Task 1.1

1. {EmpID}, {SSN}, {Email}, {Phone}, {EmpID, Name}, {Email, Name}
2. EmpID, SSN, Email, Phone
3. EmpID, because it's acting as a unique order number of each employee
4. No, because each employee possesses its own phone number.

1. StudentID, CourseCode, Section, Semester, Year
2. **StudentID** is necessary for identifying the exact student
   CourseCode shows which course the student took
   Section - different sections in the same course
   Semester is the exact part of the year of taking course
   Year - because student can take courses in every year
3. There is no additional candidate key that identify all attributes in one.

Task 1.2. Foreign Keys

- Student.Major → Department.DeptCode
- Student.AdvisorID → Professor.ProfID
- Professor.Department → Department.DeptCode
- Course.DepartmentCode → Department.DeptCode
- Department.ChairID → Professor.ProfID
- Enrollment.StudentID → Student.StudentID
- Enrollment.CourseID → Course.CourseID

Task 2.1
Entities

- Patient (strong)
- Doctor (strong)
- Department (strong)
- Appointment (weak)
- Prescription (weak)
- Hospital Room (weak)

Task 2.1 Attributes

Patient (strong)

PatientID (PK) — simple
Name — composite (First name, Last name)
Birthdate — composite
Address — composite (Street, City, State, Zip)
PhoneNumbers — multi-valued
InsuranceInfo — simple

Task 2.1 Attributes

Doctor (strong)

DoctorID (PK) — simple
Name — composite
Specializations — multi-valued
PhoneNumbers — multi-valued
OfficeLocation — composite

Task 2.1 Attributes

Department (strong)

DepartmentCode (PK) — simple
Name — simple
Location — simple

Task 2.1 Attributes

Appointment (weak)

DateTime — composite
Purpose — simple
Notes — simple
PatientID (FK), simple
DoctorID (FK), simple

Task 2.1 Attributes

Prescription (weak)

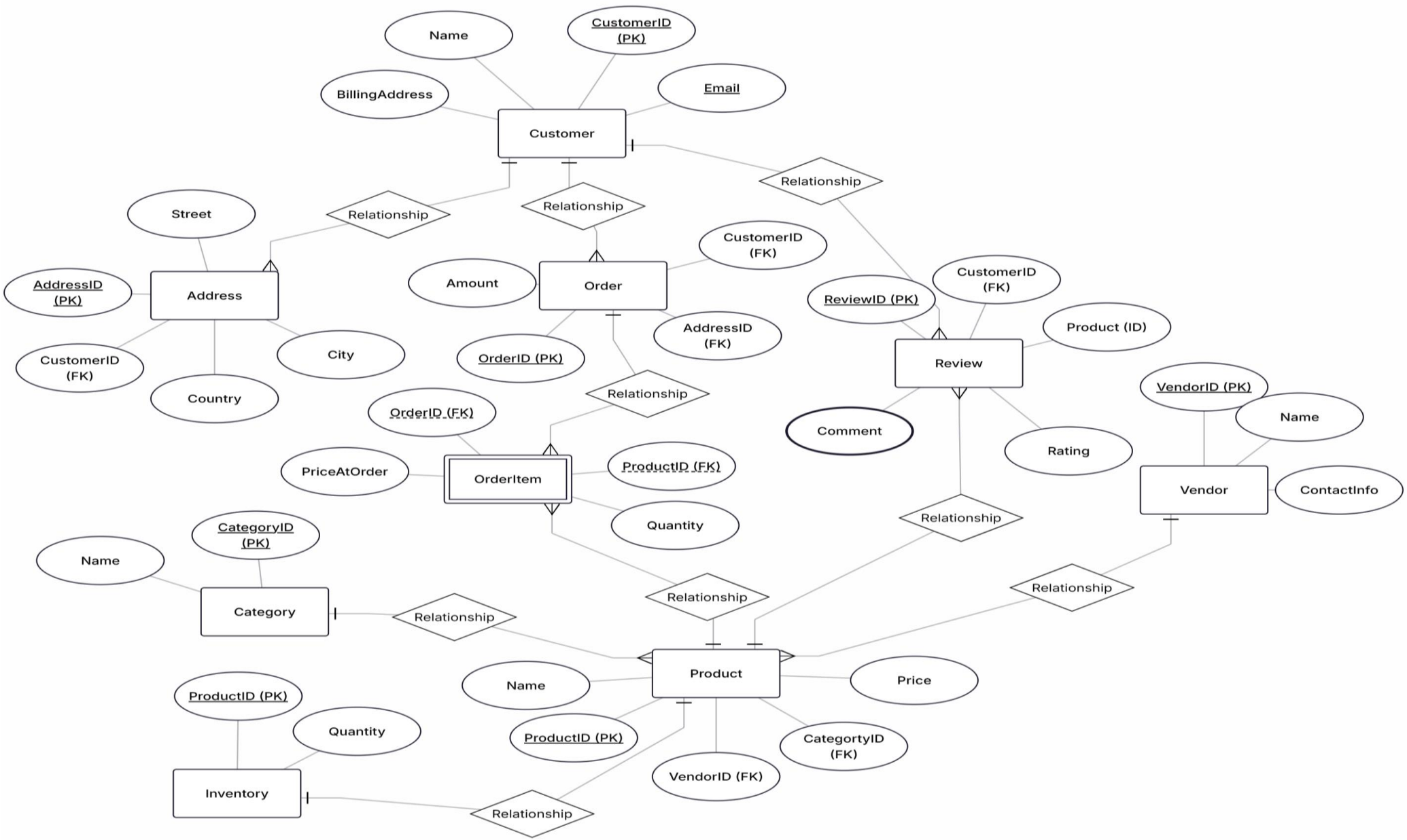
Medication — simple
Dosage — simple
Instructions — simple
AppointmentID (FK) — simple

