Homework: Advanced SQL Practice

Signal Data Science

- UNION vs UNION ALL
- COALESCE
- EXISTS
- relational division
- know about: cursors, views

This assignment is *very* focused on SQL concepts and questions which are commonly found in interviews.

For the following problems, take out multiple sheets of paper and write down your answer to each one *by hand*. At *no* point in the process should you be typing *any* SQL code *at all*. Strive to get the answer right on the very first try. Write queries in the PostgreSQL variant.

Do not collaborate on these problems. Attempt each problem without looking anything up; *only afterward* are you allowed to refer to online resources. (You may find the PostgreSQL 9.5 Documentation particularly helpful.)

When finished, check your answers against the solutions. Mark every part of every problem which was answered incorrectly and understand your error. Next week, you will redo the marked problems.

Each substantive error you find in the solutions entitles you to a $1 prize and public recognition.

Conceptual questions

Write answers to the following conceptual questions. Be concise but thorough.

- What are the differences between `RANK()`, `DENSE_RANK()`, and `ROW_NUMBER()`?
- What is the functionality of the comparison operator `<>`?
- When ordering by a timestamp column, how do you specify that recent timestamps should come before older ones?
- Examine the following series of SQLite queries:

```
sqlite> SELECT 'true' WHERE 3 IN (1, 2, 3);
true
sqlite> SELECT 'true' WHERE 3 IN (1, 2, 3, 3);
true
sqlite> SELECT 'true' WHERE 3 IN (1, 2);
sqlite> SELECT 'true' WHERE 3 NOT IN (1, 2);
true
sqlite> SELECT 'true' WHERE 3 NOT IN (1, 2, null);
```

The third and fifth queries have no output. Explain why the fifth query does not print `true`.

- Why might you choose to *not* use `SELECT *` in production code, instead writing out column names explicitly?
- What are primary keys and foreign keys?

The following questions are less essential for interviews—you should know how to answer each of them, but you do not need to practice *using* the SQL functionality described within each one.

- Briefly explain normalization and its advantages. Describe a situation where you would want to denormalize a database.

Warming up with simple queries

Each of the following problems can be solved in a single line of SQL with at most 1 level of query nesting or 1 intermediate table.

- Write a query which prints out 100 strings such that the ith string is "Fizz" if i is divisible by 3, "Buzz" if i is divisible by 5, "FizzBuzz" if i is divisible by both, or i itself as a string if i is divisible by neither.

Suppose we have an `Employees` table where each row corresponds to a single employee with columns `EmpID` for the employee's ID (a primary key), `EmpName` for the employee's name in standard form (first middle last), and `Salary` for the employee's salary.

- Write a query to find the second highest *distinct* salary in `Employees`. Do so in each of these different ways: (1) with the `NOT IN` keyword, (2) with the `<` operator, (3) with `LIMIT` and `ORDER BY`, (4) with `LIMIT` and `ORDER BY` but without any subqueries, and (5) with `DENSE_RANK()`.
- Write a query to return the names of all employees whose first names are "Adam", including those whose first names are stored with a different *case* (*e.g.*, "adAm", "ADAM", or "aDaM") but excluding those named "Adamantine" or "Quincy Adam".

An error in the company's database code has led to the insertion of new rows into `Employees` with distinct `EmpID` fields but the same `EmpName` field as preexisting rows as well as different values in *some* (potentially none or all) of the other columns. In addition, corruption in the database has completely shuffled the order of the rows, *i.e.*, the assignment of `EmpID`s.

- Write a query to list every row in `Employees` which is potentially one of the incorrectly inserted rows.

The `Employees` table cannot be fixed without correct knowledge of each employee's true salary. However, the executives have decided that in the spirit of generosity, the duplicates can be deleted such that each employee has the highest possible salary.

- Write a query which yields a version of the `Employees` table fixed in the fashion described above. (The choice of `EmpID` retained from duplicated rows doesn't matter.) Do so (1) with a window function and (2) with a subquery.

For the remaining problems, assume that the `Employees` table has been successfully fixed.

Last year, employees brought in many commissions for the company. The data for each commission are stored in the `Commissions` table with columns `EmpID` for the employee's ID, `Day` with the date of the commission, and `Amount` for the amount of money the employee brought in. At least one commission was made each day and sometimes a single employee made multiple commisssions in a single day.

- Write a query to determine, for each day of the previous year, how much money the company made via commissions from the beginning of the previous year through the end of the day.
- Write a query to list the names of employees whose total commissions in the previous year were equal to or higher than those for at least 99% of employees (*i.e.*, placed them in the 99th percentile or higher).

Information about departments is stored in a table `Departments` with columns `DeptID` for the department's ID (a primary key), `DeptName` for the name of the department, and `Goal` for the department's annual commissions goal. Correspondences between employees

More complex problems

The following problems are more complex. Using `WITH` clauses to create intermediate tables before forming the final `SELECT` query will be useful.

- Given a table `Roulette` of roulette spins with columns `Color`, either "red" or "black", and `Time`, a timestamp corresponding to when the spin was made, write a query which returns the timestamp of the longest consecutive "run" of either color.