# Marketing Analytics – Lab 3

## The Very Basic Syntax

SELECT…FROM

GROUP BY

WHERE…

GROUP BY… HAVING

COUNT()

ORDER BY

## SELECT… FROM

- The SELECT statement is used to select data from a database.

```
SELECT column1, column2, ...
FROM table_name;
```

- To select all the fields:

```
SELECT * FROM table_name;
```

```
query = """
      SELECT Name
      FROM `bigquery-public-data.pet_records.pets`
"""
```

| ID | Name | Animal |
|---|---|---|
| 1 | Dr. Harris Bonkers | Rabbit |
| 2 | Moon | Dog |
| 3 | Ripley | Cat |
| 4 | Tom | Cat |

## SELECT DISTINCT

- The SELECT DISTINCT statement is used to return only distinct (different) values.

```
SELECT DISTINCT column1, column2, ...
FROM table_name;
```

# WHERE …

- The WHERE clause is used to filter records that fulfil a specified condition.

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

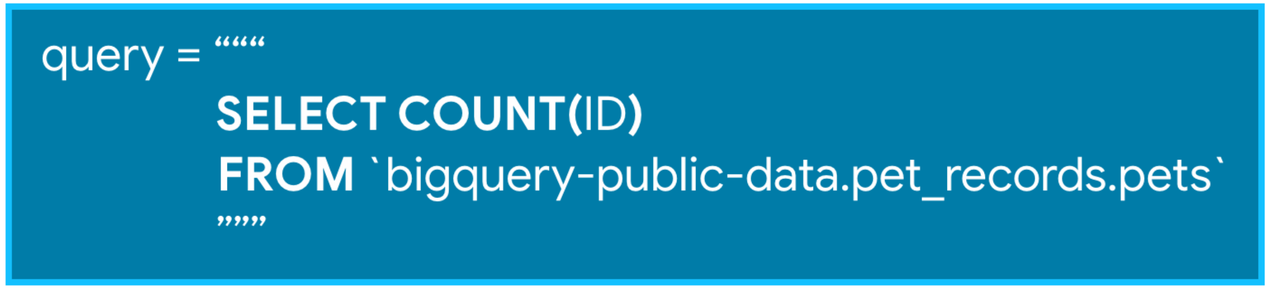
```
query = """
        SELECT Name
        FROM `bigquery-public-data.pet_records.pets`
        WHERE Animal = 'Cat'
        """
```

| ID | Name | Animal |
|---|---|---|
| 1 | Dr. Harris Bonkers | Rabbit |
| 2 | Moon | Dog |
| 3 | Ripley | Cat |
| 4 | Tom | Cat |

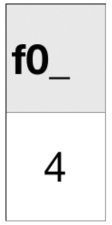
# COUNT()

- The COUNT() function returns the number of rows.

```
SELECT COUNT(column_name)
FROM table_name;
```



```
query = “““
        SELECT COUNT(ID)
        FROM `bigquery-public-data.pet_records.pets`
        ”””
```



| f0_ |
|---|
| 4 |

**COUNT(DISTINCT)** allows you to count the number of distinct or unique values in a given column. The **DISTINCT** keyword can be used in conjunction with **COUNT** to count the number of unique values within a specific column or set of columns.

## Aggregate Function

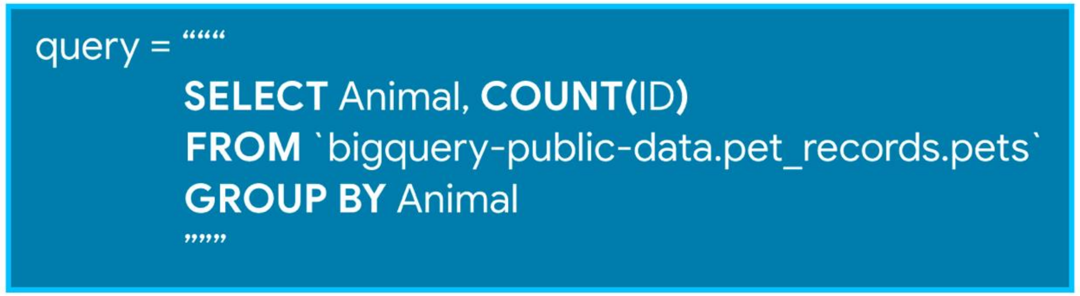
**COUNT()** is an example of an **aggregate function**, which takes many values and returns one.
Other aggregate functions include

- **SUM()**
- **AVG()**
- **MIN()**
- **MAX()**

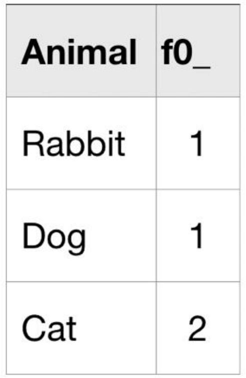
# GROUP BY

- The GROUP BY statement groups rows that have the same values into summarised rows. The statement is often used with aggregate functions.

```
SELECT column_name(s)
FROM table_name
GROUP BY column_name(s)
```



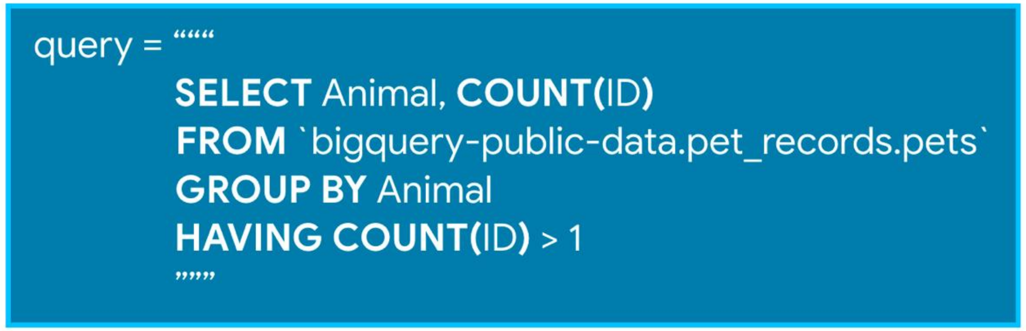
```
query = """
        SELECT Animal, COUNT(ID)
        FROM `bigquery-public-data.pet_records.pets`
        GROUP BY Animal
        """
```



| Animal | f0_ |
|---|---|
| Rabbit | 1 |
| Dog | 1 |
| Cat | 2 |

## GROUP BY… HAVING

- The HAVING clause was used because the WHERE keyword cannot be used to filter the returned values from aggregate functions.
- HAVING is used in combination with GROUP BY to ignore groups that don't meet certain criteria. So this query, for example, will only include groups that have more than one ID in them.



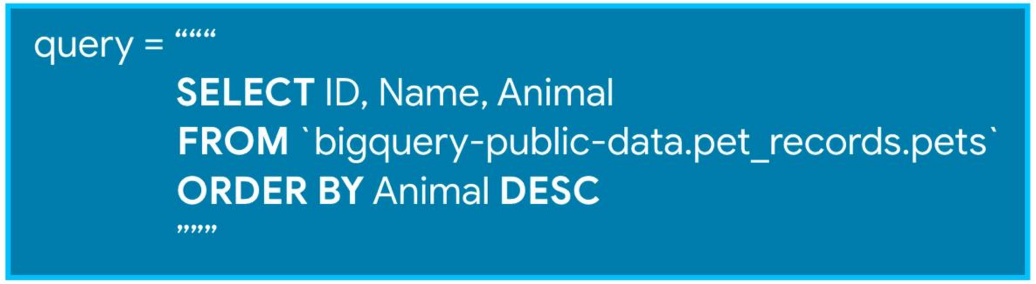
```
query = """
        SELECT Animal, COUNT(ID)
        FROM `bigquery-public-data.pet_records.pets`
        GROUP BY Animal
        HAVING COUNT(ID) > 1
        """
```

| Animal | f0_ |
|---|---|
| Cat | 2 |

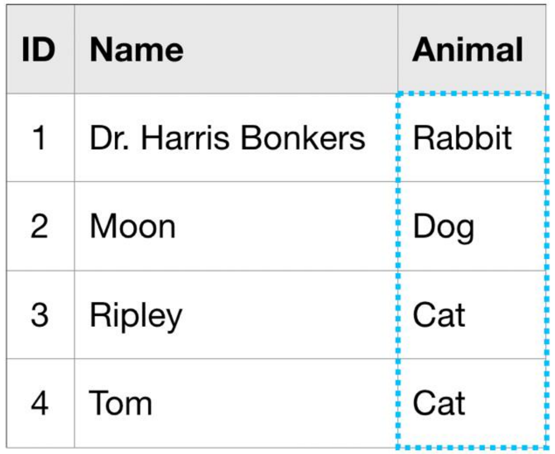
# ORDER BY

- The ORDER BY keyword is used to sort the result-set in ascending or descending order.
- The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

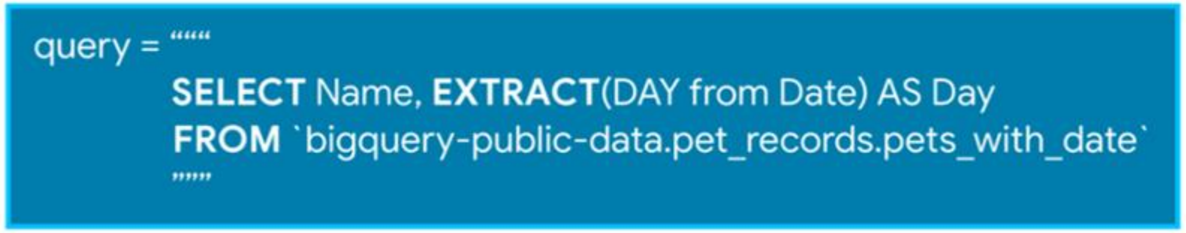


```
query = """
        SELECT ID, Name, Animal
        FROM `bigquery-public-data.pet_records.pets`
        ORDER BY Animal DESC
        """
```

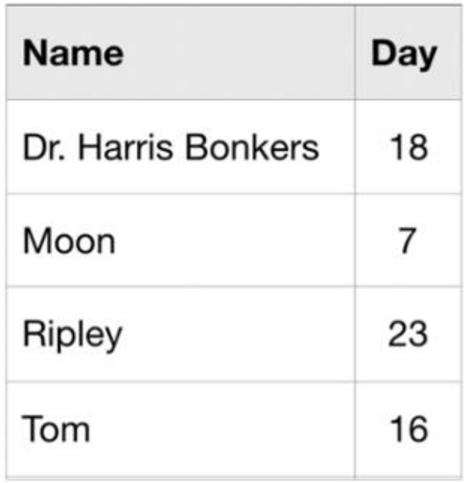


| ID | Name | Animal |
|---|---|---|
| 1 | Dr. Harris Bonkers | Rabbit |
| 2 | Moon | Dog |
| 3 | Ripley | Cat |
| 4 | Tom | Cat |

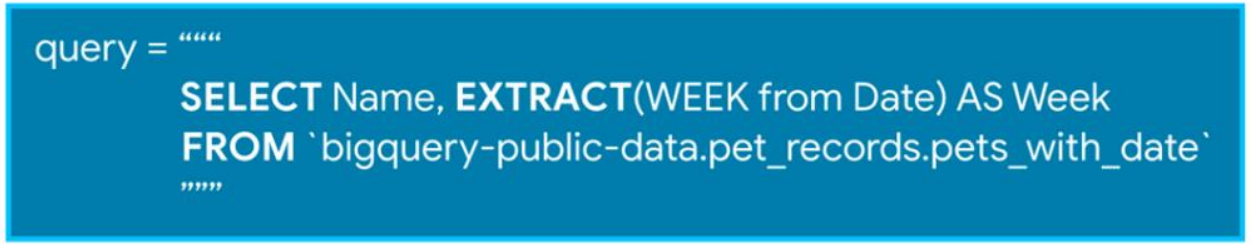
# DATE



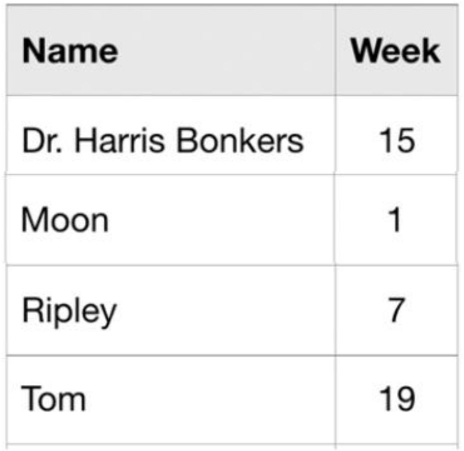
```
query = """
        SELECT Name, EXTRACT(DAY from Date) AS Day
        FROM `bigquery-public-data.pet_records.pets_with_date`
        """
```



| Name | Day |
|---|---|
| Dr. Harris Bonkers | 18 |
| Moon | 7 |
| Ripley | 23 |
| Tom | 16 |



```
query = """
        SELECT Name, EXTRACT(WEEK from Date) AS Week
        FROM `bigquery-public-data.pet_records.pets_with_date`
        """
```



| Name | Week |
|---|---|
| Dr. Harris Bonkers | 15 |
| Moon | 1 |
| Ripley | 7 |
| Tom | 19 |

# Advanced Syntax: AS

- SQL aliases are used to give a table, or a column in a table, a temporary name.
- An alias only exists for the duration of that query.
- An alias is created with the AS keyword.

```
SELECT column_name AS alias_name
FROM table_name;

SELECT column_name(s)
FROM table_name AS alias_name;
```

```
query = """
        SELECT Animal, COUNT(ID) AS Number
        FROM `bigquery-public-data.pet_records.pets`
        GROUP BY Animal
        """
```

| Animal | Number |
|---|---|
| Rabbit | 1 |
| Dog | 1 |
| Cat | 2 |

# Advanced Syntax: WITH...AS

A **common table expression** (or **CTE**) is a temporary table that you return within your query. CTEs are helpful for splitting your queries into readable chunks, and you can write queries against them.

```
query = """
          WITH Seniors AS
          (
             SELECT ID, Name
             FROM `bigquery-public-data.pet_records.pets`
             WHERE Years_old > 5
          )
                 This query is incomplete. More coming soon!
          """
```

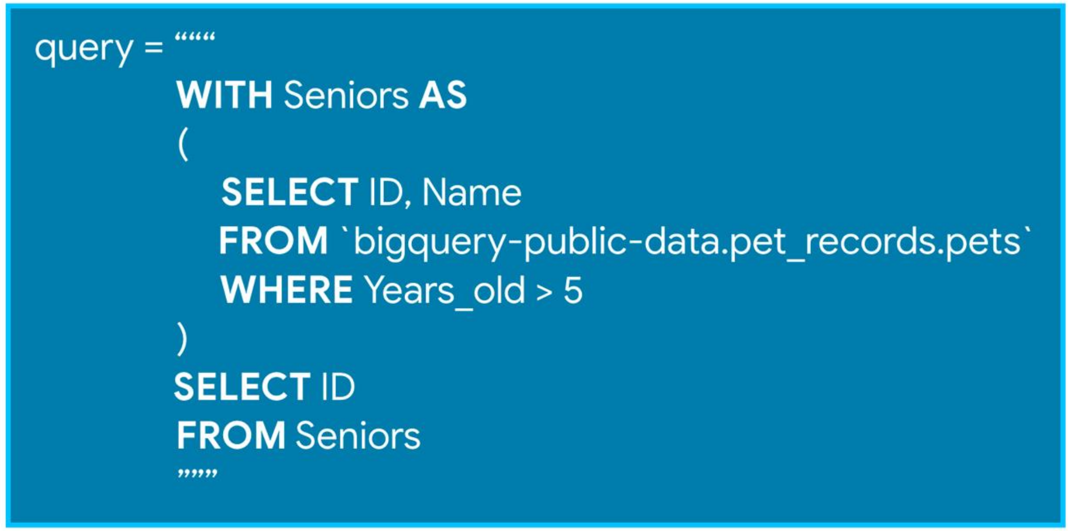
| ID | Name |
|---|---|
| 2 | Moon |
| 4 | Tom |

This is a **CTE** named `Seniors`.
*(It is not returned by the query.)*

While this incomplete query above won't return anything, it creates a CTE that we can then refer to (as Seniors) while writing the rest of the query.

## Advanced Syntax: WITH…AS

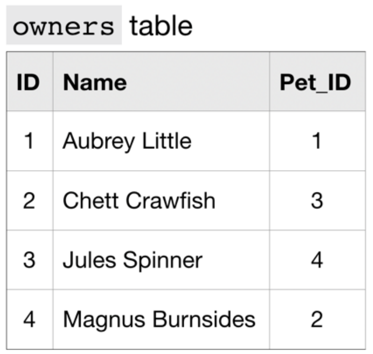
We can finish the query by pulling the information that we want from the CTE. The complete query below first creates the CTE, and then returns all of the IDs from it.



```
query = """
        WITH Seniors AS
        (
            SELECT ID, Name
            FROM `bigquery-public-data.pet_records.pets`
            WHERE Years_old > 5
        )
        SELECT ID
        FROM Seniors
        """
```

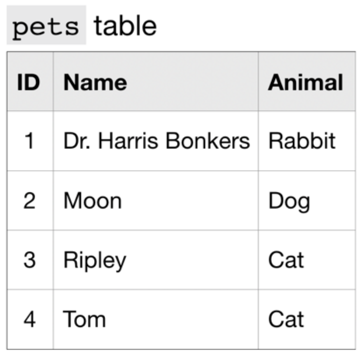
| ID |
|---|
| 2 |
| 4 |

# Advanced Syntax: JOIN



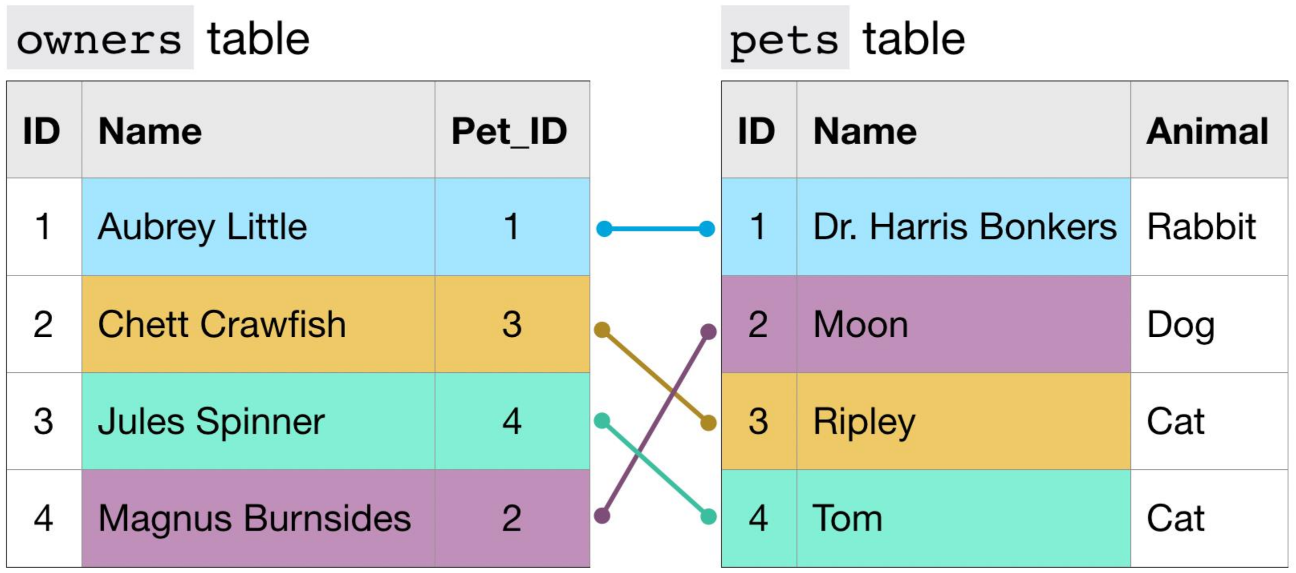
`owners` table

| ID | Name | Pet_ID |
|---|---|---|
| 1 | Aubrey Little | 1 |
| 2 | Chett Crawfish | 3 |
| 3 | Jules Spinner | 4 |
| 4 | Magnus Burnsides | 2 |



`pets` table

| ID | Name | Animal |
|---|---|---|
| 1 | Dr. Harris Bonkers | Rabbit |
| 2 | Moon | Dog |
| 3 | Ripley | Cat |
| 4 | Tom | Cat |



`owners` table

| ID | Name | Pet_ID |
|---|---|---|
| 1 | Aubrey Little | 1 |
| 2 | Chett Crawfish | 3 |
| 3 | Jules Spinner | 4 |
| 4 | Magnus Burnsides | 2 |

`pets` table

| ID | Name | Animal |
|---|---|---|
| 1 | Dr. Harris Bonkers | Rabbit |
| 2 | Moon | Dog |
| 3 | Ripley | Cat |
| 4 | Tom | Cat |

**Dr. Harris Bonkers** is owned by **Aubrey Little**.

**Moon** is owned by **Magnus Burnsides**.

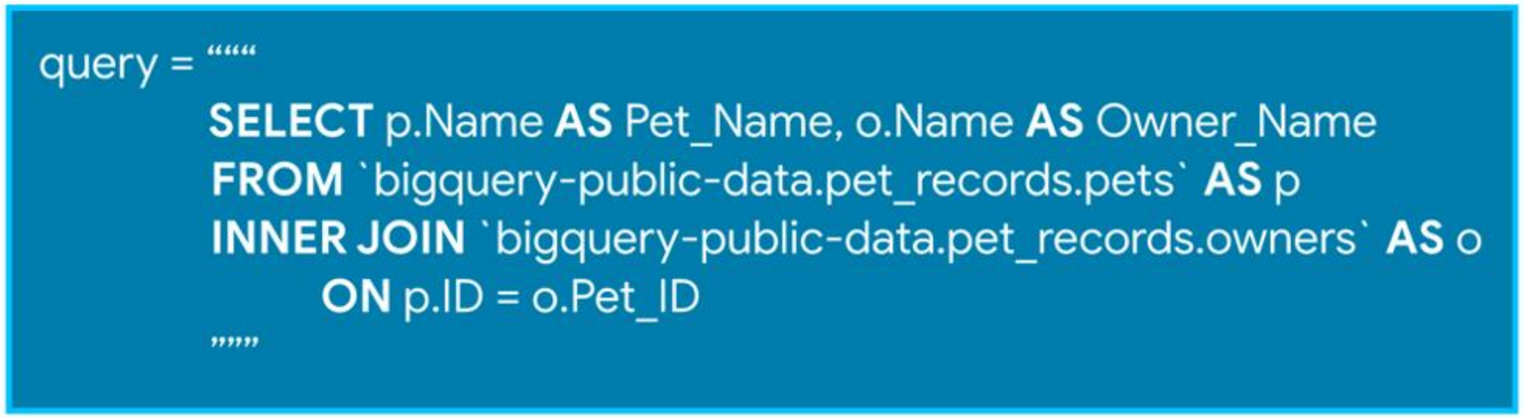
**Ripley** is owned by **Chett Crawfish**.

**Tom** is owned by **Jules Spinner**.

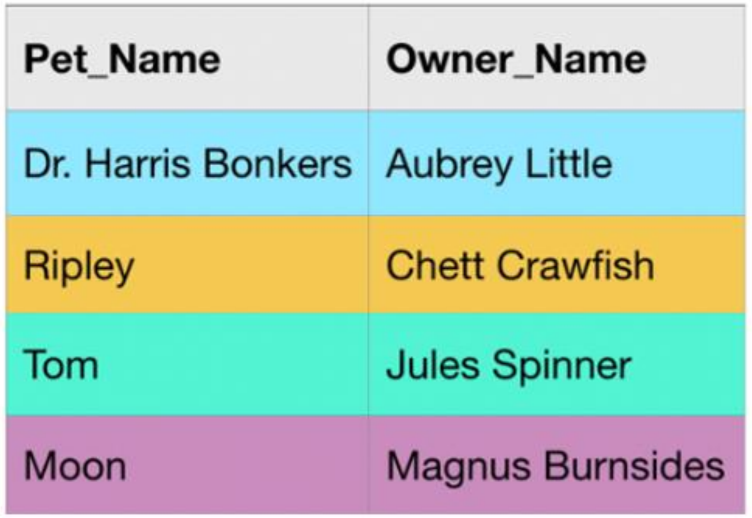
## Advanced Syntax: JOIN

- (INNER) JOIN: Returns records that have matching values in both tables

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```



```
query = """
        SELECT p.Name AS Pet_Name, o.Name AS Owner_Name
        FROM `bigquery-public-data.pet_records.pets` AS p
        INNER JOIN `bigquery-public-data.pet_records.owners` AS o
              ON p.ID = o.Pet_ID
        """
```



| Pet_Name | Owner_Name |
|---|---|
| Dr. Harris Bonkers | Aubrey Little |
| Ripley | Chett Crawfish |
| Tom | Jules Spinner |
| Moon | Magnus Burnsides |

# Write readable and maintainable Query

## 1. Use Uppercase for the Keywords

***Avoid***

```
select id, name from company.customers
```

***Prefer***

```
SELECT id, name FROM company.customers
```

## Write readable and maintainable Query

### 2. Use Snake Case for the schemas, tables, columns

***Avoid***

**nbVisit**

***Prefer***

**nb_visit**

## Write readable and maintainable Query

3. **Formatting: Carefully use Indentation & White spaces**

4. **Meaningful names based on your own conventions**

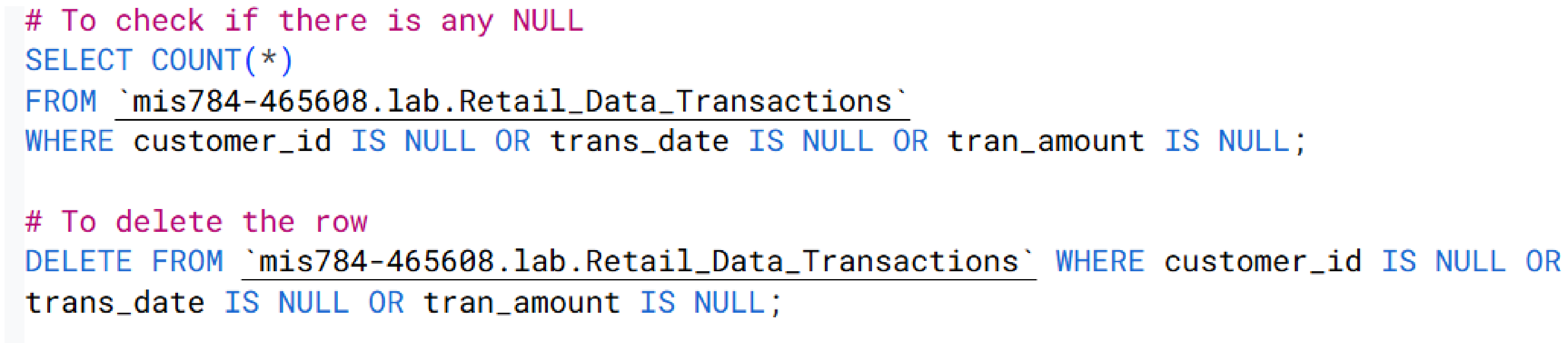
5. **Write useful comments… but not too much**
   - # this is a function for aggregation

## Data Cleansing

The `IS NULL` operator is used to check whether a value is null or not.

To delete rows that meet the same condition as the SELECT statement, we use the DELETE statement.

Example: The marketing team needs to clean up their customer database to ensure they have accurate contact information for all customers. We want to identify missing customer data in our database.



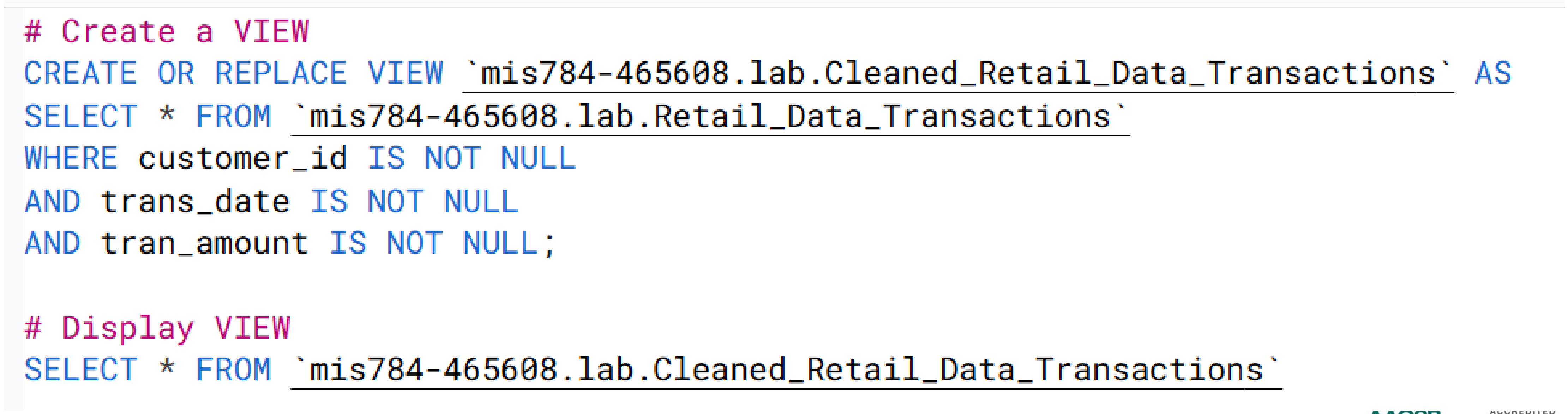
```
# To check if there is any NULL
SELECT COUNT(*)
FROM `mis784-465608.lab.Retail_Data_Transactions`
WHERE customer_id IS NULL OR trans_date IS NULL OR tran_amount IS NULL;

# To delete the row
DELETE FROM `mis784-465608.lab.Retail_Data_Transactions` WHERE customer_id IS NULL OR
trans_date IS NULL OR tran_amount IS NULL;
```

## Data Cleansing with VIEWS

In BigQuery Sandbox, we can’t use the DELETE command because it needs a Billing Account. Instead, we can use VIEW to clean the data without changing the original table.

**VIEW** is a virtual table based on the result of a SELECT query. It doesn’t store data itself — it just shows a dynamic result whenever you query it.



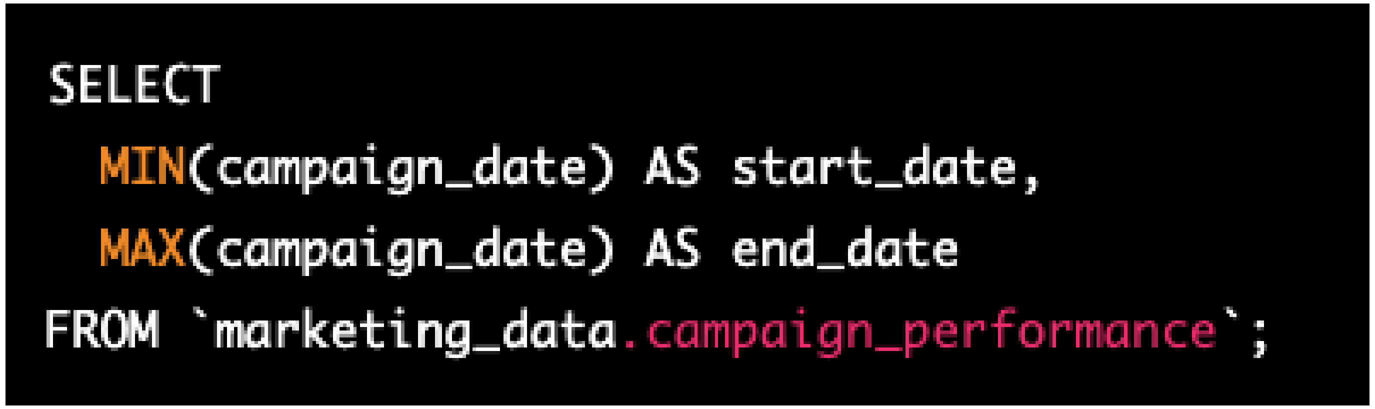
```
# Create a VIEW
CREATE OR REPLACE VIEW `mis784-465608.lab.Cleaned_Retail_Data_Transactions` AS
SELECT * FROM `mis784-465608.lab.Retail_Data_Transactions`
WHERE customer_id IS NOT NULL
AND trans_date IS NOT NULL
AND tran_amount IS NOT NULL;

# Display VIEW
SELECT * FROM `mis784-465608.lab.Cleaned_Retail_Data_Transactions`
```

## DATETIME

- The MIN and MAX functions are used to find the minimum and maximum values in a column, respectively.

Example: The marketing team wants to review the performance of their latest campaign and need to know the exact date range to analyze the data. We want to know the date range for our marketing campaign.

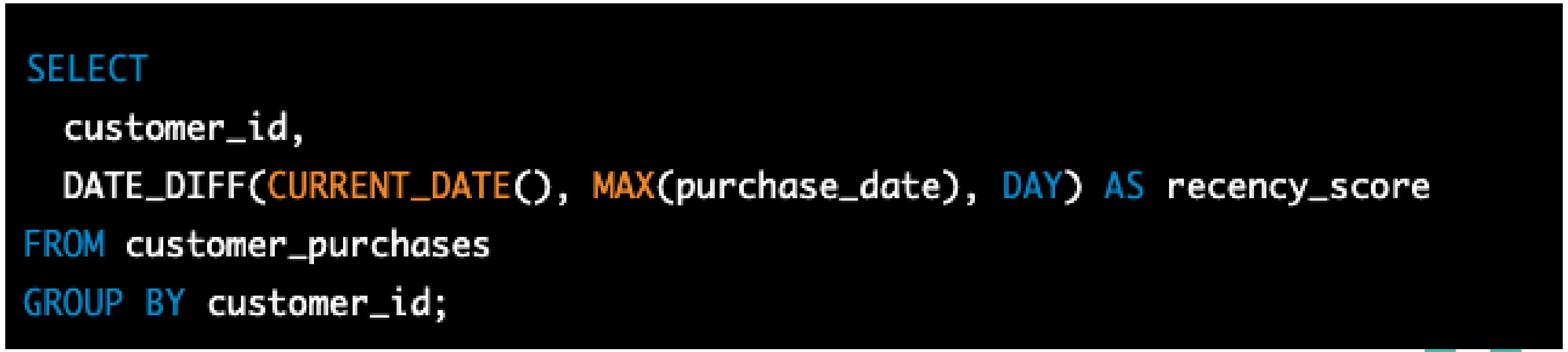


```
SELECT
  MIN(campaign_date) AS start_date,
  MAX(campaign_date) AS end_date
FROM `marketing_data.campaign_performance`;
```

## DATETIME

- The DATE_DIFF function is used to calculate the difference between two dates in days, months, or years.
- The CURRENT_DATE function is used to get the current date.

Example: The marketing team wants to review the performance of their latest campaign and need to calculate the number of days between the order date and today's date for each order.



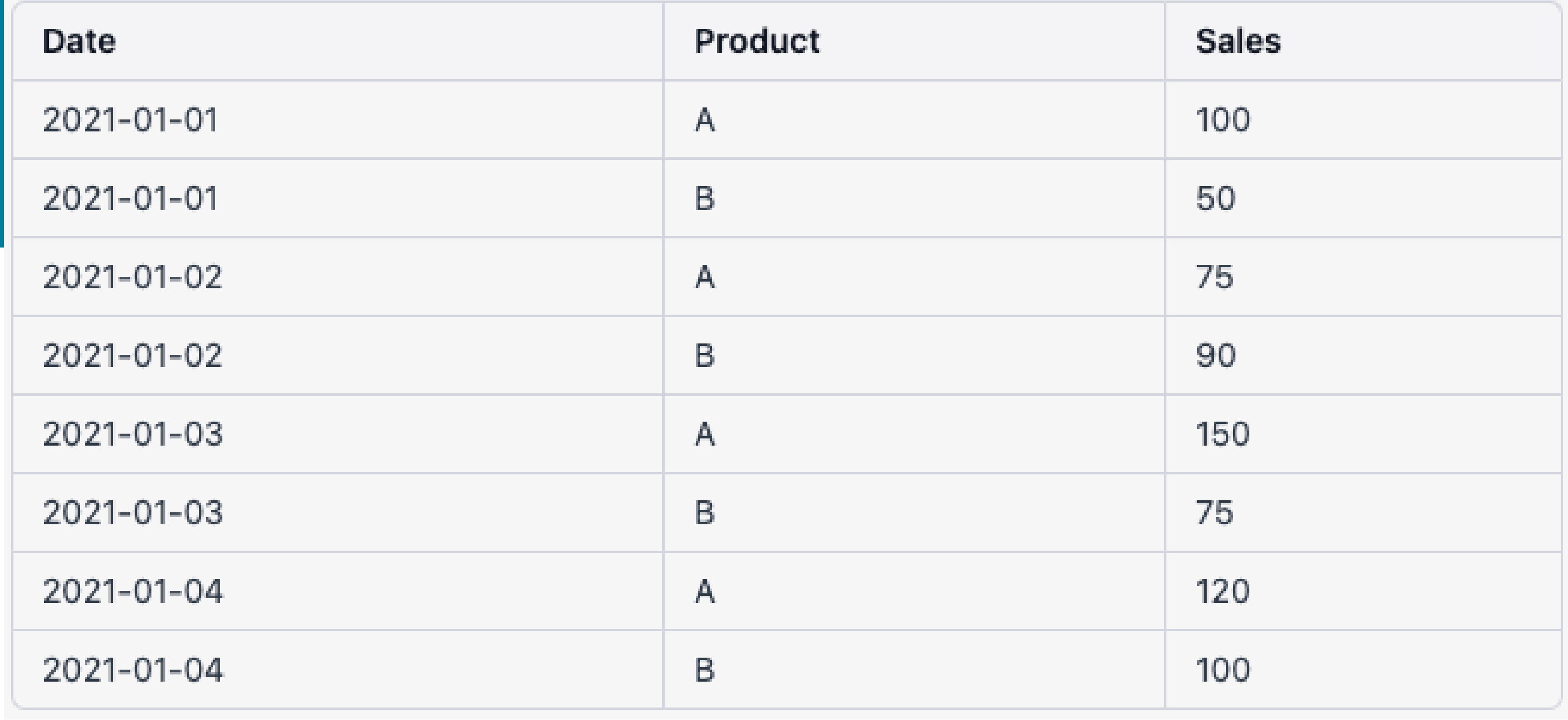
```
SELECT
  customer_id,
  DATE_DIFF(CURRENT_DATE(), MAX(purchase_date), DAY) AS recency_score
FROM customer_purchases
GROUP BY customer_id;
```

## Partition: NTILE()

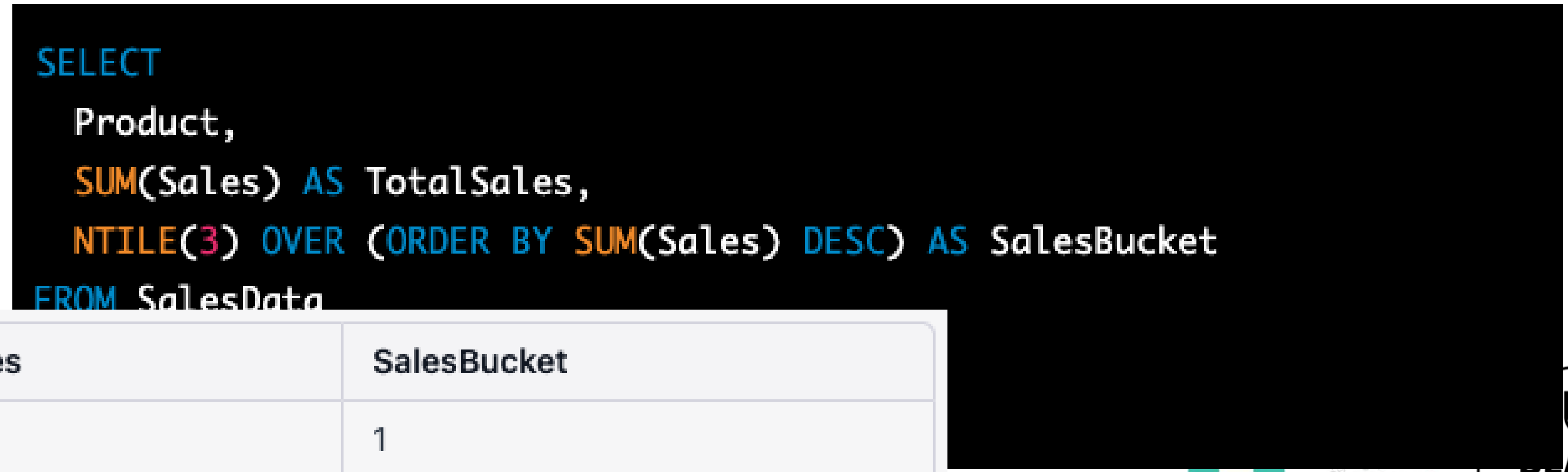
- **The NTILE function** in SQL is used to divide a result set into a specified number of groups or buckets, each containing an equal number of rows.
- It assigns a unique value to each row, indicating which group or bucket it belongs to.
- For example, if we use NTILE(4), the result set will be divided into 4 groups, with each group having roughly an equal number of rows.
- The rows in the first group will be assigned the value 1, the rows in the second group will be assigned the value 2, and so on.
- **The NTILE function can be used with the OVER clause to apply the function to a specific column in a table or result set**: to specify the ordering of the rows and the window or partition over which the function operates.

## Partition: NTILE()

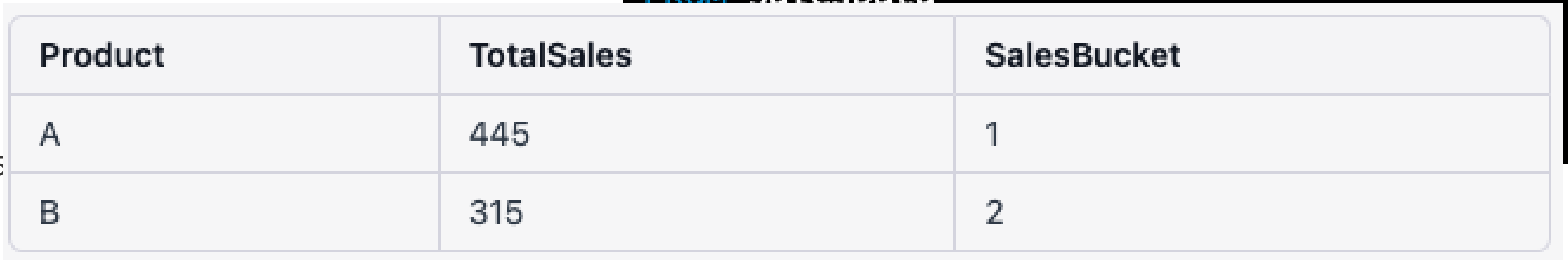
- If we want to divide the sales data into 3 buckets based on the total sales for each product, we can use the NTILE function.



| Date | Product | Sales |
|---|---|---|
| 2021-01-01 | A | 100 |
| 2021-01-01 | B | 50 |
| 2021-01-02 | A | 75 |
| 2021-01-02 | B | 90 |
| 2021-01-03 | A | 150 |
| 2021-01-03 | B | 75 |
| 2021-01-04 | A | 120 |
| 2021-01-04 | B | 100 |



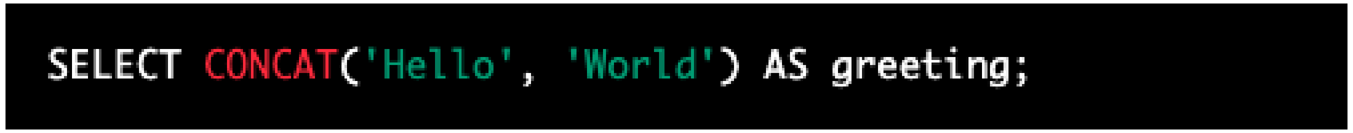
```sql
SELECT
  Product,
  SUM(Sales) AS TotalSales,
  NTILE(3) OVER (ORDER BY SUM(Sales) DESC) AS SalesBucket
FROM SalesData
```



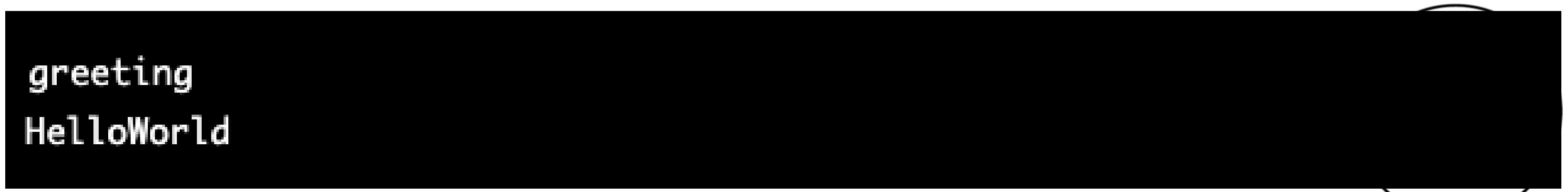
| Product | TotalSales | SalesBucket |
|---|---|---|
| A | 445 | 1 |
| B | 315 | 2 |

## CONCAT Function

- The CONCAT function is used to concatenate two or more strings into a single string.
- It accepts two or more string arguments and returns a single string that is the concatenation of those arguments.
- The syntax of CONCAT: CONCAT(string1, string2, ...)



```sql
SELECT CONCAT('Hello', 'World') AS greeting;
```



```
greeting
HelloWorld
```