PostgreSQL – more Firstar while writing rows to the table

Redshif – is optimized to perform faster aggregation function

**LIMIT:**

SELECT \* FROM orders LIMIT 10;

1. Write a query to return the 10 earliest orders in the **orders** table. Include the id, occurred\_at, and total\_amt\_usd.

Select id, occurred\_at, total\_amt\_usd from orders ORDER BY id LIMIT 10

1. Write a query to return the top 5 **orders** in terms of largest total\_amt\_usd. Include the id, account\_id, and total\_amt\_usd.

SELECT id, account\_id, total\_amt\_usd FROM orders ORDER BY total\_amt\_usd DESC LIMIT 5;

1. Write a query to return the bottom 20 **orders** in terms of least total. Include the id, account\_id, and total.

Select id, account\_id, total from orders ORDER BY total ASC LIMIT 20;

Write a query that returns the top 5 rows from **orders** ordered according to newest to oldest, but with the largest total\_amt\_usd for each date listed first for each date.

SELECT \* FROM orders ORDER BY occurred\_at DESC, total\_amt\_usd DESC LIMIT 5;

Write a query that returns the top 10 rows from **orders** ordered according to oldest to newest, but with the smallest total\_amt\_usd for each date listed first for each date.

SELECT \* FROM orders ORDER BY occurred\_at, total\_amt\_usd LIMIT 10;

1. Filter the accounts table to include the company name, website, and the primary point of contact (primary\_poc) for **Exxon Mobil** in the **accounts** table.

SELECT name, website, primary\_poc FROM accounts WHERE name = 'Exxon Mobil';

Create a column that divides the standard\_amt\_usd by the standard\_qty to find the unit price for standard paper for each order. Limit the results to the first 10 orders, and include the id and account\_id fields.

SELECT id, account\_id, standard\_amt\_usd/standard\_qty AS unit\_price FROM orders LIMIT 10;

Write a query that finds the percentage of revenue that comes from poster paper for each order. You will need to use only the columns that end with \_usd. (Try to do this without using the total column). Include the id and account\_id fields.

SELECT id, account\_id,

poster\_amt\_usd/(standard\_amt\_usd + gloss\_amt\_usd + poster\_amt\_usd) AS post\_per

FROM orders;

**LIKE:**

1. All the companies whose names start with 'C'.

SELECT name FROM accounts WHERE name LIKE 'C%';

1. All companies whose names contain the string 'one' somewhere in the name.

SELECT name FROM accounts WHERE name LIKE '%one%';

1. All companies whose names end with 's'.

SELECT name FROM accounts WHERE name LIKE '%s';

**IN:**

1. Use the **accounts** table to find the account name, primary\_poc, and sales\_rep\_id for Walmart, Target, and Nordstrom.

SELECT name, primary\_poc, sales\_rep\_id FROM accounts WHERE name IN ('Walmart', 'Target', 'Nordstrom');

1. Use the **web\_events** table to find all information regarding individuals who were contacted via the **channel** of organic or adwords.

SELECT \* FROM web\_events WHERE channel IN ('organic', 'adwords');

**NOT:**

1. Use the **accounts** table to find the account name, primary poc, and sales rep id for all stores except Walmart, Target, and Nordstrom.

SELECT name, primary\_poc, sales\_rep\_id FROM accounts WHERE name NOT IN ('Walmart', 'Target', 'Nordstrom');

1. Use the **web\_events** table to find all information regarding individuals who were contacted via any method except using organic or adwords methods.

SELECT \* FROM web\_events WHERE channel NOT IN ('organic', 'adwords');

Use the **accounts** table to find:

1. All the companies whose names do not start with 'C'.

SELECT name FROM accounts WHERE name NOT LIKE 'C%';

1. All companies whose names do not contain the string 'one' somewhere in the name.

SELECT name FROM accounts WHERE name NOT LIKE '%one%';

1. All companies whose names do not end with 's'.

SELECT name FROM accounts WHERE name NOT LIKE '%s';

**AND and BETWEEN:**

1. Write a query that returns all the **orders** where the standard\_qty is over 1000, the poster\_qty is 0, and the gloss\_qty is 0.

SELECT \* FROM orders WHERE standard\_qty > 1000 AND poster\_qty = 0 AND gloss\_qty = 0;

1. Using the **accounts** table find all the companies whose names do not start with 'C' and end with 's'.

SELECT name FROM accounts WHERE name NOT LIKE 'C%' AND name LIKE '%s';

1. Use the **web\_events** table to find all information regarding individuals who were contacted via organic or adwords and started their account at any point in 2016 sorted from newest to oldest.

SELECT \* FROM web\_events WHERE channel IN ('organic', 'adwords') AND occurred\_at BETWEEN '2016-01-01' AND '2017-01-01' ORDER BY occurred\_at DESC;

**OR:**

1. Find list of **orders** ids where either gloss\_qty or poster\_qty is greater than 4000. Only include the id field in the resulting table.

SELECT id FROM orders WHERE gloss\_qty > 4000 OR poster\_qty > 4000;

1. Write a query that returns a list of **orders** where the standard\_qty is zero and either the gloss\_qtyor poster\_qty is over 1000.

SELECT \* FROM orders WHERE standard\_qty = 0 AND (gloss\_qty > 1000 OR poster\_qty > 1000);

1. Find all the company names that start with a 'C' or 'W', and the primary contact **contains** 'ana' or 'Ana', but it doesn't contain 'eana'.

SELECT \* FROM accounts

WHERE (name LIKE 'C%' OR name LIKE 'W%')

AND ((primary\_poc LIKE '%ana%' OR primary\_poc LIKE '%Ana%')

AND primary\_poc NOT LIKE '%eana%');

| **Statement** | **How to Use It** | **Other Details** |
| --- | --- | --- |
| SELECT | SELECT **Col1**, **Col2**, ... | Provide the columns you want |
| FROM | FROM **Table** | Provide the table where the columns exist |
| LIMIT | LIMIT **10** | Limits based number of rows returned |
| ORDER BY | ORDER BY **Col** | Orders table based on the column. Used with **DESC**. |
| WHERE | WHERE **Col > 5** | A conditional statement to filter your results |
| LIKE | WHERE **Col LIKE '%me%'** | Only pulls rows where column has 'me' within the text |
| IN | WHERE **Col IN ('Y', 'N')** | A filter for only rows with column of 'Y' or 'N' |
| NOT | WHERE **Col NOT IN ('Y', 'N')** | **NOT** is frequently used with **LIKE** and **IN** |
| AND | WHERE **Col1 > 5 AND Col2 < 3** | Filter rows where two or more conditions must be true |
| OR | WHERE **Col1 > 5 OR Col2 < 3** | Filter rows where at least one condition must be true |
| BETWEEN | WHERE **Col BETWEEN 3 AND 5** | Often easier syntax than using an **AND** |

**JOIN:**

SELECT orders.\* FROM orders

JOIN accounts

ON orders.account\_id = accounts.id;

If we want to pull only the **account name** and the dates in which that account placed an order, but none of the other columns, we can do this with the following query:

**SELECT** accounts.**name**, orders.occurred\_at **FROM** orders

**JOIN** accounts

**ON** orders.account\_id = accounts.**id**;

Query pulls all the columns from both the **accounts** and **orders** table.

**SELECT** \* **FROM** orders

**JOIN** accounts

**ON** orders.account\_id = accounts.**id**;

1. Try pulling all the data from the **accounts** table, and all the data from the **orders** table.

SELECT orders.\*, accounts.\* FROM accounts

JOIN orders

ON accounts.id = orders.account\_id;

1. Try pulling **standard\_qty**, **gloss\_qty**, and **poster\_qty** from the **orders** table, and the **website** and the **primary\_poc** from the **accounts** table.

**SELECT** orders.standard\_qty, orders.gloss\_qty,

orders.poster\_qty, accounts.website,

accounts.primary\_poc

**FROM** orders

**JOIN** accounts

**ON** orders.account\_id = accounts.**id**

If we wanted to join all three of these tables, we could use the same logic. The code below pulls all of the data from all of the joined tables.

**SELECT** \* **FROM** web\_events

**JOIN** accounts

**ON** web\_events.account\_id = accounts.**id**

**JOIN** orders

**ON** accounts.**id** = orders.account\_id

Alternatively, we can create a **SELECT** statement that could pull specific columns from any of the three tables. Again, our **JOIN** holds a table, and **ON** is a link for our **PK** to equal the **FK**.

SELECT web\_events.channel, accounts.name, orders.total

Entity Relationship Diagram**(ERD):**

A common way to view data in a database. It is also a key element to understanding how we can pull data from multiple tables.

