Part 1: Key Identification Exercises

Task 1.1: Superkey and Candidate Key Analysis

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) Superkeys: {EmpID}; {SSN};{Email} ; {EmpID , Name} ; {EmpID, Phone}; {SSN, Email} ; {EmpID, SSN} ; {SSN, Name} ; {EmpID, Name, Department}
2) Candidate Keys**:** {EmpID} ; {SSN} ; {Email}
3) I would choose EmpID as primary key, because primary keys ensures uniqueness. It is only one per relation; stable, rarely changes; cannot be null. EmpID is easier to index, store, and transfer as foreign keys in other tables.
4) Yes, two employees can have the same phone number, because phone numbers are not unique keys. According to the data, the uniqueness of Phone cannot be guaranteed. Phone is not a candidate key.

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)The minimal attributes needed for the primary key are (StudentID, CourseCode, Section, Semester, Year)

2) **StudentID**: Identifies the student registering for the course.

CourseCode: Specifies the course being taken.

Section: Distinguishes between different sections of the same course in a semester.

Semester: Indicates the academic term, allowing the same course to be taken in different semesters.

Year: Differentiates registrations across different years for the same semester.

Grade and Credits aren't needed as they don't ensure uniqueness

3) There are no additional candidate keys beyond the primary key

TASK 1.2

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) Student.AdvisorID → Professor.ProfID

The AdvisorID in the Student table likely references a professor who acts as the student's academic advisor. This creates a relationship where each student has one advisor (a professor), and a professor can advise multiple students. Links a student to their academic advisor (a professor).

2) Department.ChairID → Professor.ProfID

The ChairID in the Department table represents the professor who is the department chair. This establishes a relationship where each department has one chair (a professor), and a

professor can chair only one department (assuming this is the rule). Links a department to its chair (a professor).

3) Enrollment.StudentID → Student.StudentID

The StudentID in the Enrollment table must reference a valid student. This links each enrollment record to a specific student, ensuring that only registered students can enroll in courses. Ensures an enrollment record belongs to a valid student.

4) Enrollment.CourseID → Course.CourseID

The CourseID in the Enrollment table must reference a valid course. This links each enrollment record to a specific course, ensuring that enrollments are only for courses that exist in the catalog. Ensures an enrollment record is for a valid course.

5) Course.DepartmentCode → Department.DeptCode

The DepartmentCode in the Course table likely indicates the department that offers the course. This creates a relationship where each course is offered by one department, and a department can offer many courses. Links a course to the department that offers it.

