1 Preliminaries

In this lab, you will work with a **SalesRecords** database schema similar to the schema that you used in Lab2. We've provided a lab3_create.sql script for you to use (which is similar to the create.sql in our Lab2 solution), so that everyone can start from the same place. Please remember to DROP and CREATE the Lab3 schema before running that script (as you did in previous labs), and also execute:

ALTER ROLE yourlogin SET SEARCH_PATH TO Lab3;

so that you'll always be using the Lab3 schema without having to mention it whenever you refer to a table.

You will need to log out and log back in to the server for this default schema change to take effect. (Students often forget to do this.)

We've also provided a lab3_data_loading.sql script that will load data into your tables. You'll need to run that script before executing Lab3. The command to execute a script is: \i <filename>

You will be required to combine new data (as explained below) into one of the tables. You will also need to add some new constraints to the database and do some unit testing to see that the constraints are followed. You will also create and query a view, and create an index.

New goals for Lab3:

1. Perform SQL to "combine data" from two tables
2. Add foreign key constraints
3. Add general constraints
4. Write unit tests for constraints
5. Create and query a view
6. Create an index

There are lots of parts in this assignment, and a few of them may be difficult. Lab3 will be discussed during the Lab Sections before the due date, Sunday, May 19. (You have an extra week to do this Lab because of the Midterm, which is on Wednesday, May 8.)

2. Description

2.1 Tables with Primary Keys for Lab3

The primary key for each table is underlined.

Products(productID, productName, manuf, normalPrice, discount)

Customers(customerID, custName, address, joinDate, amountOwed, lastPaidDate, status)

Stores(storeID, storeName, region, address, manager)

Days(dayDate, category)

Sales(productID, customerID, storeID, dayDate, paidPrice, quantity)

Payments(customerID, custName, paidDate, amountPaid, cleared)

NewCustomers(customerID, custName, address, joinDate)

In the lab3_create.sql file that we've provided under Resources→Lab3, the first 6 tables are the same as they were in our Lab2 solution, including NULL and UNIQUE constraints. Note that there is an additional table, NewCustomers that has some of the same attributes as Payments. As the table name suggests, each of its tuples records a SalesRecords customer, which may be either for a new customer, or a modification of an existing customer. We'll say more about NewCustomers below.

In practice, primary keys and unique constraints are almost always entered when tables are created, not added later, and lab3_create.sql handles those constraints for you. However, we will be adding some additional constraints to these tables, as described below.

Under Resources→Lab3, we've also given you a load script named lab3_data_loading.sql that loads tuples into the tables of the schema. You must run both lab3_create.sql and lab3_data_loading.sql before you run the parts of Lab3 that are described below.

2.2 Combine Data

Write a file, *combine.sql* (which should have multiple sql statements in it that are in a single Serializable transaction) that will do the following. For each "new customer" tuple in NewCustomers, there might already be a tuple in Customers that has the same primary key, customerID. If there **isn't** a tuple with that customerID, then this is a new SalesRecords database customer. If there already **is** a tuple with that customerID, then this is an update to the information about that customer. So here are the actions that you should take.

a) If there **isn't** already a tuple in Customers that has the same primary key, then insert a tuple into the Customers table corresponding to that NewCustomers tuple. Use customerID, customerName, address and joinDate, as provided in the NewCustomers tuple. Set amountOwed to be 0 , lastPaidDate to be NULL, and status to be 'L'.

b) If there **is** already a tuple in Customers that has the same primary key, then update Customers based on that NewCustomers tuple's attributes. Don't change customerID, amountOwed, lastPaidDate or status for that customer, but update customerName, address and joinDate based on the values of those attributes in the NewCustomers tuple. (The customer may have revised their name, address and joinDate.)

Your transaction may have multiple statements in it. The SQL constructs that we've already discussed in class are sufficient for you to do this part (which is one of the hardest parts of Lab3). Use only the SQL constructs that were discussed in class, not SQL extensions. A helpful hint is provided in the initial Lab3 announcement posted on Piazza.

2.3 Add Foreign Key Constraints

Important: Before running Sections 2.3, 2.4 and 2.5, recreate the Lab3 schema using the *lab3_create.sql* script, and load the data using the script *lab3_data_loading.sql*. That way, any database changes that you've done for Combine won't propagate to these other parts of Lab3.

Here's a description of the Foreign Keys that you need to add for this assignment. The default for referential integrity should be used in all cases. The load data that you're provided with should not cause any errors. There are other referential integrity constraints that probably would exist for this schema—you might want to think about those--but please just add the constraints listed below.

a) The productID field in Sales should reference the productID primary key Products.

b) The customerID field in Sales should reference the customerID primary key in Customers.

c) The storeID field in Sales should reference the storeID primary key in Stores.

d) The dayDate field in Sales should reference the dayDate primary key in Days.

e) The customerID field in Payments should reference the customerID primary key in Customers.

Write commands to add foreign key constraints in the same order that the foreign keys are described above. Save your commands to the file *foreign.sql*

2.4 Add General Constraints

General constraints for Lab3 are:

1. In Customers, amountOwed must be greater than or equal to zero. Please give a name to this constraint when you create it. We recommend that you use the name owed_is_not_negative, but you may use another name. The other general constraints don't need names.

