16 January 2023 10:56

PROBLEM STATEMENT: Find the email activity rank for each user. Email activity rank is defined by the total number of emails sent. The user with the highest number of emails sent will have a rank of 1, and so on. Output the user, total emails, and their activity rank. Order records by the total emails in descending order. Sort users with the same number of emails in alphabetical order. In your rankings, return a unique value (i.e., a unique rank) even if multiple users have the same number of emails.

google_gmail_emails

Field	Type
id	int
from_user	varchar
to_user	varchar
day	int

SAMPLE TABLE:

id	from_user	to_user	day
0	6edf0be4b2267df1fa	75d295377a46f83236	10
1	6edf0be4b2267df1fa	32ded68d89443e808	6
2	6edf0be4b2267df1fa	55e60cfcc9dc49c17e	10
3	6edf0be4b2267df1fa	e0e0defbb9ec47f6f7	6
4	6edf0be4b2267df1fa	47be2887786891367e	1
5	6edf0be4b2267df1fa	2813e59cf6c1ff698e	6

MENTAL APPROACH:

1. Find different users who have sent the emails and count the number of times same user has sent the email.
2. This will helps us to find the distinct users who have sent emails and the count of emails they have sent.
3. Now we have to provide rank on the basis of a number of emails and the user name who has sent the email.
   NOTE: Here, we need to rank on the basis of descending order of a number of emails and ascending order of user name. This was done so that each user is ranked with different numbers not the same even though they are having the same number of emails sent.

QUERY:

```
SELECT from_user
      ,COUNT(1) emails_sent
      ,DENSE_RANK() OVER (ORDER BY COUNT(1) DESC,from_user) activity_rank
FROM google_gmail_emails
GROUP BY from_user
```

QUERY EXECUTION:

1. Firstly FROM command will get executed and it will fetch the records from the table google_gmail_emails.
2. In the second step GROUP BY on the basis of from_user will be executed.
3. In third step AGGREGATE FUNCTION (here COUNT()) will be executed.
4. After that WINDOW FUNCTION (here DENSE_RANK()) on the basis of the OVER clause will be executed.
5. Finally SELECT clause will get executed.
6. So, after this we will get our final output which will display from_user, number of emails sent and rank provided.

SAMPLE OUTPUT:

from_user	emails_sent	activity_rank
32ded68d89443e808	19	1
ef5fe98c6b9f313075	19	2
5b8754928306a18b68	18	3
55e60cfcc9dc49c17e	16	4
91f59516cb9dee1e88	16	5
6edf0be4b2267df1fa	15	6
7cfe354d9a64bf8173	15	7
cbc4bd40cd1687754	15	8
e0e0defbb9ec47f6f7	15	9
8bba390b53976da0cd	14	10
a84065b7933ad01019	14	11

08 January 2023 08:58

Problem Statement: Find the average effectiveness of each advertising channel in the period from 2017 to 2018 (both included). The effectiveness is calculated as the ratio of total money spent to total customers aquired.

Output the advertising channel along with corresponding average effectiveness. Sort records by the average effectiveness in ascending order.

uber_advertising

Field	Type
year	int
advertising_channel	varchar
money_spent	int
customer_acquired	int

Query:

```sql
SELECT advertising_channel
      ,total_money_spent/total_customers average_effectiveness
FROM (
       SELECT advertising_channel
             ,SUM(money_spent) total_money_spent
             ,SUM(customers_acquired) total_customers
       FROM uber_advertising
       WHERE year BETWEEN 2017 AND 2018
       GROUP BY advertising_channel
     ) subquery
ORDER BY average_effectiveness
```

Output:

advertising_channel	average_effectiveness
radio	56
tv	58
busstops	82
celebrities	105
billboards	179
buses	827

Same Through Excel using Pivot Table

year	ertising_c	money_sp	omers_ac	ired
2019	celebrities	10000000	1800	
2019	billboards	1000000	2000	
2019	busstops	1500	400	
2019	buses	70000	2500	
2019	tv	300000	5000	
2019	radio	1500	51	
2018	celebrities	123555	2100	
2018	billboards	500000	1800	
2018	busstops	35000	600	
2018	buses	550000	2300	
2018	tv	500000	5300	
2018	radio	2500	250	
2017	celebrities	300000	1900	
2017	billboards	200200	2100	
2017	busstops	80000	800	
2017	buses	3589000	2700	
2017	tv	80000	4700	
2017	radio	80000	1200	

year	(Multiple Items)

Row Labels	Sum of customers_acquired	Sum of money_spent	Sum of Average Effectiveness
radio	1450	82500	56.89655172
tv	10000	580000	58
busstops	1400	115000	82.14285714
celebrities	4000	423555	105.88875
billboards	3900	700200	179.5384615
buses	5000	4139000	827.8
Grand Total	**25750**	**6040255**	**234.5730097**

23 January 2023 19:24

PROBLEM STATEMENT: Meta/Facebook is developing a search algorithm that will allow users to search through their post history. You have been assigned to evaluate the performance of this algorithm.

We have a table with the user's search term, search result positions, and whether or not the user clicked on the search result.

