DTSC660: Data and Database Management with SQL
Module 6
Assignment 4

Purpose

For this assignment, you will utilize the skills acquired in Modules 5 and 6 to design a database and write a series of complex queries. You will be responsible for testing your code before submission.

Part 1: Database Design

- You have been asked to create a banking database for Eastern Banking Holdings. In this part, your objective is to create a comprehensive DDL for the banking database. You will employ methods such as defining primary and foreign keys, implementing various constraints, and ensuring data integrity.

Part 2: Queries Using the Banking Database

- In this part, you will populate your banking database with the provided data from the Assignment 4 folder in Brightspace. Once the database is populated, you will be tasked with writing a series of complex queries. These queries will involve complex techniques such as JOINs, SET operators, and Subqueries.

Part 3: Advanced Queries with the University Database

- For the final part of the assignment, you will utilize the university database from Module 5. Your objective is to write another series of advanced queries, incorporating the concepts and techniques covered in Modules 5.

Submission

You will submit a total of **3** sql files to CodeGrade. Files should be named appropriately and be in .sql format. Each file must use the postgres standards taught in the course. Use of other SQL languages such as T-SQL will result in an automatic 0 for the assignment. **Ensure your files run in their entirety in pgAdmin. This means ensuring each query ends in a semicolon (;). Graders will not attempt to correct or interpret malformed SQL queries.** You will have **one submission attempt** for this assignment. It is expected that if you have questions or difficulties with any portion of this assignment that you utilize the assignment discussion board or email the GAs to gain clarity (dtsc_ga_660@eastern.edu)

- ***File 1***: You must submit a SQL document called <LastName>_Assignment4_Part1. This document must include ALL commands required to create the banking database (the DDL).

- ***File 2:*** You must submit a SQL document called <LastName>_Assignment4_Part2. This document must include Queries 1- 7 found in part 2 of this assignment.

- ***File 3:*** You must submit a SQL document called <LastName>_Assignment4_Part3. This document must include Queries 8-12 found in part 3 of this assignment.

- You will submit all files to the Assignment 4 Submission link to CodeGrade. Failure to include all three files will result in loss of credit for any missing information as specified in the rubric.

PART 1 Banking Database Design

Question 1:

Consider the bank database schema given below. Write the SQL DDL corresponding to this schema (i.e. the CREATE TABLE statements). When you are writing the DDL, make sure you do the following:

a. Create the tables in the order specified in the schema.
b. Use the exact names given for table names and attributes.
c. Do not add or remove tables or attributes.
d. For each table, assumptions have been provided regarding the attributes and relationships between them. The purpose of your DDL is to meet the outlined requirements by utilizing constraints and keys for data validation. By defining the tables, specifying appropriate data types, and enforcing constraints and keys (both primary and foreign), you should ensure that these assumptions hold true..
e. In this database, **only the data types varchar(40) and money will be used**. Please ensure that attributes are defined with only these data types for consistency.
f. By the time you have completed you will have at least one of each of the following:
    i. CHECK constraint
    ii. NOT NULL constraint
    iii. ON UPDATE CASCADE ON DELETE CASCADE
    iv. DEFAULT value constraint

Database Schema:

branch (branch_name, branch_city, assets)
customer (cust_ID, customer_name, customer_street, customer_city)
*loan (loan_number, **branch_name**, amount)*
*borrower (**cust_ID**, **loan_number**)*
*account (account_number, **branch_name**, balance)*
*depositor (**cust_ID**, **account_number**)*

*** Bold indicates **Foreign Key**, Underlined indicates Primary Key*

**************************ASSIGNMENT CONTINUED ON NEXT PAGE***************************

Database Assumptions:

- A customer can have multiple accounts and/or multiple loans. Likewise, an account or loan can be tied to more than one customer (think spouses or business partners).
- For this example, it is assumed that there is only one bank, and all the individual branches listed in the data are owned by this bank

