Intermediate SQL

SQL Aggregate Functions

- `SUM` adds together all the values in a particular column.
- `MIN` returns the lowest value in a particular column
- `MAX` returns the highest value in a particular column
- `AVG` calculates the average of a group of selected values.
- `COUNT` counts how many rows are in a particular column.

Table: `product_spend` Sample Data

category	product	user_id	spend	transaction_date
appliance	washing machine	123	219.80	03/02/2022 11:00:00
electronics	vacuum	178	152.00	04/05/2022 10:00:00
electronics	wireless headset	156	249.90	07/08/2022 10:00:00
electronics	vacuum	145	189.00	07/15/2022 10:00:00
electronics	computer mouse	195	45.00	07/01/2022 11:00:00
appliance	refrigerator	165	246.00	12/26/2021 12:00:00
appliance	refrigerator	123	299.99	03/02/2022 11:00:00
...	...	...	...	...

Let's dive into some practical examples of using aggregate functions to analyze this Amazon data.

Counting Number of Orders with SQL

The `COUNT()` function enables you to count the total number of rows in a table. We can use the `COUNT()` function on the `user_id` column as follows:

```
SELECT COUNT(user_id)
FROM product_spend;
```

Results

count
15

Calculating Total Sales with SQL SUM()

```
SELECT SUM(spend)
FROM product_spend;
```

Results

sum
2142.76

Finding the Average Price with SQL AVG()

```
SELECT AVG(spend)
FROM product_spend;
```

Finding the Minimum with SQL MIN()

```
SELECT MIN(spend)
FROM product_spend;
```

Finding the Maximum with SQL MAX()

```
SELECT MAX(spend)
FROM product_spend;
```

SQL GROUP BY

```
SELECT
    category,
    SUM(spend)
FROM product_spend
GROUP BY category;
```

The output of that `GROUP BY` query yields this result:

category	sum
electronics	1007.54
appliance	1135.22

GROUP BY two columns

```
SELECT
    ticker,
    EXTRACT(YEAR FROM date) AS year,
    ROUND(AVG(open),2) AS avg_open
FROM stock_prices
GROUP BY ticker, year
ORDER BY year DESC;
```

Here's a sample of that output:

ticker	year	avg_open
NFLX	2023	364.91

META	2023	220.35
AMZN	2023	109.27
MSFT	2023	291.96
AAPL	2023	171.21
GOOG	2023	108.10
MSFT	2022	276.78
AAPL	2022	151.56
META	2022	193.06
...	...	...

What's the difference between GROUP BY and ORDER BY?

People sometimes get confused between `GROUP BY` and `ORDER BY` because both commands have `BY` in them, but their actual function is quite different!

ORDER BY helps you output your rows in a specific order, such as alphabetically on some text column, or from smallest to biggest, for some text column.

`GROUP BY`, as you saw earlier, is all about grouping your data into categories! Because they are quite different commands, it's absolutely possible for a query to have both GROUP BY and ORDER BY, with GROUP BY coming first!

Here's a query that uses both GROUP BY and ORDER BY:

```
SELECT
    ticker,
    AVG(open) AS avg
FROM stock_prices
GROUP BY ticker
ORDER BY avg DESC;
```

SQL HAVING

Here's an example `HAVING` query that finds all FAANG stocks with an average share open price of more than $200:

```
SELECT ticker, AVG(open)
FROM stock_prices
```

```
GROUP BY ticker
HAVING AVG(open) > 200;
```

The above `HAVING` query would yield the following result:

ticker	avg
NFLX	420.6945454545454545
META	242.9279545454545455
MSFT	254.0772727272727273

WHERE vs. HAVING

The difference between WHERE vs. HAVING is a common conceptual SQL interview question, so we figured we'd cover it a bit more explicitly: WHERE filters on values in individual rows, versus HAVING filters values aggregated from groups of rows.

Here's a summary table on the difference between WHERE & HAVING:

	WHERE	HAVING
When It Filters	Values BEFORE Grouping	Values AFTER Grouping
Operates On Data From	Individual Rows	Aggregated Values from Groups of Rows
Example	SELECT username, followers FROM instagram_data WHERE followers > 1000;	SELECT country FROM instagram_data GROUP BY country HAVING AVG(followers) > 100;

Can `HAVING` be used without `GROUP BY`?

It's technically possible to have `HAVING` without `GROUP BY` in SQL, and in this case `HAVING` operates the same as `WHERE`. However, practically speaking, it's super weird to use `HAVING` in this context. For the most part, you should just use a `WHERE` clause if trying to filter on non-aggregated values.

Where should HAVING be placed in a query?

It's CRUCIAL to write clauses in the correct order, otherwise, your SQL query won't run!

SQL Distinct

The `DISTINCT` SQL command is used in conjunction with the `SELECT` statement to return only distinct (different) values. Here's an example DISTINCT SQL query used to find only unique names of pharmaceutical manufacturers:

```
SELECT DISTINCT manufacturer
FROM pharmacy_sales;
```

DISTINCT For Data Exploration

DISTINCT can be particularly helpful when exploring a new data set. In many real-world scenarios, you will generally end up writing several exploratory `SELECT DISTINCT` queries in order to figure out what data you have access too, and how you might want to group or filter the data.

DISTINCT With Two Columns

If you include two (or more) columns in a SELECT DISTINCT clause, your results will contain all of the unique pairs of those two columns.

For example, imagine you worked at stock trading app Robinhood and had access to their trades dataset. Here's a SQL query that uses DISTINCT on two columns – user_id's and trade statuses:

```
SELECT DISTINCT user_id, status
FROM trades
ORDER BY user_id;
```

Note: You only need to include DISTINCT once in your SELECT clause—you do not need to add it for each column name.