### Primary Key (PK)

A **primary key** is a unique column in a particular table. This is the first column in each of our tables. Here, those columns are all called **id**, but that doesn't necessarily have to be the name. **It is common that the primary key is the first column in our tables in most databases.**

### Foreign Key (FK)

A **foreign key** is when we see a primary key in another table. We can see these in the previous ERD the foreign keys are provided as:

# Alias

**FROM** tablename t1

**JOIN** tablename2 t2

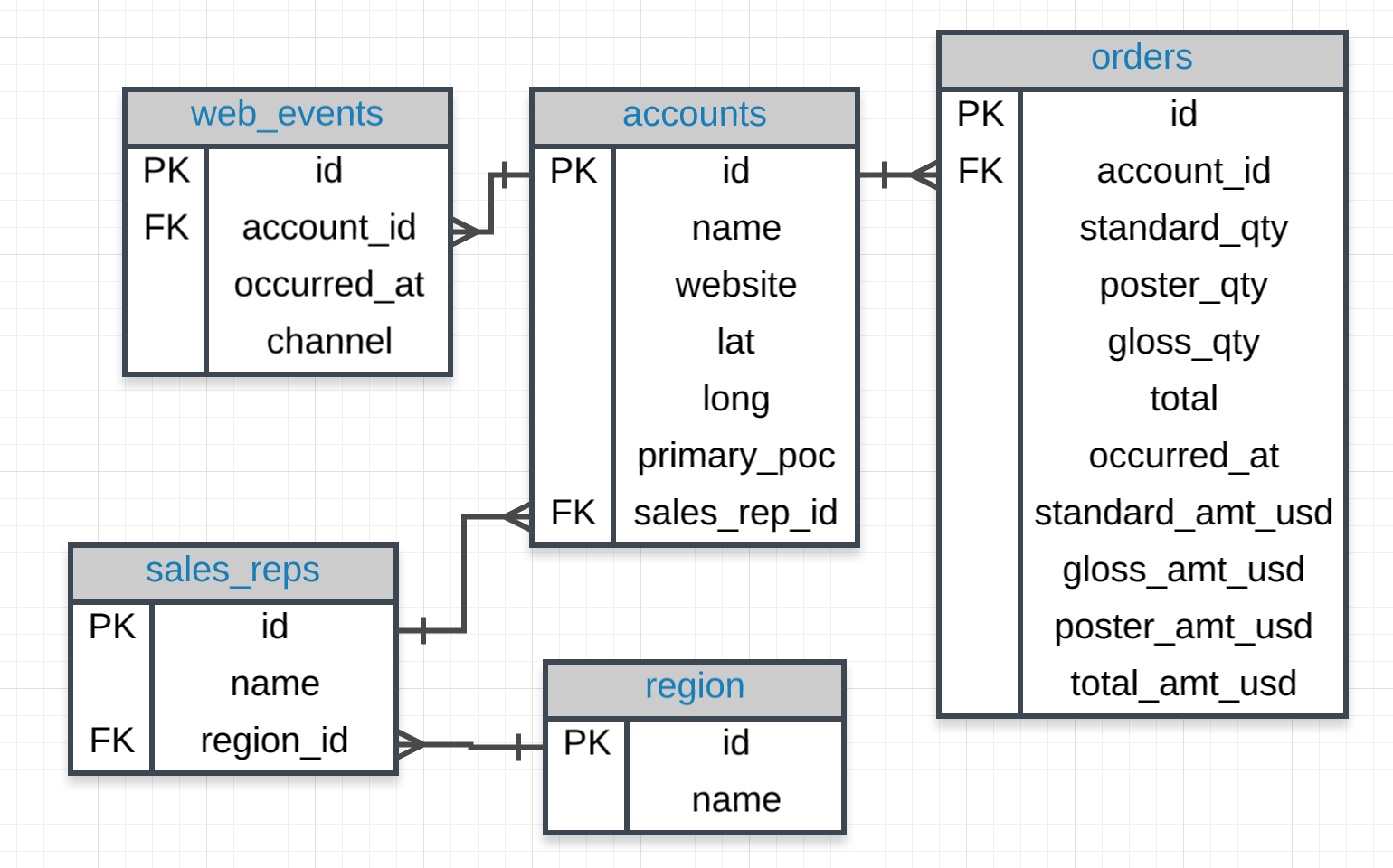
and

**SELECT** col1 + col2 total, col3

**Select** t1.column1 aliasname, t2.column2 aliasname2

**FROM** tablename **AS** t1

**JOIN** tablename2 **AS** t2



1. Provide a table for all **web\_events** associated with account **name** of Walmart. There should be three columns. Be sure to include the primary\_poc, time of the event, and the channel for each event. Additionally, you might choose to add a fourth column to assure only Walmart events were chosen.

SELECT a.primary\_poc, w.occurred\_at, w.channel, a.name

FROM web\_events w

JOIN accounts a

ON w.account\_id = a.id

WHERE a.name = 'Walmart';

1. Provide a table that provides the **region** for each **sales\_rep** along with their associated **accounts**. Your final table should include three columns: the region **name**, the sales rep **name**, and the account **name**. Sort the accounts alphabetically (A-Z) according to account name.

SELECT r.name region, s.name rep, a.name account

FROM sales\_reps s

JOIN region r

ON s.region\_id = r.id

JOIN accounts a

ON a.sales\_rep\_id = s.id

ORDER BY a.name;

1. Provide the **name** for each region for every **order**, as well as the account **name** and the **unit price**they paid (total\_amt\_usd/total) for the order. Your final table should have 3 columns: **region name**, **account name**, and **unit price**. A few accounts have 0 for **total**, so I divided by (total + 0.01) to assure not dividing by zero.

SELECT r.name region, a.name account,

o.total\_amt\_usd/(o.total + 0.01) unit\_price

FROM region r

JOIN sales\_reps s

ON s.region\_id = r.id

JOIN accounts a

ON a.sales\_rep\_id = s.id

JOIN orders o

ON o.account\_id = a.id;

# LEFT and RIGHT and Outer(L,R,Full) JOINs

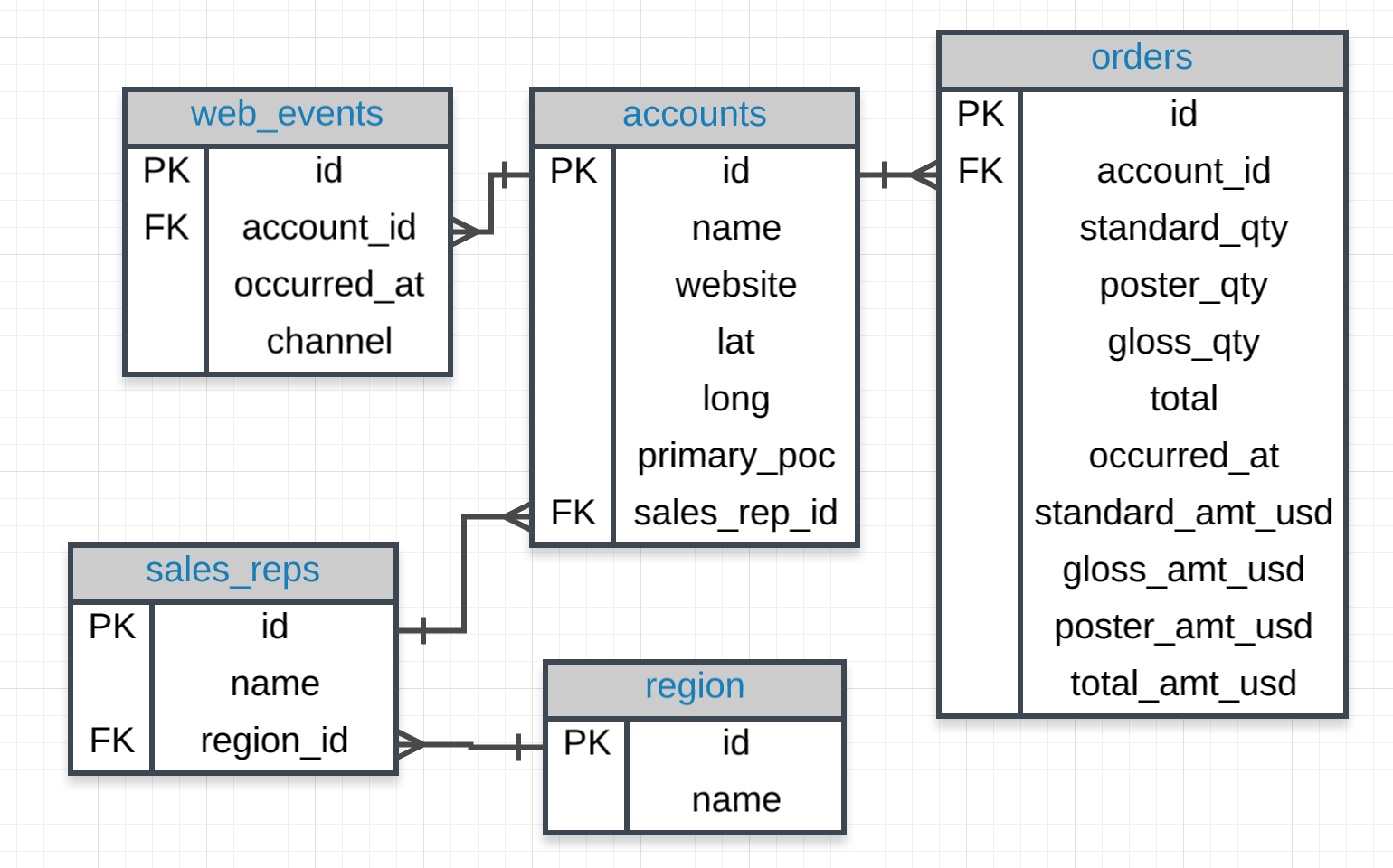
**SELECT** **c**.countryid, **c**.countryName, s.stateName

**FROM** Country **c**

**JOIN** State s

**ON** **c**.countryid = s.countryid;

# JOINs and Filtering



If we have two or more columns in your SELECT that have the same name after the table name such as accounts.name and sales\_reps.name you will need to alias them. Otherwise it will only show one of the columns. You can alias them like accounts.name AS AcountName, sales\_rep.name AS SalesRepName

1. Provide a table that provides the **region** for each **sales\_rep** along with their associated **accounts**. This time only for the Midwest region. Your final table should include three columns: the region **name**, the sales rep **name**, and the account **name**. Sort the accounts alphabetically (A-Z) according to account name.

SELECT r.name region, s.name rep, a.name account

FROM sales\_reps s

JOIN region r

ON s.region\_id = r.id

JOIN accounts a

ON a.sales\_rep\_id = s.id

WHERE r.name = 'Midwest'

ORDER BY a.name;

1. Provide a table that provides the **region** for each **sales\_rep** along with their associated **accounts**. This time only for accounts where the sales rep has a first name starting with S and in the Midwestregion. Your final table should include three columns: the region **name**, the sales rep **name**, and the account **name**. Sort the accounts alphabetically (A-Z) according to account name.

SELECT r.name region, s.name rep, a.name account

FROM sales\_reps s

JOIN region r

ON s.region\_id = r.id

JOIN accounts a

ON a.sales\_rep\_id = s.id

WHERE r.name = 'Midwest' AND s.name LIKE 'S%'

ORDER BY a.name;

1. Provide a table that provides the **region** for each **sales\_rep** along with their associated **accounts**. This time only for accounts where the sales rep has a **last** name starting with K and in the Midwestregion. Your final table should include three columns: the region **name**, the sales rep **name**, and the account **name**. Sort the accounts alphabetically (A-Z) according to account name.

SELECT r.name region, s.name rep, a.name account

FROM sales\_reps s

JOIN region r

ON s.region\_id = r.id

JOIN accounts a

ON a.sales\_rep\_id = s.id

WHERE r.name = 'Midwest' AND s.name LIKE '% K%'

ORDER BY a.name;

1. Provide the **name** for each region for every **order**, as well as the account **name** and the **unit price**they paid (total\_amt\_usd/total) for the order. However, you should only provide the results if the **standard order quantity** exceeds 100. Your final table should have 3 columns: **region name**, **account name**, and **unit price**. In order to avoid a division by zero error, adding .01 to the denominator here is helpful total\_amt\_usd/(total+0.01).

SELECT r.name region, a.name account, o.total\_amt\_usd/(o.total + 0.01) unit\_price

FROM region r

JOIN sales\_reps s

ON s.region\_id = r.id

JOIN accounts a

ON a.sales\_rep\_id = s.id

JOIN orders o

ON o.account\_id = a.id

WHERE o.standard\_qty > 100;

1. Provide the **name** for each region for every **order**, as well as the account **name** and the **unit price**they paid (total\_amt\_usd/total) for the order. However, you should only provide the results if the **standard order quantity** exceeds 100 and the **poster order quantity** exceeds 50. Your final table should have 3 columns: **region name**, **account name**, and **unit price**. Sort for the smallest **unit price** first. In order to avoid a division by zero error, adding .01 to the denominator here is helpful (total\_amt\_usd/(total+0.01).

SELECT r.name region, a.name account, o.total\_amt\_usd/(o.total + 0.01) unit\_price

FROM region r

JOIN sales\_reps s

ON s.region\_id = r.id

JOIN accounts a

ON a.sales\_rep\_id = s.id

JOIN orders o

ON o.account\_id = a.id

WHERE o.standard\_qty > 100 AND o.poster\_qty > 50

ORDER BY unit\_price;

1. Provide the **name** for each region for every **order**, as well as the account **name** and the **unit price**they paid (total\_amt\_usd/total) for the order. However, you should only provide the results if the **standard order quantity** exceeds 100 and the **poster order quantity** exceeds 50. Your final table should have 3 columns: **region name**, **account name**, and **unit price**. Sort for the largest **unit price**first. In order to avoid a division by zero error, adding .01 to the denominator here is helpful (total\_amt\_usd/(total+0.01).

SELECT r.name region, a.name account, o.total\_amt\_usd/(o.total + 0.01) unit\_price

FROM region r

JOIN sales\_reps s

ON s.region\_id = r.id

JOIN accounts a

ON a.sales\_rep\_id = s.id

JOIN orders o

ON o.account\_id = a.id

WHERE o.standard\_qty > 100 AND o.poster\_qty > 50

ORDER BY unit\_price DESC;

1. What are the different **channel**s used by **account id** 1001? Your final table should have only 2 columns: **account name** and the different **channel**s. You can try **SELECT DISTINCT** to narrow down the results to only the unique values.

SELECT DISTINCT a.name, w.channel

FROM accounts a

RIGHT JOIN web\_events w

ON a.id = w.account\_id

WHERE a.id = '1001';

1. Find all the orders that occurred in 2015. Your final table should have 4 columns: **occurred\_at**, **account name**, **order total**, and **order total\_amt\_usd**.

SELECT o.occurred\_at, a.name, o.total, o.total\_amt\_usd

FROM accounts a

JOIN orders o

ON o.account\_id = a.id

WHERE o.occurred\_at BETWEEN '01-01-2015' AND '01-01-2016'

ORDER BY o.occurred\_at DESC;

# Aggregation

**SELECT** **COUNT**(accounts.**id**) **FROM** accounts;

**SELECT** **SUM**(accounts.**id**) as total **FROM** accounts; -> treat null as 0, takes only numeric value

Select \* from orders where total is null;

Select \* from orders where total is not null;

1. Find the total amount of **poster\_qty** paper ordered in the **orders** table.

SELECT SUM(poster\_qty) AS total\_poster\_sales FROM orders;

1. Find the total amount of **standard\_qty** paper ordered in the **orders** table.

SELECT SUM(standard\_qty) AS total\_standard\_sales FROM orders;

1. Find the total dollar amount of sales using the **total\_amt\_usd** in the **orders** table.

SELECT SUM(total\_amt\_usd) AS total\_dollar\_sales FROM orders;

1. Find the total amount spent on **standard\_amt\_usd** and **gloss\_amt\_usd** paper for each order in the orders table. This should give a dollar amount for each order in the table.

SELECT standard\_amt\_usd + gloss\_amt\_usd AS total\_standard\_gloss FROM orders;

1. Find the **standard\_amt\_usd** per unit of **standard\_qty** paper. Your solution should use both an aggregation and a mathematical operator.

SELECT SUM(standard\_amt\_usd)/SUM(standard\_qty) AS standard\_price\_per\_unit

FROM orders;

# MIN & MAX $ AVG

1. When was the earliest order ever placed? You only need to return the date.

SELECT MIN(occurred\_at) FROM orders;

1. Try performing the same query as in question 1 without using an aggregation function.

SELECT occurred\_at FROM orders

ORDER BY occurred\_at

LIMIT 1;

1. When did the most recent (latest) **web\_event** occur?

SELECT MAX(occurred\_at) FROM web\_events;

1. Try to perform the result of the previous query without using an aggregation function.

SELECT occurred\_at FROM web\_events

ORDER BY occurred\_at DESC LIMIT 1;

1. Find the mean (**AVERAGE**) amount spent per order on each paper type, as well as the mean amount of each paper type purchased per order. Your final answer should have 6 values - one for each paper type for the average number of sales, as well as the average amount.

SELECT AVG(standard\_qty) mean\_standard, AVG(gloss\_qty) mean\_gloss,

AVG(poster\_qty) mean\_poster, AVG(standard\_amt\_usd) mean\_standard\_usd,

AVG(gloss\_amt\_usd) mean\_gloss\_usd, AVG(poster\_amt\_usd) mean\_poster\_usd

FROM orders;

1. Via the video, you might be interested in how to calculate the MEDIAN. Though this is more advanced than what we have covered so far try finding - what is the MEDIAN **total\_usd** spent on all **orders**?

SELECT \*

FROM (SELECT total\_amt\_usd

FROM orders

ORDER BY total\_amt\_usd

LIMIT 3457) AS Table1

ORDER BY total\_amt\_usd DESC

LIMIT 2;

# GROUP BY

1. Which **account** (by name) placed the earliest order? Your solution should have the **account name**and the **date** of the order.

SELECT a.name, o.occurred\_at

FROM accounts a

JOIN orders o

ON a.id = o.account\_id

ORDER BY occurred\_at

LIMIT 1;

1. Find the total sales in **usd** for each account. You should include two columns - the total sales for each company's orders in **usd** and the company **name**.

SELECT a.name, SUM(total\_amt\_usd) total\_sales

FROM orders o

JOIN accounts a

ON a.id = o.account\_id

GROUP BY a.name;

1. Via what **channel** did the most recent (latest) **web\_event** occur, which **account** was associated with this **web\_event**? Your query should return only three values - the **date**, **channel**, and **account name**.

SELECT w.occurred\_at, w.channel, a.name

FROM web\_events w

JOIN accounts a

ON w.account\_id = a.id

ORDER BY w.occurred\_at DESC

LIMIT 1;

1. Find the total number of times each type of **channel** from the **web\_events** was used. Your final table should have two columns - the **channel** and the number of times the channel was used.

SELECT w.channel, COUNT(\*)

FROM web\_events w

GROUP BY w.channel

1. Who was the **primary contact** associated with the earliest **web\_event**?

SELECT a.primary\_poc

FROM web\_events w

JOIN accounts a

ON a.id = w.account\_id

ORDER BY w.occurred\_at

LIMIT 1;

1. What was the smallest order placed by each **account** in terms of **total usd**. Provide only two columns - the account **name** and the **total usd**. Order from smallest dollar amounts to largest.

SELECT a.name, MIN(total\_amt\_usd) smallest\_order

FROM accounts a

JOIN orders o

ON a.id = o.account\_id

GROUP BY a.name

ORDER BY smallest\_order;

1. Find the number of **sales reps** in each region. Your final table should have two columns - the **region**and the number of **sales\_reps**. Order from fewest reps to most reps.

SELECT r.name, COUNT(\*) num\_reps

FROM region r

JOIN sales\_reps s

ON r.id = s.region\_id

GROUP BY r.name

ORDER BY num\_reps;

1. For each account, determine the average amount of each type of paper they purchased across their orders. Your result should have four columns - one for the account **name** and one for the average quantity purchased for each of the paper types for each account.

