Omer AL-mukhtar University

Department of Computer Science

First Assignment in

Advanced Database Systems

Student Name :

مريم عبدالمنعم سعيد محمد

Student ID :

262507

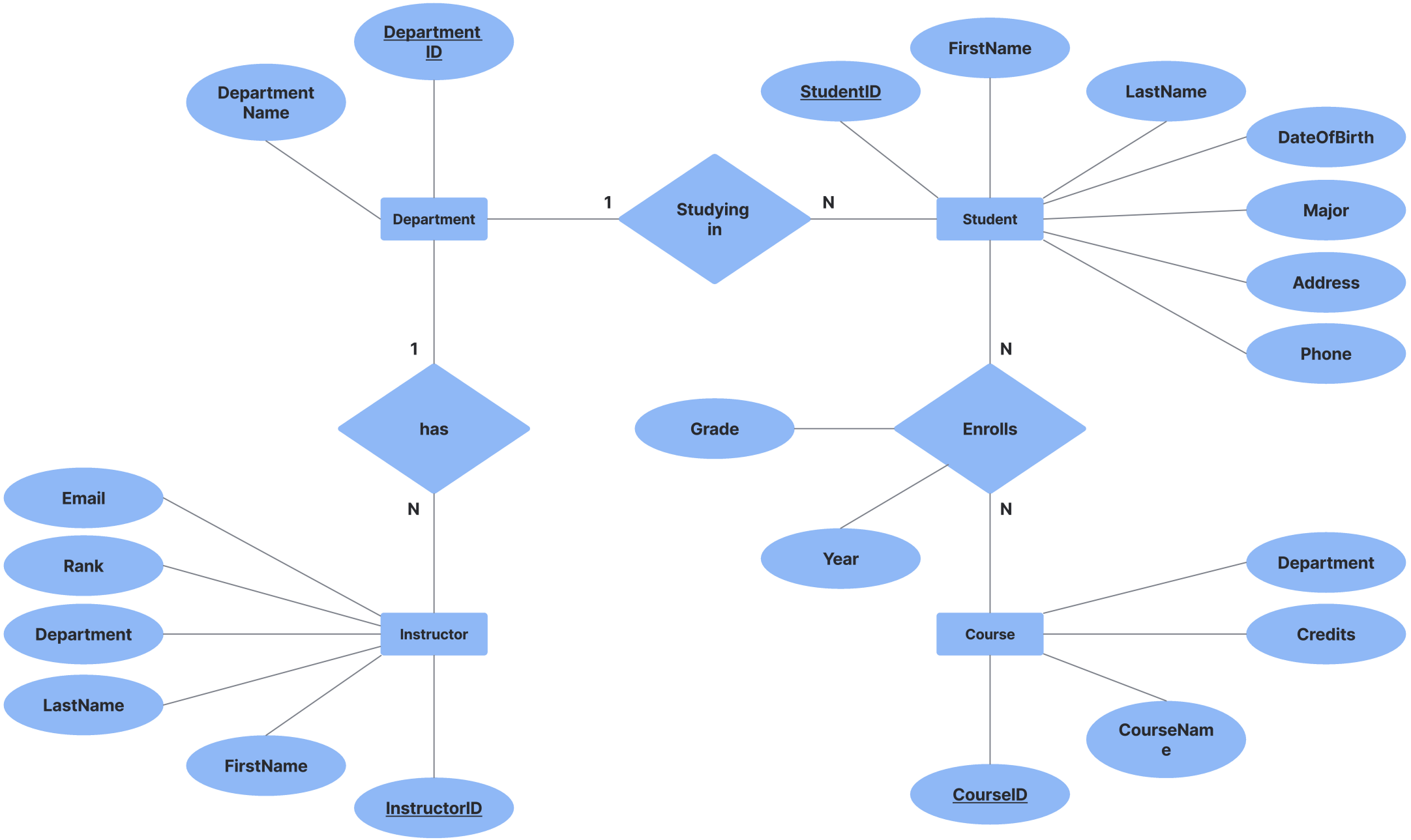
Department ID
Department Name
Department
1
Studying in
N
Student
StudentID
FirstName
LastName
DateOfBirth
Major
Address
Phone
1
has
N
N
Enrolls
Grade
Year
N
Email
Rank
Department
LastName
FirstName
Instructor
InstructorID
Course
Department
Credits
CourseName
CourseID

