LabWork 1

Given the following relations, identify all superkeys and candidate keys:

Relation A: Employee

```
Employee(EmpID, SSN, Email, Phone, Name, Department, Salary)
```

Sample Data:

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	HR	68000
103	456-78-9123	bob@company.com	555-0103	Bob	IT	72000

Part 1: Key Identification Exercises

Task 1.1: Superkey and Candidate Key Analysis

1. EmpID
2. SSN
3. Email
4. EmpID, SSN
5. EmpID, Email
6. SSN, Email

2) Candidate Keys

1. Minimal superkeys are:
2. EmpID
3. SSN
4. Email

3) Choice of Primary Key

I would choose **EmpID** as the primary key because:

1. It is a system-generated identifier, stable and efficient.
2. SSN contains sensitive personal data and may not be suitable as a primary key.
3. Email can change (for example, if an employee updates their email address).
4. EmpID is short, numeric, and better for indexing and foreign key references.

4) Can two employees have the same phone number?

In the given sample data, all phone numbers are unique. However, in practice, two employees may share the same phone number (for example, a shared office phone or home number). Therefore, Phone should not be considered a candidate key unless the business rules explicitly enforce uniqueness.

Relation B: Course Registration

Relation: Registration (StudentID, CourseCode, Section, Semester, Year, Grade, Credits)

Business Rules:

A student can take the same course in different semesters.

A student cannot register for the same course section in the same semester.

Each course section in a semester has a fixed credit value.

1) Minimum attributes for the primary key

The composite primary key should be:
(StudentID, CourseCode, Section, Semester, Year)

2) Why each attribute is necessary

StudentID: identifies which student registered.

CourseCode: identifies which course was taken.

Section: distinguishes between different sections of the same course.

Semester + Year: distinguish when the course was taken.

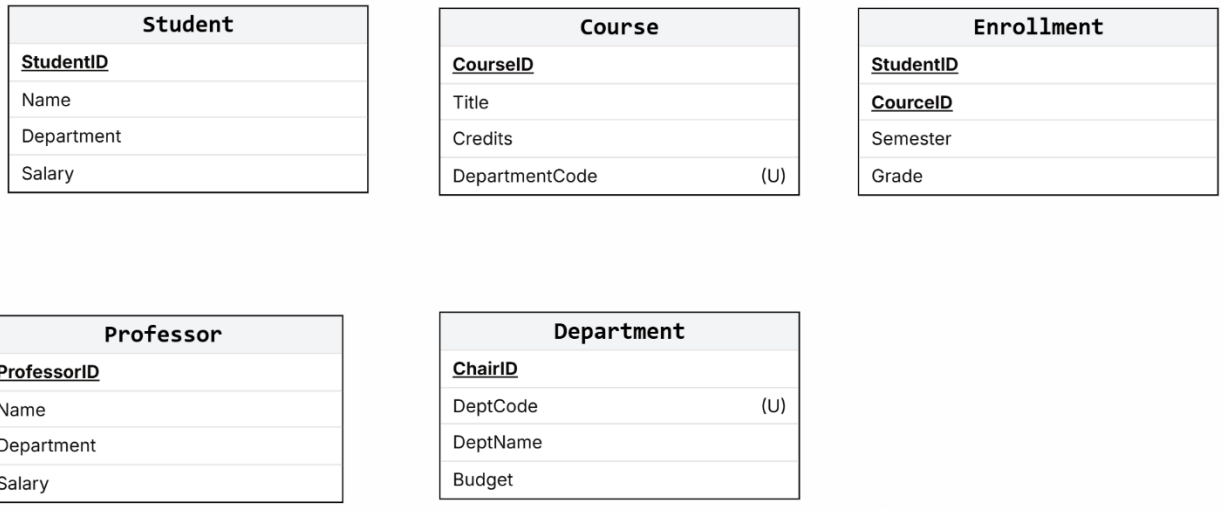
If we remove Section, a student could potentially be registered in two different sections of the same course in the same semester, which violates the business rule.

3) Additional candidate keys

A surrogate key such as RegistrationID (auto-increment integer) could serve as another candidate key.

