Part 1 – Task 1.1: Superkey and Candidate Key Analysis

Relation A: Employee

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

1. Examples of Superkeys:

- {EmpID}
- {SSN}
- {Email}
- {EmpID, Phone}
- {SSN, Email}
- {Email, Department, Name}

2. Candidate Keys:

- {EmpID}
- {SSN}
- {Email}

3. Choice of Primary Key:
The chosen Primary Key is **EmpID**, because it is:

- stable and unique,
- independent from personal attributes (SSN or Email),
- more reliable for referencing in other tables.

4. Phone Attribute:
Although the sample data shows unique phone numbers, in real systems multiple employees may share the same phone (e.g., a shared office or family number). Therefore, **Phone is not a reliable key**.

Relation B: Registration

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

1. Minimal Primary Key:
`(StudentID, CourseCode, Section, Semester, Year)`

2. Justification:

- `StudentID` identifies the student,
- `CourseCode + Section` identify the specific course section,
- `Semester + Year` distinguish multiple attempts across semesters.

3. Additional Candidate Keys:

- `(CourseCode, Section, Semester, Year)` is unique for a course offering but **not sufficient** for registration, since it does not include the student.

Part 1 – Task 1.2: Foreign Key Design

Given Tables

- **Student(StudentID, Name, Email, Major, AdvisorID)**
- **Professor(ProfID, Name, Department, Salary)**
- **Course(CourseID, Title, Credits, DepartmentCode)**
- **Department(DeptCode, DeptName, Budget, ChairID)**
- **Enrollment(StudentID, CourseID, Semester, Grade)**

Foreign Key Relationships

1. **Student → Professor**
   - `Student.AdvisorID` → `Professor.ProfID`
   - Each student has an advisor who must be a professor.
2. **Course → Department**
   - `Course.DepartmentCode` → `Department.DeptCode`
   - Each course belongs to one department.
3. **Department → Professor**
   - `Department.ChairID` → `Professor.ProfID`
   - Each department has one chair, who is a professor.
4. **Enrollment → Student**
   - `Enrollment.StudentID` → `Student.StudentID`
   - Each enrollment record must be linked to an existing student.
5. **Enrollment → Course**
   - `Enrollment.CourseID` → `Course.CourseID`
   - Each enrollment record must reference a valid course.

Primary Keys and Composite Keys

- **Student**: Primary Key = `StudentID`
- **Professor**: Primary Key = `ProfID`
- **Course**: Primary Key = `CourseID`
- **Department**: Primary Key = `DeptCode`

- **Enrollment**: Composite Primary Key = `(StudentID, CourseID, Semester)`

Part 2 – Task 2.1: Hospital Management System

1. Entities

- **Patient** (strong entity)
- **Doctor** (strong entity)
- **Department** (strong entity)
- **Appointment** (weak entity, depends on Patient + Doctor)
- **Prescription** (weak entity, depends on Patient + Doctor)
- **Room** (weak entity, depends on Department)

---

2. Attributes

Patient

- PatientID (PK)
- Name
- BirthDate
- Address (composite: Street, City, State, ZIP)
- Phone (multi-valued)
- InsuranceInfo

Doctor

- DoctorID (PK)
- Name
- Specialization (multi-valued)
- Phone (multi-valued)
- OfficeLocation

Department

- DeptCode (PK)
- DeptName
- Location

Appointment

- AppointmentID (PK)
- PatientID (FK → Patient)
- DoctorID (FK → Doctor)
- DateTime

- Purpose
- Notes

Prescription

- PrescriptionID (PK)
- PatientID (FK → Patient)
- DoctorID (FK → Doctor)
- Medication
- Dosage
- Instructions

Room

- DeptCode (FK → Department)
- RoomNumber
- **Primary Key:** (DeptCode, RoomNumber)

