SQL - Lab Assignment #1

Introduction

In this lesson, we'll run through some practice questions to reinforce your knowledge of SQL queries.

Objectives

You will be able to:

- Practice interpreting "word problems" and translating them into SQL queries
- Practice deciding and performing whichever type of `JOIN` is best for retrieving desired data
- Practice using `GROUP BY` statements in SQL to apply aggregate functions like `COUNT`, `MAX`, `MIN`, and `SUM`
- Practice using the `HAVING` clause to compare different aggregates
- Practice writing subqueries to decompose complex queries

Your Task: Querying a Customer Database

 shelves filled with colorful model cars

Photo by Karen Vardazaryan on Unsplash

Business Understanding

Your employer makes miniature models of products such as classic cars, motorcycles, and planes. They want you to pull several reports on different segments of their past customers, in order to better understand past sales as well as determine which customers will receive promotional material.

Data Understanding

You may remember this database from a previous lab. As a refresher, here's the ERD diagram for this database:

The queries you are asked to write will become more complex over the course of the lab.

Getting Started

As in previous labs, we'll make use of the `sqlite3` library as well as `pandas`. By combining them, we'll be able to write queries as Python strings, then display the results in a conveniently-formatted table.

Note: Throughout this lesson, the only thing you will need to change is the content of the strings containing SQL queries. You do NOT need to modify any of the code relating to `pandas`; this is just to help make the output more readable.

In the cell below, we:

- Import the necessary libraries, `pandas` and `sqlite3`
- Establish a connection to the database `data.sqlite`, called `conn`

In [ ]:
```
# Run this cell without changes
import sqlite3
import pandas as pd

conn = sqlite3.Connection("data/data.sqlite")
```

The basic structure of a query in this lab is:

- Write the SQL query inside of the Python string
- Use `pd.read_sql` to display the results of the query in a formatted table

For example, if we wanted to select a list of all product lines from the company, that would look like this:

In [ ]:
```
# Run this cell without changes
q0 = """
SELECT productline
FROM productlines
;
"""

pd.read_sql(q0, conn)
```

From now on, you will replace `None` within these Python strings with the actual SQL query code.

Part 1: Basic Queries

First, let's review some basic SQL queries, which do not require any joining, aggregation, or subqueries.

Query 1: Customers with Credit Over 25,000 in California

Write a query that gets the contact first name, contact last name, phone number, address line 1, and credit limit for all customers in California with a credit limit greater than 25000.00.

(California means that the `state` value is `'CA'`.)

Expected Output

In [ ]:
```
# Replace None with appropriate SQL code
q1 = """
SELECT   contactFirstName,contactLastName, phone, addressLine1, creditLimit
FROM     customers
WHERE
        "creditLimit" >= 25000 AND
        "state" = "CA"
;
"""

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

The following code checks that your result is correct:

In [ ]:
```
# Run this cell without changes

# Testing which columns are returned
assert list(q1_result.columns) == ['contactFirstName', 'contactLastName', 'phor

# Testing how many rows are returned
assert len(q1_result) == 10

# Testing the values in the first result
assert list(q1_result.iloc[0]) == ['Susan', 'Nelson', '4155551450', '5677 Stror
```

Query 2: Customers Outside of the USA with "Collect" in Their Name

Write a query that gets the customer name, state, and country, for all customers outside of the USA with `"Collect"` as part of their customer name.

We are looking for customers with names like `"Australian Collectors, Co."` or `"BG&E Collectables"`, where `country` is not `"USA"`.

Expected Output

In [ ]:
```
# Replace None with appropriate SQL code
q2 = """
SELECT  customerName, state, country
FROM    customers
WHERE   "country" != "USA" AND
        "customerName" LIKE "%Collect%"
;
"""

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

The following code checks that your result is correct:

In [ ]:
```
# Run this cell without changes

# Testing which columns are returned
assert list(q2_result.columns) == ['customerName', 'state', 'country']

# Testing how many rows are returned
assert len(q2_result) == 15

# Testing the values in the first result
assert list(q2_result.iloc[0]) == ['Australian Collectors, Co.', 'Victoria', 'A
```

Query 3: Customers without Null States

Write a query that gets the full address (line 1, line 2, city, state, postal code, country) for all customers where the `state` field is not null.

Here we'll only display the first 10 results.

