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;

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(o.total )

from accounts as a

Join orders as o

on a.id = o.account\_id

Group by (a.name)

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;

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 channel , count(\*)

Form web\_events

Group by channel

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

**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 **ASC**

**LIMIT** 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 min(o.total\_usd), a.name

From accounts as a

Join a.id = o.account\_id

Group by (name)

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

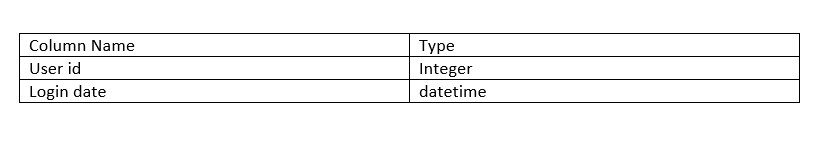
**JOIN** sales\_reps s

**ON** r.**id** = s.region\_id

**GROUP** **BY** r.**name**

**ORDER** **BY** num\_reps;

## Q4: Assume you have the given table below which contains information on user logins. Write a query to obtain the number of reactivated users (Users who did not log in the previous month and then logged in the current month)



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  | **SELECT** | |  |  | **DATE**\_TRUNC('month', current\_month.login\_date) AS current\_month, | |  |  | **COUNT**(\*) AS num\_reactivated\_users | |  |  | **FROM** | |  |  | user\_logins current\_month | |  |  | **WHERE** | |  |  | **NOT EXISTS** ( | |  |  | **SELECT** | |  |  | \* | |  |  | **FROM** | |  |  | user\_logins last\_month | |  |  | **WHERE** | |  |  | **DATE**\_**TRUNC**('month', last\_month.login\_date) BETWEEN DATE\_TRUNC('month', current\_month.login\_date) AND DATE\_TRUNC('month', current\_month.login\_date) - INTERVAL '1 month' | |  |  | ) | |

* The order of column names in your **GROUP BY** clause doesn’t matter—the results will be the same regardless. If we run the same query and reverse the order in the **GROUP BY** clause, you can see we get the same results.
* As with **ORDER BY**, you can substitute numbers for column names in the **GROUP BY** clause. It’s generally recommended to do this only when you’re grouping many columns, or if something else is causing the text in the GROUP BY clause to be excessively long.
* A reminder here that any column that is not within an aggregation must show up in your GROUP BY statement. If you forget, you will likely get an error. However, in the off chance that your query does work, you might not like the results!

**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; |

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**;

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 sr.name, Count(we.channel) countc , we.channel  From sales\_reps sr  Join account a  On a.sales\_rep\_id = sr. id  Join web\_events we  On we.account\_id = a.id  Group by (sr.name)  Order by countc DESC |

**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**

**DISTINCT** is always used in **SELECT** statements, and it provides the unique rows for all columns written in the **SELECT** statement. Therefore, you only use **DISTINCT** once in any particular **SELECT** statement.

**SELECT** **DISTINCT** column1, column2, column3

**FROM** table1;

Use **DISTINCT** to test if there are any accounts associated with more than one region.

The below two queries have the same number of resulting rows (351), so we know that every account is associated with only one region. If each account was associated with more than one region, the first query should have returned more rows than the second query.

**SELECT** a.**id** **as** "account id", r.**id** **as** "region id",

a.**name** **as** "account name", r.**name** **as** "region name"

**FROM** accounts a

**JOIN** sales\_reps s

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

**JOIN** region r

**ON** r.**id** = s.region\_id;

and

**SELECT** **DISTINCT** **id**, **name**

**FROM** accounts;

Actually all of the sales reps have worked on more than one account. The fewest number of accounts any sales rep works on is 3. There are 50 sales reps, and they all have more than one account. Using **DISTINCT** in the second query assures that all of the sales reps are accounted for in the first query.

**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**

**ORDER** **BY** num\_accounts;

and

**SELECT** **DISTINCT** **id**, **name**

**FROM** sales\_reps;

Where clause doesnot allow to filter when using aggregated function

### HAVING - Expert Tip

**HAVING** is the “clean” way to filter a query that has been aggregated, but this is also commonly done using a [subquery](https://community.modeanalytics.com/sql/tutorial/sql-subqueries/" \t "_blank). Essentially, any time you want to perform a **WHERE** on an element of your query that was created by an aggregate, you need to use **HAVING** instead.

Where is subsets that return data based on a logical condition

Where is written after select, from, join, on but before group by

Having appear after the group by clause, but before group by

Having can work on aggregated statement unlike while

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

|  |
| --- |
| Select sales\_reps.id, sales\_reps.name Count(\*) num\_account  From sales\_reps  Join accounts  on sales\_reps.id = accounts.sales\_rep\_id  Group by (sales\_reps.id)  Having Count(accounts.id) > 5 |

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

|  |
| --- |
| Select count(a.id), sum(o.total\_amt\_usd)account1  From accounts a  Join orders o  On a.id = o.account\_id  Group by(a.id)  Having account > 30000  order by account1 ; |

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

Select a.id , a.name, count(\*) count

From accounts a

Join web\_events we

a.id = we.accounts\_id

group by (a.id , a.name)

where we = “facebook”

Having count >6

**Which channel was most frequently used by most accounts?**

Select from a.id, a.name, w.channel count(\*) count1

From accounts as a

Join web\_events w

On a.id = w.account\_id

Group by a.id, a.name , w.channel

Order by count

**Date**

Date\_trunc:

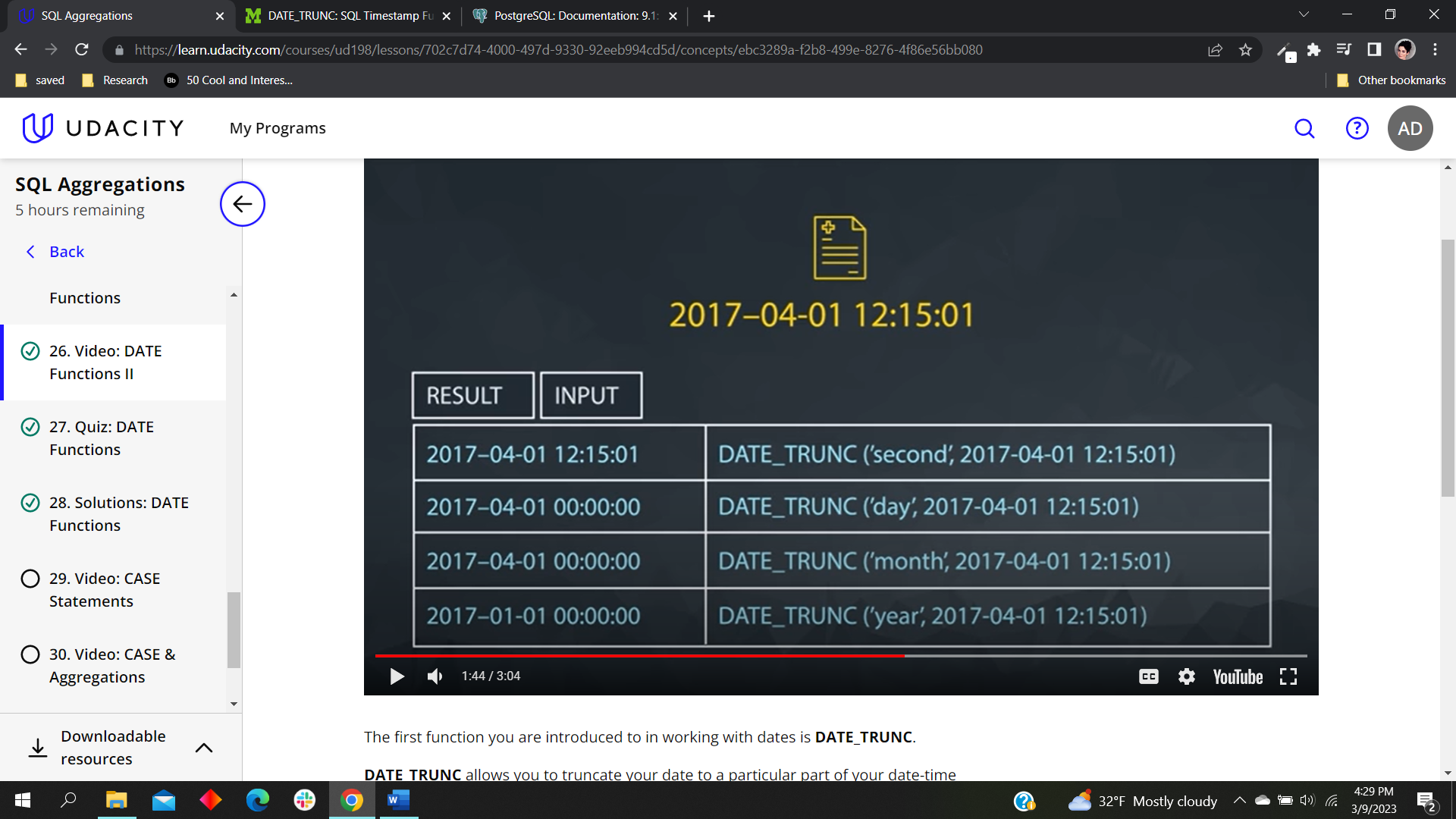


This function provides the information of all the events happening in with in 24 hours of 2017-04-01 00:00:00 if you decided to trunc in day

Similarly, provide information of weeks in you decided to trunc in week

Date\_part:

While Day\_part helps to pull out specific part from the given date time stamp then we use Date\_part



In the above figure notices that if we are trying to take information in case

**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', o.occurred\_at) or\_year, sum(o.total\_amt\_usd) total  From orders o  Group by 1  order by 2 desc |

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

**Note: If Data is not evenly represented then, this tutorial suggest not to consider**

|  |
| --- |
| **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**; |

**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('month', occurred\_at) ord\_month, COUNT(\*) **total\_sales**  **FROM** orders  **GROUP** BY 1  **ORDER** BY 2DESC**;** |

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

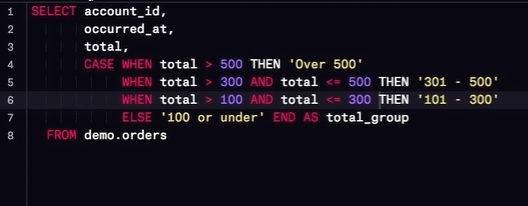
|  |
| --- |
| **SELECT DATE\_TRUNC**('month', o.occurred\_at) ord\_date, SUM(o.gloss\_amt\_usd) tot\_spent  **FROM** orders o  **JOIN** accounts a  ON a.id = o.account\_id  **WHERE** a.name = 'Walmart'  **GROUP** BY 1  **ORDER** BY 2 DESC |

**Difference between Date\_trunc and Date\_Part**

**Example:** If you want to see in which year’s month has the highest sales this time Date\_Part using ‘month’ is unable to do it because it will add up all the months from different years.

Date truct based on the months will divided all the months form each year an make a separate data set for each year.

**CASE:**

****

**Write a query to display the number of orders in each of three categories, based on the total number of items in each order. The three categories are: 'At Least 2000', 'Between 1000 and 2000' and 'Less than 1000**

**We would like to understand 3 different branches 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), date\_part ('year', o.occurred\_at) date2,**

**CASE**

**When SUM (total\_amt\_usd) > 200000 then 'top'**

**When SUM (total\_amt\_usd) > 100000 then 'middle'**

**Else 'low' END AS level**

**From orders as o**

**Join Accounts as a**

**On a.id = o.account\_id**

**Where date\_part ('year', o.occurred\_at) = 2016 or date\_part ('year' , o.occurred\_at) = 2017**

**Group by (1,3)**

**Order by 2 desc**

**Write a query to display for each order, the account ID, total amount of the order, and the level of the order - ‘Large’ or ’Small’ - depending on if the order is $3000 or more, or smaller than $3000?**

|  |
| --- |
| Select account\_id ,  Total\_amt\_usd,  CASE when Total\_amt\_usd > 3000 Then “Large”  ELSE “Smaller” end as level  From orders |

**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(\*)**

**CASE**

**When count(\*) >200 then top**

**Else ‘ not’ End as level**

**From orders o**

**Join accounts a**

**On a.id = o.account\_id**

Join sales\_Reps s

On s.id = a .sales\_reps\_id

Group by 1

Order by 2 desc

**NOTE : the order for placing joins of different table is** commutative. Unless it is a inner full join

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 500000 in 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(\*) num\_ords, sum(o.total),**

