Database Systems — LabWork1 Solutions

Course: Database Systems — Week 2: Relational Model & Keys

Contents

Part 1 — Key Identification Exercises

Task 1.1 — Relation A: Employee(EmpID, SSN, Email, Phone, Name, Department, Salary)

Given sample rows (3 rows):

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

1. List at least 6 different superkeys

A superkey is any set of attributes that functionally determines all attributes of the relation. Possible superkeys: 1. {EmpID} 2. {SSN} 3. {Email} 4. {EmpID, Email} 5. {SSN, Phone} 6. {Email, Phone, EmpID}

(Any superset of a key is a superkey — e.g., {SSN, Name}, {EmpID, Salary}, etc.)

2. Identify all candidate keys

Candidate keys are minimal superkeys (no proper subset is also a superkey). Based on common business assumptions: - {EmpID} — typically unique employee identifier - {SSN} — Social Security Number, unique - {Email} — company email (assumed unique)

If Phone is unique per employee, {Phone} could be a candidate; but often phones can be shared, so uncertain. From the sample data each phone is unique, but business rule needs to assert uniqueness.

Candidate keys (conservative answer): {EmpID}, {SSN}, {Email}.

3. Which candidate key would you choose as primary key and why?

Choose: `EmpID` as primary key. **Reason:** - EmpID is an internal integer identifier under the organization's control (stable, short, efficient for indexing). - SSN is sensitive personal data and subject to privacy/regulatory concerns — avoid using it as PK. - Email can change (people change emails), whereas EmpID remains stable.

4. Can two employees have the same phone number? Justify using data shown.

From the *sample data* all three phone numbers are distinct, but the dataset is small. Business reasoning: - If phones are company-issued and assigned per employee → likely unique. - If phone attribute stores personal phone numbers → possible that family members share a number.

Conclusion: The sample doesn't show duplicates, but you must consult the business rule. Without explicit uniqueness rule, do not assume Phone is unique — do **not** declare Phone a candidate key unless the requirement states uniqueness.

Task 1.1 — Relation B: Registration(StudentID, CourseCode, Section, Semester, Year, Grade, Credits)

Business rules: - A student can take 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

1. Minimum attributes needed for primary key

To uniquely identify a registration record you must know which student, which course section, and which offering (semester & year). A minimal key is: - `{StudentID, CourseCode, Section, Semester, Year}`

Reason: Student + course + section + semester+year uniquely identify one registration. If `Section` is unique per course per semester (e.g., `CourseCode+Section+Semester+Year` defines a course offering), then the primary key could be `{StudentID, CourseCode, Section, Semester, Year}`. If the system has a separate `OfferingID` or `SectionID` that already encodes semester & year & section, we could use `{StudentID, OfferingID}`.

2. Why each attribute is necessary

- `StudentID` — identifies student.
- `CourseCode` & `Section` — identify which course section (multiple sections may exist).
- `Semester` & `Year` — identify the term; without them the same course+section in a different semester would conflict.

(If `Section` values are only unique per semester, you still need Semester & Year.)

3. Additional candidate keys

- If there is an `RegistrationID` surrogate key (autoincrement) — {RegistrationID}
- If the system has `OfferingID` that uniquely identifies `CourseCode+Section+Semester+Year`, then candidate key `{StudentID, OfferingID}`.

Task 1.2 — Foreign Key Design (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)

Identify foreign keys and referential directions

1. `Student.AdvisorID` → references `Professor.ProfID`. (Each student may have an advisor who is a professor.)
2. `Professor.Department` → references `Department.DeptCode`. (Professor belongs to a department.)
3. `Course.DepartmentCode` → references `Department.DeptCode`. (Course belongs to a department.)
4. `Department.ChairID` → references `Professor.ProfID`. (Chair is a professor; may be NULL if no chair assigned.)
5. `Enrollment.StudentID` → references `Student.StudentID`.
6. `Enrollment.CourseID` → references `Course.CourseID`.

Optional/additional constraints: - If `Student.Major` stores a department code, then `Student.Major` → `Department.DeptCode` (FK to Department). - Enforce `AdvisorID` may be NULL for undeclared advisor. - Add composite FK for enrollment if `Course` is versioned by semester (e.g., Enrollment referencing `CourseOffering`).