Expected Output

In [ ]:
```
# Replace None with appropriate SQL code
q3 = """
SELECT addressLine1, addressLine2, city, state, postalCode, country
FROM customers
WHERE state NOT LIKE 'None'
LIMIT 10
;
"""

q3_result = pd.read_sql(q3, conn)
q3_result.head(10)
```

The following code checks that your result is correct:

In [ ]:
```
# Run this cell without changes

# Testing which columns are returned
assert list(q3_result.columns) == ['addressLine1', 'addressLine2', 'city', 'sta

# Testing how many rows are returned
assert len(q3_result) == 49

# Testing the values in the first result
assert list(q3_result.iloc[0]) == ['8489 Strong St.', '', 'Las Vegas', 'NV', '8
```

You have now completed all of the basic queries!

Part 2: Aggregate and Join Queries

Query 4: Average Credit Limit by State in USA

Write a query that gets the average credit limit per state in the USA.

The two fields selected should be `state` and `average_credit_limit`, which is the average of the `creditLimit` field for that state.

Expected Output

In [ ]:
```
# Replace None with appropriate SQL code
q4 = """
SELECT state, avg(creditLimit) as average_credit_limit
FROM customers
WHERE state NOT LIKE 'None' and country == 'USA'
GROUP BY state
;
"""

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

The following code checks that your result is correct:

In [ ]:
```
# Run this cell without changes

# Testing which columns are returned
assert list(q4_result.columns) == ['state', 'average_credit_limit']

# Testing how many rows are returned
assert len(q4_result) == 8

# Testing the values in the first result
first_result_list = list(q4_result.iloc[0])
assert first_result_list[0] == 'CA'
assert round(first_result_list[1], 3) == round(83854.54545454546, 3)
```

Query 5: Joining Customers and Orders

Write a query that uses `JOIN` statements to get the customer name, order number, and status for all orders. Refer to the ERD above to understand which tables contain these pieces of information, and the relationship between these tables.

We will only display the first 15 results.

Expected Output

In [ ]:
```
# Replace None with appropriate SQL code
q5 = """
SELECT customers.customerName, orderNumber, status
FROM customers
```

```
INNER JOIN orders ON customers.customerNumber = orders.customerNumber
LIMIT 15
;
"""
q5_result = pd.read_sql(q5, conn)
q5_result.head(15)
```

The following code checks that your result is correct:

In [ ]:
```
# Run this cell without changes

# Testing which columns are returned
assert list(q5_result.columns) == ['customerName', 'orderNumber', 'status']

# Testing how many rows are returned
assert len(q5_result) == 326

# Testing the values in the first result
assert list(q5_result.iloc[0]) == ['Atelier graphique', 10123, 'Shipped']
```

Query 6: Total Payments

Write a query that uses `JOIN` statements to get top 10 customers in terms of total payment amount. Find the customer name, customer number, and sum of all payments made. The results should be ordered by the sum of payments made, starting from the highest value.

The three columns selected should be `customerName`, `customerNumber` and `total_payment_amount`.

Expected Output

In [16]:
```
# Replace None with appropriate SQL code
q6 = """
SELECT customers.customerName, customers.customerNumber, SUM(amount) AS total_p
FROM customers
INNER JOIN payments ON payments.customerNumber == customers.customerNumber
GROUP BY customers.customerName
ORDER BY
    total_payment_amount DESC
LIMIT 10

;
"""
q6_result = pd.read_sql(q6, conn)
q6_result
```

Out[16]:

	customerName	customerNumber	total_payment_amount
0	Euro+ Shopping Channel	141	715738.98
1	Mini Gifts Distributors Ltd.	124	584188.24
2	Australian Collectors, Co.	114	180585.07
3	Muscle Machine Inc	151	177913.95
4	Dragon Souveniers, Ltd.	148	156251.03
5	Down Under Souveniers, Inc	323	154622.08
6	AV Stores, Co.	187	148410.09
7	Anna's Decorations, Ltd	276	137034.22
8	Corporate Gift Ideas Co.	321	132340.78
9	Saveley & Henriot, Co.	146	130305.35

The following code checks that your result is correct:

In [17]:
```
# Run this cell without changes

# Testing which columns are returned
assert list(q6_result.columns) == ['customerName', 'customerNumber', 'total_pay

# Testing how many rows are returned
assert len(q6_result) == 10

# Testing the values in the first result
assert list(q6_result.iloc[0]) == ['Euro+ Shopping Channel', 141, 715738.98]
```

Query 7: Products that Have Been Purchased 10 or More Times

Write a query that, for each customer, finds all of the products that they have purchased 10 or more times cumulatively. For each record, return the customer name, customer number, product name, product code, and total number ordered. Sort the rows in descending order by the quantity ordered.

The five columns selected should be `customerName`, `customerNumber`, `productName`, `productCode`, and `total_ordered`, where `total_ordered` is the sum of all quantities of that product ordered by that customer.

Hint: For this one, you'll need to make use of `HAVING`, `GROUP BY`, and `ORDER BY` — make sure you get the order of them correct!

Expected Output

In [22]:
```
# Replace None with approprite SQL code
q7 = """
SELECT customers.customerName, customers.customerNumber, productName, orderdeta
```

