#Lab1 for Database
#Part 1

Task 1.1

Relation A:

1) **Superkeys:**
   1. **{**EmpID**}**
   2. **{**SSN**}**
   3. **{**Email**}**
   4. **{**EmpID**,**Name**}**
   5. {SSN,Department}
   6. {EmpID,Phone}

2) **Candidate keys:**
{EmPID},{SSN},{Email},{Phone} only if the business rules guarantee phone numbers are unique per employee
3) **Primary key:**
Choose **`{EmpID}`** as PK: it's system-generated, stable, and non-sensitive (unlike SSN), and unlike Email it won't change if the employee changes their address.
4) The sample shows distinct numbers but no business rule states uniqueness; shared desk lines or shared contact numbers can occur. So Phone should not be treated as a key.

Relation B:

Task 1.2

All foreign keys:

1. Student → Professor
   `Student.AdvisorID` → `Professor.ProfID`

2. Student → Department
   `Student.Major` → `Department.DeptCode`

3. Professor → Department
   `Professor.Department` → `Department.DeptCode`

4. Course → Department
   `Course.DepartmentCode` → `Department.DeptCode`

5. Department → Professor
   `Department.ChairID` → `Professor.ProfID`

6. Enrollment → Student
   `Enrollment.StudentID` → `Student.StudentID`

7. Enrollment → Course
   `Enrollment.CourseID` → `Course.CourseID`

#Part 2

Task 2.1

1)

Strong Entities:

- Patient
- Doctor
- Department
- Appointment
- Prescription
- Hospital Room

Weak Entities (exist only because of multivalued/composite attributes):

- Phone Number (for Patient and Doctor)
- Specialization (for Doctor)

2) Attributes (with classification)

Patient

- PatientID *(*simple, PK*)*
- Name *(*composite: FirstName, LastName*)*
- Birthdate *(*simple*)*
- Address *(*composite: Street, City, State, Zip*)*
- PhoneNumbers *(*multi-valued*)*
- InsuranceInfo *(*simple or composite depending on detail*)*

Doctor

- DoctorID *(*simple, PK*)*
- Name *(*composite*)*
- Specializations *(*multi-valued*)*
- PhoneNumbers *(*multi-valued*)*
- OfficeLocation *(*composite: Building, RoomNo*)*

Department

- DeptCode *(*simple, PK*)*
- DeptName *(*simple*)*
- Location *(*simple*)*

Appointment

- AppointmentID *(*simple, PK)
- DateTime *(*simple*)*
- Purpose *(*simple*)*
- Notes *(*simple*)*

Prescription

- PrescriptionID *(*simple, PK*)*
- MedicationName *(*simple*)*
- Dosage *(*simple*)*
- Instructions *(*simple*)*

HospitalRoom

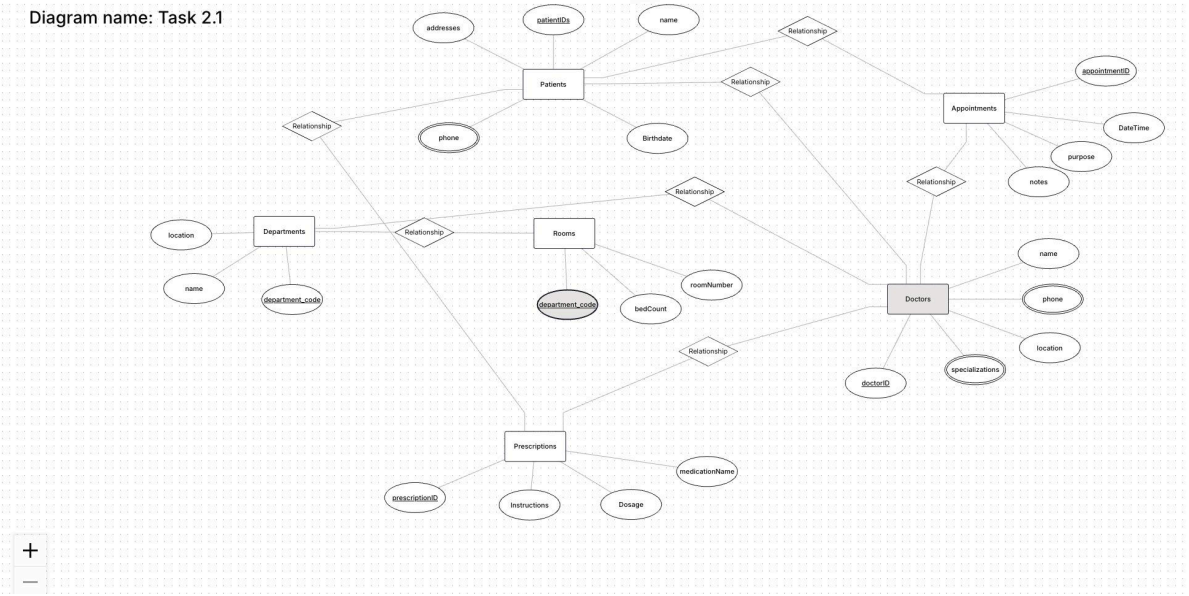
- RoomNumber *(*partial key, weak*)*
- DeptCode *(*FK, part of PK*)*
- Attributes: BedCount, RoomType *(*simple*)*