2. In Sales, revenue isn't an attribute, but it's equal to paidPrice * quantity, and those are attributes of Sales. This constraint says that revenue must be at least 10.00

3. In Customers, if lastPaidDate is NULL then status must be 'L'.

Write commands to add general constraints in the order the constraints are described above, and save your commands to the file *general.sql*. (Note that UNKNOWN for a Check constraint is okay, but FALSE isn't.)

2.5 Write unit tests

Unit tests are important for verifying that your constraints are working as you expect. We will write just a few for the common cases, but there are many more possible tests we could write.

For each of the 5 foreign key constraints specified in section 2.3, write one unit test:

- An INSERT command that violates the foreign key constraint (and elicits an error).

Also, for each of the 3 general constraints, write two unit tests:

- An UPDATE command that meets the constraint.
- An UPDATE command that violates the constraint (and elicits an error).

Save these 5 + 6 = 11 unit tests, in the order given above, in the file *unittests.sql*.

2.6 Working with a view

2.6.1 Create a view

Let's say that a payment in Payments has cleared if the value of the cleared attribute is TRUE. You'll create a view that gives the total of the cleared payments for each customer. You only have to provide this information for customers who have made at least one payment that cleared. For example, if one customer has twenty payments but none of them have cleared, there won't be a tuple for that customer in the view. But if another customer has made one payment that has cleared, there will be a tuple for that customer in the view. As you've probably already deduced, you'll need to use a GROUP BY in your view.

Create a view named ClearedPayments. For each customer, there will be a tuple in ClearedPayments if there is at least one cleared payment for that customer. That tuple should have the customer's ID and the total of the cleared payments for that customer. The 2 attributes in your view should be customerID and totalClearedPayments. Your view should have no duplicates in it.

Save the script for creating the ClearedPayments view in a file called *createview.sql*.

2.6.2 Query view

Write and run a SQL query over the ClearedPayments view to answer the following "Great Customers" question. (You'll also have to use some tables to do this, but be sure to use the view.)

> **a)** There are some customers whose totalClearedPayments is more than their amountOwed.
> **b)** Some of those customers have also bought at least 3 different products. (The Sales table tells you about products that a customer bought.)
>
> We'll say that a customer is a "Great Customer" if both of a) and b) true. Write a SQL query that identifies Great Customers. For each such customer, the 3 attributes in your result should be the customer's id, totalClearedPayments, and the number of different products that the customer bought (which should appear as attribute numDiffProducts in your result).
>
> Remember: You only want a tuple in your result for customers whose numDiffProducts value is at least 3. Your result should have no duplicates

Important: Before running this query, recreate the Lab3 schema once again using the *lab3_create.sql* script, and load the data using the script *lab3_data_loading.sql*. That way, any changes that you've done for previous parts of Lab3 (e.g., Unit Test) won't affect the results of this "Great Customers" query. Then write the results of that query in a comment.

Next, write commands that deletes just the tuples with the following primary key values from the Payments table:

> (1002, 2018-02-19)
>
> (1004, 2018-02-04)

Run the "Great Customers" query once again after those deletions. Write the output of the query in a second comment. Do you get a different answer?

You need to submit a script named *queryview.sql* containing your query on the view. In that file you must also include:

- the comment with the output of the query on the provided data before the deletions,
- the SQL statements that delete the tuples indicated above,
- and a second comment with the second output of the same query after the deletions.

You do not need to replicate the query twice in the *queryview.sql* file (but you won't be penalized if you do).

It probably was much easier to write this query using the ClearedPayments view than without that view.

2.7 Create an index

Indexes are data structures used by the database to improve query performance. Locating all the Sales made to a particular customer on a particular dayDate may be slow if the database system has to search the entire Sales table. To speed up that search, create an index named LookUpSales over the customerID and dayDate attributes (in that order) of the Sales table. Save the command to do that in the file *createindex.sql*.

Of course, you can run the same SQL statements whether or not this index exists; having indexes just changes the performance of SQL statements.

For this assignment, you need not do any searches that use the index, but if you're interested, you might want to do searches with and without the index, and look at query plans using EXPLAIN to see how queries are executed. Please refer to the documentation of PostgreSQL on EXPLAIN that's at https://www.postgresql.org/docs/10/static/sql-explain.html

3 Testing

Before you submit, login to your database via psql and execute the provided database creation and load scripts, and then test your seven scripts (combine.sql foreign.sql general.sql unittests.sql createview.sql queryview.sql createindex.sql). Note that there are two sections in this document (both labeled **Important**) where you are told to recreate the schema and reload the data before running that section, so that updates you performed earlier won't affect that section. Please be sure that you follow these directions, since your answers may be incorrect if you don't.

4 Submitting

1. Save your scripts indicated above as combine.sql foreign.sql general.sql unittests.sql createview.sql queryview.sql createindex.sql. You may add informative comments inside your scripts if you want (the server interprets lines that start with two hyphens as comment lines).

2. Zip the files to a single file with name Lab3_XXXXXXX.zip where XXXXXXX is your 7-digit student ID, for example, if a student's ID is 1234567, then the file that this student submits for Lab3 should be named Lab3_1234567.zip To create the zip file you can use the Unix command:

   zip Lab3_1234567 combine.sql foreign.sql general.sql unittests.sql createview.sql queryview.sql createindex.sql

   (Of course, you use your own student ID, not 1234567.)

3. You should already know how to transfer the files from the UNIX timeshare to your local machine before submitting to Canvas.

4. Lab3 is due on Canvas by 11:59pm on Sunday, May 19. Late submissions will not be accepted, and there will be no make-up Lab assignments.