SQL Subqueries - Lab

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 `data.sqlite`.

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

# Your code here; create the connection
conn = sqlite3.connect('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'
;
```

```
# brian-answer
# ---------------------------------------------------------------------
    # q = """
    # SELECT customerNumber, contactLastName, contactFirstName,
orderDate
    # FROM customers
    # JOIN orders
    # USING(customerNumber)
    # WHERE orderDate = '2003-01-31';
    # """
# ---------------------------------------------------------------------
q = """
SELECT customerNumber, contactLastName, contactFirstName, orderDate
FROM customers
JOIN orders
USING(customerNumber)
WHERE customerNumber IN
    (SELECT customerNumber FROM orders WHERE orderDate = '2003-01-
31');
"""
# ---------------------------------------------------------------------
pd.read_sql(q, conn)
```

	customerNumber	contactLastName	contactFirstName	orderDate
0	141	Freyre	Diego	2003-01-31
1	141	Freyre	Diego	2003-06-06
2	141	Freyre	Diego	2003-06-27
3	141	Freyre	Diego	2003-09-28
4	141	Freyre	Diego	2003-10-08
5	141	Freyre	Diego	2003-11-19
6	141	Freyre	Diego	2003-12-02
7	141	Freyre	Diego	2003-12-03
8	141	Freyre	Diego	2004-01-16
9	141	Freyre	Diego	2004-04-29
10	141	Freyre	Diego	2004-05-05
11	141	Freyre	Diego	2004-06-24
12	141	Freyre	Diego	2004-08-09
13	141	Freyre	Diego	2004-10-16
14	141	Freyre	Diego	2004-12-02
15	141	Freyre	Diego	2004-12-07
16	141	Freyre	Diego	2004-12-10
17	141	Freyre	Diego	2005-02-10
18	141	Freyre	Diego	2005-02-10
19	141	Freyre	Diego	2005-02-16
20	141	Freyre	Diego	2005-02-22
21	141	Freyre	Diego	2005-03-01
22	141	Freyre	Diego	2005-03-15
23	141	Freyre	Diego	2005-05-03
24	141	Freyre	Diego	2005-05-13
25	141	Freyre	Diego	2005-05-31

Select the `Total Number` of Orders for Each `Product Name`

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

```
# brian-answer
q = """
    SELECT
        productName,
        COUNT(orderNumber) AS numberOrders,
        SUM(quantityOrdered) AS totalUnitsSold
    FROM
        products
    JOIN orderdetails
        USING (productCode)
    GROUP BY
        productName
    ORDER BY
        totalUnitsSold DESC;
"""
pd.read_sql(q, conn)
```

```
                                  productName  numberOrders
totalUnitsSold
0                1992 Ferrari 360 Spider red            53
1808
1                      1937 Lincoln Berline            28
1111
2                  American Airlines: MD-11S            28
1085
3    1941 Chevrolet Special Deluxe Cabriolet            28
1076
4               1930 Buick Marquette Phaeton            28
1074
..                                       ...           ...
...
104             1999 Indy 500 Monte Carlo SS            25
855
105                      1911 Ford Town Car             25
832
106         1936 Mercedes Benz 500k Roadster            25
824
107               1970 Chevy Chevelle SS 454            25
803
108                    1957 Ford Thunderbird            24
767

[109 rows x 3 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
q = """
    SELECT
        productName, COUNT(DISTINCT customerNumber) AS numPurchasers
    FROM
        products
    JOIN orderdetails
        USING(productCode)
    JOIN orders
        USING(orderNumber)
    GROUP BY
        productName
    ORDER BY
        numPurchasers DESC;
"""
pd.read_sql(q, conn)
```

	productName	numPurchasers
0	1992 Ferrari 360 Spider red	40
1	Boeing X-32A JSF	27
2	1972 Alfa Romeo GTA	27
3	1952 Alpine Renault 1300	27
4	1934 Ford V8 Coupe	27
..	...	...
104	1958 Chevy Corvette Limited Edition	19
105	2002 Chevy Corvette	18
106	1969 Chevrolet Camaro Z28	18
107	1952 Citroen-15CV	18
108	1949 Jaguar XK 120	18

[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.

```
# brian-added
q = """
    SELECT
        DISTINCT employeeNumber,
        officeCode,
        o.city,
        firstName,
        lastName
    FROM
        employees AS e
    JOIN offices AS o
        USING(officeCode)
    JOIN customers AS c
        ON e.employeeNumber = c.salesRepEmployeeNumber
    JOIN orders
        USING(customerNumber)
    JOIN orderdetails
        USING(orderNumber)
    WHERE productCode IN (
        SELECT productCode FROM products
        JOIN orderdetails
            USING(productCode)
        JOIN orders
            USING(orderNumber)
        GROUP BY
            productCode
        HAVING COUNT(DISTINCT customerNumber) < 20);
"""
pd.read_sql(q, conn)
```

	employeeNumber	officeCode	city	firstName	lastName
0	1370	4	Paris	Gerard	Hernandez
1	1501	7	London	Larry	Bott
2	1337	4	Paris	Loui	Bondur
3	1166	1	San Francisco	Leslie	Thompson
4	1286	3	NYC	Foon Yue	Tseng
5	1612	6	Sydney	Peter	Marsh
6	1611	6	Sydney	Andy	Fixter
7	1401	4	Paris	Pamela	Castillo
8	1621	5	Tokyo	Mami	Nishi
9	1323	3	NYC	George	Vanauf

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

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

```
# brian-added
q = """
    SELECT
        employeeNumber,
        firstName,
        lastName,
        COUNT(customerNumber) AS numCustomers
    FROM
        employees AS e
    JOIN customers As c
        ON e.employeeNumber = c.salesRepEmployeeNumber
    GROUP BY
        employeeNumber
    HAVING
        AVG(creditLimit) > 15000;
"""
pd.read_sql(q, conn)
```

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

Finally, `close` the connection.

```
conn.close()
```

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!