Diagram 1

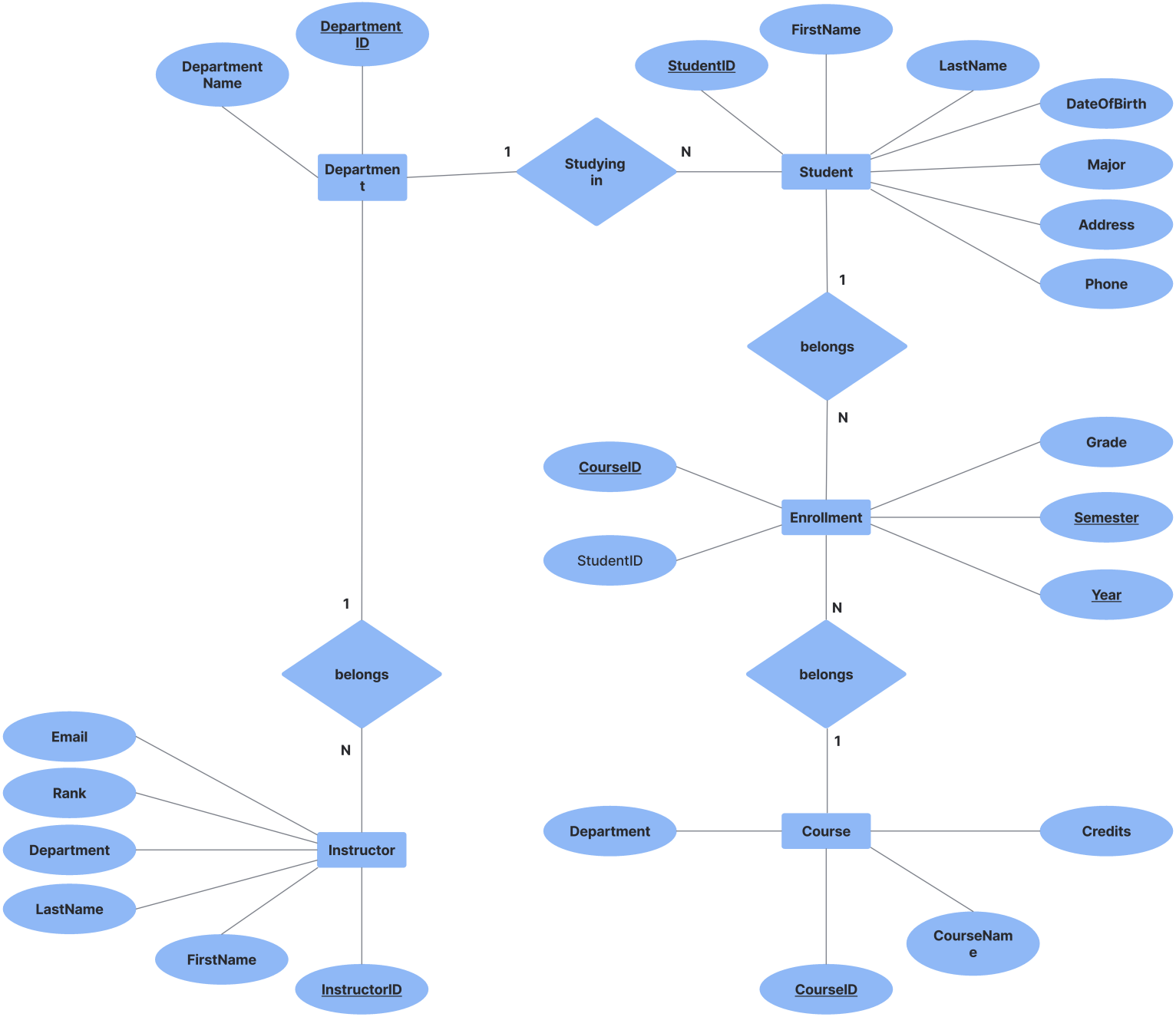
Department ID
Department Name
Department
1
Studying in
N
Student
StudentID
FirstName
LastName
DateOfBirth
Major
Address
Phone
1
belongs
N
CourseID
StudentID
Enrollment
Grade
Semester
Year
N
belongs
1
Department
Course
Credits
CourseNam e
CourseID
1
belongs
N
Email
Rank
Department
LastName
FirstName
Instructor
InstructorID

Diagram 2

1. ER Diagram

Justification for Using Two ER Diagrams

When modeling the given scenario with entities Student, Course, Instructor, and Enrollment, the Enrollment entity represents a many-to-many relationship between Student and Course. This situation can be effectively visualized and understood through two distinct ER diagrams, each serving a specific purpose:

- **Diagram 1: High-Level Overview**
    - This diagram presents a simplified, abstract view of the system.
    - It focuses on the primary entities (Student, Course, Instructor, and Department) and the main relationships between them.
    - The Enrollment entity is depicted as a relationship (often represented by a diamond) between Student and Course, emphasizing its role in connecting these two entities.
    - This high-level diagram is useful for communicating the overall structure of the database to stakeholders who may not need or understand the detailed technicalities.
    - It provides a clear understanding of how students enroll in courses, instructors are affiliated with departments, and students are associated with departments.
- **Diagram 2: Detailed Enrollment Focus**
    - This diagram zooms in on the Enrollment entity to provide a more detailed view.
    - It treats Enrollment as an entity in its own right, with its own attributes (StudentID,

CourseID, Semester, Year, and Grade).

