Task 1.1.

Relation A.

1. Super keys: {EmpID}, {SSN}, {Email}, {EmpID, SSN}, {EmpID, Email}, {EmpID, SSN, Email}, {SSN, Email, Phone}, {Email, Name, Department}, {EmpID, Phone}, {SSN, Salary} etc
2. Candidate Keys: {EmpID}, {SSN}, {Email}.
3. EmpID. Because this is the value within the company. Others are not suitable for this option, because SSN it is something personal, private and confidential; Phone, Name, Email, Department and Salary can be changed.
4. Yes, they can. For example, if they have a corporate number, a shared department phone number.

Relation B.

Primary key: {StudentID, CourseCode, Semester, Year}

StudentID is needed to identify the student, namely which student took the course.

CourseCode is needed to determine which course the student studied in. Without this, we will simply know which semester the student studied in.

Semester and Year – these attributes are necessary to determine which semester and year the student studied in. By 1st Business Rule, the student can take the same course in different semesters. Without that we will not understand which year the student studied in.

Candidate Key: {StudentID, CourseCode, Section, Semester, Year}

Task 1.2.

Student(StudentID, Name, Email, Major, AdvisorID) – foreign keys: AdvisorID.

FK**:** Student(AdvisorID) REFERENCES Professor(ProfID)

Course(CourseID, Title, Credits, DepartmentCode) - foreign keys: DepartmentCode.

FK: Course(DepartmentCode) REFERENCES Department(DeptCode)

Department(DeptCode, DeptName, Budget, ChairID) - foreign keys: ChairID.

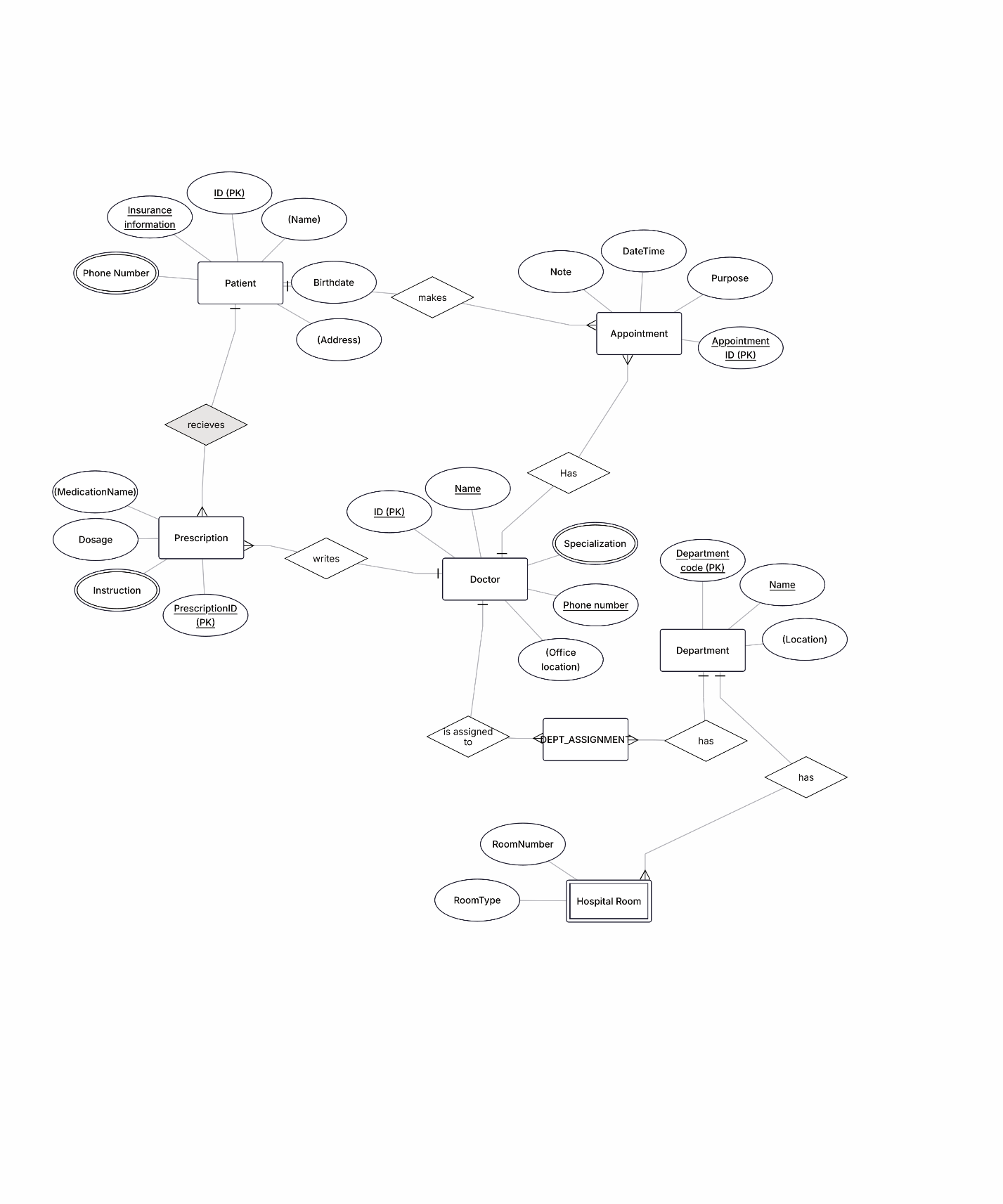
FK: Department(ChairID) REFERENCES Professor(ProfID)

