Task 1

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

Your Tasks: 1. List at least 6 different superkeys 2. Identify all candidate keys 3. Which candidate key would you choose as primary key and why? 4. Can two employees have the same phone number? Justify your answer based on the data shown.

1) 6 superkeys:

`{EmpID}{SSN}{Email}{EmpID, SSN}{EmpID, Email}{SSN, Email}{EmpID, Phone}{EmpID, Department}{Email, Phone}{SSN, Phone}`

2) Candidate keys

`{EmpID}` (internal identifier)
`{SSN}` (government identifier; typically unique)
`{Email}` (usually unique per employee)

3. Which candidate key would you choose as primary key and why?

Choose `{EmpID}` as the primary key:

- Stable and controlled by the organization (unlike SSN or Email which can change/present privacy issues).
- Compact, numeric surrogate → efficient indexing and joining.
- Avoids exposing PII (SSN) or volatile attributes (Email) in FKs.

4) Can two employees have the same phone number?

- **From the sample data alone:** all phones are different, but a sample does not imply a rule.
- **Business-wise:** desk phones can be shared; people can change phones; organizations often do not enforce phone uniqueness.

Conclusion: you cannot assume `Phone` is unique → it's not a candidate key unless there's an explicit uniqueness constraint.

Relation B: Course Registration

`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

Your Tasks: 1. Determine the minimum attributes needed for the primary key 2. Explain why each attribute in your primary key is necessary 3. Identify any additional candidate keys (if they exist)

1) Minimum attributes for the primary key
`(StudentID, CourseCode, Section, Semester, Year)`

2) Why each attribute is necessary

- **StudentID** — identifies *which* student the registration belongs to.
- **CourseCode** — which course.
- **Section** — distinguishes multiple parallel offerings in the same term.
- **Semester** — Fall/Spring/Summer etc.
- **Year** — distinguishes the semester across years (Fall 2024 ≠ Fall 2025).

Because a student cannot register for the *same section in the same semester*, we need StudentID + (CourseCode, Section, Semester, Year) to prevent duplicates and allow retakes in different terms.

3) Additional candidate keys?

- For the **offering** itself, `(CourseCode, Section, Semester, Year)` is a key of the *course-section-in-a-term* entity (it uniquely identifies the offering and thus its fixed `Credits`).
- But in **Registration**, many students register to the same offering, so `(CourseCode, Section, Semester, Year)` **is not** a candidate key for the registration rows.
- Unless you add a surrogate like `RegistrationID`, there are **no other natural candidate keys** for `Registration`.

Task 1.2: Foreign Key Design

Design the foreign key relationships for this university system:

