

# SQL\_Lab\_Assignment\_1

November 10, 2022

## 1 SQL - Lab Assignment #1

### 1.1 Introduction

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

### 1.2 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

### 1.3 Your Task: Querying a Customer Database



Photo by Karen Vardazaryan on Unsplash

### 1.3.1 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.

### 1.3.2 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.

## 1.4 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`

```
[1]: # 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:

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

pd.read_sql(q0, conn)
```

```
[2]:      productLine
0      Classic Cars
1      Motorcycles
2          Planes
3          Ships
4          Trains
5 Trucks and Buses
6      Vintage Cars
```

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

## 1.5 Part 1: Basic Queries

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

### 1.5.1 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

```
[3]: q1 = """
SELECT contactFirstName, contactLastName, phone, addressLine1, creditLimit FROM
customers WHERE creditLimit > 25000 AND state == 'CA'
;
"""

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

```
[3]:  contactFirstName contactLastName      phone      addressLine1 \
0      Susan      Nelson  4155551450      5677 Strong St.
1      Julie      Murphy  6505555787  5557 North Pendale Street
2      Juri      Hashimoto  6505556809      9408 Furth Circle
3      Julie      Young  6265557265      78934 Hillside Dr.
4      Valarie      Thompson  7605558146      361 Furth Circle
5      Julie      Brown  6505551386      7734 Strong St.
6      Brian      Chandler  2155554369      6047 Douglas Av.
7      Sue      Frick  4085553659      3086 Ingle Ln.
8      Steve      Thompson  3105553722      3675 Furth Circle
9      Sue      Taylor  4155554312      2793 Furth Circle

      creditLimit
0      210500
1      64600
2      84600
```

```

3      90700
4      105000
5      105000
6      57700
7      77600
8      55400
9      60300

```

The following code checks that your result is correct:

```

[4]: # Run this cell without changes

# Testing which columns are returned
assert list(q1_result.columns) == ['contactFirstName', 'contactLastName',
    ↪ 'phone', 'addressLine1', 'creditLimit']

# 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
    ↪ Strong St.', 210500]

```

### 1.5.2 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

```

[5]: # 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

```

```

[5]:

```

	customerName	state	country
0	Australian Collectors, Co.	Victoria	Australia
1	Clover Collections, Co.	None	Ireland
2	UK Collectables, Ltd.	None	UK
3	King Kong Collectables, Co.	None	Hong Kong
4	Heintze Collectables	None	Denmark

5	Royal Canadian Collectables, Ltd.	BC	Canada
6	BG&E Collectables	None	Switzerland
7	Reims Collectables	None	France
8	Precious Collectables	None	Switzerland
9	Salzburg Collectables	None	Austria
10	Tokyo Collectables, Ltd	Tokyo	Japan
11	Stuttgart Collectable Exchange	None	Germany
12	Bavarian Collectables Imports, Co.	None	Germany
13	Australian Collectables, Ltd	Victoria	Australia
14	Kremlin Collectables, Co.	None	Russia

The following code checks that your result is correct:

```
[6]: # 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', 'Australia']
```

### 1.5.3 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

```
[7]: # Replace None with appropriate SQL code
q3 = """
SELECT addressLine1, addressLine2, city, state, postalCode, country from
customers WHERE state != 0
;
"""

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

```
[7]:
```

	addressLine1	addressLine2	city	state	postalCode	\
0	8489 Strong St.		Las Vegas	NV	83030	
1	636 St Kilda Road	Level 3	Melbourne	Victoria	3004	
2	5677 Strong St.		San Rafael	CA	97562	
3	5557 North Pendale Street		San Francisco	CA	94217	

4	897 Long Airport Avenue		NYC	NY	10022
5	4092 Furth Circle	Suite 400	NYC	NY	10022
6	7586 Pompton St.		Allentown	PA	70267
7	9408 Furth Circle		Burlingame	CA	94217
8	149 Spinnaker Dr.	Suite 101	New Haven	CT	97823
9	4658 Baden Av.		Cambridge	MA	51247

	country
0	USA
1	Australia
2	USA
3	USA
4	USA
5	USA
6	USA
7	USA
8	USA
9	USA

The following code checks that your result is correct:

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

# Testing which columns are returned
assert list(q3_result.columns) == ['addressLine1', 'addressLine2', 'city', 'state', 'postalCode', 'country']

# 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', '83030', 'USA']
```

You have now completed all of the basic queries!

## 1.6 Part 2: Aggregate and Join Queries

### 1.6.1 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

```
[9]: # Replace None with appropriate SQL code
q4 = ""
```

```
SELECT state, AVG(creditLimit) AS average_credit_limit FROM customers WHERE
↳country == 'USA' GROUP BY state
;
"""

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

```
[9]:   state  average_credit_limit
0    CA          83854.545455
1    CT          57350.000000
2    MA          70755.555556
3    NH          114200.000000
4    NJ          43000.000000
5    NV          71800.000000
6    NY          89966.666667
7    PA          84766.666667
```

The following code checks that your result is correct:

```
[10]: # 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)
```

