Final Project: Advanced SQL Techniques

Scenario

You have to analyse the following datasets for the city of Chicago, as available on the Chicago City data portal.

- Socioeconomic indicators in Chicago
- Chicago public schools
- Chicago crime data

Based on the information available in the different tables, you have to run specific queries using Advanced SQL techniques that generate the required result sets.

The lab will be followed by a graded quiz that will have questions on all problems in this lab. Hence, remember to take screenshots of your SQL queries and their outputs for reference.

Objectives

After completing this lab, you will be able to:

- Use joins to query data from multiple tables
- Create and query views
- Write and run stored procedures
- Use transactions

Software Used in this Lab

In this lab, you will use MySQL. MySQL is a Relational Database Management System (RDBMS) designed to efficiently store, manipulate, and retrieve data.

Mysql_learners database has been used in this lab.

Here you will be creating and inserting data into the below mentioned 3 tables

- chicago_public_schools
- chicago_socioeconomic_data
- chicago_crime

Here you will be using 3 dump files for this purpose.

chicago_public_schools

chicago_crime

chicago_socioeconomic_data

Exercise 1: Using Joins

You have been asked to produce some reports about the communities and crimes in the Chicago area. You will need to use SQL join queries to access the data stored across multiple tables.

Question 1

> Write and execute a SQL query to list the school names, community names and average attendance for communities with a hardship index of 98.

Take a screenshot showing the SQL query and its results.

Question 2

> Write and execute a SQL query to list all crimes that took place at a school. Include case number, crime type and community name.

Take a screenshot showing the SQL query and its results

Exercise 2: Creating a View

For privacy reasons, you have been asked to create a view that enables users to select just the school name and the icon fields from the CHICAGO_PUBLIC_SCHOOLS table. By providing a view, you can ensure that users cannot see the actual scores given to a school, just the icon associated with their score. You should define new names for the view columns to obscure the use of scores and icons in the original table.

Question 1

Write and execute a SQL statement to create a view showing the columns listed in the following table, with new column names as shown in the second column.

Column name in CHICAGO_PUBLIC_SCHOOLS	Column name in view
NAME_OF_SCHOOL	School_Name
Safety_Icon	Safety_Rating
Family_Involvement_Icon	Family_Rating
Environment_Icon	Environment_Rating
Instruction_Icon	Instruction_Rating
Leaders_Icon	Leaders_Rating
Teachers_Icon	Teachers_Rating

Write and execute a SQL statement that returns all of the columns from the view.

Write and execute a SQL statement that returns just the school name and leaders rating from the view.

Take a screenshot showing the last SQL query and its results.

Exercise 3: Creating a Stored Procedure

The icon fields are calculated based on the value in the corresponding score field. You need to make sure that when a score field is updated, the icon field is updated too. To do this, you will write a stored procedure that receives the school id and a leaders score as input parameters, calculates the icon setting and updates the fields appropriately.

Question 1

> Write the structure of a query to create or replace a stored procedure called UPDATE_LEADERS_SCORE that takes a in_School_ID parameter as an integer and a in_Leader_Score parameter as an integer.

Take a screenshot showing the SQL query.

Question 2

> Inside your stored procedure, write a SQL statement to update the Leaders_Score field in the CHICAGO_PUBLIC_SCHOOLS table for the school identified by in_School_ID to the value in the in_Leader_Score parameter.

Take a screenshot showing the SQL query.

Question 3

> Inside your stored procedure, write a SQL IF statement to update the Leaders_Icon field in the CHICAGO_PUBLIC_SCHOOLS table for the school identified by in_School_ID using the following information.

Score lower limit	Score upper limit	Icon
80	99	Very strong
60	79	Strong
40	59	Average

20	39	Weak
0	19	Very weak

Take a screenshot showing the SQL query.

Question 4

Run your code to create the stored procedure.
Take a screenshot showing the SQL query and its results.

Write a query to call the stored procedure, passing a valid school ID and a leader score of 50, to check that the procedure works as expected.

Exercise 4: Using Transactions

You realise that if someone calls your code with a score outside of the allowed range (0-99), then the score will be updated with the invalid data and the icon will remain at its previous value. There are various ways to avoid this problem, one of which is using a transaction.

Question 1

Update your stored procedure definition. Add a generic ELSE clause to the IF statement that rolls back the current work if the score did not fit any of the preceding categories.

Take a screenshot showing the SQL query.

Question 2

Update your stored procedure definition again. Add a statement to commit the current unit of work at the end of the procedure.

Take a screenshot showing the SQL query.

Run your code to replace the stored procedure.

Write and run one query to check that the updated stored procedure works as expected when you use a valid score of 38.

Write and run another query to check that the updated stored procedure works as expected when you use an invalid score of 101.