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)
```

Your Tasks: 1. Identify all foreign key relationships

1) All foreign key relationships

Student

Student.Major → **Department(DeptCode)**

Student.AdvisorID → **Professor(ProfID)** (nullable if not assigned

Professor

Professor.Department → **Department(DeptCode)**

Course

Course.DepartmentCode → **Department(DeptCode)**

Department

Department.ChairID → **Professor(ProfID)** (nullable until a chair is appointed)

Enrollment

Enrollment.StudentID → **Student(StudentID)**

Enrollment.CourseID → **Course(CourseID)**

Part 2: ER Diagram Construction

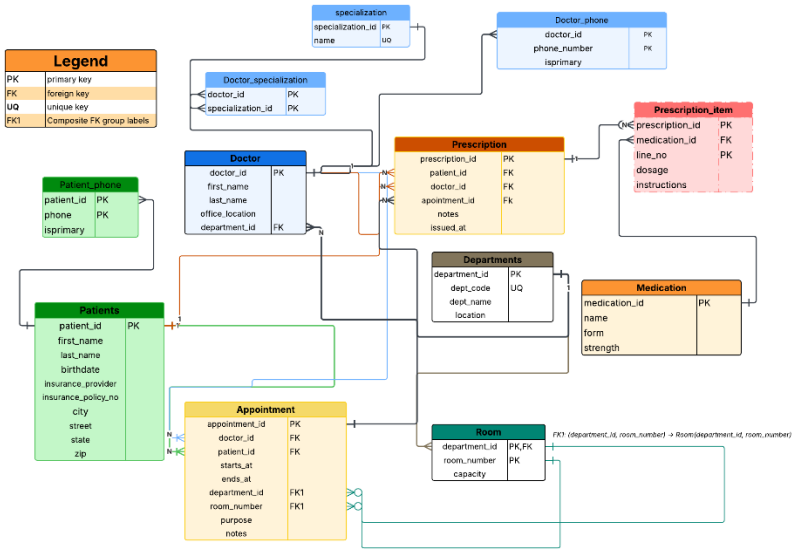
Task 2.1: Hospital Management System

Scenario: Design a database for a hospital management system.

Requirements:

- **Patients** have unique patient IDs, names, birthdates, addresses (street, city, state, zip), phone numbers (multiple allowed), and insurance information
- **Doctors** have unique doctor IDs, names, specializations (can have multiple), phone numbers, and office locations
- **Departments** have department codes, names, and locations
- **Appointments** track which patient sees which doctor at what date/time, the purpose of visit, and any notes
- **Prescriptions** track medications prescribed by doctors to patients, including dosage and instructions
- **Hospital Rooms** are numbered within departments (room 101 in Cardiology is different from room 101 in Neurology)

Your Tasks: 1. Identify all entities (specify which are strong and which are weak) 2. Identify all attributes for each entity (classify as simple, composite, multi-valued, or derived) 3. Identify all relationships with their cardinalities (1:1, 1:N, M:N) 4. Draw the complete ER diagram using proper notation 5. Mark primary keys



1) Entities (Strong vs Weak)

Strong entities

- **Patient**(PatientID, Name, BirthDate, Address, City, State, Zip, PrimaryEmail, Notes)
- **Doctor**(DoctorID, Name, DepartmentCode, OfficeLocation)
- **Department**(DeptCode, DeptName, Location)
- **Medication**(MedicationID, MedName, Form, Strength)
- **InsurancePolicy**(PolicyID, ProviderName, PolicyNumber, PlanType, StartDate, EndDate)
- **Appointment**(AppointmentID, PatientID, DoctorID, ApptDateTime, Purpose, Notes, RoomDeptCode, RoomNumber)
- **Room**(**DeptCode**, **RoomNumber**, BedCount, RoomType)
- **Specialization**(SpecID, SpecName)

Weak / Identifying-dependent entities

- **PatientPhone**(**PatientID**, **Phone**) – multivalued attribute of Patient modeled as an entity (PhoneType optional)
- **DoctorPhone**(**DoctorID**, **Phone**) – multivalued attribute of Doctor modeled as an entity (PhoneType optional)
- **DoctorSpecialization**(**DoctorID**, **SpecID**) – resolves M:N between Doctor and Specialization
- **PatientInsurance**(**PatientID**, **PolicyID**) – resolves possible 1:N (or M:N) between Patient and InsurancePolicy
- **Prescription**(PrescriptionID, PatientID, DoctorID, PrescribedAt, Notes) – header
- **PrescriptionItem**(**PrescriptionID**, **MedicationID**, Dosage, Instructions) – **weak** (identified by parent Prescription + Medication)