```
FROM customers
INNER JOIN orders, orderdetails, products ON customers.customerNumber == orders
GROUP BY
    orderdetails.productCode, customers.customerNumber
HAVING total_ordered >= 10
ORDER BY
    total_ordered


;
"""
q7_result = pd.read_sql(q7, conn)
q7_result
```

Out[22]:

	customerName	customerNumber	productName	productCode	total_ordered
0	Petit Auto	314	1913 Ford Model T Speedster	S18_2949	10
1	Extreme Desk Decorations, Ltd	412	1961 Chevrolet Impala	S24_4620	10
2	La Rochelle Gifts	119	1954 Greyhound Scenicruiser	S32_2509	11
3	Tekni Collectables Inc.	328	American Airlines: B767-300	S700_1691	11
4	The Sharp Gifts Warehouse	450	1969 Chevrolet Camaro Z28	S24_3191	13
...	...	...	...	...	...
2526	Euro+ Shopping Channel	141	2002 Chevy Corvette	S24_3432	174
2527	Euro+ Shopping Channel	141	1957 Chevy Pickup	S12_4473	183
2528	Euro+ Shopping Channel	141	1970 Dodge Coronet	S24_1444	197
2529	Euro+ Shopping Channel	141	1958 Chevy Corvette Limited Edition	S24_2840	245
2530	Euro+ Shopping Channel	141	1992 Ferrari 360 Spider red	S18_3232	308

2531 rows × 5 columns

The following code checks that your result is correct:

In [23]:

```
# Run this cell without changes

# Testing which columns are returned
assert list(q7_result.columns) == ['customerName', 'customerNumber', 'productNa

# Testing how many rows are returned
assert len(q7_result) == 2531

# Testing the values in the first result
assert list(q7_result.iloc[0]) == ['Petit Auto', 314, '1913 Ford Model T Speeds
```

Query 8: Employees in Offices with Fewer than Five Employees

Finally, get the first name, last name, employee number, and office code for employees from offices with fewer than 5 employees.

Hint: Use a subquery to find the relevant offices.

Expected Output

In [25]:
```
# Replace None with approprite SQL code
q8 = """
SELECT customers.customerName, customers.customerNumber, productName, orderdeta
FROM customers
INNER JOIN orders, orderdetails, products ON customers.customerNumber == orders
GROUP BY
    orderdetails.productCode, customers.customerNumber
HAVING total_ordered >= 10
ORDER BY
    total_ordered
;
"""
q8_result = pd.read_sql(q8, conn)
q8_result
```

Out[25]:

	customerName	customerNumber	productName	productCode	total_ordered
0	Petit Auto	314	1913 Ford Model T Speedster	S18_2949	10
1	Extreme Desk Decorations, Ltd	412	1961 Chevrolet Impala	S24_4620	10
2	La Rochelle Gifts	119	1954 Greyhound Scenicruiser	S32_2509	11
3	Tekni Collectables Inc.	328	American Airlines: B767-300	S700_1691	11
4	The Sharp Gifts Warehouse	450	1969 Chevrolet Camaro Z28	S24_3191	13
...	...	...	...	...	...
2526	Euro+ Shopping Channel	141	2002 Chevy Corvette	S24_3432	174
2527	Euro+ Shopping Channel	141	1957 Chevy Pickup	S12_4473	183
2528	Euro+ Shopping Channel	141	1970 Dodge Coronet	S24_1444	197
2529	Euro+ Shopping Channel	141	1958 Chevy Corvette Limited Edition	S24_2840	245
2530	Euro+ Shopping Channel	141	1992 Ferrari 360 Spider red	S18_3232	308

2531 rows × 5 columns

The following code checks that your result is correct:

In [ ]:
```
# Run this cell without changes

# Testing which columns are returned
assert list(q8_result.columns) == ['lastName', 'firstName', 'employeeNumber', '

# Testing how many rows are returned
assert len(q8_result) == 12

# Testing the values in the first result
assert list(q8_result.iloc[0]) == ['Patterson', 'William', 1088, 6]
```

Now that we are finished writing queries, close the connection to the database:

In [29]:
```
# Run this cell without changes
conn.close()
```

Summary

In this lesson, we produced several data queries for a model car company, mainly focused around its customer data. Along the way, we reviewed many of the major concepts and

keywords associated with SQL `SELECT` queries: `FROM`, `WHERE`, `GROUP BY`, `HAVING`, `ORDER BY`, `JOIN`, `SUM`, `COUNT`, and `AVG`.

In [ ]: