**Module/Week4:**Assignment 3

**Topic:** Incremental Refresh in Data Warehousing

**Student’s Full Name:** LIBIN NAITHELLOOR GEORGE

**Course Title:** Data Warehousing and Analytics in the Cloud

**Term name and year:** Fall 2023

**Submission Week:** [example, Week 3- Assignment 2]

**Instructor’s Name**: Dr.Nayem Rahman

**Date of Submission:** 27 July 2024

**PURPOSE:**

This assignment introduces you to the incremental refresh techniques in a data warehouse. As part of this assignment, you demonstrate your understanding of the database system, basic SQL, and incremental refresh techniques via database stored procedures. In real-world business, data warehouses are refreshed with data from operational database systems. In data warehouse refreshments, incremental techniques are used. This is a very important task that a data engineer does.

Q1.

{ A } Using the MySQL Workbench, create a database called Customer. The database must be named “Customer”.

|  |
| --- |
| CREATE DATABASE Customer; |
|  |

{ B } Check if the database was created and use the same for further questions.

|  |
| --- |
| SHOW DATABASES; |
|  |

Q2. { A } Create a staging table, \*\* Customer.CustomerChurn\_Stage \*\*, in a database system, with the column list provided in the CSV file. Define the ' CustomerId ' as the Primary Key (PK). Get the table definition (DDL) from the database system and capture it in a Word document for submission.

|  |
| --- |
| CREATE TABLE Customer.CustomerChurn\_Stage (  CustomerId INTEGER NOT NULL,  Surname VARCHAR(20) NOT NULL,  CreditScore INTEGER NOT NULL,  Geography VARCHAR(10) NOT NULL,  Gender VARCHAR(10) NOT NULL,  Age TINYINT NOT NULL,  Balance DECIMAL(10, 2) NOT NULL,  Exited BOOLEAN NOT NULL,  PRIMARY KEY(CustomerId)  );  DESCRIBE Customer.CustomerChurn\_Stage; |
|  |

{ B } Create a persistent table, \*\* Customer.CustomerChurn \*\*, with the column list provided in the CSV file + following 5 columns : << SourceSystemNm NVARCHAR(20) NOT NULL , CreateAgentId NVARCHAR(20) NOT NULL , CreateDtm DATETIME NOT NULL, ChangeAgentId NVARCHAR(20) NOT NULL , ChangeDtm DATETIME NOT NULL >> Define the ' CustomerId ' as the Primary Key (PK). Get the table definition (DDL) from the database system and capture it in a Word document for submission.

|  |
| --- |
| CREATE TABLE Customer.CustomerChurn (  CustomerId INTEGER NOT NULL,  Surname VARCHAR(20) NOT NULL,  CreditScore INTEGER NOT NULL,  Geography VARCHAR(10) NOT NULL,  Gender ENUM('Male', 'Female') NOT NULL,  Age TINYINT NOT NULL,  Balance DECIMAL(10, 2) NOT NULL,  Exited BOOLEAN NOT NULL, -- Same as TINYINT(1)  SourceSystemNm NVARCHAR(20) NOT NULL,  CreateAgentId NVARCHAR(20) NOT NULL,  CreateDtm DATETIME NOT NULL,  ChangeAgentId NVARCHAR(20) NOT NULL,  ChangeDtm DATETIME NOT NULL,  PRIMARY KEY(CustomerId)  );  DESCRIBE Customer.CustomerChurn; |
|  |

**Q3. { A } Load the staging table, \*\* Customer.CustomerChurn\_Stage \*\*, with data from the CSV file, CustomerChurn1.csv .**

|  |
| --- |
| LOAD DATA LOCAL INFILE "/Users/lngeorge/Documents/Untitled Folder/DBMS/week4/CustomerChurn1.csv"  INTO TABLE Customer.CustomerChurn\_Stage  COLUMNS TERMINATED BY ','  OPTIONALLY ENCLOSED BY '"'  ESCAPED BY '"'  LINES TERMINATED BY '\n'  IGNORE 1 LINES; |
|  |

Another option would be to use import wizard to import from CSV file.