Enrollment(StudentID, CourseID, Semester, Grade) - foreign keys: StudentID, CourseID.

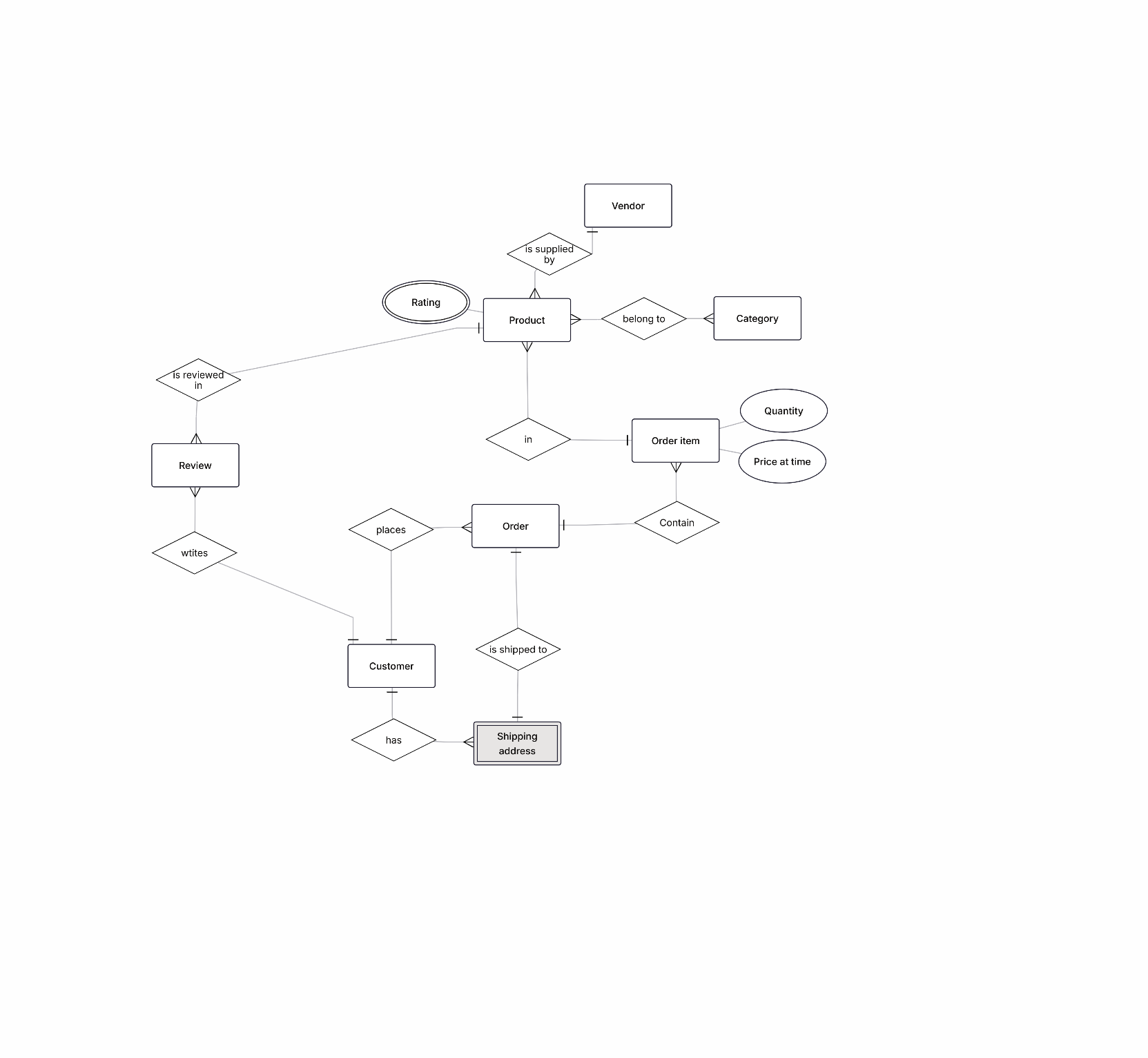
FK: Enrollment(StudentID) REFERENCES Student(StudentID)