**CASE WHEN COUNT(\*) > 200 or sum(o.total)>750000 THEN 'top'**

**when count(\*) > 150 or sum(o.total)>500000 then 'middle'**

**ELSE 'not' END AS sales\_rep\_level**

**FROM orders o**

**JOIN account 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;**

**SUB-QUERY:**

Innner query runs first

Then the out query run

**Find the number of events that occur for each day for each channel**

|  |
| --- |
| **Select date\_trunc('day', occurred\_at) as day, channel , Count(\*)**  **From web\_events**  **Group by (1 , 2)**  **order by 1 desc** |

**Find the average number of events for each channel.**

|  |
| --- |
| **select channel, avg(events\_count)as avg\_count**  **from**  **(Select date\_trunc('day', occurred\_at) as day, channel , Count(\*) as events\_count**  **From web\_events**  **Group by (1 , 2)**  **)sub**  **group by (channel)**  **order by 2 desc ;** |

**Uses date\_trunc to pull month level information about the first order ever placed in the order table. Use the result of the previous query to find the orders that took place in the same month and year as first order, and then pull the average for each type of paper qty in this month.**

|  |
| --- |
| **Select avg(standard\_qty) as std\_Avg, avg(gloss\_qty) as g\_avg, avg(poster\_qty) as p\_avg**  **from orders**  **where date\_trunc('month' , occurred\_at) =**  **(Select date\_trunc('month',min(occurred\_at))**  **from orders);** |

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

**------------------------------------------------------------------------------------------------------------------------------------**

Select rname, max(total)

From (

Select r.name as rname , sr.name as rpname , sum(o.total\_amt\_usd) total

From sales\_reps sr

Join region r

On sr.region\_id = r.id

Join accounts a

On a.sales\_rep\_id = sr.id

Join orders

On a.id = o.account\_id

Group by (1 , 2)

) t1

Group by 1

**------------------------------------------------------------------------------------------------------------------------------------**

SELECT region\_name,rep\_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 (1,2)

order by 3 desc

limit 1  
**------------------------------------------------------------------------------------------------------------------------------------**

why dose not the second solution work ?

Because first inner query returns the region name, representative name and corresponding sum pf the total\_amount\_usd made in that region

Second outer query we look for maximum total\_amt\_usd in each region we cannot use name since this will return maximum total\_amt\_usd in that region rather it will return each combination of name And Region as we have to mention representative name in the group by function

So instead what we do is that we check for the region and maximum amount came form t2 table with the value of t1 table (Inner query) The matching will be the resulted output for the problem.

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

**First:**

**Select r.name, count(o.total) count\_total , sum(total\_amt\_usd)**

**Form region r**

**Join sales\_reps sr**

**On sr.id = r.id**

**Join accounts a**

**On a.sales\_rep\_id = sr.id**

**Join orders o**

**On a.id = o.account\_id**

**Group by r.name**

**Having sum(total\_amt\_usd) =**

**(**

**Select max(stau)**

**From**

**(**

**Select r.name, sum(o.total\_Amt\_usd) stau**

**From region r**

**Join sales\_reps sr**

**On sr.id = r.id**

**Join accounts a**

**On a.sales\_rep\_id = sr.id**

**Join orders o**

**On a.id = o.account\_id**

**Group by 1 ):t1**

**):t2**

**)**

**Note:** Aggregation function doesnot work with where so use having instead

**How many accounts had more total purchases than the account name which has bought the most standard\_qty paper throughout their lifetime as a customer?**

First finding the account which has the most standard\_qty paper:

Select a.name, sum(o.standard\_qty)

From orders o

Join accounts a

On a.id = o.account\_id

Group by (a.name)

Order by sum(o.standard\_qty) desc

Limt 1

Note: just be careful with the bracket while closing inner sub query when sub query return only one value instead of row and used inside having there is no need for alise;

select count(\*)

from(

Select a.name, sum(o.total) total

From accounts a

Join orders o

On a.id = o.account\_id

Group by (a.name)

Having sum(o.total) >

(

select total\_1 from

(

Select a.name, sum(o.standard\_qty) , sum(o.total) total\_1

From orders o

Join accounts a

On a.id = o.account\_id

Group by (a.name)

Order by sum(o.standard\_qty) desc

Limit 1) inner\_query

)

)outer\_tab;

**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?**

Select a.name , we.channel , Count(\*)

From web\_events we

join accounts a

On a.id = we.account\_id AND a.id =

(

Select id

from (

Select a.id , sum(o.total\_Amt\_usd ) total\_amt

From accounts a

Join orders o

On o.account\_id = a.id

Group by (a.id)

Order by 2

Limit 1 ) sub1

)

Group by we.channel , a.name

Order by 3 desc

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

The question is asking for average of to 10 spenders:

Select avg(sped\_Amt)

Select a.id, sum(o.total\_amt\_usd) sped\_Amt

From accounts a

Join orders o

On a.id = o.account\_id

Group by a.id,

Order by 2 desc

Limit 10;

**6. What is the lifetime average amount spent in terms of \*\*total\_amt\_usd\*\*, including only the companies that spent more per order, on average, than the average of all orders.**

This problem asks about those clients who has more spending per order than average spending

Example a companies can have multiple orders with corresponding amount ie. Multiple value of total\_Amt\_usd so an account can have its own average per order based on total\_amt\_usd. Then find the average of the value of obtained all value.

select avg(avg\_amt)

from

(

Select a.id ,avg(o.total\_Amt\_usd) avg\_amt

From accounts a

Join orders o

On o.account\_id = a.id

Group by (a.id)

Having avg(o.total\_Amt\_usd) >

(

Select avg(total\_Amt\_usd) avg

from orders o))temp;

**WITH in query:**

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

**Select** r.name, sr.name, max(sum\_total)

**Select** r.name, sr.name, sum(total\_amt\_usd) sum\_total

From region r

Join sales\_reps sr

On sr.region\_id = r.id

Join accounts a

On sr.id = a.sales\_rep\_id

Join orders o

On a.id = o.account\_id

Group by (1 , 2)

Order by 3

Select rname, max(total)

