**ISS4014 – Database Systems and Web Integration**

**Chapter 08 – Activities and Homework**

Name: Logan Strong

Date: February 27th, 2024

**Chapter 08 REVIEW (10 points)**

1. What is the difference between a column constraint and a table constraint? Give an example of each. (2 pts)

***A column constraint only applies to a single column, while a table constraint may apply to several columns.***

1. What are “referential constraint actions”? (2 pts)

Referential constraint actions, such as ON, DELET, and CASCADE, are default actions that the DBMS takes when a DML command results in a referential integrity constraint violation

1. What is a sequence? How is it implemented in MySQL? (2 pts)

A sequence is used to assign values to a column on a table in SQL. A sequence is created by naming it, having a starting point attached to an increment value (like a for loop) and caching the sequence numbers in memory.

1. What is a trigger, and what is its purpose? (2 pts)

A trigger is ***procedural SQL code that is automatically invoked by the RDBMS upon the occurrence of a given data manipulation event. Its purpose is to automate critical actions, update table values, and work as a great tool for execution throughout the DBMS.***

1. What is a stored procedure, and why is it particularly useful? (2 pts)

A stored procedure is ***a named collection of procedural and SQL statements. They are useful because they can be used to encapsulate and represent business transactions.***

**Chapter 08 PROBLEMS (20 points)**

**Important Notes:**

1. **Problems that are in bold are suggested to be used for in-class activities and group work** as these problems present a higher level of challenge.
2. For each problem,
   1. Create a **NEW SQL SCRIPT TAB**
   2. **SAVE the script tab by naming it**: [your name] Problem [problem#]. For example: JUSTIN PROBLEM 1
   3. **Clear the OUTPUT at the bottom**.
3. After developing the correct SQL statement to properly solve the problem statement, get a screenshot that contains ALL of the following:

|  |  |
| --- | --- |
| * TAB NAME * SQL statements * RESULT GRID * OUTPUT AREA   The screenshot at the right shows an example of what you should copy & paste |  |

1. Refer to the **Chapter 07, 08 and 13 SQL Supplement.pdf** on CANVAS for sample SQL statements to use as a reference as you work on this assignment. It contains samples for CREATING TABLES, INSERTING rows, of MINUS/DIFFERENCE, TRIGGERS, and PROCEDURES.
2. ***Disable “safe mode”*** by unchecking the safe mode check box in Preferences under the MySQL Edit tab (scroll down to the bottom). For Mac users, it is under the MySQLWorkbench menu under Settings. Update the change and then click the “Reconnect to DBMS” icon so your DBMS connection gets re-established with this setting. This will allow you to UPDATE & DELETE when indicated.

**1.** (2 pts) Write the SQL statements to do the following:

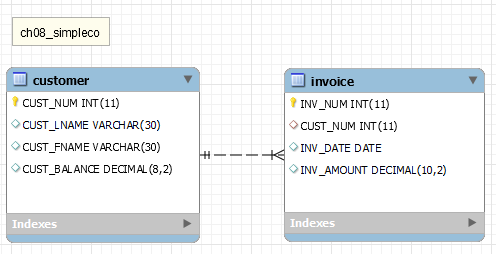
a. Drop the schema named “ch08\_simpleco” if it exists. (This will be helpful when needing to rerun the script during testing).

b. Create a database schema named “ch08\_simpleco”.

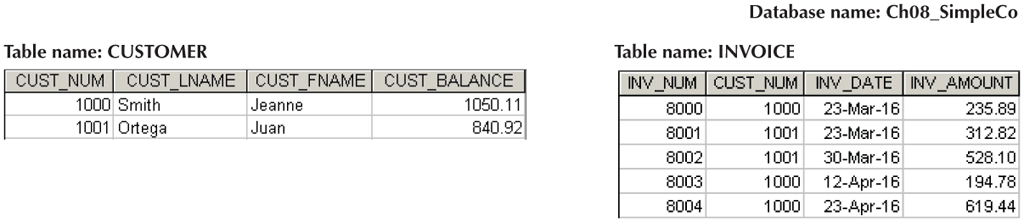
c. Use the ch08\_simpleco database.

d. Create the two tables illustrated in the ERD below. Define each attribute, primary key and foreign key constraint.

