corresponding schema model that represents the tabular representation of the database system with all of its pivotal primary and foreign keys. This would enable us to visually understand the structure of the tables that should be created before populating them with initial records.
- SQL Implementation: Once the schema is complete, we also aim to write the corresponding code that creates the initial empty tables through SQL, and apply a thorough list of INSERT operations as preliminary data entries inside the created tables. At this stage, the database is completely populated while respecting the business logic and relational constraints imposed by the database design.
- Frontend and Backend Application: After that, we aim to develop a framework that communicates with the created PostgreSQL database so that we create an interface that indirectly interacts with the database while imposing the business constraints through frontend notifications and backend functions that consistently checks these requirements.
- Documentation and Presentation: Finally, we will produce thorough documentation of the design process, including detailed ER diagrams, requirement specifications, and design rationale, serving as a foundation for future phases that expand the system created through this project.

We also aim to ensure a set of objectives that should always be ensured during the various phases of this project, such as the following:

- Support for Complex Business Logic: Incorporate and enforce essential business rules such as course capacity limits, prerequisite checks, credit hour restrictions, and instructor teaching loads through appropriate database constraints and relationships.
- Efficient Data Management: Create a normalized database schema that minimizes data redundancy and ensures efficient storage, retrieval, and management of information related to student registrations, class schedules, and academic records.
- Scalability and Flexibility: Design the database structure to be scalable, allowing for easy adaptation to accommodate future system expansions, such as additional departments, new courses, or updated academic policies.
- User-Focused Design: Lay the groundwork for a user-friendly application interface by organizing data logically and making it easily accessible for different user roles, including students, instructors, and administrators.

Methodology

To ensure a comprehensive and effective design of the UHO University Registration System, we will follow a structured approach that encompasses both requirements gathering and database design phases. Our methodology will involve the following key steps:

1. Requirements Gathering:

- Domain Analysis: We will start by analyzing existing university registration systems to understand common features, workflows, and challenges. This includes reviewing academic policies and procedures related to course registration, scheduling, and grading.
- Use Case Development: Define typical use cases and scenarios (e.g., student course enrollment, instructor assignment, schedule creation) to outline the functional requirements of the system.
- Requirement Documentation: Translate gathered insights into a structured list of functional and non-functional requirements, including business rules, constraints, and user interactions.

2. Conceptual Design:

- Entity Identification: Identify all relevant entities (e.g., Student, Course, Instructor) and their attributes based on the documented requirements.
- Relationship Mapping: Define and categorize relationships between entities, specifying cardinalities, optional attributes, and participation constraints as per business logic.
- ER Diagram Creation (Using ERDPlus):
    - Individual ER diagrams for each entity, showing all attributes.
    - Diagrams for each relationship, depicting involved entities and relationship details.
    - A full ER diagram combining all entities and relationships into a unified schema.
- Validation and Refinement: Review the ER diagram for completeness and consistency with requirements. Adjust and refine entity attributes, relationships, and constraints based on feedback and logical validation.
- Documentation: Document the rationale behind design decisions, entity definitions, relationship constraints, and assumptions made during the design process. This will form the basis for subsequent project phases.

Chapter 2 Requirements Analysis and Conceptual Design

The UHO University Registration System keeps track of persons, students, professors, departments, faculties, campuses, buildings, courses, exams, enrollments, clubs, scholarships, and transcripts.

Entities

- For each person, the system will track a unique person ID, as well as the person's first name, optional middle name, and last name. The person's date of birth will also be stored, along with multiple mobile phone numbers. Additionally, the system will keep track of each person's age (derived from date of birth) and full name (derived from first, middle and last name).
- For each student, the system will track a unique student ID, major, multiple student information, and all attributes inherited from the person entity.
- For each professor, the system will track a unique professor ID, rank (e.g., Assistant, Associate, Full), mandatory hire date, and multiple office phone numbers. The system will also track the professor's years of service (derived from hire date) along with all attributes inherited from the person entity.
- For each department, the system will store a unique department ID, department name, and multiple department phone numbers.
- For each faculty, the system will store a unique faculty ID and faculty name.
- For each campus, the system will store a unique campus ID, campus name, and location.
- For each building, the system will store a unique building ID, building name, and optional building location.
- For each course, the system will store a unique course ID, course name, and mandatory credit hours.
- For each club, the system will store a unique club ID, and club name.
- For each scholarship, the system will store a unique scholarship ID, scholarship name, and optional minimum GPA requirement.
- For each transcript, the system will store a unique transcript ID, a creation date, transcript age (derived from creation date), and a cumulative GPA (derived from the student's course grades).
- For each enrollment (which tracks student-course registration), the system will store the semester, along with the enrollment date and optional final grade.
- For each exam, the system will track a partial exam number (e.g., "Exam1," "Exam2"), and the composite key of course ID and exam number will uniquely identify each exam. The system will also store the exam date, optional location, and exam weight.
- For each room, the system will track a partial room number (e.g., "205," "102"), and the composite key of Building ID and room number will uniquely identify each room. The system will also store the number of seats in a room.

Relationships

- Each student can enroll in a minimum of 4 courses and a maximum of 6 courses; each course can have a minimum of 12 students enrolled in and a maximum of 40. The system will keep track of this many-to-many relationship via the enrollment entity.
- Each student has exactly one transcript; each transcript is assigned to exactly one student.
- Each student can join zero or a maximum of 3 clubs; each club needs to have at least 40 members and a maximum of 400 students as members.
- Each student can receive zero to 5 scholarships; each scholarship can be awarded to a minimum of 5 and a maximum of 20 students.
- Each department offers one or more courses; each course is offered by exactly one department.
- Each course is associated with one or more exams; each exam belongs to exactly one course. The partially unique attribute of Exam is identified by the full key of Course.
- Each course can have one or more prerequisites; each prerequisite is associated with one or more courses.
- Each professor can teach zero courses and a maximum of 4 courses; each course is taught by at least 1 professor and a maximum of 3 professors.
- Each professor may supervise zero or more other professors; each professor may be supervised by zero or one professor.
- Each student may mentor zero or more other students; each student may be mentored by zero or one student.
- Each faculty oversees one or more departments; each department belongs to exactly one faculty.
- Each faculty is assigned to exactly one building; each building is assigned to exactly one faculty but doesn’t have to have a faculty.
- Each campus contains one or more buildings; each building is located on exactly one campus.
- Each building has one or more rooms; each room is located in exactly one building. The partially unique attribute of Room is identified by the full key of Building.
- For each instance of professor and course assignment, the system keeps track of the course timing and location.
- For each instance of student obtaining a scholarship, the system keeps track of the percentage of amount granted for the student and the date receiving the scholarship.
- For each instance of student enrollment in a course, the system keeps track of the enrollment semester, enrollment date, and optional final grade.
- For each instance of a student joining a club, the system keeps track of the position of the student within the club and the date joined.

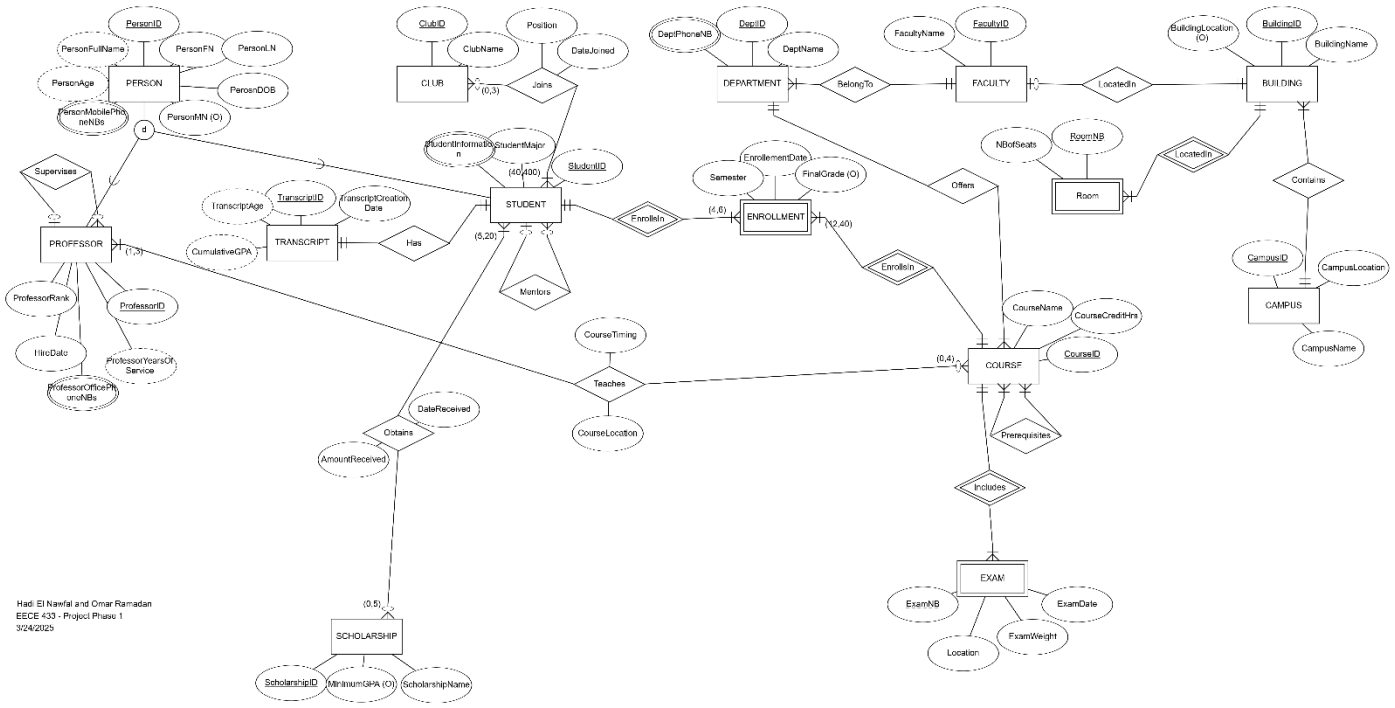

Figure 1 – Full ER Diagram of UHO registration system

Granular Entities

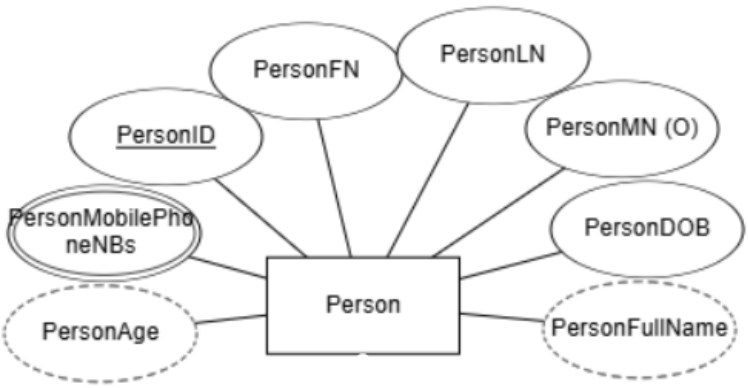

Figure 2 – Entity Person and its corresponding attributes

- For each person, the system will track a unique person ID, as well as the person's first name, optional middle name, and last name. The person's date of birth will also be stored, along with multiple mobile phone numbers. Additionally, the system will keep track of each person's age (derived from date of birth) and full name (derived from first, middle and last name).

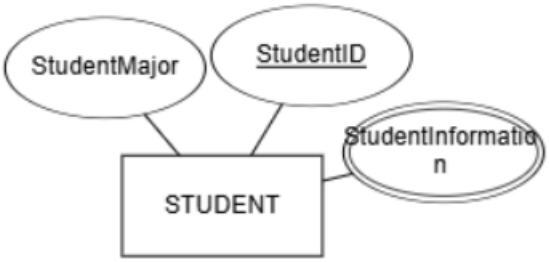

Figure 3 – Entity Student and its corresponding additional attributes.

- For each student, the system will track a unique student ID, major, multiple student information, and all attributes inherited from the person entity.

Figure 4 – Entity Professor and its corresponding attributes.

- For each professor, the system will track a unique professor ID, rank (e.g., Assistant, Associate, Full), mandatory hire date, and multiple office phone numbers. The system will also track the professor's years of service (derived from hire date) along with all attributes inherited from the person entity.

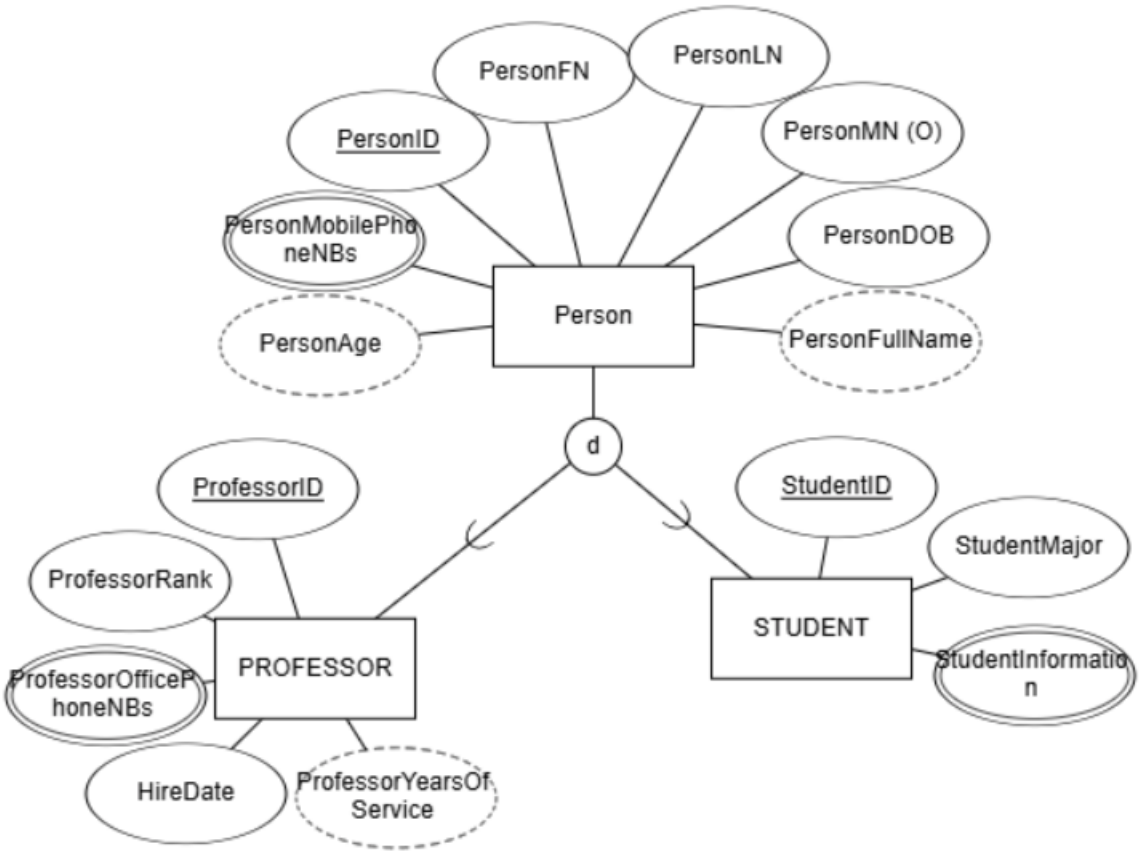

Figure 5 – Entity Person as a supertype for entities Student and Professor.

In our university management system, a person can either be a professor or a student, which are both disjointed subtypes inherited from the person entity and having common attributes such as the person's ID, a derived age, mobile phone numbers, DOB, first, middle and last name, and a derived full name. If the person is a student, they additionally have a student ID, a major and other student information as additional attributes. If the person is a professor, they additionally have a professor ID, a professor rank, one or more professor office phone numbers, a fixed hired date, and a derived years of service attribute.

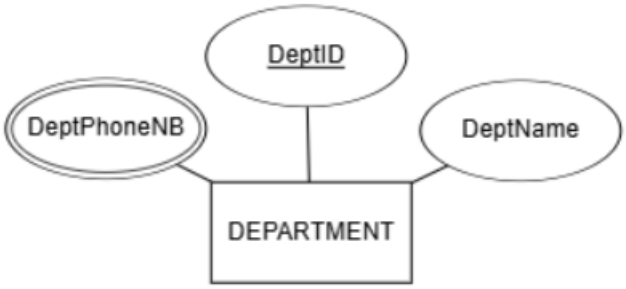

Figure 6 – Entity Department and its corresponding attributes.

- For each department, the system will store a unique department ID, department name, and multiple department phone numbers.

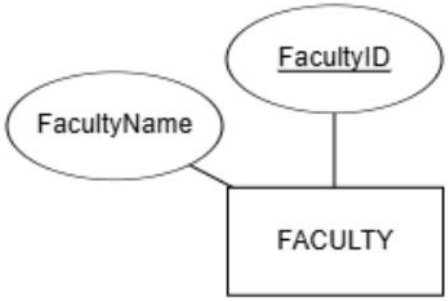

Figure 7 – Entity Faculty and its corresponding attributes.

- For each faculty, the system will store a unique faculty ID and faculty name.

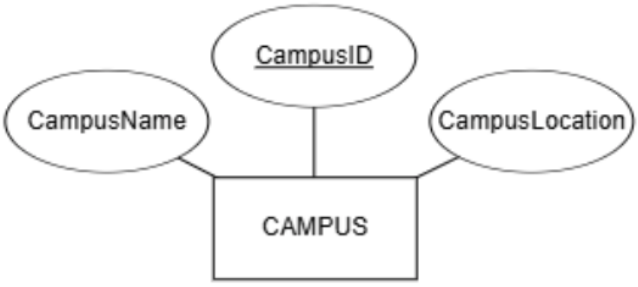

Figure 8 – Entity Campus and its corresponding attributes.

- For each campus, the system will store a unique campus ID, campus name, and location.

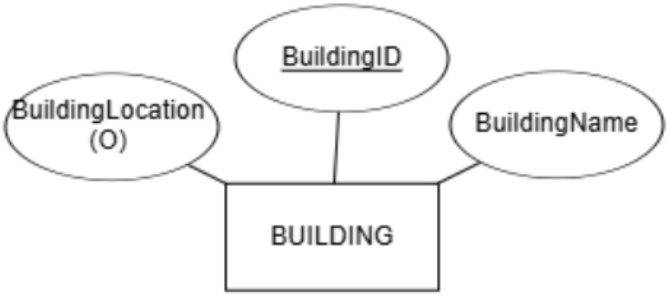

Figure 9 – Entity Building and its corresponding attributes.

- For each building, the system will store a unique building ID, building name, and optional building location.

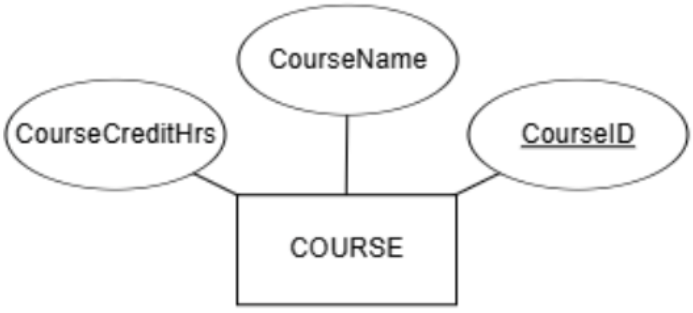

Figure 10 – Entity Course and its corresponding attributes.

- For each course, the system will store a unique course ID, course name, and mandatory credit hours.

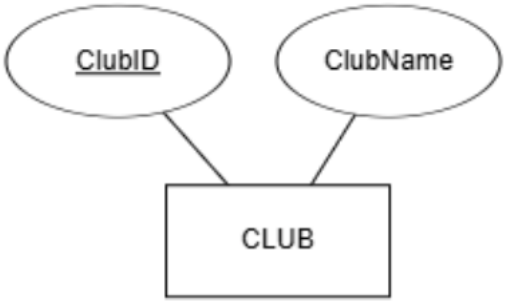

Figure 11 – Entity Club and its corresponding attributes.

- For each club, the system will store a unique club ID, and club name.

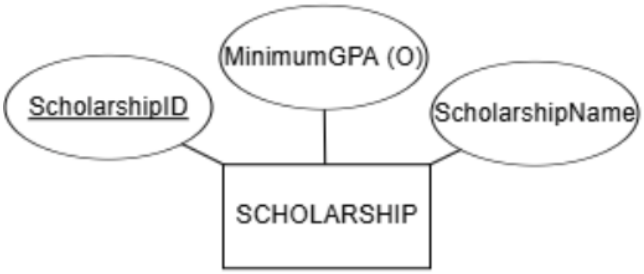

Figure 12 – Entity Scholarship and its corresponding attributes.

- For each scholarship, the system will store a unique scholarship ID, scholarship name, and optional minimum GPA requirement.

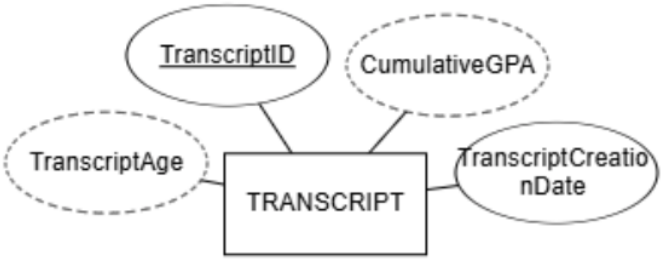

Figure 13 – Entity Transcript and its corresponding attributes.

- For each transcript, the system will store a unique transcript ID, a creation date, transcript age (derived from creation date), and a cumulative GPA (derived from the student's course grades).

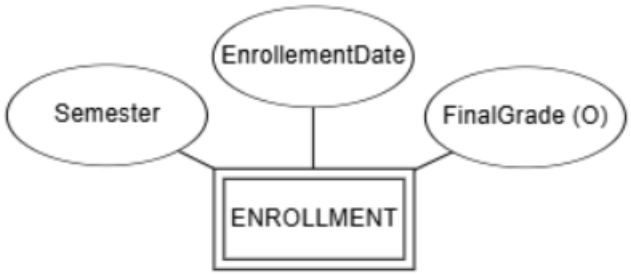

Figure 14 – Weak entity Enrollment and its corresponding attributes.

- For each enrollment (which tracks student-course registration), the system will store the semester, along with the enrollment date and optional final grade.

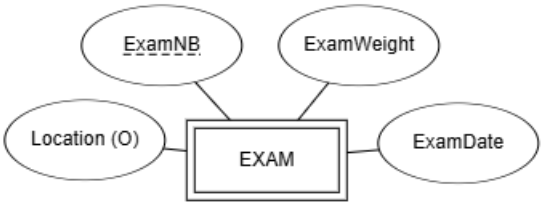

Figure 15 – Weak entity Exam and its corresponding attributes.

- For each exam, the system will track a partial exam number (e.g., “Exam1,” “Exam2”), and the composite key of course ID and exam number will uniquely identify each exam. The system will also store the exam date, optional location, and exam weight.

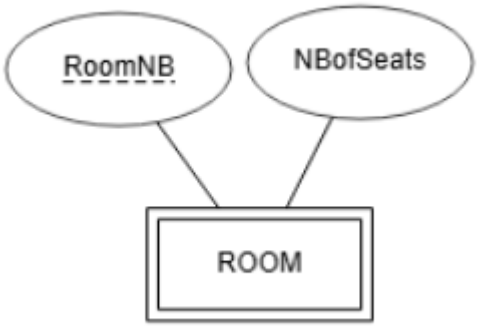

Figure 16 – Weak entity Room and its corresponding attributes

- For each room, the system will track a partial room number (e.g., “205,” “102”), and the composite key of Building ID and room number will uniquely identify each room. The system will also store the number of seats in a room.

Granular Relationships

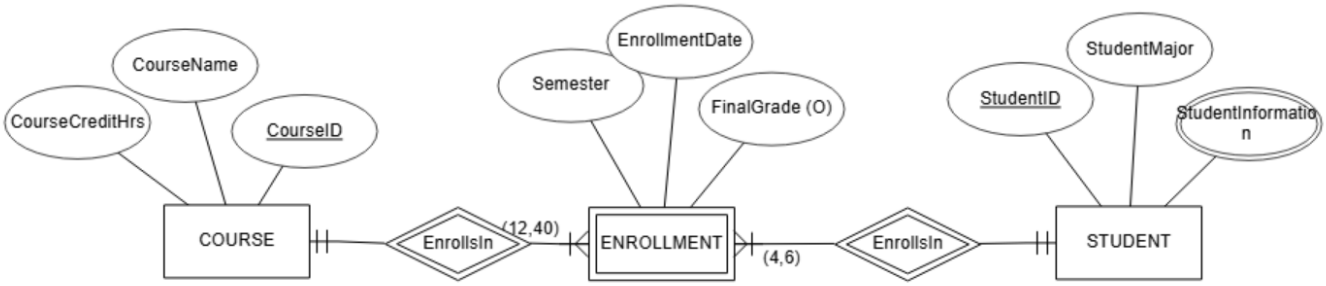

Figure 17 – M:N relationship between Student and Course through the weak Enrollment entity

- Each student can enroll in a minimum of 4 courses and a maximum of 6 courses; each course can have a minimum of 12 students enrolled in and a maximum of 40. The system will keep track of this many-to-many relationship via the enrollment entity.
- For each instance of student enrollment in a course, the system keeps track of the enrollment semester, enrollment date, and optional final grade.

