Shiraz University

Advanced Database Systems

Assignment 1

Database Design for a Banking System

Instructor:
Professor Sadreddini

Deadline:
17 Aban 1404, at 11:59 PM (Tehran Time)

Part 1: Requirements Analysis and Relational Algebra

Learning Objectives

Upon completion of this section, you should be able to:

- Understand information requirements from a banking system scenario.
- Translate these requirements into formal **Relational Algebra** expressions.
- Deduce a simple relational schema from a given short scenario.
- Apply Relational Algebra both to a pre-defined schema and to a schema designed by yourself.

Scenario: Banking Management System (Recap and Pre-defined Schema)

You are tasked with designing the core database for a bank. The system needs to manage customer information, bank accounts, their loans, branches, employees, and transactions.

Below is the extended relational database schema for our banking system. All Relational Algebra queries in **Section 1.1** must be written based on this structure:

```
Customers(customer_id, name, address, ssn)
Accounts(account_number, account_type, balance, open_date, branch_id)
Loans(loan_id, loan_type, amount, interest_rate, start_date, customer_id,
branch_id)
Customer_Accounts(customer_id, account_number)
Branch(branch_id, name, city)
Employee(emp_id, name, role, hire_date, branch_id)
Transactions(transaction_id, transaction_type, amount, transaction_date,
       source_account_number, destination_account_number)
```

Tasks

Section 1.1: Relational Algebra Queries

Write the equivalent **Relational Algebra expression** for each requirement. Use the exact table and column names defined above.

Q1.1-1. Find the names of all customers who have a `Savings` account.

Q1.1-2. List the ID and amount of all `Mortgage` loans with an amount greater than `500,000,000` Toman.

Q1.1-3. Find the SSN of all customers who have both a `Checking` account and an `Auto` loan.

Q1.1-4. List the names of customers who own **more than three** accounts.

Q1.1-5. Find the names of customers who have an account in all branches located in 'Tehran'.

Q1.1-6. Find the SSNs of customers who have a `Checking` account *and* at least one loan with `interest_rate` $> 18\%$.

Q1.1-7. List the names of all employees who work in a branch located in the city 'Shiraz'.

Q1.1-8. List accounts that have at least one **TRANSFER** transaction with `amount` $> 100,000,000$.

Section 1.2: Relational Schema Design and Relational Algebra

New Scenario: Employee and Project Management System
A software company plans to build a system to manage its employees and projects.

- Each **Employee** has a unique employee ID, name, position, and hire date.
- Each **Project** has a unique project ID, project name, and budget.
- Each employee can work on **one or more projects**, and each project can involve **one or more employees** (a many-to-many relationship).

Note: The relationship between `Employee` and `Project` is many-to-many (M:N). Be sure to include an intermediate table (e.g., `Employee_Project`) to model this relationship properly.

Your Tasks:

Q1.2-1. **Relational Schema Design:** Based on the "Employee and Project Management" scenario, design a logical relational database schema in the same format as the banking schema above (e.g., `TableName(pk_col, col2, ...)`). Clearly underline the primary keys in your schema as shown in the banking example.

Q1.2-2. **Relational Algebra for Your Designed Schema:** Assuming the schema you designed above exists, write the Relational Algebra expressions for the following queries:

- **Q1.2-2.1:** Find the names of all employees who work on a project named `'Mobile App Development'`.
- **Q1.2-2.2:** List the names of all projects that involve an employee with the position `'Senior Developer'`.

Submission Format for Part 1

Your submission must include:

- Relational Algebra expressions for all queries (Q1.1-1 to Q1.1-8).
- Your designed schema for the Employee–Project scenario.
- Relational Algebra answers for Q1.2-2.1 and Q1.2-2.2.

Part 2: Conceptual Design (ERD)

Scenario

Extend your ERD to cover the following requirements:

- **Branch**: Each branch has `branch_id`, `name`, `city`. A branch manages many accounts and loans.
- **Customer – Account (M:N)**: A customer can hold multiple accounts; an account can be joint (shared by multiple customers).
- **Loan**: Each loan belongs to exactly one customer and is issued/managed by a single branch.
- **Employee**: Each employee works at exactly one branch and can be involved in opening accounts or managing loans.
- **Transaction**: For each account, deposits, withdrawals, and transfers are recorded with `tx_id`, `tx_type`, `amount`, `tx_date`, and optional `dst_account` for transfers.

Note: Model the joint-account relationship (M:N) using a linking entity/table. Clearly indicate cardinalities and primary keys in your ERD.

Task 2.1: Entity-Relationship (ER) Diagram

Design a complete ER diagram for the banking system. Your diagram must clearly show:

- All required entities (`Customer`, `Account`, `Loan`, `Branch`, `Employee`, `Transaction`).
- All attributes for each entity, with primary keys clearly marked.
- The relationships between the entities and their cardinalities.
- Proper modeling of the joint-account (M:N) relationship.

Note: You do not need to convert your ERD into a relational schema for this part.

Submission Format for Part 2

Your submission must include:

- A clear and readable image of your final ERD.
- Proper representation of all relationships with correct cardinalities (1:N, M:N, etc.).