Task 2.1
Attributes

Hospital Room (weak)

RoomNumber (simple)
DepartmentCode (simple)

Task 2.2. ER diagram



Task 2.2. Weak entity

OrderItem is a weak entity:

1. It has no independent meaning outside an Order — an order line only exists as part of an order.
2. Its natural primary key is composite (*OrderID, ProductID*)
3. It **depends on** Order for identification and carries attributes (*Quantity, PriceAtOrder*) that describe the relationship between an *Order* and a *Product*

Task 2.2. Many-to-many relationships that *need* attributes

1. Order ⟷ *Product* via *OrderItem* (needs attributes):

Attributes required: *Quantity, PriceAtOrder*
Rationale: the relationship itself stores data that varies per occurrence

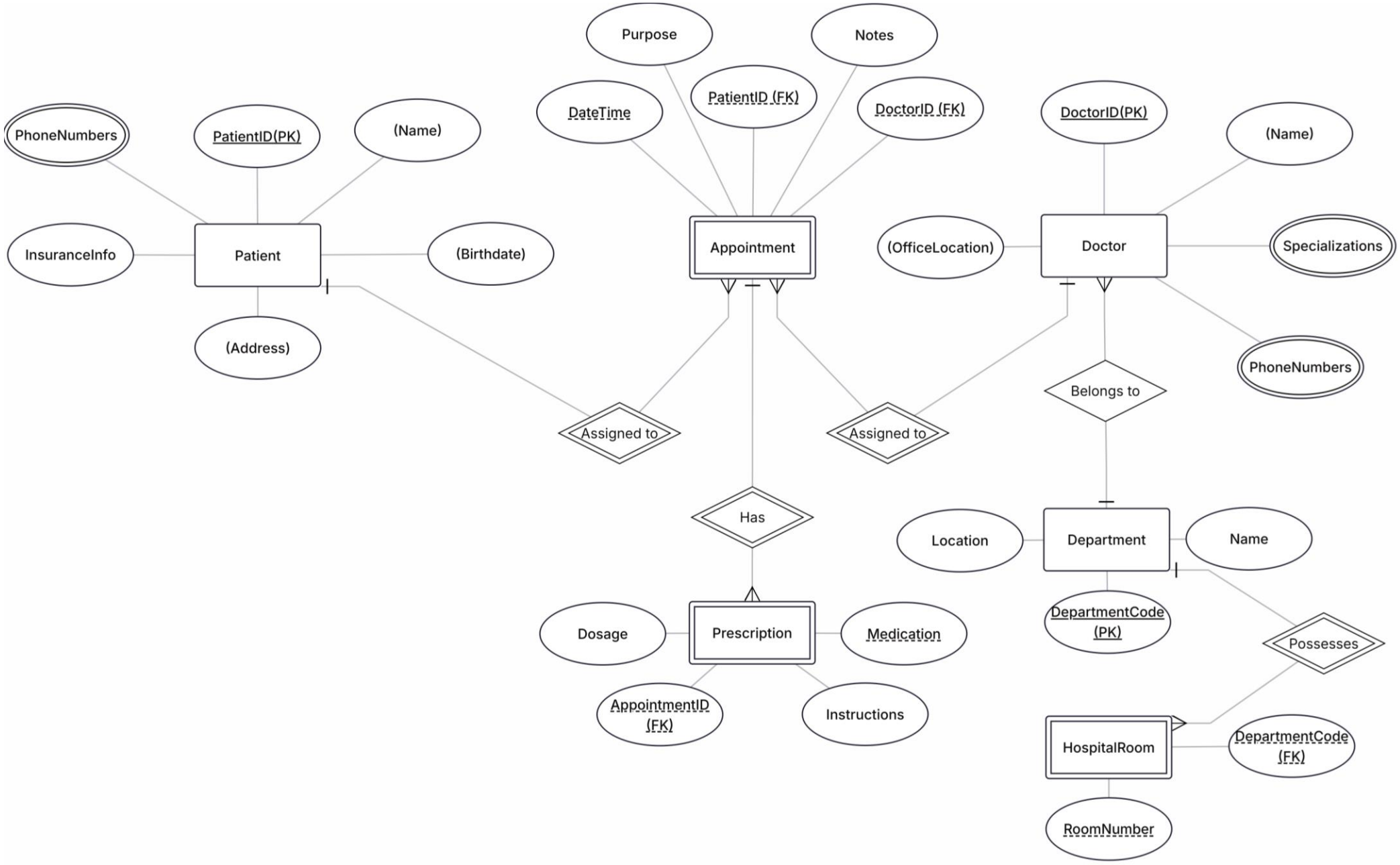
2. Customer ⟷ *Product* via *Review*

Attributes: *Rating, Comment, ReviewDate*
Rationale: the relationship (customer reviewed product) has data attached.

Task 2.3. Relationships & Cardinalities

One-to-Many	Many-to-Many
Patient —> Appointment – a patient can have several appointment, there is no limit for this case, when each appointment is connected with exactly one patient	Doctor —> Specialization - a Doctor can have multiple Specializations, and a specialization can belong to many doctors
Doctor —> Appointment – the same as with patient	
Department —> Doctor – each doctor has his own department and each department contain many doctors	
Department —> Room – each room belongs to exactly one department, when each department have many rooms	
Patient —> Prescription - each patient can have many prescription	
Doctor —> Prescription – a doctor can write many prescriptions but each prescription is written by one doctor	
Appointment —> Prescription – in one appoinment doctor can give to patient several prescriptions	

Task 2.4. ER Diagram



Task 4.1. Functional dependencies

1. *StudentID → StudentName, StudentMajor*
2. *ProjectID → ProjectTitle, ProjectType, SupervisorID*
3. *SupervisorID → SupervisorName, SupervisorDept*
4. *{StudentID, ProjectID} → Role, HoursWorked, StartDate, EndDate*

(4) says the student–project assignment determines the role, hours and dates.
From (2) and (3) we get transitive consequences

Task 4.1. Redundancy & anomalies

Redundancy examples

1. Student attributes (*StudentName, StudentMajor*) are repeated on every row for that student (one row per project).
2. Project attributes (*ProjectTitle, ProjectType, SupervisorID*) are repeated for every student on that project.
3. Supervisor attributes (*SupervisorName, SupervisorDept*) are repeated for every project supervised by that supervisor.

Task 4.1. Redundancy & anomalies

Update anomaly

If supervisor “Dr. X” changes their office/last name, you must update many rows (every project row that includes that supervisor). If you forget one, data becomes inconsistent.

Insert anomaly

If you want to add a new project into the database but no student has started it yet, you cannot insert a *ProjectTitle/SupervisorID* without inventing a fake StudentID (unless the schema allows NULLs).

Delete anomaly

If the last student working on a project is deleted, you lose the project and supervisor information for that project because it only existed as repeated data in the assignment rows.

Task 4.1. Normalization check

1NF

1NF requires atomic attribute values and no repeating groups. The relation as shown appears to be in 1NF (attributes are atomic). Assuming atomicity, 1NF is satisfied.

2NF

Because the PK is composite (*StudentID, ProjectID*), any FD where a *single* part of the key determines non-key attributes is a partial dependency:
StudentID → *StudentName, StudentMajor* (partial: depends only on part *StudentID*)
ProjectID → *ProjectTitle, ProjectType, SupervisorID* (partial: depends only on *ProjectID*)
Partial dependencies violate 2NF.

Decompose to remove partial dependencies

Split off the attributes that depend only on StudentID or only on ProjectID:
Students(StudentID PK, StudentName, StudentMajor)
Projects(ProjectID PK, ProjectTitle, ProjectType, SupervisorID)
StudentProject(StudentID FK, ProjectID FK, Role, HoursWorked, StartDate, EndDate, PK(StudentID, ProjectID))

Task 4.1. Normalization check

3NF

Transitive dependency

From the earlier FDs: *ProjectID* → *SupervisorID* and *SupervisorID* → *SupervisorName*, *SupervisorDept*. That makes *SupervisorName, SupervisorDept* transitively dependent on *ProjectID* (and therefore on the key). This violates 3NF (non-key attribute depends on another non-key attribute).

Fix (move supervisor attributes to their own table)

Create a supervisor table and reference it from Projects:
Supervisors(*SupervisorID PK, SupervisorName, SupervisorDept*)
Projects(*ProjectID PK, ProjectTitle, ProjectType, SupervisorID FK* → *Supervisors.SupervisorID*)

Task 4.1. Normalization check

FINAL 3NF

Students(*StudentID PK, StudentName, StudentMajor*)

Supervisors(*SupervisorID PK, SupervisorName, SupervisorDept*)

Projects(*ProjectID PK, ProjectTitle, ProjectType, SupervisorID FK*)

StudentProject(*StudentID FK → Students.StudentID, ProjectID FK → Projects.ProjectID, Role, HoursWorked, StartDate, EndDate, PK(StudentID, ProjectID)*)

Task 4.2. Primary key

A row records **one student's enrollment in one course *section***. A *course section* is identified by the combination of (CourseID, TimeSlot, Room) (business rule: “each course section is taught by one instructor at one time in one room”). So, an enrollment row is uniquely identified by:
Primary key = {StudentID, CourseID, TimeSlot, Room}

Task 4.2. Functional dependencies

1. *StudentID → StudentMajor*
2. *CourseID → CourseName*
3. *InstructorID → InstructorName*
4. *Room → Building (rooms are unique across campus ⇒ a room determines the building)*
5. *CourseID, TimeSlot, Room → InstructorID (a course section is taught by one instructor at one time in one room)*
6. *Trivial / implied by PK: {StudentID, CourseID, TimeSlot, Room} → (all other attributes)*

Task 4.2. Is in BCNF?

A relation is BCNF if every non-trivial FD X → Y has X a superkey.

1. *StudentID* → *StudentMajor*: *StudentID* is not a superkey of the table → violates BCNF.
2. *CourseID* → *CourseName*: violates BCNF.
3. *InstructorID* → *InstructorName*: violates BCNF.
4. *Room* → *Building*: violates BCNF.
5. *CourseID, TimeSlot, Room* → *InstructorID*: left side is not a superkey of the table (it does not determine *StudentID*) → violates BCNF.

