

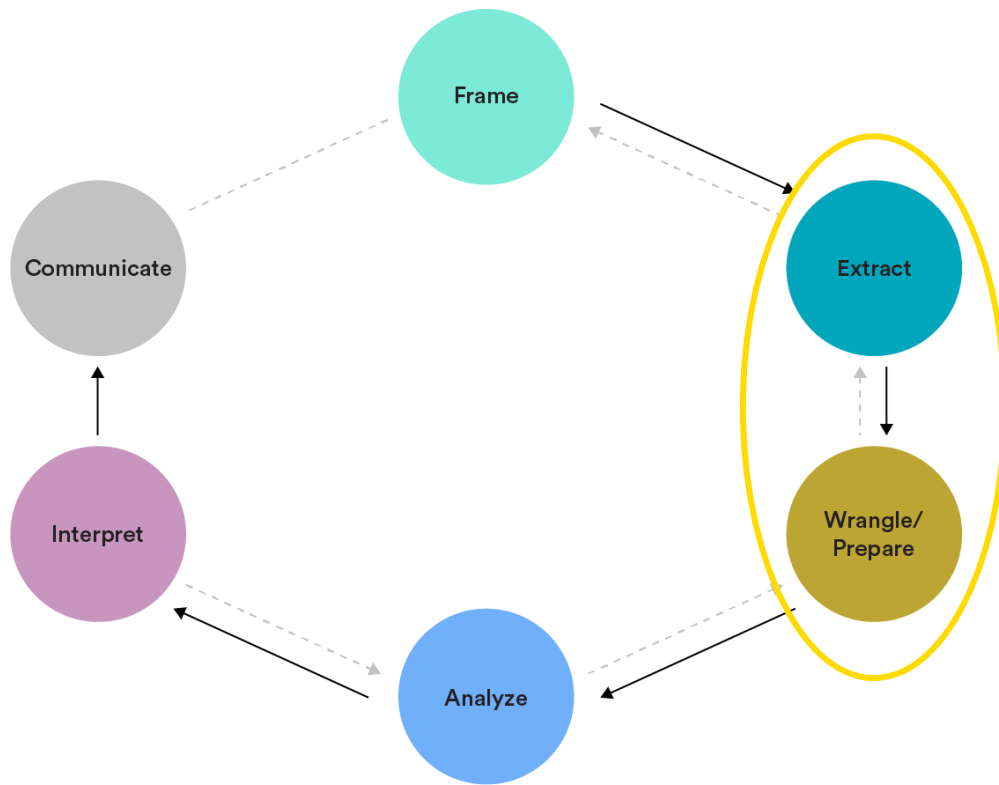
Data Analytics

Grouping in SQL

Where We Are in the DA Workflow

Extract: Select, import, and clean relevant data.

Wrangle/prepare: Clean and prepare relevant data.



Our Learning Goals

What you'll learn in class today...

- Work with **CASE** to handle multiple conditions.
- Practice writing aggregate functions: **MIN**, **MAX**, **SUM**, **AVG**, and **COUNT**.
- Use SQL commands such as **GROUP BY** and **HAVING** to group and filter data.



What If...

Let's take a look at this query from the last lesson:

```
SELECT *  
FROM products  
LIMIT 100;
```

What if, from the list of 100 products, we want to **break our product prices into groups (free, cheap, affordable, expensive)**?

To get the result, your query must include *multiple* conditions: free, cheap, affordable, and expensive.



Grouping in SQL

Handling Multiple Conditions With CASE



What Are CASE Statements?

CASE statements group data into *categories* or *classifications*. They **go through multiple conditions and return a value when the *first* condition is met.**

- When a condition is true, CASE will stop reading and return the result.
- If no conditions are true, it will return the value in the ELSE clause.



CASE Syntax

```
SELECT column,  
    CASE  
        WHEN condition THEN result  
        WHEN condition THEN result  
        ELSE condition  
    END AS output_name  
FROM table;
```

CASE syntax in plain words:

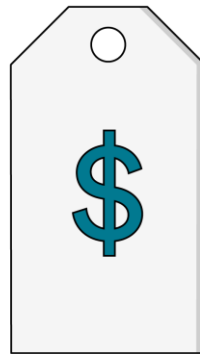
- SELECT takes the column on which you want to run **CASE**.
- WHEN **<condition is true>**.
- THEN **<what to return as a value for that row>**.
- ELSE (optional condition).
- **END AS** <header title for the new column you just made>.
- FROM <table>.