FK: Enrollment(CourseID) REFERENCES Course(CourseID)

Task 2.1.



Task 2.2.



2) Weak entity – Shipping address. The delivery address does not make sense and cannot be uniquely identified without knowing which customer it belongs to. E.g. there can be 1000 records in the database with AddressID = 1, but only one record with a combination of CustomerID = 555 and AddressID = 1.

3) Products and Categories. Product can belong to many Categories, e.g. a laptop is in the Electronics category and in the Gift category. But Category has many products.

Task 4.1.

1. StudentID -> StudentName, StudentMajor

ProjectID -> ProjectTitle, ProjectType

SupervisorID -> SupervisorName, SupervisorDept

{StudentID, ProjectID} -> Role, HoursWorked, StartDate, EndDate

1. Redundancy: StudentName, StudentMajor, ProjectTitle, ProjectType, SupervisorName, SupervisorDept

Update anomaly: If a student has changed his major (StudentMajor), you need to update this in all the lines where he is mentioned. Skipping at least one line will create a contradiction.

Delete anomaly: f you delete the last student record in a project, you may lose data about the student, the project, or the supervisor, even if they still exist.

Insert anomaly: It is not possible to add a new student who is not yet involved in the project, because StudentProject requires a ProjectID.

1. No 1NF violations. All values are atomic. Nothing that could be fixed.
2. Primary key: (StudfentID, ProjectID)

Partial dependencies:

StudfenID -> StudentName, StudentMajor (depends only on StudentID, not on ProjectID)

ProjectID -> ProjectTitle, ProjectType

SupervisorID -> SupervisorName, SupervisorDept

2NF:

Student(StudentID, StudentName, StudentMajor)

Project(ProjectID, ProjectTitle, ProjectType, SupervisorID)

Supervisor(SupervisorID, SupervisorName, SupervisorDept)

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

1. After 2NF decomposition there is no transitive dependencies. The final form of 3NF is the same as 2NF

Task 4.2.

1. PK: (StudentID, CourseID, TimeSlot, Room)
2. StudentID -> StudentMajor

CourseID -> CourseName

InstructorID -> InstructorName

Room -> Building

(CourseID, TimeSlot, Room) -> InstructorID

1. Not BCNF. FDs do not contain superkey.
2. 1 step, make student table: Student(StudentID, StudentName)

Remaining \_table = (StudentID, CourseID, CourseName, InstructorID, InstructorName, TimeSlot, Room, Building)

2 step, make course table: Course(CourseID, CourseName)

Remaining\_table = (StudentID, CourseID, InstructorID, InstructorName, TimeSlot, Room, Building)

3 step, make instruction table: Instructor(InstructorID, InstructorName)

Remaining \_table = (StudentID, CourseID, InstructorID, TimeSlot, Room, Building)

4 step, make room table: Room(Room, Building)

Remaining \_table = (StudentID, CourseID, InstructorID, TimeSlot, Room)

5 step, make course section table: CourseSection(CourseID, TimeSlot, Room, InstructorID)

6 step, add student: Enrollment(StudentID, CourseID, TimeSlot, Room)

BCNF:

Student(StudentID, StudentName)

Course(CourseID, CourseName)

Instructor(InstructorID, InstructorName)

Room(Room, Building)