Write a query that assigns ratings to the searches in the following way:
• If the search was not clicked for any term, assign the search with rating=1
• If the search was clicked but the top position of clicked terms was outside the top 3 positions, assign the search a rating=2
• If the search was clicked and the top position of a clicked term was in the top 3 positions, assign the search a rating=3

As a search ID can contain more than one search term, select the highest rating for that search ID. Output the search ID and it's highest rating.

Example: The search_id 1 was clicked (clicked = 1) and it's position is outside of the top 3 positions (search_results_position = 5), therefore it's rating is 2.

FIELD	TYPE
search_id	int
search_term	varchar
clicked	int
search_results_position	int

MENTAL APPROACH:

1. First we will provide rating for each of the search id present in the given table as per the conditions provided to us.
2. Now find the maximum rating for distinct search id and get the desired output.
   NOTE: There are many same search id in the table so for distinct search id we will find the maximum rating.

```sql
SELECT search_id
      ,MAX(CASE WHEN clicked = 0 THEN 1
                WHEN clicked ≥ 1 AND search_results_position > 3 THEN 2
                WHEN clicked ≥ 1 AND search_results_position ≤ 3 THEN 3
        END) rating
FROM fb_search_events
GROUP BY search_id
```

QUERY EXPLANATION:

1. With CASE WHEN we are giving rating to each search_id as per the given condition.

   SAMPLE OUTPUT for CASE WHEN:

search_id	rating
1	2
2	2
2	2
3	3
3	2
5	3

2. We are also using MAX so that we can get maximum rating out of it and grouping them on the basis of search id.

   SAMPLE OUTPUT for whole Query:

search_id	rating
1	2
2	2
3	3
5	3

13 January 2023 19:01

Problem Statement: You are given a table, *BST*, containing two columns: *N* and *P,* where *N* represents the value of a node in *Binary Tree*, and *P* is the parent of *N*.

Column	Type
N	Integer
P	Integer

Write a query to find the node type of *Binary Tree* ordered by the value of the node. Output one of the following for each node:

- *Root*: If node is root node.
- *Leaf*: If node is leaf node.
- *Inner*: If node is neither root nor leaf node.

Sample Input

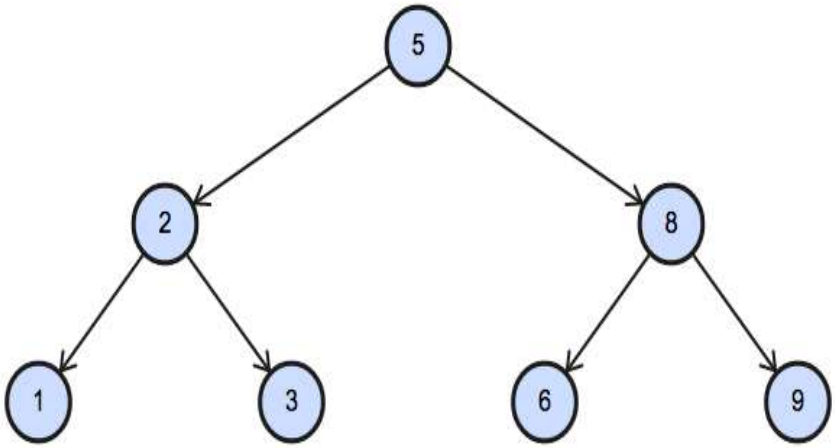
N	P
1	2
3	2
6	8
9	8
2	5
8	5
5	null

Sample Output

1 Leaf
2 Inner
3 Leaf
5 Root
6 Leaf
8 Inner
9 Leaf

Explanation

The *Binary Tree* below illustrates the sample:



SOLUTION

Mental Approach:

1. If N (node) is not having any value in P (parent node) that means it will be a Root node.

2. If for N (node) we are having both P (parent node) and if it is itself P (parent node) for other N (nodes) that means it is an Inner node.

3. Now after doing the above steps we will be left with Leaf Node. Actually, if N (node) is not P (parent node) of any other N (nodes) then we can say it is a Leaf node.

QUERY:

```
SELECT DISTINCT  N
      ,CASE WHEN P IS NULL THEN "Root"
            WHEN N NOT IN (SELECT DISTINCT P
                           FROM BST
                           WHERE P IS NOT NULL) THEN "Leaf"
        ELSE "Inner"
        END Type
FROM BST
```

QUERY EXPLANATION:

1. We are using Distinct because there is duplicate output.

2. In the first condition of the CASE statement we are simply searching for all Root nodes. As we have given the condition that if the parent node is null then it will be the Root node.

3. In the second condition we are simply searching for all Leaf nodes. Here, we have given the condition that N (node) should not be present in the parent node column.

   Note: Here we have excluded values for P (parent node) that is having Null values because with NOT IN we can't compare the null values.

4. In ELSE I am simply giving Inner because after both conditions are fulfilled we will be left with Inner nodes only.

ACTUAL OUTPUT:

N	Type
1	Leaf
2	Inner
3	Leaf
4	Inner
5	Leaf
6	Inner
7	Leaf
8	Leaf
9	Inner
10	Leaf
11	Inner
12	Leaf
13	Inner
14	Leaf
15	Root

Output with both Node and Parent Node
For better clarification

N	P	Type
1	2	Leaf
2	4	Inner
3	2	Leaf
4	15	Inner
5	6	Leaf
6	4	Inner
7	6	Leaf
8	9	Leaf
9	11	Inner
10	9	Leaf
11	15	Inner
12	13	Leaf
13	11	Inner
14	13	Leaf
15	NULL	Root

Case Study #1 - Danny's Diner

Danny Ma · May 1, 2021

Introduction

Danny seriously loves Japanese food so in the beginning of 2021, he decides to embark upon a risky venture and opens up a cute little restaurant that sells his 3 favourite foods: sushi, curry and ramen.

