SQL Statement Fundamentals

SQL

- This section focuses on basic SQL syntax that you will end up using in almost all your future queries.
- The syntax shown in this section is applicable to any major SQL engine (e.g. MS SQL Server, MySQL, Oracle, etc...)

SQL

- In general we will focus on the syntax for constructing a SQL **query**
    - **Query :** A request for information from the database.
  - After every major concept, we will have a challenge task for you to complete.
  - Let’s get started!

SELECT

SQL

- **SELECT** is the most common statement used, and it allows us to retrieve information from a table.
- Later on we will learn how to combine **SELECT** with other statements to perform more complex queries.

SQL

- Example syntax for **SELECT** statement:

SELECT column_name **FROM** table_name

SQL

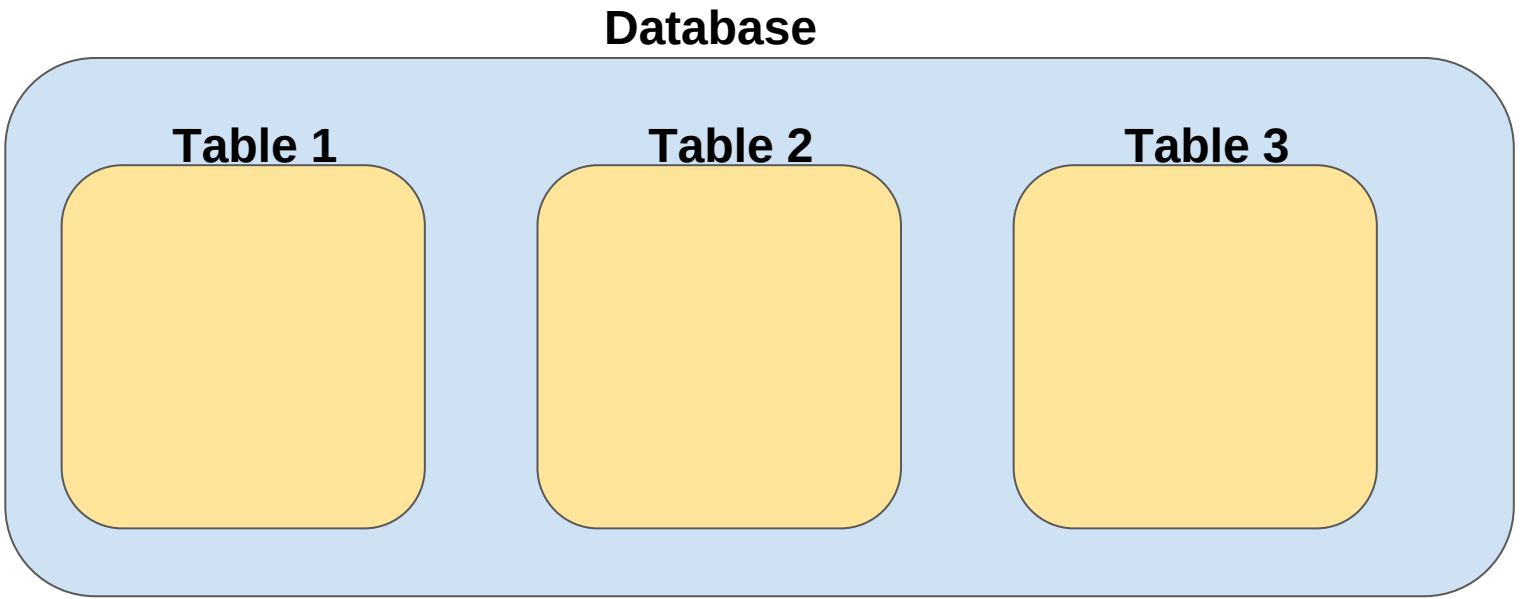
SELECT column_name **FROM** table_name

SELECT column_name **FROM** table_name

Database

SQL

SELECT column_name **FROM** table_name



PIERIAN DATA

SQL

SELECT column_name **FROM** table_name

Database

Table 1

c1	c2	c3
x	23	a
y	18	b
z	46	c

Table 2

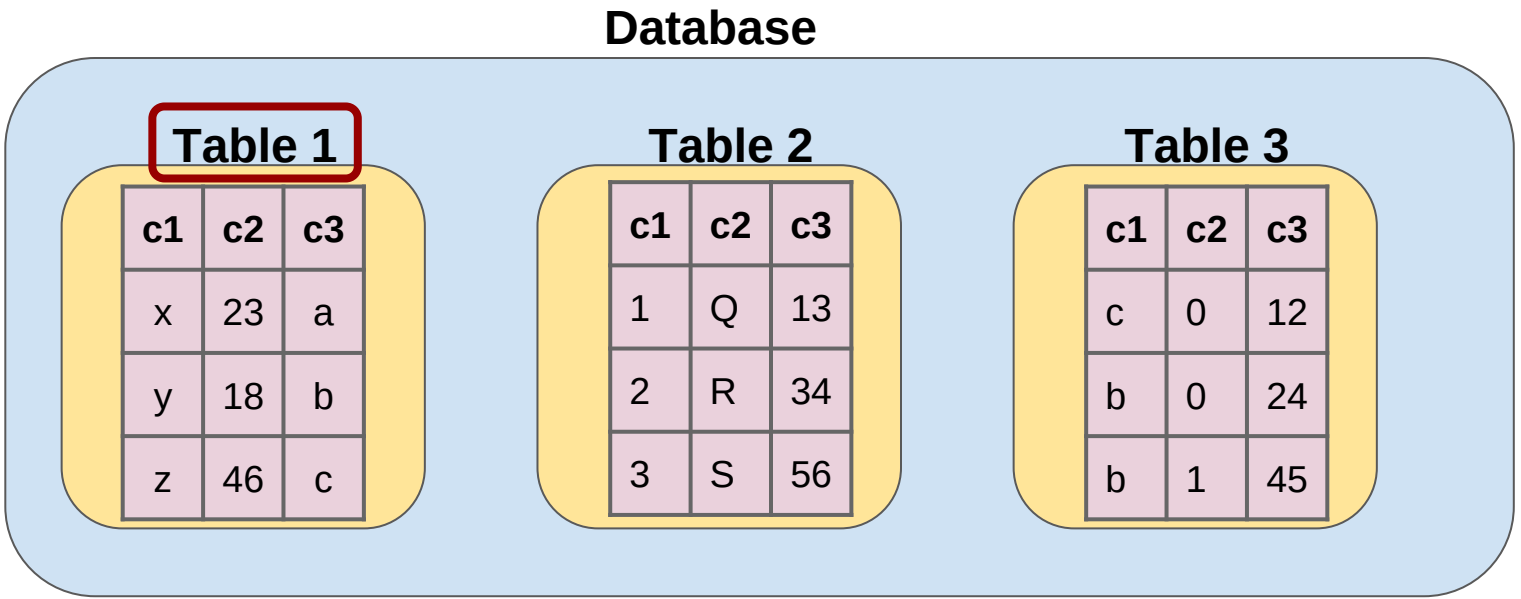
c1	c2	c3
1	Q	13
2	R	34
3	S	56

Table 3

c1	c2	c3
c	0	12
b	0	24
b	1	45

SQL

SELECT column_name **FROM** table_name



SQL

SELECT column_name **FROM** table_name

Database

Table 1

c1	c2	c3
x	23	a
y	18	b
z	46	c

Table 2

c1	c2	c3
1	Q	13
2	R	34
3	S	56

Table 3

c1	c2	c3
c	0	12
b	0	24
b	1	45

SQL

SELECT c1 **FROM** table_1

Database

Table 1

c1	c2	c3
x	23	a
y	18	b
z	46	c

Table 2

c1	c2	c3
1	Q	13
2	R	34
3	S	56

Table 3