Part 2: ER Diagram Construction

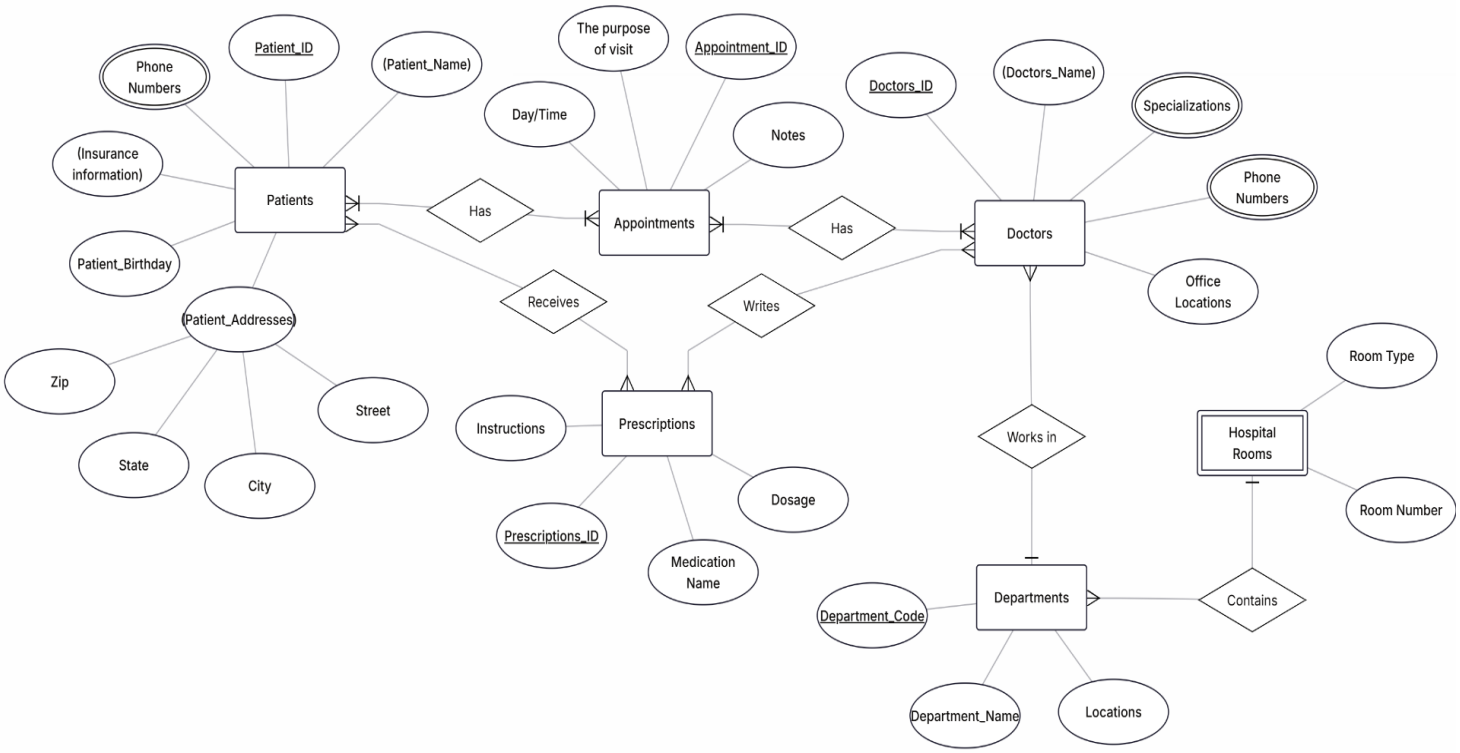
Task 2.1: Hospital Management System

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



Task 2.2

Task 2.2: E-commerce Platform

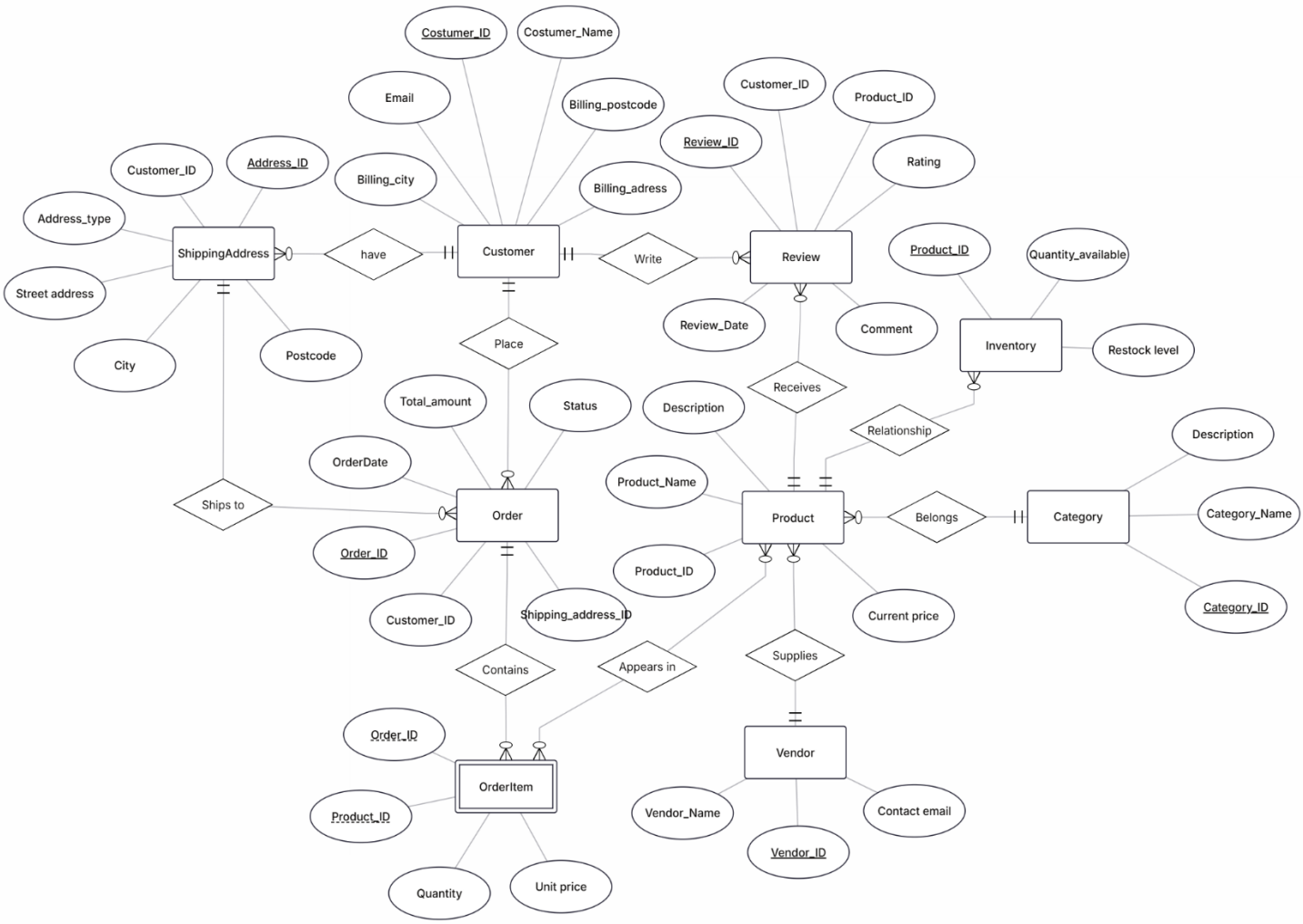
Scenario: Design a simplified e-commerce database.

