First Database System Laboratory Work

Part 1: Key Identification Exercises

Task 1.1

1. {EmpID}, {SSN}, {Email}, {EmpID, SSN}, {EmpID, Email}, {SSN, Email}, {EmpID, Phone}, {SSN, Phone}, {Email, Phone}, {EmpID, SSN, Email}, {EmpID, SSN, Phone}
2. EmpID, SSN, Email
3. EmpID. Because it doesn't depend on external data. For example, SSN and email address can change or be confidential.
4. Yes, it is possible, because two employees can have a common work phone. Therefore, Phone is not suitable as a key.

Relation B

1. {StudentID, CourseCode, Section, Semester, Year}
2. StudentID-identifies the student, CourseCode identifies the course, Section-specific group, Semester uniquifies the record and Year also uniquifies the recoed
3. StudentID, CourseCode, Section, Semester, Year

Task 1.2

1. Student.StudentID and Enrollment.StudentID
2. Student.AdvisorID and Professor.ProffID
3. Course.DepartmentCode and Department.DeptCode
4. Department.ChairID and Professor.ProfID
5. Enrollment.CourseID and Course.CourseID

Part2. ER Diagram Construction

Task 2.1

1. Strong Entities: Patient, Doctor, Department, HospitalRoom
   Weak Entities: Appointment, Prescription (They cannot exist without the Patient and Doctor entities.)
2. Attributes:
   Patient:
   - PatientID (Simple)
   - Name (composite - first_name and last_name)
   - Birthdates (simple)
   - Address (Composite - street, city, state)
   - Phone (multi-valued)
   - Insurance (simple)

   Doctor:
   - DoctorID (simple)
   - Name (composite - first_name and last_name)
   - Specialization (multi-valued)
   - Phone (simple)
   - Location (simple)

   Department:
   - DeptCode (simple)
   - DeptName (simple)
   - Location (simple)

   Appointment:
   - AppointmentID (simple)

- Date (simple)
- Time (simple)
- Purpose (simple)
- Notes (simple)

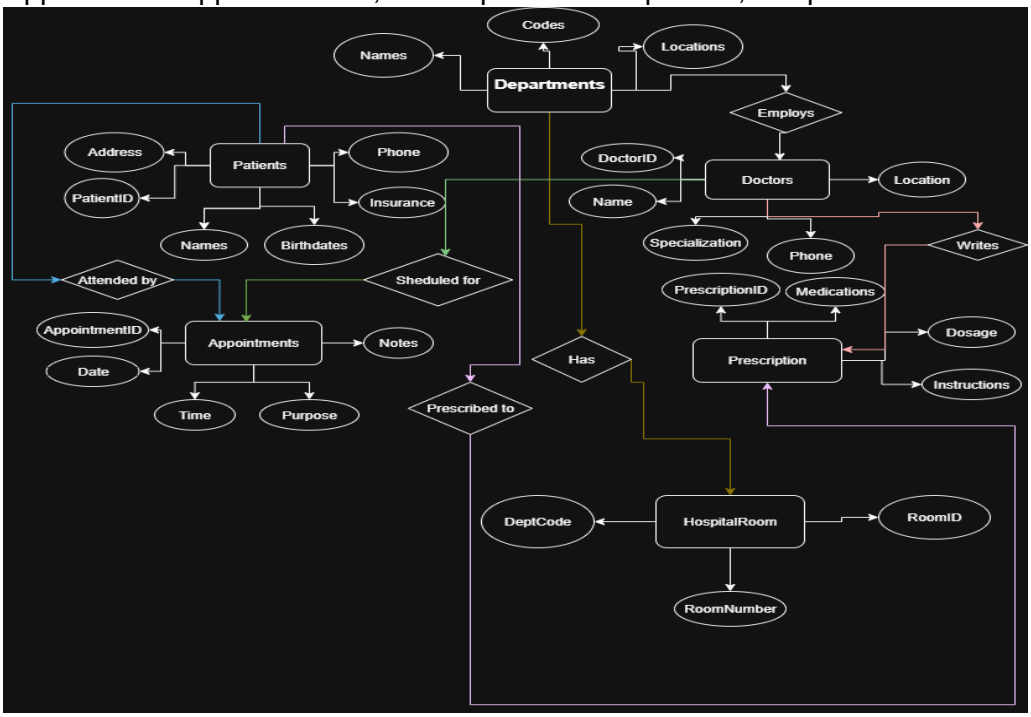
Prescription:

- PrescriptionID (simple)
- Medication (multi-valued)
- Dosage (simple)
- Instructions (multi-valued)

HospitalRoom:

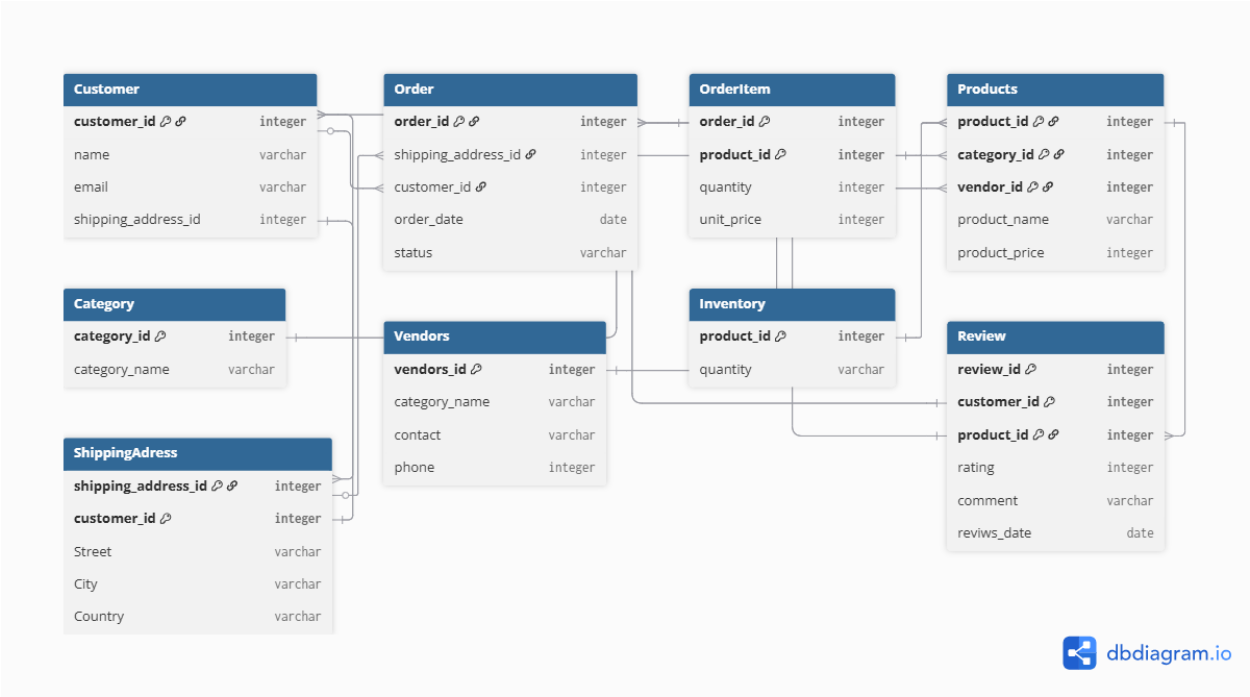
- RoomID (simple)
- RoomNumber (simple)
- DeptCode (simple)

3. Patient – Appointment – Doctor (M:N)
   Doctor – Prescription – Patient (M:N)
   Department – Doctor (1:N)
   Department – Patient (1:N)
   Department – HospitalRoom (1:N)
4. ER diagram
5. Primary keys: Patients-PatientID, Doctors-DoctorID, Department-DeptCode, Appointment-AppointmentID, Prescription-PrescriptionID, HospitalRoom-RoomID



Task2.2

1.



