**Database Systems Laboratory Work**

* **Part 1: Key Identifications Exercises**

**Task 1.1**

**Relation A: Employee**

SampleData**:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| EmpID | SSN | Email | Phone | Name | Department | Salary |
| 101 | 123-45-6789 | john@company.com | 555-0101 | John | IT | 75000 |
| 102 | 987-65-4321 | mary@company.com | 555-0102 | Mary | MR | 68000 |
| 103 | 456-78-9123 | bob@company.com | 555-0103 | Bob | IT | 72000 |

**1)**A superkey is a set of one or more attributes (columns) that can uniquely identify each row (tuple) in a table. That’s, if we have a superkey, then no two rows in the table can have the same values of all attributes from this set.Example(Relation A**):{EmpID}, {SSN}, {Email}, {Phone}, {EmpID ,SSN}, {SSN,Email,Phone}, {EmplID,Email,Name}, … and etc.(you can list many of them).**

**2**) A candidate key is a minimal superkey, i.e., a superkey that does not contain a smaller one. Therefore, a relation can have multiple candidate keys, each with a different number of attributes**.In Relaiton A (Sample Data)** the candidate keys are **:{EmpID}, {SSN}, {Email} and {Phone}.**Because all of them contain unique for each row.

**3)** **I would choose EmpID as the primary key.**

* **Justification**: EmpID is likely a surrogate key—a simple, system-generated integer with no business meaning. It is short, efficient for indexing and joins, and guaranteed not to change. Natural keys like SSN, Email, and Phone are sensitive personal data (privacy concern) and could potentially change during an employee's tenure, causing update anomalies in foreign key references**.**

**4)** **Based on the provided sample data, no, two employees cannot have the same phone number.**

* **Justification:**The Phone attribute is a candidate key. The sample data shows all values in the Phone column are unique (555-0101, 555-0102, 555-0103). For an attribute to be a candidate key, it must contain unique values for all tuples in the relation, not just the sample. Therefore, the business rule enforced by this schema is that a phone number uniquely identifies an employee**.**

**Relation B:Course registration**

**Registration(StudentID, CourseCode, Section, Semester, Year, Grade, Credits)**

**1)**The minimum set of attributes needed for the primary key is:  
**(StudentID, CourseCode, Section, Semester, Year).**

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

* **StudentID**uniquely identifies the student.
* **CourseCode**identifies the course being taken.
* **Section**distinguishes different instances of the same course offered within the same semester.
* **Semester and Year**distinguish registrations for the same course and section offered across different academic terms.

Together, these attributes form the minimal set required to guarantee tuple uniqueness based on the provided business rules. Removingany attribute would result in violations. For example, using only **(StudentID, CourseCode)** would incorrectly prevent a student from re-taking the same course in a different semester.

**3)** Given these business rules, there are no other subsets of attributes that could uniquely identify a registration record. Therefore,**(StudentID, CourseCode, Section, Semester, Year) is the only candidate key.**

**Task 1.2**

A foreign key is an attribute in one table that references the primary key of another table to create a logical link between the data.

Based on the schemas provided, the following foreign key relationships exist:

1. In the Student table:
   * AdvisorID is a foreign key that references Professor(ProfID).
   * *Rationale:* This establishes which Professor serves as the academic advisor for a Student.
2. In the Course table:
   * DepartmentCode is a foreign key that references Department(DeptCode).
   * *Rationale:* This assigns each Course to the Department that owns it (e.g., the Calculus course belongs to the Math Department).
3. In the Department table:
   * ChairID is a foreign key that references Professor(ProfID**).**
   * *Rationale:* This indicates which Professor holds the position of chair for a given Department.
4. In the Enrollment table:
   * StudentID is a foreign key that references Student(StudentID).
   * *Rationale:* This links an enrollment record to the specific Student who is enrolled.
   * CourseID is a foreign key that references Course(CourseID).
   * *Rationale:* This links an enrollment record to the specific Course in which the student is enrolled.