---

3)Relationships with Cardinalities

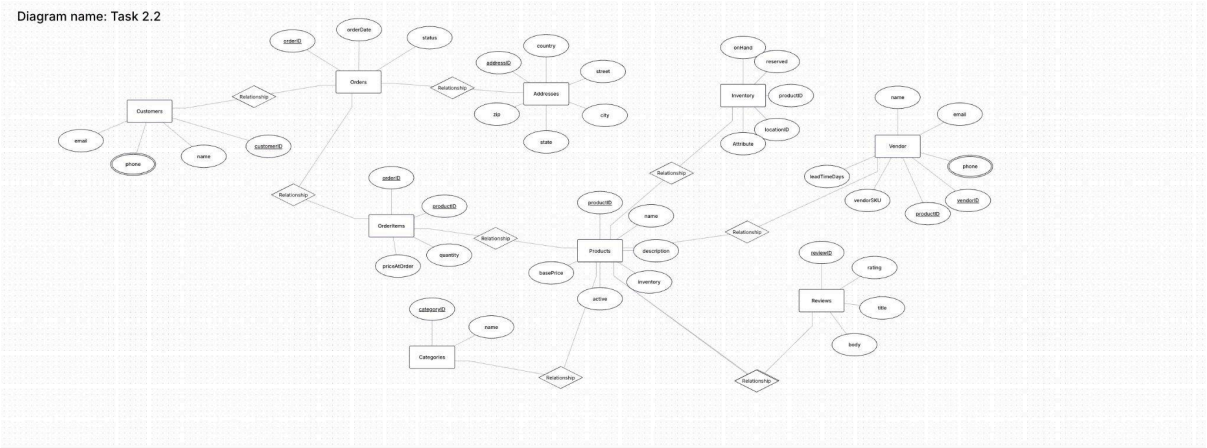
1. **Patient — Appointment — Doctor**
    - Patient has many Appointments
    - Doctor has many Appointments
    - Cardinality: **M:N** (resolved through Appointment)
2. **Patient — Prescription — Doctor**
    - Doctor prescribes many Prescriptions
    - Patient can receive many Prescriptions
    - Cardinality: **M:N** (resolved through Prescription)
3. **Doctor — Department**
    - A Doctor works in one Department
    - A Department has many Doctors
    - Cardinality: **1:N**
4. **HospitalRoom — Department**
    - A Department has many Rooms
    - Each Room belongs to one Department
    - Cardinality: **1:N**
    - HospitalRoom is weak (identified by DeptCode + RoomNumber)
5. **Patient — PhoneNumber**
    - One Patient can have multiple PhoneNumbers
    - Cardinality: **1:N**
6. **Doctor — PhoneNumber**
    - One Doctor can have multiple PhoneNumbers
    - Cardinality: **1:N**
7. **Doctor — Specialization**
    - One Doctor can have many Specializations
    - One Specialization can belong to many Doctors
    - Cardinality: **M:N**

4),5)



Task 2.2

1)



2)Weak entity : OrderItem is weak: identified by `(OrderID, ProductID)`; it depends on owner Order.

3)Many-to-many with attributes:

OrderItem carries attributes `Quantity`, `PriceAtOrder` for the M:N between Order and Product.