2. Weak Entity: OrderItem
   OderItem can't exist independently of an order. Its uniqueness and the very meaning of existence are determined by a specific order. The primary key of OrderItem is composite and includes the foreign key order_id (from the Order entity) and product_id.
3. Customers-reviews-Products
   One customer can leave reviews for many products. One product can have reviews from many customers. This relationship itself has attributes that belong specifically to the fact of leaving a review, and not to the customer or the product separately. Rating, comments, date are relationship attributes.

Task4.1

1. StudentID -> StudentName, StudentMajor
   ProjectID -> ProjectTitle, ProjectType, SupervisorID
   SupervisorID -> SupervisorName, SupervisorDept
   {StudentID, ProjectID} -> Role, HoursWorked, StartDate, EndDate, SupervisorID
2. **Redundancy**:
   StudentName and StudentMajor are repeated for each project they are involved in.
   ProjectTitle and ProjectType are repeated for each student working on it.
   SupervisorName and SupervisorDept sre repeated for each student and project they are associated with
   Update Anomaly: If a student changes his major, all lines with his participation in projects must be updated. If even one is missed, the data will become inconsistent.
   Insertion Anomaly: Cannot add a new supervisor to the database until he is assigned to at least one project. Cannot add a new student until he is assigned to a project
   Deletion Anomaly: If we delete the last student working on a project, we will also inadvertently delete all information about that project
3. No, because all attributes are indivisible. There are no repeating groups or arrays.
4. **Primary Key**: {StudentID, ProjectID}
   Partial Dependencies
   StudentID -> StudentName, StudentMajor
   ProjectID -> ProjectTitle, ProjectType, SupervisorID
   SupervisorID -> SupervisorName, SupervisorDept (attributes don't dependent on primary key at all)

2NF Decomposition
Students: StudentID (PK), StudentName, StudentMajor
Projects: ProjectsID(PK), ProjectName, ProjectType, SupervisorID
Supervisors: SupervisorID(PK), SupervisorName, SupervisorDept
Student_Project: StudentID(PK, FK to Students), ProjectID(PK, FK, to Projects), Role, HoursWorked, StartDate, EndDate