c1	c2	c3
c	0	12
b	0	24
b	1	45

SQL

SELECT c1, c2 **FROM** table_1

Database

Table 1

c1	c2	c3
x	23	a
y	18	b
z	46	c

Table 2

c1	c2	c3
1	Q	13
2	R	34
3	S	56

Table 3

c1	c2	c3
c	0	12
b	0	24
b	1	45

PIERIAN DATA

SQL

SELECT c1, c3 **FROM** table_1

Database

Table 1

c1	c2	c3
x	23	a
y	18	b
z	46	c

Table 2

c1	c2	c3
1	Q	13
2	R	34
3	S	56

Table 3

c1	c2	c3
c	0	12
b	0	24
b	1	45

PIERIAN DATA

SQL

SELECT * **FROM** table_1

Database

Table 1

c1	c2	c3
x	23	a
y	18	b
z	46	c

Table 2

c1	c2	c3
1	Q	13
2	R	34
3	S	56

Table 3

c1	c2	c3
c	0	12
b	0	24
b	1	45

SQL

- In general it is not good practice to use an asterisk (*) in the SELECT statement if you don’t really need all columns.
- It will automatically query everything, which increases traffic between the database server and the application, which can slow down the retrieval of results.

SQL

- If you only need certain columns, do your best to only query for those columns.
- Let’s walk through some examples in our dvdrental database to get some practice!

SELECT

CHALLENGE TASKS

SQL

- Challenge Structures
    - Business Situation
    - Challenge Question
    - Expected Answer
    - Hints
    - Solution

SQL

- Challenge Structures
    - Business Situation
    - Challenge Question
    - Expected Answer
    - Hints
    - Solution

SQL

- Situation
  - We want to send out a promotional email to our existing customers!

SQL

- Situation
  - We want to send out a promotional email to our existing customers!

SQL

- Challenge
  - Use a **SELECT** statement to grab the first and last names of every customer and their email address.

