Task 1.1: Superkey and Candidate Key Analysis

Relation A: Employee(EmpID, SSN, Email, Phone, Name, Department, Salary)

1. **Superkeys:**
- {EmpID}
- {SSN}
- {Email}
- {EmpID, Name}
- {SSN, Phone}
- {EmpID, Email}
2. **Candidate Keys:**
- {EmpID}
- {SSN}
- {Email}
3. **Choice of Primary Key:**
   I choose **EmpID** as the primary key because it is a short numeric identifier that is easy to use. Unlike SSN or Email, it does not change when employee information changes. It is also the standard way to identify employees in most systems.
4. **Can Phone be unique?**
   No. Two employees may share the same phone number (for example, a shared office phone). Therefore, Phone cannot serve as a key.

Relation B: Registration(StudentID, CourseCode, Section, Semester, Year, Grade, Credits)

1. **Minimal Primary Key:**
   (StudentID, CourseCode, Section, Semester, Year)
2. **Why these attributes are necessary:**
- StudentID is required to distinguish students.
- CourseCode and Section are needed because the same course can have multiple sections.
- Semester and Year are necessary because a student can retake the same course in different semesters or years.
3. **Additional candidate keys:**
   In this schema, no other candidate keys exist. However, if there were a unique attribute like SectionID for each course section, then (StudentID, SectionID, Semester, Year) could also be a candidate key.

Task 1.2: Foreign Key Design

Foreign Key Relationships:

1. Student.AdvisorID is a foreign key referencing Professor.ProfID.
2. Course.DepartmentCode is a foreign key referencing Department.DeptCode.
3. Department.ChairID is a foreign key referencing Professor.ProfID.
4. Enrollment.StudentID is a foreign key referencing Student.StudentID.
5. Enrollment.CourseID is a foreign key referencing Course.CourseID.

Explanation:

- Each student is assigned an academic advisor, who must be a professor.
- Each course belongs to a department, so DepartmentCode connects courses to departments.
- Each department has a chair, who is also a professor.
- Each enrollment record links a student with a course in a specific semester.

Task 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

2. Problems (Anomalies):

- **Redundancy:** StudentName is repeated for each project a student joins. SupervisorName is repeated for each project they supervise.
- **Update anomaly:** If a supervisor changes department, it must be updated in many rows.
- **Insert anomaly:** Cannot add a new project unless a student is already assigned.
- **Delete anomaly:** If the last student working on a project is removed, project data is lost.

3. 1NF (First Normal Form):

- The table is already in 1NF (all attributes are atomic, no multivalued attributes).

4. 2NF (Second Normal Form):

- Primary Key: (StudentID, ProjectID).
- Partial dependencies:
  - StudentID → StudentName, StudentMajor
  - ProjectID → ProjectTitle, ProjectType
  - SupervisorID → SupervisorName, SupervisorDept

Decomposition to 2NF:

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

5. 3NF (Third Normal Form):

- Check for transitive dependencies: SupervisorID → SupervisorName, SupervisorDept (already separated).
- Final 3NF Tables:
  - **Student(StudentID, StudentName, StudentMajor)**
  - **Supervisor(SupervisorID, SupervisorName, SupervisorDept)**
  - **Project(ProjectID, ProjectTitle, ProjectType, SupervisorID)**
  - **StudentProject(StudentID, ProjectID, Role, HoursWorked, StartDate, EndDate)**

Task 4.2: Advanced Normalization

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

1. Primary Key:

(StudentID, CourseID, TimeSlot, Room)
Because a student can take multiple courses, and each course section happens at a specific time in a room.

2. Functional Dependencies (FDs):

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

3. BCNF Check:

- **StudentID → StudentMajor** violates BCNF (StudentID is not a superkey).
- **CourseID → CourseName** violates BCNF.
- **InstructorID → InstructorName** violates BCNF.
- **Room → Building** violates BCNF.

4. Decomposition to BCNF:

- Student(StudentID, StudentMajor)
- Course(CourseID, CourseName)
- Instructor(InstructorID, InstructorName)
- Room(Room, Building)
- CourseSection(CourseID, TimeSlot, Room, InstructorID)
- Enrollment(StudentID, CourseID, TimeSlot, Room)

5. Loss of Information:

- No information is lost, but some **functional dependencies** are not preserved in one table (e.g., InstructorID → InstructorName is now stored in a separate table).

Task 5.1: Student Clubs and Organizations

1. Entities and Relationships

- **Student**: Each student has a unique StudentID, name, email, and major.
- **Club**: Each club has a unique ClubID, club name, description, and budget.
- **Membership**: Represents the many-to-many relationship between students and clubs. Attributes: JoinDate and Role.
- **Event**: Each club organizes multiple events. Attributes: EventID, EventName, Date, and RoomID.
- **Attendance**: Represents the participation of students in events (EventID, StudentID, Status).
- **Officer**: Special type of membership where students have officer positions (president, treasurer, secretary, etc.).
- **FacultyAdvisor**: Each club has one advisor. Attributes: AdvisorID, Name, Department, Email.
- **ClubAdvisor**: Connects clubs with their faculty advisors.
- **Room**: Rooms are reserved for club events (RoomID, Location, Capacity).
- **Expense**: Tracks club expenses (ExpenseID, Amount, Date, Description).

Relationships:

- Student ↔ Club = M:N (via Membership).
- Club ↔ Event = 1:N.
- Student ↔ Event = M:N (via Attendance).
- Club ↔ Officer = 1:N.
- Club ↔ FacultyAdvisor = 1:N (via ClubAdvisor).
- Event ↔ Room = N:1.
- Club ↔ Expense = 1:N.

2. Relational Schema

- **Student(StudentID PK, Name, Email, Major)**
- **Club(ClubID PK, ClubName, Description, Budget)**
- **Membership(StudentID FK, ClubID FK, JoinDate, Role, PK(StudentID, ClubID))**
- **Event(EventID PK, ClubID FK, EventName, Date, RoomID FK)**
- **Attendance(EventID FK, StudentID FK, Status, PK(EventID, StudentID))**
- **Officer(ClubID FK, StudentID FK, Position, PK(ClubID, StudentID))**
- **FacultyAdvisor(AdvisorID PK, Name, Department, Email)**
- **ClubAdvisor(ClubID FK, AdvisorID FK, PK(ClubID, AdvisorID))**
- **Room(RoomID PK, Location, Capacity)**
- **Expense(ExpenseID PK, ClubID FK, Amount, Date, Description)**

3. Design Decision

There are two possible ways to represent officer roles:

- Store officer positions as an attribute in Membership.
- Create a separate Officer table.

Decision: We created a separate Officer table because officer roles are official positions and need to be tracked independently from general membership.

3. Design Decision

There are two possible ways to represent officer roles:

- Store officer positions as an attribute in Membership.
- Create a separate Officer table.

Decision: We created a separate Officer table because officer roles are official positions and need to be tracked independently from general membership.

4. Example Queries (in English, not SQL)

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 budget and total expenses for each club this semester.*