From (

Select r.name as rname , sr.name as rpname , sum(o.total\_amt\_usd) total

From sales\_reps sr

Join region r

On sr.region\_id = r.id

Join accounts a

On a.sales\_rep\_id = sr.id

Join orders

On a.id = o.account\_id

Group by (1 , 2)

) t1

Group by 1

**Select r.name, max(sum\_total)**

**from(**

**Select r.name, sr.name, sum(total\_amt\_usd) sum\_total**

**From region r**

**Join sales\_reps sr**

**On sr.region\_id = r.id**

**Join accounts a**

**On sr.id = a.sales\_rep\_id**

**Join orders o**

**On a.id = o.account\_id**

**Group by (1, 2)**

**)sub**

**group by 3**

**Note: Be careful with the braces take notice while using subquery when it is used in condition there is no need of using the alias of query.**

------------------------------------------------------------------------------------------------------------------------------------

**Now let’s try using with:**

**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),**

**)**

**As t2 (**

**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;**

------------------------------------------------------------------------------------------------------------------------------------2.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

Joins 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 account 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);

**For 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 AS t1**

**(**

**Select a.name account\_name, sum(o.standard\_qty) std\_amt , sum(o.total) sum\_total**

**From accounts a**

**Join orders o**

**On a.id = o.account\_id**

**Group by (a.name)**

**Order by 2 desc**

**Limit 1**

**)**

**As t2**

**(**

**Select a.name**

**From orders o**

**Join accounts a**

**ON a.id = o.account\_id**

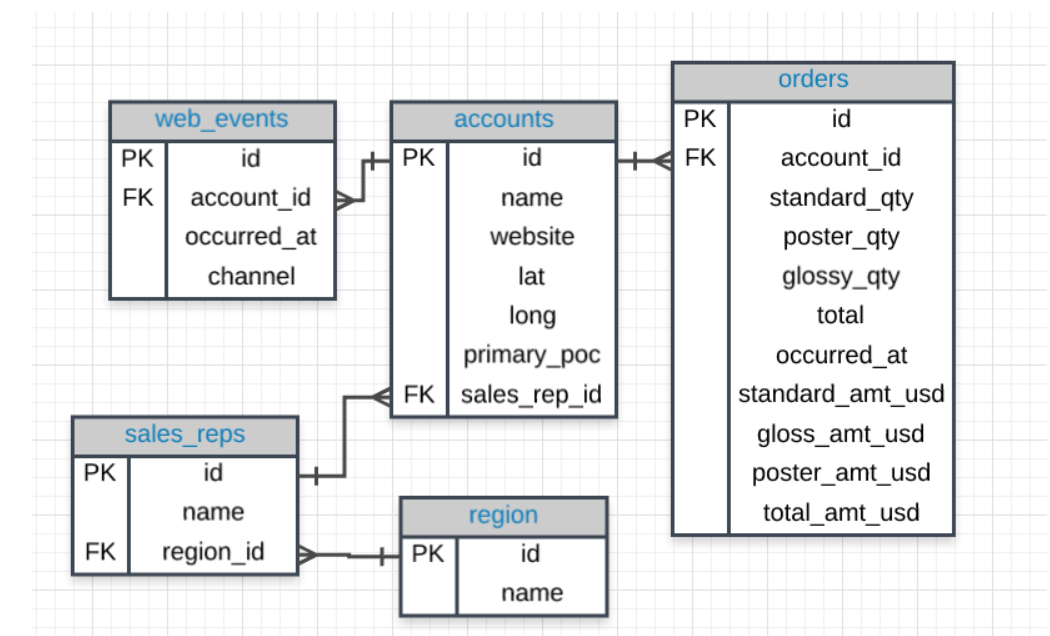
**Group by 1**

**Having Sum(o.total) > (Select sum\_total from t1 ))**

**Select Count(\*)**

**From t2 ;**

4.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 aid , a.name a\_name, sum(total\_amt\_usd) amt\_usd

From accounts a

Join orders o

On o.account\_id = a.id

Group by (1,2)

Order by 3 desc

Limit 1

),

t2 As

(

Select a.id aid2, a.name aname,we.channel we\_channel2

From accounts a

Join web\_events we

On we.account\_id = a.id

Group by 1 , 2 ,3

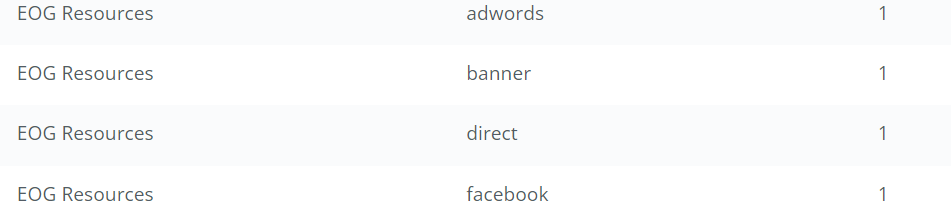
)

Select aname , we\_channel2 ,count(\*)

From t2

Where t2.aid2 = (select aid from t1)

group by (1,2)



**# why does not this query work ?**

**WITH t1 AS (**

**SELECT a.id id\_1, a.name 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(\*)**

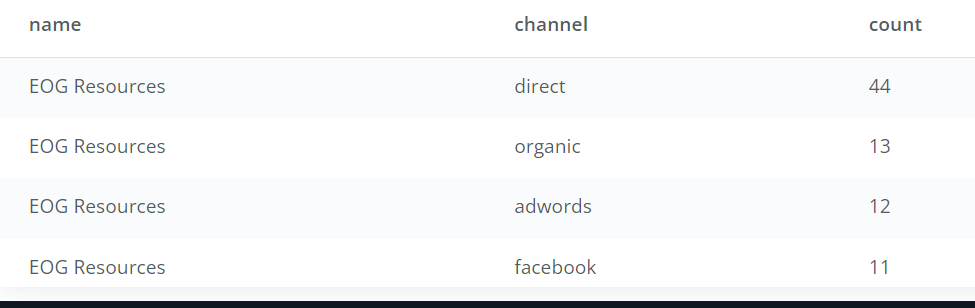
**FROM accounts a**

**JOIN web\_events w**

**ON a.id = w.account\_id AND a.id = (SELECT id\_1 FROM t1)**

**GROUP BY 1, 2**

**ORDER BY 3 DESC;**



**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 t2 as (

Select a.id , a.name , sum(o.total\_Amt\_usd) tot\_spend

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\_spend)

