Part 1. Task 1.1

Relation A

1. List at least 6 different superkeys
   - **EmpID**
   - **SSN**
   - **Email**
   - **EmpID, SSN**
   - **EmpID, Email**
   - **EmpID, Department**

2. Identify all candidate keys
   EmpID, SSN and Email

3. Which candidate key would you choose as primary key and why?
   EmpID, because it's stable value assigned on registation on work. SSN is unique, but sensitive information, so can't be used. Email is not that sensitive, but can be changed because of domain change

4. Can two employees have the same phone number? Justify your answer based on the data shown.
   In provided sample data, there are no duplicates in Phone, but we can't conclude it only on given data. Probably each employee at work has separate phone number registered, but this isn't stated

Relation B

1. Determine the minimum attributes needed for the primary key
   StudentID, CourseCode, Section, Semester, Year

2. Explain why each attribute in your primary key is necessary
   StudentID – to determine student who took course by unique ID
   CourseCode – to determine which course is taken
   Section – to determine in which section student registered
   Semester and Year – to determine on which time student took course

3. Identify any additional candidate keys (if they exist)
   For this table, there're no additional candidate keys. Because two students having different Grade might register on same course. And Credits are already "have fixed value", so it depends on CourseCode, Section, Semester, Year. So, key won't be minimal

Task 1.2

1. Identify all foreign key relationships
   - **Course.DepartmentCode -> Department.DeptCode**
   - **Professor.Department -> Department.DeptCode**
   - **Department.ChairID -> Professor.ProfID**
   - **Enrollment.StudentID -> Student.StudentID**
   - **Enrollment.CourseID -> Course.CourseID**
   - **Student.Major -> Department.DeptCode**
   - **Student.AdvisorID -> Professor.ProfID**

Part 2. Task 2.1

1. Identify all entities (specify which are strong and which are weak)
   Strong: Patient, Doctor, Department
   Weak: Prescription, Appointment, Hospital Room

2. Identify all attributes for each entity (classify as simple, composite, multi-valued, or derived)

Patient

PatientID – simple, primary key
Name – composite
Birthdate – simple
Address – composite
PhoneNumber – multi-valued
InsuranceInformation – simple

Doctor

DoctorID – simple, primary key
Name – composite
Specialization – multi-valued
PhoneNumber – multi-valued
OfficeLocation – multi-valued, composite

Department

DeptCode – simple, primary key
DeptName – simple
Location – composite

Appointment

PatientID – simple, foreign key
DoctorID – simple, foreign key
DateTime – simple
Purpose – simple
Notes – simple

Prescription

PatientID – simple, foreign key
DoctorID – simple, foreign key
Medication – simple
Dosage – simple
Instructions – simple

HospitalRoom

RoomID – simple,
DeptCode – simple, foreign key

3. Identify all relationships with their cardinalities (1:1, 1:N, M:N)
   Department -> Doctor: 1:N
   Doctor -> Department: 1:1
   Department -> HospitalRoom (identifying): 1:N
   HospitalRoom -> Department: 1:1
   Patient -> Appointment: 1:N
   Appointment -> Patient: 1:1
   Doctor -> Appointment: 1:N
   Appointment -> Doctor: 1:1
   Patient -> Prescription: 1:N
   Prescription -> Patient: 1:1
   Doctor -> Prescription: 1:N
   Prescription -> Doctor: 1:1
   Appointment -> Prescription: 1:N
   Prescription -> Appointment: 1:1
   Patient -> Doctor: M:N (through Appointment)
   Patient -> Doctor: M:N (through Prescription)

4. Draw the complete ER diagram using proper notation
   Attached as diagram_1.png

5. Mark primary keys
   Patient.PatientID, Doctor.DoctorID, Department.DeptCode, Appointment.(PatientID, DoctorID, DateTime), Prescription.(PatientID, DoctorID, Appointment.DateTime, Medication), HospitalRoom.(DeptCode, RoomID)

Task 2.2

1. Create a complete ER diagram
   Attached as diagram_2.png

2. Identify at least one weak entity and justify why it's weak
   OrderItem is weak because it can't exist without Order. Its ID is owner key + partial key, ProductID

3. Identify at least one many-to-many relationship that needs attributes
   Order-Product is many-to-many. It's resolved by OrderItem that needs attributes. Single order can contain multiple products, and product can appear in many orders; moreover, we must store per-order facts such as quantity and cost. These are attributes of relationship, so associative entity is required.

