University of Toronto
CSC343, Fall 2021

Assignment 2

Original due date: Thursday, November 4th, before 8pm
Extended to: Sunday, November 7th, before 8pm

Learning Goals

The purpose of this assignment is to give you practise writing complex stand-alone SQL queries, experience using psycopg2 to embed SQL queries in a Python program, and a sense of why a blend of SQL and a general-purpose language can be the best solution for some problems.

By the end of this assignment you will be able to:

- read and interpret a novel schema written in SQL
- write complex queries in SQL
- design datasets to test a SQL query thoroughly
- quickly find and understand needed information in the postgreSQL and psycopg2 documentation
- embed SQL in a high-level language using psycopg2
- recognize limits to the expressive power of standard SQL

We will be testing your code in the **CS Teaching Labs environment** using PostgreSQL. It is your responsibility to make sure your code runs in this environment before the deadline! **Code which works on your machine but not on the CS Teaching Labs will not receive credit**.

Introduction

For this assignment, you will build components for a recommender system for online shopping.

Begin by getting familiar with the schema that we have provided. Ask yourself questions like the ones laid out for you in Assignment 1. (By now, you don't need me to provide them.)

You will work with a database that can store information about customers, online orders they make, and what items are included in each order. (Note: Throughout this assignment, we use the terms "order" and "purchase" interchangeably.) The database also stores customer reviews, which have two parts: a numeric rating and a written comment. Customers do not have to give a review, but if they do, the numeric rating is required; the written comment is optional. Customers can also rate each other's reviews as either helpful or unhelpful.

In Part 1, you will write interactive queries on the database. and in Part 2 you will write SQL to perform insert, update, and delete operations on the database.

For part 3 of the assignment, you will create some infrastructure that could be used to recommend items to customers based on the reviews and the helpfulness ratings in the database. Interactive queries are inadequate for a good recommender system. Recommender systems are a huge area of interest in both research and industry. The ACM Conference on Recommender Systems (`http://recsys.acm.org`), is a good place to get a sense of some of the techniques that are used. You'll see phrases like "multi-value probabilistic matrix factorization" and "random walk based entity ranking." Here's a very simple technique for generating recommendations for person p: build a matrix with the recommendations of various others users for various items, find among these users the person q with the most similar taste to p (comparing a vector of p's own recommendations to the recommendations of users represented in the matrix), and then report back the favourite items of q. It's hard to imagine implementing even something as simple as that in SQL. The fundamental problem is lack of iteration or recursion. (The SQL-99 standard, also

known as SQL3, does provide a way to express recursive queries, but many DBMSs don't support it.) This is one reason why we embed SQL in programs written in general-purpose programming languages like Python.

My goal in Part 3 is for you to get some practise writing psycopg2 code, and to do it in a context that helps you appreciate why and when we might bring in a general-purpose language like Python, without you having to write a lot of Python code.

Part 1: SQL Queries

General requirements

To ensure that your query results match the form expected by the auto-tester (attribute types and order, for instance), I am providing a schema for the result of each query. These can be found in files `q1.sql`, `q2.sql`, ..., `q6.sql`. You must add your solution code for each query to the corresponding file. Make sure that each file is entirely self-contained, and not dependent on any other files; each will be run separately on a fresh database instance, and so (for example) any views you create in `q1.sql` will not be accessible in `q5.sql`.

The queries

I have tried to specify these queries precisely. But if behaviour is not specified in a particular case, we will not test that case. Note that throughout, a row in the Review table counts as a review even if it has no comment associated with it.

Write SQL queries for each of the following:

1. **Unrated products** Customers who buy unrated products are of interest when marketing new products. Find all customers who have bought at least three different items that do not have any reviews, not even a rating without a comment. Report the customer CID, first name, last name and email.

2. **Helpfulness** A review is considered helpful if it has been rated for helpfulness and it has received more helpfulness ratings of True than False. A customer's helpfulness score is a number between 0 and 1, is defined as their number of helpful reviews divided by the total number of reviews they have written. If a customer has never been rated on helpfulness or has never written a review, then their helpfulness score is zero.

   For each customer, report their CID, first name, and a helpfulness category of either:

   - "very helpful" if their score is at least 0.8,
   - "somewhat helpful" if their score is at least 0.5 and less than 0.8, or
   - "not helpful" if their score is less than 0.5.

3. **Curators** Let's say that a customer is a "curator" for a category if they have bought and reviewed every item in that category, and all of those reviews contain a comment. For each curator, report their CID and the name of the category for which they are a curator. If a customer is be a curator for more than one category, report a row for each.

4. **Best and worst categories** The sales value in a month for a category is the total dollar value of all purchases of items in that category in that month. For each month of the year 2020, find the categories which have had the highest and lowest sales value in that month. Report the month, name of the category with the highest sales, value for the highest category, name of the category with the lowest sales, and value for the lowest category. Include categories that did not have any sales. Every month must be in the result, even if it didn't have any sales.

   If there are ties, report them all. Because there can be ties for both highest and lowest, these will "multiply out". For example, if in some month 3 categories are tied for highest sales and there are 2 categories with the lowest sales, there will be 6 rows in the result for that month.