**{ B } Verify data by comparing the row counts between the CSV file and the staging table, \*\* Customer.CustomerChurn\_Stage [Data Source: CustomerChurn1.CSV] \*\*. Provide the screenshot of last few rows using the ' SELECT \* ' . Make sure the output shows all column values. The SELECT statement must use the ORDER BY ' CustomerId '**

|  |
| --- |
| Both Table and CSV has same amount of rows (Count: 100) |
|  |
|  |
| SELECT \* FROM Customer.CustomerChurn\_Stage ORDER BY CustomerId ASC; |
|  |

**Q4. Create a database stored procedure based on the template provided along with this assignment << StoredProc\_Template.txt >>. Name the stored procedure name this: \*\* Customer.PrCustomerChurn \*\* . [[ NOTE : This stored procedure will use the table, \*\* Customer.CustomerChurn\_Stage \*\* , as the source (aka, staging table). This stored procedure will use the table, \*\* Customer.CustomerChurn \*\*, as the target (aka, persistent table). ]]**

|  |
| --- |
|  |

**Q5. Execute the stored procedure, \*\* Customer.PrCustomerChurn \*\*, that was created in Q4. After execution, the stored procedure should load data from the stage to the persistent table: \*\* Customer.CustomerChurn \*\*.**

|  |
| --- |
| CALL Customer.PrCustomerChurn(); |
|  |

**{A} Verify data by comparing the row counts between the staging table, \*\* Customer.CustomerChurn\_Stage [Data Source: CustomerChurn1.CSV] \*\* and the persistent table: \*\* Customer.CustomerChurn \*\*.**

|  |
| --- |
|  |

**{ B } Provide the screenshot of last few rows using the SELECT \*. Make sure the output shows all column values. The SELECT statement must use the ORDER BY CustomerId.**

|  |
| --- |
|  |
|  |

**Q6. After data verification is completed, in Q5 ,**

**{ A } create table, \*\*** **Customer.CustomerChurn\_Version1 \*\*, with data from \*\* Customer.CustomerChurn \*\* (that was already loaded from Customer.CustomerChurn\_Stage via the stored procedure).**

|  |
| --- |
| CREATE TABLE Customer.CustomerChurn\_Version1  AS (SELECT \* FROM Customer.CustomerChurn); |
|  |

**{ B } Show table definition of Customer.CustomerChurn\_Version1 and show the row count of the table, \*\* Customer.CustomerChurn\_Version1 \*\*:**

|  |
| --- |
| DESCRIBE Customer.CustomerChurn\_Version1; |
|  |
| SELECT COUNT(\*) FROM Customer.CustomerChurn\_Version1; |
|  |

**{ C } Provide the screenshot of last few rows for \*\* Customer.CustomerChurn\_Version1 \*\* [Originally data came from: CustomerChurn1.CSV]. Make sure the output shows all column values. The SELECT statement must use the ORDER BY CustomerId.**

|  |
| --- |
| SELECT \* FROM Customer.CustomerChurn\_Version1 ORDER BY CustomerId ASC; |
|  |

**{ D } Empty the staging table, \*\*** **Customer.CustomerChurn\_Stage \*\*, and load it with data from the CSV file, "CustomerChurn2.csv ". Verify data by comparing the row counts between the CSV file and the staging table, \*\* Customer.CustomerChurn\_Stage \*\* [Data Source: CustomerChurn2.CSV]. Provide the row count of \*\* Customer.CustomerChurn\_Stage \*\* that you loaded from CustomerChurn2.csv file. Provide the screenshot of last few rows using the SELECT \*. Make sure the output shows all column values. The SELECT statement must use the ORDER BY CustomerId.**

|  |
| --- |
| TRUNCATE Customer.CustomerChurn\_Stage;  -- WORKBENCH 8.0 has a bug where we need to set --local-infile=1 while starting the connection.  -- CHANGE THE PATH BELOW ACCORDINGLY.  LOAD DATA LOCAL INFILE "/Users/lngeorge/Documents/Untitled Folder/DBMS/week4/CustomerChurn2.csv"  INTO TABLE Customer.CustomerChurn\_Stage  COLUMNS TERMINATED BY ','  OPTIONALLY ENCLOSED BY '"'  ESCAPED BY '"'  LINES TERMINATED BY '\n'  IGNORE 1 LINES;  SELECT COUNT(\*) FROM Customer.CustomerChurn\_Stage;  SELECT \* FROM Customer.CustomerChurn\_Stage ORDER BY CustomerId ASC; |
|  |
|  |
| SELECT \* FROM Customer.CustomerChurn\_Stage ORDER BY CustomerId ASC; |
|  |