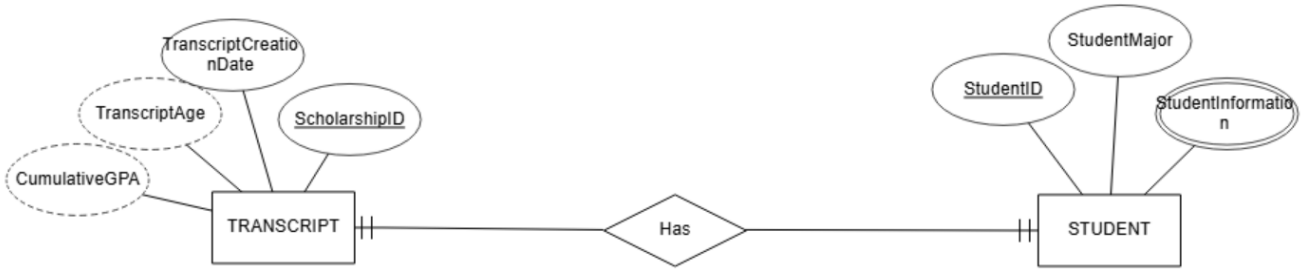

Figure 18 – 1:1 relationship between Transcript Student

- Each student has exactly one transcript; each transcript is assigned to exactly one student.

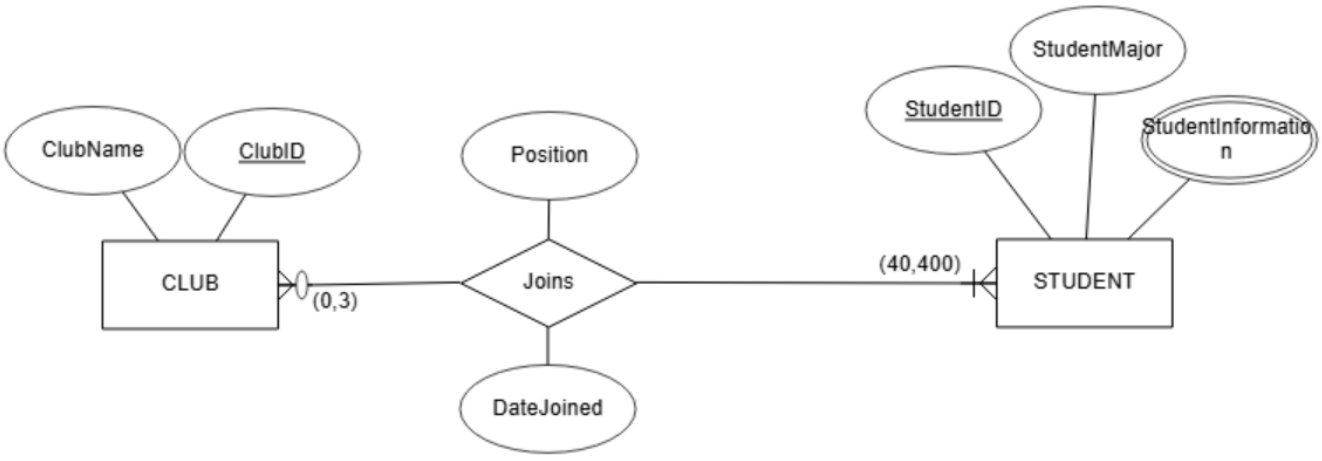

Figure 19 – M:N relationship between Club and Student

- Each student can join zero or a maximum of 3 clubs; each club needs to have at least 40 members and a maximum of 400 students as members.

- For each instance of a student joining a club, the system keeps track of the position of the student within the club and the date joined.

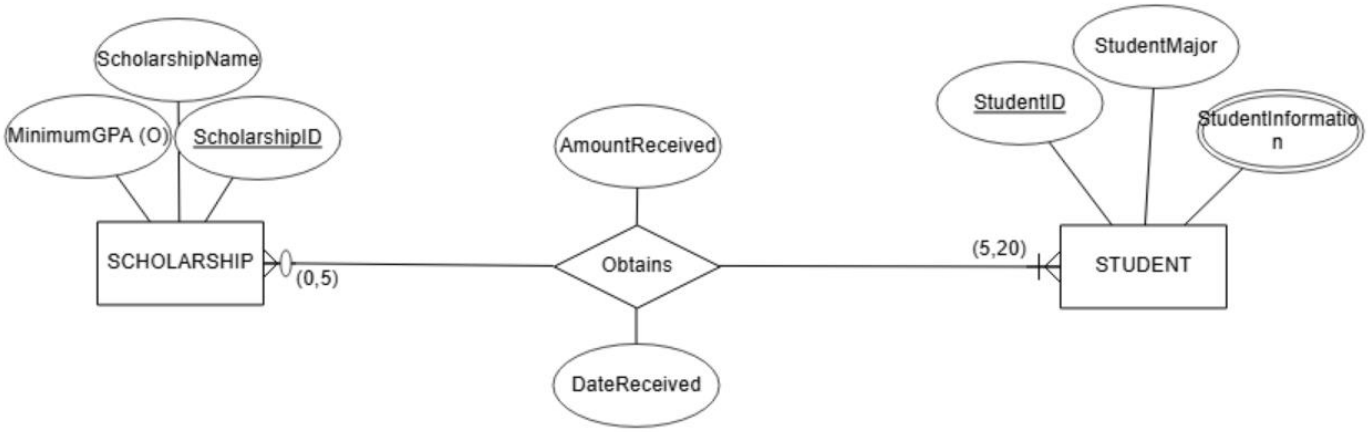

Figure 20 – M:N relationship between Scholarship and Student

- Each student can receive zero to 5 scholarships; each scholarship can be awarded to a minimum of 5 and a maximum of 20 students.
- For each instance of student obtaining a scholarship, the system keeps track of the percentage of amount granted for the student and the date receiving the scholarship.

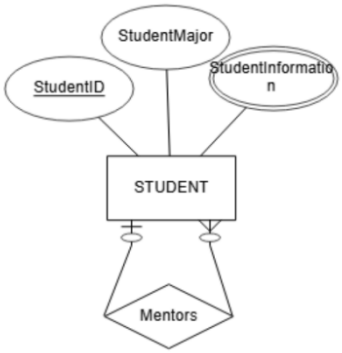

Figure 21 – unary relationship where a student mentors other students

- Each student may mentor zero or more other students; each student may be mentored by zero or one student.

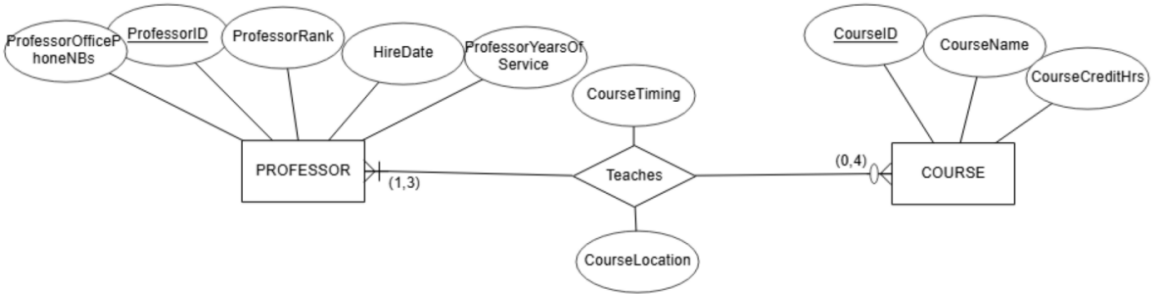

Figure 22 – M:N relationship between Professor and Course

- Each professor can teach zero courses and a maximum of 4 courses; each course is taught by at least 1 professor and a maximum of 3 professors.

- For each instance of professor and course assignment, the system keeps track of the course timing and location.

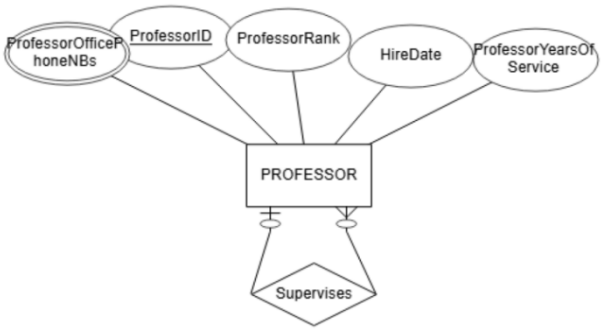

Figure 23 – unary relationship where a Professor supervises other Professors

- Each professor may supervise zero or more other professors; each professor may be supervised by zero or one professor.

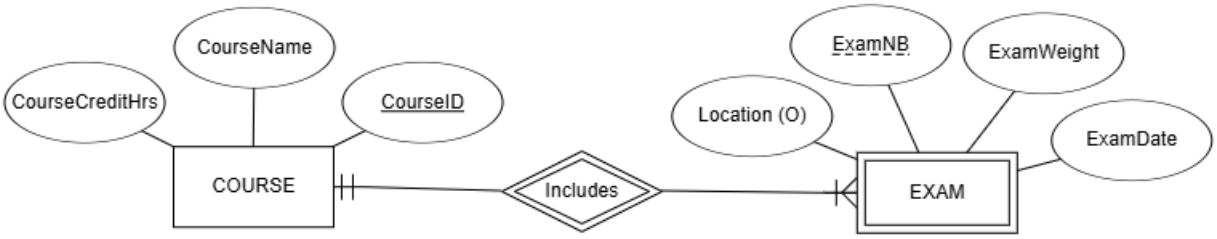

Figure 24 – 1:M identifying relationship between Course and Exam

- Each course is associated with one or more exams; each exam belongs to exactly one course. The partially unique attribute of Exam is identified by the full key of Course.

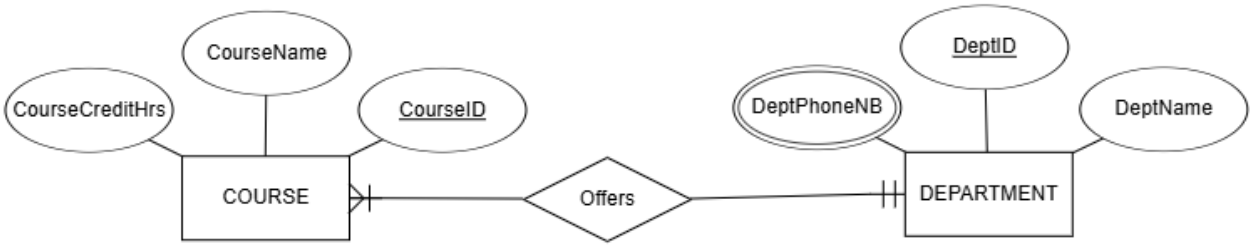

Figure 25 – 1:M relationship between Course and Department

- Each department offers one or more courses; each course is offered by exactly one department.

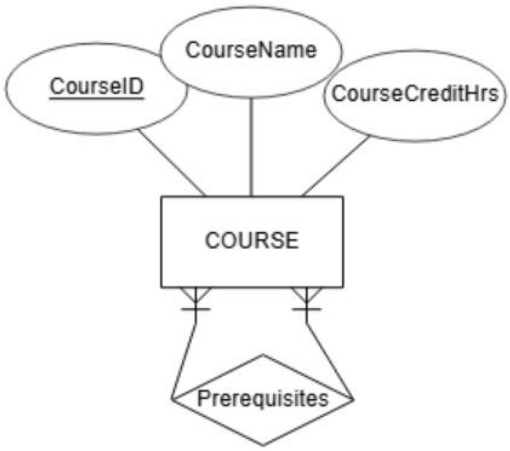

Figure 26 – unary relationship where a Course is a prerequisite of other courses

- Each course can have one or more prerequisites; each prerequisite is associated with one or more courses.

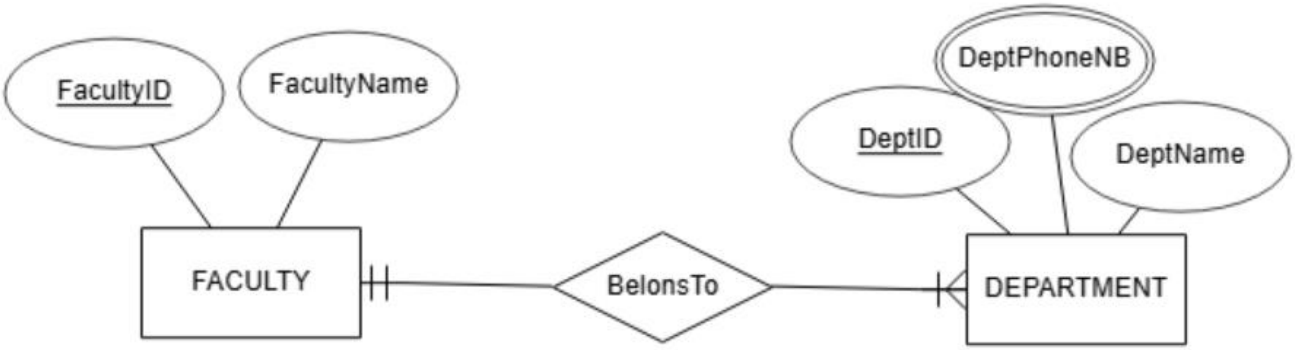

Figure 27 – 1:M relationship between Faculty and Department

- Each faculty oversees one or more departments; each department belongs to exactly one faculty.

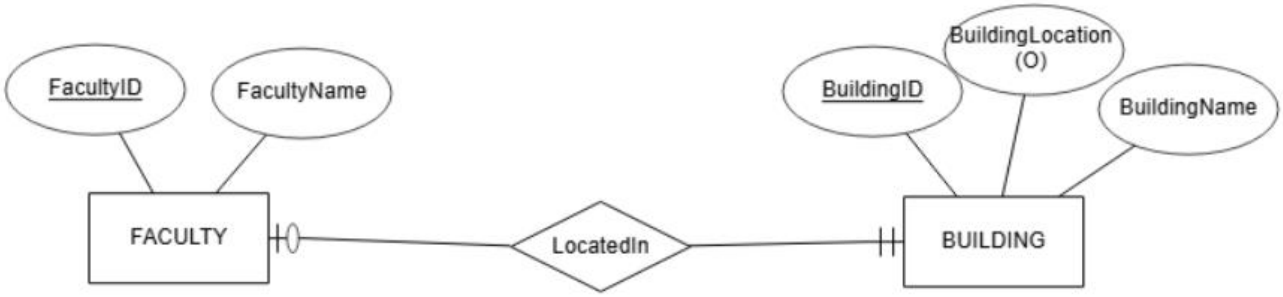

Figure 28 – 1:1 relationship between Faculty and Building

- Each faculty is assigned to exactly one building; each building is assigned to exactly one faculty but doesn't have to have a faculty.

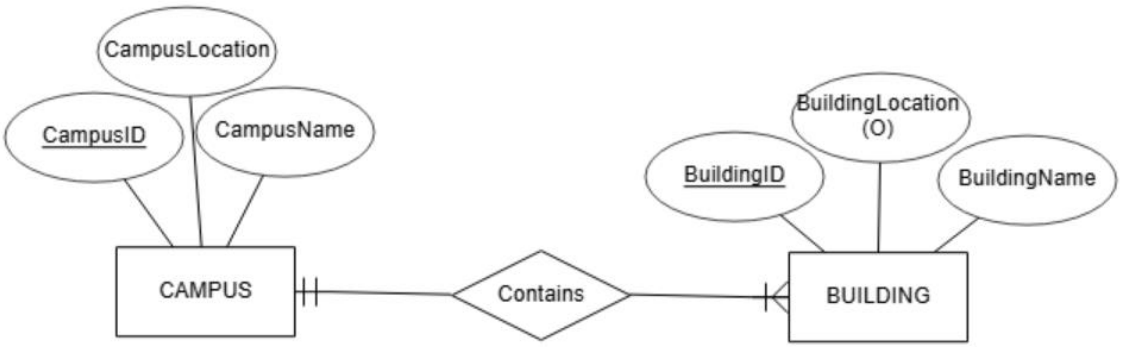

Figure 29 – 1:M relationship between Campus and Building

- Each campus contains one or more buildings; each building is located on exactly one campus.

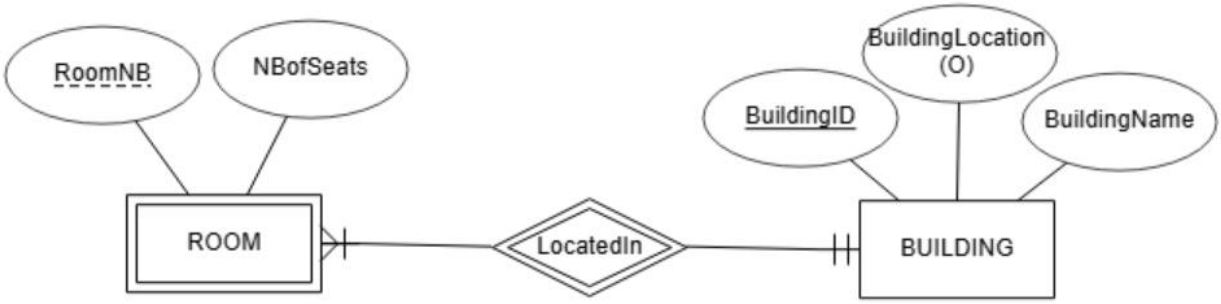

Figure 30 – 1:M identifying relationship between Room and Building

- Each building has one or more rooms; each room is located in exactly one building. The partially unique attribute of Room is identified by the full key of Building.

Chapter 3 Relational Schema

Building upon the conceptual model established in the previous phase, this chapter presents the relational schema design for the UHO University Registration System. The transition from Entity-Relationship (ER) modeling to relational design is a critical step in database development, as it transforms the conceptual structure into a formalized schema that can be implemented in a relational database management system (RDBMS).

The relational schema depicted in this chapter adheres to the constraints and requirements identified in Chapter 2, ensuring that all entities, relationships, attributes, and cardinalities are accurately represented. It follows a structured approach to normalize data, maintain integrity, and optimize efficiency while preserving the complexity of real-world university registration processes. Each entity is translated into its respective relational table, and special considerations are made for multivalued attributes, weak entities, subclass relationships, and many-to-many connections.

To enhance clarity, this chapter presents:

- A comprehensive relational schema that contains all tables in a unified database structure.
- Individual entity tables, each showcasing the entity and its corresponding relational table.
- One-to-One, One-to-Many, and Many-to-many relationship tables, illustrated separately with their participating entities.
- Unary relationships, detailing recursive associations within the system.

This relational schema representation serves as a blueprint for the logical and physical database implementation in subsequent stages, ensuring that the final system efficiently supports UHO's academic operations while adhering to best practices in relational database design.

Comprehensive Relational Schema

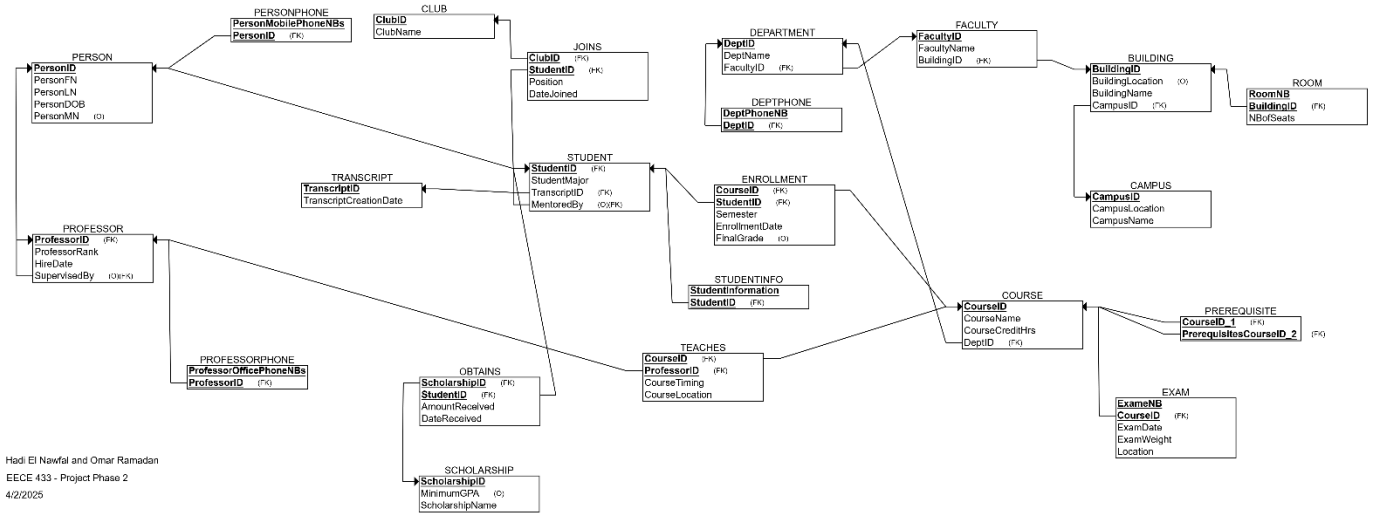

Figure 31 – Full Relational Schema

Granular Entities and Tables

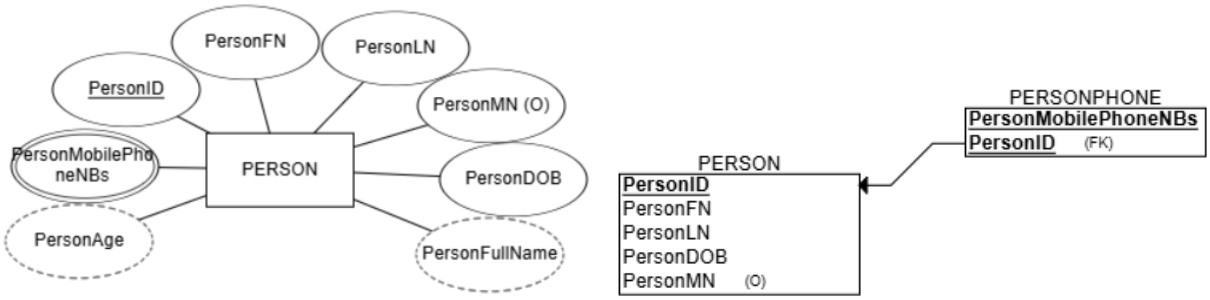

Figure 32 – ERD schema for entity Person with its corresponding relational table.

- For each person, the system will track a unique person ID, as well as the person's first name, optional middle name, and last name. The person's date of birth will also be stored, along with multiple mobile phone numbers. Additionally, the system will keep track of each person's age (derived from date of birth) and full name (derived from first, middle and last name).
- Correspondingly, all derived attributes (i.e. PersonAge and PersonFullName) are not depicted in the relational table. The PersonMobilePhoneNBs multivalued attribute is mapped as a separate relation PERSONPHONE that has a column PersonMobilePhoneNBs representing the multivalued attribute and a foreign key column PersonID referring to the primary key of the relation PERSON.

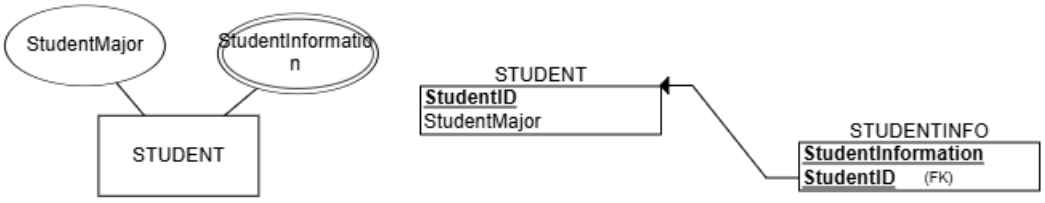

Figure 33 – ERD schema for entity Student with its corresponding relational table.

- For each student, the system will track a unique student ID, major, multiple student information, and all attributes inherited from the person entity.
- Note that because entity Student inherits attributes from entity Person, it will not have its own identifier, but instead, it will inherit the PersonID unique attribute as a foreign key, which explains why we have removed the StudentID as an attribute compared to Chapter 2's granular entity. Also, note that the StudentID in relation STUDENT is a foreign key mapping to PersonID, but this cannot be depicted when only drawing granular entities as standalone tables. We will demonstrate this relationship in the next granular depictions.
- The StudentInformation multivalued attribute is mapped as a separate relation STUDENTINFO that has a column StudentInformation representing the multivalued attribute and a foreign key column StudentID referring to the primary key of the relation STUDENT.

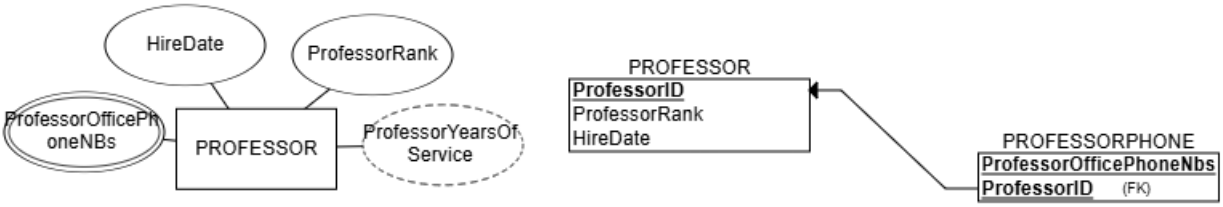

Figure 34 – ERD schema for entity Professor with its corresponding relational table.

- For each professor, the system will track a unique professor ID, rank (e.g., Assistant, Associate, Full), mandatory hire date, and multiple office phone numbers. The system will also track the professor's years of service (derived from hire date) along with all attributes inherited from the person entity.
- Note that because entity Professor inherits attributes from entity Person, it will not have its own identifier, but instead, it will inherit the PersonID unique attribute as a foreign key, which explains why we have removed the StudentID as an attribute compared to Chapter 2's granular entity. Also, note that the ProfessorID in relation PROFESSOR is a foreign key mapping to PersonID, but this cannot be depicted when only drawing granular entities as standalone tables. We will demonstrate this relationship in the next granular depictions.
- The ProfessorOfficePhoneNBs multivalued attribute is mapped as a separate relation PROFESSORPHONE that has a column ProfessorOfficePhoneNbs representing the multivalued attribute and a foreign key column ProfessorID referring to the primary key of the relation PROFESSOR.