* **Part 2: ER Diagram Construction**

**Task 2.1**

**1) Strong Entities:**These exist independently.

* + **Patient**
  + **Doctor**
  + **Department**

**Weak Entities:**Their existence depends on a strong entity.

* + **Appointment (weak on Patient and Doctor)**
  + **Prescription (weak on Patient and Doctor)**
  + **HospitalRoom (weak on Department - a room number is only unique within a department)**

**2) Patient:**

* + **PatientID (Simple)**
  + **Name (Composite)**
  + **BirthDate (Simple)**
  + **Adresses (Composite)**
  + **PhoneNumber (Multi-valued)**
  + **InsuranceID (Simple)**

**Doctor:**

* + **DoctorID (Simple)**
  + **Name(composite)**
  + **Specialization (Multi-valued)**
  + **OfficeLocation (Simple)**
  + **PhoneNumber (Multi-valued)**

**Department**

* + **DeptCode (Simple)**
  + **DeptName (Simple)**
  + **Location (Simple)**

**Appointment**

* + **AppointmentID (Simple)**
  + **DateTime (Simple)**
  + **Purpose (Simple)**
  + **Notes (Simple)**

**Prescription**

* + **PrescriptionID (Simple)**
  + **MedicationName (Simple)**
  + **Dosage (Simple)**
  + **Instructions (Simple)**

**HospitalRoom**

* + **RoomNumber (Simple)**
  + **RoomType (Simple) (e.g., General, ICU, Operating)**

**3) DOCTOR belongsto DEPARTMENT (N:1)**

A Doctor works in one Department. A Department has many Doctors.

**DEPARTMENT contains HOSPITAL\_ROOM (1:N)**

A Department has many Rooms. A Room belongs to one Department.

**PATIENT has APPOINTMENT (1:N)**

A Patient can have many Appointments. An Appointment is for one specific Patient.

**DOCTOR has APPOINTMENT (1:N)**

A Doctor can have many Appointments. An Appointment is with one specific Doctor.

