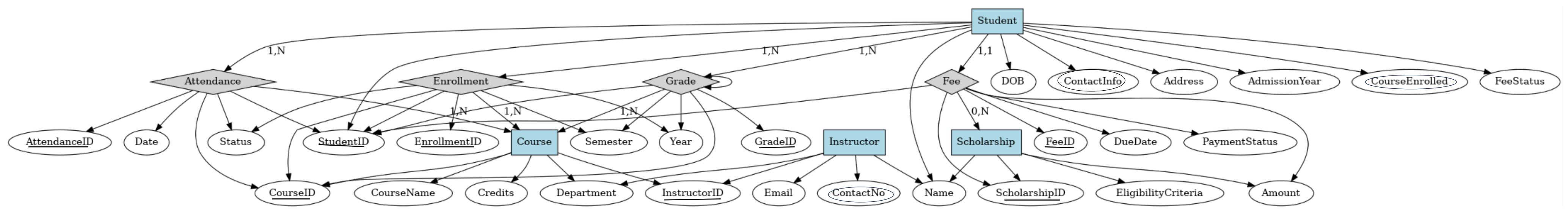
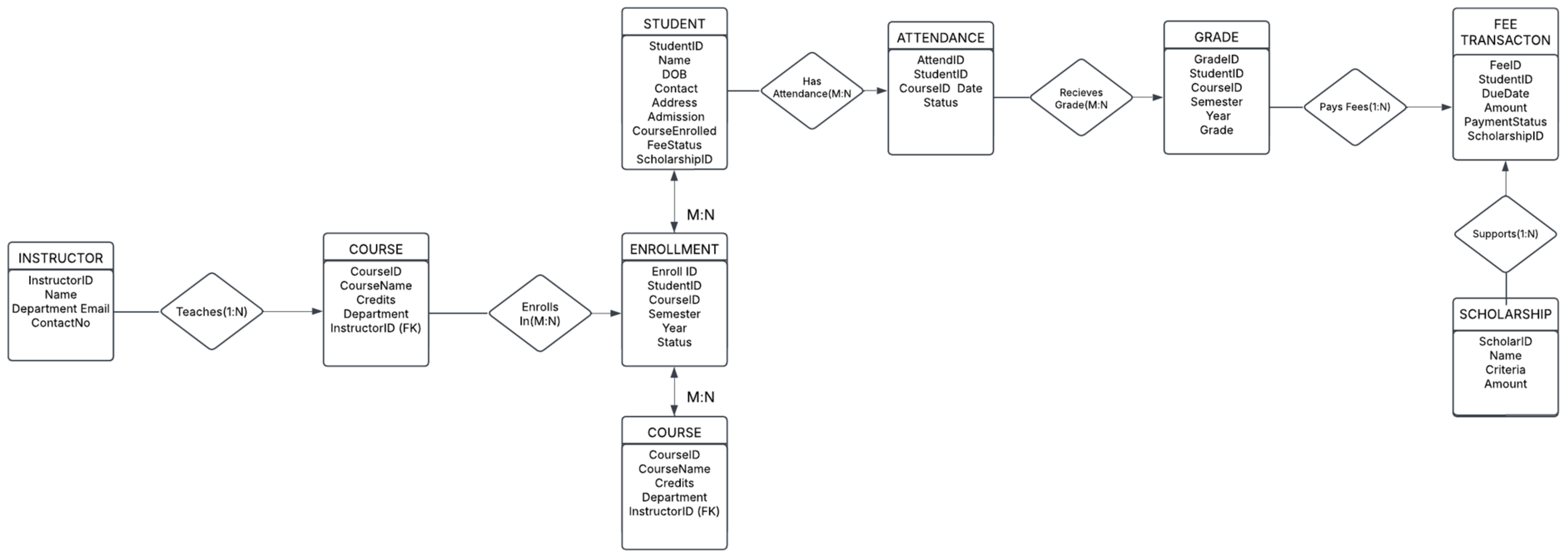
Database Design Project Part 2 - Conceptual & Logical Design Usaidkhan Pathan

Step 1 (25%) - Entity Relationship Diagram: Based on the entities and attributes derived from requirements of Database Design Project Part 1





Step 2 (25%) - Relations: Using your ER diagram developed in Step 1

Step 1: Mapping Strong Entity Types

Each strong entity type is mapped to a relation (table) with its attributes.

