SQL Subqueries - Lab Assignment #2

Introduction

Now that you've seen how subqueries work, it's time to get some practice writing them! Not all of the queries will require subqueries, but all will be a bit more complex and require some thought and review about aggregates, grouping, ordering, filtering, joins and subqueries. Good luck!

Objectives

You will be able to:

- Write subqueries to decompose complex queries

CRM Database ERD

Once again, here's the schema for the CRM database you'll continue to practice with.

Connect to the Database

As usual, start by importing the necessary packages and connecting to the database `data2.sqlite` in the data folder.

```
# Your code here; import the necessary packages
import sqlite3
import pandas as pd

# Your code here; create the connection
conn = sqlite3.Connection("data/data.sqlite")
```

Write an Equivalent Query using a Subquery

The following query works using a `JOIN`. Rewrite it so that it uses a subquery instead.

```
SELECT
    customerNumber,
    contactLastName,
    contactFirstName
FROM customers
JOIN orders
    USING(customerNumber)
WHERE orderDate = '2003-01-31'
;
```

```
# Your code here
q0 = """
SELECT customerNumber
      ,contactLastName
      ,contactFirstName
FROM customers
WHERE customerNumber IN
      (SELECT customerNumber
       FROM orders
       WHERE orderDate = '2003-01-31')
;
"""

q0_result = pd.read_sql(q0, conn)
q0_result
```

```
   customerNumber contactLastName contactFirstName
0             141          Freyre           Diego
```

Select the Total Number of Orders for Each Product Name

Sort the results by the total number of items sold for that product.

```
# Your code here
q1 = """
SELECT p.productName,
    (SELECT COUNT(*)
     FROM orderdetails o
     WHERE o.productCode = p.productCode) AS itemsSold
FROM products p
GROUP BY p.productName, itemsSold
ORDER BY itemsSold DESC;
"""

q1_result = pd.read_sql(q1, conn)
q1_result
```

```
                              productName  itemsSold
0            1992 Ferrari 360 Spider red         53
1    18th Century Vintage Horse Carriage         28
2                 1900s Vintage Bi-Plane         28
3                1900s Vintage Tri-Plane         28
4            1913 Ford Model T Speedster         28
..                                   ...        ...
105         1999 Indy 500 Monte Carlo SS         25
106                 2002 Chevy Corvette         25
107                   1952 Citroen-15CV         24
108                1957 Ford Thunderbird         24
109                    1985 Toyota Supra          0
```

```
[110 rows x 2 columns]
```

Select the Product Name and the Total Number of People Who Have Ordered Each Product

Sort the results in descending order.

A quick note on the SQL `SELECT DISTINCT` statement:

The `SELECT DISTINCT` statement is used to return only distinct values in the specified column. In other words, it removes the duplicate values in the column from the result set.

Inside a table, a column often contains many duplicate values; and sometimes you only want to list the unique values. If you apply the `DISTINCT` clause to a column that has `NULL`, the `DISTINCT` clause will keep only one NULL and eliminates the other. In other words, the DISTINCT clause treats all `NULL` "values" as the same value.

```
# Your code here
# Hint: because one of the tables we'll be joining has duplicate
customer numbers, you should use DISTINCT
q2 = """
SELECT DISTINCT p.productName,
                COUNT(o.customerNumber) AS uniqueCustomerOrders
FROM products p
JOIN orderdetails od ON p.productCode = od.productCode
JOIN orders o ON od.orderNumber = o.orderNumber
GROUP BY p.productName
ORDER BY uniqueCustomerOrders DESC;
"""

q2_result = pd.read_sql(q2, conn)
q2_result
```

```
                              productName  uniqueCustomerOrders
0            1992 Ferrari 360 Spider red                    53
1                          P-51-D Mustang                    28
2                              HMS Bounty                    28
3                      F/A 18 Hornet 1/72                    28
4       Diamond T620 Semi-Skirted Tanker                     28
..                                   ...                   ...
104  1932 Alfa Romeo 8C2300 Spider Sport                    25
105             1917 Grand Touring Sedan                    25
106                   1911 Ford Town Car                    25
107                1957 Ford Thunderbird                    24
108                    1952 Citroen-15CV                    24

[109 rows x 2 columns]
```

Select the Employee Number, First Name, Last Name, City (of the office), and Office Code of the Employees Who Sold Products That Have Been Ordered by Fewer Than 20 people.