Part 2 — ER Diagram Construction

> For both tasks below, an ER diagram should be drawn in ERDPlus or draw.io. Below I give a complete textual design and instructions for drawing.

Task 2.1 — Hospital Management System

Entities and strength

Strong entities: Patient, Doctor, Department, Appointment, Prescription, Room **Weak entity:** None strictly required — but `Room` could be considered weak relative to Department if Room numbers are only unique within department.

Entities & attributes (classification)

1. **Patient** (strong)
2. PatientID (PK)

3. FirstName, LastName (simple)
4. BirthDate (simple)
5. Address (composite: Street, City, State, Zip)
6. PhoneNumbers (multi-valued)
7. InsuranceProvider, InsuranceNumber (simple)
8. Age (derived — optional from BirthDate)
9. **Doctor** (strong)
10. DoctorID (PK)
11. Name (First, Last)
12. Specializations (multi-valued)
13. PhoneNumber (could be multi-valued)
14. OfficeLocation (simple)
15. **Department** (strong)
16. DeptCode (PK)
17. DeptName
18. Location
19. **Appointment** (strong — or could be association entity)
20. AppointmentID (PK)
21. PatientID (FK)
22. DoctorID (FK)
23. DateTime
24. Purpose (simple)
25. Notes
26. **Prescription** (strong)
27. PrescriptionID (PK)
28. PatientID (FK)
29. DoctorID (FK)
30. MedName
31. Dosage
32. Instructions
33. DateIssued
34. **Room**
35. DeptCode (FK) — part of composite key if rooms numbered per department
36. RoomNumber
37. FullRoomID (composite PK: DeptCode + RoomNumber)
38. Type (e.g., ICU, Ward)

Relationships & cardinalities

- Department 1 — * Doctor (a department has many doctors; a doctor belongs to one department). Cardinality: 1:N.
- Doctor 1 — * Appointment (doctor may have many appointments). Cardinality: 1:N.
- Patient 1 — * Appointment (patient may have many appointments). Cardinality: 1:N.
- Doctor 1 — * Prescription (1 doctor issues many prescriptions). 1:N.
- Patient 1 — * Prescription (1 patient may have many prescriptions). 1:N.
- Department 1 — * Room (rooms belong to departments). 1:N.
- Appointment M — 1 Room? If appointments reserve rooms: Appointment *—1 Room (appointment occurs in a specific room). If room optional, participation is partial.

Primary keys

- Patient.PatientID
- Doctor.DoctorID
- Department.DeptCode
- Appointment.AppointmentID
- Prescription.PrescriptionID
- Room (DeptCode + RoomNumber)

Notes about weak entities

- If Room numbers are only unique within a department, then Room is a weak entity identified by (DeptCode, RoomNumber) and owner is Department.

How to draw (ERDPlus/draw.io)

- Use rectangles for entities and underline PKs.
- Multi-valued attributes: double oval (PhoneNumbers, Specializations).
- Composite address: connect Street/City/State/Zip to Address composite.
- Relationship lines with cardinality (1:N, M:N). For M:N Appointment (if appointment has multiple doctors) use associative entity.

Task 2.2 — E-commerce Platform

Entities

- Customer (CustomerID PK, Name, Email, BillingAddress, Phone)
- Order (OrderID PK, CustomerID FK, OrderDate, TotalAmount, ShippingAddress)
- Product (ProductID PK, Name, Description, Price, InventoryLevel)
- Category (CategoryID PK, Name)
- Vendor (VendorID PK, Name, Contact)
- OrderItem (OrderID FK, ProductID FK, Quantity, PriceAtOrder) — associative entity
- Review (ReviewID PK, ProductID FK, CustomerID FK, Rating, Text, Date)
- Inventory (ProductID FK, QuantityOnHand) — could be attribute of Product

Weak entity example

- `OrderItem` is an associative entity but not weak in classical sense. A suitable weak entity is `ReviewReply` that depends on `Review` (if replies have no independent identity and use ReviewID + ReplySeq as PK).
- Another weak entity: `ProductVariant` if variants are only unique per Product (e.g., size/color); PK could be (ProductID, VariantID) and owner = Product — justification: variant cannot exist without product.