Save all the SQL statements as a script with “<your name> problem 1”. Copy and paste an image of the script after a successful run showing the SQL statements and all the log results below (no results grid will be displayed).



**2.** (2 pts) Write the SQL statements needed to insert the data into the CUSTOMER table and INVOICE table created in problem 1 based on the image below. Save all the SQL statements as a script with “<your name> problem 2”. Copy and paste an image of the script after a successful run showing the SQL statements and all the log results below (no results grid will be displayed).



**3.** (2 pts) Write the SQL statements to do the following:

a. Modify the CUSTOMER table so the customer number field is set to auto increment and the auto-increment value is set to 2000. (Note: The INVOICE table will need to be modified first to remove the foreign key constraint on the customer number. After the auto increment value property has been added to the customer number field, add the foreign key constraint back to the INVOICE table).

b. Modify the INVOICE table so the invoice number field is set to auto increment and the auto-increment value is set to 9000.

Save all the SQL statements as a script with “<your name> problem 3”. Copy and paste an image of the script after a successful run showing the SQL statements and all the log results below (no results grid will be displayed).

The MySQL command to remove a foreign key constraint has the following format:

**ALTER TABLE tbl\_name DROP FOREIGN KEY constraint\_name;**

(The constraint\_name can be found in the table properties under the foreign keys tab)

The MySQL command to modify a column to be auto increment has the following format:

**ALTER TABLE tbl\_name MODIFY COLUMN col\_name col\_type AUTO\_INCREMENT;**

The MySQL command to set an auto-increment value has the following format:

**ALTER TABLE tbl\_name AUTO\_INCREMENT = value;**

The MySQL command to add a foreign key constraint has the following format:

**ALTER TABLE tbl\_name ADD FOREIGN KEY (col\_name) REFERENCES tbl\_name(col\_name);**

**4.** (2 pts) Write the SQL statements to do the following:

Optional: you may wish to have a BEGIN statement before the following insert statements so you can ROLLBACK any changes to the database while testing the following SQL statements. (NOTE: Auto Increment values are not reset with a ROLLBACK).

a. Insert a new record into the CUSTOMER table using the auto-increment property created in problem 3. Use the following data: Last name: **Powers**, First name: **Ruth**, Customer balance: **$500.00.**

b. Write the SQL statement to display all rows and all columns of the CUSTOMER table.

c. Insert a new record into the INVOICE table using the auto-increment property created in problem 3. Use the following data: Customer number: <the auto-increment value used above>, Invoice date: **May 5, 2016**, Invoice amount: **$500.00**

d. Write the SQL statement to display all rows and all columns of the INVOICE table.

Optional: If the above insert statements did not work properly, you may execute a ROLLBACK command to reverse the changes to the database, otherwise, execute a COMMIT command to save the changes.

Save the SQL statements as a script with “<your name> problem 4”. Copy and paste an image of the script after a successful run showing the SQL statements, the result grid (for the INVOICE table), and all the log results below.

**5.** (2 pts) Write the SQL statements to do the following:

a. Modify the CUSTOMER table to include the customer’s date of birth (CUST\_DOB), which should store date data.

b. Update the CUSTOMER record for customer 1000 to have a date of birth of **March 15, 1989**.

c. Update the CUSTOMER record for customer 1001 to have a date of birth of **December 22, 1988**.

d. Write the SQL statement to display all rows and all columns of the CUSTOMER table.

Save the SQL statements as a script with “<your name> problem 5”. Copy and paste an image of the script after a successful run showing the SQL statements, the result grid and all the log results below.

**6.** (3 pts) Write the SQL statements to do the following:

a. Create a trigger named **trg\_updatecustbalance** to update the CUST\_BALANCE in the CUSTOMER table when a new invoice record is entered. (Assume that the sale is a credit sale.) Whatever value appears in the INV\_AMOUNT column of the new invoice should be added to the customer’s balance.