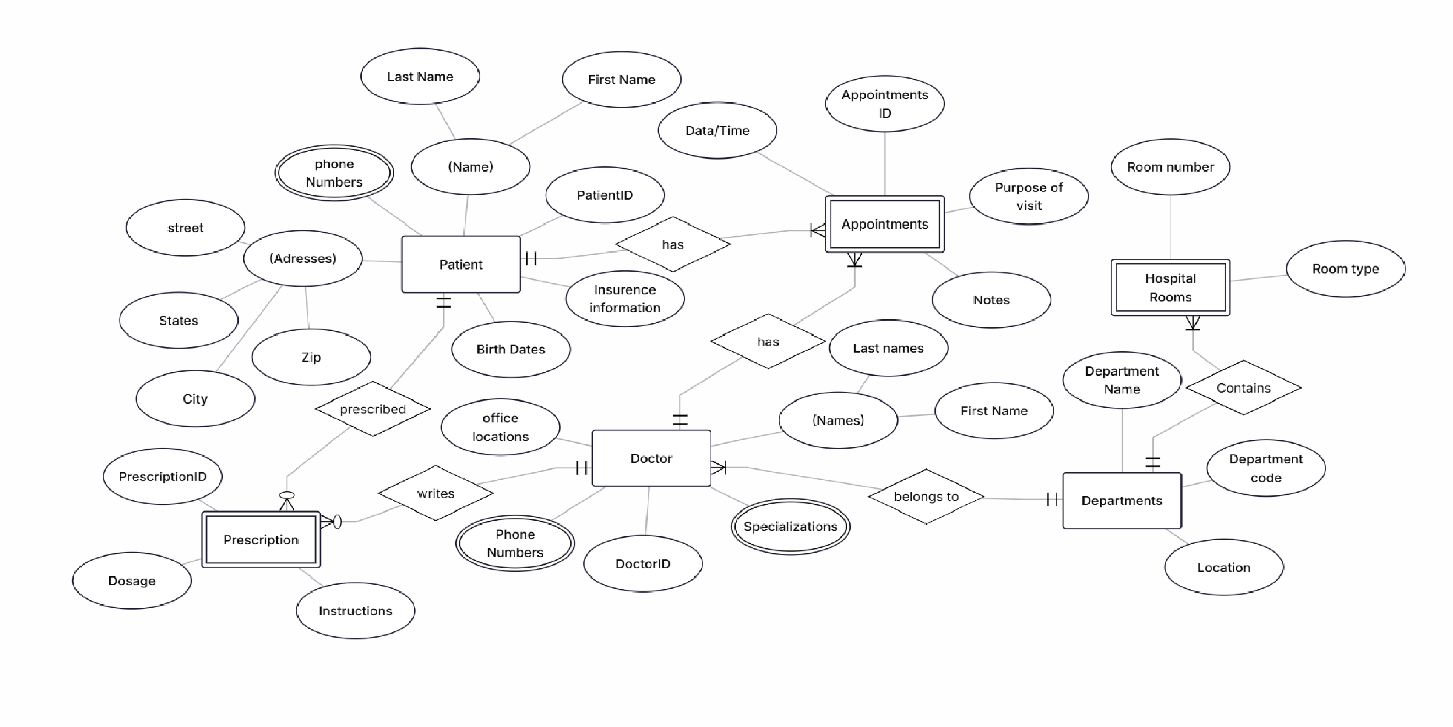
**DOCTOR *writes* PRESCRIPTION (1:N)**

A Doctor can write many Prescriptions. A Prescription is written by one Doctor.

**PATIENT is prescribed PRESCRIPTION (1:N)**

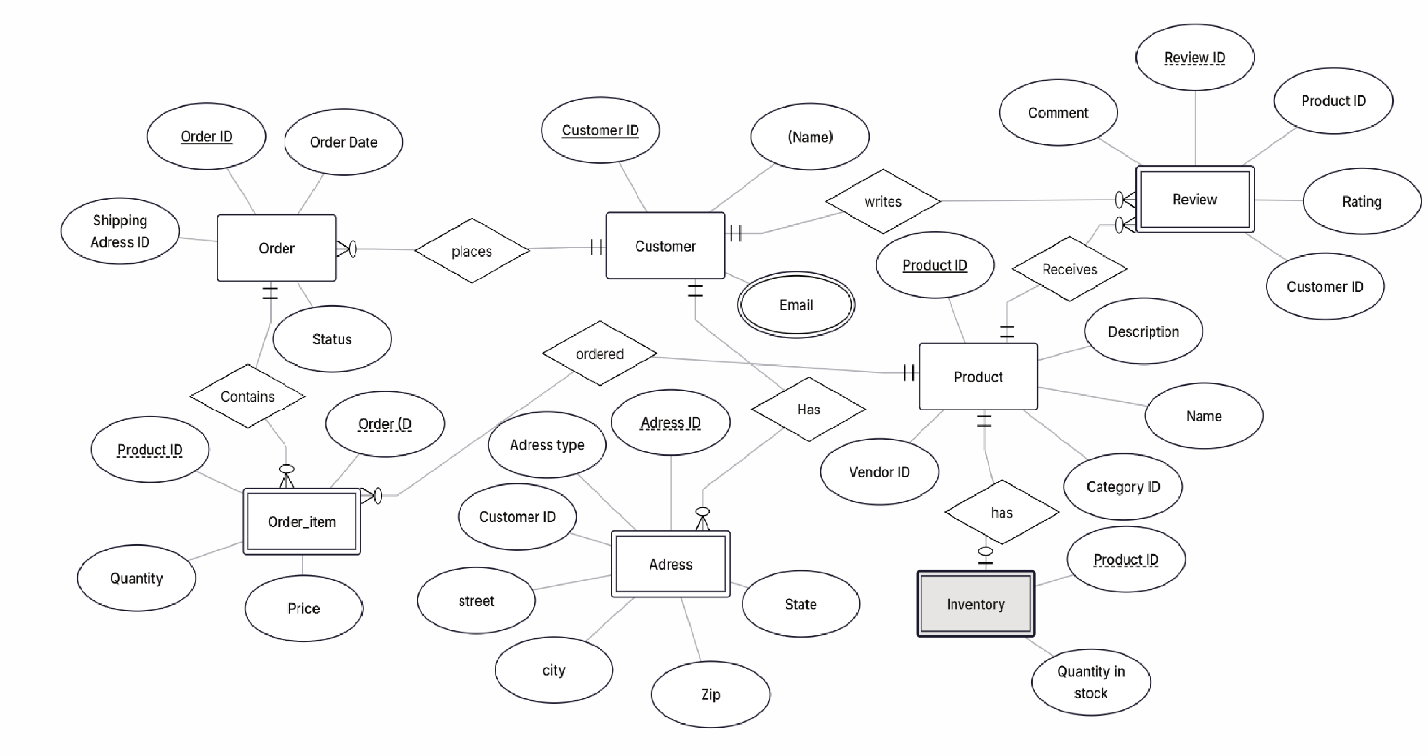
A Patient can have many Prescriptions. A Prescription is prescribed to one Patient.