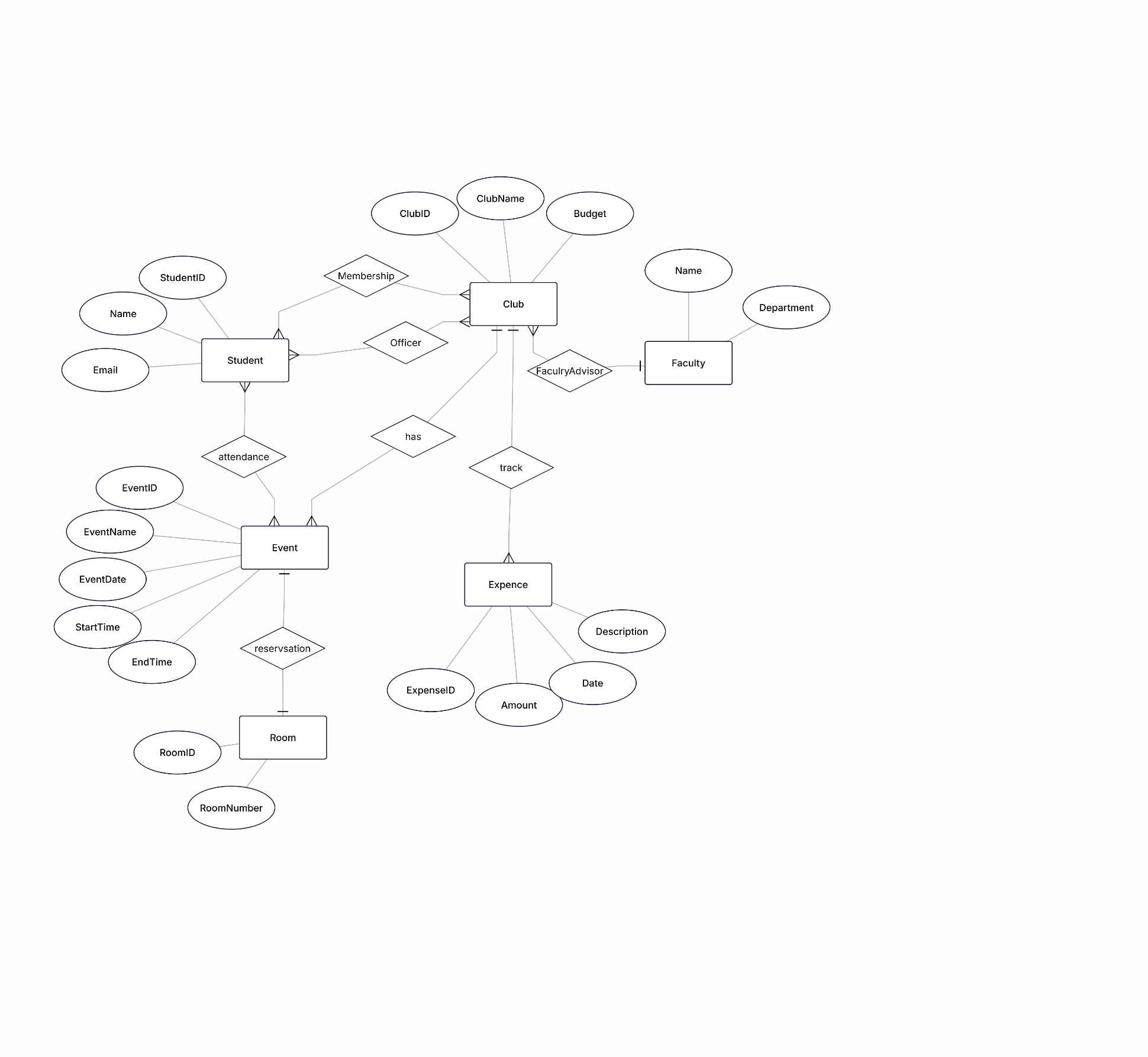
CourseSection(CourseID, TimeSlot, Room, InstructorID)