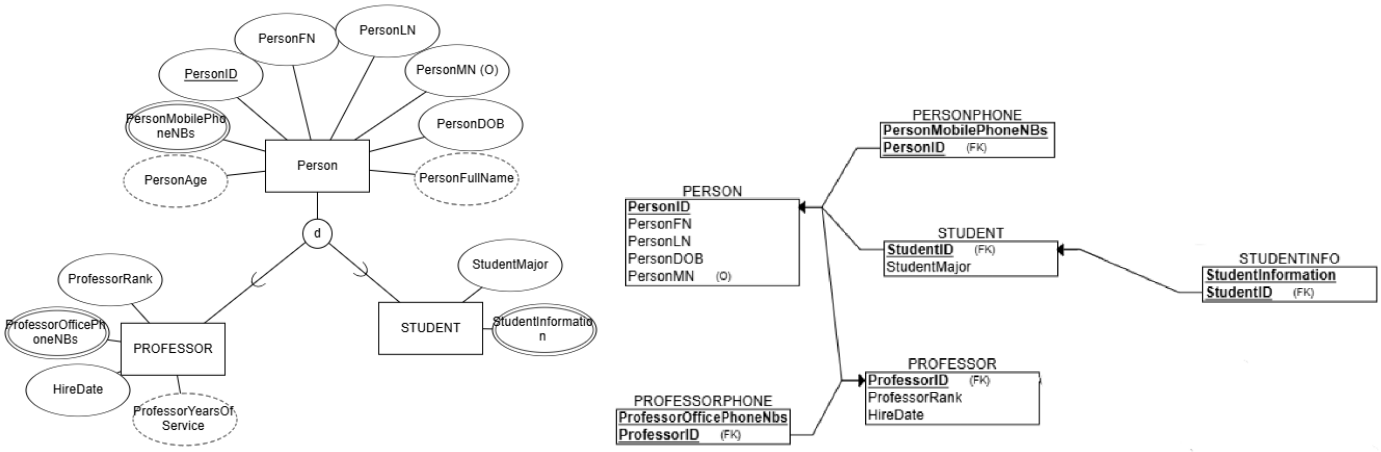

Figure 35 – ERD schema for entity Person as a supertype for entities Student and Professor with the correponding relational tables and the relationships occurring in between

- In our university management system, a person can either be a professor or a student, which are both disjointed subtypes inherited from the person entity and having common attributes such as the person's ID, a derived age, mobile phone numbers, DOB, first, middle and last name, and a derived full name. If the person is a student, they inherit the PersonID as a foreign key labelled StudentID, and additionally have major and other student information as additional attributes. If the person is a professor, they inherit the PersonID as a foreign key labelled ProfessorID, and additionally have a

professor rank, one or more professor office phone numbers, a fixed hired date, and a derived years of service attribute.

- We now see in the above ERD schema that the StudentID and ProfessorID are actually foreign keys mapping to relation Person, as previously elaborated in the previous granular ERD schemas.

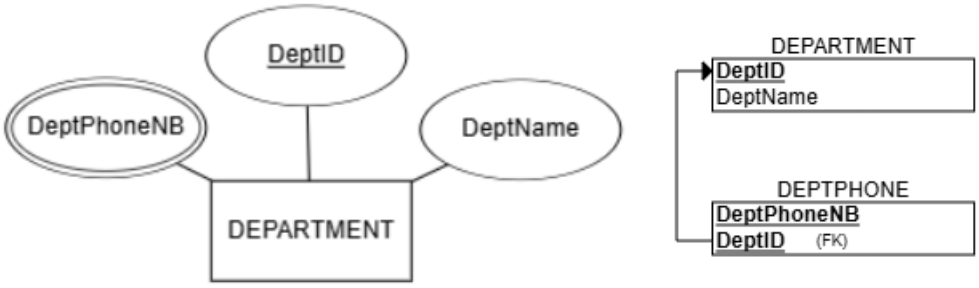

Figure 36 – ERD schema for entity Department with its corresponding relational table.

- For each department, the system will store a unique department ID, department name, and multiple department phone numbers.
- The DeptPhoneNB multivalued attribute is mapped as a separate relation DEPTPHONE that has a column DeptPhoneNB representing the multivalued attribute and a foreign key column DeptID referring to the primary key of the relation DEPARTMENT.

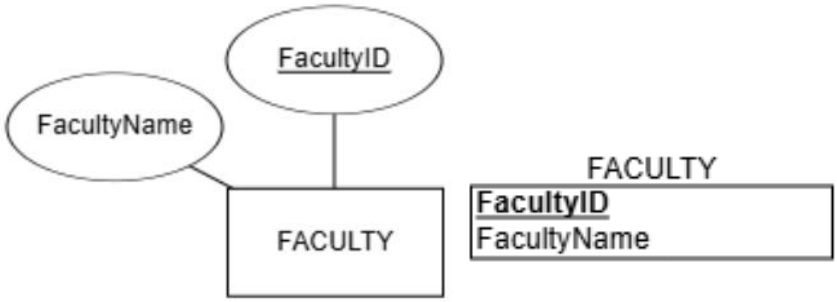

Figure 37 – ERD schema for entity Faculty with its corresponding relational table.

- For each faculty, the system will store a unique faculty ID (primary key) and faculty name.

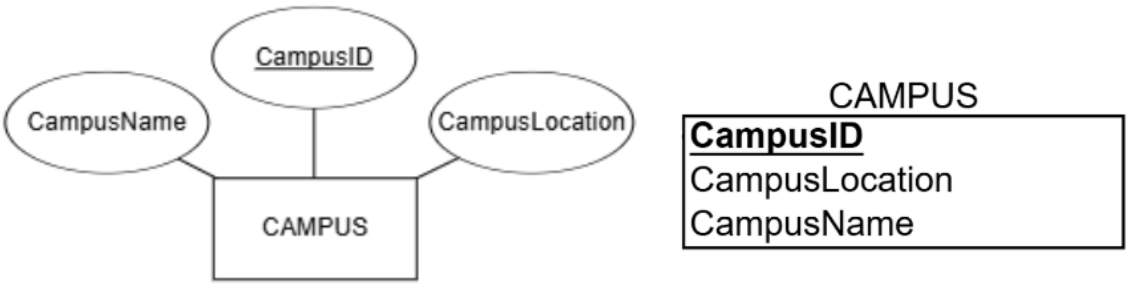

Figure 38 – ERD schema for entity Campus with its corresponding relational table.

- For each campus, the system will store a unique campus ID (primary key), campus name, and location.

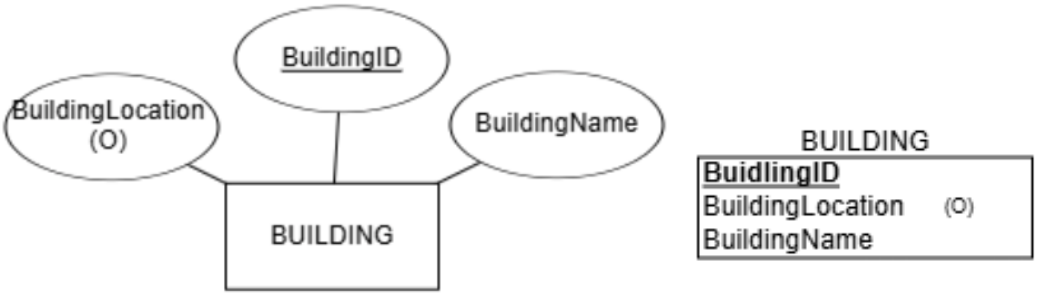

Figure 39 – ERD schema for entity Building with its corresponding relational table.

- For each building, the system will store a unique building ID (primary key), building name, and optional building location.

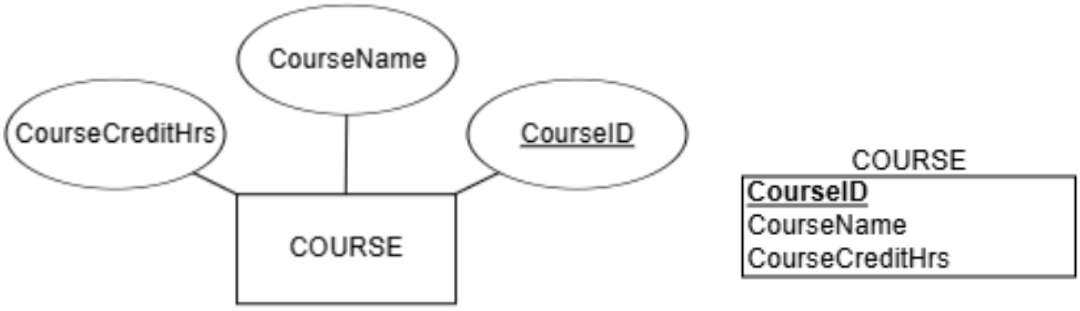

Figure 40 – ERD schema for entity Course with its corresponding relational table.

- For each course, the system will store a unique course ID (primary key), course name, and mandatory credit hours.

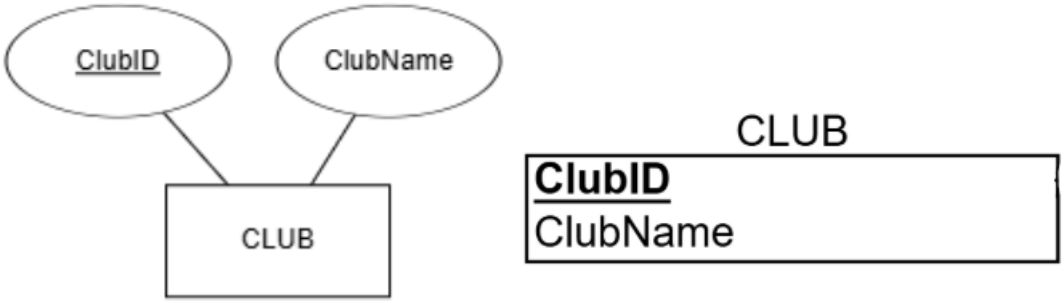

Figure 41 – ERD schema for entity Club with its corresponding relational table.

- For each club, the system will store a unique club ID (primary key), and club name.

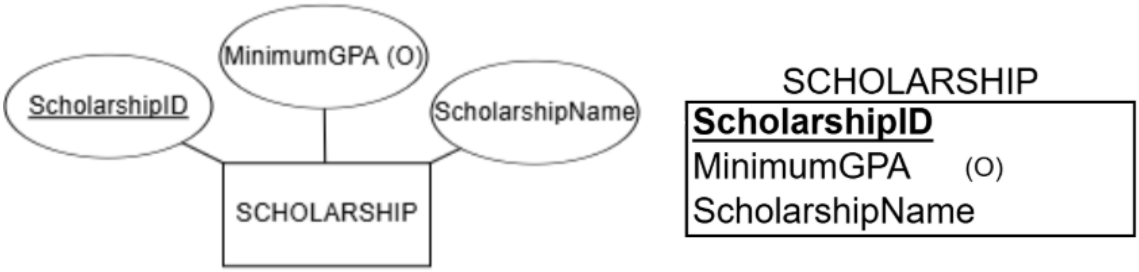

Figure 42 – ERD schema for entity Scholarship with its corresponding relational table.

- For each scholarship, the system will store a unique scholarship ID (primary key), scholarship name, and optional minimum GPA requirement.

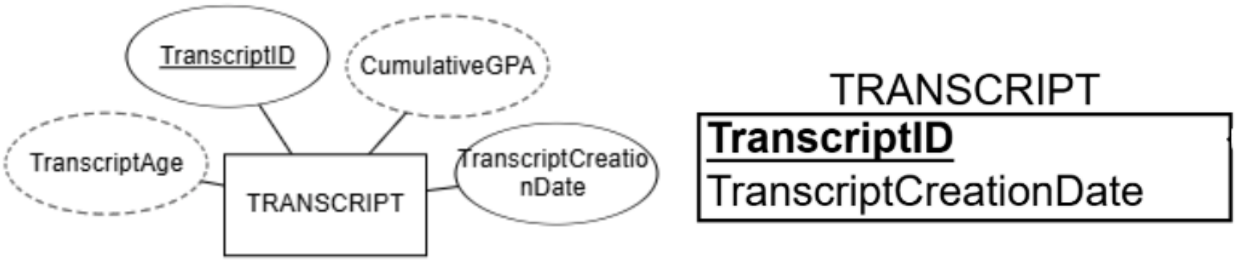

Figure 43 – ERD schema for entity Transcript with its corresponding relational table.

- For each transcript, the system will store a unique transcript ID, a creation date, transcript age (derived from creation date), and a cumulative GPA (derived from the student's course grades).
- Correspondingly, all derived attributes (i.e. TranscriptAge and CumulativeGPa) are not depicted in the relational table.

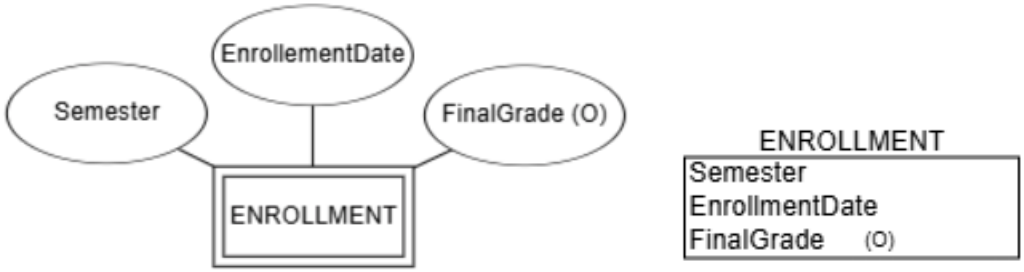

Figure 44 – ERD schema for weak entity Enrollment with its corresponding relational table.

- For each enrollment (which tracks student-course registration), the system will store the semester, along with the enrollment date and optional final grade.
- Note that because entity Enrollment is a weak entity with no partial key, it will not have any unique identifiers by itself, but when related to entities Student and Course, its composite primary key will consist of foreign keys CourseID and StudentID, and this will be shown later sections

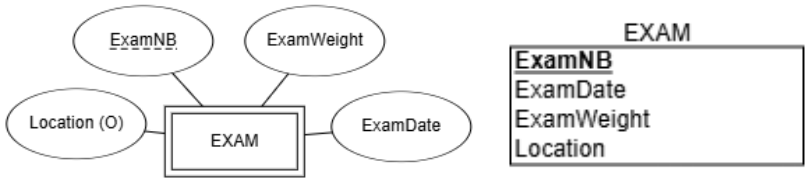

Figure 45 – ERD schema for weak entity Exam with its corresponding relational table.

- For each exam, the system will track a partial exam number (e.g., "Exam1," "Exam2"), and the composite key of course ID and exam number will uniquely identify each exam. The system will also store the exam date, optional location, and exam weight.
- Note that because entity Enrollment is a weak entity with a partial key ExamNB, it will not have a complete primary key by itself, but when related to entity Course, its composite primary key will consist of foreign keys CourseID and ExamNB, and this will be shown in later sections

Figure 46 – ERD schema for weak entity Room with its corresponding relational table.

- For each room, the system will track a partial room number (e.g., “205,” “102”), and the composite key of Building ID and room number will uniquely identify each room. The system will also store the number of seats in a room.
- Note that because entity Room is a weak entity with a partial key RoomNB, it will not have a complete primary key by itself, but when related to entity Building, its composite primary key will consist of foreign keys BuildingId and RoomNB, and this will be shown in later sections.

Granular One-to-One Relationships

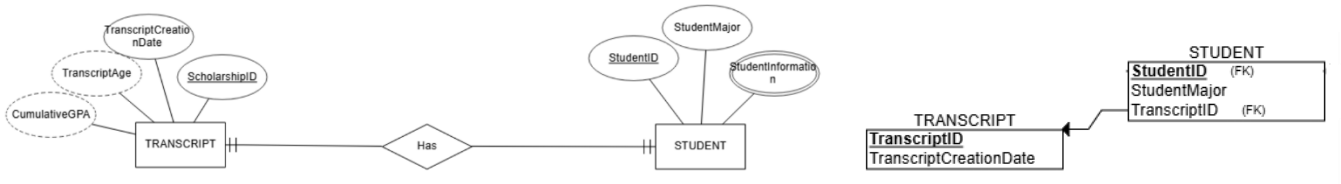

Figure 47 – ERD schema for 1:1 relationship between entities Transcript and Student with the corresponding relational tables involved.

- Each student has exactly one transcript; each transcript is assigned to exactly one student.
- The relation STUDENT has a foreign key TranscriptID pointing to the primary key of the relation TRANSCRIPT.

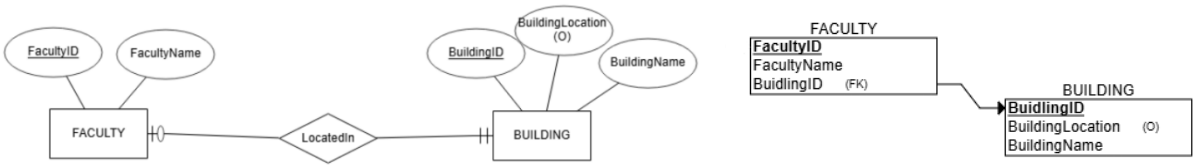

Figure 48 – ERD schema for 1:1 relationship between entities Faculty and Building with the corresponding relational tables involved.

- Each faculty is assigned to exactly one building; each building is assigned to exactly one faculty but doesn’t have to have a faculty.
- The relation FACULTY has a foreign key BuildingID pointing to the primary key of the relation BUILDING.

Granular One-to-Many Relationships

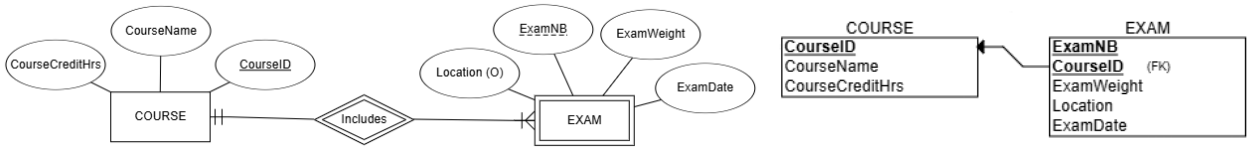

Figure 49 – ERD schema for 1:M relationship between entities Course and Exam with the corresponding relational tables involved.

- Each course is associated with one or more exams; each exam belongs to exactly one course. The partially unique attribute of Exam is identified by the full key of Course.
- The relation EXAM mapped from the entity Exam on the M side of the 1:M relationship has a foreign key CourseID that corresponds to the primary key of the relation COURSE mapped from the 1 side of the 1:M relationship.
- In this case, because we have a weak entity EXAM, the resulting relation has a composite primary key ExamNB and CourseID (as previously mentioned) that is composed of the partial identifier ExamNB and the foreign key CourseId corresponding to the primary key of the owner entity COURSE.

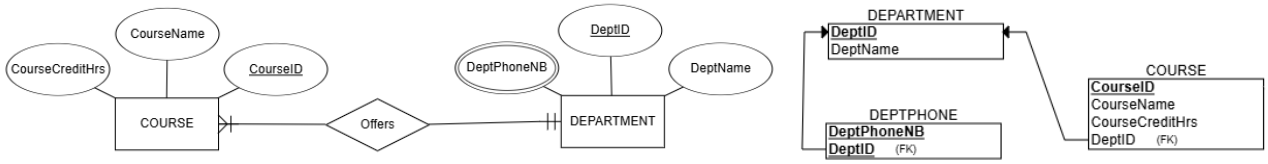

Figure 50 – ERD schema for 1:M relationship between entities Course and Department with the corresponding relational tables involved.

- Each department offers one or more courses; each course is offered by exactly one department.
- The relation COURSE mapped from the entity Course on the M side of the 1:M relationship has a foreign key DeptId that corresponds to the primary key of the relation DEPARTMENT mapped from the 1 side of the 1:M relationship.

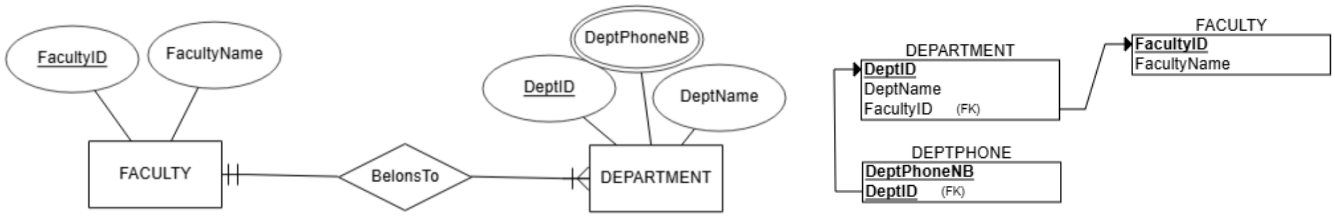

Figure 51 – ERD schema for 1:M relationship between entities Faculty and Department with the corresponding relational tables involved.

- Each faculty oversees one or more departments; each department belongs to exactly one faculty.
- The relation DEPARTMENT mapped from the entity Department on the M side of the 1:M relationship has a foreign key FacultyID that corresponds to the primary key of the relation FACULTY mapped from the 1 side of the 1:M relationship.

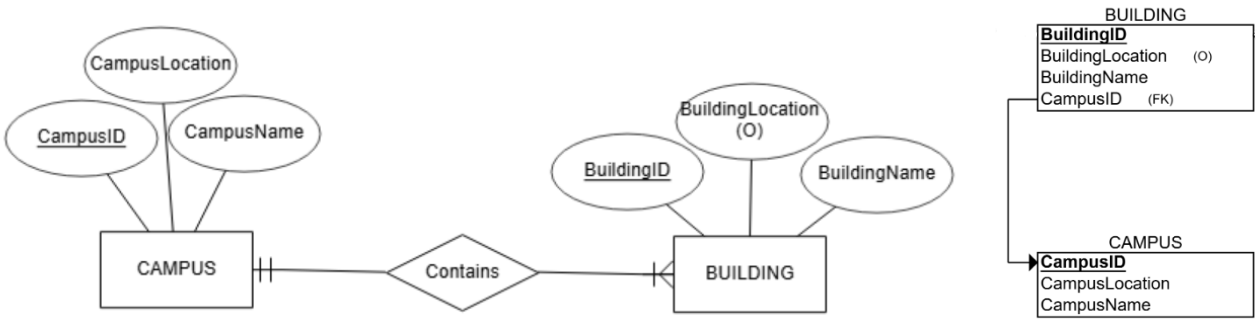

Figure 52 – ERD schema for 1:M relationship between entities Campus and Building with the corresponding relational tables involved.

- Each campus contains one or more buildings; each building is located on exactly one campus.

- The relation BUILDING mapped from the entity Building on the M side of the 1:M relationship has a foreign key CampusID that corresponds to the primary key of the relation CAMPUS mapped from the 1 side of the 1:M relationship.

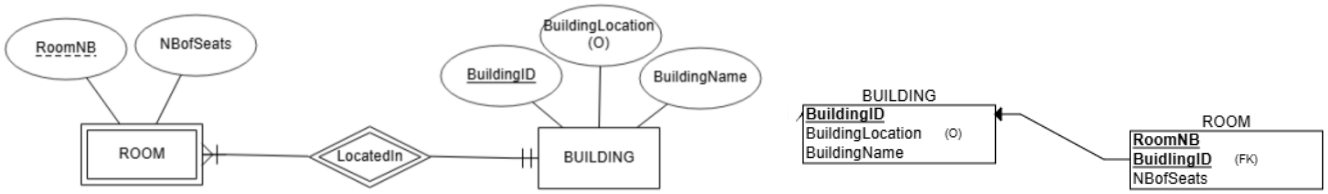

Figure 53 – ERD schema for 1:M relationship between entities Faculty and Department with the corresponding relational tables involved.

- Each building has one or more rooms; each room is located in exactly one building. The partially unique attribute of Room is identified by the full key of Building.
- The relation ROOM mapped from the entity Room on the M side of the 1:M relationship has a foreign key BuildingID that corresponds to the primary key of the relation BUILDING mapped from the 1 side of the 1:M relationship.
- In this case, because we have a weak entity ROOM, the resulting relation has a composite primary key RoomNB and BuildingID (as previously mentioned) that is composed of the partial identifier RoomNB and the foreign key BuildingID corresponding to the primary key of the owner entity BUILDING.