Part 4. Task 4.1

1. Identify functional dependencies: List all FDs in the format A → B
   - **StudentID -> StudentName, StudentMajor**
   - **ProjectID -> ProjectTitle, ProjectType**
   - **SupervisorID -> SupervisorName, SupervisorDept**
   - **(StudentID, ProjectID) -> Role, HoursWorked, StartDate, EndDate**
   - **ProjectID -> SupervisorID**

2. Identify problems:
   - What redundancy exists in this table?
   Every project contains student name and major, where ID would be enough. Same for project title and type for student, and supervisor data for projects.
   - Give specific examples of update, insert, and delete anomalies
   - **If supervisor's department will be changed, all projects' data must be updated, otherwise there will inconsistency**

- **If there is new project, it's unavoidable to add a dummy student row before this**
- **If there is only one student assigned to project and they will be deleted, it will rmove only copy of that project and supervisor's data**

3. Apply 1NF:
   Are there any 1NF violations? How would you fix them?
   No 1NF violations, because all attributes are already having atomic values

4. Apply 2NF:
   - What is the primary key of this table?
   (StudentID, ProjectID)
   - Identify any partial dependencies
   StudentID -> StudentName, StudentMajor
   ProjectID -> ProjectTitle, ProjectType
   - Show the 2NF decomposition
   - **Student(StudentID, StudentName, StudentMajor)**
   - **Supervisor(SupervisorID, SupervisorName, SupervisorDept)**
   - **Project(ProjectID, ProjectTitle, ProjectType, SupervisorID FK)**
   - **StudentProject(StudentID FK, ProjectID FK, Role, HoursWorked, StartDate, EndDate)**

   Now every non-key attribute is functionally dependent on part of composite primary key + 1NF is satisfied

5. Apply 3NF:
   - Identify any transitive dependencies
   ProjectID -> SupervisorID -> SupervisorName, SupervisorDept
   - Show the final 3NF decomposition with all table schemas
   - **Student(StudentID, StudentName, StudentMajor)**
   - **Supervisor(SupervisorID, SupervisorName, SupervisorDept)**
   - **Project(ProjectID, ProjectTitle, ProjectType, SupervisorID FK)**
   - **StudentProject(StudentID FK, ProjectID FK, Role, HoursWorked, StartDate, EndDate)**

   Now there are no transitive dependencies + 2NF is satisfied

Task 4.2

1. Determine the primary key of this table (hint: this is tricky!)
   (StudentID, CourseID, TimeSlot, Room)

2. List all functional dependencies
   StudentID -> StudentMajor
   CourseID -> CourseName
   InstructorID -> InstructorName
   Room -> Building
   (CourseID, TimeSlot, Room) -> InstructorID
3. Check if the table is in BCNF
   No, there are dependencies where left sides aren't superkeys
4. If not in BCNF, decompose it to BCNF showing your work
   - **Student(StudentID, StudentMajor)**
   - **Course(CourseID, CourseName)**
   - **Room(Room, Building)**
   - **Section(CourseID FK, TimeSlot, Room FK, InstructorID)**
   - **Instructor(InstructorID, InstructorName)**
   - **Enrollment(StudentID FK, CourseID, TimeSlot, Room)**

5. Explain any potential loss of information in your decomposition
   - **No data will be lost. We now have Enrollment, where full composite key(StudentID FK, CourseID, TimeSlot, Room) is contained. So we can reconstruct all original tuples, and all dependencies are preserved between relations**

Part 5. Task 5.1

1. Create a complete ER diagram for this system
   Attached as diagram_3.png
2. Convert your ER diagram to a normalized relational schema
   Attached as schema_1.png
3. Identify at least one design decision where you had multiple valid options and explain your choice
   Officer positions need to be represented in way that links them to club membership and still allows tracking start and end dates. But storing officer data separately could duplicate membership information and break integrity.
   So, I decided to store Role directly in the Membership table, so that officer information is always tied to active membership record. It removes redundancy, allows 3NF, and simplifies queries.
4. Write 3 example queries that your database should support (in English, not SQL)
   - **"Find all students who are officers in the OSIT Club"**

- **"List all events scheduled for next week with their reserved rooms and capacities"**
- **"Show all expenses for the Crystal Club in 2025 and compare to allocated budget"**