SELECT a.name, AVG(o.standard\_qty) avg\_stand, AVG(o.gloss\_qty) avg\_gloss, AVG(o.poster\_qty) avg\_post

FROM accounts a

JOIN orders o

ON a.id = o.account\_id

GROUP BY a.name;

1. For each account, determine the average amount spent per order on each paper type. Your result should have four columns - one for the account **name** and one for the average amount spent on each paper type.

SELECT a.name, AVG(o.standard\_amt\_usd) avg\_stand, AVG(o.gloss\_amt\_usd) avg\_gloss, AVG(o.poster\_amt\_usd) avg\_post

FROM accounts a

JOIN orders o

ON a.id = o.account\_id

GROUP BY a.name;

1. Determine the number of times a particular **channel** was used in the **web\_events** table for each **sales rep**. Your final table should have three columns - the **name of the sales rep**, the **channel**, and the number of occurrences. Order your table with the highest number of occurrences first.

SELECT s.name, w.channel, COUNT(\*) num\_events

FROM accounts a

JOIN web\_events w

ON a.id = w.account\_id

JOIN sales\_reps s

ON s.id = a.sales\_rep\_id

GROUP BY s.name, w.channel

ORDER BY num\_events DESC;

1. Determine the number of times a particular **channel** was used in the **web\_events** table for each **region**. Your final table should have three columns - the **region name**, the **channel**, and the number of occurrences. Order your table with the highest number of occurrences first.

SELECT r.name, w.channel, COUNT(\*) num\_events

FROM accounts a

JOIN web\_events w

ON a.id = w.account\_id

JOIN sales\_reps s

ON s.id = a.sales\_rep\_id

JOIN region r

ON r.id = s.region\_id

GROUP BY r.name, w.channel

ORDER BY num\_events DESC;

# DISTINCT

**SELECT** **DISTINCT** column1, column2, column3 **FROM** table1;

1. Use **DISTINCT** to test if there are any accounts associated with more than one region.
2. **SELECT** a.**id** **as** "account id", r.**id** **as** "region id",
3. a.**name** **as** "account name", r.**name** **as** "region name"
4. **FROM** accounts a
5. **JOIN** sales\_reps s
6. **ON** s.**id** = a.sales\_rep\_id
7. **JOIN** region r
8. **ON** r.**id** = s.region\_id;

**SELECT** **DISTINCT** **id**, **name FROM** accounts;

1. Have any **sales reps** worked on more than one account?
2. **SELECT** s.**id**, s.**name**, **COUNT**(\*) num\_accounts
3. **FROM** accounts a
4. **JOIN** sales\_reps s
5. **ON** s.**id** = a.sales\_rep\_id
6. **GROUP** **BY** s.**id**, s.**name**
7. **ORDER** **BY** num\_accounts;

**SELECT** **DISTINCT** **id**, **name FROM** sales\_reps;

# HAVING

1. How many of the **sales reps** have more than 5 accounts that they manage?

SELECT s.id, s.name, COUNT(\*) num\_accounts

FROM accounts a

JOIN sales\_reps s

ON s.id = a.sales\_rep\_id

GROUP BY s.id, s.name

HAVING COUNT(\*) > 5

ORDER BY num\_accounts;

**SubQuery**

SELECT COUNT(\*) num\_reps\_above5

FROM(SELECT s.id, s.name, COUNT(\*) num\_accounts

FROM accounts a

JOIN sales\_reps s

ON s.id = a.sales\_rep\_id

GROUP BY s.id, s.name

HAVING COUNT(\*) > 5

ORDER BY num\_accounts) AS Table1;

1. How many **accounts** have more than 20 orders?

SELECT a.id, a.name, COUNT(\*) num\_orders

FROM accounts a

JOIN orders o

ON a.id = o.account\_id

GROUP BY a.id, a.name

HAVING COUNT(\*) > 20

ORDER BY num\_orders;

1. Which account has the most orders?

SELECT a.id, a.name, COUNT(\*) num\_orders

FROM accounts a

JOIN orders o

ON a.id = o.account\_id

GROUP BY a.id, a.name

ORDER BY num\_orders DESC

LIMIT 1;

1. How many accounts spent more than 30,000 usd total across all orders?

SELECT a.id, a.name, SUM(o.total\_amt\_usd) total\_spent

FROM accounts a

JOIN orders o

ON a.id = o.account\_id

GROUP BY a.id, a.name

HAVING SUM(o.total\_amt\_usd) > 30000

ORDER BY total\_spent;

1. How many accounts spent less than 1,000 usd total across all orders?

SELECT a.id, a.name, SUM(o.total\_amt\_usd) total\_spent

FROM accounts a

JOIN orders o

ON a.id = o.account\_id

GROUP BY a.id, a.name

HAVING SUM(o.total\_amt\_usd) < 1000

ORDER BY total\_spent;

1. Which account has spent the most with us?

SELECT a.id, a.name, SUM(o.total\_amt\_usd) total\_spent

FROM accounts a

JOIN orders o

ON a.id = o.account\_id

GROUP BY a.id, a.name

ORDER BY total\_spent DESC

LIMIT 1;

1. Which account has spent the least with us?

SELECT a.id, a.name, SUM(o.total\_amt\_usd) total\_spent

FROM accounts a

JOIN orders o

ON a.id = o.account\_id

GROUP BY a.id, a.name

ORDER BY total\_spent

LIMIT 1;

1. Which accounts used facebook as a **channel** to contact customers more than 6 times?

SELECT a.id, a.name, w.channel, COUNT(\*) use\_of\_channel

FROM accounts a

JOIN web\_events w

ON a.id = w.account\_id

GROUP BY a.id, a.name, w.channel

HAVING COUNT(\*) > 6 AND w.channel = 'facebook'

ORDER BY use\_of\_channel;

1. Which account used facebook most as a **channel**?

SELECT a.id, a.name, w.channel, COUNT(\*) use\_of\_channel

FROM accounts a

JOIN web\_events w

ON a.id = w.account\_id

WHERE w.channel = 'facebook'

GROUP BY a.id, a.name, w.channel

ORDER BY use\_of\_channel DESC

LIMIT 1;

1. Which channel was most frequently used by most accounts?

SELECT a.id, a.name, w.channel, COUNT(\*) use\_of\_channel

FROM accounts a

JOIN web\_events w

ON a.id = w.account\_id

GROUP BY a.id, a.name, w.channel

ORDER BY use\_of\_channel DESC

LIMIT 10;

# DATE Functions

1. Find the sales in terms of total dollars for all orders in each year, ordered from greatest to least. Do you notice any trends in the yearly sales totals?

SELECT DATE\_PART('year', occurred\_at) ord\_year, SUM(total\_amt\_usd) total\_spent

FROM orders

GROUP BY 1

ORDER BY 2 DESC;

1. Which **month** did Parch & Posey have the greatest sales in terms of total dollars? Are all months evenly represented by the dataset?

SELECT DATE\_PART('month', occurred\_at) ord\_month, SUM(total\_amt\_usd) total\_spent

FROM orders

WHERE occurred\_at BETWEEN '2014-01-01' AND '2017-01-01'

GROUP BY 1

ORDER BY 2 DESC;

1. Which **year** did Parch & Posey have the greatest sales in terms of total number of orders? Are all years evenly represented by the dataset?

SELECT DATE\_PART('year', occurred\_at) ord\_year, COUNT(\*) total\_sales

FROM orders

GROUP BY 1

ORDER BY 2 DESC;

1. Which **month** did Parch & Posey have the greatest sales in terms of total number of orders? Are all months evenly represented by the dataset?

SELECT DATE\_PART('month', occurred\_at) ord\_month, COUNT(\*) total\_sales

FROM orders

WHERE occurred\_at BETWEEN '2014-01-01' AND '2017-01-01'

GROUP BY 1

ORDER BY 2 DESC;

1. In which **month** of which **year** did Walmart spend the most on gloss paper in terms of dollars?

SELECT DATE\_TRUNC('month', o.occurred\_at) ord\_date, SUM(o.gloss\_amt\_usd) tot\_spent

FROM orders o

JOIN accounts a

ON a.id = o.account\_id

WHERE a.name = 'Walmart'

GROUP BY 1

ORDER BY 2 DESC

LIMIT 1;

# CASE Statements

Create a column that divides the standard\_amt\_usd by the standard\_qty to find the unit price for standard paper for each order. Limit the results to the first 10 orders, and include the id and account\_id fields. **NOTE - you will be thrown an error with the correct solution to this question. This is for a division by zero. You will learn how to get a solution without an error to this query when you learn about CASE statements in a later section.**

**SELECT** **id**, account\_id, standard\_amt\_usd/standard\_qty **AS** unit\_price **FROM** orders **LIMIT** 10;

**SELECT** account\_id, **CASE** **WHEN** standard\_qty = 0 **OR** standard\_qty **IS** NULL **THEN** 0

**ELSE** standard\_amt\_usd/standard\_qty **END** **AS** unit\_price

**FROM** orders **LIMIT** 10;

1. We would like to understand 3 different levels of customers based on the amount associated with their purchases. The top branch includes anyone with a Lifetime Value (total sales of all orders) greater than 200,000 usd. The second branch is between 200,000 and 100,000 usd. The lowest branch is anyone under 100,000 usd. Provide a table that includes the **level** associated with each **account**. You should provide the **account name**, the **total sales of all orders** for the customer, and the **level**. Order with the top spending customers listed first.

SELECT a.name, SUM(total\_amt\_usd) total\_spent,

CASE WHEN SUM(total\_amt\_usd) > 200000 THEN 'top'

WHEN SUM(total\_amt\_usd) > 100000 THEN 'middle'

ELSE 'low' END AS customer\_level

FROM orders o

JOIN accounts a

ON o.account\_id = a.id

GROUP BY a.name

ORDER BY 2 DESC;

1. We would now like to perform a similar calculation to the first, but we want to obtain the total amount spent by customers only in 2016 and 2017. Keep the same **level**s as in the previous question. Order with the top spending customers listed first.