Granular Many-to-Many Relationships

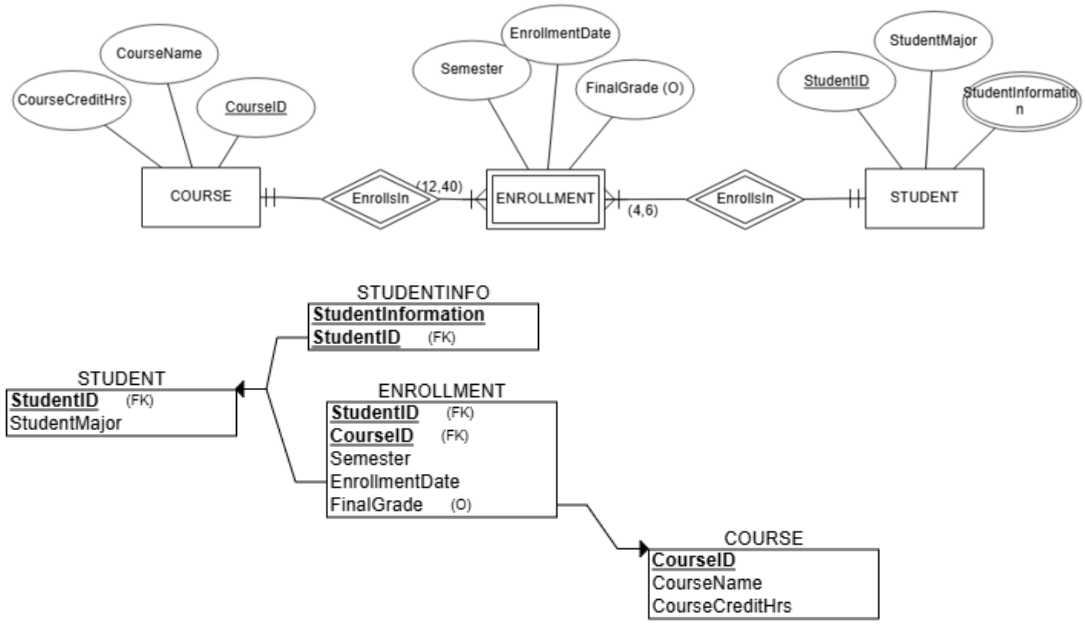

Figure 54 – ERD schema for M:N relationship between Student and Course through the weak Enrollment entity with the corresponding relational tables involved.

- Each student can enroll in a minimum of 4 courses and a maximum of 6 courses; each course can have a minimum of 12 students enrolled in and a maximum of 40. The system will keep track of this many-to-many relationship via the enrollment entity.
- For each instance of student enrollment in a course, the system keeps track of the enrollment semester, enrollment date, and optional final grade.

- In this case, we are imposing a many-to-many relationship between entity Student and entity Course via the weak entity Enrollment. Consequently, the relation ENROLLMENT has a composite primary key composed of the foreign keys StudentID and CourseID corresponding to the primary key of each of the owner entities STUDENT and COURSE.

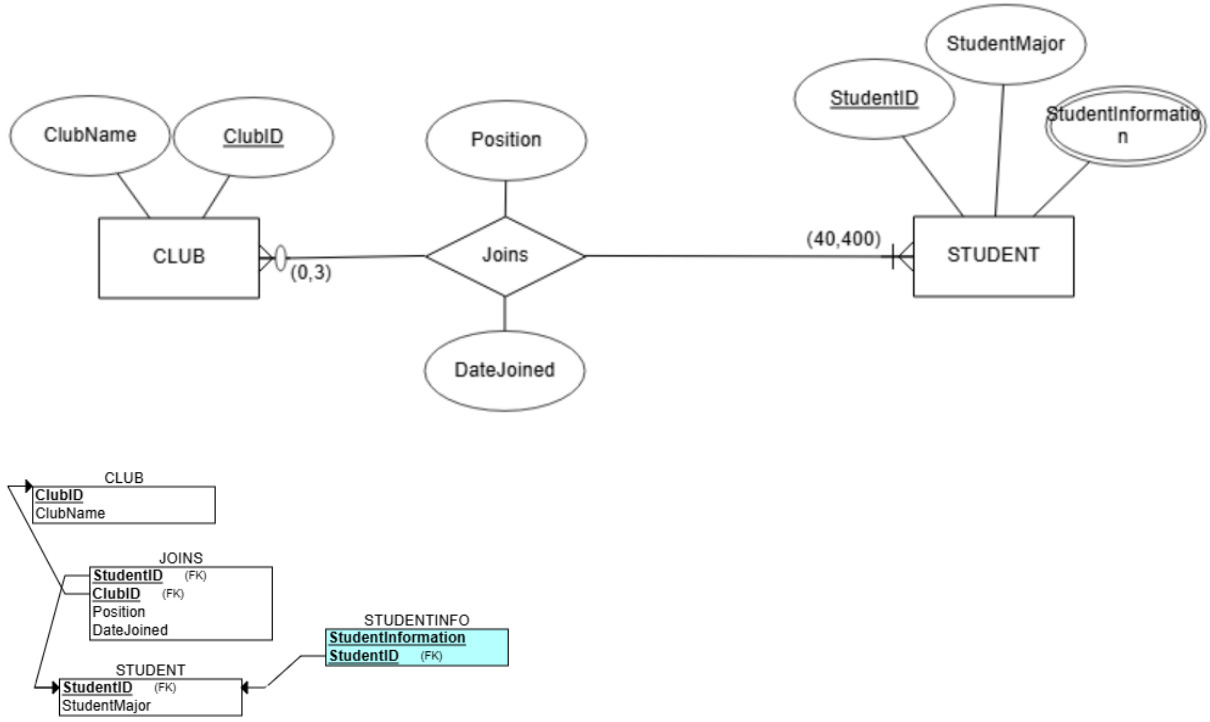

Figure 55 – ERD schema for M:N relationship between entities Club and Student with the corresponding relational tables involved.

- Each student can join zero or a maximum of 3 clubs; each club needs to have at least 40 members and a maximum of 400 students as members.
- For each instance of a student joining a club, the system keeps track of the position of the student within the club and the date joined.
- In this case, we are imposing a many-to-many relationship between entity Club and entity Student. Consequently, a new relation JOINS, which has a composite primary key composed of the foreign keys StudentID and ClubID corresponding to the primary key of each of the owner entities STUDENT and CLUB, is added to the ERD schema. This relation also has additional attributes Position and DateJoined.

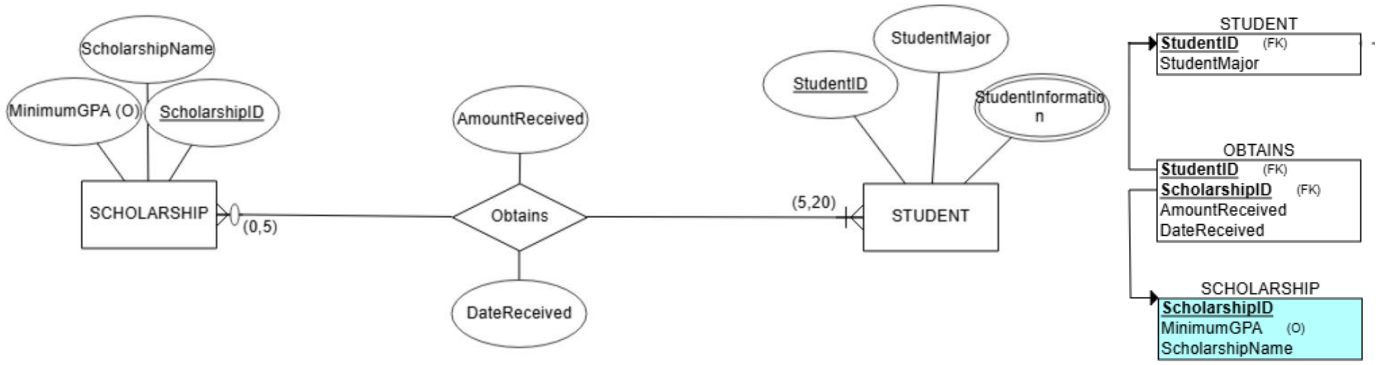

Figure 56 – ERD schema for M:N relationship between entities Scholarship and Student with the corresponding relational tables involved.

- Each student can receive zero to 5 scholarships; each scholarship can be awarded to a minimum of 5 and a maximum of 20 students.
- For each instance of student obtaining a scholarship, the system keeps track of the percentage of amount granted for the student and the date receiving the scholarship.
- In this case, we are imposing a many-to-many relationship between entity Scholarship and entity Student. Consequently, a new relation OBTAINS, which has a composite primary key composed of the foreign keys StudentID and ScholarshipID corresponding to the primary key of each of the owner entities STUDENT and SCHOLARSHIP, is added to the ERD schema. This relation also has additional attributes AmountReceived and DateReceived.

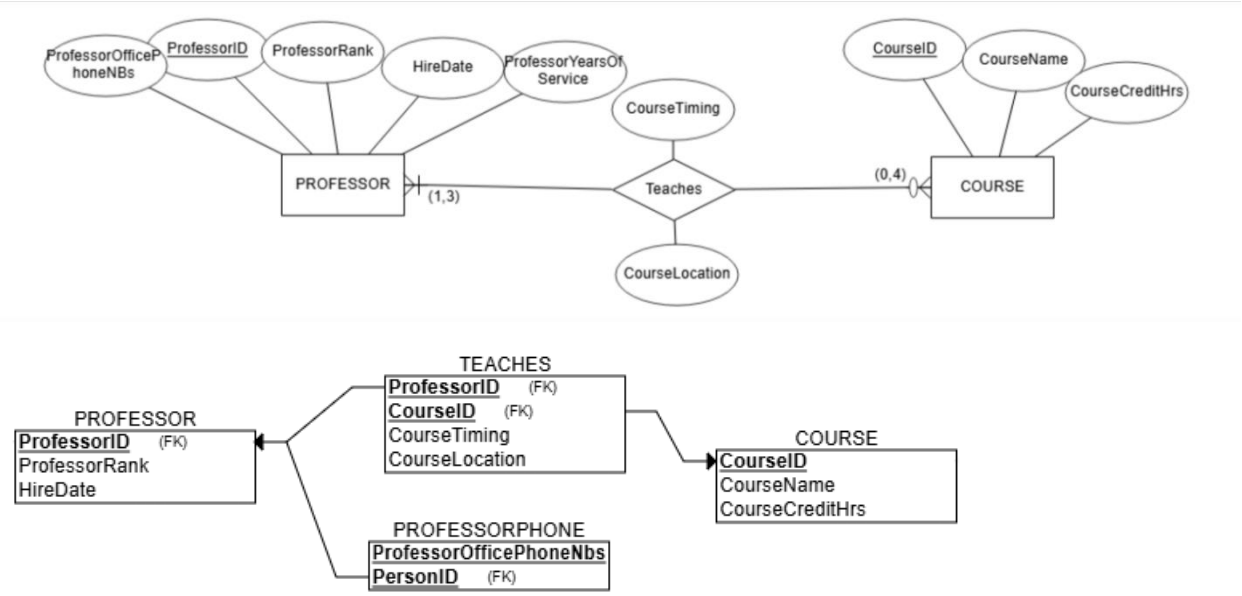

Figure 57 – ERD schema for M:N relationship between entities Professor and Course with the corresponding relational tables involved.

- Each professor can teach zero courses and a maximum of 4 courses; each course is taught by at least 1 professor and a maximum of 3 professors.
- For each instance of professor and course assignment, the system keeps track of the course timing and location.
- In this case, we are imposing a many-to-many relationship between entity Professor and entity Course. Consequently, a new relation TEACHES, which has a composite primary key composed of the foreign keys ProfessorID and CourseID corresponding to the primary key of each of the owner entities PROFESSOR and COURSE, is added to the ERD schema. This relation also has additional attributes CourseTiming and CourseLocation.

Granular Unary Relationships

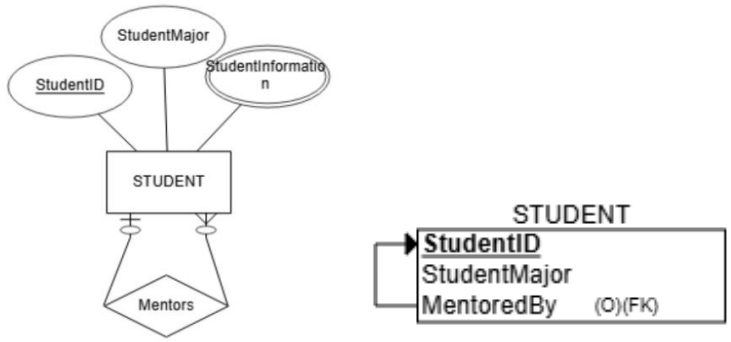

Figure 58 – ERD schema for unary relationship between entity Student and itself with the corresponding relational tables involved.

- Each student may mentor zero or more other students; each student may be mentored by zero or one student.
- In this case, we are imposing a 1:M unary relationship between entity Student and itself. Consequently, the relation STUDENT mapped from an entity involved in a 1:M unary relationship contains a foreign key MentoredBy that corresponds to its own primary key StudentID.

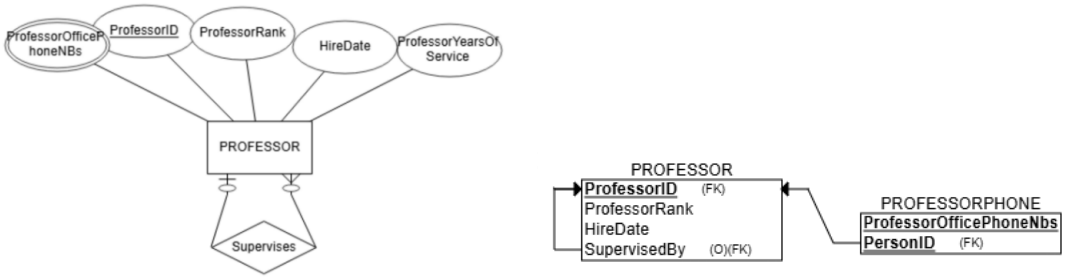

Figure 59 – ERD schema for unary relationship between entity Professor and itself with the corresponding relational tables involved.

- Each professor may supervise zero or more other professors; each professor may be supervised by zero or one professor.
- In this case, we are imposing a 1:M unary relationship between entity Professor and itself. Consequently, the relation PROFESSOR mapped from an entity involved in a 1:M unary relationship contains a foreign key SupervisedBy that corresponds to its own primary key ProfessorID.

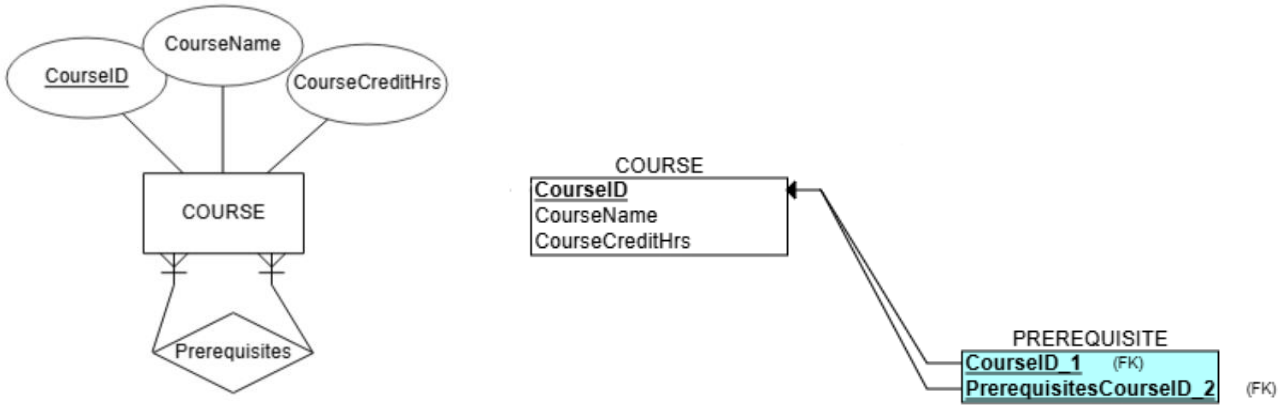

Figure 60 – ERD schema for unary relationship between entity Course and itself with the corresponding relational tables involved.

- Each course can have one or more prerequisites; each prerequisite is associated with one or more courses.
- In this case, we are imposing an M:N unary relationship between entity Course and itself. Consequently, in addition to the relation COURSE representing the entity involved in the unary M:N relationship, a new relation PREREQUISITE is created to represent the M:N relationship itself. It has two foreign keys CourseID_1 and PrerequisitesCourseID_2, both corresponding to the primary key CourseID of the relation COURSE representing the entity involved in the unary M:N relationship. Each of the foreign keys is used as a part of the composite primary key of the new relation.

Chapter 4 SQL Queries

Following the completion of the relational schema design in Chapter 3, this chapter focuses on the implementation phase of the UHO University Registration System. Here, we translate the logical relational model into executable SQL Data Definition Language (DDL) statements. Each entity and relationship from the schema is carefully mapped to SQL CREATE TABLE statements, complete with appropriate data types, primary keys, foreign keys, and integrity constraints to enforce the business logic established in the earlier phases.

This implementation ensures that the database structure is not only logically sound but also physically realizable in a relational database management system (PostgreSQL). By systematically creating the tables and defining the necessary constraints, we lay the technical foundation for data storage, retrieval, and manipulation, ultimately moving one step closer to delivering a fully operational UHO University Registration System.

Table Creations

Before initiating the creation of relational tables for the UHO University Registration System, it was necessary to set up the foundational database environment. This setup involved issuing a series of Data Control Language (DCL) and Data Definition Language (DDL) commands to create the database, users, roles, and appropriate access privileges. Establishing this structure ensures that database operations are carried out securely and efficiently, with clearly defined user permissions from the outset.

```
--Create a database called UHO.
CREATE DATABASE UHO;
--Create a new user that we will use from now on on UHO, this is a general user.
CREATE USER UHO_general_user WITH PASSWORD 'root';
--Create a new role.
CREATE ROLE UHO_general_role;
--Grant the user to this role.
GRANT UHO_general_role TO UHO_general_user;
--Grant all privileges on the database UHO to this role.
GRANT ALL PRIVILEGES ON DATABASE UHO TO UHO_general_role;
--Grant all privileges on the schema public to this role.
GRANT ALL PRIVILEGES ON SCHEMA public TO UHO_general_role;
--Grant usage on the database UHO to this role.
GRANT CONNECT ON DATABASE UHO TO UHO_general_role;
--Grant create on the database UHO to this role
GRANT CREATE ON DATABASE UHO TO UHO_general_role;
--Create a read-only role
CREATE ROLE uho_read_role;
--Create a read-only user (login)
CREATE USER uho_read_user WITH PASSWORD 'read_user';

--Grant the role to the user
GRANT uho_read_role TO uho_read_user;
```

```sql
--Grant the role privileges on the UHO database
--   a) Allow the role to CONNECT to the UHO database
GRANT CONNECT ON DATABASE UHO TO uho_read_role;
--   b) Allow the role to USAGE on the public schema
GRANT USAGE ON SCHEMA public TO uho_read_role;
--   c) Grant SELECT on all existing tables in the public schema
GRANT SELECT ON ALL TABLES IN SCHEMA public TO uho_read_role;
```

The first step was to create a new database named UHO, which will serve as the central repository for all the university's registration-related data. Following the creation of the database, a general-purpose user named UHO_general_user was established, secured with a password. To manage this user's privileges in a flexible and organized manner, a corresponding role called UHO_general_role was created. The user was then granted membership to this role.

The UHO_general_role was assigned full privileges over both the UHO database and the public schema. These privileges include the ability to create tables, insert, update, delete records, and define new schema objects as needed. Additionally, the role was granted the ability to connect to the database and create within it, ensuring that the general user has the complete operational access required for database development and management tasks.

To further enhance security and access control, a read-only role called uho_read_role and a corresponding read-only user named uho_read_user were created. The read-only user was assigned to the uho_read_role, which was granted restricted privileges. Specifically, this role was allowed to connect to the UHO database, use the public schema, and select (i.e., read) data from all existing tables within the schema. By limiting the read-only user's privileges to viewing data only, the system ensures that sensitive records are protected from unintended modifications while still allowing necessary visibility.

This initial setup of users and roles lays a strong foundation for maintaining data security, role-based access control, and separation of concerns throughout the development and deployment phases of the project. It ensures that general operations and administrative tasks are properly managed while allowing for safe, controlled external access when needed.

Data Definition Language (DDL)