From t2;

**6. What is the lifetime average amount spent in terms of \*\*total\_amt\_usd\*\*, including only the companies that spent more per order, on average, than the average of all orders.**

With t1 AS

(

Select AVG(o.total\_Amt\_usd) avgall

From orders o

),

t2 as(

SeLECT a.id , a.name, avg(o.total\_amt\_usd) each\_Avg

From accounts a

Join orders o

On o.account\_id = a.id

Group by (1 ,2 )

Having avg(o.total\_amt\_usd) > ( Select avgall from t1 )

)

Select avg(each\_Avg)

from t2

Initially, what I did wrong was I found out the average of those account that have average more than the average\_of\_all but did not find the average of all the (those account)

**SQL Data Cleaning:**

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" \t "_blank)**. Pull these extensions and provide how many of each website type exist in the **accounts** table.

|  |
| --- |
| **Select** Right (website, 3) as domain, count (\*)  **From** account  **Group by** 1  **Order by** 2 desc |

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" \t "_blank)** 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).

**FROM accounts**

**GROUP BY 1**

SE**LECT LEFT(name, 1) AS first\_letter, COUNT(\*) num\_companies**

**ORDER BY 2 DESC;**

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?

|  |
| --- |
| **With table\_1 as**  **(**  **SELECT name, CASE WHEN LEFT (UPPER (name), 1) IN ('0','1','2','3','4','5','6','7','8','9') //(‘A’, ‘E’, ‘I’ , ‘O’, ‘U’) for starting with non vowel**  **THEN 1 ELSE 0 END AS num,**  **CASE WHEN LEFT (UPPER (name), 1) not IN ('0','1','2','3','4','5','6','7','8','9') //(‘A’, ‘E’, ‘I’ , ‘O’, ‘U’)**  **THEN 1 ELSE 0 END AS letter**  **FROM accounts**  **)**  **select sum(table\_1.letter) as starting\_with\_letter,**  **sum (Table\_1.num) as starting\_with\_number**  **from table\_1** |

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

**POSITION: return the position of specific character in string using**

**STRPOS: same as position but the format is different**

**LOWER: Place into lower case to all the document**

**UPPER: Place everything into upper case**

|  |
| --- |
| Select primary\_poc,  left(primary\_poc ,Position(' 'in primary\_poc)) firstname ,  right(primary\_poc ,LENGTH(primary\_poc) -(position(' 'in primary\_poc)) )lastname  from accounts |

**Combining the two string columns into one column we have:**

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 primary\_poc,  left(primary\_poc, STRPOS(primary\_poc,' ')) first\_nam,  right(primary\_poc, length(primary\_poc) -STRPOS(primary\_poc,' '))last\_nam  from accounts  )  select first\_nam, last\_nam,  CONCAT(REPLACE(first\_nam, ' ', ''), '.',last\_nam, '@' ,'.com') as email  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 primary\_poc, name,

left(primary\_poc, STRPOS(primary\_poc,' ')) first\_nam,

right(primary\_poc, length(primary\_poc) -STRPOS(primary\_poc,' '))last\_nam

from accounts

)

select first\_nam, last\_nam,

CONCAT

(

left (lower((REPLACE(first\_nam,' ',''))),1),

right(lower((REPLACE(first\_nam,' ',''))),1),

left(lower((REPLACE(last\_nam,' ',''))),1),

right(first\_nam,1),

Length(first\_nam),

length(last\_nam),

Upper(name)

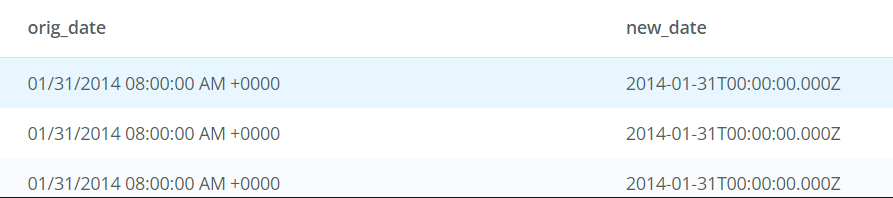
)

from t1 ;

**CAST:**

|  |
| --- |
| SELECT date orig\_date, (SUBSTR(date, 7, 4) || '-' || LEFT(date, 2) || '-' || SUBSTR(date, 4, 2))::DATE new\_date  FROM sf\_crime\_data; |

CAST wil convert one data type into another data type . In this case Cast will convert String into date.



**COLASE:**

Replace null values in the database

Coalesce ::

|  |
| --- |
| **COALESCE will return the first not null value in mention columns**    **COALESCE(o.id , a.id) This will return the first not null value from the combination of O.id and a.id**  **SELECT** **COALESCE**(o.id, a.id) filled\_id, a.name, a.website, a.lat, a.long, a.primary\_poc, a.sales\_rep\_id, o.\*   1. **FROM** accounts a 2. **LEFT** **JOIN** orders o 3. **ON** a.id = o.account\_id 4. **WHERE** o.total IS NULL; |