- **INSTRUCTOR (InstructorID as Primary Key)**

INSTRUCTOR(InstructorID, Name, Department, Email, ContactNo)

- **STUDENT (StudentID as Primary Key)**

STUDENT(StudentID, Name, DOB, Contact, Address, Admission, CourseEnrolled, FeeStatus, ScholarshipID)

- **COURSE (CourseID as Primary Key)**

COURSE(CourseID, CourseName, Credits, Department, InstructorID (FK))

- **SCHOLARSHIP (ScholarID as Primary Key)**

SCHOLARSHIP(ScholarID, Name, Criteria, Amount)

Step 2: Mapping Weak Entity Types

There are no weak entities in the given diagram.

Step 3: Mapping Binary 1:1 Relationships

There are no explicit 1:1 relationships in the diagram.

Step 4: Mapping Binary 1:N Relationships

For each **1:N relationship**, add a **foreign key** in the relation on the "N" side.

- **Teaches (1:N) → One Instructor teaches many Courses**
  - The **InstructorID** is already present as a **foreign key** in the **COURSE** table.

COURSE(CourseID, CourseName, Credits, Department, InstructorID (FK))

- **Pays Fees (1:N) → One Student can have multiple Fee Transactions**
  - The **StudentID** is already present in the **FEE_TRANSACTION** table.

FEE_TRANSACTION(FeeID, StudentID (FK), DueDate, Amount, PaymentStatus, ScholarshipID (FK))

- **Supports (1:N) → One Scholarship supports multiple Students**
  - The **ScholarshipID** is already present as a foreign key in **STUDENT** and **FEE_TRANSACTION**.

STUDENT(StudentID, Name, DOB, Contact, Address, Admission, CourseEnrolled, FeeStatus, ScholarshipID (FK))

FEE_TRANSACTION(FeeID, StudentID (FK), DueDate, Amount, PaymentStatus, ScholarshipID (FK))

Step 5: Mapping Binary M:N Relationships

For each **M:N relationship**, create a new relation with **foreign keys** from both entities.

- **Enrollment (M:N) → A Student can enroll in multiple Courses**
  - Create a separate **ENROLLMENT** table.

ENROLLMENT(EnrollID, StudentID (FK), CourseID (FK), Semester, Year, Status)

- **Has Attendance (M:N) → A Student attends multiple Courses**
    - Create a separate **ATTENDANCE** table.

ATTENDANCE(AttendID, StudentID (FK), CourseID (FK), Date, Status)

- **Receives Grade (M:N) → A Student receives a Grade for multiple Courses**
    - Create a separate **GRADE** table.

GRADE(GradeID, StudentID (FK), CourseID (FK), Semester, Year, Grade)

Step 6: Mapping Multivalued Attributes

There are no explicit **multivalued attributes** in the diagram.

Step 7: Mapping N-ary Relationships

There are no **N-ary (more than two entities) relationships** in the diagram.

FINAL RELATIONAL SCHEMA: After applying all the steps, the final relational schema is:

INSTRUCTOR

InstructorID	Name	Department	ContactNo	Email

STUDENT

StudentID	Name	DOB	Contact	Address	Admission	CourseEnrolled	FeeStatus	ScholarshipID