Why Room is not weak: it has a composite key (DeptCode, RoomNumber) but not existence-dependent on any single row other than its owning Department; we treat it as a strong entity with a composite PK.

2) Attributes Classification

Patient

- *Key:* PatientID (simple)
- *Simple:* Name, PrimaryEmail, Notes
- *Composite address:* Address (Street is stored in Address), City, State, Zip (Alternatively keep Address as Street only; City/State/Zip are separate simple attributes.)
- *Derived:* Age (from BirthDate) – **not stored**, derivable
- *Multivalued:* Phones → modeled via **PatientPhone**(PatientID, Phone[, PhoneType])

Doctor

- *Key:* DoctorID (simple)
- *Simple:* Name, OfficeLocation
- *FK:* DepartmentCode → Department
- *Multivalued:* Phones → **DoctorPhone**(DoctorID, Phone[, PhoneType])
- *Multivalued via relationship:* Specializations → **DoctorSpecialization**

Department

- *Key:* DeptCode (simple)
- *Simple:* DeptName, Location

Medication

- *Key:* MedicationID (simple)
- *Simple:* MedName, Form (tablet, capsule, syrup), Strength (e.g., 500 mg)

InsurancePolicy

- *Key:* PolicyID (simple)
- *Simple:* ProviderName, PolicyNumber, PlanType, StartDate, EndDate

PatientInsurance (assoc.)

- *Composite key:* (PatientID, PolicyID)
- *Optional attrs:* CoverageNotes, Copay, Status

Appointment

- *Key:* AppointmentID (simple surrogate)
- *FKs:* PatientID → Patient, DoctorID → Doctor
- *Simple:* ApptDateTime, Purpose, Notes
- *Optional FKs:* RoomDeptCode, RoomNumber → Room (if the visit is assigned a room)

Room

- *Composite key:* (DeptCode, RoomNumber)
- *Simple:* BedCount, RoomType (ICU, Ward, Private, etc.)
- *FK:* DeptCode → Department

Specialization

- *Key:* SpecID (simple)
- *Simple:* SpecName

DoctorSpecialization (assoc.)

- *Composite key:* (DoctorID, SpecID)

Prescription (header)

- *Key:* PrescriptionID (simple)
- *FKs:* PatientID → Patient, DoctorID → Doctor
- *Simple:* PrescribedAt (datetime), Notes

PrescriptionItem (line items; **weak**)

- *Composite key:* (PrescriptionID, MedicationID)
- *FKs:* PrescriptionID → Prescription, MedicationID → Medication
- *Simple:* Dosage, Instructions

3) Relationships & Cardinalities

- **Department–Doctor:** 1:N
  One department has many doctors; each doctor belongs to one department.
- **Department–Room:** 1:N
  One department has many rooms; room key includes DeptCode + RoomNumber.
- **Patient–Appointment–Doctor:**
  Patient to Appointment: 1:N
  Doctor to Appointment: 1:N
  (Appointments are associative events linking one Patient and one Doctor at a specific datetime.)
- **Appointment–Room:** Optional 0..1:N from Room to Appointment
  A room may host many appointments over time; an appointment may or may not be tied to a room.
- **Doctor–Specialization:** M:N via **DoctorSpecialization**
  A doctor can have multiple specializations; a specialization applies to many doctors.
- **Patient–InsurancePolicy:** 1:N (or M:N) via **PatientInsurance**
  Typically a patient can hold multiple policies over time; the same policy instance usually belongs to a single patient, but the associative table allows flexibility (e.g., family policies). Keep as (PatientID, PolicyID) with dates if needed.