**4)**

****

**Task 2.2**

**1)ER Diagram(E-commerce platform)**

****

**2) Weak Entity: ORDER\_ITEM**

**Justification:**An ORDER\_ITEM cannot exist without an ORDER. Its primary key is a composite key formed by the primary key of its owner (OrderID) and ProductID**.**

**3) Identification of a Many-to-Many Relationship with Attributes**

* **Many-to-Many Relationship:**The relationship between CUSTOMER and PRODUCT for reviews and ratings**.**
* **Attributes on the Relationship:**This M:N relationship has the attributes Rating, Comment, and ReviewDate.
* **Implementation:**This relationship is resolved by the composite/associative entity REVIEW, which holds the attributes of the relationship.

**Part 4: Normalization Workshop**

**Task 4.1: Denormalized Table Analysis**

**Given Table:`(StudentProject(StudentID, StudentName, StudentMajor, ProjectID, ProjectTitle, ProjectType, SupervisorID, SupervisorName, SupervisorDept, Role, HoursWorked, StartDate, EndDate)`**

**1)dentify Functional Dependencies (FDs):**

Based on the semantics of the data, we can assume the following functional dependencies:

StudentID → StudentName, StudentMajor`

ProjectID → ProjectTitle, ProjectType, SupervisorID`

SupervisorID → SupervisorName, SupervisorDept`

{StudentID, ProjectID} → Role, HoursWorked, StartDate, EndDate` (This determines the primary key)

**2)Identify Problems:**

Redundancy:

`StudentName` and `StudentMajor` are repeated for every project a student works on.

`ProjectTitle`, `ProjectType`, `SupervisorID` are repeated for every student working on a project.

`SupervisorName` and `SupervisorDept` are repeated for every project they supervise (and further for every student on those projects).

Update Anomaly:Changing a student's major (`StudentMajor`) requires updating multiple rows. If not done correctly, the same student could have different majors in the database.

Insertion Anomaly: Cannot add a new `Supervisor` (`SupervisorID`, `Name`, `Dept`) to the database until they are assigned to a project. Similarly, cannot add a new `Student` until they are assigned to a project.

Deletion Anomaly: If the last student leaves a project, all information about that project (`ProjectTitle`, `ProjectType`, `SupervisorID`) is deleted from the database.

**3) Apply 1NF:**

1NF Violations? The table has no repeating groups of attributes. All attribute values are atomic (assuming phone numbers or other multi-value attributes are not stored in a single field here). Therefore, it is already in 1NF.

**4)Apply 2NF:**

Primary Key:`{StudentID, ProjectID}`

Partial Dependencies:

`StudentID → StudentName, StudentMajor` (depends on only part of the PK)

`ProjectID → ProjectTitle, ProjectType, SupervisorID` (depends on only part of the PK)

The attributes `Role, HoursWorked, StartDate, EndDate` are fully functional on the entire PK.

2NF Decomposition: Remove partial dependencies into their own tables.

`Students(StudentID, StudentName, StudentMajor)` | PK: `StudentID`

`Projects(ProjectID, ProjectTitle, ProjectType, SupervisorID)` | PK: `ProjectID`

`Assignments(StudentID, ProjectID, Role, HoursWorked, StartDate, EndDate)` | PK: `{StudentID, ProjectID}`

**5) Apply 3NF:**

Transitive Dependencies: In the `Projects` table, there is a transitive dependency: `ProjectID → SupervisorID` and `SupervisorID → SupervisorName, SupervisorDept`. Therefore, `ProjectID` transitively determines `SupervisorName` and `SupervisorDept`.