SQL

- Expected Answer: (may not be displayed in the exact same order)

Data Output | Explain | Messages | Notifications

	first_name character varying (45)	last_name character varying (45)	email character varying (50)
1	Jared	Ely	jared.ely@sakilacustomer.org
2	Mary	Smith	mary.smith@sakilacustomer...
3	Patricia	Johnson	patricia.johnson@sakilacust...
4	Linda	Williams	linda.williams@sakilacusto...
5	Barbara	Jones	barbara.jones@sakilacusto...
6	Elizabeth	Brown	elizabeth.brown@sakilacust...

SQL

- Hints
  - Use the **customer** table
  - You can use the table drop-down to view what columns are available
  - You could also use **SELECT * FROM customer** to see all the columns.

SQL

- Solution
    - Let’s check it out in pgadmin!

SELECT DISTINCT

SQL

- Sometimes a table contains a column that has duplicate values, and you may find yourself in a situation where you only want to list the unique/distinct values.
- The **DISTINCT** keyword can be used to return only the distinct values in a column.

SQL

- The **DISTINCT** keyword operates *on* a column. The syntax looks like this:

SELECT DISTINCT column **FROM** table

SQL

- To clarify which column DISTINCT is being applied to, you can also use parenthesis for clarity:

  SELECT DISTINCT(column) **FROM** table

SQL

- It will work with or without parenthesis.
- Later on when we learn about adding more calls such as COUNT and DISTINCT together, the parenthesis will be necessary.

SELECT DISTINCT column **FROM** table

SQL

- What does it actually mean to call DISTINCT on a column?

Name	Choice
Zach	Green
David	Green
Claire	Yellow
David	Red

SQL

- Imagine a table of people who were surveyed to choose a color:

Name	Choice
Zach	Green
David	Green
Claire	Yellow
David	Red

SQL

- SELECT DISTINCT name FROM color_table

Name	Choice
Zach	Green
David	Green
Claire	Yellow
David	Red

SQL

- SELECT DISTINCT name FROM color_table

Name
Zach
David
Claire

SQL

- Given the previous example, we don't really know if the person with the name "David" was a duplicate entry, or two different people with the same first name.
- Calling DISTINCT here answered the question
  - *What are the unique first names are there in the table?*

SQL

- It makes more sense to ask “How many types of unique color choices were there?”

Name	Choice
Zach	Green
David	Green
Claire	Yellow
David	Red

SQL

- SELECT DISTINCT choice FROM color_table

Choice
Green
Yellow
Red

SQL

- Let’s see a use case of when DISTINCT would be useful.

SELECT DISTINCT

CHALLENGE

SQL

- Situation
- SQL Challenge
- Expected Results
- Hints
- Solution

SQL

- Situation
  - An Australian visitor isn’t familiar with MPAA movie ratings (e.g. PG , PG-13, R, etc...)
  - We want to know the types of ratings we have in our database.
  - What ratings do we have available?

SQL

- SQL Challenge
  - Use what you've learned about SELECT DISTINCT to retrieve the distinct rating types our films could have in our database.

SQL

- Expected Result

Data Output | Explain

	rating mpaa_rating
1	NC-17
2	G
3	PG
4	PG-13
5	R

SQL

- Hints
  - Use the film table
  - Use **SELECT * FROM film;** to see what columns are available.
  - Or use drop down table menu in pgadmin.

SQL

- Solution
  - SELECT DISTINCT rating FROM film;

COUNT

SQL

- The COUNT function returns the number of input rows that match a specific condition of a query.
- We can apply COUNT on a specific column or just pass COUNT(*) , we will soon see this should return the same result.

SQL

- Let’s see a simple example

Name	Choice
Zach	Green
David	Green
Claire	Yellow
David	Red

SQL

- SELECT COUNT(name) FROM table;

Name	Choice
Zach	Green
David	Green
Claire	Yellow
David	Red

SQL

- SELECT COUNT(name) FROM table;

Count
4

SQL

- SELECT COUNT(name) FROM table;
- This is simply returning the number of rows in the table.
- In fact, it should be the same regardless of the column.

Count
4

SQL

- Each column has the same number of rows

Name	Choice
Zach	Green
David	Green
Claire	Yellow
David	Red

SQL

- SELECT COUNT(name) FROM table;
- SELECT COUNT(choice) FROM table;
- SELECT COUNT(*) FROM table;
- All return the same thing, since the original table had 4 rows.

Count
4

PIERIAN DATA

SQL

- Because of this COUNT by itself simply returns back a count of the number of rows in a table.
- COUNT is much more useful when combined with other commands, such as DISTINCT

SQL

- Imagine we wanted to know: *How many unique names are there in the table?*

Name	Choice
Zach	Green
David	Green
Claire	Yellow
David	Red

SQL

- SELECT COUNT(DISTINCT name) FROM table;

Name	Choice
Zach	Green
David	Green
Claire	Yellow
David	Red

SQL

- SELECT COUNT(DISTINCT name) FROM table;

Name	Choice
Zach	Green
David	Green
Claire	Yellow
David	Red

SQL

- SELECT COUNT(DISTINCT name) FROM table;

Name	Choice
Zach	Green
David	Green
Claire	Yellow

SQL

- SELECT COUNT(DISTINCT name) FROM table;

Count
3

SQL

- SELECT COUNT(DISTINCT name) FROM table;

Count
3

SQL

- SELECT COUNT(DISTINCT(name)) FROM table;

Count
3

SQL

- Let’s explore some examples in pgAdmin!

SELECT WHERE

PART ONE

SQL

- **SELECT** and **WHERE** are the most fundamental SQL statements and you will find yourself using them often!
- The WHERE statement allows us to specify conditions on columns for the rows to be returned.