### 1.6.2 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

```
[11]: # Replace None with appropriate SQL code
q5 = """
SELECT customerName, orderNumber, status from customers INNER JOIN orders on
↳customers.customerNumber = orders.customerNumber
;
"""
```

```
q5_result = pd.read_sql(q5, conn)
q5_result.head(15)
```

```
[11]:
```

	customerName	orderNumber	status
0	Atelier graphique	10123	Shipped
1	Atelier graphique	10298	Shipped
2	Atelier graphique	10345	Shipped
3	Signal Gift Stores	10124	Shipped
4	Signal Gift Stores	10278	Shipped
5	Signal Gift Stores	10346	Shipped
6	Australian Collectors, Co.	10120	Shipped
7	Australian Collectors, Co.	10125	Shipped
8	Australian Collectors, Co.	10223	Shipped
9	Australian Collectors, Co.	10342	Shipped
10	Australian Collectors, Co.	10347	Shipped
11	La Rochelle Gifts	10275	Shipped
12	La Rochelle Gifts	10315	Shipped
13	La Rochelle Gifts	10375	Shipped
14	La Rochelle Gifts	10425	In Process

The following code checks that your result is correct:

```
[12]: # 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']
```

### 1.6.3 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

```
[13]: # Replace None with appropriate SQL code
q6 = """
```



```

SELECT customerName, customers.customerNumber, SUM(payments.amount) AS
    ↳total_payment_amount from customers INNER JOIN payments on customers.
    ↳customerNumber = payments.customerNumber GROUP BY customerName ORDER BY
    ↳total_payment_amount DESC LIMIT 10
;
"""
q6_result = pd.read_sql(q6, conn)
q6_result

```

```

[13]:

```

	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 Souvenirs, Ltd.	148	156251.03
5	Down Under Souvenirs, 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:

```

[14]: # Run this cell without changes

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

# 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]

```

#### 1.6.4 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

[15]: *# Replace None with appropriate SQL code*

```
q7 = """
SELECT customerName, customers.customerNumber, productName, products.
    ↳productCode, SUM(orderdetails.quantityOrdered) AS total_ordered from
    ↳customers INNER JOIN orders on customers.customerNumber = orders.
    ↳customerNumber INNER JOIN orderdetails on orders.orderNumber = orderdetails.
    ↳orderNumber INNER JOIN products on orderdetails.productCode = products.
    ↳productCode GROUP BY customerName, productName HAVING total_ordered > 9
    ↳ORDER BY total_ordered, customers.customerNumber
;
"""
q7_result = pd.read_sql(q7, conn)
q7_result
```

[15]:

	customerName	customerNumber	\
0	Petit Auto	314	
1	Extreme Desk Decorations, Ltd	412	
2	La Rochelle Gifts	119	
3	Tekni Collectables Inc.	328	
4	The Sharp Gifts Warehouse	450	
...	...	...	
2526	Euro+ Shopping Channel	141	
2527	Euro+ Shopping Channel	141	
2528	Euro+ Shopping Channel	141	
2529	Euro+ Shopping Channel	141	
2530	Euro+ Shopping Channel	141	

	productName	productCode	total_ordered
0	1913 Ford Model T Speedster	S18_2949	10
1	1961 Chevrolet Impala	S24_4620	10
2	1954 Greyhound Scenicruiser	S32_2509	11
3	American Airlines: B767-300	S700_1691	11
4	1969 Chevrolet Camaro Z28	S24_3191	13
...	...	...	...
2526	2002 Chevy Corvette	S24_3432	174
2527	1957 Chevy Pickup	S12_4473	183
2528	1970 Dodge Coronet	S24_1444	197
2529	1958 Chevy Corvette Limited Edition	S24_2840	245
2530	1992 Ferrari 360 Spider red	S18_3232	308

[2531 rows x 5 columns]

The following code checks that your result is correct:

[16]: *# Run this cell without changes*

```
# Testing which columns are returned
```

```

assert list(q7_result.columns) == ['customerName', 'customerNumber',
    ↪ 'productName', 'productCode', 'total_ordered']

# 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',
    ↪ 'Speedster', 'S18_2949', 10]

```

### 1.6.5 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

```

[17]: # Replace None with appropriate SQL code
q8 = """
SELECT lastName, firstName, employeeNumber, officeCode from employees WHERE
    ↪ officeCode IN (SELECT officeCode from employees GROUP BY officeCode HAVING
    ↪ COUNT(officeCode) < 5)
;
"""
q8_result = pd.read_sql(q8, conn)
q8_result

```

```

[17]:      lastName  firstName  employeeNumber  officeCode
0    Patterson    William           1088           6
1     Firrelli     Julie           1188           2
2    Patterson     Steve           1216           2
3         Tseng  Foon Yue           1286           3
4     Vanauf     George           1323           3
5         Bott     Larry           1501           7
6         Jones     Barry           1504           7
7        Fixter     Andy           1611           6
8         Marsh     Peter           1612           6
9         King      Tom           1619           6
10        Nishi     Mami           1621           5
11         Kato  Yoshimi           1625           5

```

The following code checks that your result is correct:

```

[18]: # Run this cell without changes

# Testing which columns are returned

```

```

assert list(q8_result.columns) == ['lastName', 'firstName', 'employeeNumber', 'officeCode']

# 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:

```

[19]: # Run this cell without changes
conn.close()

```

## 1.7 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.