Q7. Execute the stored procedure, Customer.PrCustomerChurn, that was created in **Q4. After execution, the stored procedure should load data from the stage to the persistent table: Customer.CustomerChurn. CALL `customer`.`PrCustomerChurn`(); This time, the table will be refreshed via DELETE, UPDATE, and INSERT/SELECT statements in the stored procedure. Show the row count results of both Customer.CustomerChurn\_Version1 table [Data Source: CustomerChurn1.CSV] and the persistent table: Customer.CustomerChurn. Compare the rows between the Customer.CustomerChurn\_Version1 [Data Source: CustomerChurn1.CSV] table and the persistent table: Customer.CustomerChurn [Data Source: CustomerChurn2.CSV]. Show the rows that are available in the Customer.CustomerChurn\_Version1 table but not in the Customer.CustomerChurn table (implementation of brand-new row DELETE statement of the stored procedure).**

|  |
| --- |
| CALL `customer`.`PrCustomerChurn`(); SELECT (SELECT COUNT(\*) FROM Customer.CustomerChurn\_Version1) as CustomerChurn\_Version1Count, (SELECT COUNT(\*) FROM Customer.CustomerChurn) as CustomerChurnCount; |
|  |
| SELECT TT.CustomerId as CustomerId,  TTV1.Surname AS SurnameOld, TT.Surname AS SurnameNew,  TTV1.CreditScore AS CreditScoreOld, TT.CreditScore AS CreditScoreNew,  TTV1.Gender AS GenderOld, TT.Gender AS GenderNew,  TTV1.Age AS AgeOld, TT.Age AS AgeNew,  TTV1.Balance AS BalanceOld, TT.Balance AS BalanceNew,  TTV1.Exited AS ExitedOld, TT.Exited AS ExitedNew,  TTV1.SourceSystemNm AS SourceSystemNmOld, TT.SourceSystemNm AS SourceSystemNmNew,  TTV1.CreateAgentId AS CreateAgentIdOld, TT.CreateAgentId AS CreateAgentIdNew,  TTV1.CreateDtm AS CreateDtmOld, TT.CreateDtm AS CreateDtmNew,  TTV1.ChangeAgentId AS ChangeAgentIdOld, TT.ChangeAgentId AS ChangeAgentIdNew,  TTV1.ChangeDtm AS ChangeDtmOld, TT.ChangeDtm AS ChangeDtmNew  FROM  Customer.CustomerChurn AS TT  LEFT OUTER JOIN Customer.CustomerChurn\_Version1 AS TTV1  ON TT.CustomerId = TTV1.CustomerId  WHERE  (  COALESCE(TT.Surname, '\*') <> COALESCE(TTV1.Surname, '\*')  OR COALESCE(TT.CreditScore, '\*') <> COALESCE(TTV1.CreditScore, '\*')  OR COALESCE(TT.Geography, '\*') <> COALESCE(TTV1.Geography, '\*')  OR COALESCE(TT.Gender, '\*') <> COALESCE(TTV1.Gender, '\*')  OR COALESCE(TT.Age, '\*') <> COALESCE(TTV1.Age, '\*')  OR COALESCE(TT.Balance, '\*') <> COALESCE(TTV1.Balance, '\*')  OR COALESCE(TT.Exited, '\*') <> COALESCE(TTV1.Exited, '\*')  )  UNION  SELECT TTV1.CustomerId AS CustomerId,  TTV1.Surname AS SurnameOld, TT.Surname AS SurnameNew,  TTV1.CreditScore AS CreditScoreOld, TT.CreditScore AS CreditScoreNew,  TTV1.Gender AS GenderOld, TT.Gender AS GenderNew,  TTV1.Age AS AgeOld, TT.Age AS AgeNew,  TTV1.Balance AS BalanceOld, TT.Balance AS BalanceNew,  TTV1.Exited AS ExitedOld, TT.Exited AS ExitedNew,  TTV1.SourceSystemNm AS SourceSystemNmOld, TT.SourceSystemNm AS SourceSystemNmNew,  TTV1.CreateAgentId AS CreateAgentIdOld, TT.CreateAgentId AS CreateAgentIdNew,  TTV1.CreateDtm AS CreateDtmOld, TT.CreateDtm AS CreateDtmNew,  TTV1.ChangeAgentId AS ChangeAgentIdOld, TT.ChangeAgentId AS ChangeAgentIdNew,  TTV1.ChangeDtm AS ChangeDtmOld, TT.ChangeDtm AS ChangeDtmNew  FROM  Customer.CustomerChurn AS TT  RIGHT OUTER JOIN Customer.CustomerChurn\_Version1 AS TTV1  ON TT.CustomerId = TTV1.CustomerId  WHERE  (  COALESCE(TT.Surname, '\*') <> COALESCE(TTV1.Surname, '\*')  OR COALESCE(TT.CreditScore, '\*') <> COALESCE(TTV1.CreditScore, '\*')  OR COALESCE(TT.Geography, '\*') <> COALESCE(TTV1.Geography, '\*')  OR COALESCE(TT.Gender, '\*') <> COALESCE(TTV1.Gender, '\*')  OR COALESCE(TT.Age, '\*') <> COALESCE(TTV1.Age, '\*')  OR COALESCE(TT.Balance, '\*') <> COALESCE(TTV1.Balance, '\*')  OR COALESCE(TT.Exited, '\*') <> COALESCE(TTV1.Exited, '\*')  ); |
|  |
| 1. Here the we can see the updated columns as having both old and new values. 2. The new columns lack any old data. 3. The deleted columns lack any new data. |
| **-- Show the rows that are available in the Customer.CustomerChurn\_Version1 table but not in the Customer.CustomerChurn table (implementation of brand-new row DELETE statement of the stored procedure).**  SELECT  TTV1.\*  FROM  Customer.CustomerChurn\_Version1 AS TTV1  LEFT OUTER JOIN Customer.CustomerChurn AS TT  ON TTV1.CustomerId = TT.CustomerId  WHERE  TT.CustomerId IS NULL; |
|  |

**Q8. Show the rows (SELECT \*) that changed (one or many non-Primary Key columns), in the Customer.CustomerChurn table (implementation of UPDATE statement of the stored procedure). You need to perform a comparison between Customer.CustomerChurn table [Data Source: CustomerChurn2.CSV] and Customer.CustomerChurn\_Version1 table [Data Source: CustomerChurn1.CSV] in terms of non-PK columns (Excluds: SourceSystemNm, CreateAgentId, CreateDtm, ChangeAgentId, ChangeDtm), and with a join condition using the PK column(s). You must do ORDER BY CustomerId. The output of this query should show different values for the CreateDtm and ChangeDtm columns in Customer.CustomerChurn table for the changed rows. Take a screenshot and capture it in the Word document. Make sure all columns including CreateDtm and ChangeDtm of CustomerChurn table are displayed.**

**I have added two cases. One with select \* and one with select ColumsNames**

CASE 1: Using Select with column names makes it easier to compare.

|  |
| --- |
| SELECT  TT.CustomerId as CustomerId,  TTV1.Surname AS SurnameOld, TT.Surname AS SurnameNew,  TTV1.CreditScore AS CreditScoreOld, TT.CreditScore AS CreditScoreNew,  TTV1.Gender AS GenderOld, TT.Gender AS GenderNew,  TTV1.Age AS AgeOld, TT.Age AS AgeNew,  TTV1.Balance AS BalanceOld, TT.Balance AS BalanceNew,  TTV1.Exited AS ExitedOld, TT.Exited AS ExitedNew,  TTV1.SourceSystemNm AS SourceSystemNmOld, TT.SourceSystemNm AS SourceSystemNmNew,  TTV1.CreateAgentId AS CreateAgentIdOld, TT.CreateAgentId AS CreateAgentIdNew,  TTV1.CreateDtm AS CreateDtmOld, TT.CreateDtm AS CreateDtmNew,  TTV1.ChangeAgentId AS ChangeAgentIdOld, TT.ChangeAgentId AS ChangeAgentIdNew,  TTV1.ChangeDtm AS ChangeDtmOld, TT.ChangeDtm AS ChangeDtmNew  FROM  Customer.CustomerChurn AS TT  INNER JOIN Customer.CustomerChurn\_Version1 AS TTV1  ON TT.CustomerId = TTV1.CustomerId  WHERE  (  COALESCE(TT.Surname, '\*') <> COALESCE(TTV1.Surname, '\*')  OR COALESCE(TT.CreditScore, '\*') <> COALESCE(TTV1.CreditScore, '\*')  OR COALESCE(TT.Geography, '\*') <> COALESCE(TTV1.Geography, '\*')  OR COALESCE(TT.Gender, '\*') <> COALESCE(TTV1.Gender, '\*')  OR COALESCE(TT.Age, '\*') <> COALESCE(TTV1.Age, '\*')  OR COALESCE(TT.Balance, '\*') <> COALESCE(TTV1.Balance, '\*')  OR COALESCE(TT.Exited, '\*') <> COALESCE(TTV1.Exited, '\*')  ) ORDER BY TT.CustomerId; |
|  |
|  |

