**Part 1**

***Task 1.1***

Relation A:

1. Super keys: EmpID, Email, Phone, SSN, {EmpID, Name}, {Email, Department};
2. Candidate keys: EmpID, SSN, Email,Phone;
3. Primary key: EmpID. Cause it is short, easy to recognize and unchangeable.
4. According this table it can’t be, due to each phone is connected to each person in the table.

Relation B:

1. Attributes for the primary key: Student Id, CourseCode, Section, Semester, Year;
2. Student id – without it, it is impossible to determine who exactly has registered.

CourseCode – without it, it is impossible to understand what course the registration is for.

Section – without it, it is impossible to create a protocol for different groups (for example, Section A and Section B).

Semester – a student can take the same course in spring and fall → required.

Year – two identical semesters (fall 2024 and fall 2025) must also be different.

1. Candidate keys: Student Id, CourseCode, Section, Semester, Year;

***Task 1.2***

Foreign key relationships:

1. Enrollment.StudentID 🡪 Student.StudentID;
2. Enrollment.CourseID 🡪 Course.CourseID;
3. Student.Major → Department.DeptCode
4. Student.AdvisorID → Professor.ProfID
5. Professor.Department → Department.DeptCode
6. Course.DepartmentCode → Department.DeptCode
7. Department.ChairID → Professor.ProfID

**Part 2**

***Task 2.1***

1. Strong entities: Patients(PatientID, Name, Birthdate, Adress, Insurance), Doctor(DoctorID, Name, OfficeLocation), Department(DeptCode, DeptName, Location), , , Inctructions);

Weak: Appointment(AppointmentID, DoctorID, VisitDateTime, Purpose, Notes), Prescription(PrescriptionID, DoctorID, Dosage, Hospital\_Room(room\_number, DepartmentCode);

1. Simple: PatientID, DoctorID, DeptCode, Purpose, Notes and etc.

Composite: Adress = {street, city, state, zip};

Multi-valued: Patient\_phone\_number, Doctor\_specialization;

Derived: Age(from Birthdate),

1. 1 : N: Department --< Doctors, Department --< Hospital\_Rooms, Hospital\_Room --< Appointments, Patient --< Phone\_Numbers, Doctor\_Phones;

M : N: Patient --< Appointment >-- Doctor, Doctor >-- Prescription >-- Medication, Doctor ──< Doctor\_Specialization >── Specialization

1. Image

***Task 2.2***

1. Image
2. **INVENTORY** is a **weak entity**:

It **depends on PRODUCT** for its identity and existence.

Its primary key is **ProductID** (a foreign key), and it has no independent key of its own.

If a product is deleted, the inventory row is meaningless and must be deleted as well.

1. Many-to-many relationship(s) that need attributes

ORDER ↔ PRODUCT → ORDER\_ITEM

Attributes needed on the relationship: Quantity, UnitPriceAtOrder (price snapshot at time of purchase).

**Part 4**

***Task 4.1***

1. Core business rules implied by the columns:

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

From 2 & 3:

* **ProjectID → SupervisorName, SupervisorDept**

Candidate key of the current wide table: **(StudentID, ProjectID)**.  
(Everything in a row is determined once you know the student and the project.)

1. **Redundancy:**

* Student data repeats across every project row for the same student.
* Project data (title, type, supervisor) repeats across every student on that project.
* Supervisor data repeats across every project they supervise.

**Update anomaly:**

* If a student changes major, you must update it in many rows; missing one leaves inconsistent data.
* If a supervisor moves to another department, every row for their projects must be updated.

**Insert anomaly:**

* You can’t insert a new Project (with its title/type/supervisor) until at least one student is assigned—otherwise you don’t have a (StudentID, ProjectID) row.
* You can’t store a new Student until they join a project.

**Delete anomaly:**

* If the only student on a project is removed, deleting that row also deletes the only stored copy of the Project and Supervisor info.

1. **Apply 1NF:**

* All attributes are already atomic (no repeating groups or multi-valued fields in this design), so 1NF is satisfied.
* No change needed for 1NF.

1. **Apply 2NF:**

* Since the key is **(StudentID, ProjectID)**, anything depending on only **StudentID** or only **ProjectID** violates 2NF.
* Move student-only attributes to **Student**.
* Move project-only attributes to **Project**.

