In this section, we will go over Wildcard statements, as well as ORDER BY and GROUP BY statements.

We will start by importing and connceting to our SQL database, then creating the function to convert SQL queries to a pandas DataFrame.

In [30]:
```
# Imports
import sqlite3
import pandas as pd
con = sqlite3.connect("sakila.db")

# Set function as our sql_to_pandas

def sql_to_df(sql_query):

    # Use pandas to pass sql query using connection form SQLite3
    df = pd.read_sql(sql_query, con)

    # Show the resulting DataFrame
    return df
```

Before we begin with Wildcards, ORDER BY, and GROUP BY. Let's take a look at aggregate functions.

- AVG() - Returns the average value.
- COUNT() - Returns the number of rows.
- FIRST() - Returns the first value.
- LAST() - Returns the last value.
- MAX() - Returns the largest value.
- MIN() - Returns the smallest value.
- SUM() - Returns the sum.

You can call any of these aggregate functions on a column to get the resulting values back. For example:

In [32]:
```
# Count the number of customers
query = ''' SELECT COUNT(customer_id)
            FROM customer; '''

# Grab
sql_to_df(query).head()
```

Out[32]:

| | COUNT(customer_id) |
|---|---|
| **0** | 599 |

Go ahead and experiment with the other aggregate functions. The usual syntax is:

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name
```

## SQL Wildcards

A wildcard character can be used to substitute for any other characters in a string. In SQL, wildcard characters are used with the SQL LIKE operator. The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

There are several wildcard operators:

| Wildcard | Description |
|---|---|
| % | A substitute for zero or more characters |
| _ | A substitute for a single character |
| [character_list] | Sets and ranges of characters to match |

Let's see them in action now!

In [5]:
```
# First the % wildcard

# Select any customers whose name start with an M
query = ''' SELECT *
            FROM customer
            WHERE first_name LIKE 'M%' ; '''

# Grab
sql_to_df(query).head()
```

Out[5]:

| | customer_id | store_id | first_name | last_name | email |
|---|---|---|---|---|---|
| **0** | 1 | 1 | MARY | SMITH | MARY.SMITH@sakilacustomer.org |
| **1** | 7 | 1 | MARIA | MILLER | MARIA.MILLER@sakilacustomer.org |
| **2** | 9 | 2 | MARGARET | MOORE | MARGARET.MOORE@sakilacustomer.org |
| **3** | 21 | 1 | MICHELLE | CLARK | MICHELLE.CLARK@sakilacustomer.org |
| **4** | 30 | 1 | MELISSA | KING | MELISSA.KING@sakilacustomer.org |

In [23]:
```
# Next the _ wildcard

# Select any customers whose last name ends with ing
query = ''' SELECT *
            FROM customer
            WHERE last_name LIKE '_ING' ; '''

# Grab
sql_to_df(query).head()
```

Out[23]:

| | customer_id | store_id | first_name | last_name | email | addr |
|---|---|---|---|---|---|---|
| **0** | 30 | 1 | MELISSA | KING | MELISSA.KING@sakilacustomer.org | 34 |

Now we will move on to the [Character_list] wildcard.

## IMPORTANT NOTE!

Using [charlist] with SQLite is a little different than with other SQL formats, such as MySQL.

In MySQL you would use:

WHERE value LIKE '[charlist]%'

In SQLite you use:

WHERE value GLOB '[charlist]*'

In [22]:
```
# Finally the [character_list] wildcard

# Select any customers whose first name begins with an A or a B
query = ''' SELECT *
            FROM customer
            WHERE first_name GLOB '[AB]*' ; '''

# Grab
sql_to_df(query).head()
```

Out[22]:

| | customer_id | store_id | first_name | last_name | email |
|---|---|---|---|---|---|
| **0** | 4 | 2 | BARBARA | JONES | BARBARA.JONES@sakilacustomer.org |
| **1** | 14 | 2 | BETTY | WHITE | BETTY.WHITE@sakilacustomer.org |
| **2** | 29 | 2 | ANGELA | HERNANDEZ | ANGELA.HERNANDEZ@sakilacustomer. |
| **3** | 31 | 2 | BRENDA | WRIGHT | BRENDA.WRIGHT@sakilacustomer.org |
| **4** | 32 | 1 | AMY | LOPEZ | AMY.LOPEZ@sakilacustomer.org |

# SQL ORDER BY

The ORDER BY keyword is used to sort the result-set by one or more columns. The ORDER BY keyword sorts the records in ascending order by default. To sort the records in a descending order, you can use the DESC keyword. The syntax is:

SELECT column_name
FROM table_name
ORDER BY column_name ASC|DESC

Let's see it in action:

In [24]:
```
# Select all customers and order results by last name
query = ''' SELECT *
            FROM customer
            ORDER BY last_name ; '''

# Grab
sql_to_df(query).head()
```

Out[24]:

| | customer_id | store_id | first_name | last_name | email |
|---|---|---|---|---|---|
| **0** | 505 | 1 | RAFAEL | ABNEY | RAFAEL.ABNEY@sakilacustomer.org |
| **1** | 504 | 1 | NATHANIEL | ADAM | NATHANIEL.ADAM@sakilacustomer.org |
| **2** | 36 | 2 | KATHLEEN | ADAMS | KATHLEEN.ADAMS@sakilacustomer.org |
| **3** | 96 | 1 | DIANA | ALEXANDER | DIANA.ALEXANDER@sakilacustomer.or |
| **4** | 470 | 1 | GORDON | ALLARD | GORDON.ALLARD@sakilacustomer.org |

In [25]:
```
# Select all customers and order results by last name, DESCENDING
query = ''' SELECT *
            FROM customer
            ORDER BY last_name DESC; '''

# Grab
sql_to_df(query).head()
```

Out[25]:

| | customer_id | store_id | first_name | last_name | email | ad |
|---|---|---|---|---|---|---|
| **0** | 28 | 1 | CYNTHIA | YOUNG | CYNTHIA.YOUNG@sakilacustomer.org | 32 |
| **1** | 413 | 2 | MARVIN | YEE | MARVIN.YEE@sakilacustomer.org | 41 |
| **2** | 402 | 1 | LUIS | YANEZ | LUIS.YANEZ@sakilacustomer.org | 40 |
| **3** | 318 | 1 | BRIAN | WYMAN | BRIAN.WYMAN@sakilacustomer.org | 32 |
| **4** | 31 | 2 | BRENDA | WRIGHT | BRENDA.WRIGHT@sakilacustomer.org | 35 |

# SQL GROUP BY

The GROUP BY statement is used with the aggregate functions to group the results by one or more columns. The syntax is:

SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name;

Let's see how it works.

In [29]:
```
# Count the number of customers per store

query = ''' SELECT store_id , COUNT(customer_id)
            FROM customer
            GROUP BY store_id; '''

# Grab
sql_to_df(query).head()
```

Out[29]:

| | store_id | COUNT(customer_id) |
|---|---|---|
| **0** | 1 | 326 |
| **1** | 2 | 273 |

Good job! That's it for now!

In [ ]:
```

```