Final 3NF Decomposition: Remove transitive dependencies into their own tables.

`Students(StudentID, StudentName, StudentMajor)` | PK: `StudentID`

`Supervisors(SupervisorID, SupervisorName, SupervisorDept)` | PK: `SupervisorID`

`Projects(ProjectID, ProjectTitle, ProjectType, SupervisorID)` | PK: `ProjectID`, FK: `SupervisorID` REFERENCES `Supervisors(SupervisorID)`

`Assignments(StudentID, ProjectID, Role, HoursWorked, StartDate, EndDate)` | PK: `{StudentID, ProjectID}`, FK: `StudentID` REFERENCES `Students(StudentID)`, FK: `ProjectID` REFERENCES `Projects(ProjectID)`

**Task 4.2: Advanced Normalization**

**Given Table:`CourseSchedule(StudentID, StudentMajor, CourseID, CourseName, InstructorID, InstructorName, TimeSlot, Room, Building)`**

**Business Rules:**

1. Each student has exactly one major → `StudentID → StudentMajor`

2. Each course has a fixed name → `CourseID → CourseName`

3. Each instructor has exactly one name → `InstructorID → InstructorName`

4. Each time slot in a room determines the building (rooms are unique across campus) → `{Room, TimeSlot} → Building` and importantly, `Room → Building` (if a room is in one building, it can't be in another)

5. Each course section is taught by one instructor at one time in one room → `{CourseID, TimeSlot} → InstructorID, Room` (Assuming `CourseID` here represents a specific section. A better attribute might be `SectionID`. We'll proceed with the given attributes.)

6. A student can be enrolled in multiple course sections → A student can appear with multiple `{CourseID, TimeSlot}` combinations.

**1. Determine the Primary Key:**

This is tricky. A student can be in many courses (`CourseID`), and a course (`CourseID`) occurs at a specific `TimeSlot`. Therefore, to uniquely identify a student's enrollment in a specific class meeting, the key is `{StudentID, CourseID, TimeSlot}`.

**2. List all Functional Dependencies (FDs):**

Based on the business rules and the key above:

1. `StudentID → StudentMajor`

2. `CourseID → CourseName`

3. `InstructorID → InstructorName`

4. `Room → Building` (From rule 4, since rooms are unique across campus)

5. `{CourseID, TimeSlot} → InstructorID, Room` (From rule 5)

6. `{StudentID, CourseID, TimeSlot} → StudentMajor, CourseName, InstructorID, InstructorName, Room, Building` (This is the full dependency based on our chosen key)

**3. Check if the table is in BCNF:**

BCNF requires that for every non-trivial FD X → Y, X must be a superkey.

Let's check each FD against this requirement:

`StudentID → StudentMajor`: `StudentID` is not a superkey of the entire table. Violates BCNF.

`CourseID → CourseName`: `CourseID` is not a superkey. Violates BCNF.

`InstructorID → InstructorName`: `InstructorID` is not a superkey. Violates BCNF.

`Room → Building`: `Room` is not a superkey. Violates BCNF.

`{CourseID, TimeSlot} → InstructorID, Room`: `{CourseID, TimeSlot}` is a candidate key (it determines `InstructorID` and `Room`, and through other FDs, all other attributes). This satisfies BCNF.

**Since multiple FDs violate BCNF, the table is not in BCNF.**

**4. Decompose it to BCNF:**

We decompose based on the violating FDs.