Part 3: SQL Implementation (DDL)

Advanced SQL Requirements

Your `schema.sql` must include:

- All primary keys, foreign keys, and NOT NULL / UNIQUE constraints.
- **CHECK** constraints:
  - `Transactions.amount > 0`
  - `Transactions.tx_type` $\in$ {`'DEPOSIT'`, `'WITHDRAW'`, `'TRANSFER'`}
  - If `tx_type = 'TRANSFER'`, then `dst_account IS NOT NULL`
- **DEFAULT** values (e.g., `open_date` and `tx_date` default to `CURRENT_DATE`).

- Two **VIEW**s:
  - `branch_account_summary(branch_id, total_accounts, total_balance)`
  - `high_risk_loans(loan_id, customer_id, interest_rate, branch_id)`

Implementation Guide for Part 3

Follow these steps to successfully complete the SQL implementation part of this assignment:

1. **Install PostgreSQL:** Download and install PostgreSQL version 15 or higher from `https://www.postgresql.org/download/`. You may use the command-line tool `psql` or a graphical client such as **PgAdmin** or **DBeaver**.
2. **Create a New Database:** After installation, create a new database (e.g., `assignment1`) where you will execute your SQL script.
3. **Define Tables (DDL):** In a file named `schema.sql`, define all tables described in the banking system schema using `CREATE TABLE`. - Include **PRIMARY KEY** and **FOREIGN KEY** constraints. - Add **NOT NULL**, **UNIQUE**, and **CHECK** constraints. - Use **DEFAULT** values (e.g., `CURRENT_DATE` for `open_date` and `tx_date`).
4. **Model M:N Relationships:** Implement the many-to-many relationship between customers and accounts using a linking table (i.e., an intermediate table that connects the two entities and stores their foreign keys — for example, `Customer_Accounts`).
5. **Add Advanced Constraints:** - Ensure `Transactions.amount > 0`. - Restrict `Transactions.tx_type` to {`'DEPOSIT'`, `'WITHDRAW'`, `'TRANSFER'`}. - Require `dst_account` to be NOT NULL when `tx_type = 'TRANSFER'`.
6. **Create Views:** Define the following views with `CREATE VIEW`:

```
branch_account_summary(
branch_id,
total_accounts,
total_balance
)

high_risk_loans(
loan_id,
customer_id,
interest_rate,
branch_id
)
```

7. **Test Your Script:** Run the script in PostgreSQL:

```
psql -U your_username -d assignment1 -f schema.sql
```

   Or paste the script into the query editor of PgAdmin or DBeaver and execute it.

8. **Submit Your Work:** Upload your final `schema.sql` file containing all table definitions, constraints, views.

Tip: You do not need to insert any sample data into the database. This part focuses entirely on **database structure design and implementation**.

Submission Format for Part 3

Your submission must include:

- A single runnable `schema.sql` file.
- All tables with correct constraints, data types, and keys.
- The required views implementation.

Verification Tasks (Mandatory)

After you create and run your `schema.sql` script in PostgreSQL, you must verify that your database schema and objects have been correctly created. Run the following commands in the `psql` command-line interface and take clear screenshots of their output.

1. `\dt` — List all available tables in the current database.
2. `\d` — See all tables and their structure.
3. `\d <table_name>` — View the detailed structure (columns, types, constraints) of a specific table. Replace `<table_name>` with the actual name of each table you created.
4. `\dv` — List all available views in the current database.

Submission Requirement: Your submission must include:

- A single runnable `schema.sql` file.
- All tables with correct constraints, data types, and keys.
- Screenshots of the output of each verification command listed above (`\dt`, `\d`, `\d <table_name>`, `\dv`).

Part 4: Normalization Analysis

Submission Format for Part 4

Your submission must include an analysis **only for the tables `Accounts` and `Loans` from Part 1.**

For **each** of these two tables, include:

- A clear list of functional dependencies (FDs).
- Identification of candidate keys.
- Explanation showing that the table is in **3NF or BCNF**, with justification.
- A short justification of how your design avoids **insertion**, **deletion**, and **update** anomalies.

Submission Guidelines

To successfully complete and submit this assignment, please carefully follow the instructions below:

1. Your submission must include **exactly two files**:
   - A single **`report.pdf`** file containing all required materials, including:
     - All answers to the questions in Part 1 (Relational Algebra).
     - The ER diagram created in Part 2 (Conceptual Design).
     - The logical relational schema derived from your ERD (if applicable).
     - SQL code snippets and verification screenshots required in Part 3.
     - The normalization analysis for the specified tables in Part 4.
     - All screenshots of the verification commands (`\dt`, `\d`, `\d <table_name>`, `\dv`).
     - Any photos of handwritten relational algebra solutions (if you choose to write them by hand).
   - A runnable **`schema.sql`** file that:
     - Creates all tables with correct keys, constraints, and data types.
     - Includes definitions of views where required.
2. Submit a single zipped archive (`.zip` only; **not** `.rar`) by email to: `adb.graders@gmail.com`. Use the following exact format:
   - **Archive Name:** `ADB-HW1-Jack_White-90012768.zip`
   - **Email Subject:** `Assignment 1: 90012768`

   Replace "Jack White" with your full name and "90012768" with your student number.
3. All deliverables must be included inside this single compressed archive file.
4. For questions that require writing **Relational Algebra** expressions, you are allowed to write their solutions by hand on paper and include a clear, readable photo of their handwritten work in the final `report.pdf`. This is recommended because typing relational algebra notation processors can be difficult.
5. Late submissions will **not** be accepted unless explicitly approved by the instructor.