Task 4.2. Decomposition

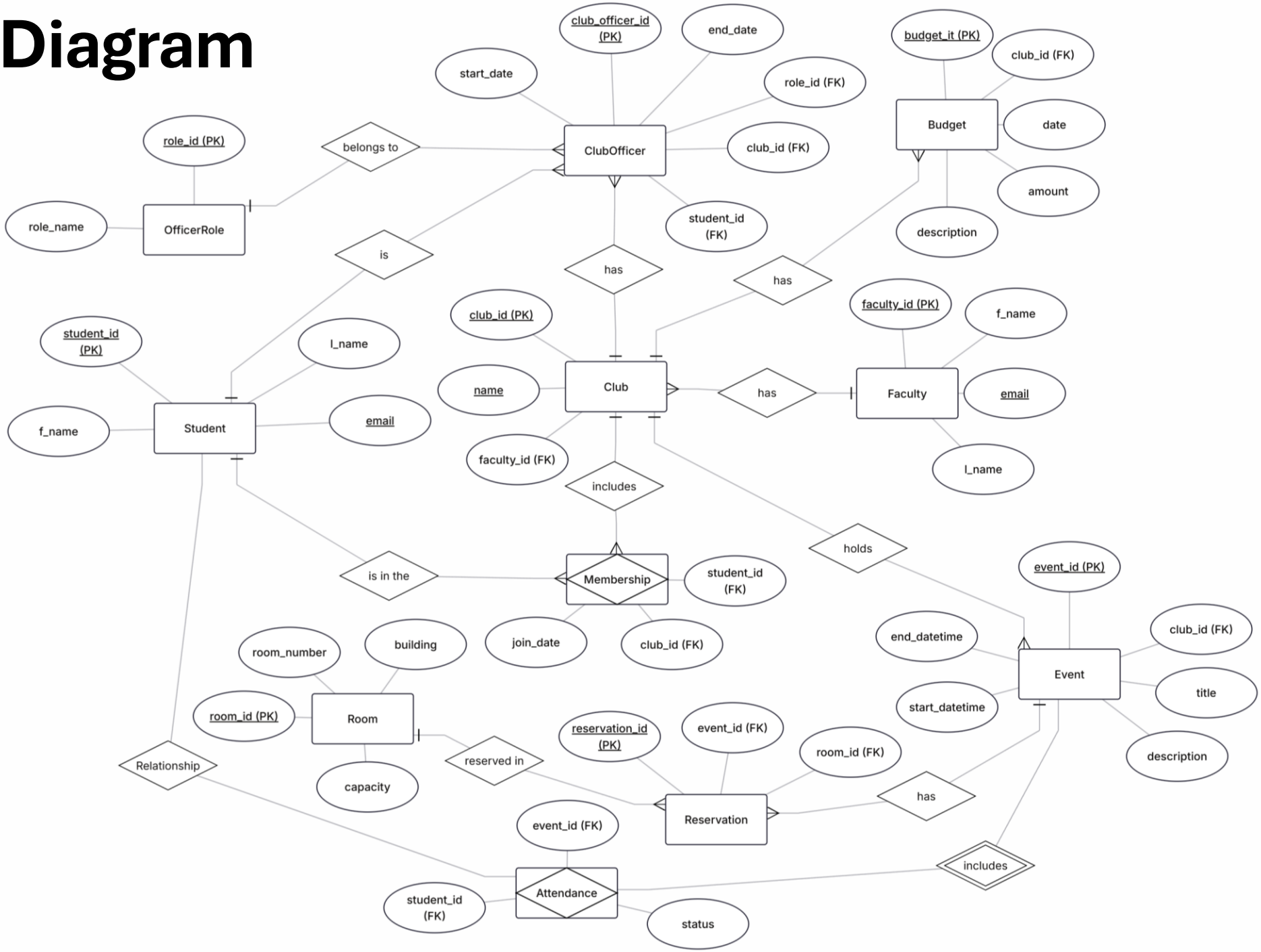
1. **Student**(*StudentID PK, StudentMajor*)
from *StudentID* → *StudentMajor*
2. **Course**(*CourseID PK, CourseName*)
from *CourseID* → *CourseName*
3. **Instructor**(*InstructorID PK, InstructorName*)
from *InstructorID* → *InstructorName*
4. **Room**(*Room PK, Building)*
from *Room* → *Building*
5. **Section**(*CourseID, TimeSlot, Room, InstructorID*)
PK = {*CourseID, TimeSlot, Room*}
FD: {*CourseID, TimeSlot, Room*} → *InstructorID*
6. **Enrollment**(*StudentID, CourseID, TimeSlot, Room*)
PK = {*StudentID, CourseID, TimeSlot, Room*}