---

3. Relationships and Cardinalities

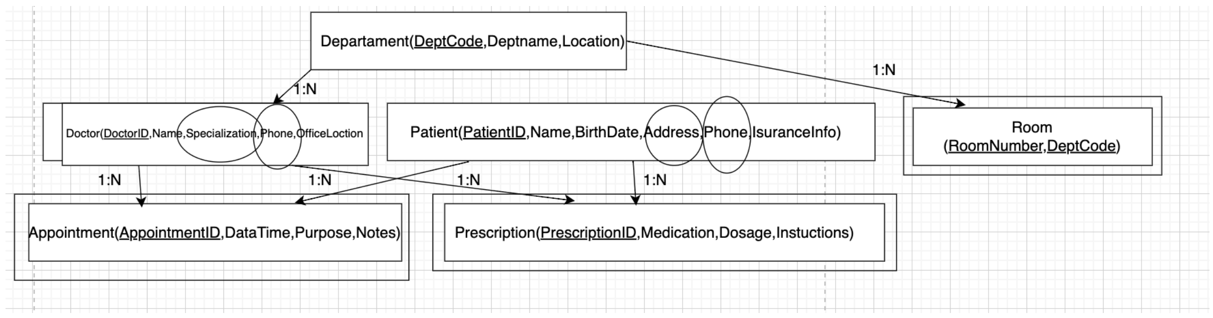
- **Patient — Appointment — Doctor**:
  - Patient (1:N) Appointment (N:1) Doctor
- **Patient — Prescription — Doctor**:
  - Patient (1:N) Prescription (N:1) Doctor
- **Department — Doctor**:
  - Department (1:N) Doctor
- **Department — Room**:
  - Department (1:N) Room

---

4. ER Diagram (to be drawn separately)

The ERD should include:

- Entities with primary keys underlined
- Weak entities with composite keys (including owner’s key)
- Multi-valued attributes represented by double ovals
- Correct relationship cardinalities (1:1, 1:N, M:N)


Departament(DeptCode,Deptname,Location)
1:N
1:N
Doctor(DoctorID,Name,Specialization,Phone,OfficeLoction
Patient(PatientID,Name,BirthDate,Address,Phone,IsuranceInfo)
Room
(RoomNumber,DeptCode)
1:N
1:N
1:N
1:N
Appointment(AppointmentID,DataTime,Purpose,Notes)
Prescription(PrescriptionID,Medication,Dosage,Instuctions)

Part 2.2: E-commerce Platform

Entities:

- **Customer** (CustomerID, Name, Email, BillingAddress)
- **Order** (OrderID, OrderDate, TotalAmount, CustomerID)
- **OrderItem** *(weak entity)* (OrderID, ProductID, Quantity, PriceAtOrder)
- **Product** (ProductID, Name, Description, Price, CategoryID, VendorID)
- **Category** (CategoryID, CategoryName)
- **Vendor** (VendorID, VendorName, ContactInfo)
- **Review** *(weak entity)* (ReviewID, Rating, Comment, CustomerID, ProductID)
- **Inventory** (ProductID, StockLevel)
- **ShippingAddress** *(weak entity)* (AddressID, Street, City, State, Zip, CustomerID)

Relationships:

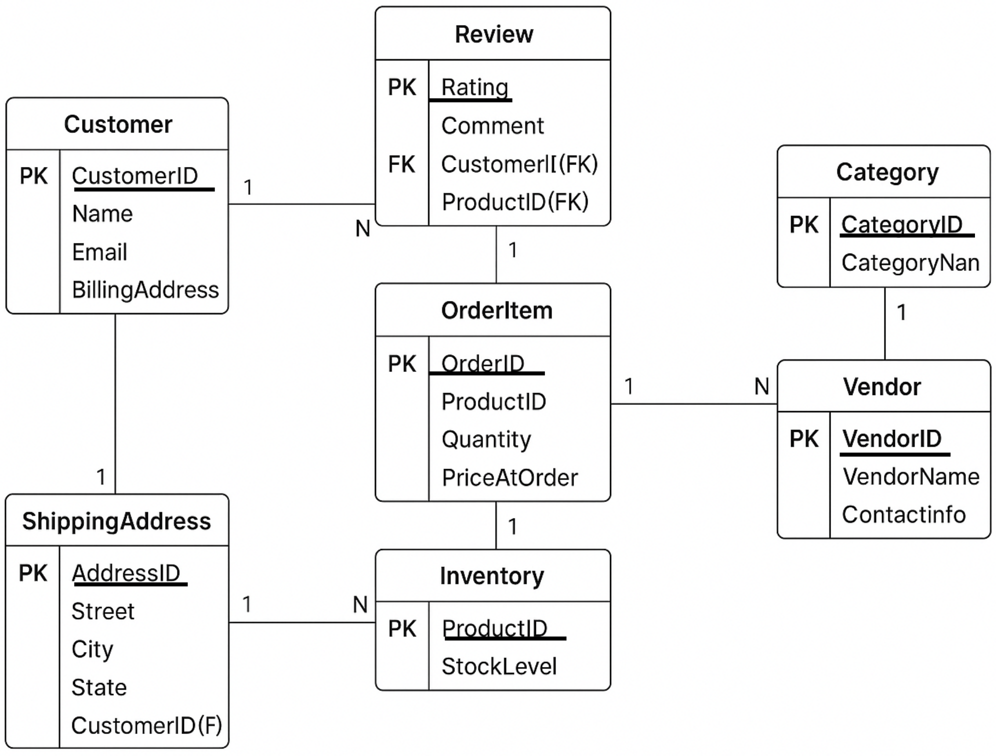
- **Customer – Order:** One customer can place many orders (1:N).
- **Order – Product:** Many-to-many relationship implemented through the weak entity **OrderItem**, which also stores attributes (Quantity, PriceAtOrder).
- **Product – Category:** Each product belongs to one category, but a category can include many products (1:N).
- **Product – Vendor:** One vendor supplies many products (1:N).
- **Product – Review – Customer:** Many-to-many relationship, since customers can review many products and products can have many reviews.
- **Product – Inventory:** One-to-one relationship, since each product has exactly one stock record.
- **Customer – ShippingAddress:** One-to-many, since a customer can have multiple shipping addresses.

Weak Entities:

- **OrderItem** is weak because it cannot exist without both an Order and a Product.
- **Review** is weak because it depends on both Customer and Product.
- **ShippingAddress** is weak because it depends on the Customer.

Many-to-Many with attributes:

- The **Order–Product** relationship is many-to-many and requires additional attributes such as *Quantity* and *PriceAtOrder*. These are stored in the **OrderItem** entity.