Without such surrogate keys, there are no other natural candidate keys.

Task 1.2: Foreign Key Design



Foreign Key Relationships

Student.AdvisorID → Professor.ProfID
(Each student has an advisor who is a professor.)

Professor.Department → Department.DeptCode
(Each professor belongs to a department.)

Course.DepartmentCode → Department.DeptCode
(Each course belongs to a department.)

Department.ChairID → Professor.ProfID
(Each department has a chair who is a professor.)

Enrollment.StudentID → Student.StudentID
(Each enrollment references an existing student.)

Enrollment.CourseID → Course.CourseID
(Each enrollment references an existing course.)

Task 2.1 — Hospital Management System

1. Entities (с указанием strong / weak)

Patient (strong) — PatientID, Name, Birthdate, Address, Phone, Insurance.

Doctor (strong) — DoctorID, Name, Specialization, Phone, Office.

Department (strong) — DeptCode, DeptName, Location.

Room (weak, зависит от Department) — RoomNumber (PK = DeptCode + RoomNumber).

Appointment (associative entity between Patient and Doctor) — DateTime, Purpose, Notes.

Prescription (associative entity between Doctor and Patient) — Medication, Dosage, Instructions.

---

2. Attributes

Patient: PatientID (PK, simple), Name (simple),Birthdate (simple), Address (composite: Street, City, State, Zip),Phone (multi-valued), Insurance (simple), Age (derived, from Birthdate)

Doctor: DoctorID (PK, simple), Name (simple), Specialization (multi-valued), Phone (multi-valued), Office (simple)

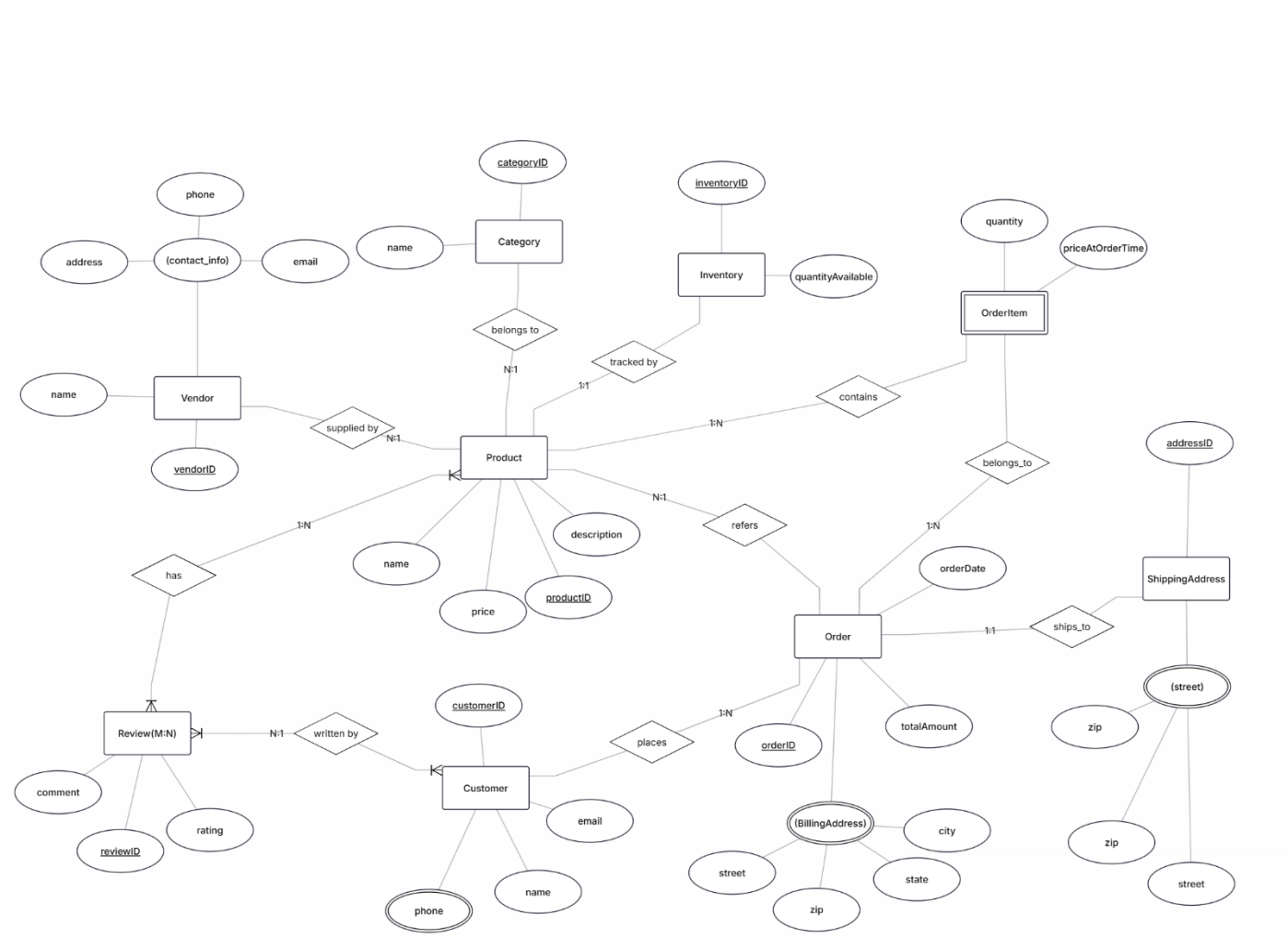
Department: DeptCode (PK, simple), DeptName (simple), Location (simple)