Handling Multiple Conditions With CASE

Let's try classifying the discounts in the Superstore data set into these groups:

- Free (100%)
- High (25–99%)
- Low (1–25%)
- None (0%)



Using CASE statements, let's write this out.



Handling Multiple Conditions With CASE (Cont.)

Below is our query with a CASE statement. **Remember:** SQL will *only* return values that meet these conditions, but we can also add an “other” category using ELSE.

```
SELECT
  discount,
  CASE
    WHEN discount = 1 THEN 'Free'
    WHEN discount BETWEEN .25 AND 1 THEN 'High'
    WHEN discount = 0 THEN 'None'
    WHEN discount < .25 THEN 'Low'
  END AS discount_level
FROM orders
LIMIT 50;
```



Group Exercise:

Handling Multiple Conditions With CASE

10 minutes



Work with your group to create a CASE statement that groups **orders by whether or not they had a positive profit.**





Group Exercise:

How Did It Go? | Solution

How did it go? Your query should look like the following:

```
SELECT profit,  
       CASE WHEN profit > 0 THEN 'Positive'  
            ELSE 'Negative' END AS profit_level  
FROM orders  
LIMIT 100;
```

Grouping in SQL

GROUP BY and HAVING



Clauses for Aggregate Functions

Aggregate functions are also used in these clauses:

- **GROUP BY** indicates the dimensions you want to group your data by (e.g., a category that you wish to sort into subgroups).
- **HAVING** is used to filter measures you've aggregated (e.g., to filter a SUM over a certain value).

Where They Live in a Query

SELECT picks the columns.

FROM points to the table.

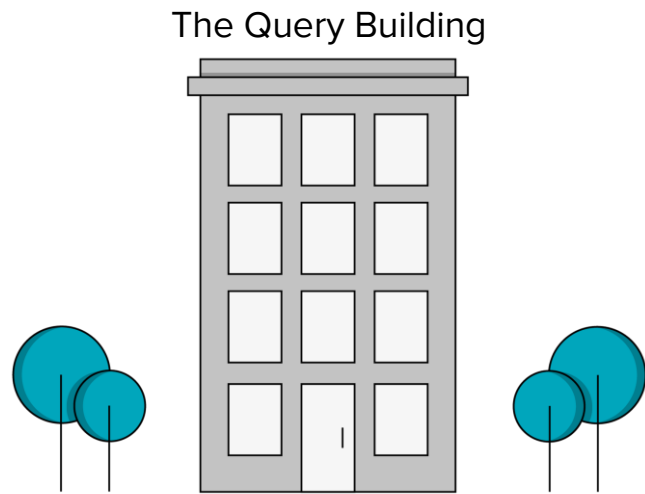
WHERE puts filters on rows.

GROUP BY aggregates multiple rows, based on one or more aggregate functions (MIN, AVG, etc.).

HAVING filters aggregated values *after* they have been grouped.

ORDER BY sorts the results.

LIMIT limits results to the first **n** rows.





What Will the Code Look Like?

Let's look at an example using a table called "People" with the columns "Gender" and "Height." From this table, we want to:

- Find the average height of people by gender.

What will our query look like?

```
SELECT gender, AVG(height) AS avg_height  
FROM people  
GROUP BY gender;
```

People	
Gender	Height
Female	5.8
Male	5.9
Non-binary	5.5
Female	5.3
...	...
...	...



What Will the Code Look Like? (Cont.)

Now, we want to limit our analysis to only those people taller than three feet:

- Find the average height of people taller than three feet by gender.

What will our query look like?

```
SELECT gender, AVG(height) AS avg_height  
FROM people  
WHERE height > 3  
GROUP BY gender;
```

People	
Gender	Height
Female	5.8
Male	5.9
Non-binary	5.5
Female	5.3
...	...
...	...



What Will the Code Look Like? (Cont.)

Finally, we want to only return genders that have an average height of more than 5.5 feet.

- Determine which of those people have an average height greater than 5.5 feet tall, sorted by gender.

What will our query look like?

```
SELECT gender, AVG(height) AS avg_height  
FROM people  
WHERE height > 3  
GROUP BY gender  
HAVING AVG(height)>5.5;
```

People	
Gender	Height
Female	5.8
Male	5.9
Non-binary	5.5
Female	5.3
...	...
...	...



What each part of the query does:

```
SELECT gender, AVG(height)
FROM people
WHERE height >3
GROUP BY gender
HAVING AVG(height) >5.5
ORDER BY gender;
```

- SELECT picks the columns.
- FROM determines and filters rows.
- WHERE adds more filters on those rows.
- **GROUP BY** combines those rows into groups.
- **HAVING** filters groups.
- ORDER BY arranges the remaining rows/groups.



