Shiraz University

Database Systems

Assignment 2

Advanced SQL Querying, Integrity and Security (Banking & Pagila)

Instructor: Professor Sadreddini

Deadline:
03 Azar 1404, at 11:59 PM (Tehran Time)

Overview

This assignment has **two data contexts**:

- **Part 1** uses the *Banking Management System* schema from Assignment 1.
- **Parts 2–6** use the real-world *Pagila* sample database (PostgreSQL port of Sakila).

You will practice SQL querying (JOINs, subqueries, aggregation) and apply *Integrity* and *Security* concepts (constraints, GRANT/REVOKE).

Deliverables. Submit exactly two files inside one .zip:

- report.pdf: all answers, including screenshots and SQL results.
- hw2.sql: all queries/DDL, runnable on PostgreSQL 15+. Clearly separate **Part 1 (Banking)** and **Parts 2–6 (Pagila)** with comments.

Note (Two Databases). **Part 1** must be executed on your HW1 banking database (e.g., assignment1). **Parts 2–6** must be executed on pagila. When working in psql, switch connections explicitly before running each part, e.g., \c assignment1 for Part 1 and \c pagila for Parts 2–6.

Part 1: SQL Querying on the Banking Management System

Schema Reminder (from Assignment 1)

- Customers(customer_id, name, address, ssn)
- Branch(branch_id, name, city)
- 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)
- Transactions(transaction_id, transaction_type, amount, transaction_date, source_account_number, destination_account_number)

Assumptions for Transactions**.** transaction_type $\in$ {'TRANSFER', 'DEPOSIT', 'WITHDRAW'}.

Learning Objectives

- Retrieve and analyze banking data using SELECT, JOIN, WHERE.
- Write complex subqueries with aggregation and HAVING clauses.

Tasks (Banking)

B1-1. Find the names of customers who have at least one Savings account.

B1-2. List the names of customers who have both a Checking account and an Auto loan.

B1-3. Find the branch name(s) with the *highest* total loan amount issued.

B1-4. List accounts that have at least one TRANSFER transaction with amount > 100,000,000.

B1-5. List the names of customers whose *total* loan amount (sum of all their loans) is greater than the *average* loan amount of all customers.

Tools & Setup for Pagila (Step by Step)

Prerequisites

- PostgreSQL 15+ installed (psql available in PATH).
- The Pagila scripts: pagila-schema.sql and pagila-data.sql.
- **Note:** These files are provided in the GitHub repository alongside this assignment.

Install Pagila

1. **Create the database:**

   ```
   createdb pagila
   ```

2. **Load schema:**

   ```
   psql -d pagila -f pagila-schema.sql
   ```

3. **Load data:**

   ```
   psql -d pagila -f pagila-data.sql
   ```

4. **Verify in psql:**

   ```
   psql -d pagila
   \dt
   \dv
   \d customer
   \q
   ```

Part 2: BASIC SQL QUERYING (PAGILA)

Learning Objectives

- Use selection, projection, and joins for retrieval.

Task

- **P2-1:** Find the names of customers who have rented at least one film with rating 'PG-13'. (Tables: customer, rental, inventory, film)

Part 3: Complex Queries with JOINs and Subqueries (Pagila)

Learning Objectives

- Practice nested subqueries and anti-joins (NOT EXISTS).
- Combine multiple tables to answer advanced questions.

Tasks

- **P3-1:** Find customers (full name) who have *never* rented a film from the store located in 'Woodridge'.
- **P3-2:** Find films whose rental_rate is strictly greater than the *overall average* rental_rate. (Table: film)

Part 4: Aggregation and Views (Pagila)

Learning Objectives

- Use aggregates (SUM, COUNT, AVG) with GROUP BY.
- Create views for frequently used summaries.

Task

- **P4-1:** For each store, report total number of rentals and total amount of payments. Then create a view store_kpi(store_id, rentals_count, total_amount) encapsulating this summary. (Tables: store, inventory, rental, payment)

Part 5: Integrity Constraints (Pagila)

Learning Objectives

- Apply CHECK, UNIQUE, and FOREIGN KEY constraints.
- Reason about business rules via constraints.