Danny’s Diner is in need of your assistance to help the restaurant stay afloat - the restaurant has captured some very basic data from their few months of operation but have no idea how to use their data to help them run the business.

Problem Statement

Danny wants to use the data to answer a few simple questions about his customers, especially about their visiting patterns, how much money they’ve spent and also which menu items are their favourite. Having this deeper connection with his customers will help him deliver a better and more personalised experience for his loyal customers.

He plans on using these insights to help him decide whether he should expand the existing customer loyalty program - additionally he needs help to generate some basic datasets so his team can easily inspect the data without needing to use SQL.

Danny has provided you with a sample of his overall customer data due to privacy issues - but he hopes that these examples are enough for you to write fully functioning SQL queries to help him answer his questions!

Danny has shared with you 3 key datasets for this case study:

- sales
- menu
- members

You can inspect the entity relationship diagram and example data below.

Entity Relationship Diagram

Example Datasets

All datasets exist within the dannys_diner database schema - be sure to include this reference within your SQL scripts as you start exploring the data and answering the case study questions.

Table 1: sales

The sales table captures all customer_id level purchases with an corresponding order_date and product_id information for when and what menu items were ordered.

Table 2: menu

The `menu` table maps the `product_id` to the actual `product_name` and `price` of each menu item.

Table 3: members

The final `members` table captures the `join_date` when a `customer_id` joined the beta version of the Danny's Diner loyalty program.

Interactive SQL Session

You can use the embedded DB Fiddle below to easily access these example datasets - this interactive session has everything you need to start solving these questions using SQL.

You can click on the `Edit on DB Fiddle` link on the top right hand corner of the embedded session below and it will take you to a fully functional SQL editor where you can write your own queries to analyse the data.

You can feel free to choose any SQL dialect you'd like to use, the existing Fiddle is using PostgreSQL 13 as default.

Serious SQL students have access to a dedicated SQL script in the 8 Week SQL Challenge section of the course which they can use to generate relevant temporary tables like we've done throughout the entire course!

Case Study Questions

Each of the following case study questions can be answered using a single SQL statement:

1. What is the total amount each customer spent at the restaurant?
2. How many days has each customer visited the restaurant?
3. What was the first item from the menu purchased by each customer?
4. What is the most purchased item on the menu and how many times was it purchased by all customers?
5. Which item was the most popular for each customer?
6. Which item was purchased first by the customer after they became a member?
7. Which item was purchased just before the customer became a member?
8. What is the total items and amount spent for each member before they became a member?
9. If each $1 spent equates to 10 points and sushi has a 2x points multiplier - how many

points would each customer have?

10. In the first week after a customer joins the program (including their join date) they earn 2x points on all items, not just sushi - how many points do customer A and B have at the end of January?

C	2021-01-07	ramen	12	N	null

Danny's Diner

30 November 2022 12:02

```
--1. What is the total amount each customer spent at the restaurant?
SELECT s.customer_id,
       SUM(m.price) total_amount
FROM sales s
LEFT JOIN menu m
ON s.product_id=m.product_idsq
GROUP BY s.customer_id
ORDER BY s.customer_id;
```

customer_id	total_amount
A	76
B	74
C	36

```
--2.How many days has each customer visited the restaurant?
SELECT customer_id,
       COUNT(DISTINCT order_date) no_of_days_visited
FROM sales
GROUP BY customer_id;
```

customer_id	no_of_days_visited
A	4
B	6
C	2

```
--3.What was the first item from the menu purchased by each customer?
SELECT customer_id,
      STRING_AGG(product_name,' , ') most_purchased_product
FROM (SELECT DISTINCT s.customer_id,
             m.product_name,
             RANK() OVER (PARTITION BY s.customer_id ORDER BY order_date)
rank
      FROM sales s
      LEFT JOIN menu m
      ON s.product_id=m.product_id
      ) subquery
WHERE rank=1
GROUP BY customer_id;
```

customer_id	most_purchased_product
A	curry , sushi
B	curry
C	ramen

```
--4. What is the most purchased item on the menu and how many times was it
purchased by all customers?
SELECT TOP 1 m.product_name,
       COUNT(1) number_of_times_purchased
FROM sales s
LEFT JOIN menu m
ON s.product_id=m.product_id
GROUP BY m.product_name
ORDER BY number_of_times_purchased DESC;
```

product_name	number_of_times_purchased
ramen	8

```
--5. Which item was the most popular for each customer?
SELECT customer_id
      ,STRING_AGG(product_name,' , ') popular_dish
FROM (
SELECT  s.customer_id
        ,m.product_name
        ,COUNT(1) no_of_orders
        ,RANK() OVER (PARTITION BY s.customer_id ORDER BY COUNT(1) DESC) rnk
FROM sales s
INNER JOIN menu m
ON s.product_id=m.product_id
GROUP BY s.customer_id
         ,m.product_name
         ) subquery
WHERE rnk=1
GROUP BY customer_id;
```