COURSE

CourseID	CourseName	Credits	Department	InstructorID

ENROLLMENT

EnrollID	StudentID	CourseID	Semester	Year	Status

ATTENDANCE

AttendID	StudentID	CourseID	Date	Status

GRADE

<u>GradeID</u>	<u>StudentID</u>	<u>CourseID</u>	Semester	Year	Grade

FEE_TRANSACTION

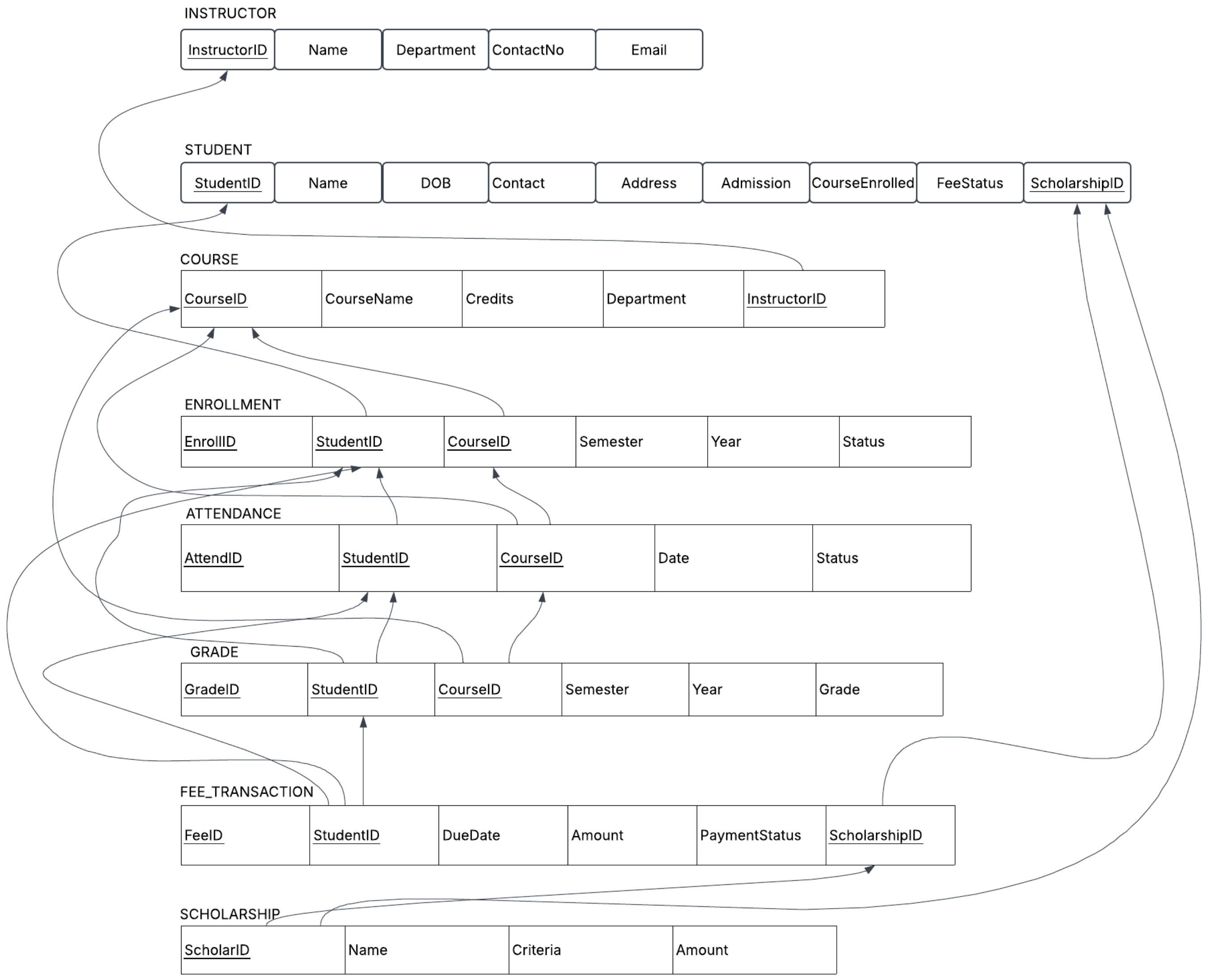
<u>FeeID</u>	<u>StudentID</u>	DueDate	Amount	PaymentStatus	<u>ScholarshipID</u>