Part 4.1: Denormalized Table Analysis

Given table:

```
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 → SupervisorName, SupervisorDept
- {StudentID, ProjectID} → Role, HoursWorked, StartDate, EndDate
- ProjectID → SupervisorID (each project is supervised by exactly one supervisor)

---

2. Problems in the table:

- **Redundancy:** StudentName and StudentMajor repeat for every project of the same student. SupervisorName and SupervisorDept repeat for every project under the same supervisor.
- **Update anomaly:** If the supervisor changes department, it must be updated in multiple rows.
- **Insert anomaly:** A new student cannot be added without assigning a project.
- **Delete anomaly:** If a project is deleted, information about its supervisor may be lost.

---

3. 1NF:

- The table is already in **1NF** because all attributes are atomic.

---

4. 2NF:

- Candidate key: **{StudentID, ProjectID}**
- Partial dependencies:
  - StudentID → StudentName, StudentMajor
  - ProjectID → ProjectTitle, ProjectType, SupervisorID
- Decomposition into 2NF:
  - **Student(StudentID, StudentName, StudentMajor)**
  - **Supervisor(SupervisorID, SupervisorName, SupervisorDept)**
  - **Project(ProjectID, ProjectTitle, ProjectType, SupervisorID)**
  - **StudentProject(StudentID, ProjectID, Role, HoursWorked, StartDate, EndDate)**

---

5. 3NF:

- Transitive dependency: ProjectID → SupervisorID → SupervisorName, SupervisorDept
- To remove transitivity, Supervisor is separated.
- Final 3NF schema:
  - **Student(StudentID, StudentName, StudentMajor)**
  - **Supervisor(SupervisorID, SupervisorName, SupervisorDept)**

- **Project(ProjectID, ProjectTitle, ProjectType, SupervisorID)**
- **StudentProject(StudentID, ProjectID, Role, HoursWorked, StartDate, EndDate)**

Part 4.2: Advanced Normalization

Given table:

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

Business Rules:

- Each student has exactly one major
- Each course has a fixed name
- Each instructor has exactly one name
- Each time slot in a room determines the building
- Each course section is taught by one instructor at one time in one room
- A student can be enrolled in multiple course sections

---

1. Primary Key

Candidate key: **{StudentID, CourseID, TimeSlot}**

- StudentID → identifies student
- CourseID + TimeSlot → uniquely identifies course section
- Together: {StudentID, CourseID, TimeSlot} uniquely identifies enrollment.

---

2. Functional Dependencies (FDs)

- StudentID → StudentMajor
- CourseID → CourseName
- InstructorID → InstructorName
- (Room, TimeSlot) → Building
- (CourseID, TimeSlot, Room) → InstructorID
- {StudentID, CourseID, TimeSlot} → (all attributes)

---

3. BCNF Check

- Not in BCNF because some non-key attributes determine others (e.g., CourseID → CourseName, InstructorID → InstructorName).

---

4. Decomposition into BCNF

Decompose into separate relations:

1. **Student(StudentID, StudentMajor)**
2. **Course(CourseID, CourseName)**
3. **Instructor(InstructorID, InstructorName)**
4. **Room(Room, TimeSlot, Building)**
5. **Section(CourseID, TimeSlot, Room, InstructorID)**
6. **Enrollment(StudentID, CourseID, TimeSlot)**

---

5. Information Loss

- No information loss because all original dependencies are preserved across decomposed tables.