SELECT a.name, SUM(total\_amt\_usd) total\_spent,

CASE WHEN SUM(total\_amt\_usd) > 200000 THEN 'top'

WHEN SUM(total\_amt\_usd) > 100000 THEN 'middle'

ELSE 'low' END AS customer\_level

FROM orders o

JOIN accounts a

ON o.account\_id = a.id

WHERE occurred\_at > '2015-12-31'

GROUP BY 1

ORDER BY 2 DESC;

1. We would like to identify top performing **sales reps**, which are sales reps associated with more than 200 orders. Create a table with the **sales rep name**, the total number of orders, and a column with top or not depending on if they have more than 200 orders. Place the top sales people first in your final table.

SELECT s.name, COUNT(\*) num\_ords,

CASE WHEN COUNT(\*) > 200 THEN 'top'

ELSE 'not' END AS sales\_rep\_level

FROM orders o

JOIN accounts a

ON o.account\_id = a.id

JOIN sales\_reps s

ON s.id = a.sales\_rep\_id

GROUP BY s.name

ORDER BY 2 DESC;

1. The previous didn't account for the middle, nor the dollar amount associated with the sales. Management decides they want to see these characteristics represented as well. We would like to identify top performing **sales reps**, which are sales reps associated with more than 200 orders or more than 750000 in total sales. The middle group has any **rep** with more than 150 orders or 500000in sales. Create a table with the **sales rep name**, the total number of orders, total sales across all orders, and a column with top, middle, or low depending on this criteria. Place the top sales people based on dollar amount of sales first in your final table. You might see a few upset sales people by this criteria!

SELECT s.name, COUNT(\*), SUM(o.total\_amt\_usd) total\_spent,

CASE WHEN COUNT(\*) > 200 OR SUM(o.total\_amt\_usd) > 750000 THEN 'top'

WHEN COUNT(\*) > 150 OR SUM(o.total\_amt\_usd) > 500000 THEN 'middle'

ELSE 'low' END AS sales\_rep\_level

FROM orders o

JOIN accounts a

ON o.account\_id = a.id

JOIN sales\_reps s

ON s.id = a.sales\_rep\_id

GROUP BY s.name

ORDER BY 3 DESC;

**On which day-channel pair did the most events occur. (Mark all that are true)**

SELECT DATE\_TRUNC('day',occurred\_at) AS day,

channel, COUNT(\*) as events

FROM web\_events

GROUP BY 1,2

ORDER BY 3 DESC;

**Mark all of the below that are true regarding writing your subquery.**

SELECT \*

FROM (SELECT DATE\_TRUNC('day',occurred\_at) AS day,

channel, COUNT(\*) as events

FROM web\_events

GROUP BY 1,2

ORDER BY 3 DESC) sub;

**CHANNEL - AVERAGE NUMBER OF EVENTS/DAY**

SELECT channel, AVG(events) AS average\_events

FROM (SELECT DATE\_TRUNC('day',occurred\_at) AS day,

channel, COUNT(\*) as events

FROM web\_events

GROUP BY 1,2) sub

GROUP BY channel

ORDER BY 2 DESC;

# Subquery

What was the month/year combo for the first order placed?

**SELECT** DATE\_TRUNC('month', **MIN**(occurred\_at)) **FROM** orders;

The average amount of standard paper sold on the first month that any order was placed in the **orders** table (in terms of quantity).

The average amount of gloss paper sold on the first month that any order was placed in the **orders** table (in terms of quantity).

The average amount of poster paper sold on the first month that any order was placed in the **orders** table (in terms of quantity).

The total amount spent on all orders on the first month that any order was placed in the **orders**table (in terms of usd).

**SELECT** **AVG**(standard\_qty) avg\_std, **AVG**(gloss\_qty) avg\_gls, **AVG**(poster\_qty) avg\_pst

**FROM** orders

**WHERE** DATE\_TRUNC('month', occurred\_at) =

(**SELECT** DATE\_TRUNC('month', **MIN**(occurred\_at)) **FROM** orders);

**SELECT** **SUM**(total\_amt\_usd)

**FROM** orders

**WHERE** DATE\_TRUNC('month', occurred\_at) =

(**SELECT** DATE\_TRUNC('month', **MIN**(occurred\_at)) **FROM** orders);

1. Provide the **name** of the **sales\_rep** in each **region** with the largest amount of **total\_amt\_usd** sales.

First, I wanted to find the total\_amt\_usd totals associated with each sales rep, and I also wanted the region in which they were located. The query below provided this information.

SELECT s.name rep\_name, r.name region\_name, SUM(o.total\_amt\_usd) total\_amt

FROM sales\_reps s

JOIN accounts a

ON a.sales\_rep\_id = s.id

JOIN orders o

ON o.account\_id = a.id

JOIN region r

ON r.id = s.region\_id

GROUP BY 1,2

ORDER BY 3 DESC;

Next, I pulled the max for each region, and then we can use this to pull those rows in our final result.

SELECT region\_name, MAX(total\_amt) total\_amt

FROM(SELECT s.name rep\_name, r.name region\_name, SUM(o.total\_amt\_usd) total\_amt

FROM sales\_reps s

JOIN accounts a

ON a.sales\_rep\_id = s.id

JOIN orders o

ON o.account\_id = a.id

JOIN region r

ON r.id = s.region\_id

GROUP BY 1, 2) t1

GROUP BY 1;

Essentially, this is a JOIN of these two tables, where the region and amount match.

SELECT t3.rep\_name, t3.region\_name, t3.total\_amt

FROM(SELECT region\_name, MAX(total\_amt) total\_amt

FROM(SELECT s.name rep\_name, r.name region\_name, SUM(o.total\_amt\_usd) total\_amt

FROM sales\_reps s

JOIN accounts a

ON a.sales\_rep\_id = s.id

JOIN orders o

ON o.account\_id = a.id

JOIN region r

ON r.id = s.region\_id

GROUP BY 1, 2) t1

GROUP BY 1) t2

JOIN (SELECT s.name rep\_name, r.name region\_name, SUM(o.total\_amt\_usd) total\_amt

FROM sales\_reps s

JOIN accounts a

ON a.sales\_rep\_id = s.id

JOIN orders o

ON o.account\_id = a.id

JOIN region r

ON r.id = s.region\_id

GROUP BY 1,2

ORDER BY 3 DESC) t3

ON t3.region\_name = t2.region\_name AND t3.total\_amt = t2.total\_amt;

1. For the region with the largest (sum) of sales **total\_amt\_usd**, how many **total** (count) orders were placed?

The first query I wrote was to pull the total\_amt\_usd for each region.

SELECT r.name region\_name, SUM(o.total\_amt\_usd) total\_amt

FROM sales\_reps s

JOIN accounts a

ON a.sales\_rep\_id = s.id

JOIN orders o

ON o.account\_id = a.id

JOIN region r

ON r.id = s.region\_id

GROUP BY r.name;

Then we just want the region with the max amount from this table. There are two ways I considered getting this amount. One was to pull the max using a subquery. Another way is to order descending and just pull the top value.

SELECT MAX(total\_amt)

FROM (SELECT r.name region\_name, SUM(o.total\_amt\_usd) total\_amt

FROM sales\_reps s

JOIN accounts a

ON a.sales\_rep\_id = s.id

JOIN orders o

ON o.account\_id = a.id

JOIN region r

ON r.id = s.region\_id

GROUP BY r.name) sub;

Finally, we want to pull the total orders for the region with this amount:

SELECT r.name, COUNT(o.total) total\_orders

FROM sales\_reps s

JOIN accounts a

ON a.sales\_rep\_id = s.id

JOIN orders o

ON o.account\_id = a.id

JOIN region r

ON r.id = s.region\_id

GROUP BY r.name

HAVING SUM(o.total\_amt\_usd) = (

SELECT MAX(total\_amt)

FROM (SELECT r.name region\_name, SUM(o.total\_amt\_usd) total\_amt

FROM sales\_reps s

JOIN accounts a

ON a.sales\_rep\_id = s.id

JOIN orders o

ON o.account\_id = a.id

JOIN region r

ON r.id = s.region\_id

GROUP BY r.name) sub);

This provides the Northeast with 2357 orders.

1. For the **name** of the account that purchased the most (in total over their lifetime as a customer) **standard\_qty** paper, **how many accounts** still had more in **total** purchases?

First, we want to find the account that had the most standard\_qty paper. The query here pulls that account, as well as the total amount:

SELECT a.name account\_name, SUM(o.standard\_qty) total\_std, SUM(o.total) total

FROM accounts a

JOIN orders o

ON o.account\_id = a.id

GROUP BY 1

ORDER BY 2 DESC

LIMIT 1;

Now, I want to use this to pull all the accounts with more total sales:

SELECT a.name

FROM orders o

JOIN accounts a

ON a.id = o.account\_id

GROUP BY 1

HAVING SUM(o.total) > (SELECT total

FROM (SELECT a.name act\_name, SUM(o.standard\_qty) tot\_std, SUM(o.total) total

FROM accounts a

JOIN orders o

ON o.account\_id = a.id

GROUP BY 1

ORDER BY 2 DESC

LIMIT 1) sub);

This is now a list of all the accounts with more total orders. We can get the count with just another simple subquery.

SELECT COUNT(\*)

FROM (SELECT a.name

FROM orders o

JOIN accounts a

ON a.id = o.account\_id

GROUP BY 1

HAVING SUM(o.total) > (SELECT total

FROM (SELECT a.name act\_name, SUM(o.standard\_qty) tot\_std, SUM(o.total) total

FROM accounts a

JOIN orders o

ON o.account\_id = a.id

GROUP BY 1

ORDER BY 2 DESC

LIMIT 1) inner\_tab)

) counter\_tab;

1. For the customer that spent the most (in total over their lifetime as a customer) **total\_amt\_usd**, how many **web\_events** did they have for each channel?