SCHOLARSHIP

<u>ScholarID</u>	Name	Criteria	Amount

FINAL RELATIONAL SCHEMA AS PER SAMPLE

Database Design Project Part 2 - Conceptual & Logical Design Usaidkhan Pathan



Step 1: Understanding the Current Schema

The given schema consists of the following tables:

1. **INSTRUCTOR** - Stores instructor details.
2. **STUDENT** - Contains student-related information.
3. **COURSE** - Stores course-related data.
4. **ENROLLMENT** - Records students enrolled in courses.
5. **ATTENDANCE** - Tracks student attendance.
6. **GRADE** - Stores grades awarded to students.
7. **FEE_TRANSACTION** - Tracks fee payments.
8. **SCHOLARSHIP** - Contains scholarship details.

Step 2: Applying Normalization

1st Normal Form (1NF) - Remove Repeating Groups & Ensure Atomicity

Violations:

- The STUDENT table has multi-valued attributes such as CourseEnrolled (students can enroll in multiple courses).
- The FeeStatus might be dependent on different payments over time.
- COURSE has InstructorID, meaning one course might have multiple instructors, which is not represented properly.

Modifications:

- Separate **Course Enrollment** into a junction table (ENROLLMENT) that links students and courses.
- Remove FeeStatus from STUDENT, as fee transactions should be managed independently in FEE_TRANSACTION.
- Ensure each attribute holds only atomic (indivisible) values.

2nd Normal Form (2NF) - Remove Partial Dependencies

A table is in 2NF if:

1. It is in 1NF.
2. There are no **partial dependencies** (i.e., no non-key attribute should be dependent on part of a composite primary key).

Violations & Fixes:

- In ENROLLMENT, attributes like Semester, Year, and Status might be dependent only on StudentID or CourseID but not both together. These attributes should be moved to separate tables.
  - Create a STUDENT_COURSE table to store StudentID, CourseID, Semester, and Year.
  - Keep ENROLLMENT to track the status of student enrollment separately.
- GRADE should be separated into a table linking StudentID, CourseID, and Semester/Year, with Grade stored in a different table.

- FEE_TRANSACTION should store the ScholarshipID in a separate table to avoid redundancy.

3rd Normal Form (3NF) - Remove Transitive Dependencies

A table is in 3NF if:

1. It is in 2NF.
2. There are no **transitive dependencies** (i.e., no non-key column should depend on another non-key column).

Violations & Fixes:

- FEE_TRANSACTION contains ScholarshipID, but scholarships should be linked to students separately.
  - Create a STUDENT_SCHOLARSHIP table with StudentID and ScholarshipID to track awarded scholarships.
- COURSE includes InstructorID, which could be an issue if multiple instructors teach one course.
  - Create a COURSE_INSTRUCTOR table to handle many-to-many relationships.
- STUDENT had FeeStatus, which depended on payments made. This is now removed.

Final Normalized Schema

1. INSTRUCTOR

InstructorID	Name	Department	ContactNo	Email

2. STUDENT

StudentID	Name	DOB	Contact	Address	Admission

3. COURSE

CourseID	CourseName	Credits	Department

4. COURSE_INSTRUCTOR (New)

CourseID	InstructorID

5. ENROLLMENT

EnrollID	StudentID	CourseID	Status

6. STUDENT_COURSE (New)

StudentID	CourseID	Semester	Year

7. ATTENDANCE

AttendID	**StudentID**	**CourseID**	**Date**	**Status**

8. GRADE

GradeID	**StudentID**	**CourseID**	**Semester**	**Year**	**Grade**

9. FEE_TRANSACTION

FeeID	**StudentID**	**DueDate**	**Amount**	**PaymentStatus**

10. STUDENT_SCHOLARSHIP (New)

StudentID	**ScholarshipID**

11. SCHOLARSHIP

ScholarID	**Name**	**Criteria**	**Amount**

Key Takeaways

- **1NF**: Removed multi-valued attributes by creating separate tables (ENROLLMENT, COURSE_INSTRUCTOR).
- **2NF**: Eliminated partial dependencies by moving semester-related attributes into a new table (STUDENT_COURSE).
- **3NF**: Eliminated transitive dependencies by creating a separate table for student scholarships (STUDENT_SCHOLARSHIP).