customer_id	popular_dish
A	ramen
B	sushi , curry , ramen
C	ramen

```
--6. Which item was purchased first by the customer after they became a
member?
WITH rank_cte AS (
SELECT  s.customer_id,
        m.product_name,
        s.order_date,
        DENSE_RANK() OVER (PARTITION BY s.customer_id ORDER BY order_date
ASC) rnk
FROM sales s
INNER JOIN menu m
ON s.product_id=m.product_id
INNER JOIN members mb
ON s.customer_id=mb.customer_id
WHERE s.order_date>=join_date
)
SELECT customer_id,
       product_name
FROM rank_cte
WHERE rnk =1
```

customer_id	product_name
A	curry
B	sushi

```
--7.Which item was purchased just before the customer became a member?
WITH rank_cte AS (
SELECT  s.customer_id,
        m.product_name,
        s.order_date,
        DENSE_RANK() OVER (PARTITION BY s.customer_id ORDER BY order_date
ASC) rnk
FROM sales s
INNER JOIN menu m
ON s.product_id=m.product_id
INNER JOIN members mb
ON s.customer_id=mb.customer_id
WHERE s.order_date<join_date
)
SELECT customer_id,
       STRING_AGG(product_name,' , ') items_puchased_before_joining
FROM rank_cte
WHERE rnk =1
GROUP BY customer_id;
```

customer_id	items_puchased_before_joining
A	sushi , curry
B	curry

```
--8.What is the total items and amount spent for each member before they
became a member?
SELECT  s.customer_id,
        COUNT(1) total_items,
        SUM(m.price) amount_spent

FROM sales s
INNER JOIN menu m
ON s.product_id=m.product_id
INNER JOIN members mb
ON s.customer_id=mb.customer_id
WHERE s.order_date<join_date
GROUP BY s.customer_id;
```

customer_id	total_items	amount_spent
A	2	25
B	3	40

```
--9.If each $1 spent equates to 10 points and sushi has a 2x points
multiplier - how many points would each customer have?
SELECT  s.customer_id,
        SUM(CASE WHEN m.product_name ='sushi' THEN m.price*20
        ELSE m.price*10
        END ) points
FROM sales s
LEFT JOIN menu m
ON s.product_id=m.product_id
GROUP BY s.customer_id;
```

customer_id	points
A	860
B	940
C	360

```
--10.In the first week after a customer joins the program (including their
join date) they earn 2x points on all items,
--not just sushi - how many points do customer A and B have at the end of
January?
SELECT s.customer_id,
       SUM(CASE WHEN s.order_date BETWEEN mb.join_date AND
DATEADD(DAY,DATEDIFF(DAY,0,mb.join_date),6) THEN m.price*20
                 WHEN m.product_name='Sushi' THEN  m.price*20
            ELSE m.price*10
          END) points
FROM sales s
INNER JOIN members mb
ON s.customer_id=mb.customer_id
INNER JOIN menu m
ON s.product_id=m.product_id
WHERE MONTH(order_date)=1
GROUP BY s.customer_id;
```

customer_id	points
A	1370
B	820

24 January 2023 09:45

PROBLEM STATEMENT: Julia asked her students to create some coding challenges. Write a query to print the *hacker_id*, *name*, and the total number of challenges created by each student. Sort your results by the total number of challenges in descending order. If more than one student created the same number of challenges, then sort the result by *hacker_id*. If more than one student created the same number of challenges and the count is less than the maximum number of challenges created, then exclude those students from the result.

Input Format

The following tables contain challenge data:

- *Hackers:* The *hacker_id* is the id of the hacker, and *name* is the name of the hacker.

Column	Type
hacker_id	Integer
name	String

- *Challenges:* The *challenge_id* is the id of the challenge, and *hacker_id* is the id of the student who created the challenge.

Column	Type
challenge_id	Integer
hacker_id	Integer

Sample Input 0

Hackers Table:

hacker_id	name
5077	Rose
21283	Angela
62743	Frank
88255	Patrick
96196	Lisa

Challenges Table:

challenge_id	hacker_id
61654	5077
58302	21283
40587	88255
29477	5077
1220	21283
69514	21283
46561	62743
58077	62743
18483	88255
76766	21283
52382	5077
74467	21283
33625	96196
26053	88255
42665	62743
12859	62743
70094	21283
34599	88255
54680	88255
61881	5077

Sample Output 0

21283 Angela 6
88255 Patrick 5
96196 Lisa 1

Sample Input 1

Hackers Table:

hacker_id	name
12299	Rose
34856	Angela
79345	Frank
80491	Patrick
81041	Lisa

Challenges Table:

challenge_id	hacker_id
63963	81041
63117	79345
28225	34856
21989	12299
4653	12299
70070	79345
36905	34856
61136	80491
17234	12299
80308	79345
40510	34856
79820	80491
22720	12299
21394	12299
36261	34856
15334	12299
71435	79345
23157	34856
54102	34856
69065	80491

Sample Output 1

12299 Rose 6
34856 Angela 6
79345 Frank 4
80491 Patrick 3
81041 Lisa 1

Explanation

For *Sample Case 0*, we can get the following details:

hacker_id	name	challenges_created
21283	Angela	6
88255	Patrick	5
5077	Rose	4
62743	Frank	4
96196	Lisa	1

Students and both created challenges, but the maximum number of challenges created is so these students are excluded from the result.

