LAB 1

Task 1

Task 1.1

Relation A: Employee

1) Superkeys:

1) **{EmpID}**: This is a superkey because EmpID is already unique for each employee.
2) **{SSN}:** Social Security Numbers are unique
3) **{Email}:** Each employee has unique email address
4) **{EmpID, Name}:** Since {EmpID} is already a superkey, adding the Name attribute still makes it a superkey
5) **{SSN, Department}:** Since {SSN} is a superkey, adding Department still makes it a superkey
6) **{EmpID, SSN, Email, Phone**}: This is a combination of unique attributes that makes it a super key

2) Candidate keys:

1. **{EmpID}**: unique for every employee
2. **{SSN}:** unique for every people
3. **{Email}:** unique for every employee

3) Which candidate key would you choose as primary key and why?

I choose EmpID as a primary key. This is an internal system identifier that is unlikely to change during an employee's career. External identifiers, such as the SSN, are confidential and may change in rare cases (for example, due to identity theft issues).

4) Can two employees have the same phone number?

Yes, employees can have the same phone numbers. For example, employees might have the same work phone number due to a company shortage. However, in the sample, each employee has a unique phone number.

Relation B: Course Registration:

1)Determine the minimum attributes needed for the primary key

The minimum set of attributes required for a primary key is: {StudentID, CourseCode, Section, Semester, Year}. This combination uniquely identifies a specific course registration record for a student.

2) Explain why each attribute in your primary key is necessary

{StudentID}: This attribute identifies a specific student. Without it, it is impossible to distinguish between different students enrolled in the same course section in the same semester.

{CourseCode}: This attribute identifies a specific course. A student can take the same course multiple times, but this attribute is necessary to distinguish between different courses enrolled in the same semester.

{Section}: This attribute identifies a specific course section. The business rule states that a student cannot enroll in the same course section in the same semester.

{Semester and Year}: These two attributes are part of the key and are necessary to distinguish between students taking the same course section in different semesters. The business rule explicitly states, "A student can take the same course in different semesters." Without the Semester and Year attributes, a student's repeated enrollment in the same course would be considered a duplicate.

Task 1.2

1)Student (AdvisorID) is related to Professor (ProfID). The AdvisorID attribute in the Student table is a foreign key that references the ProfID primary key in the Professor table. This relationship indicates which professor is the academic advisor for each student.

2) Department (ChairID) is related to Professor (ProfID). The ChairID attribute in the Department table is a foreign key that references the ProfID primary key in the Professor table. This indicates which professor is the department chair.

3) Course (DepartmentCode) is related to Department (DeptCode). The DepartmentCode attribute in the Course table is a foreign key that references the DeptCode primary key in the Department table. This relationship indicates which department each course belongs to.

4) Enrollment (StudentID) is related to Student (StudentID). The StudentID attribute in the Enrollment table is a foreign key that references the StudentID primary key in the Student table. This relationship connects a course enrollment record to a specific student.

5) Enrollment (CourseID) is related to Course (CourseID). The CourseID attribute in the Enrollment table is a foreign key that references the CourseID primary key in the Course table. This relationship connects a course enrollment record to a specific course.

Task 2

Task 2.1

1. Identify all entities

Strong Entities:

- **Patient**: Has unique patient ID.
- **Doctor**: Has unique doctor ID.
- **Department**: Has a unique departament code

Weak Entities:

- **Hospital Room**: Its number is only unique within a specific department, meaning its existence depends on the Department entity.
- **Appointment**: It depends on Patient and Doctor to identify a specific instance of a visit.

- **Prescription**: It depends on a relationship between a Doctor and a Patient to exist.