This ensures **data integrity, avoids redundancy, and enhances maintainability** in your database design.

Step 4 (25%) - Final Reflective Report:

Introduction

Relational database design is a process with multiple steps involving strict analysis and logical structure to yield efficiency, accuracy, and data integrity. This report describes the entire process of database designing, ranging from deriving the Entity-Relationship Diagram (ERD) to mapping it into a relational schema and normalizatio n. We also describe the issues that were encountered at each step and how they were overcome.

Deriving the Entity-Relationship Diagram (ERD)

Step one was to identify entities, attributes, and relationships from the given requirements. The primary entities found were Instructor, Student, Course, Fee Transaction, Scholarship, Enrollment, Attendance, and Grade. The type of relationship between these entities, such as 1:N (one-to-many) and M:N (many-to-many), was identified in order to create a proper ERD.

Difficulties Encountered

Understanding Requirements: To start with, some properties were unclear about where they belonged in what entity. For example, FeeStatus in the Student entity was unclear since fee payments need to be recorded as transactions and not a static property.

Handling Relationships: Establishing relationships between entities such as Instructor-Course and Student-Course was complex since there were varying cases to be considered (e.g., numerous instructors in one course).

Weak Entities and Multivalued Attributes: We ensured that all attributes were atomic in nature and not multi-valued data.

Mapping ERD to Relations

Once the ERD was finalized, the process of mapping it into relational tables followed. Strong entity types were mapped to individual tables, while relationships were shown using foreign keys and additional linking tables for M:N relationships.

Challenges Faced:

Identifying Primary and Foreign Keys: While establishing tables, making sure there was a correct Primary Key (PK) and Foreign Key (FK) for every relation to help in referential integrity was important.

M:N Relationships: Numerous many-to-many relations like Student-Course (Enrollment) needed individual tables to manage enrollment, attendance, and grading properly.

Normalization Impact: Certain table structures in the early stages had redundancy, which had to be mapped with caution to ensure effectiveness.

Normalization Process

The relational schema was normalized to eliminate redundancies, ensure atomicity, and enhance maintainability.

1st Normal Form (1NF):

Removed multi-valued attributes such as CourseEnrolled in the Student table by introducing an Enrollment table.

Ensured that each attribute held a single atomic value.

2nd Normal Form (2NF):

Eliminated partial dependencies by introducing STUDENT_COURSE to separate semester-related attributes.

Addressed composite primary key dependencies in Enrollment and Grade tables.

3rd Normal Form (3NF):

Removed transitive dependencies by adding STUDENT_SCHOLARSHIP to keep separately granted scholarships.

Resolved dependencies in the Fee Transaction table by properly modeling the students and scholarships relationship.

Challenges Faced:

Redundancy Management: Initially, columns such as FeeStatus were redundantly stored in multiple tables and required restructuring.

Dependency Conflicts: Ensuring that all dependencies were correctly assigned in order not to create anomalies.

Balancing Normalization and Performance: While higher normalization ensures integrity, over-decomposition can lead to performance inefficiencies. A balancing act was needed.

Conclusion

The database design process required iterative refinement to reach an optimized and well-structured schema. The ERD served as a foundation for entity and relationship definition, while normalization provided for efficiency and integrity. Such problems of relationship mapping, redundancy elimination, and dependency conflicts were solved effectively through logical restructuring. The resultant schema ensures data consistency, prevents redundancy, and is more maintainable and hence mature enough for real-world deployment.