Room (weak): RoomNumber (partial key)

Appointment: DateTime (simple), Purpose (simple), Notes (simple)

Prescription: Medication (simple), Dosage (simple), Instructions (simple)

4. ER Diagram



5. Primary Keys

PatientID, DoctorID, DeptCode, (DeptCode + RoomNumber), AppointmentID (or composite), PrescriptionID (or composite).

Task 2.2 — E-commerce Platform

1. Entities

Customer (CustomerID, Name, BillingAddress, ShippingAddress, Phone, Email).

Order (OrderID, Date, Status, CustomerID).

OrderItem (weak, depends on Order + Product) — Quantity, PriceAtOrder.

Product (ProductID, Name, Price, InventoryLevel).

Category (CategoryID, CategoryName).

Vendor (VendorID, VendorName, ContactInfo).

Review (ReviewID, Rating, Comment, CustomerID, ProductID).

2. Weak Entity

OrderItem is weak because it has no independent existence — identified by OrderID + ProductID.

3. Many-to-Many with attributes

Customer ↔ Product through Review (M:N with attributes Rating, Comment).

- Product ↔ Order through **OrderItem** (M:N with attributes Quantity, PriceAtOrder).

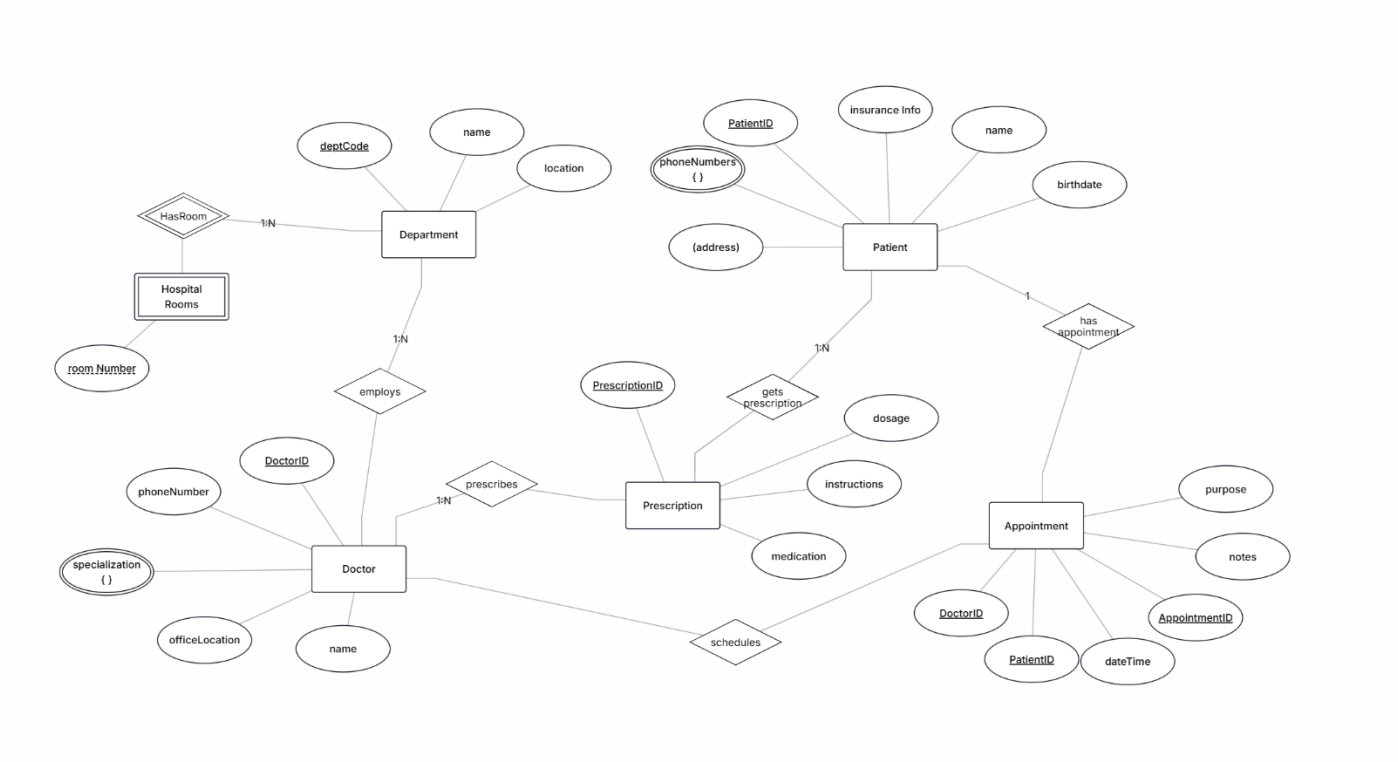
3. Relationships and cardinalities

Patient — Appointment — Doctor: M:N (one patient can see many doctors, one doctor sees many patients).

Doctor — Prescription — Patient: 1:N (a doctor prescribes many prescriptions, each prescription belongs to one patient).

Department — Room: 1:N (each department has many rooms, each room belongs to one department).

Doctor — Department: N:1 (many doctors belong to one department, one department has many doctors)



Part 4: Normalization Workshop