With the database environment now prepared, we proceed to implement the relational schema through SQL DDL statements, creating each table according to the previously designed logical model.

```sql
-- 1. PERSON
CREATE TABLE Person (
  person_id      SERIAL          NOT NULL,
  person_ssn     VARCHAR(11)     NOT NULL DEFAULT 'XXX-XX-XXXX',
  first_name     VARCHAR(50)     NOT NULL,
  middle_name    VARCHAR(50),
  last_name      VARCHAR(50)     NOT NULL,
  date_of_birth  DATE            NOT NULL,
```

```
  CONSTRAINT PK_Person
    PRIMARY KEY (person_id),
  CONSTRAINT UQ_Person_SSN UNIQUE (person_ssn),
  CONSTRAINT CHK_Person_dob_past
    CHECK (date_of_birth <= CURRENT_DATE)
);

-- 2. PERSONPHONE (multi-valued phones for Person)
CREATE TABLE PersonPhone (
  person_id  INT   NOT NULL,
  phone_num  VARCHAR(24) NOT NULL DEFAULT '000-000-000',
  CONSTRAINT PK_PersonPhone
    PRIMARY KEY (person_id, phone_num),
  CONSTRAINT FK_PersonPhone_Person
    FOREIGN KEY (person_id)
    REFERENCES Person(person_id)
    ON DELETE CASCADE
);

-- 3. TRANSCRIPT
CREATE TABLE Transcript (
  transcript_id   SERIAL  NOT NULL,
  creation_date   DATE    NOT NULL DEFAULT CURRENT_DATE,
  CONSTRAINT PK_Transcript
    PRIMARY KEY (transcript_id)
);

-- 4. STUDENT
CREATE TABLE Student (
  student_id     INT      NOT NULL,
  transcript_id  INT      NOT NULL,
  major          VARCHAR(50)      NOT NULL DEFAULT 'Undeclared',
  mentor_id      INT,
  CONSTRAINT PK_Student
    PRIMARY KEY (student_id),
  CONSTRAINT FK_Student_Person
    FOREIGN KEY (student_id)
    REFERENCES Person(person_id)
    ON DELETE CASCADE,
  CONSTRAINT FK_Student_Transcript
    FOREIGN KEY (transcript_id)
    REFERENCES Transcript(transcript_id)
    ON DELETE CASCADE,
  CONSTRAINT FK_Student_Mentor
    FOREIGN KEY (mentor_id)
    REFERENCES Student(student_id)
    ON DELETE SET NULL
);
-- 5. STUDENTINFO (multi-valued info for Student)
CREATE TABLE StudentInfo (
```

```
  student_id    INT      NOT NULL,
  student_info  VARCHAR(100) NOT NULL DEFAULT 'No Information',
  CONSTRAINT PK_StudentInfo
    PRIMARY KEY (student_id, student_info),
  CONSTRAINT FK_StudentInfo_Student
    FOREIGN KEY (student_id)
    REFERENCES Student(student_id)
    ON DELETE CASCADE
);

-- 6. PROFESSOR
CREATE TABLE Professor (
  professor_id   INT      NOT NULL,
  professor_rank           VARCHAR(50) NOT NULL DEFAULT 'Undeclared',
  hire_date      DATE        NOT NULL DEFAULT CURRENT_DATE,
  supervisor_id  INT,
  CONSTRAINT PK_Professor
    PRIMARY KEY (professor_id),
  CONSTRAINT FK_Professor_Person
    FOREIGN KEY (professor_id)
    REFERENCES Person(person_id)
    ON DELETE CASCADE,
  CONSTRAINT FK_Professor_Supervisor
    FOREIGN KEY (supervisor_id)
    REFERENCES Professor(professor_id)
    ON DELETE SET NULL,
  CONSTRAINT CHK_Professor_hire_date_past
    CHECK (hire_date <= CURRENT_DATE)
);

-- 7. PROFESSORPHONE (multi-valued office phones)
CREATE TABLE ProfessorPhone (
  professor_id INT    NOT NULL,
  phone_num    VARCHAR(24) NOT NULL DEFAULT '000-000-000',
  CONSTRAINT PK_ProfessorPhone
    PRIMARY KEY (professor_id, phone_num),
  CONSTRAINT FK_ProfessorPhone_Professor
    FOREIGN KEY (professor_id)
    REFERENCES Professor(professor_id)
    ON DELETE CASCADE
);

-- 8. CAMPUS
CREATE TABLE Campus (
  campus_id       SERIAL     NOT NULL,
  campus_name     VARCHAR(100) NOT NULL,
  campus_location VARCHAR(100) NOT NULL,
  CONSTRAINT PK_Campus
    PRIMARY KEY (campus_id) );
-- 9. BUILDING
```

```sql
CREATE TABLE Building (
  building_id       SERIAL     NOT NULL,
  building_name     VARCHAR(100) NOT NULL,
  building_location VARCHAR(100),
  campus_id         INT         NOT NULL,
  CONSTRAINT PK_Building
    PRIMARY KEY (building_id),
  CONSTRAINT FK_Building_Campus
    FOREIGN KEY (campus_id)
    REFERENCES Campus(campus_id)
    ON DELETE CASCADE
);

-- 10. ROOM
CREATE TABLE Room (
  building_id INT     NOT NULL,
  room_number VARCHAR(10) NOT NULL,
  nb_seats    INT     NOT NULL DEFAULT 30,
  CONSTRAINT PK_Room
    PRIMARY KEY (building_id, room_number),
  CONSTRAINT FK_Room_Building
    FOREIGN KEY (building_id)
    REFERENCES Building(building_id)
    ON DELETE CASCADE,
  CONSTRAINT CHK_Room_seats_positive
    CHECK (nb_seats > 0)
);

-- 11. FACULTY
CREATE TABLE Faculty (
  faculty_id   SERIAL      NOT NULL,
  faculty_name VARCHAR(100) NOT NULL,
  building_id  INT          NOT NULL,
  CONSTRAINT PK_Faculty
    PRIMARY KEY (faculty_id),
  CONSTRAINT FK_Faculty_Building
    FOREIGN KEY (building_id)
    REFERENCES Building(building_id)
    ON DELETE CASCADE
);

-- 12. DEPARTMENT
CREATE TABLE Department (
  dept_id    SERIAL      NOT NULL,
  dept_name  VARCHAR(100) NOT NULL,
  faculty_id INT          NOT NULL,
  CONSTRAINT PK_Department
    PRIMARY KEY (dept_id),
  CONSTRAINT FK_Department_Faculty
    FOREIGN KEY (faculty_id)
```

```
    REFERENCES Faculty(faculty_id)
    ON DELETE CASCADE
);

-- 13. DEPARTMENTPHONE (multi-valued phones)
CREATE TABLE DepartmentPhone (
  dept_id   INT    NOT NULL,
  phone_num VARCHAR(24) NOT NULL DEFAULT '000-000-000',
  CONSTRAINT PK_DepartmentPhone
    PRIMARY KEY (dept_id, phone_num),
  CONSTRAINT FK_DepartmentPhone_Department
    FOREIGN KEY (dept_id)
    REFERENCES Department(dept_id)
    ON DELETE CASCADE
);

-- 14. COURSE
CREATE TABLE Course (
  course_id    SERIAL      NOT NULL,
  course_name  VARCHAR(100) NOT NULL,
  credit_hours INT          NOT NULL,
  dept_id      INT          NOT NULL,
  CONSTRAINT PK_Course
    PRIMARY KEY (course_id),
  CONSTRAINT FK_Course_Department
    FOREIGN KEY (dept_id)
    REFERENCES Department(dept_id)
    ON DELETE CASCADE,
  CONSTRAINT CHK_Course_credit_positive
    CHECK (credit_hours > 0)
);

-- 15. EXAM
CREATE TABLE Exam (
  course_id    INT           NOT NULL,
  exam_number  VARCHAR(10)   NOT NULL,
  exam_date    DATE          NOT NULL,
  exam_location VARCHAR(50),
  exam_weight  DECIMAL(5,2)  NOT NULL,
  CONSTRAINT PK_Exam
    PRIMARY KEY (course_id, exam_number),
  CONSTRAINT FK_Exam_Course
    FOREIGN KEY (course_id)
    REFERENCES Course(course_id)
    ON DELETE CASCADE,
  CONSTRAINT CHK_Exam_weight_range
    CHECK (exam_weight >= 0 AND exam_weight <= 100)
);
```

```sql
-- 16. PREREQUISITE
CREATE TABLE Prerequisite (
  course_id      INT NOT NULL,
  prereq_course  INT NOT NULL,
  CONSTRAINT PK_Prerequisite
    PRIMARY KEY (course_id, prereq_course),
  CONSTRAINT FK_Prerequisite_Course
    FOREIGN KEY (course_id)
    REFERENCES Course(course_id)
    ON DELETE CASCADE,
  CONSTRAINT FK_Prerequisite_Prereq
    FOREIGN KEY (prereq_course)
    REFERENCES Course(course_id)
    ON DELETE CASCADE
);

-- 17. ENROLLMENT
CREATE TABLE Enrollment (
  student_id      INT     NOT NULL,
  course_id       INT     NOT NULL,
  semester        VARCHAR(10) NOT NULL,
  enrollment_date DATE    NOT NULL DEFAULT CURRENT_DATE,
  final_grade     DECIMAL(4,2),
  CONSTRAINT PK_Enrollment
    PRIMARY KEY (student_id, course_id),
  CONSTRAINT FK_Enrollment_Student
    FOREIGN KEY (student_id)
    REFERENCES Student(student_id)
    ON DELETE CASCADE,
  CONSTRAINT FK_Enrollment_Course
    FOREIGN KEY (course_id)
    REFERENCES Course(course_id)
    ON DELETE CASCADE,
  CONSTRAINT CHK_Enrollment_date_past
    CHECK (enrollment_date <= CURRENT_DATE),
  CONSTRAINT CHK_Enrollment_grade_range
    CHECK (final_grade IS NULL OR (final_grade >= 0 AND final_grade <= 4))
);

-- 18. TEACHES
CREATE TABLE Teaches (
  professor_id   INT     NOT NULL,
  course_id      INT     NOT NULL,
  course_timing  VARCHAR(50)  NOT NULL,
  course_location VARCHAR(100) NOT NULL,
  CONSTRAINT PK_Teaches
    PRIMARY KEY (professor_id, course_id),
  CONSTRAINT FK_Teaches_Professor
    FOREIGN KEY (professor_id)
```

```
    REFERENCES Professor(professor_id)
    ON DELETE CASCADE,
  CONSTRAINT FK_Teaches_Course
    FOREIGN KEY (course_id)
    REFERENCES Course(course_id)
    ON DELETE CASCADE
);

-- 19. CLUB
CREATE TABLE Club (
  club_id   SERIAL      NOT NULL,
  club_name VARCHAR(100) NOT NULL,
  CONSTRAINT PK_Club
    PRIMARY KEY (club_id)
);

-- 20. STUDENTCLUB (JOINS)
CREATE TABLE StudentClub (
  student_id  INT          NOT NULL,
  club_id     INT          NOT NULL,
  position    VARCHAR(50),
  date_joined DATE DEFAULT CURRENT_DATE,
  CONSTRAINT PK_StudentClub
    PRIMARY KEY (student_id, club_id),
  CONSTRAINT FK_StudentClub_Student
    FOREIGN KEY (student_id)
    REFERENCES Student(student_id)
    ON DELETE CASCADE,
  CONSTRAINT FK_StudentClub_Club
    FOREIGN KEY (club_id)
    REFERENCES Club(club_id)
    ON DELETE CASCADE
);

-- 21. SCHOLARSHIP
CREATE TABLE Scholarship (
  scholarship_id   SERIAL       NOT NULL,
  scholarship_name VARCHAR(100) NOT NULL,
  min_gpa          DECIMAL(3,2) DEFAULT 0.00,
  CONSTRAINT PK_Scholarship
    PRIMARY KEY (scholarship_id),
  CONSTRAINT CHK_Scholarship_min_gpa_range
    CHECK (min_gpa >= 0 AND min_gpa <= 4)
);

-- 22. STUDENTSCHOLARSHIP (OBTAINS)
CREATE TABLE StudentScholarship (
  student_id     INT           NOT NULL,
  scholarship_id INT           NOT NULL,
  amount_received_percentage DECIMAL(5,2),
```

```
  date_received  DATE          NOT NULL DEFAULT CURRENT_DATE,
  CONSTRAINT PK_StudentScholarship
    PRIMARY KEY (student_id, scholarship_id),
  CONSTRAINT FK_StudentScholarship_Student
    FOREIGN KEY (student_id)
    REFERENCES Student(student_id)
    ON DELETE CASCADE,
  CONSTRAINT FK_StudentScholarship_Scholarship
    FOREIGN KEY (scholarship_id)
    REFERENCES Scholarship(scholarship_id)
    ON DELETE CASCADE,
  CONSTRAINT CHK_StudSchol_amount_pct
    CHECK (amount_received_percentage >= 0 AND amount_received_percentage <= 100)
);
```

In this section, we translated the relational design of the UHO University Registration System into executable SQL DDL statements. Each relational entity, as previously defined in the conceptual and relational models, has been carefully implemented as a corresponding SQL table. Special attention was given to ensuring that entity integrity, referential integrity, and business logic constraints are fully respected.

Each entity from the ER model was mapped to a CREATE TABLE statement. Tables were designed with appropriate data types to suit the nature of the attributes, such as VARCHAR for textual data, DATE for date fields, and DECIMAL for numeric fields with precision requirements. All primary keys were explicitly defined to ensure entity uniqueness, and foreign key constraints were implemented to enforce relationships between tables. Where necessary, cascading actions (e.g., ON DELETE CASCADE) were incorporated to maintain consistency during deletions.

Multivalued attributes were handled through separate associative tables (e.g., PersonPhone, ProfessorPhone, DepartmentPhone, StudentInfo), maintaining full normalization. Weak entities like Exam and associative relationships like Enrollment, StudentClub, and StudentScholarship were modeled using composite primary keys, ensuring proper dependency on their owning entities.

Business rules such as value constraints were enforced through CHECK constraints. Examples include ensuring the hire_date and date_of_birth are not in the future, restricting grades and GPA values within acceptable academic ranges, and verifying the positivity of seat counts for rooms and percentages for scholarships.

Additionally, unary relationships like student mentorship and professor supervision were implemented via self-referencing foreign keys (mentor_id, supervisor_id) with ON DELETE SET NULL behavior to gracefully handle disassociations.

This structured SQL DDL implementation guarantees that the UHO University Registration System database not only mirrors the relational design accurately but is also robust, consistent, and ready for the insertion of operational data in subsequent steps.

Testing Queries:

- See all tables created:

```
SELECT table_name
FROM information_schema.tables
WHERE table_schema = 'public';
```

Data Output | Messages | Notifications

	table_name name
1	person
2	personphone
3	student
4	transcript
5	studentinfo
6	professor
7	professorphone
8	campus
9	building
10	room
11	faculty
12	department
13	departmentphone
14	course
15	exam
16	prerequisite
17	enrollment
18	teaches
19	studentclub
20	club
21	studentscholarship
22	scholarship

Total rows: 22 Query complete 00:00:00.164

- Check foreign keys were created:

```
SELECT conname, confrelid::regclass AS referenced_table
FROM pg_constraint
WHERE contype = 'f';
```

Data Output | Messages | Notifications

	conname name	referenced_table regclass
1	fk_personphone_person	person
2	fk_student_person	person
3	fk_student_transcript	transcript
4	fk_student_mentor	student
5	fk_studentinfo_student	student
6	fk_professor_person	person
7	fk_professor_supervisor	professor
8	fk_professorphone_professor	professor
9	fk_building_campus	campus
10	fk_room_building	building
11	fk_faculty_building	building
12	fk_department_faculty	faculty
13	fk_departmentphone_department	department
14	fk_course_department	department
15	fk_exam_course	course
16	fk_prerequisite_course	course
17	fk_prerequisite_prereq	course
18	fk_enrollment_student	student
19	fk_enrollment_course	course
20	fk_teaches_professor	professor
21	fk_teaches_course	course
22	fk_studentclub_student	student
23	fk_studentclub_club	club
24	fk_studentscholarship_student	student
25	fk_studentscholarship_scholarship	scholarship

- Check constraints (checks and uniques):
    - For table Person:

```
SELECT conname, contype
FROM pg_constraint
WHERE conrelid = 'person'::regclass;
```

Data Output | Messages | Notifications

	conname name	contype "char"
1	chk_person_dob_past	c
2	pk_person	p
3	uq_person_ssn	u

    - For table Enrollment:

```
SELECT conname, contype
FROM pg_constraint
WHERE conrelid = 'enrollment'::regclass;
```

Data Output | Messages | Notifications

	conname name	contype "char"
1	chk_enrollment_date_past	c
2	chk_enrollment_grade_range	c
3	pk_enrollment	p
4	fk_enrollment_student	f
5	fk_enrollment_course	f

Data Insertion:

With the relational schema successfully implemented, the next step will involve populating the database with sample records and testing query operations.

After successfully creating the database structure through DCL and DDL statements, the next step was to populate the UHO University Registration System with initial sample data. This stage aims to validate the relational integrity, enforce business rules through constraints, and simulate realistic academic operations through meaningful records across all entities.

The data insertion process included:

The Person table was populated with 25 individuals, each assigned a unique Social Security Number (SSN), name, and date of birth, accurately representing a mix of students and professors. Multivalued attributes such as multiple phone numbers were handled through the PersonPhone table, ensuring normalization.

The Transcript table was initialized with 20 records corresponding to the student population. Subsequently, 20 Student records were created, each linked to a unique person and transcript, and including mentor relationships among students to validate unary relationships. Additional student details, such as GPA-related information, were inserted into the StudentInfo table.

Five Professor records were inserted, linked to persons and demonstrating supervision relationships among faculty members. Their office phone numbers were captured separately in the ProfessorPhone table.

At the structural level, three Campus records were created, with multiple Building entries mapped to the respective campuses. The Room table included multiple classrooms with seat counts to simulate realistic teaching environments.

Faculties and Departments were populated next, linked appropriately to buildings and faculties, while DepartmentPhone entries were added for multivalued contact information. The Course table was filled with academic offerings spanning different departments, while course Prerequisites were set to reflect course progression.

Exams were added for several courses into the Exam table, testing the weak entity structure. Enrollments were inserted to validate the many-to-many relationship between students and courses, including semester and final grade information.

The Teaches table was populated to show the many-to-many teaching assignments between professors and courses. Students were also assigned to various Clubs via the StudentClub table, simulating club participation with dates and roles.

Finally, several Scholarships were created with associated minimum GPA requirements, and awards were linked to students through the StudentScholarship table, complete with awarded percentages and dates.

```sql
-- 1) Persons (serial person_id: 1–25)
INSERT INTO Person (person_ssn, first_name, middle_name, last_name, date_of_birth) VALUES
  ('111-11-1111','Alice', NULL,     'Smith',     '1990-04-15'),
  ('222-22-2222','Bob',   'J.',     'Brown',     '1988-07-23'),
  ('333-33-3333','Carol', NULL,     'Davis',     '1992-12-02'),
  ('444-44-4444','David', NULL,     'Evans',     '1991-03-10'),
  ('555-55-5555','Eva',   NULL,     'Frank',     '1993-09-30'),
  ('666-66-6666','Frank', NULL,     'Green',     '1987-05-25'),
  ('777-77-7777','Grace', NULL,     'Hall',      '1994-11-12'),
  ('888-88-8888','Henry', NULL,     'Irwin',     '1989-02-20'),
  ('999-99-9999','Irene', NULL,     'Johnson',   '1995-06-18'),
  ('101-01-0101','Jack',  NULL,     'King',      '1990-12-01'),
  ('121-21-2121','Karen', NULL,     'Lee',       '1992-01-09'),
  ('131-31-3131','Leo',   NULL,     'Martinez',  '1991-08-23'),
  ('141-41-4141','Maya',  NULL,     'Nelson',    '1993-04-04'),
  ('151-51-5151','Nathan',NULL,     'OBrien',    '1988-10-14'),
  ('161-61-6161','Olivia',NULL,     'Perez',     '1994-07-07'),
  ('171-71-7171','Paul',  NULL,     'Quinn',     '1989-09-09'),
  ('181-81-8181','Quinn', NULL,     'Roberts',   '1990-02-02'),
  ('191-91-9191','Rachel',NULL,     'Scott',     '1992-05-05'),
  ('202-02-0202','Steve', NULL,     'Turner',    '1987-12-12'),
  ('212-12-1212','Tina',  NULL,     'Underwood', '1991-03-03'),
  ('313-13-1313','Uma',   NULL,     'Vargas',    '1975-05-20'),
  ('414-14-1414','Victor',NULL,     'White',     '1972-02-12'),
  ('515-15-1515','Wendy', NULL,     'Xu',        '1980-07-07'),
  ('616-16-1616','Xavier',NULL,     'Young',     '1978-11-11'),
  ('717-17-1717','Zoe',   NULL,     'Zhang',     '1982-03-03');

-- 2) PersonPhone
INSERT INTO PersonPhone (person_id, phone_num) VALUES
  (1,'555-0001'), (2,'555-0002'), (3,'555-0003');

-- 3) Transcripts (IDs 1–20)
INSERT INTO Transcript DEFAULT VALUES;  -- repeat 20 times
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
```

```sql
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;

-- 4) Students (IDs 1–20 → person 1–20, transcripts 1–20)
INSERT INTO Student (student_id, transcript_id, major, mentor_id) VALUES
  (1,1,'Computer Science',   NULL),
  (2,2,'Mathematics',        1),
  (3,3,'Physics',            1),
  (4,4,'Engineering',        2),
  (5,5,'Biology',            3),
  (6,6,'Chemistry',          4),
  (7,7,'History',            5),
  (8,8,'Art',                6),
  (9,9,'Music',              7),
  (10,10,'Economics',        8),
  (11,11,'Philosophy',       9),
  (12,12,'Psychology',       10),
  (13,13,'Sociology',        11),
  (14,14,'Literature',       12),
  (15,15,'Geology',          13),
  (16,16,'Astronomy',        14),
  (17,17,'Comp Eng',         15),
  (18,18,'Elec Eng',         16),
  (19,19,'Mech Eng',         17),
  (20,20,'Civil Eng',        18);

-- 5) StudentInfo
INSERT INTO StudentInfo (student_id, student_info) VALUES
  (1,'GPA:3.50'), (2,'GPA:3.75'), (3,'GPA:3.20'), (4,'GPA:3.80'),
  (5,'GPA:3.10'), (6,'GPA:3.40'), (7,'GPA:3.60'), (8,'GPA:3.30'),
  (9,'GPA:3.90'), (10,'GPA:3.45'), (11,'GPA:3.55'), (12,'GPA:3.25'),
  (13,'GPA:3.15'), (14,'GPA:3.85'), (15,'GPA:3.95'), (16,'GPA:3.05'),
  (17,'GPA:3.65'), (18,'GPA:3.70'), (19,'GPA:3.35'), (20,'GPA:3.00');

-- 6) Professors (IDs 21–25 → persons 21–25)
INSERT INTO Professor (professor_id, professor_rank, hire_date, supervisor_id) VALUES
  (21,'Assistant', '2015-08-01', NULL),
  (22,'Associate', '2010-06-15', 21),
  (23,'Full',      '2005-09-01', 22),
  (24,'Assistant', '2018-01-20', 21),
  (25,'Associate', '2012-11-30', 23);

-- 7) ProfessorPhone
INSERT INTO ProfessorPhone (professor_id, phone_num) VALUES
  (21,'555-1001'), (22,'555-1002'), (23,'555-1003'),
  (24,'555-1004'), (25,'555-1005');

-- 8) Campuses
INSERT INTO Campus (campus_name, campus_location) VALUES
```

```
  ('Main Campus',     'City A'),
  ('Downtown Campus', 'City B'),
  ('West Campus',     'City C');

-- 9) Buildings
INSERT INTO Building (building_name, building_location, campus_id) VALUES
  ('Science Hall',     'North end',  1),
  ('Library',          'Center',     1),
  ('Engineering Bldg','East side',  2),
  ('Arts Building',    'West side',  2),
  ('Business Center',  'Downtown',   3);

-- 10) Rooms
INSERT INTO Room (building_id, room_number, nb_seats) VALUES
  (1,'101',30),(1,'102',25),
  (2,'201',40),(2,'202',35),
  (3,'301',50),(3,'302',45),
  (4,'401',20),(4,'402',15),
  (5,'501',60),(5,'502',55);

-- 11) Faculties
INSERT INTO Faculty (faculty_name, building_id) VALUES
  ('Science',     1),
  ('Engineering', 3),
  ('Arts',        4);

-- 12) Departments
INSERT INTO Department (dept_name, faculty_id) VALUES
  ('Computer Science',        2),
  ('Mechanical Engineering',  2),
  ('Biology',                 1),
  ('History',                 3),
  ('Business',                3);

-- 13) DepartmentPhone
INSERT INTO DepartmentPhone (dept_id, phone_num) VALUES
  (1,'555-2001'), (2,'555-2002'),
  (3,'555-2003'), (4,'555-2004'),
  (5,'555-2005');

-- 14) Courses
INSERT INTO Course (course_name, credit_hours, dept_id) VALUES
  ('Data Structures',   3, 1),
  ('Algorithms',        3, 1),
  ('Thermodynamics',    4, 2),
  ('Fluid Mechanics',   3, 2),
  ('Genetics',          3, 3),
  ('Microbiology',      3, 3),
  ('World History',     3, 4),
  ('European History',  3, 4),
```

```
  ('Marketing 101',     3, 5),
  ('Finance',           3, 5);

-- 15) Exams (2 each for courses 1–3)
INSERT INTO Exam (course_id, exam_number, exam_date, exam_location, exam_weight) VALUES
  (1,'Exam1','2024-05-01','Room 101',50),
  (1,'Exam2','2024-06-01','Room 102',50),
  (2,'Exam1','2024-05-02','Room 103',60),
  (2,'Exam2','2024-06-02','Room 104',40),
  (3,'Exam1','2024-05-03','Room 105',55),
  (3,'Exam2','2024-06-03','Room 106',45);

-- 16) Prerequisites
INSERT INTO Prerequisite (course_id, prereq_course) VALUES
  (2,1),   -- Algorithms ← Data Structures
  (4,3);   -- Fluid Mechanics ← Thermodynamics

-- 17) Enrollments (one per student 1–6; you can add more to reach 4–6 each)
INSERT INTO Enrollment (student_id, course_id, semester, enrollment_date, final_grade)
VALUES
  (1,1,'Spring2024','2024-01-10',NULL),
  (2,2,'Spring2024','2024-01-11',NULL),
  (3,3,'Spring2024','2024-01-12',NULL),
  (4,4,'Spring2024','2024-01-13',NULL),
  (5,5,'Spring2024','2024-01-14',NULL),
  (6,6,'Spring2024','2024-01-15',NULL);

-- 18) Teaches
INSERT INTO Teaches (professor_id, course_id, course_timing, course_location) VALUES
  (21,1,'MWF 9–10','Room 101'),
  (22,2,'TTh 10–11','Room 102'),
  (23,3,'MWF 11–12','Room 103'),
  (24,4,'TTh 1–2','Room 104'),
  (25,5,'MWF 2–3','Room 105');

-- 19) Clubs
INSERT INTO Club (club_name) VALUES
  ('Robotics'),
  ('Debate'),
  ('Chess');

-- 20) StudentClub
INSERT INTO StudentClub (student_id, club_id, position, date_joined) VALUES
  (1,1,'President','2024-01-15'),
  (2,1,NULL,'2024-02-01'),
  (3,2,'Secretary','2024-02-10'),
  (4,2,NULL,'2024-03-05'),
  (5,3,'Member','2024-01-20'),
  (6,3,NULL,'2024-02-15');
```

```sql
-- 21) Scholarships
INSERT INTO Scholarship (scholarship_name, min_gpa) VALUES
  ('Academic Excellence',3.50),
  ('Athletic Merit',     2.50),
  ('Need-Based',         NULL);

-- 22) StudentScholarship
INSERT INTO StudentScholarship (student_id, scholarship_id, amount_received_percentage,
date_received) VALUES
  (1,1,100,'2024-02-01'),
  (2,1, 50,'2024-03-01'),
  (3,2,100,'2024-02-15'),
  (4,3, 75,'2024-04-01');
```

This comprehensive data set ensures that all primary keys, foreign keys, multivalued attributes, unary relationships, weak entities, many-to-many mappings, and constraints are actively tested, providing a solid basis for system validation and query operations in the next phases.