#Part 4

Task 4.1

1) Functional dependencies (FDs)

- StudentID → StudentName, StudentMajor
- ProjectID → ProjectTitle, ProjectType, SupervisorID
- SupervisorID → SupervisorName, SupervisorDept
- (StudentID, ProjectID) → Role, HoursWorked, StartDate, EndDate
- (By transitivity) ProjectID → SupervisorName, SupervisorDept

2) Problems (redundancy & anomalies)

- **Redundancy:** Student and Supervisor names/departments repeat across many rows; Project info repeats per student on the same project.
- **Update anomaly:** Changing a supervisor's department requires multiple row updates.
- **Insert anomaly:** Can't record a new project (or supervisor) until some student is assigned.
- **Delete anomaly:** If the last student on a project is removed, project/supervisor info is lost.

3) 1NF

- All attributes are atomic as written → **1NF satisfied**. (If a student could have multiple roles per project, that's multiple rows, not a repeating group.)

4) 2NF

- **Natural PK:** (StudentID, ProjectID) (assignment row).
  Partial deps:
  - StudentID → StudentName, StudentMajor
  - ProjectID → ProjectTitle, ProjectType, SupervisorID
- **Decompose to 2NF:**
  - **Student**(**StudentID**, StudentName, StudentMajor)
  - **Supervisor**(**SupervisorID**, SupervisorName, SupervisorDept)
  - **Project**(**ProjectID**, ProjectTitle, ProjectType, SupervisorID)
  - **StudentProject**(**StudentID**, **ProjectID**, Role, HoursWorked, StartDate, EndDate)

5) 3NF

- **Transitive:** ProjectID → SupervisorID → SupervisorName, SupervisorDept — already isolated via **Supervisor** table.
- **Final 3NF schema (PKs underlined, FKs noted):**
  - **Student**(**StudentID**, StudentName, StudentMajor)
  - **Supervisor**(**SupervisorID**, SupervisorName, SupervisorDept)
  - **Project**(**ProjectID**, ProjectTitle, ProjectType, SupervisorID) // FK→Supervisor
  - **StudentProject**(**StudentID**, **ProjectID**, Role, HoursWorked, StartDate, EndDate) // FKs→Student, Project

*(If your instructor expects a student to have **multiple roles on the same project**, extend PK of StudentProject to (__StudentID__, __ProjectID__, __Role__).)*

Task 4.2

1) Primary key (tricky part)

A **section** is uniquely identified by (CourseID, TimeSlot, Room). An **enrollment row** identifies which student is in that section.
→ **PK:** {StudentID, CourseID, TimeSlot, Room}

2) FDs (from business rules)

- StudentID → StudentMajor
- CourseID → CourseName
- InstructorID → InstructorName
- Room → Building (rooms unique across campus → building determined)
- (CourseID, TimeSlot, Room) → InstructorID (each section has one instructor)
- (CourseID, TimeSlot, Room) → *(by above)* InstructorName (via InstructorID)

3) BCNF check

The big table violates BCNF because several non-key determinants exist (e.g., StudentID → StudentMajor, CourseID → CourseName, Room → Building, InstructorID → InstructorName).

4) BCNF decomposition (lossless, dependency-preserving where possible)

- **Student**(**StudentID**, StudentMajor)
- **Course**(**CourseID**, CourseName)

- **Instructor**(**InstructorID**, InstructorName)
- **Room**(**Room**, Building)
- **Section**(**CourseID**, **TimeSlot**, **Room**, InstructorID) // FK Room→Room, InstructorID→Instructor
- **Enrollment**(**StudentID**, **CourseID**, **TimeSlot**, **Room**) // FKs Student→Student, (CourseID,TimeSlot,Room)→Section