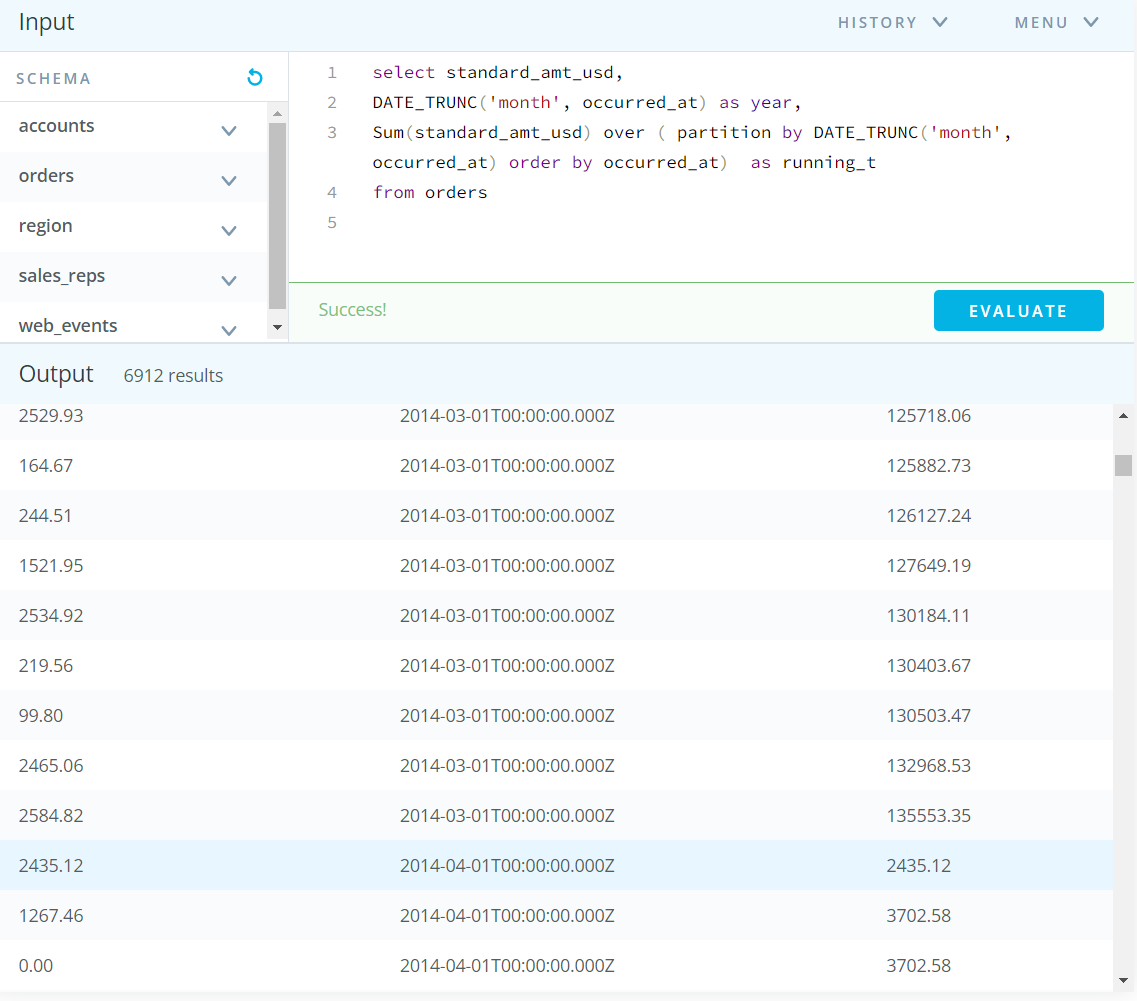
|  |
| --- |
| Run the query that will show the row with missing data:   1. **SELECT** \* 2. **FROM** accounts a 3. **LEFT** **JOIN** orders o 4. **ON** a.id = o.account\_id 5. **WHERE** o.total IS NULL;     **Why does this work?**   1. **SELECT** \* 2. **FROM** orders o 3. **LEFT** **JOIN** accounts a 4. **ON** a.id = o.account\_id 5. **WHERE** o.total IS NULL;     **This does not work because this will look for total column in SQL table where total is empty not the corresponding account id where account id is null.**  This gives the output of those cases where o.total is null but still have the customer account and information on.  **Use COLASE to fill in each of the qty and usd columns with 0 for the table in 1**    **SELECT COALASE** (o.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.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**  **Run the query in 1 with the were removed and count the number of id s.**    **Run the query in 5 but with coalesce function used in question 2 through 4**   1. **SELECT** **COALESCE**(o.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 2. **FROM** accounts a 3. **LEFT** **JOIN** orders o 4. **ON** a.id = o.account\_id;   NOTE: COALESCE can do two functions first return the first not null value among the column mention, Second it replaces the value form the column. |
|  |

**#Window function:**

**Partition by:**

To narrow the window from entire dataset to individual groups within a dataset:

**Eg: In this example the running total is calculated based on each month by using the partition function.**

****

**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,**

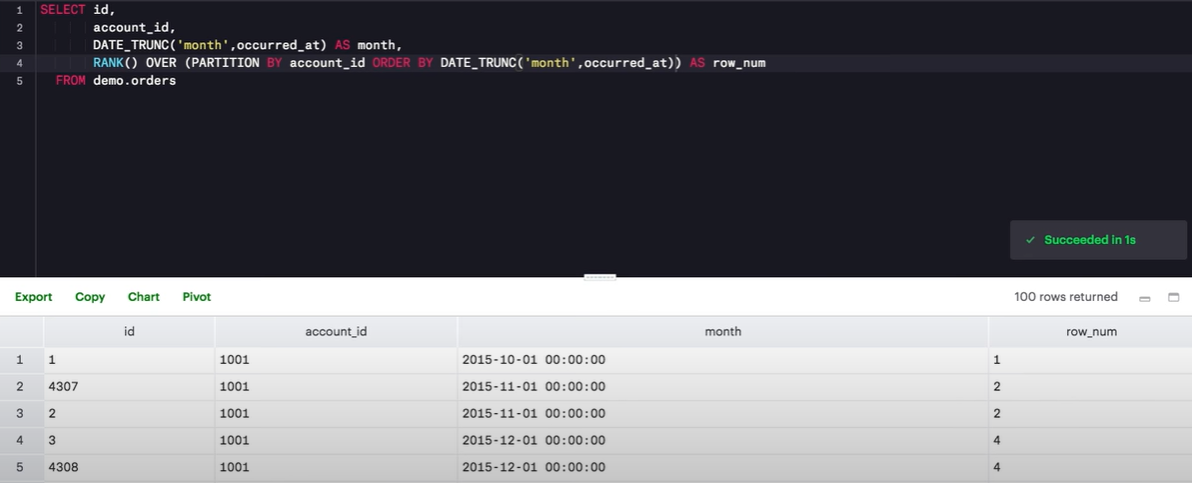
**total,**

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

**FROM orders**

I miss understood and suppose that the total\_rank is the based on the sum of the total not the given total.

The difference between the rank and row is that if a column has the same value the thy will have same rank unlike row, they will have different row.

Entry with same month has same rank but the value is skipped i.e 3 to directlty 4 not 3.