2)Identify all attributes for each entity

Patient

- **patientID**: Simple, Primary Key
- **name**: Simple
- **birthdate**: Simple
- **address**: Composite (composed of street, city, state, zip)
- **phone**: Multi-valued
- **insurance**: Simple

Doctor

- **doctorID**: Simple, Primary Key
- **name**: Simple
- **specializations**: Multi-valued
- **phone**: Simple
- **office location**: Simple

Department

- **department code**: Simple, Primary Key
- **name**: Simple
- **location**: Simple

Appointment

- **date/time**: Simple (Part of the weak entity's key)
- **purpose**: Simple
- **notes**: Simple

Prescription

- **medication**: Simple (Part of the weak entity's key)
- **dosage**: Simple
- **instructions**: Simple

Hospital Room:

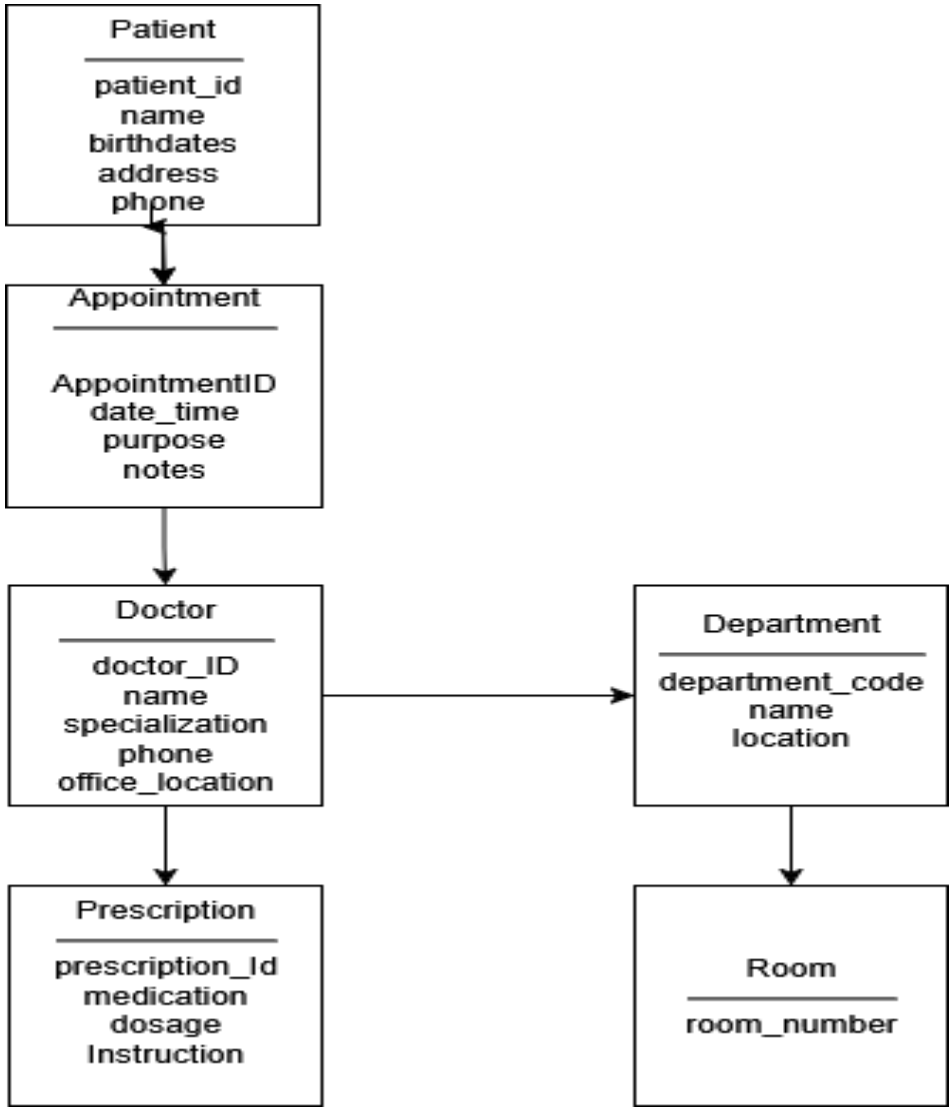
- **room_number**: Simple (Part of the weak entity's key)

3) Identify all relationships with their cardinalities (1:1, 1:N, M:N)

- **Patient has Appointment** with **Doctor**: This is a many-to-many (M:N) relationship. A patient can have multiple appointments with one or more doctors, and a doctor can have multiple appointments with one or more patients.
- **Doctor prescribes Prescription** to **Patient**: This is a many-to-many (M:N) relationship. A doctor can prescribe multiple medications to many patients, and a patient can receive prescriptions from multiple doctors
- **Department has Doctor**: This is a one-to-many (1:N) relationship. A department can have multiple doctors, but each doctor is typically associated with only one department.