CASE 2: Using Select \*.

|  |
| --- |
| SELECT \*  FROM  Customer.CustomerChurn AS TT  INNER JOIN Customer.CustomerChurn\_Version1 AS TTV1  ON TT.CustomerId = TTV1.CustomerId  WHERE  (  COALESCE(TT.Surname, '\*') <> COALESCE(TTV1.Surname, '\*')  OR COALESCE(TT.CreditScore, '\*') <> COALESCE(TTV1.CreditScore, '\*')  OR COALESCE(TT.Geography, '\*') <> COALESCE(TTV1.Geography, '\*')  OR COALESCE(TT.Gender, '\*') <> COALESCE(TTV1.Gender, '\*')  OR COALESCE(TT.Age, '\*') <> COALESCE(TTV1.Age, '\*')  OR COALESCE(TT.Balance, '\*') <> COALESCE(TTV1.Balance, '\*')  OR COALESCE(TT.Exited, '\*') <> COALESCE(TTV1.Exited, '\*')  ) ORDER BY TT.CustomerId ASC; |
|  |
|  |

**Q9. Provide** **the screenshot of last few rows using the SELECT \* FROM Customer.CustomerChurn. Make sure the output shows all column values. The SELECT statement must use the ORDER BY CustomerId. Show the rows that are available in the Customer.CustomerChurn table [Data Source: CustomerChurn2.CSV] but not in the Customer.CustomerChurn\_Version1 table (implementation of brand-new rows INSERT by the stored procedure). Do a SELECT \* along with ORDER BY CustomerId. Take a screenshot and capture it in the Word document.**

First part asks for the screenshot of last few rows using the SELECT \* FROM Customer.CustomerChurn.

A screenshot of a computer

Description automatically generated

|  |
| --- |
| SELECT  TT.\*  FROM  Customer.CustomerChurn AS TT  LEFT OUTER JOIN Customer.CustomerChurn\_Version1 AS TTV1 ON TTV1.CustomerId = TT.CustomerId  WHERE  TTV1.CustomerId IS NULL ORDER BY TTV1.CustomerId ASC; |
|  |

**Q10. Show the final version of the stored procedure code that was used to load the persistent table, Customer.CustomerChurn. Submit it as a \*.TXT file. Submit your work (QUESTIONS 1-9) in a single Word Document/PDF file.**

