Filtering Data with SQL - Lab

Introduction

NASA wants to go to Mars! Before they build their rocket, NASA needs to track information about all of the planets in the Solar System. In this lab, you'll practice querying the database with various `SELECT` statements. This will include selecting different columns and implementing other SQL clauses like `WHERE` to return the data desired.

Objectives

You will practice the following:

- Retrieve a subset of records from a table using a `WHERE` clause
- Filter results using conditional operators such as `BETWEEN`, `IS NULL`, and `LIKE`
- Apply an aggregate function to the result of a filtered query

Connecting to the Database

To get started, import `sqlite3` as well as `pandas` for conveniently displaying results. Then, connect to the SQLite database located at `planets.db`.

```
# import pandas
import pandas as pd
# import module
import sqlite3
# creating a connection
conn = sqlite3.connect('planets.db')
# print feedback!
print('A success!')
```

```
A success!
```

Database Schema

This database contains a single table, `planets`. This is the schema:

```
CREATE TABLE planets (
  id INTEGER PRIMARY KEY,
  name TEXT,
  color TEXT,
  num_of_moons INTEGER,
  mass REAL,
```

```
  rings BOOLEAN
);
```

The data looks something like this:

id	name	color	num_of_moons	mass	rings
1	Mercury	gray	0	0.55	FALSE
2	Venus	yellow	0	0.82	FALSE
3	Earth	blue	1	1.00	FALSE
4	Mars	red	2	0.11	FALSE
5	Jupiter	orange	67	317.90	FALSE
6	Saturn	hazel	62	95.19	TRUE
7	Uranus	light blue	27	14.54	TRUE
8	Neptune	dark blue	14	17.15	TRUE

SQL Queries

Write SQL queries for each of the statements below using the same pandas wrapping syntax from the previous lesson.

1. Select just the `name` and `color` of each planet

```
# brian-answer
pd.read_sql('''
    SELECT name, color FROM planets;
''', conn)
```

```
      name       color
0  Mercury        gray
1    Venus      yellow
2    Earth        blue
3     Mars         red
4  Jupiter      orange
5   Saturn       hazel
6   Uranus  light blue
7  Neptune   dark blue
```

2. Select all columns for each planet whose `num_of_moons` is 0

```
# Your code here
pd.read_sql('''
    SELECT * FROM planets
    WHERE num_of_moons = 0;
''', conn)
```

```
   id     name   color  num_of_moons  mass  rings
0   1  Mercury    gray             0  0.55      0
1   2    Venus  yellow             0  0.82      0
```

3. Select the `name` and `mass` of each planet whose `name` has exactly 7 letters

```
# Your code here
pd.read_sql('''
    SELECT name, mass FROM planets
    WHERE length(name) = 7;
''', conn)
```

```
      name    mass
0  Mercury    0.55
1  Jupiter  317.90
2  Neptune   17.15
```

4. Select all columns for each planet whose `mass` is greater than 1.00

```
# Your code here
pd.read_sql('''
    SELECT * FROM planets
    WHERE mass > 1.00;
''', conn)
```

```
   id     name       color  num_of_moons    mass  rings
0   5  Jupiter      orange            68  317.90      0
1   6   Saturn       hazel            62   95.19      1
2   7   Uranus  light blue            27   14.54      1
3   8  Neptune   dark blue            14   17.15      1
```

5. Select the `name` and `mass` of each planet whose `mass` is less than or equal to 1.00

```
# Your code here
pd.read_sql("""
    SELECT name, mass FROM planets
    WHERE mass <= 1.00;
""", conn)
```

```
      name  mass
0  Mercury  0.55
1    Venus  0.82
2    Earth  1.00
3     Mars  0.11
```

6. Select the `name` and `mass` of each planet whose `mass` is between 0 and 50

```
# Your code here
pd.read_sql('''
    SELECT name, mass FROM planets
    WHERE mass BETWEEN 0 AND 50;
''', conn)
```

```
      name   mass
0  Mercury   0.55
1    Venus   0.82
2    Earth   1.00
3     Mars   0.11
4   Uranus  14.54
5  Neptune  17.15
```

7. Select all columns for planets that have at least one moon and a `mass` less than 1.00

Hint: You can use `AND` to chain together two conditions in SQL, similar to `and` in Python

```
# Your code here
pd.read_sql("""
    SELECT * FROM planets
    WHERE
        num_of_moons >= 1 AND mass < 1.00;
""", conn)
```

```
   id  name color  num_of_moons  mass  rings
0   4  Mars   red             2  0.11      0
```

8. Select the `name` and `color` of planets that have a `color` containing the string "blue"

```
# Your code here
pd.read_sql('''
    SELECT name, color FROM planets
    WHERE color LIKE '%blue%';
''', conn)
```

```
      name       color
0    Earth        blue
1   Uranus  light blue
2  Neptune   dark blue
```

9. Select the count of planets that don't have rings as `planets_without_rings`

Note: even though the schema states that `rings` is a `BOOLEAN` and the example table shows values `TRUE` and `FALSE`, SQLite does not actually support booleans natively. From the documentation:

> SQLite does not have a separate Boolean storage class. Instead, Boolean values are stored as integers 0 (false) and 1 (true).

```
# brian-answer
pd.read_sql('''
    SELECT COUNT(*) AS planets_without_rings
    FROM planets
    WHERE rings IS NOT 1;
''', conn)
```

```
   planets_without_rings
0                      5
```

10. Select the name of all planets, along with a value `has_rings` that returns "Yes" if the planet does have rings, and "No" if it does not

```
# brian-answer
pd.read_sql('''
    SELECT name,
        CASE rings
        WHEN 1 THEN "Yes"
        WHEN 0 THEN "No"
        END AS VERDICT
    FROM planets;
''', conn)
```

```
      name VERDICT
0  Mercury      No
1    Venus      No
2    Earth      No
3     Mars      No
4  Jupiter      No
5   Saturn     Yes
6   Uranus     Yes
7  Neptune     Yes
```

```
# disconnecting
conn.close()
```

Summary

Congratulations! NASA is one step closer to embarking upon its mission to Mars. In this lab, You practiced writing `SELECT` statements that query a single table to get specific information. You

also used other clauses and specified column names to cherry-pick the data we wanted to retrieve.