- **Department has Hospital Room**: This is a one-to-many (1:N) relationship. A department can have many rooms, but each room belongs to only one department.

4)Draw the complete ER diagram using proper notation

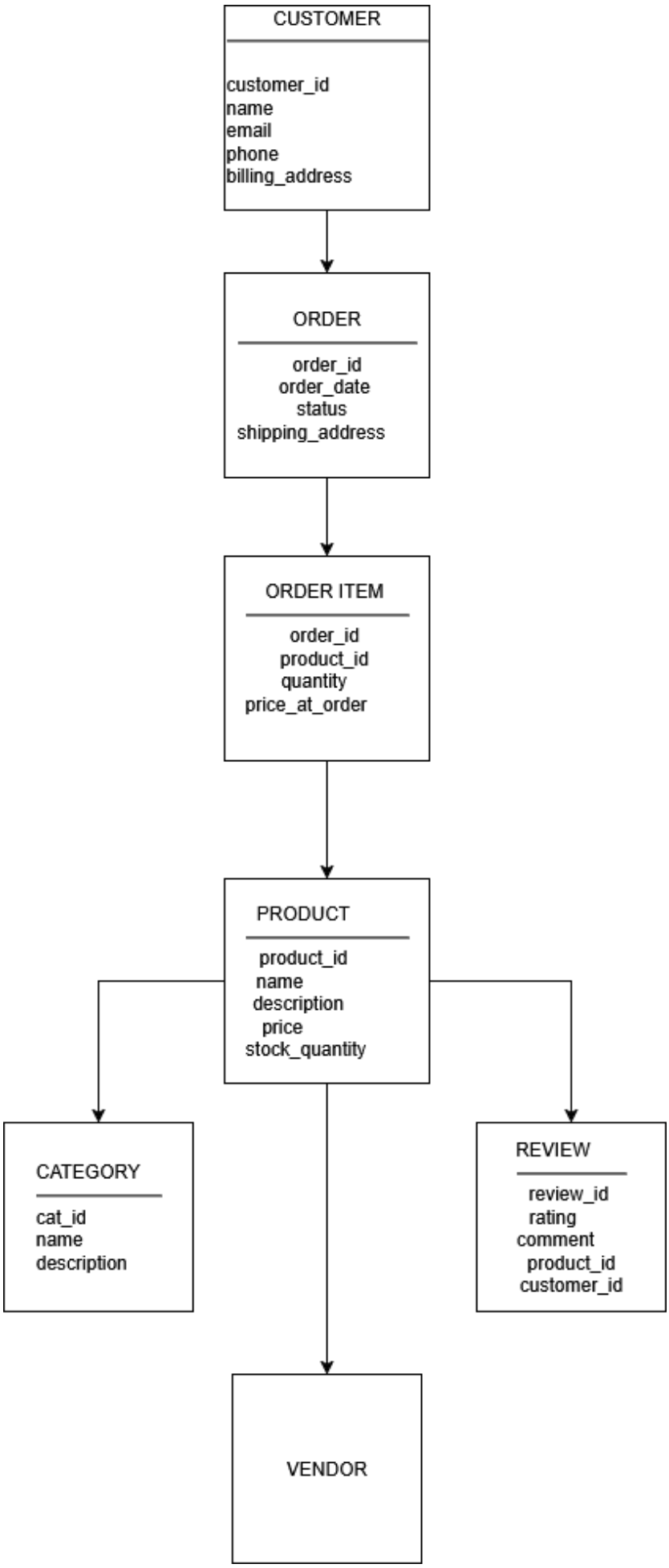


5) Mark primary keys

- **Patient**: patientID
- **Doctor**: doctorID

- **Department**: department code
- **Hospital Room**: {department code, room number} (composite key combining the owner's primary key and its own partial key).
- **Appointment**: {patientID, doctorID, date/time} (composite key combining the owner entities' primary keys and its own partial key).
- **Prescription**: {doctorID, patientID, medication} (composite key combining the owner entities' primary keys and its own partial key).

