**1.1**  
Relation 1  
1.  
SK1=(EmpID) → unique because each employee has their own ID.  
SK2=(SSN) → unique since every person has a distinct SSN.  
SK3=(Email) → work emails are unique identifiers.  
SK4=(Phone) → in the sample data it is unique.  
SK5=(EmpID, Name) → this combination guarantees uniqueness.  
SK6=(SSN, Email) → this combination also provides uniqueness.  
2.  
CK1=(EmpID)  
CK2=(SSN)  
CK3=(Email)  
These are minimal superkeys, because if you remove any attribute, the key would no longer uniquely identify a record.  
3.  
EmpID as the primary key because it is numeric and short, it does not change over time,  
it is safer and more practical than using SSN or Email.  
4.  
Phone Numbers in the sample data, all phone numbers are unique.However, in real-world situations, two employees might share the same work phone (for example, in one department).  
  
Relation 2  
1.  
Minimum attributes needed for primary key are: StudentID + CourseCode + Section + Semester + Year  
2.  
StudentID: identifies which student is registering. Without it, we could not distinguish registrations of different students for the same course.  
CourseCode: identifies which course is being taken. Without it, we could not tell which course the student is registering for.  
Section: a course can have multiple sections in the same semester. Without Section, two students in different sections of the same course would appear identical.  
Semester: a student can take the same course in different semesters. Without Semester, repeated registrations across semesters would be treated as duplicates.  
Year: he same semester repeats every year. Without Year, registrations in different academic years could not be distinguished.  
3.  
The only candidate key for the Registration relation is the combination:  
(StudentID, CourseCode, Section, Semester, Year).  
StudentID alone is not unique, since one student can register for multiple courses.  
CourseCode alone is not unique, since many students can register for the same course.  
Therefore, no single attribute is sufficient.  
The minimal unique combination is (StudentID, CourseCode, Section, Semester, Year), which serves as both the candidate key and the primary key.

**1.2 task**  
  
Student - Professor  
AdvisorID in Student references ProfID in Professor.  
Each student has an academic advisor who is a professor.  
Professor - Department  
Department in Professor references DeptCode in Department.  
Each professor belongs to a department.  
Course - Department  
DepartmentCode in Course references DeptCode in Department.  
Each course is offered by a department.  
Department - Professor  
ChairID in Department references ProfID in Professor.  
Each department has a chair who is a professor.  
Enrollment - Student  
StudentID in Enrollment references StudentID in Student.  
Each enrollment record specifies which student is registered.  
Enrollment - Course  
CourseID in Enrollment references CourseID in Course.  
Each enrollment record specifies which course the student is registered for.

**Task 2.1**

**1. Entities (Strong and Weak)**

* **Patient** (Strong) – uniquely identified by PatientID
* **Doctor** (Strong) – uniquely identified by DoctorID
* **Department** (Strong) – uniquely identified by DeptCode
* **Appointment** (Weak/Associative) – depends on Patient and Doctor
* **Prescription** (Weak/Associative) – depends on Patient and Doctor
* **HospitalRoom** (Weak) – identified by (DeptCode, RoomNo)
* (DeptCode = «Cardio», RoomNo = 101) (DeptCode = «Neuro», RoomNo = 101)

**2. Attributes**

**Patient:**

* PatientID (Primary Key, simple)
* Name (simple/composite)
* Birthdate (simple) - *Age (derived, вычесляется из других полей)*
* Address (composite: Street, City, State)
* PhoneNumbers (multi-valued)
* InsuranceInfo (simple)

**Doctor:**

* DoctorID (Primary Key, simple)
* Name (simple/composite)
* Specializations (multi-valued)
* Phone (simple)
* OfficeLocation (simple)

**Department:**

* DeptCode (Primary Key, simple)
* DeptName (simple)
* Location (simple)

**Appointment(назначение):**

* AppointmentID (Primary Key, simple) *or composite (PatientID, DoctorID, DateTime)*
* DateTime (simple/composite)
* Purpose (simple)
* Notes (simple)
* *FK:* PatientID - Patient, DoctorID - Doctor

**Prescription:**

* PrescriptionID (Primary Key, simple)
* Date (simple)
* Medication (multi-valued, can be composite with Name, Dosage, Instructions)