- **Prescription (header) – Patient & Doctor:** N:1 to each
  Each prescription is issued by exactly one doctor to exactly one patient.
- **Prescription – PrescriptionItem:** 1:N (identifying)
  A prescription contains many medication lines.
- **PrescriptionItem – Medication:** N:1
  Each line references one medication.
- **Patient–PatientPhone:** 1:N (identifying)
  Implemented as a separate table to store multiple numbers.
- **Doctor–DoctorPhone:** 1:N (identifying)

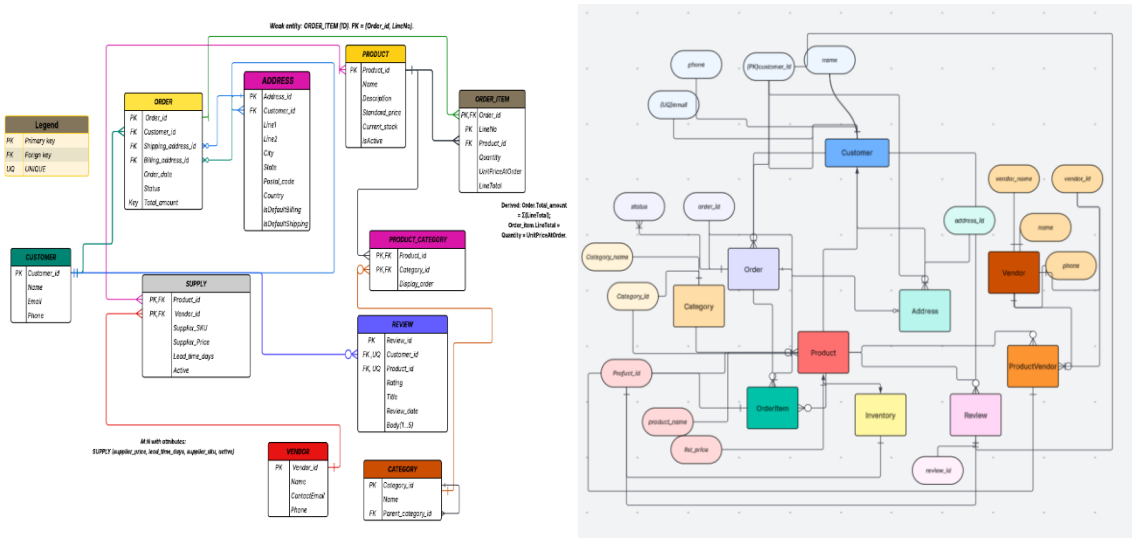
Task 2.2: E-commerce Platform

Scenario: Design a simplified e-commerce database.

Requirements:

- **Customers** place **Orders** for **Products**
- **Products** belong to **Categories** and are supplied by **Vendors**
- **Orders** contain multiple **Order Items** (quantity and price at time of order)
- **Products** have reviews and ratings from customers
- Track **Inventory** levels for each product
- **Shipping addresses** can be different from customer billing addresses

Your Tasks: 1. Create a complete ER diagram 2. Identify at least one weak entity and justify why it's weak 3. Identify at least one many-to-many relationship that needs attributes



1. Complete ER Diagram

The ER diagram includes all the required entities and relationships:

- **Customer(CustomerID, Name, Email, Phone)**
- **Address(AddressID, CustomerID FK, …)** – supports multiple addresses (billing and shipping).
- **Order(OrderID, CustomerID FK, ShippingAddressID FK, BillingAddressID FK, OrderDate, Status, TotalAmount)**
- **OrderItem(OrderID PK/FK, LineNo PK, ProductID FK, Quantity, UnitPriceAtOrder, LineTotal)**
- **Product(ProductID, Name, Description, StandardPrice, CurrentStock, IsActive)**
- **Category(CategoryID, Name, ParentCategoryID FK)**
- **ProductCategory(ProductID FK, CategoryID FK, DisplayOrder)** – resolves many-to-many between products and categories (if a product can belong to multiple categories).
- **Vendor(VendorID, Name, ContactEmail, Phone)**
- **Supply(ProductID FK, VendorID FK, SupplierSKU, SupplierPrice, LeadTimeDays, Active)** – resolves many-to-many between products and vendors.
- **Review(ReviewID, CustomerID FK, ProductID FK, Rating, Title, Body, ReviewDate, UQ(CustomerID, ProductID))** – enforces at most one review per customer per product.

2. Weak Entity

OrderItem is a *weak entity* because:

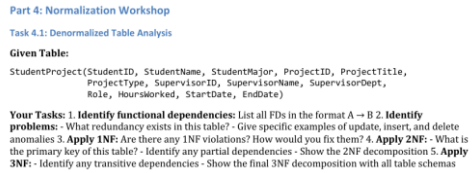
- It cannot exist without its parent **Order**.
- Its primary key is composite and includes the parent's key (OrderID) plus its own discriminator (LineNo).
- Order items only make sense in the context of their parent order.

3. Many-to-Many Relationship with Attributes

Product–Vendor is a *many-to-many* relationship:

- One product can be supplied by many vendors.
- One vendor can supply many products.

- The relationship has attributes (SupplierSKU, SupplierPrice, LeadTimeDays, Active) that describe the supplier-specific conditions.