5. **Hyperconsumers** Let's define a "hyperconsumer" fo a given year to be a customer whose total number of units of all items bought in that year is among the top 5 highest. If there are ties, there could be more than 5 hyperconsumers in a year. For example, if the top consumers for a year, in order, were:

A 926 units
B 901 units
C 901 units
D 884 units
E 850 units
F 790 units
G 790 units
H 790 units
I 722 units
lower amounts for the rest

then the 5 highest numbers of units are 926, 901, 884, 850, and 790, and there are 8 hyperconsumers (customers A through I). On the other hand, if few customer made purchases in a year, there may be fewer than 5 hyperconsumers. In the extreme case, in a year where no one bough anything, there would be none.

For each year in the database, report a row for each hyperconsumer that year. The row should contain the year, name of that hyperconsumer, their email address, and number of units bought. For name, concatenate the first and last name separated by a space.

6. **Year-over-year sales** The total unit sales for an item in a given month and year is the total number of units of that item purchased in that month and year. The average monthly unit sales for an item in a given a year is the average of the item's total unit sales for each of the 12 months that year.

   For each item and every pair of consecutive years starting from the earliest year in the database and going to the last year in the database, report the year-over-year change in the item's average monthly unit sales. There must be no gaps in the pairs of consecutive years reported, even if some years had no purchases.

   Year-over-year change is the percentage increase or decrease in from one year to the next. For example, if an item had 100 units sold in 2019 and 125 units sold in 2020, its year-over-year change was 25%. If a product has no sales in one year and then more than zero the next, report the increase as 'Infinity' (this is a special value that you can use in a `float` column).

   Your result should report the item id, the first year in a pair of years, the average for that year, the second year in a pair of years, the average for the second year year, and the change as a FLOAT between -100 (representing -100% sales) and 'Infinity'.

SQL Tips

- There are many details of the SQL library functions that we are not covering. It's just too vast! Expect to use the postgreSQL documentation to find the things you need. Chapter 9 on functions and operators is particularly useful. Google search is great too, but the best answers tend to come from the postgreSQL documentation.
- You may find this code helpful. It creates the 12 months of 2014.

```
CREATE VIEW Months as
SELECT to_char(DATE '2014-01-01' + (interval '1 month' * generate_series(0,11)), 'MM') as mo;
```

- This code creates a table with a series of values. It doesn't have a FROM because it doesn't need to refer to any tables to do its job.

```
csc343h-dianeh=> select generate_series(3, 8) as range;
 range
-------
     3
     4
     5
     6
     7
     8
(6 rows)
```

- When dealing with a value of type TIMESTAMP, DATE, or TIME, the EXTRACT function is handy for pulling out pieces.
- The CASE statement turns out to be quite useful, as does COALESCE.
- You may use any features defined in our version of postgreSQL on the Teaching Labs. This is not a pointed hint; I have not deliberately left features for you to discover that will dramatically simplify your work. However, I do expect that you will need to look up details in the documentation, and in fact being able to do that quickly and effectively is one of the learning outcomes of the assignment.
- Please use line breaks so that your queries do not exceed an 80-character line length.

Part 2: SQL Updates

General requirements

In this section, you will write SQL statements to perform updates. Some questions can be accomplished a single DELETE, UPDATE, or INSERT statement, but others will require several steps.

The updates

Write SQL code for each of the following:

7. **Fraud Prevention** In order to prevent credit card fraud, the online store has a limit on the number of purchases that can be made using the same credit card in 24 hours. Find any credit card that has been used on more than five purchases in the last 24 hours. Delete any purchases made after the fifth order, as well as all line items for those purchases. You can use `NOW()` to get the current date and time.

   Don't worry about the exact edge of those 24 hours (whether to use $<$ or $\leq$ and so on). We won't test your code to on cases where that makes a difference.

8. **SALE!SALE!SALE!** It's time for the annual site-wide mega sale and item prices need to be set to the new sale prices. For any item that has sold at least ten units, you will need to set the item price by applying a discount as follows: If the item costs between \$10 and \$50 inclusive, then apply a 20% discount; if the item costs more than \$50 and less than \$100, then apply a 30% discount; and if the item costs more than \$100, apply a 50% discount. There is no discount on items under \$10.

9. **Customer appreciation week** This week, the store wants to reward all customers who make a purchase with a free gift. Customers who place an order any time yesterday (the promotion period) will automatically receive a free item. Start by adding a record for the free item. The item is a mug with description "Company logo mug". The category for this item is "Housewares" and the price is free. You will need to generate a new, unique item ID. Next, find all customers who placed an order yesterday and add a line item for the free item with a quantity of one to the first order that each customer placed yesterday. (They may have made multiple separate purchases yesterday.)

   Don't assume that purchase ID's are in the same order as purchase dates.

Part 3: Embedded SQL with psycopg2

Part 3 will be available shortly.

Submission instructions

You must declare your team on MarkUs even if you are working solo, and must do so before the due date even if you are handing in during the late-with-penalty period. If you plan to work with a partner, declare this as soon as you begin working together.

For this assignment, you will hand in numerous files. MarkUs shows you if an expected file has not been submitted; check that feedback so you don't accidentally overlook a file. Also check that you have submitted the correct version of your file by downloading it from MarkUs. New files will not be accepted after the due date.