With your partner, build the following queries with GROUP BY and HAVING:

```
SELECT segment,  
        COUNT(*) AS  
num_customers  
FROM customers  
GROUP BY segment
```

```
SELECT segment,  
        COUNT(*) AS  
num_customers  
FROM customers  
GROUP BY segment  
HAVING COUNT(*) > 300
```

- How many results do you get with the **GROUP BY** statement?
- How many results do you get with the **HAVING** statement included?



Aggregating Data With GROUP BY and HAVING

Superstore wants an order discount analysis to identify average order qty and sales amount by discount level. To write our query, we'll use:

1. **WHERE** to filter discount levels greater than 15%.
2. **GROUP BY** in our query to aggregate qty and sales.
3. **HAVING** to filter discount levels above an average sales threshold.

```
SELECT discount, ROUND(AVG(quantity), 2) AS quantity, AVG(sales)::money as "Average Sales"  
FROM orders  
WHERE 1 > 0.15  
GROUP BY discount  
HAVING AVG(sales) > 500  
ORDER BY 3 DESC
```



Solo Exercise: Over to You

5 Minutes



Use the starter code below to find **the number of products by sub category** in the Superstore data set.

```
SELECT sub_category, (aggregate of rows)
FROM products
GROUP BY sub_category;
```





Solo Exercise:

Over to You | An Extra Challenge

5 minutes



Once you have filled in the parts of the starter code, try the following:

- Include only products with “computer” or “color” (case-insensitive) in the name.
- Further refine to those that *have* an aggregate 100 or more products.
- Alias your aggregate column to “count_of_products.”
- Sort the results by the “count_of_products” column.
- Limit the output to the first 10 rows.

Ask for help if you have questions or need a hint.





Solo Exercise:

How Did It Go? | Solution

Here is what your end query might look like:

```
SELECT sub_category, COUNT(*) as count_of_products
FROM products
WHERE
    product_name ILIKE '%computer%'
    OR product_name ILIKE '%color%'
GROUP BY 1
HAVING COUNT(*) > 100
ORDER BY 2 DESC
LIMIT 10
;
```

— Combining Group By and Case



Using CASE and GROUP BY

Now, we're able to look at the total number of sales made in each of our discount categories from earlier:

```
SELECT CASE
  WHEN discount = 1 THEN 'Free'
  WHEN discount BETWEEN .25 AND 1 THEN 'High'
  WHEN discount = 0 THEN 'None'
  WHEN discount < .25 THEN 'Low'
  END AS discount_level, COUNT(*) AS num_sales
FROM orders
GROUP BY 1
;
```



Solo Exercise: Over to You

5 minutes



Use the starter code below to find **the number of orders by profit category** in the Superstore data set.

```
SELECT CASE WHEN profit > 0 THEN 'Positive'
          ELSE 'Negative' END AS profit_level,
       (aggregate of rows)
FROM ORDERS
;
```





Solo Exercise:

How Did It Go? | Solution

```
SELECT CASE WHEN profit > 0 THEN 'Positive'
        ELSE 'Negative' END AS profit_level,
        COUNT(*) as num_sales
FROM ORDERS
GROUP BY 1
;
```



Solo Exercise: Over to You

5 minutes



We want to dig into how many of our sales have been for more than 1 item in a single sale (i.e., quantity 2+).

First, create a CASE statement that will group orders by their quantity, then count up the number of orders associated with your groupings.





Solo Exercise:

How Did It Go? | Solution

Here is what your end query might look like:

```
SELECT CASE
  WHEN quantity>1 THEN 'Multiple'
  ELSE 'Single' END AS quantity,
  COUNT(*) AS num_sales
FROM orders
GROUP BY 1
;
```

— Wrapping Up

Recap

In today's class, we...

- Worked with **CASE** to handle if/then logic and apply multiple conditions.
- Practiced writing aggregate functions: **MIN**, **MAX**, **SUM**, **AVG**, and **COUNT**.
- Used SQL commands such as **GROUP BY** and **HAVING** to group and filter data.

Looking Ahead

Homework:

Up Next: JOINS



Additional Resources

- SQL **HAVING** clause overview: <https://goo.gl/Je3M85>
- “Difference between WHERE, GROUP BY, and HAVING clauses,” by Manoj Pandey: <https://goo.gl/cNCtBa>