**HospitalRoom:**

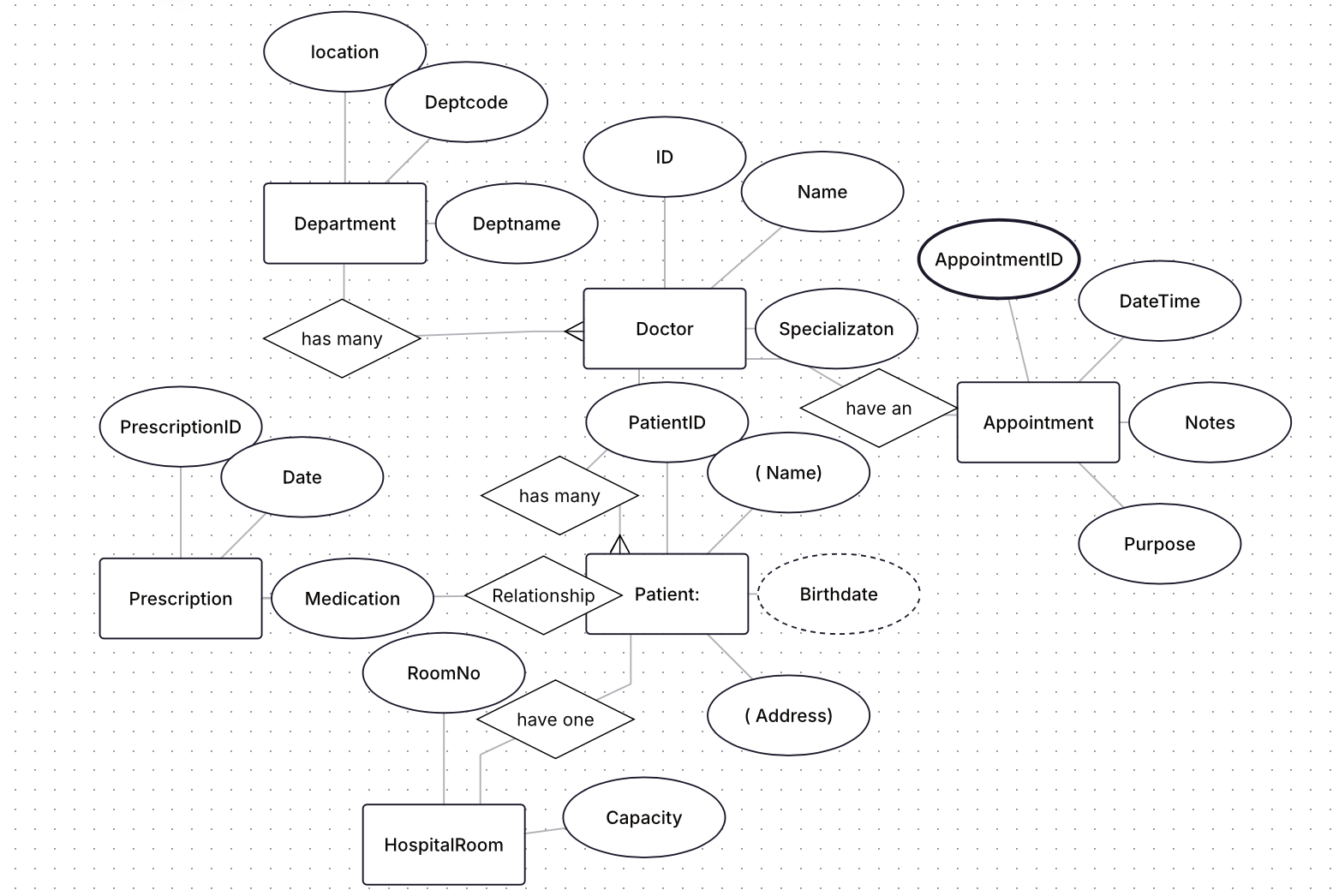
* RoomNo (partial key, simple)
* Capacity (simple)
* *PK:* (DeptCode, RoomNo)
* *FK:* DeptCode → Department

**3. Relationships with Cardinalities**

* **Department – Doctor:** 1 : N (a department has many doctors)
* **Patient – HospitalRoom:** 1 : 1 (for 1 patient given one patient room)
* **Patient – Prescription – Doctor:** M : N resolved by Prescription (a doctor can prescribe many prescriptions to many patients)

**5. Primary Keys**

* Patient → PatientID
* Doctor → DoctorID
* Department → DeptCode
* Appointment → AppointmentID *(or composite: PatientID + DoctorID + DateTime)*
* Prescription → PrescriptionID
* HospitalRoom → (DeptCode, RoomNo)

****

**Task 2.2**

**Entities & Primary Keys**

* Customer(CustomerID PK, Name, Email, Phone)
* Address(AddressID PK, CustomerID FK, Street, City, State, Zip, Country, AddressType {Billing, Shipping})
* Order(OrderID PK, CustomerID FK, OrderDate, Status, BillingAddressID FK→Address, ShippingAddressID FK→Address, TotalAmount *derived*)
* OrderItem(OrderID PK/FK, ProductID PK/FK, Quantity, UnitPriceAtOrder)
* Product(ProductID PK, Name, Description, ListPrice, CategoryID FK)
* Category(CategoryID PK, CategoryName, ParentCategoryID FK (nullable, self))
* Vendor(VendorID PK, VendorName, Email, Phone)
* VendorProduct(VendorID PK/FK, ProductID PK/FK, VendorSKU, Cost, LeadTimeDays)
* Review(ReviewID PK, ProductID FK, CustomerID FK, Rating, Comment, ReviewDate)
* Inventory(ProductID PK/FK, OnHandQty, ReorderLevel)