Task 4.1

Given relation
StudentProject(StudentID, StudentName, StudentMajor, ProjectID, ProjectTitle, ProjectType, SupervisorID, SupervisorName, SupervisorDept, Role, HoursWorked, StartDate, EndDate)

1. Functional dependencies (FDs)

StudentID → StudentName, StudentMajor

ProjectID → ProjectTitle, ProjectType, SupervisorID

SupervisorID → SupervisorName, SupervisorDept

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

2. Problems — redundancy and anomalies

Redundancy: Student data repeats in every row for that student; project and supervisor data repeat for every student on the same project.

Update anomaly example: If a supervisor's department changes, you must update SupervisorDept in all rows for that supervisor; if you miss rows, data is inconsistent.

Insert anomaly example: You cannot add a new project record without assigning at least one student (the table mixes student participation and project metadata).

Delete anomaly example: If the last student working on a project is removed, deleting that row may remove all information about the project and its supervisor.

3. 1NF check

All attributes are atomic (no lists or composite fields shown). Therefore the table is in 1NF.

If multi-valued attributes existed (e.g., phone lists), fix by creating separate tables for those attributes.

4. 2NF

Primary key (natural): (StudentID, ProjectID) because each row is one student's participation in one project.

Partial dependencies: StudentName and StudentMajor depend only on StudentID; ProjectTitle, ProjectType, SupervisorID depend only on ProjectID. These violate 2NF.