SQL

- Basic syntax example:
    - SELECT column1, column2

      FROM table

      WHERE conditions;

SQL

- The **WHERE** clause appears immediately after the FROM clause of the SELECT statement.
- The conditions are used to filter the rows returned from the SELECT statement.
- PostgreSQL provides a variety of standard operators to construct the conditions

SQL

- Comparison Operators
  - Compare a column value to something.
    - Is the price *greater than* $3.00?
    - Is the pet's name *equal to* **"Sam"?**

SQL

- Comparison Operators

Operator	Description
=	Equal
>	Greater than
<	Less Than
>=	Greater than or equal to
<=	Less than or equal to
<> or !=	Not equal to

SQL

- Logical Operators
  - Allow us to combine multiple comparison operators
    - **AND**
    - **OR**
    - **NOT**

SQL

- Simple Syntax Example

Name	Choice
Zach	Green
David	Green
Claire	Yellow
David	Red

SQL

- **SELECT** name,choice **FROM** table

Name	Choice
Zach	Green
David	Green
Claire	Yellow
David	Red

SQL

- **SELECT** name,choice **FROM** table
- Now let’s get only the people named David

Name	Choice
Zach	Green
David	Green
Claire	Yellow
David	Red

PIERIAN DATA

SQL

- **SELECT** name,choice **FROM** table **WHERE** name = ‘David’

Name	Choice
David	Green
David	Red

SQL

- **SELECT** name **FROM** table **WHERE** name = ‘David’

Name
David
David

PIERIAN DATA

SQL

- **SELECT** name,choice **FROM** table **WHERE** name = ‘David’

Name	Choice
David	Green
David	Red

SQL

- **SELECT** name,choice **FROM** table **WHERE** name = ‘David’ **AND** choice= ‘Red’

Name	Choice
David	Red

SELECT WHERE

PART TWO

Code Along Examples

SELECT WHERE

Challenge Task

SQL

- We now know enough to answer more realistic business questions and tasks instead of directly asking for specific SQL tasks.
- From now on we will focus more on directly asking the business related questions, to more realistically model a typical task.

SQL

- For example:
  - How many customers have the first name Jared?
- Instead of:
  - Use **SELECT WHERE** to find “Jared” in the **first_name** column in the **customer** table.

SQL

- One last thing to keep in mind is that as we continue to learn more about SQL, you will soon realize there are usually many different ways to arrive at the same solution
- Verify your work mainly against the expected result instead of our SQL solution

SQL

- Business Situation/Question
- Expected Result
- Hints
- Solution

SQL

- Challenge No. 1
  - A customer forgot their wallet at our store! We need to track down their email to inform them.
  - What is the email for the customer with the name Nancy Thomas?

SQL

- Expected Answer for Challenge No. 1

	email character varying (50)
1	nancy.thomas@sakilacustomer.org

SQL

- Hints for Challenge No. 1
  - Use the customer table
  - Make sure the capitalization and spelling of the names is correct
  - Use AND to combine conditions
  - Use single quotes around the ‘string’

SQL

- Solution for Challenge No. 1
  - SELECT email FROM customer

    WHERE first_name = ‘Nancy’

    AND last_name = ‘Thomas’;

SQL

- Challenge No. 2
  - A customer wants to know what the movie “Outlaw Hanky” is about.
  - Could you give them the description for the movie “Outlaw Hanky”?