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)



2) Weak Entity: ORDER_ITEM

Why? Because existence dependency, Non-Identifying Primary Key. If an order is deleted, all its associated order items must also be deleted, which is a classic characteristic of a weak entity.

3) The relationship between PRODUCT and ORDERS is many-to-many. One order can contain many different products, and one product can be sold in many different orders.

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

1)StudentID → StudentName, StudentMajor

ProjectID → ProjectTitle, ProjectType, SupervisorID

SupervisorID → SupervisorName, SupervisorDept

(StudentID, ProjectID) → Role, HoursWorked, StartDate, EndDate

2) Redundancy:

Student details (Name, Major) are repeated for every project the student works on.

Project details (Title, Type, SupervisorID, SupervisorName, SupervisorDept) are repeated for every student working on the project.

Supervisor details (Name, Dept) are repeated for every project they supervise and every student in those projects.

Update Anomaly:

If a student changes their major, we must update multiple rows (all projects they are involved in). If we miss one, the data becomes inconsistent.

If a project's title changes, we must update all rows for that project.

If a supervisor changes departments, we must update all rows for all projects they supervise.

Insertion Anomaly:

We cannot add a new project (with its details) until at least one student is assigned to it.

We cannot add a new supervisor until they have at least one project (and at least one student assigned to that project).

Deletion Anomaly:

If we delete the only student working on a project, we lose all information about that project.

If we delete all students from a project, we lose the project details and supervisor details.

3)1NF requires that all attributes are atomic. There are no repeating groups or composite attributes that need splitting. The table appears to be in 1NF.

4) The primary key is (StudentID, ProjectID) because it uniquely identifies each row. 2NF- no partial dependencies.

Partial Dependencies:

StudentID → StudentName, StudentMajor (depends on part of the key)

ProjectID → ProjectTitle, ProjectType, SupervisorID (depends on part of the key)

SupervisorID → SupervisorName, SupervisorDept (transitive, but also a partial dependency via ProjectID)

2NF Decomposition:

To remove partial dependencies, we decompose into:

1.Students(StudentID, StudentName, StudentMajor)

PK: StudentID

2.Projects(ProjectID, ProjectTitle, ProjectType, SupervisorID)

PK: ProjectID

3.Supervisors(SupervisorID, SupervisorName, SupervisorDept)

PK: SupervisorID

4.StudentProjects(StudentID, ProjectID, Role, HoursWorked, StartDate, EndDate)

PK: (StudentID, ProjectID)

5)3NF- no transitive dependencies. To remove transitive dependencies, we further decompose the Projects table

Projects(ProjectID, ProjectTitle, ProjectType, SupervisorID) (keep as is, but SupervisorID is a FK to Supervisors)

Supervisors(SupervisorID, SupervisorName, SupervisorDept)

Decomposition (3NF):

Students(StudentID, StudentName, StudentMajor)

Supervisors(SupervisorID, SupervisorName, SupervisorDept)

Projects(ProjectID, ProjectTitle, ProjectType, SupervisorID)

StudentProjects(StudentID, ProjectID, Role, HoursWorked, StartDate, EndDate)

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) The primary key of this table is StudentID, CourseID, TimeSlot, Room. Because the course provided when we have student and the row stores a record that the student is enrolled in a specific course section. The course section is specified by a triple: (CourseID, TimeSlot, Room(was not stated that it has unique room))
2) List all functional dependencies.