- This detailed view is crucial for developers and database designers as it explicitly shows all the components of the Enrollment entity, which will likely translate into a separate table in the database.
- It clarifies the precise data that needs to be stored regarding each enrollment record.
- It aids in the accurate implementation of the database schema, ensuring that all necessary attributes are accounted for.

The ER diagram (as shown in the **Diagram 2**) illustrates the database schema for the university management system. It includes the following entities and relationships:

- **Entities:**
  - **Department:** Attributes include DepartmentID and DepartmentName.
  - **Student:** Attributes include StudentID, FirstName, LastName, DateOfBirth, Major, Address, and Phone.
  - **Course:** Attributes include CourseID, CourseName, Credits, and Department.
  - **Instructor:** Attributes include InstructorID, FirstName, LastName, Department, Rank, and Email.
- **Relationships:**
  - **Studying in:** A one-to-many relationship between Department and Student.

- **has:** A one-to-many relationship between Department and Instructor.
- **Enrolls:** A many-to-many relationship between Student and Course, with an associative entity (or a "weak entity" in the code) that includes the attribute Grade, Semester, and Year.

2. Mapping between the ER Diagram and the File Structure

The ER diagram is mapped to a set of CSV files, where each entity is represented by a separate CSV file. The many-to-many relationship "Enrolls" is also represented by a CSV file.

- **Department Entity:** Department.csv
- **Student Entity:** Student.csv
- **Course Entity:** Course.csv
- **Instructor Entity:** Instructor.csv
- **Enrolls Relationship:** Enrollment.csv

CSV files were chosen for their simplicity, portability, and ease of implementation. Their straightforward format is human-readable and compatible with various tools, including Python's Pandas. This facilitates development and data sharing, and they are suitable for handling moderate data volumes.

3. Data Structures and File Formats

- **Data Structures:** Python lists are used to store the data in memory before being written to CSV files. Pandas DataFrames are used for data manipulation, validation, and saving to CSV.
- **File Formats:** CSV (Comma-Separated Values) files are used to store the data. Each file represents an entity or a relationship from the ER diagram. The first row of each CSV file contains the column headers. The encoding used is 'utf-8-sig' to handle Arabic characters correctly.

4. Scalability and Potential Bottlenecks

- **Scalability:** The current implementation uses Pandas DataFrames, which can handle moderately large datasets. However, for very large datasets (millions of records), performance might degrade.
- **Potential Bottlenecks:**
    - **File I/O:** Reading and writing large CSV files can become a bottleneck.
    - **Pandas Operations:** Certain Pandas operations (e.g., searching, filtering, and updating) can be slow for very large DataFrames.
    - **Memory Usage:** Loading entire CSV files into memory might be a limitation for extremely large datasets.

- **Scaling Strategies:**

    Database System: For improved scalability, consider using a database management system (DBMS) like PostgreSQL or MySQL instead of CSV files. Databases are optimized for handling large datasets, indexing, and efficient querying.

5. Justification for the Chosen Programming Language

Python was chosen for the following reasons:

- **Ease of Use and Readability:** Python's syntax is clear and readable, making development faster and easier to maintain.
- **Libraries:** Python has powerful libraries like Pandas and Faker, which simplify data manipulation, CSV handling, and data generation. Pandas is particularly useful for working with structured data, and Faker is excellent for generating realistic fake data.
- **Rapid Development:** Python's features enable rapid development, which is crucial for this project.
- **Community and Support:** Python has a large and active community, providing ample support and resources.

6. Handling Referential Integrity

Referential integrity is partially handled in the code through validation and update/delete functions:

- **Validation:**
    - _validate_not_null(): Ensures that required fields are not null.
    - _validate_unique(): Ensures that certain fields (e.g., Email, StudentID, CourseID, InstructorID) are unique within their respective files.
    - _validate_credits(): Validates that course credits are within a valid range.
    - _validate_dob(): Validates the student's date of birth to ensure they are at least 17 years old.
    - _validate_grade(): Validates that the grade is between 0 and 100.
- **Update Functions:** The update_student(), update_course(), and update_instructor() functions update corresponding IDs in the enrollments.csv file to maintain consistency when a primary key is changed.
- **Delete Functions:** The delete_student() and delete_course() functions call _delete_related_records() to delete related records in the enrollments.csv file when a student or course is deleted.

Limitations:

- The current implementation does not enforce referential integrity as strictly as a relational database. For example, it doesn't prevent inserting an enrollment record with a non-existent StudentID or CourseID *before* the CSV files are created. Validation primarily occurs during data generation or addition/update of records.
- There are no explicit constraints defined at the file level (like foreign key constraints in a database). The code attempts to maintain integrity through programmatic checks.

Disclaimer:

- This report was created in part using AI tools to help structure the content and improve language. All technical explanations and programming logic were reviewed and validated to ensure accuracy..