SQL

- Expected Answer for Challenge No. 2

	description text
1	A Thoughtful Story of a Astronaut And a Composer who must Conquer a Dog in The Sahara Desert

SQL

- Hints for Challenge No. 2
  - Use the film table
  - Make sure the capitalization and spelling of the movie name is correct
  - Use single quotes around the ‘string’

SQL

- Solution for Challenge No. 2
  - SELECT description FROM film

    WHERE title = ‘Outlaw Hanky’;

SQL

- Challenge No. 3
  - A customer is late on their movie return, and we've mailed them a letter to their address at **'259 Ipoh Drive'**. We should also call them on the phone to let them know.
  - Can you get the phone number for the customer who lives at **'259 Ipoh Drive'?**

SQL

- Expected Answer for Challenge No. 3

Data Output | Explain | Messages

	phone character varying (20)
1	419009857119

SQL

- Hints for Challenge No. 3
  - Use the address table
  - Make sure the capitalization and spelling of the address is correct
  - Use single quotes around the ‘string’

SQL

- Solution for Challenge No. 3
  - SELECT phone FROM address

    WHERE address= ‘259 Ipoh Drive’;

ORDER BY

SQL

- You may have noticed PostgreSQL sometimes returns the same request query results in a different order.
- You can use ORDER BY to sort rows based on a column value, in either ascending or descending order.

SQL

- Basic syntax for ORDER BY
    - **SELECT** column_1,column_2
      FROM table
      ORDER BY column_1 **ASC / DESC**

SQL

- Notice ORDER BY towards the end of a query, since we want to do any selection and filtering first, before finally sorting.
    - **SELECT** column_1,column_2
      FROM table
      ORDER BY column_1 **ASC / DESC**

SQL

- Use ASC to sort in ascending order
- Use DESC to sort in descending order
- If you leave it blank, ORDER BY uses ASC by default.

SQL

- You can also ORDER BY multiple columns
- This makes sense when one column has duplicate entries.

Company	Name	Sales
Apple	Andrew	100
Google	David	500
Apple	Zach	300
Google	Claire	200
Xerox	Steven	100

PIERIAN DATA

SQL

- **SELECT** company,name,sales **FROM** table **ORDER BY** company,sales

Company	Name	Sales
Apple	Andrew	100
Apple	Zach	300
Google	Claire	200
Google	David	500
Xerox	Steven	100

SQL

- Let’s explore some examples in pgAdmin!

LIMIT

SQL

- The LIMIT command allows us to limit the number of rows returned for a query.
- Useful for not wanting to return every single row in a table, but only view the top few rows to get an idea of the table layout.
- LIMIT also becomes useful in combination with ORDER BY

SQL

- LIMIT goes at the very end of a query request and is the last command to be executed.
- Let’s learn the basic syntax of LIMIT through some examples.

ORDER BY

Challenge Tasks

SQL

- Challenge Task / Business Situation
- Expected Result
- Hints
- Solution

SQL

- Challenge Task
  - We want to reward our first 10 paying customers.
  - What are the customer ids of the first 10 customers who created a payment?

SQL

- Expected Result

Data Output | Explain

	customer_id smallint
1	416
2	516
3	239
4	592
5	49
6	264
7	46
8	481
9	139
10	595

SQL

- Hints
  - Use the payment table
  - You will need to use both ORDER BY and LIMIT
  - Remember you may need to specify ASC or DESC

SQL

- Solution

```
SELECT customer_id FROM payment
ORDER BY payment_date ASC
LIMIT 10;
```

SQL

- Challenge Task
  - A customer wants to quickly rent a video to watch over their short lunch break.
  - What are the titles of the 5 shortest (in length of runtime) movies?

SQL

- Expected Results

Data Output | Explain | Messages | Notificati

	title character varying (255)	length smallint
1	Labyrinth League	46
2	Alien Center	46
3	Iron Moon	46
4	Kwai Homeward	46
5	Ridgemont Submarine	46

PIERIAN DATA

SQL

- Hints
  - Use the film table
  - Take a look at the length column
  - You can use ORDER BY and LIMIT
  - Remember to use ASC or DESC to get desired results

SQL

- Example Solution

```
SELECT title,length FROM film
ORDER BY length ASC
LIMIT 5;
```

SQL

- Quick Bonus Question
  - If the previous customer can watch any movie that is 50 minutes or less in run time, how many options does she have?