**2NF Decomposition:**

* **Student(StudentID PK, StudentName, StudentMajor)**
* **Project(ProjectID PK, ProjectTitle, ProjectType, SupervisorID FK)**
* **StudentProject(StudentID FK, ProjectID FK, Role, HoursWorked, StartDate, EndDate, PRIMARY KEY(StudentID, ProjectID))**
* Keep supervisor attributes for 3NF next.

1. **Apply 3NF:**

* In Project, we still have SupervisorID → SupervisorName, SupervisorDept.
* Transitivity: ProjectID → SupervisorID → (SupervisorName, SupervisorDept).
* Split supervisor details into their own table.

**Final 3NF Schemas:**

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

***Task 4.2***

1. Primary keys: (StudentID, TimeSlot, Room)
2. StudentID → StudentMajor

* CourseID → CourseName
* InstructorID → InstructorName
* Room → Building
* (TimeSlot, Room) → CourseID, InstructorID
* (TimeSlot, Room) → CourseName (through CourseID)
* (TimeSlot, Room) → InstructorName (through InstructorID)

1. StudentID → StudentMajor (StudentID is not a superkey).

* CourseID → courseName (not a superkey).
* InstructorID → InstructorName (not a superkey).
* Room → Building (not a superkey).
* (TimeSlot, Room) → CourseID, InstructorID (not a superkey for the entire table, since many students listen to one section).

**Therefore, the source table is not in BCNF.**

1. **Decomposition to BCNF (lossless)**

* We select the entities and the "enrollment" relationship:
* Student(StudentID PK, StudentMajor)
* Course(CourseID PK, CourseName)
* Instructor(InstructorID PK, InstructorName)
* Room(Room PK, Building)

**Defining a "section" (unique in time and room):**

* Section(TimeSlot, Room, CourseID FK, InstructorID FK, PK(TimeSlot, Room))
* And enrollment of students in the sections:
* Enrollment(StudentID FK, TimeSlot FK, Room FK, PK(StudentID, TimeSlot, Room))

1. **Loss of information / addiction:**
   * There is no loss of information (lossless decomposition): CourseSchedule = enrollment ⨝ section ⨝ student ⨝ course ⨝ instructor ⨝ number according to the corresponding keys.
   * Dependencies are stored and locally verifiable.:
   * Student ID → The senior student is a Student.
   * Course ID → Course Name — in Course.
   * Instructor's ID → The instructor's name is in Teacher.
   * Room → Building — in the Room.
   * (Time interval, room) → Course ID, InstructorID — go to the Section.

**Part 5**

***Task 5.1***

1. Image
2. Student(StudentID)

Clubs(ClubID, FacultyAdvisorID)

Organization(OrganizationID, FacultyAdvisorID)

Membership(MembershipID, StudentID, ClubID, OrganizationID)

Event(EventID, ClubID, OrganizationID)

Attendance(AttendanceID, EventID, StudentID)

OfficerPosition(OfficerPositionID, ClubID, StudentID, PositionTitle)

FacultyAdvisor(FacultyAdvisorID)

RoomReservation(ReservationID, EventID)

ClubBudget(BudgetID, ClubID)

1. **Design Decision:**

* **Decision: Should we combine "Club" and "Organization" as one entity?**

**Option 1: Combine "Club" and "Organization" into one entity.**

Since both clubs and organizations are very similar (both have members, events, etc.), we could consider them as a single entity.

**Pros:** Simpler schema, fewer tables.

**Cons:** Loss of flexibility, as some organizations may have additional attributes or constraints not relevant to clubs.

* **Option 2: Keep them as separate entities.**

**Chosen Option:** We decided to keep **Club** and **Organization** as separate entities because the system may need to track specific attributes that apply only to either clubs or organizations (such as unique types of memberships or specific advisor relationships).

* **Why this choice?** It provides more flexibility in the future, and allows for easier expansion if needed (e.g., if organizations require additional features or attributes not shared with clubs).

1. **Example Queries:**

* **Query 1: List all students who are members of a specific club.**
* **Description:** This query will help administrators quickly see which students belong to a specific club.
* **Example:** *Show all students who are members of the "Engineering Club".*
* **Query 2: Find all events organized by a specific club in a given year.**
* **Description:** This query will list all the events held by a club in a particular year.
* **Example:** *List all events organized by the "Science Club" in 2023.*
* **Query 3: Get the total budget and remaining funds for a specific club.**
* **Description:** This query allows club leaders or administrators to track how much money a club has left in its budget.
* **Example:** *Show the total budget and remaining funds for the "Drama Club".*