For *Sample Case 1*, we can get the following details:

hacker_id	name	challenges_created
12299	Rose	6
34856	Angela	6
79345	Frank	4
80491	Patrick	3
81041	Lisa	1

Students and both created challenges. Because is the maximum number of challenges created, these students are included in the result.

MENTAL APPROACH:

1. First count the number of challenges created by each hacker.
2. Now look for hackers having the same number of challenges created by them.
3. Find the maximum number of challenges that were created by any hacker in the whole list.
4. Let us find the hackers who have created the same number of challenges.
5. If multiple hackers created the same number of challenges and it is not equal to the maximum number of challenges created then we will drop those records.

QUERY:

```sql
WITH count_rank_cte AS (
SELECT h.hacker_id
      ,h.name
      ,COUNT(challenge_id)  challenges_created
      ,DENSE_RANK() OVER (ORDER BY COUNT(challenge_id) DESC) rnk
FROM challenges c
LEFT JOIN hackers h
ON c.hacker_id = h.hacker_id
GROUP BY h.hacker_id, h.name
)
SELECT hacker_id
      ,name
      ,challenges_created
FROM count_rank_cte
WHERE rnk = 1
UNION
SELECT hacker_id
      ,name
      ,challenges_created
FROM count_rank_cte
WHERE rnk > 1
AND challenges_created IN (SELECT challenges_created
                           FROM count_rank_cte
                           GROUP BY challenges_created
                           HAVING COUNT(1) =1)
ORDER BY challenges_created DESC, hacker_id
```

QUERY EXPLANATION:

1. In CTE, we count the number of challenges created by each hacker and rank them on the basis of the number of challenges created by each hacker in descending order.
   Here, we gave descending order so that we can rank the higher number of challenges created at the top.

What it will give as output:

hacker_id	name	challenges_created	rnk
5120	Julia	50	1
69471	Michelle	50	1
77173	Mildred	50	1
52462	Nicholas	50	1
96009	Russell	50	1
72866	Eugene	42	2
37068	Patrick	41	3
12766	Jacqueline	40	4
86280	Beverly	36	5
19835	Joyce	36	5

2. Now we are querying for required columns.
3. In the first SELECT query we are querying only for the records whose rank is 1.

What it will give as output:

hacker_id	name	challenges_created
5120	Julia	50
69471	Michelle	50
77173	Mildred	50
52462	Nicholas	50
96009	Russell	50

4. Now join (using UNION) the first SELECT query to the second SELECT query where we are querying for the records whose rank is greater than 1 and using a subquery inside WHERE clause to filter out those records where the COUNT of the number of challenges created by different hacker is same
Basically, we are counting the number of challenges created and filtering where this count is equal to 1. It will give a count of the number of challenges created which is appearing only one time.

5. At last we are ordering them on the basis of descending order of number of challenges created by each hacker and if it is same then on the basis of ascending order of hacker id.

What it will give as output (without UNION) :

hacker_id	name	challenges_created
72866	Eugene	42
37068	Patrick	41
12766	Jacqueline	40
86280	Beverly	36
19835	Joyce	36

What it will give as output (with UNION): **FINAL OUTPUT**

hacker_id	name	challenges_created
5120	Julia	50
69471	Michelle	50
77173	Mildred	50
52462	Nicholas	50
96009	Russell	50
72866	Eugene	42
37068	Patrick	41
12766	Jacqueline	40

BY MANISH KUMAR CHAUDHARY

PROBELEM STATEMENT: Write a query that identifies cities with higher than average home prices when compared to the national average. Output the city names.

Table : **zillow_transactions**

FIELD	TYPE
id	int
state	varchar
city	varchar
street_address	varchar
mkt_price	int

MENTAL APPROACH:

1. First, we must determine the national average. This can be accomplished by calculating the average of all mk_prices.
2. Now compute the average of mkt_prices for each city.
   It will assist us in determining the average home prices in each city.
3. Now compare them and list the records where the average home price in the city exceeds the national average home price.

QUERY:

```
SELECT city
FROM zillow_transactions
GROUP BY city
HAVING AVG(mkt_price) > (SELECT AVG(mkt_price) FROM zillow_transactions)
```

QUERY EXPLANATION:

1. We only need the city name as output, thus city is in the SELECT statement.
2. We have now grouped them by city so that we can receive records for each particular city.
3. Using the having clause, we can only have records where each city's average home price is more than the national average home price.
   Basically, in execution, AVG(mkt price) will be generated for each city and then compared to AVG(mkt price) calculated via a subquery.
   The subquery will return the national or overall average home price in this case.

WHAT HAPPENS IN BACKGROUND:

city	avg_home_price_citywise	avg_national_home_price
Chicago	294495	550122
Houston	377289	550122
Los Angeles	389598	550122
Mountain View	995936	550122
New York City	279617	550122
Philadelphia	441121	550122
Phoenix	352474	550122
San Diego	310169	550122
San Francisco	1077403	550122
Santa Clara	983119	550122

INSIGHTS:

1. We can see from the table above that just three cities have average home prices that are higher than the national average home price.