2NF decomposition (conceptual): split into Student, Project, and StudentProject (the last keeps Role, HoursWorked, StartDate, EndDate with PK (StudentID, ProjectID)).

5. 3NF

Transitive dependency: ProjectID → SupervisorID and SupervisorID → SupervisorName, SupervisorDept implies ProjectID → SupervisorName, SupervisorDept transitively.

Final 3NF design (conceptual): separate Supervisor into its own relation, keep Project referencing Supervisor, keep Student, and keep StudentProject as the associative relation. This removes redundancy, preserves dependencies, and is lossless.

Task 4.2 — CourseSchedule

Given relation
CourseSchedule(StudentID, StudentMajor, CourseID, CourseName, InstructorID, InstructorName, TimeSlot, Room, Building)

1. Primary key

A row records a student's enrollment in a specific course section. A course section is defined by CourseID, TimeSlot, and Room. So the natural primary key for the enrollment row is the combination: StudentID + CourseID + TimeSlot + Room. (Alternatively, you could create a separate SectionID, but with given attributes this composite key is natural.)

2. Functional dependencies (FDs)

StudentID → StudentMajor

CourseID → CourseName

InstructorID → InstructorName

Room → Building

(CourseID, TimeSlot, Room) → InstructorID (each section has one instructor)

The full key (StudentID, CourseID, TimeSlot, Room) determines the rest of the row.

3. BCNF check

BCNF requires every determinant to be a superkey. Here StudentID, CourseID, InstructorID, Room, and (CourseID, TimeSlot, Room) are not superkeys of the full relation, so several FDs violate BCNF. Therefore the table is not in BCNF.

4. Decomposition to BCNF (conceptual steps)

Separate Student information using StudentID → StudentMajor (create Student relation).

Separate Course information using CourseID → CourseName (create Course relation).

Separate Instructor information using InstructorID → InstructorName (create Instructor relation).

Separate Room information using Room → Building (create Room relation).

Create Section relation to capture a course section: CourseID + TimeSlot + Room → InstructorID.

Create Enrollment relation that links StudentID to a specific Section (StudentID + CourseID + TimeSlot + Room).
This decomposition makes each FD apply within a relation where the left-hand side is a key, so it meets BCNF and is lossless.