Task 2.2 1)


CUSTOMER
customer_id
name
email
phone
billing_address
ORDER
order_id
order_date
status
shipping_address
ORDER ITEM
order_id
product_id
quantity
price_at_order
PRODUCT
product_id
name
description
price
stock_quantity
CATEGORY
cat_id
name
description
REVIEW
review_id
rating
comment
product_id
customer_id
VENDOR

2) The primary weak entity in this scenario is Order items. Order item is part of order and cannot be exist without order.

3) A Many-to-Many relationship that requires attributes: Order ↔ Product This relationship is implemented through the Order Item entity because it has its own attributes: quantity price_at_order

Task 4

Task 4.1

1. Functional Dependencies (FDs)Based on the StudentProject table schema, here are the functional dependencies:

StudentID → StudentName, StudentMajor: A specific student ID determines the student's name and major.

ProjectID → ProjectTitle, ProjectType: A specific project ID determines the project's title and type.

SupervisorID → SupervisorName, SupervisorDept: A specific supervisor ID determines the supervisor's name and department.

StudentID, ProjectID → Role, HoursWorked, StartDate, EndDate: A specific combination of a student and a project determines their role, hours worked, and the project's start and end dates.

SupervisorID, ProjectID → Role: A specific supervisor on a project has a specific role.

2. Problems with the Table

Redundancy: There is significant data redundancy.

The Student's name and major are repeated for every project they are on.

The Project's title and type are repeated for every student working on it.

The Supervisor's name and department are repeated for every project they supervise.

Anomalies:

Update Anomaly: If a student changes their major, you would have to update their StudentMajor field in every row where that StudentID appears. If you miss one, you'll have inconsistent data.

Insert Anomaly: You cannot add a new student or a new supervisor until they are assigned to a project. If you have a new student without an assigned project yet, you can't insert their details into the table.

Delete Anomaly: If the last student working on a specific project is removed from the table, you lose all information about that project (ProjectTitle, ProjectType) and the supervisor's details.

3. First Normal Form (1NF)\

There are no apparent 1NF violations. All attributes are atomic (single-valued), and there are no repeating groups. Each row is unique, assuming the primary key is correctly identified.

4. Second Normal Form (2NF)

Primary Key: The primary key is the minimal set of attributes that uniquely identifies a row. In this case, the primary key is {StudentID, ProjectID}. This composite key is required to uniquely identify a student's participation in a project.

Partial Dependencies: Partial dependencies exist when a non-key attribute depends on only a part of the composite primary key.

StudentID → StudentName, StudentMajor (depends on a part of the key)

ProjectID → ProjectTitle, ProjectType (depends on a part of the key)

SupervisorID → SupervisorName, SupervisorDept (depends on SupervisorID, which is a determinant, but not a candidate key)

2NF Decomposition: To achieve 2NF, we separate the tables to eliminate these partial dependencies.

Student(StudentID, StudentName, StudentMajor)

Project(ProjectID, ProjectTitle, ProjectType, SupervisorID)

StudentProjectRole(StudentID, ProjectID, Role, HoursWorked, StartDate, EndDate)

5. Third Normal Form (3NF)

Transitive Dependencies: A transitive dependency exists when a non-key attribute depends on another non-key attribute. In the 2NF decomposition above, a transitive dependency still exists in the Project table.

SupervisorID → SupervisorName, SupervisorDept: SupervisorName and SupervisorDept are dependent on SupervisorID, and SupervisorID is not the primary key of the Project table.

Final 3NF Decomposition: To achieve 3NF, we must remove this transitive dependency by creating a new table for supervisors.