SQL

- Expected Result
  - 37

SQL

- SELECT COUNT(title) FROM film WHERE length <= 50

BETWEEN

SQL

- The BETWEEN operator can be used to match a value against a range of values:
    - value BETWEEN low AND high

SQL

- The BETWEEN operator is the same as:
  - value >= low AND value <= high
  - value BETWEEN low AND high

SQL

- You can also combine BETWEEN with the NOT logical operator:
    - value NOT BETWEEN low AND high

SQL

- The NOT BETWEEN operator is the same as:
  - value < low OR value > high
  - value NOT BETWEEN low AND high

SQL

- The BETWEEN operator can also be used with dates. Note that you need to format dates in the ISO 8601 standard format, which is YYYY-MM-DD
    - date BETWEEN ‘2007-01-01’ AND ‘2007-02-01’

SQL

- When using BETWEEN operator with dates that also include timestamp information, pay careful attention to using BETWEEN versus <=,>= comparison operators, due to the fact that a datetime starts at 0:00.
- Later on we will study more specific methods for datetime information types.

SQL

- Let’s get some quick practice in pgAdmin!

IN

SQL

- In certain cases you want to check for multiple possible value options, for example, if a user's name shows up **IN** a list of known names.
- We can use the **IN** operator to create a condition that checks to see if a value in included in a list of multiple options.

SQL

- The general syntax is:
    - value **IN** (option1,option2,...,option_n)

SQL

- Example query:
    - SELECT color FROM table

      WHERE color IN (‘red’,’blue’)

SQL

- Example query:
    - SELECT color FROM table

      WHERE color IN (‘red’,’blue’,’green’)

SQL

- Example query:
    - SELECT color FROM table

      WHERE color NOT IN (‘red’,’blue’)

SQL

- Let’s see some examples in pgAdmin!

LIKE and ILIKE

Using Pattern Matching

SQL

- We've already been able to perform direct comparisons against strings, such as:
  - WHERE first_name= 'John'
  - But what if we want to match against a general pattern in a string?
    - All emails ending in '@gmail.com'
    - All names that begin with an 'A'

SQL

- The LIKE operator allows us to perform pattern matching against string data with the use of **wildcard** characters:
    - Percent %
        - Matches any sequence of characters
    - Underscore _
        - Matches any single character

SQL

- All names that begin with an ‘A’
  - WHERE name LIKE ‘A%’
- All names that end with an ‘a’
  - WHERE name LIKE ‘%a’

Notice that LIKE is case-sensitive, we can use ILIKE which is case-insensitive

SQL

- Using the underscore allows us to replace just a single character
  - Get all Mission Impossible films
  - WHERE title LIKE ‘Mission Impossible _’

SQL

- You can use multiple underscores
- Imagine we had version string codes in the format ‘Version#A4’ , ‘Version#B7’, etc...
  - **WHERE** value **LIKE** ‘Version#__’

SQL

- We can also combine pattern matching operators to create more complex patterns
  - WHERE name LIKE ‘_her%’
    - Cheryl
    - Theresa
    - Sherri

SQL

- We can also combine pattern matching operators to create more complex patterns
  - WHERE name LIKE '_her%'
    - Cher**yl**
    - Ther**esa**
    - Sher**ri**

SQL

- Here we just focus on LIKE and ILIKE for now, but keep in mind PostgreSQL does support full regex capabilities:
  - https://www.postgresql.org/docs/12/functions-matching.html

SQL

- Let’s check out some examples in PgAdmin!

SQL Statement Fundamentals

GENERAL CHALLENGE

PIERIAN DATA

SQL

- Time to test your skills!
- We’re going to set you up with a few situations and your task is to figure out the SQL query to solve them.
- Note! You can also check out the supplemental resource link for this lecture to view these tasks in a simple document.

SQL

- The online doc has the task and the expected result.
- This video will have:
  - The Task
  - The Expected Result
  - Hints
- The next video will work through the solutions.

SQL

- Task No. 1