FINAL OUTPUT:

city
Mountain View
San Francisco
Santa Clara

Visit my LinkedIn page by clicking here for more answers to interview questions.

19 January 2023 18:39

PROBLEM STATEMENT: Classify each business as either a restaurant, cafe, school, or other.

- A restaurant should have the word 'restaurant' in the business name.
- A cafe should have either 'cafe', 'café', or 'coffee' in the business name.
- A school should have the word 'school' in the business name.
- All other businesses should be classified as 'other'.

Output the business name and their classification.

FIELD	TYPE
business_id	int
business_name	varchar
business_address	varchar
business_city	varchar
business_state	varchar
business_postal_code	float
business_latitude	float
business_longitude	float
business_location	varchar
business_phone_number	float
inspection_id	varchar
inspection_date	datetime
inspection_score	float
inspection_type	varchar
violation_id	varchar
violation_description	varchar
risk_category	varchar

MENTAL APPROACH:

1. We will simply find the particular words like 'restaurant' , 'cafe' , etc in business name and write its business type according to this information.

QUERY:

```sql
SELECT business_name
      ,CASE WHEN LOWER(business_name) LIKE '%restaurant%'
                THEN 'restaurant'
           WHEN LOWER(business_name) LIKE '%cafe%'
                OR LOWER(business_name) LIKE '%café%'
                OR LOWER(business_name) LIKE '%coffee%'
                THEN 'cafe'
           WHEN LOWER(business_name) LIKE '%school%'
                THEN 'school'
        ELSE 'other'
        END AS business_type
FROM sf_restaurant_health_violations
GROUP BY business_name;
```

QUERY EXPLANATION:

1. This is a classification problem so we are use CASE WHN for classifying the business type for each business.
   Here, in CASE statement we are using LIKE operator to search for particular word so that we can find the business type of that particular business.

2. As there were multiple business with same name, we want only distinct ones.
   Here, for getting distinct names we are simply using GROUP BY business_name instead of DISTINCT keyword in SELECT clause because DISTINCT take more execution time as compare to GROUP BY.

SAMPLE OUTPUT:

business_name	business_type
24 Hour Fitness Club, #273	other
24th and Folsom Eatery	other
Akira Japanese Restaurant	restaurant
Allstars Cafe Inc	cafe
Andersen Bakery	other
Angel Cafe and Deli	cafe
Annie's Hot Dogs & Pretzels	other
Antonelli Brothers Meat, Fish, and Poultry Inc.	other
AT&T - COMMISARY KITCHEN [145184]	other
AT&T Park - Coffee and Ice Cream (5A+5B)	cafe
Azalina's	other
BALBOA HIGH SCHOOL	school

18 January 2023 18:35

PROBLEM STATEMENT: You did such a great job helping Julia with her last coding contest challenge that she wants you to work on this one, too!

The total score of a hacker is the sum of their maximum scores for all of the challenges. Write a query to print the *hacker_id*, *name*, and total score of the hackers ordered by the descending score. If more than one hacker achieved the same total score, then sort the result by ascending *hacker_id*. Exclude all hackers with a total score of from your result.

Input Format

The following tables contain contest data:

- *Hackers:* The *hacker_id* is the id of the hacker, and *name* is the name of the hacker.

Column	Type
hacker_id	Integer
name	String

- *Submissions:* The *submission_id* is the id of the submission, *hacker_id* is the id of the hacker who made the submission, *challenge_id* is the id of the challenge for which the submission belongs to, and *score* is the score of the submission.

Column	Type
submission_id	Integer
hacker_id	Integer
challenge_id	Integer
score	Integer

Sample Input

Hackers Table:

hacker_id	name
4071	Rose
4806	Angela
26071	Frank
49438	Patrick
74842	Lisa
80305	Kimberly
84072	Bonnie
87868	Michael
92118	Todd
95895	Joe

Submissions Table:

submission_id	hacker_id	challenge_id	score
67194	74842	63132	76
64479	74842	19797	98
40742	26071	49593	20
17513	4806	49593	32
69846	80305	19797	19
41002	26071	89343	36
52826	49438	49593	9
31093	26071	19797	2
81614	84072	49593	100
44829	26071	89343	17
75147	80305	49593	48
14115	4806	49593	76
6943	4071	19797	95
12855	4806	25917	13
73343	80305	49593	42
84264	84072	63132	0
9951	4071	49593	43
45104	49438	25917	34
53795	74842	19797	5
26363	26071	19797	29
10063	4071	49593	96

Sample Output
4071 Rose 191
74842 Lisa 174
84072 Bonnie 100
4806 Angela 89
26071 Frank 85
80305 Kimberly 67
49438 Patrick 43

Explanation

Hacker *4071* submitted solutions for challenges *19797* and *49593*, so the total score .

Hacker *74842* submitted solutions for challenges *19797* and *63132*, so the total score

Hacker *84072* submitted solutions for challenges *49593* and *63132*, so the total score .

The total scores for hackers *4806*, *26071*, *80305*, and *49438* can be similarly calculated.