Here, we first want to pull the customer with the most spent in lifetime value.

SELECT a.id, a.name, SUM(o.total\_amt\_usd) tot\_spent

FROM orders o

JOIN accounts a

ON a.id = o.account\_id

GROUP BY a.id, a.name

ORDER BY 3 DESC

LIMIT 1;

Now, we want to look at the number of events on each channel this company had, which we can match with just the id.

SELECT a.name, w.channel, COUNT(\*)

FROM accounts a

JOIN web\_events w

ON a.id = w.account\_id AND a.id = (SELECT id

FROM (SELECT a.id, a.name, SUM(o.total\_amt\_usd) tot\_spent

FROM orders o

JOIN accounts a

ON a.id = o.account\_id

GROUP BY a.id, a.name

ORDER BY 3 DESC

LIMIT 1) inner\_table)

GROUP BY 1, 2

ORDER BY 3 DESC;

I added an ORDER BY for no real reason, and the account name to assure I was only pulling from one account.

1. What is the lifetime average amount spent in terms of **total\_amt\_usd** for the top 10 total spending **accounts**?

First, we just want to find the top 10 accounts in terms of highest total\_amt\_usd.

SELECT a.id, a.name, SUM(o.total\_amt\_usd) tot\_spent

FROM orders o

JOIN accounts a

ON a.id = o.account\_id

GROUP BY a.id, a.name

ORDER BY 3 DESC

LIMIT 10;

Now, we just want the average of these 10 amounts.

SELECT AVG(tot\_spent)

FROM (SELECT a.id, a.name, SUM(o.total\_amt\_usd) tot\_spent

FROM orders o

JOIN accounts a

ON a.id = o.account\_id

GROUP BY a.id, a.name

ORDER BY 3 DESC

LIMIT 10) temp;

1. What is the lifetime average amount spent in terms of **total\_amt\_usd** for only the companies that spent more than the average of all orders.

First, we want to pull the average of all accounts in terms of total\_amt\_usd:

SELECT AVG(o.total\_amt\_usd) avg\_all

FROM orders o

JOIN accounts a

ON a.id = o.account\_id;

Then, we want to only pull the accounts with more than this average amount.

SELECT o.account\_id, AVG(o.total\_amt\_usd)

FROM orders o

GROUP BY 1

HAVING AVG(o.total\_amt\_usd) > (SELECT AVG(o.total\_amt\_usd) avg\_all

FROM orders o

JOIN accounts a

ON a.id = o.account\_id);

Finally, we just want the average of these values.

SELECT AVG(avg\_amt)

FROM (SELECT o.account\_id, AVG(o.total\_amt\_usd) avg\_amt

FROM orders o

GROUP BY 1

HAVING AVG(o.total\_amt\_usd) > (SELECT AVG(o.total\_amt\_usd) avg\_all

FROM orders o

JOIN accounts a

ON a.id = o.account\_id)) temp\_table;

# WITH

The **WITH** statement is often called a **Common Table Expression** or **CTE**.

QUESTION: You need to find the average number of events for each channel per day.

SOLUTION:

SELECT channel, AVG(events) AS average\_events

FROM (SELECT DATE\_TRUNC('day',occurred\_at) AS day,

channel, COUNT(\*) as events

FROM web\_events

GROUP BY 1,2) sub

GROUP BY channel

ORDER BY 2 DESC;

Let's try this again using a WITH statement.

SELECT DATE\_TRUNC('day',occurred\_at) AS day,

channel, COUNT(\*) as events

FROM web\_events

GROUP BY 1,2

This is the part we put in the WITH statement. Notice, we are aliasing the table as events below:

WITH events AS (

SELECT DATE\_TRUNC('day',occurred\_at) AS day,

channel, COUNT(\*) as events

FROM web\_events

GROUP BY 1,2)

Now, we can use this newly created events table as if it is any other table in our database:

WITH events AS (

SELECT DATE\_TRUNC('day',occurred\_at) AS day,

channel, COUNT(\*) as events

FROM web\_events

GROUP BY 1,2)

SELECT channel, AVG(events) AS average\_events

FROM events

GROUP BY channel

ORDER BY 2 DESC;

For the above example, we don't need anymore than the one additional table, but imagine we needed to create a second table to pull from. We can create an additional table to pull from in the following way:

WITH table1 AS (

SELECT \*

FROM web\_events),

table2 AS (

SELECT \*

FROM accounts)

SELECT \*

FROM table1

JOIN table2

ON table1.account\_id = table2.id;

1. Provide the **name** of the **sales\_rep** in each **region** with the largest amount of **total\_amt\_usd** sales.

WITH t1 AS (

SELECT s.name rep\_name, r.name region\_name, SUM(o.total\_amt\_usd) total\_amt

FROM sales\_reps s

JOIN accounts a

ON a.sales\_rep\_id = s.id

JOIN orders o

ON o.account\_id = a.id

JOIN region r

ON r.id = s.region\_id

GROUP BY 1,2

ORDER BY 3 DESC),

t2 AS (

SELECT region\_name, MAX(total\_amt) total\_amt

FROM t1

GROUP BY 1)

SELECT t1.rep\_name, t1.region\_name, t1.total\_amt

FROM t1

JOIN t2

ON t1.region\_name = t2.region\_name AND t1.total\_amt = t2.total\_amt;

1. For the region with the largest sales **total\_amt\_usd**, how many **total** orders were placed?

WITH t1 AS (

SELECT r.name region\_name, SUM(o.total\_amt\_usd) total\_amt

FROM sales\_reps s

JOIN accounts a

ON a.sales\_rep\_id = s.id

JOIN orders o

ON o.account\_id = a.id

JOIN region r

ON r.id = s.region\_id

GROUP BY r.name),

t2 AS (

SELECT MAX(total\_amt)

FROM t1)

SELECT r.name, COUNT(o.total) total\_orders

FROM sales\_reps s

JOIN accounts a

ON a.sales\_rep\_id = s.id

JOIN orders o

ON o.account\_id = a.id

JOIN region r

ON r.id = s.region\_id

GROUP BY r.name

HAVING SUM(o.total\_amt\_usd) = (SELECT \* FROM t2);

1. For the **name** of the account that purchased the most (in total over their lifetime as a customer) **standard\_qty** paper, **how many accounts** still had more in **total** purchases?

WITH t1 AS (

SELECT a.name account\_name, SUM(o.standard\_qty) total\_std, SUM(o.total) total

FROM accounts a

JOIN orders o

ON o.account\_id = a.id

GROUP BY 1

ORDER BY 2 DESC

LIMIT 1),

t2 AS (

SELECT a.name

FROM orders o

JOIN accounts a

ON a.id = o.account\_id

GROUP BY 1

HAVING SUM(o.total) > (SELECT total FROM t1))

SELECT COUNT(\*)

FROM t2;

1. For the customer that spent the most (in total over their lifetime as a customer) **total\_amt\_usd**, how many **web\_events** did they have for each channel?

WITH t1 AS (

SELECT a.id, a.name, SUM(o.total\_amt\_usd) tot\_spent

FROM orders o

JOIN accounts a

ON a.id = o.account\_id

GROUP BY a.id, a.name

ORDER BY 3 DESC

LIMIT 1)

SELECT a.name, w.channel, COUNT(\*)

FROM accounts a

JOIN web\_events w

ON a.id = w.account\_id AND a.id = (SELECT id FROM t1)

GROUP BY 1, 2

ORDER BY 3 DESC;

1. What is the lifetime average amount spent in terms of **total\_amt\_usd** for the top 10 total spending **accounts**?

WITH t1 AS (

SELECT a.id, a.name, SUM(o.total\_amt\_usd) tot\_spent

FROM orders o

JOIN accounts a

ON a.id = o.account\_id

GROUP BY a.id, a.name

ORDER BY 3 DESC

LIMIT 10)

SELECT AVG(tot\_spent)

FROM t1;

1. What is the lifetime average amount spent in terms of **total\_amt\_usd** for only the companies that spent more than the average of all **accounts**.

WITH t1 AS (

SELECT AVG(o.total\_amt\_usd) avg\_all

FROM orders o

JOIN accounts a

ON a.id = o.account\_id),

t2 AS (

SELECT o.account\_id, AVG(o.total\_amt\_usd) avg\_amt

FROM orders o

GROUP BY 1

HAVING AVG(o.total\_amt\_usd) > (SELECT \* FROM t1))

SELECT AVG(avg\_amt)

FROM t2;

# LEFT & RIGHT & LENGTH

1. In the **accounts** table, there is a column holding the **website** for each company. The last three digits specify what type of web address they are using. A list of extensions (and pricing) is provided [**here**](https://iwantmyname.com/domains/domain-name-registration-list-of-extensions). Pull these extensions and provide how many of each website type exist in the **accounts** table.

SELECT RIGHT(website, 3) AS domain, COUNT(\*) num\_companies

FROM accounts

GROUP BY 1

ORDER BY 2 DESC;

1. There is much debate about how much the name [**(or even the first letter of a company name)**](https://www.quora.com/Does-a-companys-name-matter)matters. Use the **accounts** table to pull the first letter of each company name to see the distribution of company names that begin with each letter (or number).

SELECT LEFT(UPPER(name), 1) AS first\_letter, COUNT(\*) num\_companies

FROM accounts

GROUP BY 1

ORDER BY 2 DESC;

1. Use the **accounts** table and a **CASE** statement to create two groups: one group of company names that start with a number and a second group of those company names that start with a letter. What proportion of company names start with a letter?