This problem is a bit tougher. To start, think about how you might break the problem up. Be sure that your results only list each employee once.

```
# Your code here

# Find productCode for products ordered by fewer than 20 people
# Join orderdetails table to employees table
# Filter out productCodes that were ordered by fewer than 20 people

q3 = """
SELECT DISTINCT e.employeeNumber,
       e.firstName,
       e.lastName,
       off.city,
       off.officeCode
FROM employees e
JOIN offices off ON e.officeCode = off.officeCode
JOIN customers c ON e.employeeNumber = c.salesRepEmployeeNumber
JOIN orders ord ON c.customerNumber = ord.customerNumber
JOIN orderdetails od ON ord.orderNumber = od.orderNumber
WHERE od.productCode IN (
       SELECT od2.productCode
       FROM orderdetails od2
       JOIN orders ord2 ON od2.orderNumber = ord2.orderNumber
       GROUP BY od2.productCode
       HAVING COUNT(DISTINCT ord2.customerNumber) < 20
       );
"""

q3_result = pd.read_sql(q3, conn)
q3_result
```

	employeeNumber	firstName	lastName	city	officeCode
0	1370	Gerard	Hernandez	Paris	4
1	1501	Larry	Bott	London	7
2	1337	Loui	Bondur	Paris	4
3	1166	Leslie	Thompson	San Francisco	1
4	1286	Foon Yue	Tseng	NYC	3
5	1612	Peter	Marsh	Sydney	6
6	1611	Andy	Fixter	Sydney	6
7	1401	Pamela	Castillo	Paris	4
8	1621	Mami	Nishi	Tokyo	5

```
9              1323    George     Vanauf            NYC           3
10             1165    Leslie   Jennings  San Francisco           1
11             1702    Martin     Gerard          Paris           4
12             1216     Steve  Patterson         Boston           2
13             1188     Julie   Firrelli         Boston           2
14             1504     Barry      Jones         London           7
```

Select the Employee Number, First Name, Last Name, and Number of Customers for Employees Whose Customers Have an Average Credit Limit Over 15K

```
# Your code here

# Join customers table to employees table
# Group count of customers by employeeNumber
# Filter out groups that don't have an average > 15k

q4 = """
SELECT DISTINCT e.employeeNumber,
       e.firstName,
       e.lastName,
       COUNT(c.customerNumber) AS numberOfCustomers
FROM employees e
JOIN customers c ON e.employeeNumber = c.salesRepEmployeeNumber
GROUP BY e.employeeNumber
HAVING AVG(c.creditLimit) > 15000
ORDER BY numberOfCustomers DESC;
"""

q4_result = pd.read_sql(q4, conn)
q4_result
```

```
    employeeNumber firstName   lastName  numberOfCustomers
0             1401    Pamela   Castillo                 10
1             1504     Barry      Jones                  9
2             1501     Larry       Bott                  8
3             1323    George     Vanauf                  8
4             1370    Gerard  Hernandez                  7
5             1286  Foon Yue      Tseng                  7
6             1702    Martin     Gerard                  6
7             1337      Loui     Bondur                  6
8             1216     Steve  Patterson                  6
9             1188     Julie   Firrelli                  6
10            1166    Leslie   Thompson                  6
11            1165    Leslie   Jennings                  6
12            1621      Mami      Nishi                  5
13            1612     Peter      Marsh                  5
14            1611      Andy     Fixter                  5
```

Summary

In this lesson, you got to practice some more complex SQL queries, some of which required subqueries. There's still plenty more SQL to be had though; hope you've been enjoying some of these puzzles!