MENTAL APPROACH:

1. Firstly combine both table on the basis of common column.
2. Now we will find hacker id and name with their total score.
   NOTE: Here, we are told to find total score. So, to find it we will add all the scores for each hacker for different challenge. For hacker who have submitted same challenge multiple time we will find the maximum marks obtained in that particular challenge.
3. After this we will order them on the basis of decreasing order of total scores.
4. If they are having same total scores then we will order them on the basis of ascending order of hacker_id.

QUERY:

```
WITH max_score_cte AS (
SELECT h.hacker_id
      ,h.name
      ,MAX(s.score)  maximum_score
FROM submissions s
LEFT JOIN hackers h
ON s.hacker_id = h.hacker_id
WHERE s.score ! = 0
GROUP BY h.hacker_id, h.name, s.challenge_id
)
SELECT hacker_id
      ,name
      ,SUM(maximum_score) total_score
FROM max_score_cte
GROUP BY hacker_id, name
ORDER BY total_score DESC, hacker_id ASC
```

QUERY EXPLANATION:

1. We have used here CTE so that we can get the hacker id, hacker name and maximum scores scored by each hacker for the distinct challenge.
   Here, we have used GROUP BY so that we can have distinct combination of hacker id, hacker name and challenge id.

   SAMPLE OUTPUT OF CTE:

hacker_id	name	maximum_score
486	Rose	45
486	Rose	29
597	Angela	107
775	Frank	31
964	Patrick	55
1700	Lisa	66
1700	Lisa	49
1746	Kimberly	32

   NOTE: Here, hackers who have submitted same challenge multiple time has got aggregated and maximum score or best score has been provided against that particular challenge for that particular hacker.

2. Now we will query hacker id, hacker name and SUM of maximum scores from the CTE so that for a particular hacker we can find the overall total score scored by them for all the challenges.

   FINAL SAMPLE OUTPUT:

hacker_id	name	maximum_score
76971	Ashley	760
84200	Susan	710
76615	Ryan	700
82382	Sara	640
79034	Marilyn	580
78552	Harry	570
74064	Helen	540
78688	Sean	540

Case Study #1 - Danny's Diner

Danny Ma · May 1, 2021

Introduction

Danny seriously loves Japanese food so in the beginning of 2021, he decides to embark upon a risky venture and opens up a cute little restaurant that sells his 3 favourite foods: sushi, curry and ramen.

Danny’s Diner is in need of your assistance to help the restaurant stay afloat - the restaurant has captured some very basic data from their few months of operation but have no idea how to use their data to help them run the business.

Problem Statement

Danny wants to use the data to answer a few simple questions about his customers, especially about their visiting patterns, how much money they’ve spent and also which menu items are their favourite. Having this deeper connection with his customers will help him deliver a better and more personalised experience for his loyal customers.

He plans on using these insights to help him decide whether he should expand the existing customer loyalty program - additionally he needs help to generate some basic datasets so his team can easily inspect the data without needing to use SQL.

Danny has provided you with a sample of his overall customer data due to privacy issues - but he hopes that these examples are enough for you to write fully functioning SQL queries to help him answer his questions!

Danny has shared with you 3 key datasets for this case study:

- sales
- menu
- members

You can inspect the entity relationship diagram and example data below.

Entity Relationship Diagram

Example Datasets

All datasets exist within the dannys_diner database schema - be sure to include this reference within your SQL scripts as you start exploring the data and answering the case study questions.

Table 1: sales

The sales table captures all customer_id level purchases with an corresponding order_date and product_id information for when and what menu items were ordered.

Table 2: menu

The `menu` table maps the `product_id` to the actual `product_name` and `price` of each menu item.

Table 3: members

The final `members` table captures the `join_date` when a `customer_id` joined the beta version of the Danny's Diner loyalty program.

Interactive SQL Session

You can use the embedded DB Fiddle below to easily access these example datasets - this interactive session has everything you need to start solving these questions using SQL.

You can click on the `Edit on DB Fiddle` link on the top right hand corner of the embedded session below and it will take you to a fully functional SQL editor where you can write your own queries to analyse the data.

You can feel free to choose any SQL dialect you'd like to use, the existing Fiddle is using PostgreSQL 13 as default.

Serious SQL students have access to a dedicated SQL script in the 8 Week SQL Challenge section of the course which they can use to generate relevant temporary tables like we've done throughout the entire course!

Case Study Questions

Each of the following case study questions can be answered using a single SQL statement:

1. What is the total amount each customer spent at the restaurant?
2. How many days has each customer visited the restaurant?
3. What was the first item from the menu purchased by each customer?
4. What is the most purchased item on the menu and how many times was it purchased by all customers?
5. Which item was the most popular for each customer?
6. Which item was purchased first by the customer after they became a member?
7. Which item was purchased just before the customer became a member?
8. What is the total items and amount spent for each member before they became a member?
9. If each $1 spent equates to 10 points and sushi has a 2x points multiplier - how many

points would each customer have?

10. In the first week after a customer joins the program (including their join date) they earn 2x points on all items, not just sushi - how many points do customer A and B have at the end of January?

