SQL Querying Exercise

Part 1: SQLZoo

Head to SQL Zoo <https://sqlzoo.net/> and complete tutorials 0, 1, 2, and 3.

Part 2: More SQL (Including aggregates)

Complete the following code SQL challenges on Codewars:

SQL Basics: Simple WHERE and ORDER BY <https://www.codewars.com/kata/sql-basics-simple-where-and-order-by/train/sql>

SQL Basics: Simple SUM <https://www.codewars.com/kata/sql-basics-simple-sum>

SQL Basics: Simple MIN / MAX <https://www.codewars.com/kata/sql-basics-simple-min-slash-max/train/sql>

Find all active students <https://www.codewars.com/kata/1-find-all-active-students/train/sql>

SQL Basics: Simple GROUP BY <https://www.codewars.com/kata/sql-basics-simple-group-by/train/sql>

SQL Basics: Simple HAVING <https://www.codewars.com/kata/sql-basics-simple-having/train/sql>

Then, complete tutorial 5 (SUM_and_COUNT) on SQL Zoo. <https://sqlzoo.net/>

Part 3: Products Querying

Setup

Download exercise starter code <../sql-querying.zip>

To seed your database, run the following command from within the starter code directory.

```
$ psql < seed_products.sql
```

This will create the database and table.

If using Postico, you will want to utilize the ***SQL Query*** interface. But you should favor using ***psql*** for this exercise, so that you can get used to the terminal interface.

There is a table called ***products*** with the following columns:

- ***id****, an auto-incrementing integer that is the unique identifier for a product*
- ***name****, text that cannot be left null*

- ***price**, which is a non-blank floating-point number;*
- ***can_be_returned**, which is a boolean, and not nullable*

Write the SQL commands necessary to do the following in ***crud.sql***. When you get a query to work correctly, copy and paste it into the `queries_products.sql` file with a comment for which query it is.

1. Add a product to the table with the name of “chair”, price of 44.00, and can_be_returned of false.
2. Add a product to the table with the name of “stool”, price of 25.99, and can_be_returned of true.
3. Add a product to the table with the name of “table”, price of 124.00, and can_be_returned of false.
4. Display all of the rows and columns in the table.
5. Display all of the names of the products.
6. Display all of the names and prices of the products.
7. Add a new product - make up whatever you would like!
8. Display only the products that ***can_be_returned***.
9. Display only the products that have a price less than 44.00.
10. Display only the products that have a price in between 22.50 and 99.99.
11. There’s a sale going on: Everything is $20 off! Update the database accordingly.
12. Because of the sale, everything that costs less than $25 has sold out. Remove all products whose price meets this criteria.
13. And now the sale is over. For the remaining products, increase their price by $20.
14. There is a new company policy: everything is returnable. Update the database accordingly.

Part 4: Google Play Store Querying

For this exercise, you are going to practice querying a relatively large dataset: close to 10k apps listed in the Google Play Store (data scraped mid 2018).

If you’re curious, the original data is from here <https://www.kaggle.com/lava18/google-play-store-apps>.

Practice running your SQL queries in the Postico SQL query box or in ***psql*** in the terminal if you prefer.

When you get a query to work correctly, copy and paste it into the `queries_playstore.sql` file with a comment for which query it is.

To seed your database, run the following command from within the starter code directory.

```
$ psql < seed_playstore.sql
```

This will create the database and table. You should now have a new database, called ***playstore***, along with a single table, ***analytics***

Here are the rest of the queries you should write, phrased like your boss is asking for specific stuff:

1. Find the app with an ID of ***1880***.
2. Find the ID and app name for all apps that were last updated on August 01, 2018.
3. Count the number of apps in each category, e.g. “Family | 1972”.
4. Find the top 5 most-reviewed apps and the number of reviews for each.
5. Find the app that has the most reviews with a rating greater than equal to 4.8.

6. Find the average rating for each category ordered by the highest rated to lowest rated.
7. Find the name, price, and rating of the most expensive app with a rating that's less than 3.
8. Find all apps with a min install not exceeding 50, that have a rating. Order your results by highest rated first.
9. Find the names of all apps that are rated less than 3 with at least 10000 reviews.
10. Find the top 10 most-reviewed apps that cost between 10 cents and a dollar.
11. Find the most out of date app. Hint: You don't need to do it this way, but it's possible to do with a subquery: http://www.postgresqltutorial.com/postgresql-max-function/ <http://www.postgresqltutorial.com/postgresql-max-function/>
12. Find the most expensive app (the query is very similar to #11).
13. Count all the reviews in the Google Play Store.
14. Find all the categories that have more than 300 apps in them.
15. Find the app that has the highest proportion of min_installs to reviews, among apps that have been installed at least 100,000 times. Display the name of the app along with the number of reviews, the min_installs, and the proportion.

Further Study: Bonus Queries

You will have to research PostgreSQL pattern matching <https://www.postgresql.org/docs/current/functions-matching.html> in addition to array functions and operators <https://www.postgresql.org/docs/current/functions-array.html>. You may also find it helpful to do some research on "subqueries."

FS1. Find the name and rating of the top rated apps in each category, among apps that have been installed at least 50,000 times.

FS2. Find all the apps that have a name similar to "facebook".

FS3. Find all the apps that have more than 1 genre.

FS4. Find all the apps that have education as one of their genres.

Futher Study: More on Subqueries

Complete tutorial 4 on SQL Zoo. <https://sqlzoo.net/>

Further Study: Choose your own data set!

Find a set of tabular data you'd be interested in querying, and practice adding and querying records using SQL. You could explore data on movies, music, sports, voting, or anything else you're curious about.

Solution

See our solution <solution/>