Decomposition 1: Based on `StudentID → StudentMajor`

R1(StudentID, StudentMajor) // X is now the key

R2(StudentID, CourseID, TimeSlot, CourseName, InstructorID, InstructorName, Room, Building) // Original table minus `StudentMajor`

Decomposition 2: Based on `CourseID → CourseName` (on R2)

R21(CourseID, CourseName) , X is now the key

R22(StudentID, CourseID, TimeSlot, InstructorID, InstructorName, Room, Building) , R2 minus `CourseName`

Decomposition 3: Based on `InstructorID → InstructorName` (on R22)\*\*

R31(InstructorID, InstructorName) ,X is now the key

R32(StudentID, CourseID, TimeSlot, InstructorID, Room, Building) , R22 minus `InstructorName`

Decomposition 4: Based on `Room → Building` (on R32)

R41(Room, Building) , X is now the key

R42(StudentID, CourseID, TimeSlot, InstructorID, Room) , R32 minus `Building`

Final Check: The remaining table is R42(StudentID, CourseID, TimeSlot, InstructorID, Room). Let's check its FDs.

The key is `{StudentID, CourseID, TimeSlot}`.

The FD `{CourseID, TimeSlot} → InstructorID, Room` exists. `{CourseID, TimeSlot}` is a superkey of this relation (it determines all other attributes). No other FDs violate BCNF.

**Final BCNF Schemas:**

1. `Students(StudentID, StudentMajor)`

2. `Courses(CourseID, CourseName)`

3. `Instructors(InstructorID, InstructorName)`

4. `Rooms(Room, Building)`

5. `Sections(CourseID, TimeSlot, InstructorID, Room)` FK: `CourseID` refs `Courses`, `InstructorID` refs `Instructors`, `Room` refs `Rooms`

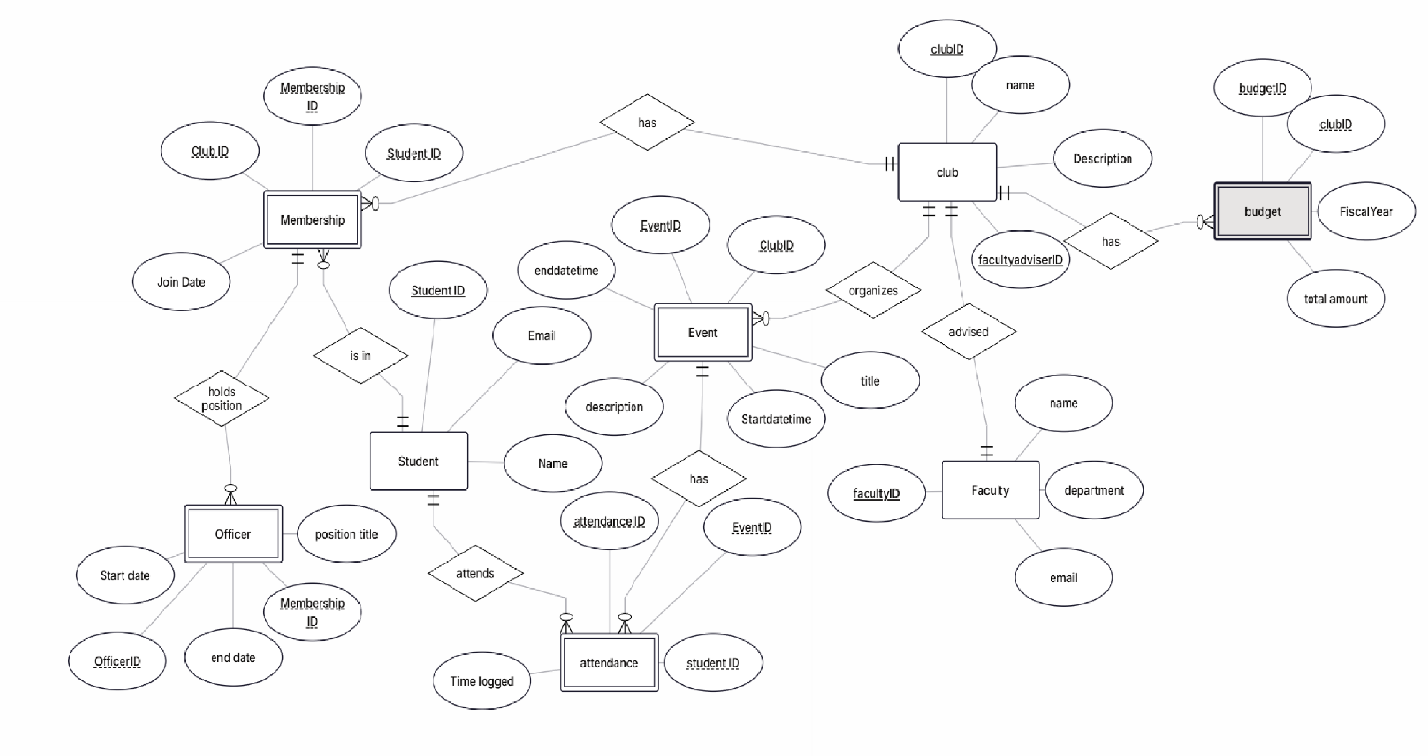
6. `Enrollments(StudentID, CourseID, TimeSlot)` FK: `StudentID` refs `Students`, `{CourseID, TimeSlot}` refs `Sections`