Testing Queries:

- Check Persons:

```
1  SELECT person_id, first_name, last_name, date_of_birth
2  FROM Person
3  ORDER BY person_id
4  LIMIT 10;
```

Data Output | Messages | Notifications

Showing rows: 1 to 10 | Page No: 1 of 1

	person_id [PK] integer	first_name character varying (50)	last_name character varying (50)	date_of_birth date
1	1	Alice	Smith	1990-04-15
2	2	Bob	Brown	1988-07-23
3	3	Carol	Davis	1992-12-02
4	4	David	Evans	1991-03-10
5	5	Eva	Frank	1993-09-30
6	6	Frank	Green	1987-05-25
7	7	Grace	Hall	1994-11-12
8	8	Henry	Irwin	1989-02-20
9	9	Irene	Johnson	1995-06-18
10	10	Jack	King	1990-12-01

- Check PersonPhone:

```
6  SELECT * FROM PersonPhone;
```

Data Output | Messages | Notifications

Showing rows: 1 to 3 | Page No: 1

	person_id [PK] integer	phone_num [PK] character varying (24)
1	1	555-0001
2	2	555-0002
3	3	555-0003

Check Transcripts:

```
8  SELECT transcript_id, creation_date
9  FROM Transcript
10 ORDER BY transcript_id
11 LIMIT 10;
```

Data Output | Messages | Notifications

Showing rows: 1 to 10 | Page No: 1 of

	transcript_id [PK] integer	creation_date date
1	1	2025-04-28
2	2	2025-04-28
3	3	2025-04-28
4	4	2025-04-28
5	5	2025-04-28
6	6	2025-04-28
7	7	2025-04-28
8	8	2025-04-28
9	9	2025-04-28
10	10	2025-04-28

- Check Students:

```
13  SELECT student_id, major, mentor_id
14  FROM Student
15  ORDER BY student_id
16  LIMIT 10;
```

Data Output | Messages | Notifications

Showing rows: 1 to 10 | Page No: 1 of 1

	student_id [PK] integer	major character varying (50)	mentor_id integer
1	1	Computer Science	[null]
2	2	Mathematics	1
3	3	Physics	1
4	4	Engineering	2
5	5	Biology	3
6	6	Chemistry	4
7	7	History	5
8	8	Art	6
9	9	Music	7
10	10	Economics	8

- Check StudentInfo:

```
18  SELECT * FROM StudentInfo
19  ORDER BY student_id
20  LIMIT 10;
```

Data Output | Messages | Notifications

Showing rows: 1 to 10 | Page No: 1

	student_id [PK] integer	student_info [PK] character varying (100)
1	1	GPA:3.50
2	2	GPA:3.75
3	3	GPA:3.20
4	4	GPA:3.80
5	5	GPA:3.10
6	6	GPA:3.40
7	7	GPA:3.60
8	8	GPA:3.30
9	9	GPA:3.90
10	10	GPA:3.45

- Check Professors:

```
22  SELECT professor_id, professor_rank, hire_date, supervisor_id
23  FROM Professor
24  ORDER BY professor_id;
```

Data Output | Messages | Notifications

Showing rows: 1 to 5 | Page No: 1 of 1

	professor_id [PK] integer	professor_rank character varying (50)	hire_date date	supervisor_id integer
1	21	Assistant	2015-08-01	[null]
2	22	Associate	2010-06-15	21
3	23	Full	2005-09-01	22
4	24	Assistant	2018-01-20	21
5	25	Associate	2012-11-30	23

- Check ProfessorPhone:

```
26  SELECT * FROM ProfessorPhone;
```

Data Output | Messages | Notifications

Showing rows: 1 to 5 | Page No: 1

	professor_id [PK] integer	phone_num [PK] character varying (24)
1	21	555-1001
2	22	555-1002
3	23	555-1003
4	24	555-1004
5	25	555-1005

- Check Campuses:

```
28  SELECT * FROM Campus;
```

Data Output | Messages | Notifications

Showing rows: 1 to 3 | Page No: 1 of 1

	campus_id [PK] integer	campus_name character varying (100)	campus_location character varying (100)
1	1	Main Campus	City A
2	2	Downtown Campus	City B
3	3	West Campus	City C

- Check Buildings:

```
30  SELECT building_id, building_name, campus_id
31  FROM Building
32  ORDER BY building_id;
```

Data Output | Messages | Notifications

Showing rows: 1 to 5 | Page No: 1 of 1

	building_id [PK] integer	building_name character varying (100)	campus_id integer
1	1	Science Hall	1
2	2	Library	1
3	3	Engineering Bldg	2
4	4	Arts Building	2
5	5	Business Center	3

- Check Rooms:

```
34  SELECT building_id, room_number, nb_seats
35  FROM Room
36  ORDER BY building_id, room_number;
```

Data Output | Messages | Notifications

Showing rows: 1 to 10 | Page No: 1 of 1

	building_id [PK] integer	room_number [PK] character varying (10)	nb_seats integer
1	1	101	30
2	1	102	25
3	2	201	40
4	2	202	35
5	3	301	50
6	3	302	45
7	4	401	20
8	4	402	15
9	5	501	60
10	5	502	55

- Check Faculties:

```
38 SELECT faculty_id, faculty_name, building_id
39 FROM Faculty;
```

Data Output | Messages | Notifications

Showing rows: 1 to 3 | Page No: 1 of 1

	faculty_id [PK] integer	faculty_name character varying (100)	building_id integer
1	1	Science	1
2	2	Engineering	3
3	3	Arts	4

- Check Departments:

```
41 SELECT dept_id, dept_name, faculty_id
42 FROM Department;
```

Data Output | Messages | Notifications

Showing rows: 1 to 5 | Page No: 1 of 1

	dept_id [PK] integer	dept_name character varying (100)	faculty_id integer
1	1	Computer Science	2
2	2	Mechanical Engineering	2
3	3	Biology	1
4	4	History	3
5	5	Business	3

- Check DepartmentPhone:

```
44 SELECT * FROM DepartmentPhone;
```

Data Output | Messages | Notifications

Showing rows: 1 to 5 | Page No: 1

	dept_id [PK] integer	phone_num [PK] character varying (24)
1	1	555-2001
2	2	555-2002
3	3	555-2003
4	4	555-2004
5	5	555-2005

- Check Courses:

```
46 SELECT course_id, course_name, credit_hours
47 FROM Course
48 ORDER BY course_id;
```

Data Output | Messages | Notifications

Showing rows: 1 to 10 | Page No: 1 of 1

	course_id [PK] integer	course_name character varying (100)	credit_hours integer
1	1	Data Structures	3
2	2	Algorithms	3
3	3	Thermodynamics	4
4	4	Fluid Mechanics	3
5	5	Genetics	3
6	6	Microbiology	3
7	7	World History	3
8	8	European History	3
9	9	Marketing 101	3
10	10	Finance	3

- Check Exams:

```
50  SELECT course_id, exam_number, exam_date, exam_weight
51  FROM Exam
52  ORDER BY course_id, exam_number;
```

Data Output | Messages | Notifications

Showing rows: 1 to 6 | Page No: 1 of 1

	course_id [PK] integer	exam_number [PK] character varying (10)	exam_date date	exam_weight numeric (5,2)
1	1	Exam1	2024-05-01	50.00
2	1	Exam2	2024-06-01	50.00
3	2	Exam1	2024-05-02	60.00
4	2	Exam2	2024-06-02	40.00
5	3	Exam1	2024-05-03	55.00
6	3	Exam2	2024-06-03	45.00

- Check Prerequisites:

```
54  SELECT course_id, prereq_course
55  FROM Prerequisite;
```

Data Output | Messages | Notifications

Showing rows: 1 to 2 | Page No: 1

	course_id [PK] integer	prereq_course [PK] integer
1	2	1
2	4	3

- Check Enrollments:

```
57  SELECT student_id, course_id, semester
58  FROM Enrollment
59  ORDER BY student_id;
```

Data Output | Messages | Notifications

Showing rows: 1 to 6 | Page No: 1 of 1

	student_id [PK] integer	course_id [PK] integer	semester character varying (10)
1	1	1	Spring2024
2	2	2	Spring2024
3	3	3	Spring2024
4	4	4	Spring2024
5	5	5	Spring2024
6	6	6	Spring2024

- Check Teaches:

```
61 SELECT professor_id, course_id, course_timing
62 FROM Teaches;
```

Data Output | Messages | Notifications

Showing rows: 1 to 5 | Page No: 1 of 1

	professor_id [PK] integer	course_id [PK] integer	course_timing character varying (50)
1	21	1	MWF 9–10
2	22	2	TTh 10–11
3	23	3	MWF 11–12
4	24	4	TTh 1–2
5	25	5	MWF 2–3

- Check Clubs:

```
64 SELECT * FROM Club;
```

Data Output | Messages | Notifications

Showing rows: 1 to 3 | Page No: 1

	club_id [PK] integer	club_name character varying (100)
1	1	Robotics
2	2	Debate
3	3	Chess

- Check StudentClub:

```
66 SELECT student_id, club_id, position
67 FROM StudentClub;
```

Data Output | Messages | Notifications

Showing rows: 1 to 6 | Page No: 1 of 1

	student_id [PK] integer	club_id [PK] integer	position character varying (50)
1	1	1	President
2	2	1	[null]
3	3	2	Secretary
4	4	2	[null]
5	5	3	Member
6	6	3	[null]

- Check Scholarships:

```
69 SELECT scholarship_id, scholarship_name, min_gpa
70 FROM Scholarship;
```

Data Output | Messages | Notifications

Showing rows: 1 to 3 | Page No: 1 of 1

	scholarship_id [PK] integer	scholarship_name character varying (100)	min_gpa numeric (3,2)
1	1	Academic Excellence	3.50
2	2	Athletic Merit	2.50
3	3	Need-Based	[default]

- Check StudentScholarship:

```
72  SELECT student_id, scholarship_id, amount_received_percentage
73  FROM StudentScholarship;
```

Data Output | Messages | Notifications

Showing rows: 1 to 4 | Page No: 1 of 1

	student_id [PK] integer	scholarship_id [PK] integer	amount_received_percentage numeric (5,2)
1	1	1	100.00
2	2	1	50.00
3	3	2	100.00
4	4	3	75.00

Indexing Strategy

In order to optimize query performance and ensure efficient access to data in the UHO University Registration System, an indexing strategy was developed as part of the database implementation phase. This strategy focuses on enhancing the speed of data retrieval for commonly executed queries while maintaining database integrity and minimizing performance bottlenecks as the data volume grows.

```
-- 1. Foreign Key Indices
-- Student
-- Student → Transcript
CREATE INDEX idx_student_transcript   ON Student(transcript_id);
-- Student → Student (mentor)
CREATE INDEX idx_student_mentor       ON Student(mentor_id);

-- Professor: Professor → Professor (supervisor)
CREATE INDEX idx_professor_supervisor ON Professor(supervisor_id);

-- Building & Campus: Building → Campus
CREATE INDEX idx_building_campus      ON Building(campus_id);

-- Faculty & Department
-- Faculty → Building
CREATE INDEX idx_faculty_building     ON Faculty(building_id);
-- Department → Faculty
CREATE INDEX idx_department_faculty   ON Department(faculty_id);

-- Course & Prerequisites
-- Course → Department
CREATE INDEX idx_course_department    ON Course(dept_id);
-- Prerequisite → Course
CREATE INDEX idx_prerequisite_prereq  ON Prerequisite(prereq_course);
CREATE INDEX idx_prerequisite_course  ON Prerequisite(course_id);

-- Enrollment (both columns): Enrollment → Student, Course
CREATE INDEX idx_enrollment_student   ON Enrollment(student_id);
CREATE INDEX idx_enrollment_course    ON Enrollment(course_id);
```

```sql
-- Teaches (both columns)
CREATE INDEX idx_teaches_professor    ON Teaches(professor_id);
CREATE INDEX idx_teaches_course       ON Teaches(course_id);

-- StudentClub (both columns): Teaches → Professor, Course
CREATE INDEX idx_studentclub_student  ON StudentClub(student_id);
CREATE INDEX idx_studentclub_club     ON StudentClub(club_id);

-- StudentScholarship (both columns): StudentScholarship → Student, Scholarship
CREATE INDEX idx_studentschol_student ON StudentScholarship(student_id);
CREATE INDEX idx_studentschol_sch     ON StudentScholarship(scholarship_id);

-- StudentInfo: StudentInfo → Student
CREATE INDEX idx_studentinfo_student  ON StudentInfo(student_id);

-- PersonPhone: PersonPhone → Person
CREATE INDEX idx_personphone_person    ON PersonPhone(person_id);

-- ProfessorPhone: ProfessorPhone → Professor
CREATE INDEX idx_professorphone_prof   ON ProfessorPhone(professor_id);

-- DepartmentPhone: DepartmentPhone → Department
CREATE INDEX idx_departmentphone_dept  ON DepartmentPhone(dept_id);

-- Room: Room → Building
CREATE INDEX idx_room_building         ON Room(building_id);

-- Exam: Exam → Course
CREATE INDEX idx_exam_course           ON Exam(course_id);

--================================================================================

--2. Five columns with Performance Indices
-- 2.1 lookups by last name in Person (e.g. directory searches)
CREATE INDEX idx_person_last_name
  ON Person(last_name);
-- speeds up:  SELECT * FROM Person WHERE last_name = 'Smith';
-- person.last_name: lookups by last name are extremely common in directory-style queries.

-- 2.2 filtering students by major
CREATE INDEX idx_student_major
  ON Student(major);
-- speeds up:  SELECT * FROM Student WHERE major = 'Computer Science';
-- student.major: reports and dashboards often group or filter students by major.
-- 2.3 finding today’s enrollments by semester
CREATE INDEX idx_enrollment_semester
  ON Enrollment(semester);
-- speeds up:  SELECT * FROM Enrollment WHERE semester = 'Fall2025';
-- enrollment.semester: enrollment information is always queried per semester.
-- 2.4 searching courses by name
```

```
CREATE INDEX idx_course_name
  ON Course(course_name);
-- speeds up:  SELECT * FROM Course WHERE course_name ILIKE '%Data%';
-- course.course_name: course catalog searches by name fragment benefit from an index.

-- 2.5 selecting scholarships by minimum-GPA requirement
CREATE INDEX idx_scholarship_min_gpa
  ON Scholarship(min_gpa);
-- speeds up:  SELECT * FROM Scholarship WHERE min_gpa <= 3.5;
-- scholarship.min_gpa: when students look up scholarships they qualify for, the DB filters
on this field.
```

Indexes were created in three key areas:

Foreign Key Indexes

Although primary key constraints automatically generate indexes, foreign key constraints do not. To optimize join operations and maintain referential integrity performance, explicit indexes were created on all foreign key columns across the database. These indexes improve the efficiency of queries that involve joins between related tables, particularly as datasets grow larger.

Examples include:

- Indexes on foreign key columns such as `transcript_id` and `mentor_id` in the **Student** table, supporting both student-transcript linking and mentoring relationships.
- Indexes on `campus_id` in **Building**, `faculty_id` in **Department**, and `dept_id` in **Course** to accelerate hierarchical queries within the university's structure.
- Many-to-many linking tables such as **Enrollment**, **Teaches**, **StudentClub**, and **StudentScholarship** also received indexes on their foreign key columns to facilitate efficient navigation between associated entities.

Overall, **over 20 foreign key indexes** were created to fully support relational queries.

Performance-Oriented Indexes

In addition to foreign key indexing, five **performance-driven indexes** were created on columns that are frequently filtered or searched against in typical operational scenarios:

- **Person.last_name:** An index was created to accelerate directory-style searches by last name, a common operation when locating individuals within the university.
- **Student.major:** Filtering students by their academic major is a frequent task for reporting and advising, making an index on the `major` attribute critical.
- **Enrollment.semester:** Since enrollments are often queried by academic term, indexing the `semester` field improves retrieval speeds for semester-specific reports.

- **Course.course_name:** To enhance the user experience when searching for courses by name or keywords, an index was created on the course name attribute.
- **Scholarship.min_gpa:** Indexing the minimum GPA field supports efficient queries for students seeking scholarships they are eligible for based on academic performance.

Each of these indexes was carefully selected based on expected query patterns and is intended to significantly reduce query execution times on larger datasets.

Validation of Index Creation

To verify successful index creation, queries were executed against PostgreSQL's system catalog (`pg_indexes`) to retrieve and review all existing indexes in the public schema. The results confirmed that all primary, foreign key, and performance-oriented indexes were properly implemented as intended.

This indexing strategy ensures that the UHO University Registration System will maintain high query performance, efficient join operations, and scalable access even as the underlying data grows in complexity and volume.

Testing Queries

- Check current indices (before executing indexing queries):

```sql
--Query to check all current indices. If tables are dropped, all indices are droped too
SELECT
  schemaname,
  tablename,
  indexname,
  indexdef
FROM
  pg_catalog.pg_indexes
WHERE
  schemaname = 'public'
ORDER BY
  tablename,
  indexname;
```

Data Output | Messages | Notifications

	schemaname name	tablename name	indexname name	indexdef text
1	public	building	pk_building	CREATE UNIQUE INDEX pk_building ON public.building USING btree (building_id)
2	public	campus	pk_campus	CREATE UNIQUE INDEX pk_campus ON public.campus USING btree (campus_id)
3	public	club	pk_club	CREATE UNIQUE INDEX pk_club ON public.club USING btree (club_id)
4	public	course	pk_course	CREATE UNIQUE INDEX pk_course ON public.course USING btree (course_id)
5	public	department	pk_department	CREATE UNIQUE INDEX pk_department ON public.department USING btree (dept_id)
6	public	departmentphone	pk_departmentphone	CREATE UNIQUE INDEX pk_departmentphone ON public.departmentphone USING btree (dept_id, phone_num)
7	public	enrollment	pk_enrollment	CREATE UNIQUE INDEX pk_enrollment ON public.enrollment USING btree (student_id, course_id)
8	public	exam	pk_exam	CREATE UNIQUE INDEX pk_exam ON public.exam USING btree (course_id, exam_number)
9	public	faculty	pk_faculty	CREATE UNIQUE INDEX pk_faculty ON public.faculty USING btree (faculty_id)
10	public	person	pk_person	CREATE UNIQUE INDEX pk_person ON public.person USING btree (person_id)
11	public	person	uq_person_ssn	CREATE UNIQUE INDEX uq_person_ssn ON public.person USING btree (person_ssn)
12	public	personphone	pk_personphone	CREATE UNIQUE INDEX pk_personphone ON public.personphone USING btree (person_id, phone_num)
13	public	prerequisite	pk_prerequisite	CREATE UNIQUE INDEX pk_prerequisite ON public.prerequisite USING btree (course_id, prereq_course)
14	public	professor	pk_professor	CREATE UNIQUE INDEX pk_professor ON public.professor USING btree (professor_id)
15	public	professorphone	pk_professorphone	CREATE UNIQUE INDEX pk_professorphone ON public.professorphone USING btree (professor_id, phone_num)
16	public	room	pk_room	CREATE UNIQUE INDEX pk_room ON public.room USING btree (building_id, room_number)
17	public	scholarship	pk_scholarship	CREATE UNIQUE INDEX pk_scholarship ON public.scholarship USING btree (scholarship_id)
18	public	student	pk_student	CREATE UNIQUE INDEX pk_student ON public.student USING btree (student_id)
19	public	studentclub	pk_studentclub	CREATE UNIQUE INDEX pk_studentclub ON public.studentclub USING btree (student_id, club_id)
20	public	studentinfo	pk_studentinfo	CREATE UNIQUE INDEX pk_studentinfo ON public.studentinfo USING btree (student_id, student_info)
21	public	studentscholarsh...	pk_studentscholarsh...	CREATE UNIQUE INDEX pk_studentscholarship ON public.studentscholarship USING btree (student_id, scholarship_id)
22	public	teaches	pk_teaches	CREATE UNIQUE INDEX pk_teaches ON public.teaches USING btree (professor_id, course_id)
23	public	transcript	pk_transcript	CREATE UNIQUE INDEX pk_transcript ON public.transcript USING btree (transcript_id)

Total rows: 23 | Query complete 00:00:00.076

- Check current indices (after executing indexing queries):

```sql
--Query to check all current indices. If tables are dropped, all indices are droped too
SELECT
  schemaname,
  tablename,
  indexname,
  indexdef
FROM
  pg_catalog.pg_indexes
WHERE
  schemaname = 'public'
ORDER BY
  tablename,
  indexname;
```

Data Output | Messages | Notifications

	schemaname name	tablename name	indexname name	indexdef text
1	public	building	idx_building_campus	CREATE INDEX idx_building_campus ON public.building USING btree (campus_id)
2	public	building	pk_building	CREATE UNIQUE INDEX pk_building ON public.building USING btree (building_id)
3	public	campus	pk_campus	CREATE UNIQUE INDEX pk_campus ON public.campus USING btree (campus_id)
4	public	club	pk_club	CREATE UNIQUE INDEX pk_club ON public.club USING btree (club_id)
5	public	course	idx_course_department	CREATE INDEX idx_course_department ON public.course USING btree (dept_id)
6	public	course	idx_course_name	CREATE INDEX idx_course_name ON public.course USING btree (course_name)
7	public	course	pk_course	CREATE UNIQUE INDEX pk_course ON public.course USING btree (course_id)
8	public	department	idx_department_faculty	CREATE INDEX idx_department_faculty ON public.department USING btree (faculty_id)
9	public	department	pk_department	CREATE UNIQUE INDEX pk_department ON public.department USING btree (dept_id)
10	public	departmentphone	idx_departmentphone_...	CREATE INDEX idx_departmentphone_dept ON public.departmentphone USING btree (dept_id)
11	public	departmentphone	pk_departmentphone	CREATE UNIQUE INDEX pk_departmentphone ON public.departmentphone USING btree (dept_id, phone_num)
12	public	enrollment	idx_enrollment_course	CREATE INDEX idx_enrollment_course ON public.enrollment USING btree (course_id)
13	public	enrollment	idx_enrollment_semester	CREATE INDEX idx_enrollment_semester ON public.enrollment USING btree (semester)
14	public	enrollment	idx_enrollment_student	CREATE INDEX idx_enrollment_student ON public.enrollment USING btree (student_id)
15	public	enrollment	pk_enrollment	CREATE UNIQUE INDEX pk_enrollment ON public.enrollment USING btree (student_id, course_id)
16	public	exam	idx_exam_course	CREATE INDEX idx_exam_course ON public.exam USING btree (course_id)
17	public	exam	pk_exam	CREATE UNIQUE INDEX pk_exam ON public.exam USING btree (course_id, exam_number)
18	public	faculty	idx_faculty_building	CREATE INDEX idx_faculty_building ON public.faculty USING btree (building_id)
19	public	faculty	pk_faculty	CREATE UNIQUE INDEX pk_faculty ON public.faculty USING btree (faculty_id)
20	public	person	idx_person_last_name	CREATE INDEX idx_person_last_name ON public.person USING btree (last_name)
21	public	person	pk_person	CREATE UNIQUE INDEX pk_person ON public.person USING btree (person_id)
22	public	person	uq_person_ssn	CREATE UNIQUE INDEX uq_person_ssn ON public.person USING btree (person_ssn)
23	public	personphone	idx_personphone_person	CREATE INDEX idx_personphone_person ON public.personphone USING btree (person_id)
24	public	personphone	pk_personphone	CREATE UNIQUE INDEX pk_personphone ON public.personphone USING btree (person_id, phone_num)
25	public	prerequisite	idx_prerequisite_course	CREATE INDEX idx_prerequisite_course ON public.prerequisite USING btree (course_id)
26	public	prerequisite	idx_prerequisite_prereq	CREATE INDEX idx_prerequisite_prereq ON public.prerequisite USING btree (prereq_course)
27	public	prerequisite	pk_prerequisite	CREATE UNIQUE INDEX pk_prerequisite ON public.prerequisite USING btree (course_id, prereq_course)
28	public	professor	idx_professor_supervisor	CREATE INDEX idx_professor_supervisor ON public.professor USING btree (supervisor_id)
29	public	professor	pk_professor	CREATE UNIQUE INDEX pk_professor ON public.professor USING btree (professor_id)
30	public	professorphone	idx_professorphone_prof	CREATE INDEX idx_professorphone_prof ON public.professorphone USING btree (professor_id)
31	public	professorphone	pk_professorphone	CREATE UNIQUE INDEX pk_professorphone ON public.professorphone USING btree (professor_id, phone_num)
32	public	room	idx_room_building	CREATE INDEX idx_room_building ON public.room USING btree (building_id)
33	public	room	pk_room	CREATE UNIQUE INDEX pk_room ON public.room USING btree (building_id, room_number)
34	public	scholarship	idx_scholarship_min_gpa	CREATE INDEX idx_scholarship_min_gpa ON public.scholarship USING btree (min_gpa)
35	public	scholarship	pk_scholarship	CREATE UNIQUE INDEX pk_scholarship ON public.scholarship USING btree (scholarship_id)
36	public	student	idx_student_major	CREATE INDEX idx_student_major ON public.student USING btree (major)
37	public	student	idx_student_mentor	CREATE INDEX idx_student_mentor ON public.student USING btree (mentor_id)
38	public	student	idx_student_transcript	CREATE INDEX idx_student_transcript ON public.student USING btree (transcript_id)
39	public	student	pk_student	CREATE UNIQUE INDEX pk_student ON public.student USING btree (student_id)
40	public	studentclub	idx_studentclub_club	CREATE INDEX idx_studentclub_club ON public.studentclub USING btree (club_id)
41	public	studentclub	idx_studentclub_student	CREATE INDEX idx_studentclub_student ON public.studentclub USING btree (student_id)
42	public	studentclub	pk_studentclub	CREATE UNIQUE INDEX pk_studentclub ON public.studentclub USING btree (student_id, club_id)
43	public	studentinfo	idx_studentinfo_student	CREATE INDEX idx_studentinfo_student ON public.studentinfo USING btree (student_id)
44	public	studentinfo	pk_studentinfo	CREATE UNIQUE INDEX pk_studentinfo ON public.studentinfo USING btree (student_id, student_info)
45	public	studentscholarsh...	idx_studentschol_sch	CREATE INDEX idx_studentschol_sch ON public.studentscholarship USING btree (scholarship_id)
46	public	studentscholarsh...	idx_studentschol_stude...	CREATE INDEX idx_studentschol_student ON public.studentscholarship USING btree (student_id)
47	public	studentscholarsh...	pk_studentscholarship	CREATE UNIQUE INDEX pk_studentscholarship ON public.studentscholarship USING btree (student_id, scholarship_i...
48	public	teaches	idx_teaches_course	CREATE INDEX idx_teaches_course ON public.teaches USING btree (course_id)
49	public	teaches	idx_teaches_professor	CREATE INDEX idx_teaches_professor ON public.teaches USING btree (professor_id)
50	public	teaches	pk_teaches	CREATE UNIQUE INDEX pk_teaches ON public.teaches USING btree (professor_id, course_id)
51	public	transcript	pk_transcript	CREATE UNIQUE INDEX pk_transcript ON public.transcript USING btree (transcript_id)

Total rows: 51 Query complete 00:00:00.048

- Check Indexes for Specific Important Tables:
    - For Student Table:

```sql
SELECT
  indexname,
  indexdef
FROM
  pg_indexes
WHERE
  tablename = 'student';
```

Data Output | Messages | Notifications

	indexname name	indexdef text
1	pk_student	CREATE UNIQUE INDEX pk_student ON public.student USING btree (student_id)
2	idx_student_transcript	CREATE INDEX idx_student_transcript ON public.student USING btree (transcript_id)
3	idx_student_mentor	CREATE INDEX idx_student_mentor ON public.student USING btree (mentor_id)
4	idx_student_major	CREATE INDEX idx_student_major ON public.student USING btree (major)

- For Professor Table:

```sql
SELECT
  indexname,
  indexdef
FROM
  pg_indexes
WHERE
  tablename = 'professor';
```

Data Output | Messages | Notifications

	indexname name	indexdef text
1	pk_professor	CREATE UNIQUE INDEX pk_professor ON public.professor USING btree (professor_id)
2	idx_professor_supervisor	CREATE INDEX idx_professor_supervisor ON public.professor USING btree (supervisor_id)

- For Enrollment Table:

```sql
SELECT
  indexname,
  indexdef
FROM
  pg_indexes
WHERE
  tablename = 'enrollment';
```

Data Output | Messages | Notifications

	indexname name	indexdef text
1	pk_enrollment	CREATE UNIQUE INDEX pk_enrollment ON public.enrollment USING btree (student_id, course_id)
2	idx_enrollment_student	CREATE INDEX idx_enrollment_student ON public.enrollment USING btree (student_id)
3	idx_enrollment_course	CREATE INDEX idx_enrollment_course ON public.enrollment USING btree (course_id)
4	idx_enrollment_semester	CREATE INDEX idx_enrollment_semester ON public.enrollment USING btree (semester)

- For Course Table:

```sql
SELECT
  indexname,
  indexdef
FROM
  pg_indexes
WHERE
  tablename = 'course';
```

Data Output | Messages | Notifications

	indexname name	indexdef text
1	pk_course	CREATE UNIQUE INDEX pk_course ON public.course USING btree (course_id)
2	idx_course_department	CREATE INDEX idx_course_department ON public.course USING btree (dept_id)
3	idx_course_name	CREATE INDEX idx_course_name ON public.course USING btree (course_nam...