- Therefore, it is modeled as a separate table: **Supply(ProductID, VendorID, SupplierSKU, SupplierPrice, LeadTimeDays, Active)**.



Part 4: Normalization Workshop

Task 4.1: Denormalized Table Analysis

Given Table:

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

Your Tasks: 1. **Identify functional dependencies:** List all FDs in the format A → B 2. **Identify problems:** - What redundancy exists in this table? - Give specific examples of update, insert, and delete anomalies 3. **Apply 1NF:** Are there any 1NF violations? How would you fix them? 4. **Apply 2NF:** - What is the primary key of this table? - Identify any partial dependencies - Show the 2NF decomposition 5. **Apply 3NF:** - Identify any transitive dependencies - Show the final 3NF decomposition with all table schemas

Task 4.1 — Denormalized Table Analysis

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

1) Functional Dependencies (FDs)
Natural FDs implied by the attributes:

- **Student facts**
  - StudentID → StudentName, StudentMajor
- **Supervisor facts**
  - SupervisorID → SupervisorName, SupervisorDept
- **Project facts** (with one supervisor per project)
  - ProjectID → ProjectTitle, ProjectType, SupervisorID, StartDate, EndDate
- **Participation facts** (student's role/time on a project)
  - (StudentID, ProjectID) → Role, HoursWorked

From these, transitive implications exist, e.g. ProjectID → SupervisorID → SupervisorName, SupervisorDept.
If StartDate/EndDate were per student assignment, then use (StudentID, ProjectID) → StartDate, EndDate instead of the project-level FD above.

2) Problems
Redundancy examples

- Repeating student data: every row with the same StudentID repeats StudentName, StudentMajor.
- Repeating project data: every row with the same ProjectID repeats ProjectTitle, ProjectType, StartDate, EndDate.
- Repeating supervisor data: every project row repeats SupervisorName, SupervisorDept.

Update anomalies

- Change Mary's major → must update it in **every** row where StudentID = S123. Miss one row ⇒ inconsistent data.
- Change a project's title or dates → must update on all rows with that ProjectID.
- Change supervisor department → must change it in all rows for projects they supervise.

Insert anomalies

- You can't insert a new **project** (with its title/dates) until at least one student is assigned (because this table needs StudentID).
- You can't insert a new **student** record (just to register them) until they are placed on some project.

Delete anomalies

- If the last student leaves project P42 and you delete their row, you also accidentally delete the **only** occurrences of the project's title/dates and its supervisor linkage.
- If you remove the last project row referencing a supervisor, you lose the only stored details about that supervisor.

3) 1NF Check (atomic values)

- All attributes look atomic (no repeating groups, no arrays).
- **1NF status:** already in 1NF.
- **Fix, if needed:** ensure single values per cell and consistent types (e.g., HoursWorked numeric, dates are proper date/time).

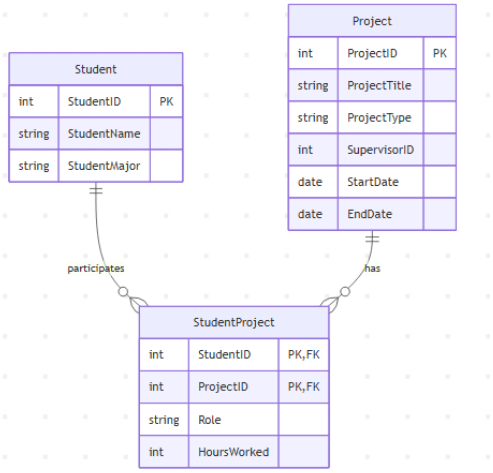
4) 2NF

(a) Choose the primary key of the *current* wide table