Relationships (with cardinalities)

* Customer 1:N Order
* Customer 1:N Address (a customer can store multiple billing/shipping addresses)
* Order 1:N OrderItem; Product 1:N OrderItem → combined gives M:N between Order and Product resolved via OrderItem
* Product N:1 Category (optional self-hierarchy via ParentCategoryID)
* Product M:N Vendor via VendorProduct (a product may be supplied by many vendors; a vendor supplies many products)
* Product 1:N Review and Customer 1:N Review
* Product 1:1 Inventory (one inventory record per product)

Weak Entity (and justification)

**OrderItem** → weak entity, because it exists only with Order and Product.

At least one M:N relationship that needs attributes

* Order ↔ Product is naturally many-to-many and requires attributes Quantity and UnitPrice at the time of order. Therefore it is implemented via the associative entity OrderItem(OrderID, ProductID, Quantity, UnitPriceAtOrder).
* (Another example) Vendor ↔ Product is many-to-many and needs attributes such as Cost, VendorSKU, LeadTimeDays, implemented via VendorProduct.

**Task 4.1**

**Functional Dependencies (FDs)**

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

**2) Problems (redundancy & anomalies)**

* **Redundancy:** student, project, and supervisor details repeat across many rows.
* **Update anomaly:** changing a supervisor’s department requires updates in multiple rows.
* **Insert anomaly:** cannot insert a new project or supervisor unless a student is assigned.
* **Delete anomaly:** deleting the last student on a project loses the project/supervisor info.

**3) Apply 1NF**

* Attributes are atomic, simple : 1NF holds.
* But table still mixes independent facts, causing duplication.

**4) Apply 2NF**

* **Primary key:** (StudentID, ProjectID).
* **Partial dependencies:**
  + StudentID → StudentName, StudentMajor
  + ProjectID → ProjectTitle, ProjectType
  + SupervisorID → SupervisorName, SupervisorDept (non-key, but should live in its own entity)
* **2NF decomposition:**
  + **Student**(StudentID PK, StudentName, StudentMajor)
  + **Project**(ProjectID PK, ProjectTitle, ProjectType, SupervisorID FK)
  + **Supervisor**(SupervisorID PK, SupervisorName, SupervisorDept)
  + **StudentProject**(StudentID PK/FK, ProjectID PK/FK, Role, HoursWorked, StartDate, EndDate)

**5) Apply 3NF**

* Transitive deps moved out (SupervisorID : SupervisorDept is inside **Supervisor**).
* **Final 3NF schema:**
  1. **Student**(StudentID, StudentName, StudentMajor)
  2. **Project**(ProjectID, ProjectTitle, ProjectType, SupervisorID FK)
  3. **Supervisor**(SupervisorID, SupervisorName, SupervisorDept)
  4. **StudentProject**(StudentID FK, ProjectID FK, Role, HoursWorked, StartDate, EndDate)

**Task 4.2**

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

**1) Primary key of the given table**

**Candidate/primary key:** **(StudentID, CourseID, TimeSlot, Room)**  
*Why:* a row represents one student’s enrollment in a specific **course section**; a section is identified by **(CourseID, TimeSlot, Room)** and a student can take many sections.

**2) Functional dependencies (FDs)**

* StudentID - StudentMajor
* CourseID - CourseName
* InstructorID - InstructorName
* Room - Building  *(rooms are unique across campus; building is determined by room)*
* (CourseID, TimeSlot, Room) → InstructorID  *(each section is taught by one instructor)*
* And by the key: (StudentID, CourseID, TimeSlot, Room) → all attributes

**3) Is the table in BCNF?**

**No.** Violations (determinant is **not** a superkey):

* StudentID - StudentMajor
* CourseID - CourseName
* InstructorID - InstructorName
* Room - Building
* (CourseID, TimeSlot, Room) - InstructorID  *(not a superkey of the whole table because it lacks StudentID)*

**4) Decompose to BCNF**

Create the following relations (all in BCNF):

1. **Student**(StudentID **PK**, StudentMajor)
2. **Course**(CourseID **PK**, CourseName)
3. **Instructor**(InstructorID **PK**, InstructorName)
4. **Room**(Room **PK**, Building)
5. **Section**(  
   CourseID **FK**→Course,  
   TimeSlot,  
   Room **FK**→Room,  
   InstructorID **FK**→Instructor,  
   **PK** = (CourseID, TimeSlot, Room)  
   ) — and FD PK → InstructorID holds inside this table
6. **Enrollment**(  
   StudentID **FK**→Student,  
   CourseID, TimeSlot, Room **FK**→Section,  
   **PK** = (StudentID, CourseID, TimeSlot, Room)  
   )

**5) Loss of information / dependency preservation**

* Lossless-join: Yes. Joins are lossless because each decomposition step shares a key (Section key (CourseID, TimeSlot, Room) with Enrollment, and simple keys with Student, Course, Instructor, Room)