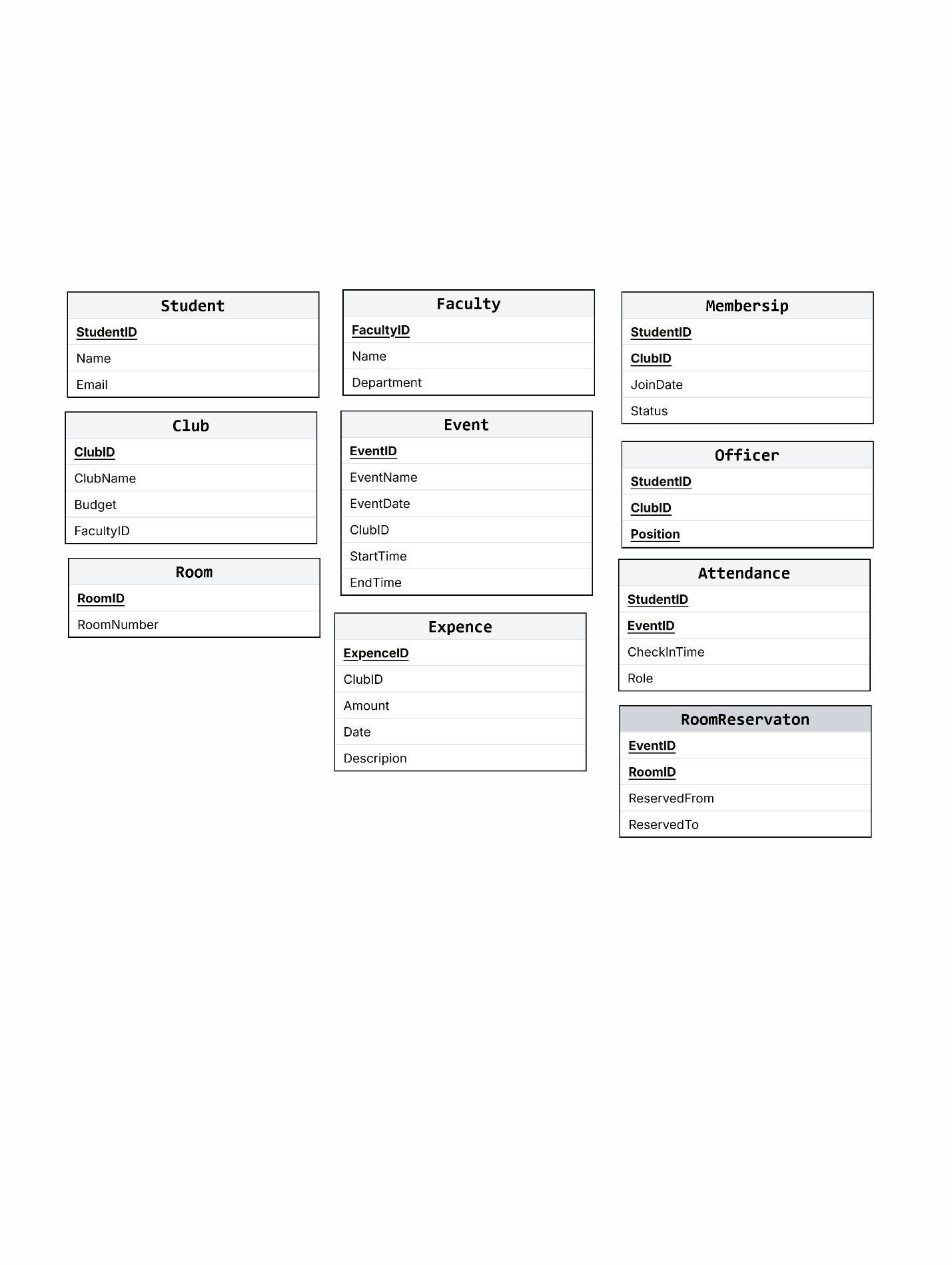
Enrollment(StudentID, CourseID, TimeSlot, Room)

1. Information is not lost with the classic join of all tables.

Task 5.1.

1)



2)

3) Question: how to store Officer Positions?

Option 1: The Position attribute is directly in Membership (each member can be an officer).

Option 2: Separate Officer table.

Choice: a separate Officer table,because:

* A student can be an ordinary member and hold several positions at the same time;
* It allows you to keep a history of positions regardless of membership (for example, a former officer, but the member remains).

4) 1. How would you write a query to find all the rooms that are available on a certain day and time?

2. How do I calculate the average budget of clubs with more than 100 active members?

3. How do I find students who are not members of any club, but have attended at least one event?