Task 4.2. Loss of information

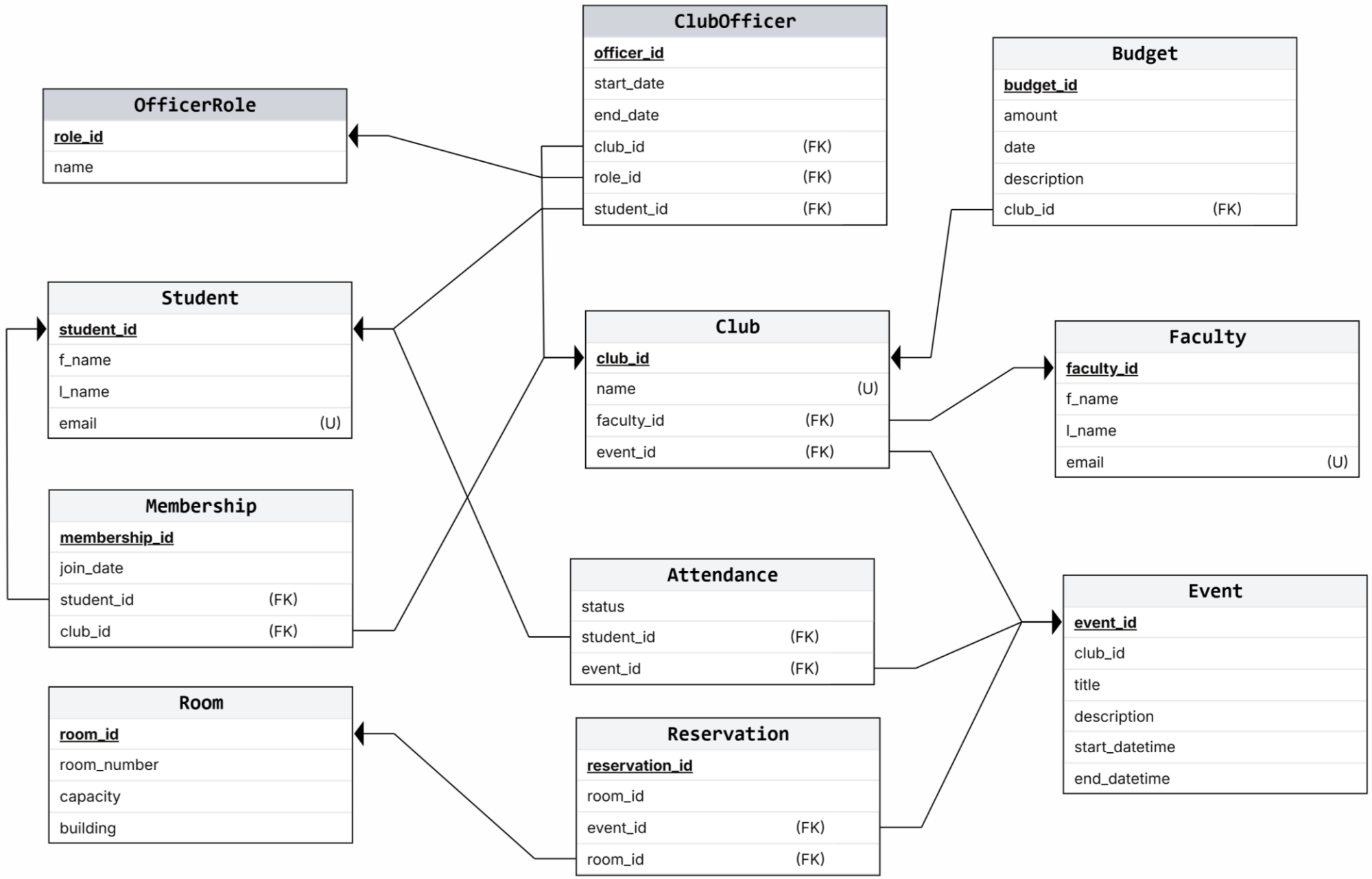
This decomposition is **lossless**.

Reason: when we decompose R into *Enrollment* and *Section* and the other entity tables, we keep the join attributes that link relations (*CourseID, TimeSlot, Room and StudentID*) so joining the decomposed relations on those keys recovers the original tuples. In relational theory terms, each decomposition step used an FD whose left side is a key of one of the projected relations, which guarantees a lossless join.

Task 5.1. ER Diagram



Task 5.1. Normalization



Task 5.1. Proofs

All relations in the database schema satisfy the requirements of **First Normal Form (1NF)** (no repeating groups, atomic attributes)

Second Normal Form (2NF) (all non-key attributes fully depend on the whole key for composite PKs)

Third Normal Form (3NF) (no transitive dependencies; non-key attributes depend only on the primary key).

Therefore, the database schema is normalized up to **Third Normal Form (3NF)**.

Task 5.1. Design decision and rationale

Officer representation — one join table with role + term vs. storing role in Membership

Option 1: store officer role as field on Membership
Option 2: separate table ClubOfficer linking Student → Club → Role

Choice: Option 2 because officer assignments are a time-bounded responsibility, and a student may be a regular member and also an officer at different times. A separate table keeps historical officer records and allows multiple officer roles over time. It's more flexible and keeps membership distinct from office-holding.

Task 5.1. Three example queries

“Retrieve the names of clubs that currently do not have any assigned faculty advisor”

“List the students who have attended at least three different events organized by the Math Club during the past semester”

“Find all rooms that are reserved more than once at overlapping times, to detect scheduling conflicts”