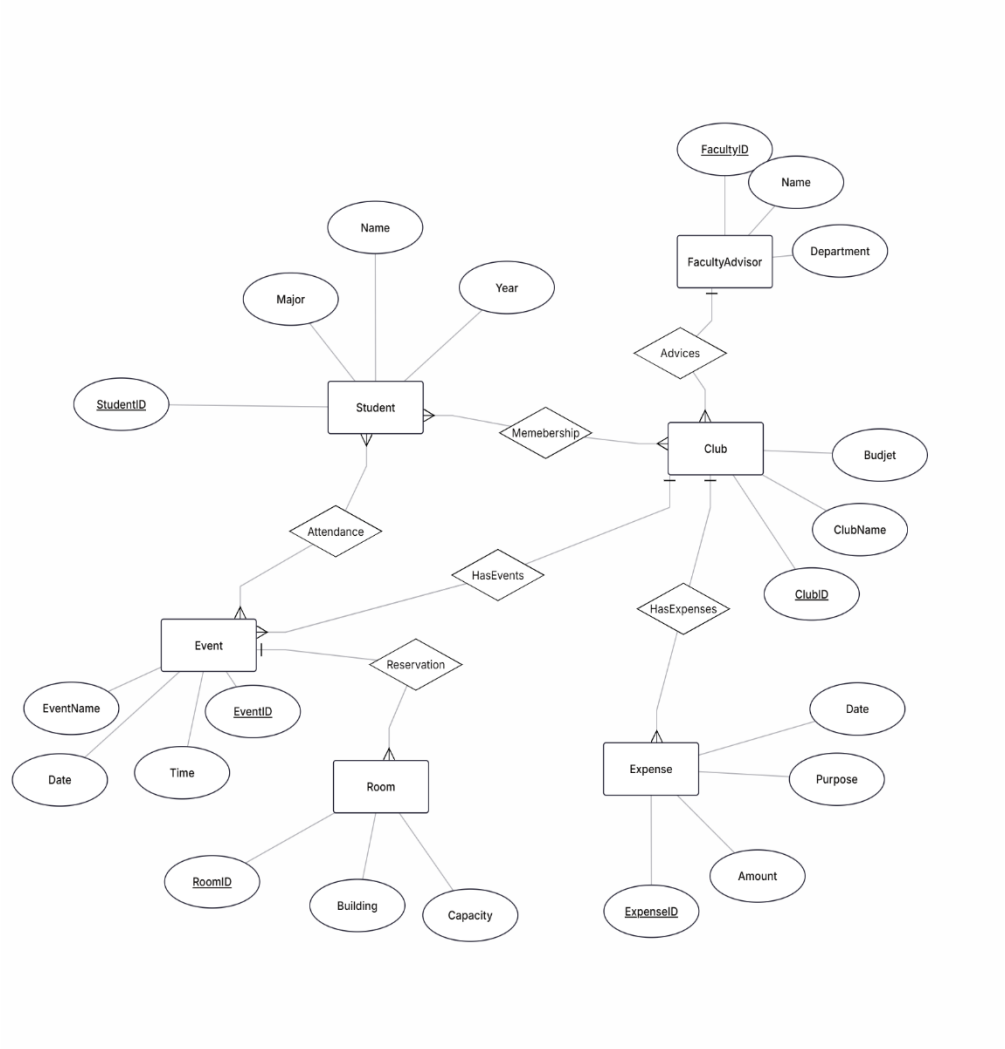
5. Potential information loss

The decomposition is lossless: joining Enrollment with Section, Course, Room, Instructor and Student reconstructs the original flattened records.

The main dependencies are preserved because each FD is represented in an appropriate relation (Student, Course, Instructor, Room, Section). Using a surrogate SectionID would simplify keys, but is optional; the conceptual decomposition above preserves data and constraints.

Task 5.1 — Real-World Application (Student Clubs System)

1. ER Diagram



Entities:

Student(StudentID, Name, Major, Year)

Club(ClubID, ClubName, Budget)

FacultyAdvisor(FacultyID, Name, Department)

Room(RoomID, Building, Capacity)

Event(EventID, EventName, Date, Time)

Expense(ExpenseID, Amount, Date, Purpose)

Relationships:

Membership (M:N) → between Student and Club.

Officer (M:N with attribute Position) → between Student and Club.

Attendance (M:N) → between Student and Event.

Advises (1:N) → FacultyAdvisor to Club (one club has one advisor, advisor can advise many).

Reservation (1:N) → Event to Room.

HasEvents (1:N) → Club to Event.

HasExpenses (1:N) → Club to Expense.

2. Normalized Relational Schema (3NF)

Student(StudentID PK, Name, Major, Year)

Club(ClubID PK, ClubName, Budget, FacultyID FK)

FacultyAdvisor(FacultyID PK, Name, Department)

Membership(StudentID FK, ClubID FK, PK = StudentID+ClubID)

Officer(StudentID FK, ClubID FK, Position, PK = StudentID+ClubID+Position)

Event(EventID PK, ClubID FK, RoomID FK, EventName, Date, Time)

Attendance(StudentID FK, EventID FK, PK = StudentID+EventID)

Room(RoomID PK, Building, Capacity)

Expense(ExpenseID PK, ClubID FK, Amount, Date, Purpose)

3. Design Decision (multiple valid options)

Officer roles:

Option A: Store officer positions as a simple attribute inside Membership (like Role).

Option B: Create a separate Officer relation (StudentID, ClubID, Position).

I chose Option B because a student can be a regular member and also hold one or more officer roles. Keeping a separate Officer relation makes it easier to track multiple roles per student in the same club and avoids storing NULL values for students who are not officers.

4. Example Queries (in English, not SQL)

Find all students who are officers in the Computer Science Club.

List all events scheduled for next week with their room reservations.

Show the total expenses recorded for each club in the current semester.