- For Scholarship Table:

```sql
SELECT
  indexname,
  indexdef
FROM
  pg_indexes
WHERE
  tablename = 'scholarship';
```

Data Output | Messages | Notifications

	indexname name	indexdef text
1	pk_scholarship	CREATE UNIQUE INDEX pk_scholarship ON public.scholarship USING btree (scholarship_id)
2	idx_scholarship_min_gpa	CREATE INDEX idx_scholarship_min_gpa ON public.scholarship USING btree (min_gpa)

- Confirm Primary Key Indexes Exist (Bonus Validation):

```sql
SELECT
  conname AS constraint_name,
  contype AS constraint_type,
  conrelid::regclass AS table_name
FROM
  pg_constraint
WHERE
  contype = 'p'
ORDER BY
  table_name;
```

Data Output | Messages | Notifications

	constraint_name name	constraint_type "char"	table_name regclass
1	pg_default_acl_oid_index	p	pg_default_acl
2	pg_tablespace_oid_index	p	pg_tablespace
3	pg_type_oid_index	p	pg_type
4	pg_attribute_relid_attnum_index	p	pg_attribute
5	pg_proc_oid_index	p	pg_proc
6	pg_class_oid_index	p	pg_class
7	pg_authid_oid_index	p	pg_authid
8	pg_auth_members_oid_index	p	pg_auth_members
9	pg_database_oid_index	p	pg_database
10	pg_foreign_server_oid_index	p	pg_foreign_server
11	pg_user_mapping_oid_index	p	pg_user_mapping
12	pg_sequence_seqrelid_index	p	pg_sequence
13	pg_foreign_data_wrapper_oid_index	p	pg_foreign_data_wrapper
14	pg_shdescription_o_c_index	p	pg_shdescription
15	pg_aggregate_fnoid_index	p	pg_aggregate
16	pg_am_oid_index	p	pg_am
17	pg_amop_oid_index	p	pg_amop
18	pg_amproc_oid_index	p	pg_amproc

Total rows: 84 | Query complete 00:00:00.062

- Test if an Index is Actually Used (e.g. Search by last name to see if it uses the index):

```
52 EXPLAIN ANALYZE
53 SELECT * FROM Person WHERE last_name = 'Smith';
```

Data Output | Messages | Notifications

	QUERY PLAN text
1	Seq Scan on person (cost=0.00..1.31 rows=1 width=402) (actual time=0.012..0.014 rows=1 loops=...
2	Filter: ((last_name)::text = 'Smith'::text)
3	Rows Removed by Filter: 24
4	Planning Time: 1.653 ms
5	Execution Time: 0.028 ms

Views, Stored Procedures and Triggers

As part of the database functional enhancement for the UHO University Registration System, a series of views, stored procedures, and triggers were developed to support operational efficiency, enforce business rules, and maintain data integrity at the system level.

```
-- ========================================
-- 1) VIEWS
-- ========================================

-- View 1: student_course_counts
--   Shows for each student how many courses they're enrolled in,
--   so we can quickly spot anyone outside the 4-6 course rule.
CREATE OR REPLACE VIEW student_course_counts AS
SELECT
  s.student_id,
  p.first_name || ' ' || p.last_name AS student_name,
  COUNT(e.course_id) AS course_count
FROM Student       s
JOIN Person        p ON s.student_id = p.person_id
LEFT JOIN Enrollment e ON e.student_id = s.student_id
GROUP BY s.student_id, p.first_name, p.last_name;
-- Justification:  helps enforce the business rule “each student must take 4-6 courses.”

-- View 2: course_enrollment_stats
--   Shows for each course its name and current enrollment size,
--   so we can watch the 12-40 student-per-course constraint.
CREATE OR REPLACE VIEW course_enrollment_stats AS
SELECT
  c.course_id,
  c.course_name,
  COUNT(e.student_id) AS enrolled_students
FROM Course        c
LEFT JOIN Enrollment e ON e.course_id = c.course_id
GROUP BY c.course_id, c.course_name;
-- Justification: monitors class sizes against the policy “12-40 students per course.”
```

```
SELECT * FROM student_course_counts;
SELECT * FROM course_enrollment_stats;

-- ========================================
-- 2) STORED PROCEDURES
-- ========================================

-- 2a) sp_promote_professor
--   Simple update: change a professor's rank.
CREATE OR REPLACE PROCEDURE sp_promote_professor(
  IN  p_professor_id INT,
  IN  p_new_rank      VARCHAR
)
LANGUAGE plpgsql
AS $$
BEGIN
  UPDATE Professor
    SET professor_rank = p_new_rank
    WHERE professor_id = p_professor_id;
  IF NOT FOUND THEN
    RAISE EXCEPTION 'Professor % not found', p_professor_id;
  END IF;
END;
$$;

-- 2b) sp_award_scholarship
--   Inserts or updates a scholarship award.
CREATE OR REPLACE PROCEDURE sp_award_scholarship(
  IN p_student_id     INT,
  IN p_scholarship_id INT,
  IN p_amount_pct     NUMERIC
)
LANGUAGE plpgsql
AS $$
BEGIN
  -- upsert style
  LOOP
    UPDATE StudentScholarship
      SET amount_received_percentage = LEAST(100, p_amount_pct),
          date_received = CURRENT_DATE
      WHERE student_id = p_student_id
        AND scholarship_id = p_scholarship_id;
    IF FOUND THEN
      RETURN;
    END IF;
    BEGIN
      INSERT          INTO          StudentScholarship(student_id,          scholarship_id,
amount_received_percentage, date_received)
      VALUES (p_student_id, p_scholarship_id, LEAST(100, p_amount_pct), CURRENT_DATE);
      RETURN;
```

```
    EXCEPTION WHEN unique_violation THEN
      -- concurrent insert, retry the update
      CONTINUE;
    END;
  END LOOP;
END;
$$;

-- 2c) sp_bulk_enroll_students
--   Takes a course_id and an array of student_ids, enrolls each one.
--   Demonstrates a PL/pgSQL loop.
CREATE OR REPLACE PROCEDURE sp_bulk_enroll_students(
  IN p_course_id   INT,
  IN p_student_ids INT[]
)
LANGUAGE plpgsql
AS $$
DECLARE
  sid INT;
BEGIN
  FOREACH sid IN ARRAY p_student_ids
  LOOP
    PERFORM 1
      FROM Enrollment
     WHERE course_id = p_course_id
       AND student_id = sid;
    IF NOT FOUND THEN
      INSERT INTO Enrollment(student_id, course_id, semester, enrollment_date)
      VALUES (sid, p_course_id, 'Fall2025', CURRENT_DATE);
    END IF;
  END LOOP;
END;
$$;

-- Promote professor #22
CALL sp_promote_professor(22, 'Full');
-- Award scholarship #2 to student #5 for 80%
CALL sp_award_scholarship(5, 2, 80);
-- Bulk enroll students [7,8,9] into course #3
CALL sp_bulk_enroll_students(3, ARRAY[7,8,9]);

-- ========================================
-- 3) TRIGGER & TRIGGER-FUNCTION
--    Enforce: no student > 6 courses, no course > 40 students.
-- ========================================

-- 3a) Trigger-function
CREATE OR REPLACE FUNCTION trg_check_enrollment_limits()
RETURNS TRIGGER
LANGUAGE plpgsql
```

```
AS $$
DECLARE
  stu_count INT;
  crs_count INT;
BEGIN
  -- count current enrollments for that student
  SELECT COUNT(*) INTO stu_count
    FROM Enrollment
   WHERE student_id = NEW.student_id;
  IF stu_count >= 6 THEN
    RAISE EXCEPTION 'Enrollment limit exceeded: student % has % courses',
      NEW.student_id, stu_count;
  END IF;

  -- count current enrollment for that course
  SELECT COUNT(*) INTO crs_count
    FROM Enrollment
   WHERE course_id = NEW.course_id;
  IF crs_count >= 40 THEN
    RAISE EXCEPTION 'Course capacity reached: course % has % students',
      NEW.course_id, crs_count;
  END IF;

  RETURN NEW;
END;
$$;

-- 3b) Trigger binding
CREATE TRIGGER trg_enrollment_limits
  BEFORE INSERT ON Enrollment
  FOR EACH ROW
  EXECUTE FUNCTION trg_check_enrollment_limits();

-- Justification:
-- This BEFORE INSERT trigger guarantees at the database level that no student
-- can exceed 6 enrollments and no course can exceed 40 students, enforcing
-- critical business rules automatically on data modification.

INSERT INTO Enrollment(student_id, course_id, semester, enrollment_date) VALUES (1, 7,
'Fall2025', CURRENT_DATE);
```

Two database views were created to provide dynamic, real-time insights into critical academic metrics. The first view, student_course_counts, aggregates enrollment information for each student, displaying their full name alongside the number of courses they are registered in. This view plays an essential role in monitoring compliance with institutional policies requiring students to maintain an academic load within a specified range (typically four to six courses). The second view, course_enrollment_stats, offers a summary of current enrollment numbers per course, allowing academic administrators to quickly identify classes that may be under- or over-subscribed relative to standard thresholds (usually between twelve and forty students per

course). Both views serve to simplify monitoring processes that would otherwise require complex and repetitive manual queries.

In addition to views, three stored procedures were implemented to automate routine administrative tasks. The sp_promote_professor procedure facilitates the updating of a professor's rank based on their identifier, ensuring that academic promotions can be processed directly through the database with minimal risk of human error. The sp_award_scholarship procedure handles the awarding of scholarships to students using an "upsert" logic that either inserts a new record or updates an existing one, depending on the presence of prior awards. This procedure includes a loop structure to handle concurrent modifications safely, illustrating a practical use of control flows in database programming. Furthermore, the sp_bulk_enroll_students procedure was designed to enroll multiple students into a given course efficiently. By iterating over an array of student identifiers, this procedure inserts enrollment records while checking for duplicates, significantly reducing the administrative overhead associated with mass registrations during peak periods.

Finally, to enforce critical business rules automatically, a trigger named trg_enrollment_limits was developed and attached to the Enrollment table. This trigger, which fires before any insert operation, ensures that no student may enroll in more than six courses and that no course may exceed forty students. By embedding this enforcement directly at the database layer through a supporting trigger function, the system guarantees adherence to core academic policies regardless of the access method, whether through application interfaces or direct database manipulation. Controlled testing validated that the trigger effectively prevents policy violations, raising appropriate exceptions when limits are exceeded.

This set of views, procedures, and triggers collectively strengthens the UHO University Registration System by improving usability, operational speed, and the reliability of critical business rule enforcement, laying the groundwork for a highly stable and responsive academic database environment.

Testing Queries:

Test Views:

```
33  SELECT * FROM course_enrollment_stats;
```

Data Output | Messages | Notifications

	course_id integer	course_name character varying (100)	enrolled_students bigint
1	8	European History	0
2	10	Finance	0
3	9	Marketing 101	0
4	7	World History	0
5	1	Data Structures	1
6	5	Genetics	1
7	4	Fluid Mechanics	1
8	2	Algorithms	1
9	6	Microbiology	1
10	3	Thermodynamics	1

```
32  SELECT * FROM student_course_counts;
```

Data Output | Messages | Notifications

	student_id integer	student_name text	course_count bigint
1	16	Paul Quinn	0
2	9	Irene Johnson	0
3	6	Frank Green	1
4	13	Maya Nelson	0
5	15	Olivia Perez	0
6	14	Nathan OBrien	0
7	2	Bob Brown	1
8	8	Henry Irwin	0
9	1	Alice Smith	1
10	12	Leo Martinez	0
11	4	David Evans	1
12	18	Rachel Scott	0
13	3	Carol Davis	1
14	11	Karen Lee	0
15	19	Steve Turner	0
16	17	Quinn Roberts	0
17	5	Eva Frank	1
18	10	Jack King	0
19	7	Grace Hall	0
20	20	Tina Underwood	0

Test Stored Procedures

- Promote a professor:

```
117  -- Promote professor #22
118  CALL sp_promote_professor(22, 'Full');
119  -- Verify result:
120  SELECT professor_id, professor_rank
121  FROM Professor
122  WHERE professor_id = 22;
```

Data Output | Messages | Notifications

Showing rows: 1 to 1 | Page No: 1 of 1

	professor_id [PK] integer	professor_rank character varying (50)
1	22	Full

- Award scholarship:

```sql
-- Award scholarship #2 to student #5 for 80%
CALL sp_award_scholarship(5, 2, 80);
-- Verify result:
SELECT student_id, scholarship_id, amount_received_percentage
FROM StudentScholarship
WHERE student_id = 5 AND scholarship_id = 2;
```

Data Output | Messages | Notifications

Showing rows: 1 to 1 | Page No: 1 of 1

	student_id [PK] integer	scholarship_id [PK] integer	amount_received_percentage numeric (5,2)
1	5	2	80.00

- Bulk enroll students into course:

```sql
-- Bulk enroll students [7,8,9] into course #3
CALL sp_bulk_enroll_students(3, ARRAY[7,8,9]);
-- Verify result:
SELECT student_id, course_id, semester
FROM Enrollment
WHERE course_id = 3
ORDER BY student_id;
```

Data Output | Messages | Notifications

Showing rows: 1 to 4 | Page No: 1 of 1

	student_id [PK] integer	course_id [PK] integer	semester character varying (10)
1	3	3	Spring2024
2	7	3	Fall2025
3	8	3	Fall2025
4	9	3	Fall2025

Test Trigger on Enrollment:

- Insert exceeding 6 courses for one student:

```sql
INSERT INTO Enrollment(student_id, course_id, semester, enrollment_date)
VALUES (1, 7, 'Fall2025', CURRENT_DATE);
```

Data Output | Messages | Notifications

```
ERROR:  Enrollment limit exceeded: student 1 has 6 courses
CONTEXT:  PL/pgSQL function trg_check_enrollment_limits() line 11 at RAISE

SQL state: P0001
```

Bonus: See All Triggers on Enrollment Table:

```
196 SELECT tgname
197 FROM pg_trigger
198 WHERE tgrelid = 'enrollment'::regclass;
```

Data Output | Messages | Notifications

Showing rows: 1 to 5 | Page No: 1 of 1

	tgname name
1	RI_ConstraintTrigger_c_17219
2	RI_ConstraintTrigger_c_17220
3	RI_ConstraintTrigger_c_17224
4	RI_ConstraintTrigger_c_17225
5	trg_enrollment_limits