This problem is solved by the Dense\_Rank function

**Aggregates in window:**

Rank with partition by and order by function

Rank\_dense with partition by and order by function

Count\_row with partition by and order by function

|  |
| --- |
| **SELECT** id,  account\_id,  standard\_qty,  DATE\_TRUNC('month', occurred\_at) **AS** **month**,  DENSE\_RANK() **OVER** (**PARTITION** **BY** account\_id **ORDER** **BY** DATE\_TRUNC('month',occurred\_at)) **AS** dense\_rank,  **SUM**(standard\_qty) **OVER** (**PARTITION** **BY** account\_id **ORDER** **BY** DATE\_TRUNC('month',occurred\_at)) **AS** sum\_std\_qty,  **COUNT**(standard\_qty) **OVER** (**PARTITION** **BY** account\_id **ORDER** **BY** DATE\_TRUNC('month',occurred\_at)) **AS** count\_std\_qty,  **AVG**(standard\_qty) **OVER** (**PARTITION** **BY** account\_id **ORDER** **BY** DATE\_TRUNC('month',occurred\_at)) **AS** avg\_std\_qty,  **MIN**(standard\_qty) **OVER** (**PARTITION** **BY** account\_id **ORDER** **BY** DATE\_TRUNC('month',occurred\_at)) **AS** min\_std\_qty,  **MAX**(standard\_qty) **OVER** (**PARTITION** **BY** account\_id **ORDER** **BY** DATE\_TRUNC('month',occurred\_at)) **AS** max\_std\_qty  **FROM** orders |

For more details:

<https://stackoverflow.com/questions/41364665/analytic-count-over-partition-with-and-without-order-by-clause>

# **Alias: Aliases for Multiple Window Functions**

If you create an Aliases for multiple Window function:

SELECT id,

account\_id,

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

DENSE\_RANK() OVER main\_window AS dense\_rank,

total\_amt\_usd,

SUM(total\_amt\_usd) OVER main\_window AS sum\_total\_amt\_usd,

COUNT(total\_amt\_usd) OVER main\_window AS count\_total\_amt\_usd,

AVG(total\_amt\_usd) OVER main\_window AS avg\_total\_amt\_usd,

MIN(total\_amt\_usd) OVER main\_window AS min\_total\_amt\_usd,

MAX(total\_amt\_usd) OVER main\_window AS max\_total\_amt\_usd

FROM orders

Window main\_window as (PARTITION BY account\_id ORDER BY DATE\_TRUNC('year',occurred\_at))

Here we can clearly see the window created by partition and ordered by function in which the aggregation function is applied.

**Comparing a row to preceding row**: In case you go the data that makes sense:

Using lag and lead difference:

Lag difference : Shows the difference between current row and prior row

Lead difference : Shows the difference between current row and next row

Lag : One step behind one’s (before itself) compare with Prior row

Lead = next step after us (after itself) compare with 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 (Ordered by standard\_sum as lag\_difference  LEAD(standard\_sum) Over (Ordered by standard\_sum) \* standard\_sum as load\_difference  From (  Select Account\_id,  ) |

**Percentile:**

**NTILE** Function allows you see the percentile :

You can see three percentile:

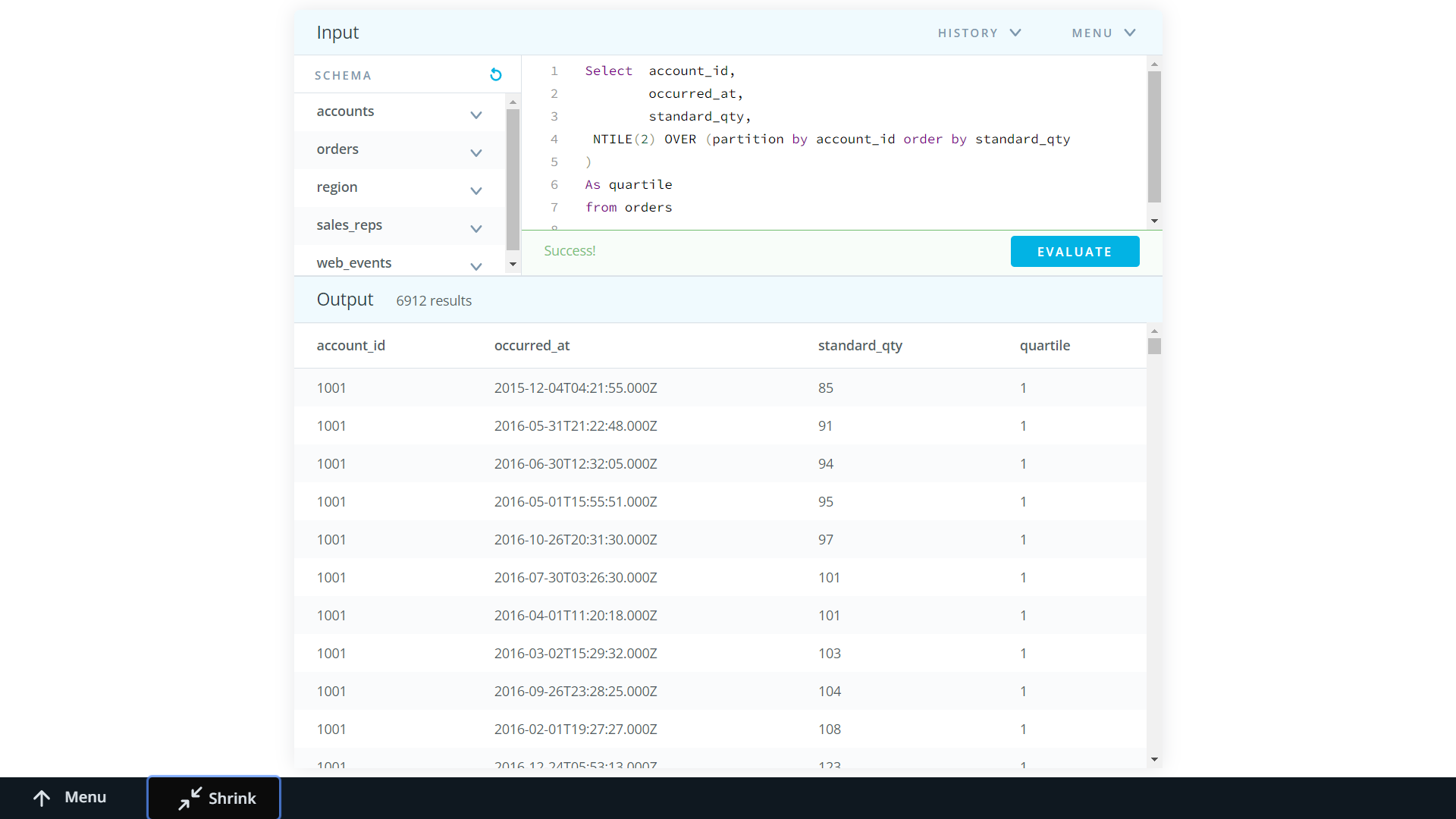
100

5

4

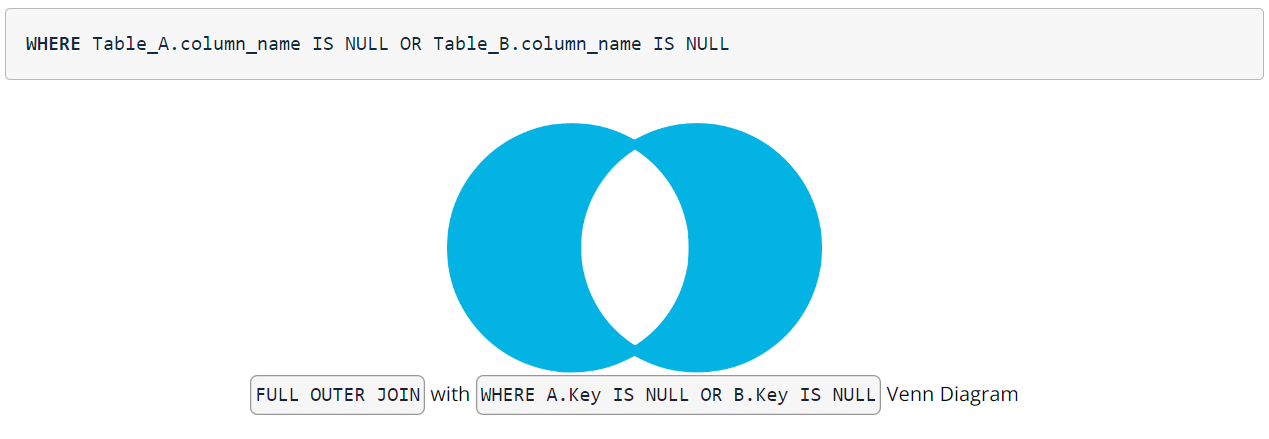
Use the NTILE functionality to divide the accounts into 4 levels in terms of the amount of standard\_qty for their orders. Your resulting table should have the account\_id, the occurred\_at time for each order, the total amount of standard\_qty paper purchased, and one of four levels in a standard\_quartile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

|  |
| --- |
| Select account\_id,  occurred\_at,  standard\_qty,  NTILE(2) OVER (partition by account\_id order by standard\_qty  )  As quartile  from orders |



**Full Outer Joins:**

we covered inner joins, which produce results for which the join condition is matched in both tables



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:

a.primary\_poc < sr.name

This means: The primary Point of contact’s full name comes before the sales representative’s name alphabetically:

**Self Joins:**

We used the self joins to find the events which occur one after another:

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

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'

#wanting the record in oo2 to be with in 28 days after a record in o1

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

**Modify the query from the previous video, which is pre-populated in the SQL Explorer below, to perform the same interval analysis except for the web\_events table. Also:**

* **change the interval to 1 day to find those web events that occurred after, but not more than 1 day after, another web event**
* **add a column for the channel variable in both instances of the table in your query**

**SELECT** we1.id **AS** we\_id,

we1.account\_id **AS** we1\_account\_id,

we1.occurred\_at **AS** we1\_occurred\_at,

we1.channel **AS** we1\_channel,

we2.id **AS** we2\_id,

we2.account\_id **AS** we2\_account\_id,

we2.occurred\_at **AS** we2\_occurred\_at,

we2.channel **AS** we2\_channel

**FROM** web\_events we1

**LEFT** **JOIN** web\_events we2

**ON** we1.account\_id = we2.account\_id

AND we1.occurred\_at > we2.occurred\_at

This join this will gives the data of web\_Event that occur before a event that happen in same table

AND we1.occurred\_at <= we2.occurred\_at + **INTERVAL** '1 day'

Then this will filter out those events that occur one day before that event

**ORDER** **BY** we1.account\_id, we2.occurred\_at

The above code will give us the Information

**UNION:**

**Select**

**Pre-Requisite:** both tables must have same number of columns

Must have same data Types as in same order

It is not necessary that the columns name needs to be same

Union appends the data set of two columns But when there is data that is exactly same as in the another columns then the column is dropped. (Not repeated twice)

To consider the same columns one should one should use the Union all.

**Performance tuning:**

This can be done using:

Subqueries or as with

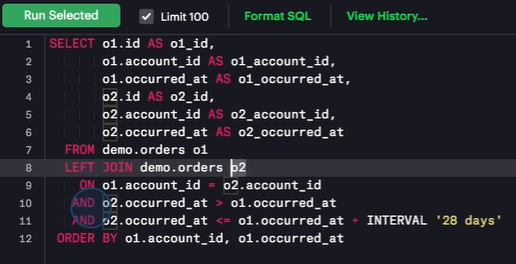
Delete the unwanted processing required to process the query

#One can add explain to see the details of Query plan which will show the orders as well as the time taken to execute the query

**An example of Non-Equi Self Join**

Joins with inequality operator:

One of the interesting techniques in SQL I came across during the course for SQl\_For\_data Analysis is Self with inequality Joins with inequality. This was one of the smoothest tricks I found about SQL



In the Query shown above picture:

**Desired Data: Which account made more orders with in 28 days**

Record in O2 to be with in 28 days after the records in O1

**Line7:** Joins table to itself on account id of table order

**Line10**: Finds the orders that happen after the original order is placed.

i.e orders that happens after the O1 orders is placed. Suppose Taking an order from O1 table and finding the orders that happens after that O1 table order in O2 Table (made by same id since table is join ON O1.account\_id = O2.account\_id).

**Line11**: Now from the above filtered table we find orders in O2 to be with in the 28 days of orders in O1 made by same id.

Source: <https://learnsql.com/blog/illustrated-guide-sql-non-equi-join/>

<https://learnsql.com/blog/sql-non-equi-joins-examples/>