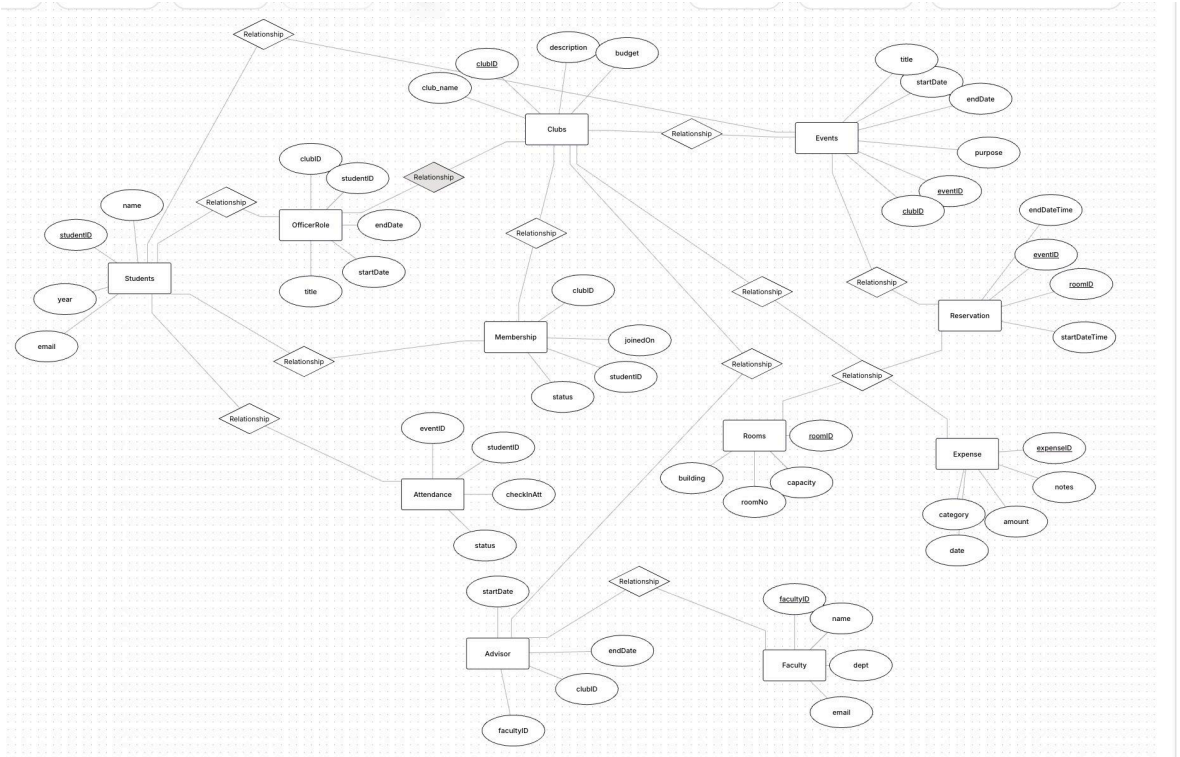
5) Information loss?

- **Lossless join:** Yes (via PK/FK joins through keys like Section and Enrollment).
- **Dependency preservation:** Most FDs preserved across the decomposed tables; the join of Section+Enrollment recovers section details per student.

#Part 5

Task 5.1

1)



2)

Entities

- Student(StudentID, Name, Email, Major, Year)
- Club(ClubID, ClubName, Description, CreatedOn, Budget, AdvisorFacultyID)

    - AdvisorFacultyID as FK → Faculty (only current advisor, no history).
- Faculty(FacultyID, Name, Dept, Email)
- Membership(StudentID, ClubID, JoinedOn, Status, IsOfficer, Title)
    - Officer fields live here. If IsOfficer = false, Title is NULL.
- Event(EventID, ClubID, Title, StartDateTime, EndDateTime, Purpose, RoomID)
    - Direct FK to Room; one room per event.
- Room(RoomID, Building, RoomNo, Capacity)
- Attendance(EventID, StudentID, CheckedInAt, Status)
- Expense(ExpenseID, ClubID, Date, Amount, Category, Notes)

3)

Officer modeling:

- Option A (chosen above): Separate OfficerRole table — better for history, multiple terms.
- Option B (alternative): Add officer fields to Membership.
    - Pros: Simpler, fewer joins, easier for small projects.
    - Cons: Hard to track officer history; multiple officer terms per student would need duplicates.

Advisor modeling:

- Option A (chosen above): Separate Advisor table — supports multiple/advisors over time.
- Option B (alternative): Store advisor as a field in Club (FacultyID).
    - Pros: Simple, easy queries.
    - Cons: Only one advisor per club; can't keep history.

Room reservation:

- Option A (chosen above): Separate Reservation table (supports multiple rooms per event, flexible time blocks).
- Option B (alternative): Make RoomID a direct attribute of Event.
    - Pros: Simple if events only ever need one room.
    - Cons: Can't handle multi-room events; no detailed time control.

4)

- "List all current officers (any title) in the Computer Science Club."

- “Show all events scheduled next week with their room reservations and capacities.”
- “Report total expenses by category for the Robotics Club this semester.”