SELECT SUM(num) nums, SUM(letter) letters

FROM (SELECT name, CASE WHEN LEFT(UPPER(name), 1) IN ('0','1','2','3','4','5','6','7','8','9')

THEN 1 ELSE 0 END AS num,

CASE WHEN LEFT(UPPER(name), 1) IN ('0','1','2','3','4','5','6','7','8','9')

THEN 0 ELSE 1 END AS letter

FROM accounts) t1;

1. Consider vowels as a, e, i, o, and u. What proportion of company names start with a vowel, and what percent start with anything else?

SELECT SUM(vowels) vowels, SUM(other) other

FROM (SELECT name, CASE WHEN LEFT(UPPER(name), 1) IN ('A','E','I','O','U')

THEN 1 ELSE 0 END AS vowels,

CASE WHEN LEFT(UPPER(name), 1) IN ('A','E','I','O','U')

THEN 0 ELSE 1 END AS other

FROM accounts) t1;

# POSITION, STRPOS, & SUBSTR & LOWER & UPPER

1. Use the accounts table to create **first** and **last** name columns that hold the first and last names for the primary\_poc.

SELECT LEFT(primary\_poc, STRPOS(primary\_poc, ' ') -1 ) first\_name,

RIGHT(primary\_poc, LENGTH(primary\_poc) - STRPOS(primary\_poc, ' ')) last\_name

FROM accounts;

1. Now see if you can do the same thing for every rep name in the sales\_reps table. Again provide **first**and **last** name columns.

SELECT LEFT(name, STRPOS(name, ' ') -1 ) first\_name,

RIGHT(name, LENGTH(name) - STRPOS(name, ' ')) last\_name

FROM sales\_reps;

# CONCAT

**CONCAT(first\_name, ' ', last\_name)** or with piping as **first\_name || ' ' || last\_name**.

1. Each company in the accounts table wants to create an email address for each primary\_poc. The email address should be the first name of the **primary\_poc** . last name **primary\_poc** @ company name .com.

WITH t1 AS (

SELECT LEFT(primary\_poc, STRPOS(primary\_poc, ' ') -1 ) first\_name, RIGHT(primary\_poc, LENGTH(primary\_poc) - STRPOS(primary\_poc, ' ')) last\_name, name

FROM accounts)

SELECT first\_name, last\_name, CONCAT(first\_name, '.', last\_name, '@', name, '.com')

FROM t1;

1. You may have noticed that in the previous solution some of the company names include spaces, which will certainly not work in an email address. See if you can create an email address that will work by removing all of the spaces in the account name, but otherwise your solution should be just as in question 1. Some helpful documentation is [**here**](https://www.postgresql.org/docs/8.1/static/functions-string.html).

WITH t1 AS (

SELECT LEFT(primary\_poc, STRPOS(primary\_poc, ' ') -1 ) first\_name, RIGHT(primary\_poc, LENGTH(primary\_poc) - STRPOS(primary\_poc, ' ')) last\_name, name

FROM accounts)

SELECT first\_name, last\_name, CONCAT(first\_name, '.', last\_name, '@', REPLACE(name, ' ', ''), '.com')

FROM t1;

1. We would also like to create an initial password, which they will change after their first log in. The first password will be the first letter of the primary\_poc's first name (lowercase), then the last letter of their first name (lowercase), the first letter of their last name (lowercase), the last letter of their last name (lowercase), the number of letters in their first name, the number of letters in their last name, and then the name of the company they are working with, all capitalized with no spaces.

WITH t1 AS (

SELECT LEFT(primary\_poc, STRPOS(primary\_poc, ' ') -1 ) first\_name, RIGHT(primary\_poc, LENGTH(primary\_poc) - STRPOS(primary\_poc, ' ')) last\_name, name

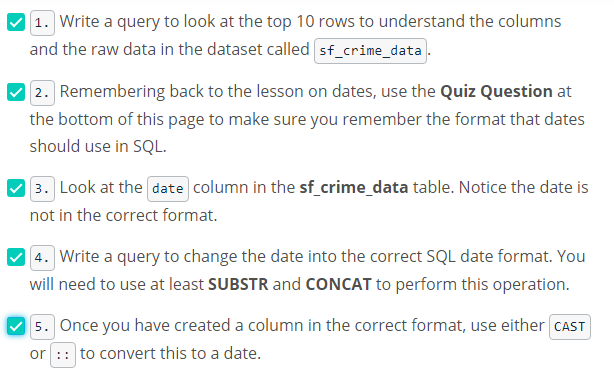
FROM accounts)

SELECT first\_name, last\_name, CONCAT(first\_name, '.', last\_name, '@', name, '.com'), LEFT(LOWER(first\_name), 1) || RIGHT(LOWER(first\_name), 1) || LEFT(LOWER(last\_name), 1) || RIGHT(LOWER(last\_name), 1) || LENGTH(first\_name) || LENGTH(last\_name) || REPLACE(UPPER(name), ' ', '')

FROM t1;

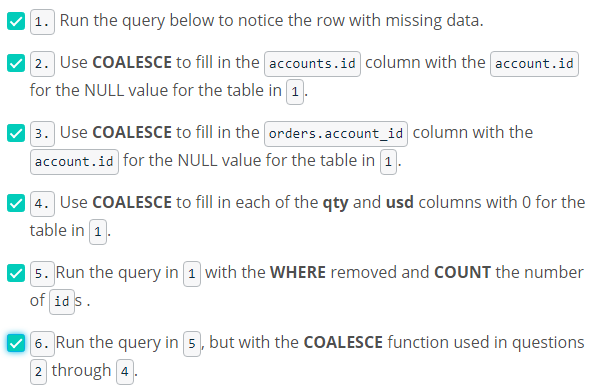
# CAST

1. **TO\_DATE**
2. **CAST**
3. Casting with ::



1. **SELECT** \* **FROM** sf\_crime\_data **LIMIT** 10;
2. **yyyy-mm-dd**
3. **mm/dd/yyyy**
4. SELECT date orig\_date, (SUBSTR(date, 7, 4) || '-' || LEFT(date, 2) || '-' || SUBSTR(date, 4, 2)) new\_date FROM sf\_crime\_data;
5. SELECT date orig\_date, (SUBSTR(date, 7, 4) || '-' || LEFT(date, 2) || '-' || SUBSTR(date, 4, 2))::DATE new\_date FROM sf\_crime\_data;

# COALESCE



SELECT \*

FROM accounts a

LEFT JOIN orders o

ON a.id = o.account\_id

WHERE o.total IS NULL;

SELECT COALESCE(a.id, a.id) filled\_id, a.name, a.website, a.lat, a.long, a.primary\_poc, a.sales\_rep\_id, o.\*

FROM accounts a

LEFT JOIN orders o

ON a.id = o.account\_id

WHERE o.total IS NULL;

SELECT COALESCE(a.id, a.id) filled\_id, a.name, a.website, a.lat, a.long, a.primary\_poc, a.sales\_rep\_id, COALESCE(o.account\_id, a.id) account\_id, o.occurred\_at, o.standard\_qty, o.gloss\_qty, o.poster\_qty, o.total, o.standard\_amt\_usd, o.gloss\_amt\_usd, o.poster\_amt\_usd, o.total\_amt\_usd

FROM accounts a

LEFT JOIN orders o

ON a.id = o.account\_id

WHERE o.total IS NULL;

SELECT COALESCE(a.id, a.id) filled\_id, a.name, a.website, a.lat, a.long, a.primary\_poc, a.sales\_rep\_id, COALESCE(o.account\_id, a.id) account\_id, o.occurred\_at, COALESCE(o.standard\_qty, 0) standard\_qty, COALESCE(o.gloss\_qty,0) gloss\_qty, COALESCE(o.poster\_qty,0) poster\_qty, COALESCE(o.total,0) total, COALESCE(o.standard\_amt\_usd,0) standard\_amt\_usd, COALESCE(o.gloss\_amt\_usd,0) gloss\_amt\_usd, COALESCE(o.poster\_amt\_usd,0) poster\_amt\_usd, COALESCE(o.total\_amt\_usd,0) total\_amt\_usd

FROM accounts a

LEFT JOIN orders o

ON a.id = o.account\_id

WHERE o.total IS NULL;

SELECT COUNT(\*)

FROM accounts a

LEFT JOIN orders o

ON a.id = o.account\_id;

SELECT COALESCE(a.id, a.id) filled\_id, a.name, a.website, a.lat, a.long, a.primary\_poc, a.sales\_rep\_id, COALESCE(o.account\_id, a.id) account\_id, o.occurred\_at, COALESCE(o.standard\_qty, 0) standard\_qty, COALESCE(o.gloss\_qty,0) gloss\_qty, COALESCE(o.poster\_qty,0) poster\_qty, COALESCE(o.total,0) total, COALESCE(o.standard\_amt\_usd,0) standard\_amt\_usd, COALESCE(o.gloss\_amt\_usd,0) gloss\_amt\_usd, COALESCE(o.poster\_amt\_usd,0) poster\_amt\_usd, COALESCE(o.total\_amt\_usd,0) total\_amt\_usd

FROM accounts a

LEFT JOIN orders o

ON a.id = o.account\_id;

# Window Functions

Using Derek's previous video as an example, create another running total. This time, create a running total of standard\_amt\_usd (in the orders table) over order time with no date truncation. Your final table should have two columns: one with the amount being added for each new row, and a second with the running total.

**SELECT** standard\_amt\_usd,

**SUM**(standard\_amt\_usd) **OVER** (**ORDER** **BY** occurred\_at) **AS** running\_total

**FROM** orders

Now, modify your query from the previous quiz to include partitions. Still create a running total of standard\_amt\_usd (in the orders table) over order time, but this time, date truncate occurred\_at by year and partition by that same year-truncated occurred\_at variable. Your final table should have three columns: One with the amount being added for each row, one for the truncated date, and a final columns with the running total within each year.