Task

- **P5-1:** Your task is to ensure the following constraints exist in the pagila database. In your hw2.sql file, write the ALTER TABLE commands to add them. (You may check if they exist first using \d <table> to avoid errors).
  - **(i)** payment.amount must be non-negative.
  - **(ii)** customer.email must be unique across the table.
  - **(iii)** rental.inventory_id must correctly reference inventory(inventory_id).

Part 6: Security and Authorization (Pagila)

Learning Objectives

- Use GRANT and REVOKE to control access.
- Test and verify the application of security roles.

Task

- **P6-1:** First, create two new users (roles) for testing (you may need superuser privileges for this):

```
CREATE USER student1 WITH PASSWORD 'pass123';
CREATE USER student2 WITH PASSWORD 'pass123';
```

  Next, create a limited view limited_customers(name, city) by joining customer with address/city. Then, GRANT SELECT on this view to user student1.
  Finally, **verify your security rule**: run the following commands in psql, include screenshots of the output in your report, and explain *why* one command succeeds and the other fails.

```
SET ROLE student1;
SELECT * FROM limited_customers;  -- This should succeed.
SELECT * FROM customer;           -- This should fail.
RESET ROLE;
```

Part 7: Triggers (Banking & Pagila)

Learning Objectives

- Understand how triggers enforce business rules automatically.
- Implement both **BEFORE** and **AFTER** triggers for validation and automatic updates.
- Use OLD and NEW references effectively within trigger functions.

Basic Tasks

T7-B-1. **(Pagila; BEFORE UPDATE)** On customer, create a **BEFORE UPDATE** trigger that converts any blank email (”) to NULL.

T7-B-2. **(Pagila; BEFORE INSERT OR UPDATE)** On payment, create a trigger that rejects any transaction with a negative amount. You can raise an exception in PostgreSQL using RAISE EXCEPTION ’Invalid amount’;.

Triggers on the Banking System

T7-C-1. **(Banking; AFTER UPDATE — Overdraft-as-Loan Policy)** Extend the banking system to automatically handle overdrafts. Whenever an account balance becomes negative after an update, the system should:

T7-C-1.1. set the account's balance to 0,

T7-C-1.2. insert a new record in the Loans table for the overdraft amount (absolute value of the negative balance),

T7-C-1.3. assign the new loan a loan_type = 'Overdraft' and link it to the same branch and account.

Use an **AFTER UPDATE** trigger on Accounts. *Hint:* If an account has multiple owners, assume one of them (e.g., the smallest customer_id) will be assigned as the loan owner.

T7-C-2. **(Pagila; BEFORE INSERT — Prevent Double Rental)** On rental, create a **BEFORE INSERT** trigger that checks if another active rental already exists for the same inventory_id (return_date IS NULL). If such a record exists, the trigger should raise an exception and prevent the insertion. This ensures that no film copy is rented out to more than one customer at the same time.

Submission Notes for Part 7

- Include all your CREATE FUNCTION and CREATE TRIGGER statements in hw2.sql.
- In your report.pdf, provide short test scenarios (including screenshots) showing the trigger behavior.
- For each trigger, briefly explain why it is the best method to enforce the intended business rule.

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 SQL query answers and results/screenshots for Part 1 (Banking).
     - The SQL query answer and result/screenshot for Part 2 (RA to SQL).
     - All SQL query answers and results/screenshots for Part 3 (JOINs/Subqueries).
     - The SQL for your view and its verification for Part 4 (Aggregation).
     - Your ALTER TABLE commands and analysis for Part 5 (Integrity).
     - The SQL commands, test screenshots, and explanation for Part 6 (Security).
     - For Part 7 (Triggers): the CREATE FUNCTION / CREATE TRIGGER code, plus **s**hort test scenarios with screenshots showing the trigger behavior.
     - All verification screenshots (e.g., \dt, \dv, \d customer).
   - A runnable hw2.sql file that:
     - Contains all SQL queries, DDL, and DCL commands from Part 1 through Part 7.
     - Includes definitions of views and users where required.
     - Is clearly commented, separating each Part.

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-HW2-Jack_White-90012768.zip
   - **Email Subject:** Assignment 2: 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. Late submissions will **not** be accepted unless explicitly approved by the instructor.