Table Assumptions:

- **Branch**:
    - All branches must have assets that are monetary.
    - Asset amount cannot be negative
    - There are four and only four cities with branches: Brooklyn, Bronx, Manhattan, and Yonkers.
- **Customer**:
    - Customers must have a name.
    - Customers must have a street address (You may choose to include an additional CONSTRAINT for the city; however, please note that only the street address is a requirement.)
- **Loan**:
    - All loans must have a monetary amount.
    - The default amount for loans is zero dollars and zero cents.
    - A loan amount cannot be negative
    - If a branch closes or changes its name, these activities should be reflected in the loan table.
- **Borrower**:
    - A borrower is a type of customer, so if the cust_ID is deleted or changed, the borrower table should reflect these actions.
    - The same is true of the loan_number.
- **Account**:
    - All accounts must have a monetary amount.
    - The default amount for loans is zero dollars and zero cents.
    - If a branch closes or changes its name, these activities should be reflected in the account table.
- **Depositor**:
    - A depositor is a type of customer, so if the cust_ID is deleted or changed, the depositor table should reflect these actions.
    - The same is true of the account_number.

Note: Please remember that for this assignment, you should only use the data types **varchar(40)** and **money** for consistency purposes. While these data types might not be the absolute best choice in all cases, the goal here is to demonstrate your understanding of constraints, data validation, and keys. Data type declaration was already assessed in a prior assignment, and it is not the primary focus of this assignment. Therefore, choose the most appropriate data type between varchar(40) and money for each attribute to maintain consistency throughout the database design.

**************************ASSIGNMENT CONTINUED ON NEXT PAGE***************************

PART 2 Banking Database Queries

For the next section, you will need to use the SQL data file located in the Assignment 4 folder on Brightspace. You will use this data to populate the database you have created using your Banking DDL. If your queries do not run, you will need to go back and fix your Banking DDL. Make sure you use the correct table names and attributes. Queries that do not function correctly in the given database will result in point deductions.

Important: Note that when a query request specifies the use of a JOIN, subquery, SET operator, or a combination of these techniques, you must ensure that your query incorporates the required method at least once. However, you are not limited to using only that technique, unless otherwise specified. Feel free to employ the required technique multiple times or combine it with other techniques to achieve the desired outcome. For example, if a question mandates the use of a join, you may opt to use multiple joins, a join with a subquery, or any other valid approach as long as the minimum requirement is met.

Query 1:

Write a query to find all customers who have at least one loan and one deposit account. Include the **cust_ID**, **account_number**, and **loan_number** in your results. Note: Some customers may appear multiple times due to having multiple loans or deposit accounts. *Your solution must include a JOIN.*

Query 2:

Write a query that identifies all customers who have a deposit account in the same city in which they live. The results should include the **cust_id**, **customer_city**, **branch_city**, **branch_name**, and **account_number**. Note: The city of a deposit account is the city where its branch is located. *Your solution must use a JOIN.*

Query 3:
Write a query that returns the **cust_ID** and **customer_name** of customers who hold at least one loan with the bank, but do not have any deposit accounts. *Your solution must use a subquery and a SET operator.*

Query 4:

Write a query to obtain the **cust_ID** and **customer_name** for all customers residing on the same street and in the same city as customer '12345'. Include customer '12345' in the results. Avoid hardcoding the address for customer '12345' as their information might change. *Your solution must include a subquery.*

Query 5:

Write a query to retrieve a list of **branch_names** for every branch that has at least one customer living in 'Harrison' that has a deposit account with them. Branch names should not be duplicated. *Your solution must include a subquery and a JOIN.*

Query 6:

Write a query to return each **cust_ID** and **customer_name** who has a deposit account at *every* branch located in Brooklyn. Do not hardcode the Brooklyn branch names directly into your query as these may change over time. *Your solution must include a subquery.*

Hint: If you're finding this question challenging, think about using SET operators. If you are still having difficulty, you may find it useful to wait until you have completed Module 7 and utilize techniques learned there to complete the question.