Since rows describe a **student's participation on a project**, the natural key is:
(StudentID, ProjectID)
(We assume one row per student per project. If a student can have multiple roles/time slices on the *same* project, extend the key with Role or a LineNo.)
(b) Partial dependencies (violate 2NF)
With composite key (StudentID, ProjectID), any non-key attribute depending on only part is a partial dependency:

- On StudentID only: StudentName, StudentMajor
- On ProjectID only: ProjectTitle, ProjectType, SupervisorID, StartDate, EndDate

(c) 2NF decomposition (remove partial deps)
Create separate tables for the determinants:

- **Student**(StudentID PK, StudentName, StudentMajor)
- **Project**(ProjectID PK, ProjectTitle, ProjectType, SupervisorID, StartDate, EndDate)
- **StudentProject**(StudentID FK, ProjectID FK, Role, HoursWorked, **PK(StudentID,ProjectID)**)



5) 3NF

(a) Transitive dependencies

- ProjectID → SupervisorID and SupervisorID → SupervisorName, SupervisorDept
  So ProjectID → SupervisorName, SupervisorDept is **transitive** via SupervisorID.

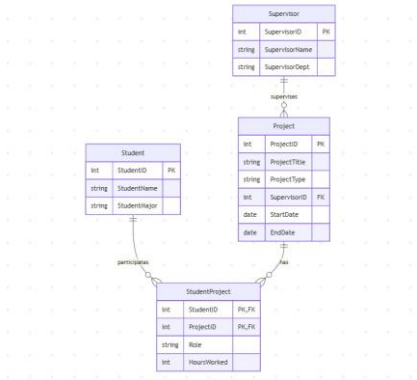
(b) 3NF decomposition (remove transitives)

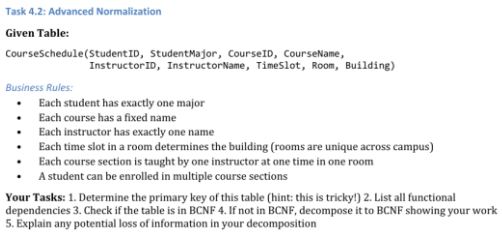
Add a separate Supervisor table and reference it from Project:

- **Supervisor**(SupervisorID PK, SupervisorName, SupervisorDept)
- **Project**(ProjectID PK, ProjectTitle, ProjectType, SupervisorID **FK→Supervisor**, StartDate, EndDate)
- **Student**(StudentID PK, StudentName, StudentMajor)
- **StudentProject**(StudentID **FK→Student**, ProjectID **FK→Project**, Role, HoursWorked, **PK(StudentID,ProjectID)**)

All schemas (final 3NF):

1. Student(StudentID PK, StudentName, StudentMajor)
2. Supervisor(SupervisorID PK, SupervisorName, SupervisorDept)
3. Project(ProjectID PK, ProjectTitle, ProjectType, SupervisorID FK→Supervisor.SupervisorID, StartDate, EndDate)
4. StudentProject(StudentID FK→Student.StudentID, ProjectID FK→Project.ProjectID, Role, HoursWorked, PRIMARY KEY(StudentID, ProjectID))





Task 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 (rooms are unique across campus)
- Each course section is taught by one instructor at one time in one room
- A student can be enrolled in multiple course sections

Your Tasks: 1. Determine the primary key of this table (hint: this is tricky!) 2. List all functional dependencies 3. Check if the table is in BCNF 4. If not in BCNF, decompose it to BCNF showing your work 5. Explain any potential loss of information in your decomposition

1) Primary Key of the table (tricky part)

Each row represents a **student's enrollment in a specific course section**.
The natural identifier for a section (without an explicit SectionID) is **(CourseID, TimeSlot, Room)**.
So a row is uniquely identified by:
PK = (StudentID, CourseID, TimeSlot, Room)
(InstructorID is not needed in the PK since it is functionally dependent on the section.)

2) Functional Dependencies (FDs)

- Student:
  StudentID → StudentMajor
- Course:
  CourseID → CourseName
- Instructor:
  InstructorID → InstructorName
- Room/Building:
  Room → Building
- Section:
  (CourseID, TimeSlot, Room) → InstructorID
- Transitive dependency:
  (CourseID, TimeSlot, Room) → InstructorID → InstructorName
- Whole row:
  (StudentID, CourseID, TimeSlot, Room) → all other attributes

3) BCNF Check

BCNF requires that for every nontrivial FD, the determinant must be a superkey.
Here we have FDs with determinants that are **not superkeys**:

- StudentID → StudentMajor
- CourseID → CourseName
- InstructorID → InstructorName
- Room → Building
- (CourseID, TimeSlot, Room) → InstructorID (doesn't include StudentID)

⇒ **The table is not in BCNF.**

4) Decomposition into BCNF

Break into separate relations based on the determinants:

1. **Student**(StudentID PK, StudentMajor) — BCNF
2. **Course**(CourseID PK, CourseName) — BCNF
3. **Instructor**(InstructorID PK, InstructorName) — BCNF
4. **Room**(Room PK, Building) — BCNF
5. **Section**(CourseID, TimeSlot, Room, InstructorID,
   PK = (CourseID, TimeSlot, Room),
   InstructorID FK→Instructor, Room FK→Room) — BCNF
6. **Enrollment**(StudentID, CourseID, TimeSlot, Room,
   PK = (StudentID, CourseID, TimeSlot, Room),
   FKs: StudentID→Student, (CourseID, TimeSlot, Room)→Section) — BCNF

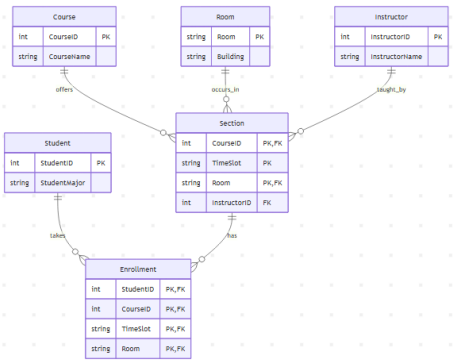
Final BCNF set of relations:

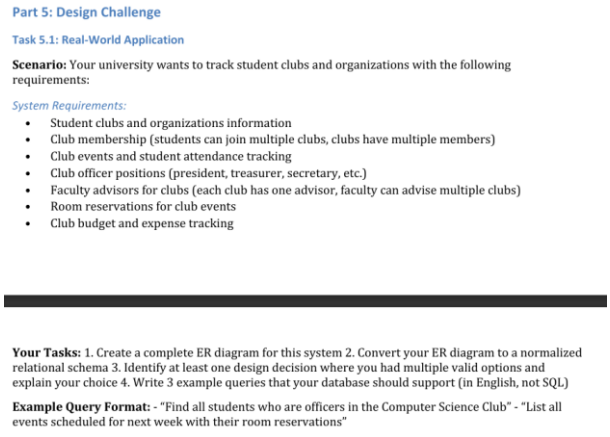
- Student(StudentID PK, StudentMajor)
- Course(CourseID PK, CourseName)
- Instructor(InstructorID PK, InstructorName)
- Room(Room PK, Building)
- Section(CourseID, TimeSlot, Room, InstructorID, PK=(CourseID,TimeSlot,Room), FK Room, FK Instructor)
- Enrollment(StudentID, CourseID, TimeSlot, Room, PK=(StudentID,CourseID,TimeSlot,Room), FK Student, FK Section)

(Optionally, add a surrogate SectionID to simplify keys: then Enrollment PK = (StudentID, SectionID).)

5) Loss of Information?

- **Lossless decomposition:** The original table can be reconstructed by Enrollment ⋈ Section ⋈ Student ⋈ Course ⋈ Instructor ⋈ Room.
- **No information is lost:** each original row is split into facts about enrollment + reference tables (Student, Course, Instructor, Room) + section description.
- **Dependency preservation:** all original FDs appear in the corresponding tables (or as consequences of foreign keys). In strict theory, BCNF decompositions sometimes don't preserve every FD in one table, but here each FD has been assigned to its natural relation.





Part 5: Design Challenge

Task 5.1: Real-World Application

Scenario: Your university wants to track student clubs and organizations with the following requirements:

System Requirements:

- Student clubs and organizations information
- Club membership (students can join multiple clubs, clubs have multiple members)
- Club events and student attendance tracking
- Club officer positions (president, treasurer, secretary, etc.)
- Faculty advisors for clubs (each club has one advisor, faculty can advise multiple clubs)
- Room reservations for club events
- Club budget and expense tracking