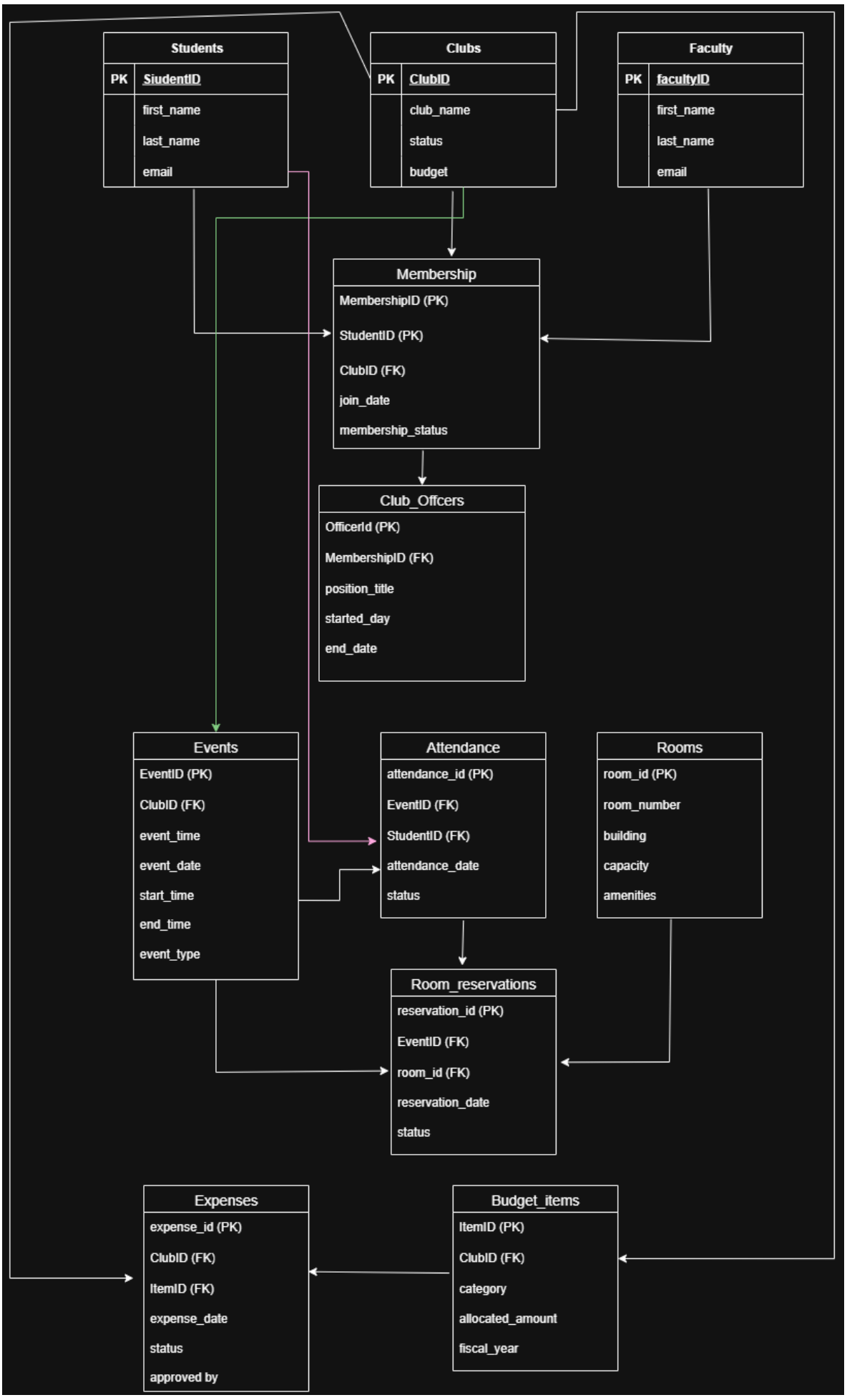
5. Transitive Dependencies: The resulting Projects table has a transitive dependency: SupervisorID->SupervisorName, SupervisorDept, but ProjectID -> SupervisorID
   Final 3NF Decomposition
   Students: StudentID (PK), StudentName, StudentMajor
   Projects: ProjectsID(PK), ProjectName, ProjectType, SupervisorID
   Supervisors: SupervisorID(PK), SupervisorName, SupervisorDept
   Student_Project: StudentID(PK, FK to Students), ProjectID(PK, FK, to Projects), Role, HoursWorked, StartDate, EndDate

Task4.2

1. {StudentID, CourseID, TimeSlot, Room}
2. StudenrID -> StudentMajor
   CourseID -> CourseName
   InstructorID -> InstructorName
   {TimeSlot, Room, Building} -> InstructorID
   {StudentID, CourseID, TimeSlot, Room} -> StudentMajor, CourseName, InstructorID, InstructorName, Building
3. StudenrID -> StudentMajor: StudentID is not a potential key, because PK contain with 4 elements
   CourseID -> CourseName: CourseID is not a potential key, because PK contain with 4 elements
   InstructorID -> InstructorName: InstructionID is not a potential key, because PK contain with 4 elements
   {TimeSlot, Room, Building} -> InstructorID: {TimeSlot, Room, Building} it is a potential key
   So, the table is not in BCNF
4. **StudentID -> StudentMajor:**
   Create a table Students: StudentID(PK), StudentMajor
   Delete attribute StudentMajor from source table, leave StudentID as a FK
   CourseID -> CourseName
   Create a table Courses: CourseID(PK), CourseName
   Delete attribute CourseName from source table and leave CourseID
   InstructorID -> InstructorName
   Create a table Instructors: InstructorID (PK), InstructorName
   Delete attribute InstructorName from source table and leave InstructorID
   Room->Building
   Create a table Rooms: Room(PK), Building
   Delete attribute Building from the source table
   The final BCNF diagram:
   Students(StudentID, StudentMajor)
   Courses(CourseID, CourseName)
   Instructors(InstructorID, InstructorName)
   Rooms(Room, Building)
   CourseShedule(StudentID(PK, FK to Students), CourseID(PK, FK to Courses), TimeSlot(PK), Room(PK, FK to Rooms), InstructorID(FK to Instructors))

Task5.1

1.