Student(StudentID, StudentName, StudentMajor)

Project(ProjectID, ProjectTitle, ProjectType, SupervisorID)

StudentProjectRole(StudentID, ProjectID, Role, HoursWorked, StartDate, EndDate)

Supervisor(SupervisorID, SupervisorName, SupervisorDept)

Task 4.2

1) **Primary Key**
The primary key of the `CourseSchedule` table is a composite key consisting of **{StudentID, CourseID, TimeSlot, InstructorID}**.
This is tricky because the business rules state that each course section is taught by one instructor at one time in one room. This implies that **{CourseID, TimeSlot, InstructorID}** could define a unique course section. However, a student can be in multiple course sections, and the table needs to track which student is in which section. Therefore, **{StudentID}** must be included.
The **Room** attribute is not part of the primary key because TimeSlot and Room are related, as each TimeSlot in a Room is unique.

2)Functional Dependencies (FDs)

- StudentID → StudentMajor
- CourseID → CourseName
- InstructorID → InstructorName
- Room, TimeSlot → Building
- CourseID, TimeSlot, InstructorID → Room

3) **Check for BCNF**
The table is not in BCNF (Boyce-Codd Normal Form).
A table is in BCNF if for every non-trivial functional dependency **X → Y**, X is a superkey. Here, we have functional dependencies where the determinant is not a superkey.

- StudentID → StudentMajor
- CourseID → CourseName
- InstructorID → InstructorName
- Room, TimeSlot → Building
- CourseID, TimeSlot, InstructorID → Room

The primary key is {StudentID, CourseID, TimeSlot, InstructorID}. None of the determinants on the left side of the dependencies listed above are superkeys of the main table. Therefore, the table is not in BCNF.

4. 4) **BCNF Decomposition**
To decompose the table into BCNF, we need to separate the FDs into their own tables.

- FD: StudentID → StudentMajor
→ Student(StudentID, StudentMajor)
- FD: CourseID → CourseName
→ Course(CourseID, CourseName)
- FD: InstructorID → InstructorName
→ Instructor(InstructorID, InstructorName)
- FD: Room, TimeSlot → Building
→ RoomBuilding(Room, TimeSlot, Building)
- FD: CourseID, TimeSlot, InstructorID → Room
→ CourseSection(CourseID, TimeSlot, InstructorID, Room)
- Remaining Attributes (Main Table):
→ Enrollment(StudentID, CourseID, TimeSlot, InstructorID)