|  |
| --- |
| DELIMITER $$  DROP PROCEDURE IF EXISTS Customer.PrCustomerChurn $$  CREATE PROCEDURE Customer.PrCustomerChurn() -- Replace this with actual database name, Customer and table name (with prefix Pr) that you use  BEGIN -- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  DECLARE VarCurrentTimestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP;  DECLARE VarSourceRowCount, VarTargetRowCount, VarThresholdNbr INTEGER DEFAULT 0;  DECLARE VarTinyIntVal TINYINT;  -- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  SELECT  COUNT(\*) INTO VarSourceRowCount  FROM  Customer.CustomerChurn\_Stage;  -- Replace this with actual database name and table name (e.g., CustomerChurn\_Stage) that you use.  SELECT  COUNT(\*) INTO VarTargetRowCount  FROM  Customer.CustomerChurn;  -- Replace this with actual database name and table name (e.g., CustomerChurn) that you use.  -- (TargetCount \* 20%)  SELECT  CAST(  (VarTargetRowCount \*.2) AS UNSIGNED INTEGER  ) INTO VarThresholdNbr  FROM  DUAL;  -- The DUMMY is system table which might vary from database to database. For your database, you need to figure out.  -- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  -- Fail the Stored Proc if the Source Row Count is less than the Threshold Number (i.e., 20% of the Target Table row count).  -- This ensures that the Target table is not refreshed with incomplete set of Source Data  IF VarSourceRowCount < VarThresholdNbr THEN  SELECT  -129 INTO VarTinyIntVal  FROM  DUAL;  END IF;  -- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  -- DELETE target table rows which are no longer available in source database table.  DELETE FROM  Customer.CustomerChurn AS TrgtTbl  WHERE  EXISTS (  SELECT  \*  FROM  (  SELECT  TT.CustomerID -- Primary Key Column(s)  FROM  Customer.CustomerChurn AS TT -- Example table name: CustomerChurn  LEFT OUTER JOIN Customer.CustomerChurn\_Stage AS ST -- Example table name: CustomerChurn\_Stage  ON TT.CustomerId = ST.CustomerId  WHERE  ST.CustomerId IS NULL  ) AS SrcTbl  WHERE  TrgtTbl.CustomerId = SrcTbl.CustomerId  );  -- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*UPDATE ROWS THAT CHANGED IN SOURCE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  -- Update the rows for which new version of rows have arrived as part of delta/incremental feed (i.e., change to non-key values).  UPDATE  Customer.CustomerChurn AS TrgtTbl  INNER JOIN Customer.CustomerChurn\_Stage AS SrcTbl  ON TrgtTbl.CustomerId = SrcTbl.CustomerId  SET  TrgtTbl.Surname = SrcTbl.Surname,  TrgtTbl.CreditScore = SrcTbl.CreditScore ,  TrgtTbl.Geography = SrcTbl.Geography ,  TrgtTbl.Gender = SrcTbl.Gender ,  TrgtTbl.Age = SrcTbl.Age ,  TrgtTbl.Balance = SrcTbl.Balance ,  TrgtTbl.Exited = SrcTbl.Exited ,  TrgtTbl.ChangeDtm = VarCurrentTimestamp  WHERE  (  COALESCE(TrgtTbl.Surname, '\*') <> COALESCE(SrcTbl.Surname, '\*')  OR COALESCE(TrgtTbl.CreditScore, '\*') <> COALESCE(SrcTbl.CreditScore, '\*')  OR COALESCE(TrgtTbl.Geography, '\*') <> COALESCE(SrcTbl.Geography, '\*')  OR COALESCE(TrgtTbl.Gender, '\*') <> COALESCE(SrcTbl.Gender, '\*')  OR COALESCE(TrgtTbl.Age, '\*') <> COALESCE(SrcTbl.Age, '\*')  OR COALESCE(TrgtTbl.Balance, '\*') <> COALESCE(SrcTbl.Balance, '\*')  OR COALESCE(TrgtTbl.Exited, '\*') <> COALESCE(SrcTbl.Exited, '\*')  );  -- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*INSERT BRAND NEW ROWS INTO TARGET\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  -- Identify brand new rows in source table and load into target table.  INSERT INTO Customer.CustomerChurn (  CustomerId, Surname, CreditScore,  Geography, Gender, Age, Balance, Exited,  SourceSystemNm, CreateAgentId, CreateDtm,  ChangeAgentId, ChangeDtm  )  SELECT  SrcTbl.CustomerId,  SrcTbl.Surname,  SrcTbl.CreditScore,  SrcTbl.Geography,  SrcTbl.Gender,  SrcTbl.Age,  SrcTbl.Balance,  SrcTbl.Exited,  'Kaggle-CSV' AS SourceSystemNm,  current\_user() AS CreateAgentId,  VarCurrentTimestamp AS CreateDtm,  current\_user() AS ChangeAgentId,  VarCurrentTimestamp AS ChangeDtm  FROM  Customer.CustomerChurn\_Stage AS SrcTbl  INNER JOIN (  SELECT  ST.CustomerId  FROM  Customer.CustomerChurn\_Stage AS ST  LEFT OUTER JOIN Customer.CustomerChurn AS TT ON ST.CustomerId = TT.CustomerId  WHERE  TT.CustomerId IS NULL  ) AS ChgdNew ON SrcTbl.CustomerId = ChgdNew.CustomerId;  -- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  END$$  DELIMITER ; |