Advanced Topics

This section details the implementation of two critical database script best practices: idempotency and logical organization. The script below demonstrates a robust approach to creating a database schema that can be executed multiple times without errors while maintaining clear dependencies between database objects.

```
-- ========================
-- Idempotent Db Script
-- ========================

-- ========================
-- 1. DROP SECTION
-- ========================

-- Drop Trigger Functions
DROP FUNCTION IF EXISTS trg_check_enrollment_limits() CASCADE;

-- Drop Triggers
DROP TRIGGER IF EXISTS trg_enrollment_limits ON Enrollment;

-- Drop Views
DROP VIEW IF EXISTS student_course_counts;
DROP VIEW IF EXISTS course_enrollment_stats;

-- Drop Stored Procedures
DROP PROCEDURE IF EXISTS sp_promote_professor;
DROP PROCEDURE IF EXISTS sp_award_scholarship;
DROP PROCEDURE IF EXISTS sp_bulk_enroll_students;

-- Drop Indexes
DROP INDEX IF EXISTS idx_student_transcript;
DROP INDEX IF EXISTS idx_student_mentor;
DROP INDEX IF EXISTS idx_professor_supervisor;
DROP INDEX IF EXISTS idx_building_campus;
```

```sql
DROP INDEX IF EXISTS idx_faculty_building;
DROP INDEX IF EXISTS idx_department_faculty;
DROP INDEX IF EXISTS idx_course_department;
DROP INDEX IF EXISTS idx_prerequisite_prereq;
DROP INDEX IF EXISTS idx_prerequisite_course;
DROP INDEX IF EXISTS idx_enrollment_student;
DROP INDEX IF EXISTS idx_enrollment_course;
DROP INDEX IF EXISTS idx_teaches_professor;
DROP INDEX IF EXISTS idx_teaches_course;
DROP INDEX IF EXISTS idx_studentclub_student;
DROP INDEX IF EXISTS idx_studentclub_club;
DROP INDEX IF EXISTS idx_studentschol_student;
DROP INDEX IF EXISTS idx_studentschol_sch;
DROP INDEX IF EXISTS idx_studentinfo_student;
DROP INDEX IF EXISTS idx_personphone_person;
DROP INDEX IF EXISTS idx_professorphone_prof;
DROP INDEX IF EXISTS idx_departmentphone_dept;
DROP INDEX IF EXISTS idx_room_building;
DROP INDEX IF EXISTS idx_exam_course;
DROP INDEX IF EXISTS idx_person_last_name;
DROP INDEX IF EXISTS idx_student_major;
DROP INDEX IF EXISTS idx_enrollment_semester;
DROP INDEX IF EXISTS idx_course_name;
DROP INDEX IF EXISTS idx_scholarship_min_gpa;

-- Drop Tables (reverse dependency order)
DROP TABLE IF EXISTS StudentScholarship CASCADE;
DROP TABLE IF EXISTS Scholarship CASCADE;
DROP TABLE IF EXISTS StudentClub CASCADE;
DROP TABLE IF EXISTS Club CASCADE;
DROP TABLE IF EXISTS Teaches CASCADE;
DROP TABLE IF EXISTS Enrollment CASCADE;
DROP TABLE IF EXISTS Prerequisite CASCADE;
DROP TABLE IF EXISTS Exam CASCADE;
DROP TABLE IF EXISTS Course CASCADE;
DROP TABLE IF EXISTS DepartmentPhone CASCADE;
DROP TABLE IF EXISTS Department CASCADE;
DROP TABLE IF EXISTS Faculty CASCADE;
DROP TABLE IF EXISTS Room CASCADE;
DROP TABLE IF EXISTS Building CASCADE;
DROP TABLE IF EXISTS Campus CASCADE;
DROP TABLE IF EXISTS ProfessorPhone CASCADE;
DROP TABLE IF EXISTS Professor CASCADE;
DROP TABLE IF EXISTS StudentInfo CASCADE;
DROP TABLE IF EXISTS Student CASCADE;
DROP TABLE IF EXISTS Transcript CASCADE;
DROP TABLE IF EXISTS PersonPhone CASCADE;
DROP TABLE IF EXISTS Person CASCADE;

-- ========================
```

```sql
-- 2. RECREATION SECTION
-- ========================

-- 2a) CREATE TABLES

-- 1. PERSON
CREATE TABLE Person (
  person_id      SERIAL          NOT NULL,
  person_ssn     VARCHAR(11)     NOT NULL DEFAULT 'XXX-XX-XXXX',
  first_name     VARCHAR(50)     NOT NULL,
  middle_name    VARCHAR(50),
  last_name      VARCHAR(50)     NOT NULL,
  date_of_birth  DATE            NOT NULL,
  CONSTRAINT PK_Person PRIMARY KEY (person_id),
  CONSTRAINT UQ_Person_SSN UNIQUE (person_ssn),
  CONSTRAINT CHK_Person_dob_past CHECK (date_of_birth <= CURRENT_DATE)
);

-- 2. PERSONPHONE
CREATE TABLE PersonPhone (
  person_id  INT   NOT NULL,
  phone_num  VARCHAR(24) NOT NULL DEFAULT '000-000-000',
  CONSTRAINT PK_PersonPhone PRIMARY KEY (person_id, phone_num),
  CONSTRAINT FK_PersonPhone_Person FOREIGN KEY (person_id) REFERENCES Person(person_id) ON
DELETE CASCADE
);

-- 3. TRANSCRIPT
CREATE TABLE Transcript (
  transcript_id   SERIAL  NOT NULL,
  creation_date   DATE    NOT NULL DEFAULT CURRENT_DATE,
  CONSTRAINT PK_Transcript PRIMARY KEY (transcript_id)
);

-- 4. STUDENT
CREATE TABLE Student (
  student_id     INT      NOT NULL,
  transcript_id  INT      NOT NULL,
  major          VARCHAR(50) NOT NULL DEFAULT 'Undeclared',
  mentor_id      INT,
  CONSTRAINT PK_Student PRIMARY KEY (student_id),
  CONSTRAINT FK_Student_Person FOREIGN KEY (student_id) REFERENCES Person(person_id) ON
DELETE CASCADE,
  CONSTRAINT FK_Student_Transcript FOREIGN KEY (transcript_id) REFERENCES
Transcript(transcript_id) ON DELETE CASCADE,
  CONSTRAINT FK_Student_Mentor FOREIGN KEY (mentor_id) REFERENCES Student(student_id) ON
DELETE SET NULL
);
```

```sql
-- 5. STUDENTINFO
CREATE TABLE StudentInfo (
  student_id    INT      NOT NULL,
  student_info  VARCHAR(100) NOT NULL DEFAULT 'No Information',
  CONSTRAINT PK_StudentInfo PRIMARY KEY (student_id, student_info),
  CONSTRAINT FK_StudentInfo_Student FOREIGN KEY (student_id) REFERENCES
Student(student_id) ON DELETE CASCADE
);

-- 6. PROFESSOR
CREATE TABLE Professor (
  professor_id   INT      NOT NULL,
  professor_rank VARCHAR(50) NOT NULL DEFAULT 'Undeclared',
  hire_date      DATE     NOT NULL DEFAULT CURRENT_DATE,
  supervisor_id  INT,
  CONSTRAINT PK_Professor PRIMARY KEY (professor_id),
  CONSTRAINT FK_Professor_Person FOREIGN KEY (professor_id) REFERENCES Person(person_id)
ON DELETE CASCADE,
  CONSTRAINT FK_Professor_Supervisor FOREIGN KEY (supervisor_id) REFERENCES
Professor(professor_id) ON DELETE SET NULL,
  CONSTRAINT CHK_Professor_hire_date_past CHECK (hire_date <= CURRENT_DATE)
);

-- 7. PROFESSORPHONE
CREATE TABLE ProfessorPhone (
  professor_id INT    NOT NULL,
  phone_num    VARCHAR(24) NOT NULL DEFAULT '000-000-000',
  CONSTRAINT PK_ProfessorPhone PRIMARY KEY (professor_id, phone_num),
  CONSTRAINT FK_ProfessorPhone_Professor FOREIGN KEY (professor_id) REFERENCES
Professor(professor_id) ON DELETE CASCADE
);

-- 8. CAMPUS
CREATE TABLE Campus (
  campus_id       SERIAL     NOT NULL,
  campus_name     VARCHAR(100) NOT NULL,
  campus_location VARCHAR(100) NOT NULL,
  CONSTRAINT PK_Campus PRIMARY KEY (campus_id)
);

-- 9. BUILDING
CREATE TABLE Building (
  building_id       SERIAL     NOT NULL,
  building_name     VARCHAR(100) NOT NULL,
  building_location VARCHAR(100),
  campus_id         INT         NOT NULL,
  CONSTRAINT PK_Building PRIMARY KEY (building_id),
  CONSTRAINT FK_Building_Campus FOREIGN KEY (campus_id) REFERENCES Campus(campus_id) ON
DELETE CASCADE
);
```

```sql
-- 10. ROOM
CREATE TABLE Room (
  building_id INT     NOT NULL,
  room_number VARCHAR(10) NOT NULL,
  nb_seats    INT     NOT NULL DEFAULT 30,
  CONSTRAINT PK_Room PRIMARY KEY (building_id, room_number),
  CONSTRAINT FK_Room_Building FOREIGN KEY (building_id) REFERENCES Building(building_id)
ON DELETE CASCADE,
  CONSTRAINT CHK_Room_seats_positive CHECK (nb_seats > 0)
);

-- 11. FACULTY
CREATE TABLE Faculty (
  faculty_id   SERIAL      NOT NULL,
  faculty_name VARCHAR(100) NOT NULL,
  building_id  INT          NOT NULL,
  CONSTRAINT PK_Faculty PRIMARY KEY (faculty_id),
  CONSTRAINT FK_Faculty_Building FOREIGN KEY (building_id) REFERENCES
Building(building_id) ON DELETE CASCADE
);

-- 12. DEPARTMENT
CREATE TABLE Department (
  dept_id    SERIAL      NOT NULL,
  dept_name  VARCHAR(100) NOT NULL,
  faculty_id INT          NOT NULL,
  CONSTRAINT PK_Department PRIMARY KEY (dept_id),
  CONSTRAINT FK_Department_Faculty FOREIGN KEY (faculty_id) REFERENCES Faculty(faculty_id)
ON DELETE CASCADE
);

-- 13. DEPARTMENTPHONE
CREATE TABLE DepartmentPhone (
  dept_id   INT    NOT NULL,
  phone_num VARCHAR(24) NOT NULL DEFAULT '000-000-000',
  CONSTRAINT PK_DepartmentPhone PRIMARY KEY (dept_id, phone_num),
  CONSTRAINT FK_DepartmentPhone_Department FOREIGN KEY (dept_id) REFERENCES
Department(dept_id) ON DELETE CASCADE
);

-- 14. COURSE
CREATE TABLE Course (
  course_id    SERIAL      NOT NULL,
  course_name  VARCHAR(100) NOT NULL,
  credit_hours INT          NOT NULL,
  dept_id      INT          NOT NULL,
  CONSTRAINT PK_Course PRIMARY KEY (course_id),
  CONSTRAINT FK_Course_Department FOREIGN KEY (dept_id) REFERENCES Department(dept_id) ON
DELETE CASCADE,
```

```
  CONSTRAINT CHK_Course_credit_positive CHECK (credit_hours > 0)
);
-- 15. EXAM
CREATE TABLE Exam (
  course_id    INT           NOT NULL,
  exam_number  VARCHAR(10)   NOT NULL,
  exam_date    DATE          NOT NULL,
  exam_location VARCHAR(50),
  exam_weight  DECIMAL(5,2)  NOT NULL,
  CONSTRAINT PK_Exam PRIMARY KEY (course_id, exam_number),
  CONSTRAINT FK_Exam_Course FOREIGN KEY (course_id) REFERENCES Course(course_id) ON DELETE
CASCADE,
  CONSTRAINT CHK_Exam_weight_range CHECK (exam_weight >= 0 AND exam_weight <= 100)
);

-- 16. PREREQUISITE
CREATE TABLE Prerequisite (
  course_id      INT NOT NULL,
  prereq_course  INT NOT NULL,
  CONSTRAINT PK_Prerequisite PRIMARY KEY (course_id, prereq_course),
  CONSTRAINT FK_Prerequisite_Course FOREIGN KEY (course_id) REFERENCES Course(course_id)
ON DELETE CASCADE,
  CONSTRAINT FK_Prerequisite_Prereq FOREIGN KEY (prereq_course) REFERENCES
Course(course_id) ON DELETE CASCADE
);

-- 17. ENROLLMENT
CREATE TABLE Enrollment (
  student_id      INT     NOT NULL,
  course_id       INT     NOT NULL,
  semester        VARCHAR(10) NOT NULL,
  enrollment_date DATE    NOT NULL DEFAULT CURRENT_DATE,
  final_grade     DECIMAL(4,2),
  CONSTRAINT PK_Enrollment PRIMARY KEY (student_id, course_id),
  CONSTRAINT FK_Enrollment_Student FOREIGN KEY (student_id) REFERENCES Student(student_id)
ON DELETE CASCADE,
  CONSTRAINT FK_Enrollment_Course FOREIGN KEY (course_id) REFERENCES Course(course_id) ON
DELETE CASCADE,
  CONSTRAINT CHK_Enrollment_date_past CHECK (enrollment_date <= CURRENT_DATE),
  CONSTRAINT CHK_Enrollment_grade_range CHECK (final_grade IS NULL OR (final_grade >= 0
AND final_grade <= 4))
);

-- 18. TEACHES
CREATE TABLE Teaches (
  professor_id   INT     NOT NULL,
  course_id      INT     NOT NULL,
  course_timing  VARCHAR(50)  NOT NULL,
  course_location VARCHAR(100) NOT NULL,
  CONSTRAINT PK_Teaches PRIMARY KEY (professor_id, course_id),
```

```
  CONSTRAINT FK_Teaches_Professor FOREIGN KEY (professor_id) REFERENCES
Professor(professor_id) ON DELETE CASCADE,
  CONSTRAINT FK_Teaches_Course FOREIGN KEY (course_id) REFERENCES Course(course_id) ON
DELETE CASCADE
);

-- 19. CLUB
CREATE TABLE Club (
  club_id   SERIAL      NOT NULL,
  club_name VARCHAR(100) NOT NULL,
  CONSTRAINT PK_Club PRIMARY KEY (club_id)
);

-- 20. STUDENTCLUB
CREATE TABLE StudentClub (
  student_id  INT          NOT NULL,
  club_id     INT          NOT NULL,
  position    VARCHAR(50),
  date_joined DATE DEFAULT CURRENT_DATE,
  CONSTRAINT PK_StudentClub PRIMARY KEY (student_id, club_id),
  CONSTRAINT FK_StudentClub_Student FOREIGN KEY (student_id) REFERENCES
Student(student_id) ON DELETE CASCADE,
  CONSTRAINT FK_StudentClub_Club FOREIGN KEY (club_id) REFERENCES Club(club_id) ON DELETE
CASCADE
);

-- 21. SCHOLARSHIP
CREATE TABLE Scholarship (
  scholarship_id   SERIAL       NOT NULL,
  scholarship_name VARCHAR(100) NOT NULL,
  min_gpa          DECIMAL(3,2) DEFAULT 0.00,
  CONSTRAINT PK_Scholarship PRIMARY KEY (scholarship_id),
  CONSTRAINT CHK_Scholarship_min_gpa_range CHECK (min_gpa >= 0 AND min_gpa <= 4)
);

-- 22. STUDENTSCHOLARSHIP
CREATE TABLE StudentScholarship (
  student_id     INT          NOT NULL,
  scholarship_id INT          NOT NULL,
  amount_received_percentage DECIMAL(5,2),
  date_received  DATE         NOT NULL DEFAULT CURRENT_DATE,
  CONSTRAINT PK_StudentScholarship PRIMARY KEY (student_id, scholarship_id),
  CONSTRAINT FK_StudentScholarship_Student FOREIGN KEY (student_id) REFERENCES
Student(student_id) ON DELETE CASCADE,
  CONSTRAINT FK_StudentScholarship_Scholarship FOREIGN KEY (scholarship_id) REFERENCES
Scholarship(scholarship_id) ON DELETE CASCADE,
  CONSTRAINT CHK_StudSchol_amount_pct CHECK (amount_received_percentage >= 0 AND
amount_received_percentage <= 100)
);
```

```sql
-- 1) Persons (serial person_id: 1–25)
INSERT INTO Person (person_ssn, first_name, middle_name, last_name, date_of_birth) VALUES
  ('111-11-1111','Alice', NULL,     'Smith',     '1990-04-15'),
  ('222-22-2222','Bob',   'J.',     'Brown',     '1988-07-23'),
  ('333-33-3333','Carol', NULL,     'Davis',     '1992-12-02'),
  ('444-44-4444','David', NULL,     'Evans',     '1991-03-10'),
  ('555-55-5555','Eva',   NULL,     'Frank',     '1993-09-30'),
  ('666-66-6666','Frank', NULL,     'Green',     '1987-05-25'),
  ('777-77-7777','Grace', NULL,     'Hall',      '1994-11-12'),
  ('888-88-8888','Henry', NULL,     'Irwin',     '1989-02-20'),
  ('999-99-9999','Irene', NULL,     'Johnson',   '1995-06-18'),
  ('101-01-0101','Jack',  NULL,     'King',      '1990-12-01'),
  ('121-21-2121','Karen', NULL,     'Lee',       '1992-01-09'),
  ('131-31-3131','Leo',   NULL,     'Martinez',  '1991-08-23'),
  ('141-41-4141','Maya',  NULL,     'Nelson',    '1993-04-04'),
  ('151-51-5151','Nathan',NULL,     'OBrien',    '1988-10-14'),
  ('161-61-6161','Olivia',NULL,     'Perez',     '1994-07-07'),
  ('171-71-7171','Paul',  NULL,     'Quinn',     '1989-09-09'),
  ('181-81-8181','Quinn', NULL,     'Roberts',   '1990-02-02'),
  ('191-91-9191','Rachel',NULL,     'Scott',     '1992-05-05'),
  ('202-02-0202','Steve', NULL,     'Turner',    '1987-12-12'),
  ('212-12-1212','Tina',  NULL,     'Underwood', '1991-03-03'),
  ('313-13-1313','Uma',   NULL,     'Vargas',    '1975-05-20'),
  ('414-14-1414','Victor',NULL,     'White',     '1972-02-12'),
  ('515-15-1515','Wendy', NULL,     'Xu',        '1980-07-07'),
  ('616-16-1616','Xavier',NULL,     'Young',     '1978-11-11'),
  ('717-17-1717','Zoe',   NULL,     'Zhang',     '1982-03-03');

-- 2) PersonPhone
INSERT INTO PersonPhone (person_id, phone_num) VALUES
  (1,'555-0001'), (2,'555-0002'), (3,'555-0003');

-- 3) Transcripts (IDs 1–20)
INSERT INTO Transcript DEFAULT VALUES;  -- repeat 20 times
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
```

```sql
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;
INSERT INTO Transcript DEFAULT VALUES;

-- 4) Students (IDs 1–20 → person 1–20, transcripts 1–20)
INSERT INTO Student (student_id, transcript_id, major, mentor_id) VALUES
  (1,1,'Computer Science',   NULL),
  (2,2,'Mathematics',        1),
  (3,3,'Physics',            1),
  (4,4,'Engineering',        2),
  (5,5,'Biology',            3),
  (6,6,'Chemistry',          4),
  (7,7,'History',            5),
  (8,8,'Art',                6),
  (9,9,'Music',              7),
  (10,10,'Economics',        8),
  (11,11,'Philosophy',       9),
  (12,12,'Psychology',       10),
  (13,13,'Sociology',        11),
  (14,14,'Literature',       12),
  (15,15,'Geology',          13),
  (16,16,'Astronomy',        14),
  (17,17,'Comp Eng',         15),
  (18,18,'Elec Eng',         16),
  (19,19,'Mech Eng',         17),
  (20,20,'Civil Eng',        18);

-- 5) StudentInfo
INSERT INTO StudentInfo (student_id, student_info) VALUES
  (1,'GPA:3.50'), (2,'GPA:3.75'), (3,'GPA:3.20'), (4,'GPA:3.80'),
  (5,'GPA:3.10'), (6,'GPA:3.40'), (7,'GPA:3.60'), (8,'GPA:3.30'),
  (9,'GPA:3.90'), (10,'GPA:3.45'), (11,'GPA:3.55'), (12,'GPA:3.25'),
  (13,'GPA:3.15'), (14,'GPA:3.85'), (15,'GPA:3.95'), (16,'GPA:3.05'),
  (17,'GPA:3.65'), (18,'GPA:3.70'), (19,'GPA:3.35'), (20,'GPA:3.00');

-- 6) Professors (IDs 21–25 → persons 21–25)
INSERT INTO Professor (professor_id, professor_rank, hire_date, supervisor_id) VALUES
  (21,'Assistant', '2015-08-01', NULL),
  (22,'Associate', '2010-06-15', 21),
  (23,'Full',      '2005-09-01', 22),
  (24,'Assistant', '2018-01-20', 21),
  (25,'Associate', '2012-11-30', 23);

-- 7) ProfessorPhone
INSERT INTO ProfessorPhone (professor_id, phone_num) VALUES
  (21,'555-1001'), (22,'555-1002'), (23,'555-1003'),
  (24,'555-1004'), (25,'555-1005');

-- 8) Campuses
INSERT INTO Campus (campus_name, campus_location) VALUES
```

```
  ('Main Campus',     'City A'),
  ('Downtown Campus', 'City B'),
  ('West Campus',     'City C');

-- 9) Buildings
INSERT INTO Building (building_name, building_location, campus_id) VALUES
  ('Science Hall',     'North end',  1),
  ('Library',          'Center',     1),
  ('Engineering Bldg','East side',  2),
  ('Arts Building',    'West side',  2),
  ('Business Center',  'Downtown',   3);

-- 10) Rooms
INSERT INTO Room (building_id, room_number, nb_seats) VALUES
  (1,'101',30),(1,'102',25),
  (2,'201',40),(2,'202',35),
  (3,'301',50),(3,'302',45),
  (4,'401',20),(4,'402',15),
  (5,'501',60),(5,'502',55);

-- 11) Faculties
INSERT INTO Faculty (faculty_name, building_id) VALUES
  ('Science',     1),
  ('Engineering', 3),
  ('Arts',        4);

-- 12) Departments
INSERT INTO Department (dept_name, faculty_id) VALUES
  ('Computer Science',        2),
  ('Mechanical Engineering',  2),
  ('Biology',                 1),
  ('History',                 3),
  ('Business',                3);

-- 13) DepartmentPhone
INSERT INTO DepartmentPhone (dept_id, phone_num) VALUES
  (1,'555-2001'), (2,'555-2002'),
  (3,'555-2003'), (4,'555-2004'),
  (5,'555-2005');

-- 14) Courses
INSERT INTO Course (course_name, credit_hours, dept_id) VALUES
  ('Data Structures',   3, 1),
  ('Algorithms',        3, 1),
  ('Thermodynamics',    4, 2),
  ('Fluid Mechanics',   3, 2),
  ('Genetics',          3, 3),
  ('Microbiology',      3, 3),
  ('World History',     3, 4),
  ('European History',  3, 4),
```

```
  ('Marketing 101',     3, 5),
  ('Finance',           3, 5);

-- 15) Exams (2 each for courses 1–3)
INSERT INTO Exam (course_id, exam_number, exam_date, exam_location, exam_weight) VALUES
  (1,'Exam1','2024-05-01','Room 101',50),
  (1,'Exam2','2024-06-01','Room 102',50),
  (2,'Exam1','2024-05-02','Room 103',60),
  (2,'Exam2','2024-06-02','Room 104',40),
  (3,'Exam1','2024-05-03','Room 105',55),
  (3,'Exam2','2024-06-03','Room 106',45);

-- 16) Prerequisites
INSERT INTO Prerequisite (course_id, prereq_course) VALUES
  (2,1),   -- Algorithms ← Data Structures
  (4,3);   -- Fluid Mechanics ← Thermodynamics

-- 17) Enrollments (one per student 1–6; you can add more to reach 4–6 each)
INSERT INTO Enrollment (student_id, course_id, semester, enrollment_date, final_grade)
VALUES
  (1,1,'Spring2024','2024-01-10',NULL),
  (2,2,'Spring2024','2024-01-11',NULL),
  (3,3,'Spring2024','2024-01-12',NULL),
  (4,4,'Spring2024','2024-01-13',NULL),
  (5,5,'Spring2024','2024-01-14',NULL),
  (6,6,'Spring2024','2024-01-15',NULL);

-- 18) Teaches
INSERT INTO Teaches (professor_id, course_id, course_timing, course_location) VALUES
  (21,1,'MWF 9–10','Room 101'),
  (22,2,'TTh 10–11','Room 102'),
  (23,3,'MWF 11–12','Room 103'),
  (24,4,'TTh 1–2','Room 104'),
  (25,5,'MWF 2–3','Room 105');

-- 19) Clubs
INSERT INTO Club (club_name) VALUES
  ('Robotics'),
  ('Debate'),
  ('Chess');

-- 20) StudentClub
INSERT INTO StudentClub (student_id, club_id, position, date_joined) VALUES
  (1,1,'President','2024-01-15'),
  (2,1,NULL,'2024-02-01'),
  (3,2,'Secretary','2024-02-10'),
  (4,2,NULL,'2024-03-05'),
  (5,3,'Member','2024-01-20'),
  (6,3,NULL,'2024-02-15');
```

```sql
-- 21) Scholarships
INSERT INTO Scholarship (scholarship_name, min_gpa) VALUES
  ('Academic Excellence',3.50),
  ('Athletic Merit',     2.50),
  ('Need-Based',         NULL);

-- 22) StudentScholarship
INSERT INTO StudentScholarship (student_id, scholarship_id, amount_received_percentage,
date_received) VALUES
  (1,1,100,'2024-02-01'),
  (2,1, 50,'2024-03-01'),
  (3,2,100,'2024-02-15'),
  (4,3, 75,'2024-04-01');

-- 2b) FOREIGN KEY INDEXES
CREATE INDEX idx_student_transcript   ON Student(transcript_id);
CREATE INDEX idx_student_mentor       ON Student(mentor_id);
CREATE INDEX idx_professor_supervisor ON Professor(supervisor_id);
CREATE INDEX idx_building_campus      ON Building(campus_id);
CREATE INDEX idx_faculty_building     ON Faculty(building_id);
CREATE INDEX idx_department_faculty   ON Department(faculty_id);
CREATE INDEX idx_course_department    ON Course(dept_id);
CREATE INDEX idx_prerequisite_prereq  ON Prerequisite(prereq_course);
CREATE INDEX idx_prerequisite_course  ON Prerequisite(course_id);
CREATE INDEX idx_enrollment_student   ON Enrollment(student_id);
CREATE INDEX idx_enrollment_course    ON Enrollment(course_id);
CREATE INDEX idx_teaches_professor    ON Teaches(professor_id);
CREATE INDEX idx_teaches_course       ON Teaches(course_id);
CREATE INDEX idx_studentclub_student  ON StudentClub(student_id);
CREATE INDEX idx_studentclub_club     ON StudentClub(club_id);
CREATE INDEX idx_studentschol_student ON StudentScholarship(student_id);
CREATE INDEX idx_studentschol_sch     ON StudentScholarship(scholarship_id);
CREATE INDEX idx_studentinfo_student  ON StudentInfo(student_id);
CREATE INDEX idx_personphone_person   ON PersonPhone(person_id);
CREATE INDEX idx_professorphone_prof  ON ProfessorPhone(professor_id);
CREATE INDEX idx_departmentphone_dept  ON DepartmentPhone(dept_id);
CREATE INDEX idx_room_building         ON Room(building_id);
CREATE INDEX idx_exam_course           ON Exam(course_id);

-- 2c) PERFORMANCE INDEXES
CREATE INDEX idx_person_last_name      ON Person(last_name);
CREATE INDEX idx_student_major         ON Student(major);
CREATE INDEX idx_enrollment_semester   ON Enrollment(semester);
CREATE INDEX idx_course_name           ON Course(course_name);
CREATE INDEX idx_scholarship_min_gpa    ON Scholarship(min_gpa);

-- 2d) VIEWS
CREATE OR REPLACE VIEW student_course_counts AS
SELECT s.student_id, p.first_name || ' ' || p.last_name AS student_name,
COUNT(e.course_id) AS course_count
```

```
FROM Student s
JOIN Person p ON s.student_id = p.person_id
LEFT JOIN Enrollment e ON e.student_id = s.student_id
GROUP BY s.student_id, p.first_name, p.last_name;

CREATE OR REPLACE VIEW course_enrollment_stats AS
SELECT c.course_id, c.course_name, COUNT(e.student_id) AS enrolled_students
FROM Course c
LEFT JOIN Enrollment e ON e.course_id = c.course_id
GROUP BY c.course_id, c.course_name;

-- 2e) STORED PROCEDURES
CREATE OR REPLACE PROCEDURE sp_promote_professor(IN p_professor_id INT, IN p_new_rank
VARCHAR)
LANGUAGE plpgsql AS $$
BEGIN
  UPDATE Professor SET professor_rank = p_new_rank WHERE professor_id = p_professor_id;
  IF NOT FOUND THEN RAISE EXCEPTION 'Professor % not found', p_professor_id; END IF;
END;$$;

CREATE OR REPLACE PROCEDURE sp_award_scholarship(IN p_student_id INT, IN p_scholarship_id
INT, IN p_amount_pct NUMERIC)
LANGUAGE plpgsql AS $$
BEGIN
  LOOP
    UPDATE StudentScholarship SET amount_received_percentage = LEAST(100,p_amount_pct),
date_received = CURRENT_DATE
      WHERE student_id=p_student_id AND scholarship_id=p_scholarship_id;
    IF FOUND THEN RETURN; END IF;
    BEGIN
      INSERT INTO
StudentScholarship(student_id,scholarship_id,amount_received_percentage,date_received)
      VALUES(p_student_id,p_scholarship_id,LEAST(100,p_amount_pct),CURRENT_DATE);
      RETURN;
    EXCEPTION WHEN unique_violation THEN CONTINUE;
    END;
  END LOOP;
END;$$;

CREATE OR REPLACE PROCEDURE sp_bulk_enroll_students(IN p_course_id INT, IN p_student_ids
INT[])
LANGUAGE plpgsql AS $$
DECLARE sid INT;
BEGIN
  FOREACH sid IN ARRAY p_student_ids LOOP
    PERFORM 1 FROM Enrollment WHERE course_id=p_course_id AND student_id=sid;
    IF NOT FOUND THEN
      INSERT INTO Enrollment(student_id,course_id,semester,enrollment_date)
      VALUES(sid,p_course_id,'Fall2025',CURRENT_DATE);
    END IF;
```

```
  END LOOP;
END;$$;

-- 2f) TRIGGER FUNCTION & TRIGGER
CREATE OR REPLACE FUNCTION trg_check_enrollment_limits() RETURNS TRIGGER LANGUAGE plpgsql
AS $$
DECLARE stu_count INT; crs_count INT;
BEGIN
  SELECT COUNT(*) INTO stu_count FROM Enrollment WHERE student_id = NEW.student_id;
  IF stu_count >= 6 THEN RAISE EXCEPTION 'Student % already has %
enrollments',NEW.student_id,stu_count; END IF;
  SELECT COUNT(*) INTO crs_count FROM Enrollment WHERE course_id = NEW.course_id;
  IF crs_count >= 40 THEN RAISE EXCEPTION 'Course % full (%
students)',NEW.course_id,crs_count; END IF;
  RETURN NEW;
END;$$;

CREATE TRIGGER trg_enrollment_limits
  BEFORE INSERT ON Enrollment
  FOR EACH ROW
  EXECUTE FUNCTION trg_check_enrollment_limits();
```

Idempotent Script Design:

The script achieves idempotency by first removing all existing database objects in a controlled manner before recreating them. This approach ensures that the script can be executed multiple times without encountering errors due to pre-existing objects. The DROP SECTION is carefully structured to eliminate objects in reverse dependency order, beginning with trigger functions and cascading down to base tables. Each DROP statement incorporates the IF EXISTS clause, preventing failures when objects are absent. Additionally, the CASCADE option is applied where necessary to automatically remove dependent constraints and objects, thereby maintaining referential integrity.

Tables are dropped in reverse order of their creation to respect foreign key relationships. For instance, the StudentScholarship table, which depends on both Student and Scholarship, is dropped before its parent tables. This sequencing prevents conflicts that could arise if a referenced table were removed before its dependents. The script also follows a strict hierarchy when removing other database objects—triggers, views, stored procedures, and indexes are dropped before tables to avoid orphaned dependencies. While the script does not explicitly wrap operations in a transaction, it assumes an implicit transaction model where any failure would result in a rollback, ensuring no partial execution states persist.

Logical Organization of Database Objects

The script is structured to reflect the natural dependencies between database entities, progressing from foundational tables to higher-level constructs. The initial table creation phase begins with independent entities such as Person, which serves as the base for Student and Professor tables. Subsequent tables are

introduced in an order that ensures all referenced tables exist before they are needed for foreign key constraints. For example, Transcript is created before Student since the latter references it, and Department is established before Course to maintain referential integrity.

Following table creation, the script populates the database with sample data, again respecting dependencies to avoid constraint violations. Once the schema and data are in place, the script proceeds to optimize performance by creating indexes on foreign key columns, such as idx_student_mentor on the mentor_id field in the Student table. Additional performance-focused indexes are then applied to frequently queried columns, including idx_person_last_name and idx_enrollment_semester, ensuring efficient data retrieval.

The final phases of the script introduce views and stored procedures, which rely on the underlying table structure. Views like student_course_counts aggregate data from multiple tables, while stored procedures such as sp_award_scholarship encapsulate business logic for scholarship management. The trigger function trg_check_enrollment_limits is defined last, as it depends on the Enrollment table being fully instantiated. This hierarchical organization—from base tables to derived objects and procedural logic—ensures clarity and maintainability.

Conclusion

In this report, we have laid the foundation for the development of the UHO University Registration System by conducting a thorough requirements analysis and crafting a comprehensive conceptual design. Chapter 1 presented a detailed overview of the system's purpose, real-world relevance, and the rationale for its selection, alongside the business logic and design objectives that will guide the remainder of the project. Through a structured methodology, we established a clear roadmap for gathering requirements and designing the database.

In Chapter 2, we translated the identified requirements into a robust conceptual model, detailing all relevant entities, attributes, and relationships. The ER diagrams accurately capture complex business rules, including multivalued and derived attributes, weak entities, and intricate cardinalities, all while satisfying the project's technical criteria. This phase not only demonstrates our understanding of core database concepts but also sets the stage for the logical design and implementation phases that follow.

In Chapter 3, we transformed the Full Entity Relation Diagram into a Full Relational Schema for the logical database modeling part. Moreover, we included the granular entities and tables while displaying their attributes, as well as all types of relationships involved in our database application (one-to-one, one-to-many, many-to-many and unary relationships).

In Chapter 4, we implemented the transformed Relational Schema into an applicable Database in the Postgres RDBMS, where we defined and created the tables, their columns, and their constraints. We inserted data into

the newly created tables and properly indexed them for performance enhancement. Finally, we created views, stored procedures, and triggers in the database.

Moving forward, this solid conceptual groundwork will guide the development of a normalized schema, SQL implementation, and ultimately, a fully functional system that effectively supports UHO's academic operations.

Appendix

Symbols	Description
	**Strong Entity** Strong entities are also referred to as the parent entity as sometimes weak entities can rely on them. They also have primary keys distinguishing them from the other categories.
	**Weak Entity** The second category is the weak entity. They lose their meaning if a parent entity does not exist and they have no primary key.
	**Simple Attribute** Simple attributes can also represent only one value. They can either show a relation of one-to-many or many-to-many.
	**Weak Entity** These attributes, at times, do not exist in the database, but their values can be derived from the preexisting attributes. Things such as age, work experience, and academic tenures can be derived from quantified attributes.
	**Multivalued Attribute** Multivalued attributes, as the name suggests, can have more than one value. An example of such attributes would be emails, phone numbers, etc.
	**Strong Relationship** A strong relationship shows a direct link between two entities.
	**Weak Relationship** A weak relationship is between a weak entity and the parent entity.

Source: Edraw (https://www.edrawsoft.com/er-diagram-symbols.html)