**5. Explain any potential loss of information:**

This decomposition is lossless. Because we decomposed based on functional dependencies, and the final table `Enrollments` contains a foreign key `{CourseID, TimeSlot}` that can be joined back to the `Sections` table to recover all the original information without any spurious tuples. The decomposition preserves all functional dependencies except possibly `{CourseID, TimeSlot} → InstructorID, Room`, which is preserved in the `Sections` table. The other FDs (`StudentID→Major`, `CourseID→Name`, etc.) are preserved in their respective tables. Therefore, there is no loss of information.

**Part 5: Design Challenge**

**Task 5.1**

**1)**

**2)** **This is the SQL-like schema derived from the ER diagram.**

Strong Entities:

* Students (StudentID, Name, Email)
* Clubs (ClubID, Name, Description, FacultyAdvisorID)
* Faculty (FacultyID, Name, Department, Email)

Weak/Associative Entities:

* Memberships (MembershipID, ClubID, StudentID, JoinDate)
  + FK: ClubID REFERENCES Clubs(ClubID)
  + FK: StudentID REFERENCES Students(StudentID)
  + *Unique Constraint (Candidate Key):* (ClubID, StudentID) to prevent duplicate memberships.
* Officers (OfficerID, MembershipID, PositionTitle, StartDate, EndDate)
  + FK: MembershipID REFERENCES Memberships(MembershipID)
* Events (EventID, ClubID, Title, StartDateTime, EndDateTime, Description)
  + FK: ClubID REFERENCES Clubs(ClubID)
* Attendance (AttendanceID, EventID, StudentID, TimeLogged)
  + FK: EventID REFERENCES Events(EventID)
  + FK: StudentID REFERENCES Students(StudentID)
* Budgets (BudgetID, ClubID, FiscalYear, TotalAmount)
  + FK: ClubID REFERENCES Clubs(ClubID)
  + *Unique Constraint (Candidate Key):* (ClubID, FiscalYear) to ensure one budget per club per year.

**3)Design Decision with Multiple Valid Options**

* Decision: Modeling the Officers entity.
* Option 1: (The chosen one) Have a separate Officers table that references the Memberships table. This validates that only a member of a club can be an officer of that club. It also allows us to easily track the history of officer positions for a member (e.g., StartDate and EndDate).
* Option 2: Add an OfficerPosition column (e.g., 'President', 'Treasurer', NULL) directly in the Memberships table.
* Justification for Choice: I chose Option 1 because it is far more flexible and robust. Option 2 has significant limitations:
  1. It cannot track the history of who held a position over time; it only shows the current state.
  2. It makes it difficult to represent members holding multiple positions or a single position being shared by multiple members (e.g., "Co-Presidents").
  3. It poorly models the fact that being an officer is a specific role with its own metadata (start/end dates) attached to a membership, not just a simple attribute. Option 1 correctly models this as a weak entity and avoids these problems.

**4) Example Queries (in English)**

1. Membership Query: "Find the names of all members of the 'Quantum Computing Club' who joined after September 1, 2023."
2. Event & Logistics Query: "List all club events happening in the first week of October 2024, along with the name of the club hosting them and the building and room number where they are reserved."
3. Financial Query: "For the 'AI Ethics Society', show the total expenses incurred in the 2024 fiscal year that exceeded $100."