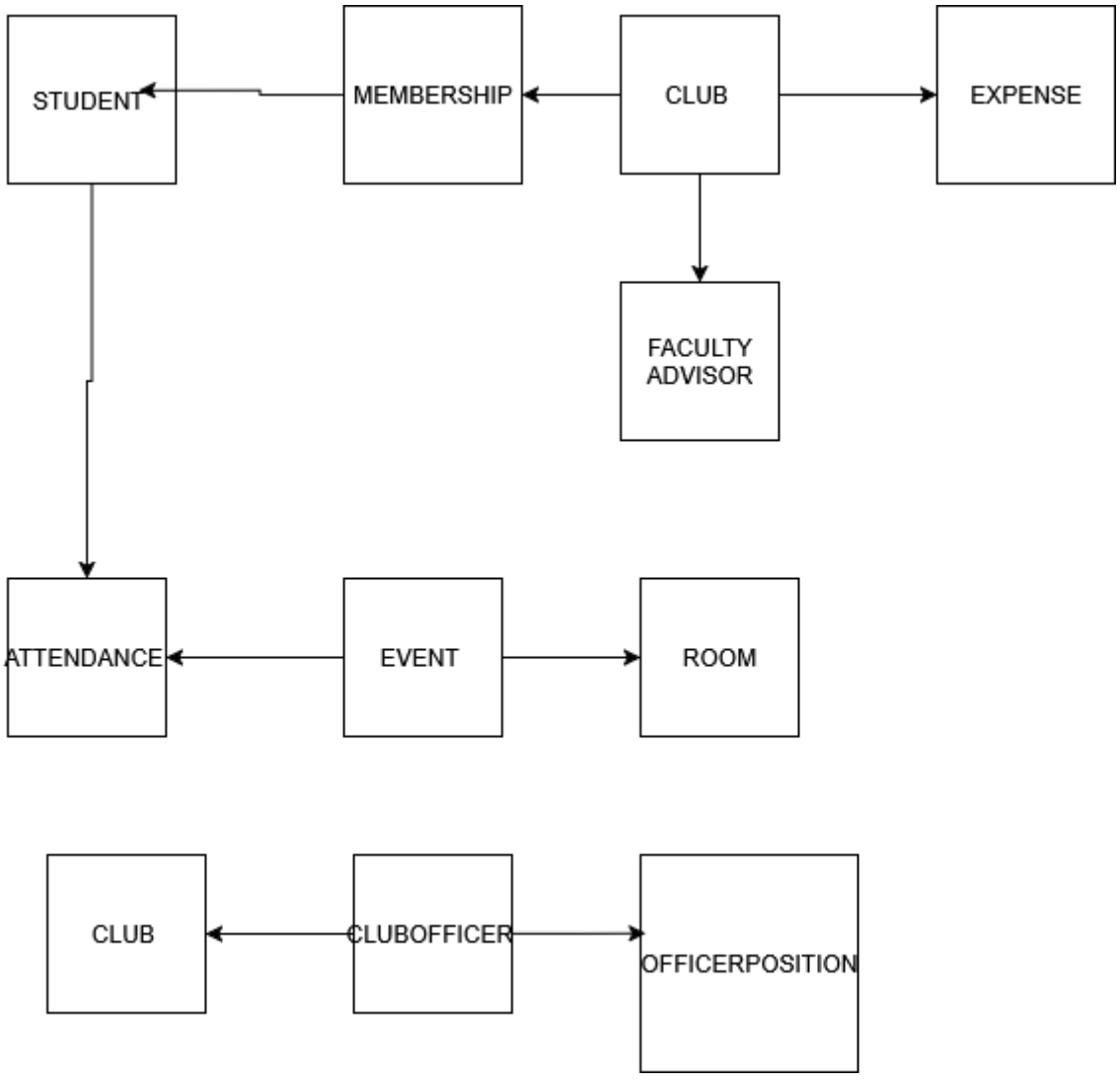
Final BCNF decomposition:

- Student(StudentID, StudentMajor)
- Course(CourseID, CourseName)
- Instructor(InstructorID, InstructorName)
- RoomBuilding(Room, TimeSlot, Building)
- CourseSection(CourseID, TimeSlot, InstructorID, Room)
- Enrollment(StudentID, CourseID, TimeSlot, InstructorID)

5) **Potential Loss of Information**
The decomposition does not result in a loss of information. This is because the decomposition is based on functional dependencies, which are lossless. By using the primary keys of the new tables as foreign keys in the connecting tables (**CourseSection** and **Enrollment**), the original data can be fully reconstructed.

Task 5

1)



2)

Student(StudentID PK, Name, Major, Year)
FacultyAdvisor(FacultyID PK, Name, Department)
Club(ClubID PK, ClubName, Description, Budget, FacultyID FK)
Membership(StudentID PK, ClubID PK, JoinDate, Role)
OfficerPosition(PositionID PK, Title)
ClubOfficer(ClubID PK, StudentID PK, PositionID PK, StartDate, EndDate)
Room(RoomID PK, Building, RoomNumber, Capacity)
Event(EventID PK, ClubID FK, RoomID FK, Title, Date, Time)
Attendance(StudentID PK, EventID PK, Status)
Expense(ExpenseID PK, ClubID FK, Amount, Description, Date)

3) Officer positions (President, Treasurer, Secretary, …):

- **Option 1:** Store the role directly as a text attribute in `ClubOfficer` (e.g., Role = "President").
- **Option 2 (chosen):** Create a separate table `OfficerPosition(PositionID, Title)` and use foreign keys in `ClubOfficer`.

Why Option 2?

- Easier to manage (new roles can be added without changing the schema).
- Avoids data redundancy (no repeated text values).
- Ensures data consistency (controlled list of valid officer positions).

4)

- **List all clubs that a given student belongs to, including the student's role in each club.**

- **Find all upcoming events for a given club, including event title, date, time, and room.**

- **Show the total budget and total expenses for each club, and calculate the remaining balance.**