Optional: BEGIN statement to have the option to ROLLBACK the test if needed.

b. Test the trigger by adding the following new INVOICE record: Invoice Number: **8005**, Customer number **1001**, Invoice date: **April 27, 2018**, Invoice balance: **$225.40**

c. Select all records and rows from the CUSTOMER table.

Optional: ROLLBACK the updates if the test did not work otherwise COMMIT the changes. (NOTE: if a bug needs to be corrected in the trigger, you must delete the trigger using the DROP TRIGGER command and recreate it).

The MySQL command to create a trigger has the following format:

**delimiter //**

**CREATE TRIGGER trg\_name**

**AFTER dml\_cmd ON tbl\_name**

**FOR EACH ROW**

**BEGIN**

**sql\_statement(s)**

**END;//**

**delimiter ;**

Save the SQL statements as a script with “<your name> problem 6”. Copy and paste an image of the script after a successful run showing the SQL statements, the result grid and all the log results below.

**7.** (2 pts) Write the SQL statements to do the following:

a. Create a procedure named **prc\_cust\_add** that accepts the customer number, last name, first name and customer balance to add a new customer to the CUSTOMER table.

Optional: BEGIN statement to have the option to ROLLBACK the test if needed.

b. Test the procedure by calling it using: Customer number: **1002**, Last name: **Rauther**, First name: **Peter**, customer balance: **$0.00**, Date of birth: **January 1, 1970**.

c. Select all records and rows from the CUSTOMER table.

Optional: ROLLBACK the updates if the test did not work, otherwise COMMIT the changes. (NOTE: if a bug needs to be corrected in the trigger, you must delete the procedure using the DROP PROCEDURE command and recreate it).

The MySQL command to create a procedure has the following format:

**delimiter //**

**CREATE PROCEDURE prc\_name (arg\_name arg\_type, …)**

**BEGIN**

**sql\_statement(s)**

**END;//**

**delimiter ;**

The MySQL command to call a procedure has the following format:

**CALL prc\_name (arg\_value, …);**

Save the SQL statements as a script with “<your name> problem 7”. Copy and paste an image of the script after a successful run showing the SQL statements, the result grid and all the log results below.

**8.** (2 pts) Write the SQL statements to do the following:

a. Create a procedure named **prc\_invoice\_add** that accepts the invoice number, customer number, invoice date and invoice amount to add a new invoice to the INVOICE table.

Optional: BEGIN statement to have the option to ROLLBACK the test if needed.

b. Test the procedure by calling it using: Invoice number: **8006**, Customer number: **1000**, Invoice date: **April 30, 2018**, Invoice balance: **$301.72**.

c. Select the Customer number and Customer balance, Invoice Number and Invoice amount, from the CUSTOMER and INVOICE tables where the customer number is 1000.

Optional: ROLLBACK the updates if the test did not work, otherwise COMMIT the changes.

Save the SQL statements as a script with “<your name> problem 8”. Copy and paste an image of the script after a successful run showing the SQL statements, the result grid and all the log results below.

**9.** (3 pts) Write the SQL statements to do the following:

a. Create a trigger named **trg\_updatecustbalance2** to update the CUST\_BALANCE in the CUSTOMER table when a new invoice record is deleted. Whatever value appears in the INV\_AMOUNT column of the deleted invoice should be subtracted to the customer’s balance.

b. Create a procedure named **prc\_invoice\_delete** that accepts the invoice number and invoice deletes the invoice from the INVOICE table.

Optional: BEGIN statement to have the option to ROLLBACK the test if needed.

c. Test the procedure (and trigger) by call int **prc\_invoice\_delete** procedure to delete invoice **8005**.

d. Select the Customer number and Customer balance, Invoice Number and Invoice amount, from the CUSTOMER and INVOICE tables where the customer number is 1001.

Optional: ROLLBACK the updates if the test did not work, otherwise COMMIT the changes.

Save the SQL statements as a script with “<your name> problem 9”. Copy and paste an image of the script after a successful run showing the SQL statements, the result grid and all the log results below.