StudentID → StudentMajor

CourseID → CourseName

InstructorID → InstructorName

Room → Building

(CourseID, TimeSlot, Room) → InstructorID

(StudentID, CourseID, TimeSlot, Room) → all

3)BCNF – all dependencies from superkeys

BCNF requires that for every FD X → Y, X is a superkey.

StudentID → StudentMajor: StudentID is not a superkey (it cannot determine the entire row). So violates BCNF.

CourseID → CourseName: CourseID is not a superkey. Violates BCNF.

InstructorID → InstructorName: InstructorID is not a superkey. Violates BCNF.

Room → Building: Room is not a superkey. Violates BCNF.

So it is not BCNF.

4) Decompose using StudentID → StudentMajor

Create Students(StudentID, StudentMajor)

Remove StudentMajor from the original table, and keep StudentID as a foreign key.

The original table now has: (StudentID, CourseID, CourseName, InstructorID, InstructorName, TimeSlot, Room, Building)

But we still have other FDs.

Decompose using CourseID → CourseName

Create Courses(CourseID, CourseName)

Remove CourseName from the original table, and keep CourseID as a foreign key.

Now the original table has: (StudentID, CourseID, InstructorID, InstructorName, TimeSlot, Room, Building)

Decompose using InstructorID → InstructorName

Create Instructors(InstructorID, InstructorName)

Remove InstructorName from the original table, and keep InstructorID as a foreign key.

Now the original table has: (StudentID, CourseID, InstructorID, TimeSlot, Room, Building)

Decompose using Room → Building

Create Rooms(Room, Building)

Remove Building from the original table, and keep Room as a foreign key.

Now the original table has: (StudentID, CourseID, InstructorID, TimeSlot, Room)

Now consider (CourseID, TimeSlot, Room) → InstructorID

Create Sections(CourseID, TimeSlot, Room, InstructorID)

Remove InstructorID from the original table, and keep (CourseID, TimeSlot, Room) as a foreign key.

The original table now becomes Enrollments(StudentID, CourseID, TimeSlot, Room)

Now, we have the following tables:

Students(StudentID, StudentMajor)

Courses(CourseID, CourseName)

Instructors(InstructorID, InstructorName)

Rooms(Room, Building)

Sections(CourseID, TimeSlot, Room, InstructorID)

PK: (CourseID, TimeSlot, Room)

FK: CourseID references Courses(CourseID)

FK: InstructorID references Instructors(InstructorID)

FK: Room references Rooms(Room)

Enrollments(StudentID, CourseID, TimeSlot, Room)

PK: (StudentID, CourseID, TimeSlot, Room)

FK: StudentID references Students(StudentID)

FK: (CourseID, TimeSlot, Room) references Sections(CourseID, TimeSlot, Room)

Check each table for BCNF:

Students: StudentID is key, and StudentID → StudentMajor. So BCNF.

Courses: CourseID is key, and CourseID → CourseName. So BCNF.

Instructors: InstructorID is key, and InstructorID → InstructorName. So BCNF.

Rooms: Room is key (since rooms are unique across campus), and Room → Building. So BCNF.

Sections: The key is (CourseID, TimeSlot, Room). We have (CourseID, TimeSlot, Room) → InstructorID, which is fine. Also, no other FDs. So BCNF.

Enrollments: The key is (StudentID, CourseID, TimeSlot, Room). There are no non-trivial FDs (since all attributes are part of the key). So BCNF.