Students
PK
SiudentID
first_name
last_name
email
Clubs
PK
ClubID
club_name
status
budget
Faculty
PK
facultyID
first_name
last_name
email
Membership
MembershipID (PK)
StudentID (PK)
ClubID (FK)
join_date
membership_status
Club_Offcers
OfficerId (PK)
MembershipID (FK)
position_title
started_day
end_date
Events
EventID (PK)
ClubID (FK)
event_time
event_date
start_time
end_time
event_type
Attendance
attendance_id (PK)
EventID (FK)
StudentID (FK)
attendance_date
status
Rooms
room_id (PK)
room_number
building
capacity
amenities
Room_reservations
reservation_id (PK)
EventID (FK)
room_id (FK)
reservation_date
status
Expenses
expense_id (PK)
ClubID (FK)
ItemID (FK)
expense_date
status
approved by
Budget_items
ItemID (PK)
ClubID (FK)
category
allocated_amount
fiscal_year

2. **Students**
   StudentID (PK, INT)
   first_name (VARCHAR)
   last_name (VARCHAR)
   email (VARCHAR)
   Faculty
   FacultyID (PK, INT)
   first_name (VARCHAR)
   last_name (VARCHAR)
   email (VARCHAR)
   department (VARCHAR)
   Faculty_advisor
   faculty_advisor_id (PK)
   first_name (VARCHAR)
   last_name (VARCHAR)
   faculty_name(VARCHAR)
   Clubs
   ClubID(PK, INT)
   club_name (VARCHAR)
   status (VARCHAR)
   budget (DECIMAL)
   faculty_advisor_id (FK)
   Membership
   MembershipID (PK, INT)
   StidentID (FK)
   ClubID (FK)
   join_date (DATE)
   membership_status (VARCHAR)
   Club_Officers
   OfficerID(PK, INT)
   MembershipID(FK)
   position_title (VARCHAR)
   start_date (DATE)
   end_date (DATE)
   Events
   EventID (PK, INT)
   ClubID (FK)
   event_name (VARCHAR)
   description (TEXT)
   event_date (DATE)
   start_time (TIME)
   end_time (TIME)
   event_type (VARCHAR)
   Attendance
   AttendanceID (PK, INT)
   EventID (FK)
   StudentID (FK)
   attendance_date (DATE)
   status (VARCHAR)
   Rooms
   room_id (PK, INT)
   room_number (VARCHAR)
   building (VARCHAR)

capacity (INT)
amenities (TEXT)
Room_Reservations
reservation_id (PK, INT)
EventID (FK)
room_id (FK)
reservation_date (DATE)
status (VARCHAR)
Budget_Items
ItemID (PK, INT)
ClubID (FK)
category (VARCHAR)
allocated_amount (DECIMAL)
fiscal_year (INT)
Expenses
expense_id (PK, INT)
ClubID(FK)
ItenID (FK)
amount (DECIMAL)
expense_date (DATE)
approved_by (FK)
status (VARCHAR)

3. **How to handle club officer positions**
   Option 1: Store officer positions directly in the Club_Membership table with a "position" column
   Option 2: Create a separate Club_Officers table with foreign key to Club_Membership
   Multiple Positions: A student could hold different positions in the same club over time
   Data Integrity: Separating officer data prevents NULL values in the membership table for non-officers
   Flexibility: Allows for tracking historical officer positions and multiple concurrent officers
   Normalization: Follows better normalization practices by separating concerns
4. List all students who have attended events for both academic and sports clubs this semester.
   Show all events that are scheduled in the same room on the same day next month.
   Find the total amount spent by each club category (academic, cultural, sports) in the current fiscal year.