index

February 25, 2022

1 SQL Subqueries - Lab

1.1 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!

1.2 Objectives

You will be able to:

- Write subqueries to decompose complex queries

1.3 CRM Database ERD

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

1.4 Connect to the Database

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

[1]:
```
# Your code here; import the necessary packages
import pandas as pd
import sqlite3
conn = sqlite3.connect("data.sqlite")
cur = conn.cursor()
```

[2]:
```
# Your code here; create the connection
```

1.5 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'
;
```

[3]:
```
# Your code here
cur.execute("""
SELECT customerNumber, contactLastName, contactFirstName
FROM customers
WHERE customerNumber in (
SELECT customerNumber
FROM orders
WHERE orderDate = "2003-01-31"
);


""")

df = pd.DataFrame(cur.fetchall(), columns = [i[0] for i in cur.description])
df.head()
```

[3]:
```
   customerNumber contactLastName contactFirstName
0             141          Freyre           Diego
```

[4]:
```
# From GitHub

q = """
SELECT customerNumber, contactLastName, contactFirstName
FROM customers
WHERE customerNumber in (
SELECT customerNumber
FROM orders
WHERE orderDate = "2003-01-31"
);
"""
df = pd.read_sql(q, conn)
df.head()
```

[4]:
```
   customerNumber contactLastName contactFirstName
0             141          Freyre           Diego
```

1.6 Select the Total Number of Orders for Each Product Name

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

[5]:
```
# Your code here
cur.execute("""
SELECT productName, COUNT(productName)
FROM products
JOIN orderdetails
```

```
USING(productcode)
GROUP BY productName
ORDER BY COUNT(productName) DESC
""")

df = pd.DataFrame(cur.fetchall(), columns = [i[0] for i in cur.description] )
df
```

[5]:

	productName	COUNT(productName)
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]

[13]:
```
# From GitHub

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)
```

[13]:

	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]
```

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

Sort the results in descending order.

1.7.1 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.

[6]:
```
# Your code here
# Hint: because one of the tables we'll be joining has duplicate customer␣
 ↪numbers, you should use DISTINCT

cur.execute("""
SELECT productName, COUNT(DISTINCT customerNumber) AS num_distinct
FROM products
JOIN orderdetails
    USING (productCode)
JOIN orders
    USING(orderNumber)
JOIN customers
    USING(customerNumber)
GROUP BY productName
ORDER BY num_distinct DESC

""")

df = pd.DataFrame(cur.fetchall(), columns = [i[0] for i in cur.description])
df
```

[6]:
```
                           productName  num_distinct
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]
```

[7]:
```
# From GitHub

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)
```

[7]:
```
                             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]
```

1.8 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.

[8]:
```
# Your code here
cur.execute("""
SELECT
    DISTINCT em.employeeNumber,
    em.firstName,
    em.lastName,
    of.city,
    of.officeCode

FROM employees AS em

JOIN offices AS of
    USING(officeCode)

JOIN customers AS c
    ON em.employeeNumber = c.salesRepEmployeeNumber

JOIN orders AS ord
    USING(customerNumber)

JOIN orderdetails AS od
    USING(orderNumber)
WHERE orderNumber in ()
""")
df = pd.DataFrame(cur.fetchall(), columns = [i[0] for i in cur.description])
df
```

[8]:
```
Empty DataFrame
Columns: [employeeNumber, firstName, lastName, city, officeCode]
Index: []
```

1.9 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
```

1.10 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!

[ ]:
```

```