M:N relationship that needs attributes

- `Order` M:N `Product` implemented via `OrderItem` — has attributes `Quantity`, `PriceAtOrder` (these are attributes of the relationship).
- `Product` M:N `Category` (product can belong to many categories, category has many products) — if you need attributes about assignment, use associative table `ProductCategory`.

ER diagram notes

- Show `OrderItem` as relationship entity connecting Order and Product; underline composite PK (OrderID, ProductID) or introduce `OrderItemID` as surrogate.
- Reviews link Customer and Product (1 Customer may write many Reviews; 1 Product has many Reviews): 1:N both ways through Review entity.

Part 4 — Normalization Workshop

Task 4.1 — StudentProject(StudentID, StudentName, StudentMajor, ProjectID, ProjectTitle, ProjectType, SupervisorID, SupervisorName, SupervisorDept, Role, HoursWorked, StartDate, EndDate)

1. Functional dependencies (FDs)

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

So list FDs: - StudentID → StudentName, StudentMajor - ProjectID → ProjectTitle, ProjectType, SupervisorID, StartDate, EndDate - SupervisorID → SupervisorName, SupervisorDept - StudentID, ProjectID → Role, HoursWorked

2. Redundancy & anomalies

Redundancy: SupervisorName and SupervisorDept repeated for every student working on the same project; StudentName repeated for each of the student's projects; ProjectTitle repeated for each student on that project.

Update anomaly: If SupervisorName changes, many rows must be updated. **Insert anomaly:** To add a new Supervisor not yet supervising a project, there's no place to store them unless a project exists. **Delete anomaly:** If last student on a project leaves and you delete row, you lose Project info and Supervisor info.

3. Apply 1NF

No repeating groups shown; attributes are atomic — assume 1NF holds. If `Role` could be multi-valued per student per project, you'd need to normalize.

4. Apply 2NF

Primary key of the table is composite: (StudentID, ProjectID) because a student can appear on multiple projects and a project has many students. Partial dependencies (where part of the key determines non-key attributes): - StudentID → StudentName, StudentMajor (partial dependency) - ProjectID → ProjectTitle, ProjectType, SupervisorID, StartDate, EndDate (partial)

Decompose into 2NF: - Student(StudentID PK, StudentName, StudentMajor) - Project(ProjectID PK, ProjectTitle, ProjectType, SupervisorID, StartDate, EndDate) - Superisor(SupervisorID PK, SupervisorName, SupervisorDept) - StudentProject(StudentID PK, ProjectID PK, Role, HoursWorked)

5. Apply 3NF

Check transitive dependencies: Project.SupervisorID → SupervisorName (so in Project table we have SupervisorID; SupervisorName should be in Supervisor table — already separated). After above decomposition, transitive dependencies removed. Final 3NF schemas (same as 2NF decomposition above): - Student(StudentID, StudentName, StudentMajor) - Project(ProjectID, ProjectTitle, ProjectType, SupervisorID, StartDate, EndDate) - Supervisor(SupervisorID, SupervisorName, SupervisorDept) - StudentProject(StudentID, ProjectID, Role, HoursWorked)

Foreign keys: - Project.SupervisorID → Supervisor.SupervisorID - StudentProject.StudentID → Student.StudentID - StudentProject.ProjectID → Project.ProjectID

Task 4.2 — CourseSchedule(StudentID, StudentMajor, CourseID, CourseName, InstructorID, InstructorName, TimeSlot, Room, Building)

Business rules restated: - 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 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

1. Determine the primary key (tricky)

We need to uniquely identify a student's enrollment in a particular course section (offering). Candidate key is composite: (StudentID, CourseID, TimeSlot) or (StudentID, CourseID, InstructorID, TimeSlot). But more canonical: - Each course section is uniquely identified by (CourseID, TimeSlot, Room) or by a `SectionID`. If we assume (CourseID, TimeSlot, Room) uniquely identifies a section, then the primary

key of CourseSchedule is (StudentID, CourseID, TimeSlot, Room). However since Room determines Building and TimeSlot+Room together define a unique offering, you can simplify.

Choose primary key: `(StudentID, CourseID, TimeSlot, Room)` — this identifies one student's enrollment in a specific course section occurrence.

2. List all functional dependencies

Using given rules: - StudentID → StudentMajor - CourseID → CourseName - InstructorID → InstructorName - (TimeSlot, Room) → Building (given) - (CourseID, TimeSlot, Room) → InstructorID (because a course section at that time/room has one instructor) - Composite key (StudentID, CourseID, TimeSlot, Room) → (all remaining attributes like InstructorID, Room, Building, InstructorName)

3. Check BCNF

BCNF requires that for every FD X → Y, X is a superkey. - StudentID → StudentMajor: `StudentID` is not a superkey → violates BCNF. - CourseID → CourseName: `CourseID` is not a superkey → violates BCNF. - InstructorID → InstructorName: also violates BCNF. - (TimeSlot, Room) → Building: (TimeSlot, Room) is not a superkey → violates BCNF.

So table is **not in BCNF**.

4. Decompose to BCNF

Stepwise decomposition: 1. From StudentID → StudentMajor, separate Student: - Student(StudentID PK, StudentMajor) - Remaining: CourseSchedule1(StudentID, CourseID, CourseName, InstructorID, InstructorName, TimeSlot, Room, Building) 2. From CourseID → CourseName, separate Course: - Course(CourseID PK, CourseName) - Remaining: CourseSchedule2(StudentID, CourseID, InstructorID, InstructorName, TimeSlot, Room, Building) 3. From InstructorID → InstructorName, separate Instructor: - Instructor(InstructorID PK, InstructorName) - Remaining: CourseSchedule3(StudentID, CourseID, InstructorID, TimeSlot, Room, Building) 4. From (TimeSlot, Room) → Building, separate RoomTime: - RoomSlot(TimeSlot, Room, Building) with PK (TimeSlot, Room) - Remaining: Enrollment(StudentID, CourseID, InstructorID, TimeSlot, Room)

Now ensure keys and FKs: - Enrollment PK: (StudentID, CourseID, TimeSlot, Room) - Enrollment.InstructorID → Instructor - Enrollment.StudentID → Student - Enrollment.CourseID → Course - RoomSlot(TimeSlot, Room) → Building

This decomposition is in BCNF because each dependency left side is a key in its relation.

5. Potential loss of information

This decomposition is lossless if FKs are enforced and original join constraints hold. There is **no loss** provided referential integrity: Enrollment references Course, Student, Instructor, and RoomSlot. However if the original relation encoded inconsistent FDs, decomposition may expose inconsistent rows (e.g., same CourseID with different CourseName) — but that indicates prior data anomalies.

Part 5 — Design Challenge: University Clubs

Requirements recap

Students join clubs (many-to-many), club events & attendance, officers (role per student per club), faculty advisor (one per club), room reservations, club budget/expenses.

ER design (textual)

Entities: - Student(StudentID PK, Name, Email, Major) - Club(ClubID PK, ClubName, Description, Budget, AdvisorID FK) - Faculty(FacultyID PK, Name, Department, Email) - Membership(StudentID FK, ClubID FK, JoinDate, Role (nullable), IsOfficer boolean) — associative - Event(EventID PK, ClubID FK, Title, StartDateTime, EndDateTime, RoomID FK, Description) - Attendance(EventID FK, StudentID FK, Attended boolean, Timestamp) - Room(RoomID PK, Building, RoomNumber, Capacity) - Expense(ExpenseID PK, ClubID FK, Amount, Date, Category, Description)

Relationships & cardinalities: - Student M:N Club → Membership (with attributes: Role, JoinDate) - Club 1 — 1 Faculty (Advisor) (each club has one advisor; a faculty can advise many clubs): 1:N from Faculty to Club - Club 1 — * Event (a club has many events) - Event M:N Student → Attendance (students attend events) — represented by Attendance with (EventID, StudentID) - Club 1 — * Expense - Event * — 1 Room (event is scheduled in a room)

Weak entity example: Membership could be considered weak because it depends on Student + Club; but it's an associative entity rather than owner/weak context.