Your Tasks: 1. Create a complete ER diagram for this system 2. Convert your ER diagram to a normalized relational schema 3. Identify at least one design decision where you had multiple valid options and explain your choice 4. Write 3 example queries that your database should support (in English, not SQL)

Example Query Format: - "Find all students who are officers in the Computer Science Club" - "List all events scheduled for next week with their room reservations"

Normalized relational schema (PKs, FKs)

- **Student**(**StudentID** PK, Name, Email UQ, Major, Phone)
- **Faculty**(**FacultyID** PK, Name, Email UQ, Department)
- **Club**(**ClubID** PK, ClubName UQ, Description, **AdvisorID** FK→Faculty(FacultyID), FoundedOn)
- **Membership**(**ClubID** FK→Club, **StudentID** FK→Student, JoinDate, Status, **PK(ClubID, StudentID)**)
- **OfficerRole**(**RoleID** PK, RoleName UQ)
- **ClubOfficer**(**ClubID** FK→Club, **StudentID** FK→Student, **RoleID** FK→OfficerRole, StartDate, EndDate, **PK(ClubID, StudentID, RoleID)**)
- **Room**(**RoomID** PK, Building, RoomNumber, Capacity, UQ(Building, RoomNumber))
- **ClubEvent**(**EventID** PK, **ClubID** FK→Club, Title, EventType, StartTime, EndTime, Notes)
- **EventRoomReservation**(**EventID** FK→ClubEvent, **RoomID** FK→Room, ReservedStart, ReservedEnd, **PK(EventID, RoomID)**)
- **EventAttendance**(**EventID** FK→ClubEvent, **StudentID** FK→Student, Status, CheckinMethod, **PK(EventID, StudentID)**)
- **ClubBudget**(**ClubID** FK→Club, **FiscalYear**, ApprovedAmount, Notes, **PK(ClubID, FiscalYear)**)
- **ClubExpense**(**ExpenseID** PK, **ClubID** FK→Club, **FiscalYear** FK→ClubBudget(FiscalYear) with ClubID, ExpenseDate, Category, Amount, Payee, Description)
    - (Implement ClubExpense → ClubBudget as FK on the pair (**ClubID**, **FiscalYear**).)

All tables are in at least 3NF/BCNF under the stated business rules:

- Associative tables use composite keys and contain only attributes fully dependent on those keys.
- No transitive dependencies remain inside base entities (e.g., Advisor is a FK, not duplicated names).

3) Design decision (multiple valid options)

Room reservations per event: one room vs. many rooms

- **Option A (simpler):** put RoomID directly on ClubEvent (1 event → 1 room).
  Simpler schema & queries.
  Can't represent multi-room events or split reservations.
- **Option B (chosen):** separate **EventRoomReservation(EventID, RoomID, ReservedStart, ReservedEnd)**.
  Supports events using multiple rooms or multiple time blocks (e.g., setup in Room A 16:00–17:00, main session in Room B 17:00–19:00).
  Works well if rescheduling fragments occur.
  I chose **Option B** to satisfy broader scheduling and prevent future redesign.

(Another valid decision you could justify: modeling officer roles as a lookup vs. storing role text inline; I used a lookup **OfficerRole** for consistency and integrity.)

---

4) Example user queries (English, not SQL)

1. **"Find all students who are officers in the Computer Science Club."**
2. **"List all club events scheduled for next week with their reserved rooms and buildings."**
3. **"Show the total approved budget and total expenses by category for the Robotics Club this fiscal year."**