Query 7:

Write a query to retrieve the **loan_number, customer_name,** and **branch_name** of customers who have a loan at the Yonkahs Bankahs branch and whose loan amount exceeds the average loan amount for that branch. *Your solution must include a JOIN and a subquery*.

HINT: Remember, money data types do note perform well with aggregation. You will have to CAST twice here- on both sides of the comparison.

**************************ASSIGNMENT CONTINUED ON NEXT PAGE***************************

PART 3 University Database Queries

For the next portion of the assignment, you will be working with the university database introduced in Module 5. It is crucial that you test your queries against this database using the exact column names and schema provided in Module 5. If needed, you may refer to the visual schema of this database provided in the Assignment 4 Resources folder to help you understand the relationships between the tables and the structure of the database. Queries that do not function correctly in the given database will result in point deductions.

Important: Keep in mind that when a query request specifies the use of a JOIN, subquery, SET operator, or a combination of these techniques, you must ensure that your query incorporates the required method at least once. However, you are not limited to using only that technique, unless otherwise specified. Feel free to employ the required technique multiple times or combine it with other techniques to achieve the desired outcome. For example, if a question mandates the use of a join, you may opt to use multiple joins, a join with a subquery, or any other valid approach as long as the minimum requirement is met.

Query 8:

Write a query to return an alphabetical list of **dept_names** from the department table showing all departments that are assigned to at least one instructor in the instructor table. *You must use a SET operator in your solution.*

Query 9:

Write a query that returns a list of **course_ids** from the course table for courses that do not have any prerequisites listed in the prereq table. This should be sorted from smallest to largest. *Your solution must use a SET operator.*

Query 10:

Write a query that returns an alphabetical list of **dept_names** for departments that satisfy one or more of the following conditions:

- The department has a budget less than $50,000
- The department has at least one instructor whose salary is greater than $100,000
- The department has at least one student whose total credits are equal to the highest total credits taken by any student.

Do not hardcode the maximum total credits, instead use an approach that works if this number changes.*Your solution must include at least one SET operator and at least one subquery. Do not use JOINs.*

Hint: Ensure your chosen SET operator(s) remove(s) duplicates.

Query 11:

Write a query that returns the course_id and title of courses and their prerequisites. Your output should name the returned columns: **course_id, course_name, prereq_id, prereq_name** (in that order). Only include courses that have prerequisites in the results.*Your solution must use a JOIN.*

Hint: You may need to JOIN the same table multiple times here to accomplish what you need.

Query 12:

Write a query to find the **id** of each student who has never taken a course at the university. *Your solution must use an OUTER JOIN- do not use any subqueries or set operations*.

Hint: Do **not** simply use the tot_cred field in the student table as it may not accurately indicate if a student has taken courses. Consider scenarios where students have transferred, failed courses, or are currently enrolled. Adding a student to the database who has not attended any classes can help validate your query.

******************************GRADING RUBRIC ON NEXT PAGE********************************

This assignment will be graded on the following rubric. Please see *DTSC 660 Assignment Grading* in **Module 0** for our course's guidelines on point allocation and deduction. Incorrect syntax, extraneous results, or incorrectly addressing all question requirements will result in loss of points. Graders will NOT attempt to correct malformed sql code. :

Assignment Component	**Points**
PART 1: Banking DDL	**40**
Table Creation	12 (2 Points Per Table)
Primary Keys and Foreign Keys	12 (1 Point Per Key)
Constraints	16 (1 Point Per Constraint)
PART 2: Banking Queries	**35**
Query 1	5
Query 2	5
Query 3	5
Query 4	5
Query 5	5
Query 6	5
Query 7	5
PART 3: University Queries	**25**
Query 8	5
Query 9	5
Query 10	5
Query 11	5
Query 12	5
Total	**100**