Normalized relational schema (3NF)

- `Student(StudentID PK, Name, Email, Major)`
- `Faculty(FacultyID PK, Name, Dept, Email)`
- `Club(ClubID PK, ClubName, Description, Budget, AdvisorID FK -> Faculty)`
- `Membership(StudentID FK, ClubID FK, JoinDate, Role, IsOfficer, PRIMARY KEY(StudentID, ClubID))`
- `Event(EventID PK, ClubID FK, Title, StartDateTime, EndDateTime, RoomID FK, Description)`
- `Attendance(EventID FK, StudentID FK, Timestamp, Attended, PRIMARY KEY(EventID, StudentID))`
- `Room(RoomID PK, Building, RoomNumber, Capacity)`
- `Expense(ExpenseID PK, ClubID FK, Amount, Date, Category, Description)`

Design decision example

Decision: Use composite PK for `Membership(StudentID, ClubID)` vs surrogate `MembershipID`. **Choice & reason:** Use composite PK because Membership is identified by Student+Club pairing (natural key) and it avoids extra surrogate column. If we expect to attach many history rows per membership (versions), surrogate might be better.

Example queries (English)

1. "Find all students who are officers in the Computer Science Club." — Look up `Membership` for ClubName = 'Computer Science' where `IsOfficer = true`.
2. "List all events scheduled for next week with their room reservations." — Query `Event` where `StartDateTime` between next week's range; join `Room`.
3. "Calculate remaining budget for each club after expenses this semester." — For each club, compute `Budget - SUM(Expense.Amount WHERE Date in semester)`.

Relational Schemas & Example SQL `CREATE TABLE` statements

(Shown for the Club system as an example; other schemas analogous.)

```sql
CREATE TABLE Faculty (
  FacultyID INT PRIMARY KEY,
  Name VARCHAR(100) NOT NULL,
  Department VARCHAR(50),
  Email VARCHAR(100)
);

CREATE TABLE Student (
  StudentID INT PRIMARY KEY,
  Name VARCHAR(100),
  Email VARCHAR(100) UNIQUE,
  Major VARCHAR(50)
);

CREATE TABLE Club (
  ClubID INT PRIMARY KEY,
  ClubName VARCHAR(100) NOT NULL,
  Description TEXT,
  Budget DECIMAL(12,2),
  AdvisorID INT,
  FOREIGN KEY (AdvisorID) REFERENCES Faculty(FacultyID)
);

CREATE TABLE Membership (
  StudentID INT,
  ClubID INT,
  JoinDate DATE,
  Role VARCHAR(50),
  IsOfficer BOOLEAN DEFAULT FALSE,
  PRIMARY KEY (StudentID, ClubID),
  FOREIGN KEY (StudentID) REFERENCES Student(StudentID),
  FOREIGN KEY (ClubID) REFERENCES Club(ClubID)
);
```

```
CREATE TABLE Room (
  RoomID INT PRIMARY KEY,
  Building VARCHAR(50),
  RoomNumber VARCHAR(20),
  Capacity INT
);

CREATE TABLE Event (
  EventID INT PRIMARY KEY,
  ClubID INT,
  Title VARCHAR(200),
  StartDateTime DATETIME,
  EndDateTime DATETIME,
  RoomID INT,
  Description TEXT,
  FOREIGN KEY (ClubID) REFERENCES Club(ClubID),
  FOREIGN KEY (RoomID) REFERENCES Room(RoomID)
);

CREATE TABLE Attendance (
  EventID INT,
  StudentID INT,
  Attended BOOLEAN,
  Timestamp DATETIME,
  PRIMARY KEY (EventID, StudentID),
  FOREIGN KEY (EventID) REFERENCES Event(EventID),
  FOREIGN KEY (StudentID) REFERENCES Student(StudentID)
);

CREATE TABLE Expense (
  ExpenseID INT PRIMARY KEY,
  ClubID INT,
  Amount DECIMAL(12,2),
  Date DATE,
  Category VARCHAR(50),
  Description TEXT,
  FOREIGN KEY (ClubID) REFERENCES Club(ClubID)
);
```

Drawing tips & submission checklist

1. Underline PKs on your ER diagrams.
2. Clearly mark FK relationships and cardinalities (1:1, 1:N, M:N).
3. For M:N relationships, show associative entity with relationship attributes.
4. For normalization steps include intermediate relations and show elimination of partial/transitive dependencies.
5. Save diagrams as PNG or include exported PDF pages in final submission.

If you want, I can: - Export this document as a PDF for direct submission, - Generate clean ER diagrams (draw.io or ERDPlus) and attach them, - Produce `CREATE TABLE` statements for all relations in the lab (not just clubs), - Or walk through any specific task line-by-line with more explanation.

Tell me which next step you'd like and I will continue immediately.