PART 5: Design Challenge

1

Task 5.1: Real-World Application

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

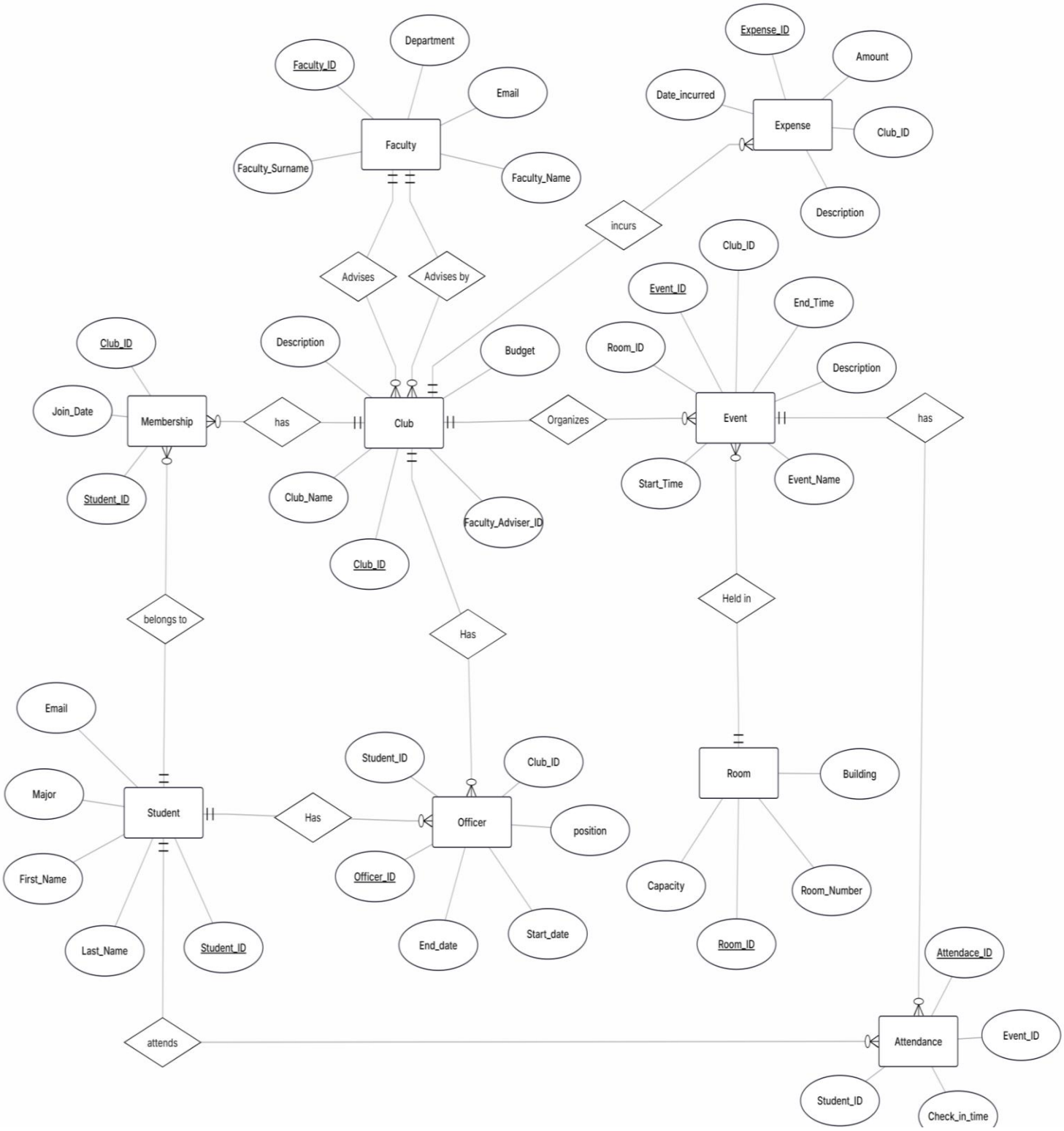
System Requirements:

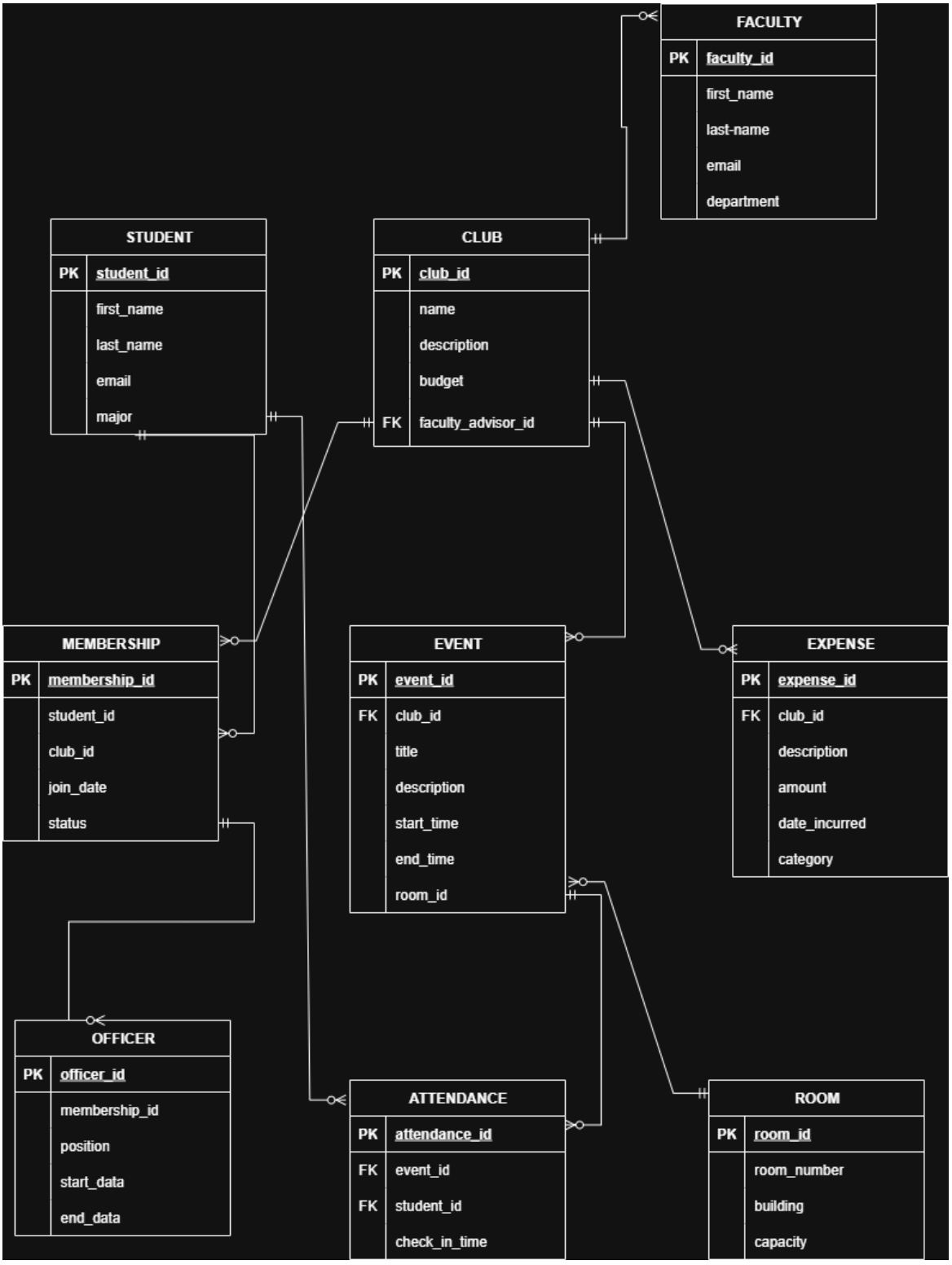
- Student clubs and organizations information
- Club membership (students can join multiple clubs, clubs have multiple members)
- Club events and student attendance tracking
- Club officer positions (president, treasurer, secretary, etc.)
- Faculty advisors for clubs (each club has one advisor, faculty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

1)


Faculty_ID
Department
Email
Faculty
Faculty_Surname
Faculty_Name
Advises
Advises by
Expense_ID
Amount
Date_incurred
Expense
Club_ID
Description
incurs
Club_ID
Event_ID
End_Time
Room_ID
Description
Budget
Club_ID
Join_Date
Membership
has
Club
Organizes
Event
has
Start_Time
Event_Name
Student_ID
Club_Name
Faculty_Adviser_ID
Club_ID
Held in
belongs to
Has
Email
Student_ID
Club_ID
Room
Building
Major
Student
Has
Officer
position
First_Name
Officer_ID
Capacity
Room_Number
Last_Name
Student_ID
End_date
Start_date
Room_ID
Attendace_ID
attends
Attendance
Event_ID
Student_ID
Check_in_time

2) 

3) In designing the room reservation system, we had to decide whether each event should be linked to exactly one room (one-to-one) or if multiple events could share the same room at different times (many-to-one). We chose the one-to-one approach because it keeps the database design simple and avoids the need for complicated checks within the schema. This assumes that once a room is reserved for an event, it cannot be shared at the same time. If overlapping bookings need to be prevented, that can be

handled at the application level by checking the event's date and time before saving it. This approach reduces complexity while still supporting the university's scheduling needs.

4) Find all students who are officers in the Computer Science Club.

List all events scheduled for next week with their room reservations.

Show the total expenses for each club this semester.

List all clubs advised by faculty members from the Computer Science department.

Retrieve the total expenses for each club in the current fiscal year, sorted by club name.