SELECT standard\_amt\_usd,

DATE\_TRUNC('year', occurred\_at) as year,

SUM(standard\_amt\_usd) OVER (PARTITION BY DATE\_TRUNC('year', occurred\_at) ORDER BY occurred\_at) AS running\_total

FROM orders

# ROW\_NUMBER & RANK

Select the id, account\_id, and total variable from the orders table, then create a column called total\_rank that ranks this total amount of paper ordered (from highest to lowest) for each account using a partition. Your final table should have these four columns.

**SELECT** **id**,

account\_id,

total,

**RANK**() **OVER** (**PARTITION** **BY** account\_id **ORDER** **BY** total **DESC**) **AS** total\_rank

**FROM** orders

# Aggregates in Window Functions

**SELECT** **id**,

account\_id,

DATE\_TRUNC('year',occurred\_at) **AS** **year**,

**DENSE\_RANK**() **OVER** account\_year\_window **AS** **dense\_rank**,

total\_amt\_usd,

**SUM**(total\_amt\_usd) **OVER** account\_year\_window **AS** sum\_total\_amt\_usd,

**COUNT**(total\_amt\_usd) **OVER** account\_year\_window **AS** count\_total\_amt\_usd,

**AVG**(total\_amt\_usd) **OVER** account\_year\_window **AS** avg\_total\_amt\_usd,

**MIN**(total\_amt\_usd) **OVER** account\_year\_window **AS** min\_total\_amt\_usd,

**MAX**(total\_amt\_usd) **OVER** account\_year\_window **AS** max\_total\_amt\_usd

**FROM** orders

WINDOW account\_year\_window **AS** (**PARTITION** **BY** account\_id **ORDER** **BY** DATE\_TRUNC('year',occurred\_at))

# Comparing a Row to Previous Row

**SELECT** account\_id,

standard\_sum,

LAG(standard\_sum) **OVER** (**ORDER** **BY** standard\_sum) **AS** lag,

**LEAD**(standard\_sum) **OVER** (**ORDER** **BY** standard\_sum) **AS** **lead**,

standard\_sum - LAG(standard\_sum) **OVER** (**ORDER** **BY** standard\_sum) **AS** lag\_difference,

**LEAD**(standard\_sum) **OVER** (**ORDER** **BY** standard\_sum) - standard\_sum **AS** lead\_difference

**FROM** (

**SELECT** account\_id,

**SUM**(standard\_qty) **AS** standard\_sum

**FROM** orders

**GROUP** **BY** 1

) sub

Modify Derek's query from the previous video in the SQL Explorer below to perform this analysis. You'll need to use occurred\_at and total\_amt\_usd in the orders table along with LEAD to do so. In your query results, there should be four columns: occurred\_at, total\_amt\_usd, lead, and lead\_difference.

**SELECT** occurred\_at,

total\_amt\_usd,

**LEAD**(total\_amt\_usd) **OVER** (**ORDER** **BY** occurred\_at) **AS** **lead**,

**LEAD**(total\_amt\_usd) **OVER** (**ORDER** **BY** occurred\_at) - total\_amt\_usd **AS** lead\_difference

**FROM** (

**SELECT** occurred\_at,

**SUM**(total\_amt\_usd) **AS** total\_amt\_usd

**FROM** orders

**GROUP** **BY** 1

) sub

# Percentiles

1. Use the NTILE functionality to divide the accounts into 4 levels in terms of the amount of standard\_qty for their orders. Your resulting table should have the account\_id, the occurred\_at time for each order, the total amount of standard\_qty paper purchased, and one of four levels in a standard\_quartile column.

SELECT id,

account\_id,

occurred\_at,

standard\_qty,

NTILE(4) OVER (PARTITION BY account\_id ORDER BY standard\_qty) AS standard\_quartile

FROM orders

ORDER BY account\_id DESC

1. Use the NTILE functionality to divide the accounts into two levels in terms of the amount of gloss\_qty for their orders. Your resulting table should have the account\_id, the occurred\_at time for each order, the total amount of gloss\_qty paper purchased, and one of two levels in a gloss\_half column.

SELECT id,

account\_id,

occurred\_at,

gloss\_qty,

NTILE(2) OVER (PARTITION BY account\_id ORDER BY gloss\_qty) AS gloss\_half

FROM orders

ORDER BY account\_id DESC

1. Use the NTILE functionality to divide the accounts into 100 levels in terms of the amount of total\_amt\_usd for their orders. Your resulting table should have the account\_id, the occurred\_attime for each order, the total amount of total\_amt\_usd paper purchased, and one of 100 levels in a total\_percentile column.

SELECT id,

account\_id,

occurred\_at,

total\_amt\_usd,

NTILE(100) OVER (PARTITION BY account\_id ORDER BY total\_amt\_usd) AS total\_percentile

FROM orders

ORDER BY account\_id DESC

# Advanced SQL

**SELECT** column\_name(s)

**FROM** Table\_A

**INNER** **JOIN** Table\_B **ON** Table\_A.column\_name = Table\_B.column\_name;

**SELECT** column\_name(s)

**FROM** Table\_A

**LEFT** **JOIN** Table\_B **ON** Table\_A.column\_name = Table\_B.column\_name;

**SELECT** column\_name(s)

**FROM** Table\_A

**RIGHT** **JOIN** Table\_B **ON** Table\_A.column\_name = Table\_B.column\_name;

**SELECT** column\_name(s)

**FROM** Table\_A

**FULL** **OUTER** **JOIN** Table\_B **ON** Table\_A.column\_name = Table\_B.column\_name;

If you wanted to return unmatched rows only, which is useful for some cases of data assessment, you can isolate them by adding the following line to the end of the query:

WHERE Table\_A.column\_name IS **NULL** **OR** Table\_B.column\_name IS **NULL**

1. Are there any unmatch row? - No
2. **SELECT** \* **FROM** accounts **FULL** **JOIN** sales\_reps **ON** accounts.sales\_rep\_id = sales\_reps.**id**

If unmatched rows existed (they don't for this query), you could isolate them by adding the following line to the end of the query:

WHERE accounts.sales\_rep\_id IS **NULL** **OR** sales\_reps.id IS **NULL**

# JOINs with Comparison Operators

**SELECT** accounts.**name** **as** account\_name,

accounts.primary\_poc **as** poc\_name,

sales\_reps.**name** **as** sales\_rep\_name

**FROM** accounts

**LEFT** **JOIN** sales\_reps

**ON** accounts.sales\_rep\_id = sales\_reps.**id**

**AND** accounts.primary\_poc < sales\_reps.**name**

What is the relationship between accounts.primary\_poc and sales\_reps.name?

Ans- The primary point of contact's full name comes before the sales representative's name alphabetically

# Self JOINs

**SELECT** o1.**id** **AS** o1\_id,

o1.account\_id **AS** o1\_account\_id,

o1.occurred\_at **AS** o1\_occurred\_at,

o2.**id** **AS** o2\_id,

o2.account\_id **AS** o2\_account\_id,

o2.occurred\_at **AS** o2\_occurred\_at

**FROM** orders o1

**LEFT** **JOIN** orders o2

**ON** o1.account\_id = o2.account\_id

**AND** o2.occurred\_at > o1.occurred\_at

**AND** o2.occurred\_at <= o1.occurred\_at + INTERVAL '28 days'

**ORDER** **BY** o1.account\_id, o1.occurred\_at

* change the interval to 1 day to find web events that occur within one after another within one day
* add a column for the channel variable in both instances of the table in your query

# UNION

Tables has to have same no.of rows and same data types in same order

Adds only distinct value

Write a query that uses UNION ALL on two instances (and selecting all columns) of the accounts table. Then inspect the results and answer the subsequent quiz.

**SELECT** \*

**FROM** accounts

**UNION** ALL

**SELECT** \*

**FROM** accounts

Add a WHERE clause to each of the tables that you unioned in the query above, filtering the first table where name equals Walmart and filtering the second table where name equals Disney. Inspect the results then answer the subsequent quiz.

**SELECT** \*

**FROM** accounts

**WHERE** **name** = 'Walmart'

**UNION** ALL

**SELECT** \*

**FROM** accounts

**WHERE** **name** = 'Disney'

Count 2 tables after union all - 351

WITH double\_accounts AS (

**SELECT** \*

**FROM** accounts

**UNION** ALL

**SELECT** \*

**FROM** accounts

)

**SELECT** **name**,

**COUNT**(\*) **AS** name\_count

**FROM** double\_accounts

**GROUP** **BY** 1

**ORDER** **BY** 2 **DESC**

# Performance Tuning

Count distinct takes long time then count(\*)

Limit subquery not in result

Explain

Select ---

**Link:**

<https://www.w3schools.com/sql/sql_union.asp>

<https://www.w3resource.com/sql/joins/cross-join.php>

<https://www.w3schools.com/sql/sql_join_self.asp>

subquary - <https://community.modeanalytics.com/sql/tutorial/sql-subqueries/>

date - <https://en.wikipedia.org/wiki/Date_format_by_country>

<https://blog.modeanalytics.com/date-trunc-sql-timestamp-function-count-on/>

<https://www.postgresql.org/docs/9.1/static/functions-datetime.html>

Functions - <http://www.postgresql.org/docs/9.1/static/functions-string.html>

<https://www.postgresql.org/docs/9.1/static/tutorial-window.html>

<https://www.postgresql.org/docs/8.4/static/functions-window.html>

interval - <https://www.postgresql.org/docs/8.2/static/functions-datetime.html>