- Some redundancy is removed (e.g., course name not repeated for each enrollment).

Part 5.1: Student Clubs and Organizations System

1. Entities

- **Student** (StudentID, Name, Major, Email)
- **Club** (ClubID, ClubName, Description, Budget)
- **Membership** *(weak entity)* (StudentID, ClubID, JoinDate, Role)
- **Event** (EventID, ClubID, Title, DateTime, Location, Description)
- **Attendance** *(weak entity)* (StudentID, EventID, Status)
- **OfficerPosition** (PositionID, PositionName)
- **ClubOfficer** *(weak entity)* (ClubID, StudentID, PositionID, StartDate, EndDate)
- **FacultyAdvisor** (AdvisorID, Name, Department, Email)
- **RoomReservation** (ReservationID, EventID, Room, Building, ReservedBy)
- **Expense** (ExpenseID, ClubID, Amount, Purpose, Date)

---

2. Relationships

- **Student – Membership – Club:** Many-to-many via Membership.
- **Club – Event:** One-to-many (a club can organize many events).
- **Student – Attendance – Event:** Many-to-many via Attendance.
- **Club – ClubOfficer – OfficerPosition – Student:** Weak entity connecting clubs, positions, and students.
- **Club – FacultyAdvisor:** One-to-one (each club has one advisor, but a faculty can advise many clubs → 1:N).

- **Event – RoomReservation:** One-to-one (each event has exactly one room reservation).
- **Club – Expense:** One-to-many (clubs track multiple expenses).

---

3. Normalized Relational Schema

- **Student(StudentID, Name, Major, Email)**
- **Club(ClubID, ClubName, Description, Budget, AdvisorID)**
- **Membership(StudentID, ClubID, JoinDate, Role, PK = {StudentID, ClubID})**
- **Event(EventID, ClubID, Title, DateTime, Location, Description)**
- **Attendance(StudentID, EventID, Status, PK = {StudentID, EventID})**
- **OfficerPosition(PositionID, PositionName)**
- **ClubOfficer(ClubID, StudentID, PositionID, StartDate, EndDate, PK = {ClubID, StudentID, PositionID})**
- **FacultyAdvisor(AdvisorID, Name, Department, Email)**
- **RoomReservation(ReservationID, EventID, Room, Building, ReservedBy)**
- **Expense(ExpenseID, ClubID, Amount, Purpose, Date)**

---

4. Design Decision

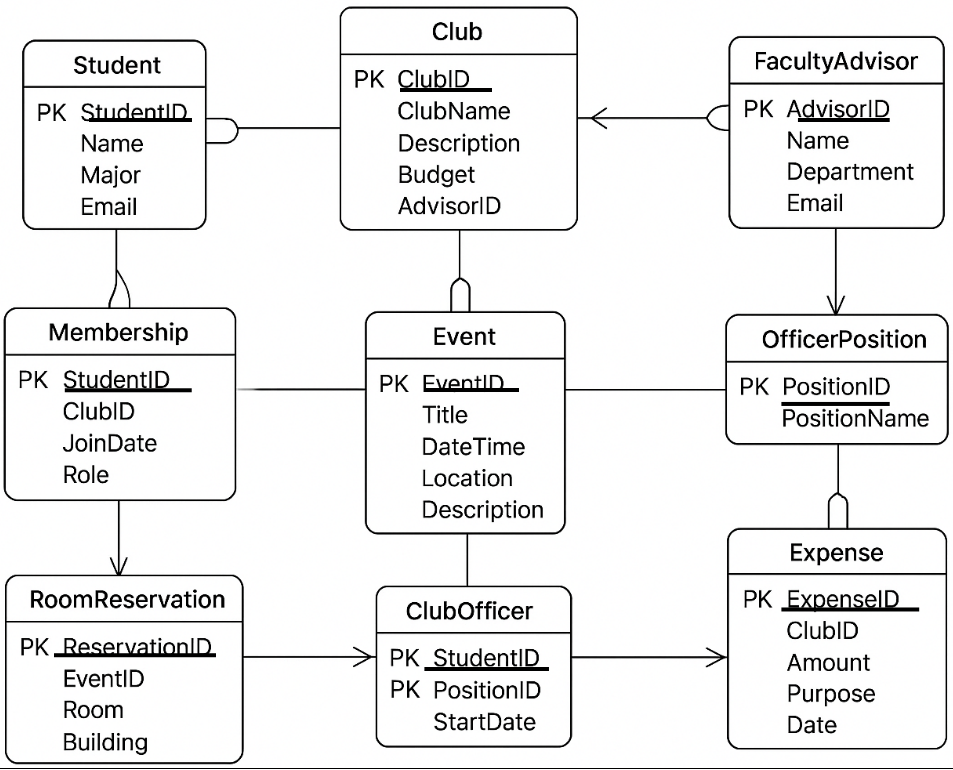
One design choice was how to model **officer positions**.

- Option 1: Store officer roles directly as attributes in the Membership entity.
- Option 2 (chosen): Create separate entities **OfficerPosition** and **ClubOfficer**. This choice improves flexibility, since a student can hold multiple officer roles in different clubs and positions can be reused across clubs.

---

5. Example Queries (in English)

1. “Find all students who are officers in the Computer Science Club.”
2. “List all events scheduled for next week with their room reservations.”
3. “Show the total expenses for each club in the current semester.


Student
PK StudentID
Name
Major
Email
Club
PK ClubID
ClubName
Description
Budget
AdvisorID
FacultyAdvisor
PK AdvisorID
Name
Department
Email
Membership
PK StudentID
ClubID
JoinDate
Role
Event
PK EventID
Title
DateTime
Location
Description
OfficerPosition
PK PositionID
PositionName
RoomReservation
PK ReservationID
EventID
Room
Building
ClubOfficer
PK StudentID
PK PositionID
StartDate
Expense
PK ExpenseID
ClubID
Amount
Purpose
Date