C	2021-01-07	ramen	12	N	null

21 January 2023 20:00

PROBLEM STATEMENT: Find the top 10 users that have traveled the greatest distance. Output their id, name and a total distance traveled.

lyft_rides_log

FIELD	TYPE
id	int
user_id	int
distance	int

lyft_users

FIELD	TYPE
id	int
name	varchar

MENTAL APPROACH:

1. For each user look for the distance traveled by them and sort them in descending order according to distance.
2. Now pick the top 10 users who have traveled the maximum distance.

QUERY:

```sql
WITH row_num_cte AS (
SELECT r.id
      ,u.name
      ,SUM(r.distance) total_distance
      ,ROW_NUMBER() OVER (ORDER BY SUM(r.distance)
DESC) row_no
FROM lyft_rides_log r
LEFT JOIN lyft_users u
ON r.user_id = u.id
GROUP BY r.id, u.name
)
SELECT id
      ,name
      ,total_distance
FROM row_num_cte
WHERE row_no ⩽ 10
```

QUERY EXPLANATION:

1. We using CTE here for getting the required data along with rank to it.
2. Here, we used ROW_NUMBER() as we want to rank them in descending order of distance traveled by each user.
   We have not used RANK() here because of the same distance would give the same rank but we just want top 10 highest distance thus we are not concerned about whether it is repeating or not. (But if the requirement is that then we can use RANK())
3. Now after giving them rank or row number now we will extract the required data from the CTE we created where we will use the condition that our rank should be less than or equal to 10. This is because we want top 10 records.

OUTPUT:

id	name	distance
144	Barbara Larson	98
197	Christina Price	98
173	Crystal Berg	96
133	Christopher Schmitt	96
185	Kimberly Potter	95
115	Pamela Cox	94
147	Barbara Larson	94
101	Patrick Gutierrez	93
154	Dennis Douglas	93
158	Sean Parker	92

25 January 2023 14:18

PROBLEM STATEMENT: Find the number of matches played by each team along with the number of matches won and matches lost by each team.

icc_world_cup

Team_1	Team_2	Winner
India	SL	India
SL	Aus	Aus
SA	Eng	Eng
Eng	NZ	NZ
Aus	India	India

In table icc_world_cup Team_1 and Team_2 are teams who are playing against each other and the winner column lists those who won that match.

MENTAL APPROACH:

1. To get all teams who played in icc_world_cup we need to get team names from both team_1 and team_2.
2. Now we need to choose only distinct team as there can be duplicate teams.
   There duplicate because the same team is playing against different teams.
3. For total matches played by each team we can simply go through both teams and see which team played how many times.
4. To count the number of matches wins we can count from the winner column.
5. Now for matches lost we can subtract matches won from the total matches played.

QUERY:

```
WITH cte AS (
SELECT team_1 team
      ,winner
FROM icc_world_cup
UNION ALL
SELECT team_2 team
      ,winner
FROM icc_world_cup
)
SELECT team
      ,COUNT(1) matches_played
      ,SUM(CASE WHEN team=winner THEN 1 ELSE 0 END) matches_wins
      ,SUM(CASE WHEN team<> winner THEN 1 ELSE 0 END) matches_losses
```

```
      ,COUNT(1) matches_played
      ,SUM(CASE WHEN team=winner THEN 1 ELSE 0 END) matches_wins
      ,SUM(CASE WHEN team<> winner THEN 1 ELSE 0 END) matches_losses
FROM cte
GROUP BY team
```

QUERY EXPLANATION:

1. In CTE we are using union to join team_1 and team_2 in one column.
   Basically, this will help us to find total matches played and also it will help to get matches won and lost by each team.

   Output we will get with this step:

team	winner
India	India
SL	Aus
SA	Eng
Eng	NZ
Aus	India
SL	India
Aus	Aus
Eng	Eng
NZ	NZ
India	India

2. Now in SELECT clause after CTE we are querying for team, COUNT() and SUM with CASE WHEN statement.
   Here, COUNT(team) will helps to find the total matches played by each team.
   Here, CASE WHEN will help us to flag 1 if team is same as team in winner column else flag 0 and then we are summing this as it will give matches won by each team.
   Here, CASE WHEN will help us to flag 1 if team is not same as team in winner column else flag 0 and then we are summing this as it will give matches lost by each team.

ANKIT BANSAL SIR'S APPROACH
QUERY:

```
WITH flag_cte AS (
SELECT team_1 team
      ,CASE WHEN team_1 = winner THEN 1
            ELSE 0
        END win_flag
FROM icc_world_cup
UNION ALL
SELECT team_2 team
      ,CASE WHEN team_2 = winner THEN 1
```

```
UNION ALL
SELECT team_2 team
      ,CASE WHEN team_2 = winner THEN 1
            ELSE 0
        END win_flag
FROM icc_world_cup
)
SELECT team
      ,COUNT(team) matches_played
      ,SUM(win_flag) matches_wins
      ,COUNT(team)-SUM(win_flag) matches_losses
FROM flag_cte
GROUP BY team
```

QUERY EXPLANATION:

1. In CTE we are flagging 1 for win and 0 for loss along with joining team_1 and team_2 using UNION ALL in one single column.

   Basically , will give us two column team and win_flag.

2. After CTE we are SELECTING the required output from the above CTE i.e COUNT for total matches played and SUM of win_flag column to get total matches won by a team and last difference between both to get matches lost by each team.

FINAL OUTPUT FROM BOTH QUERY

team	matches_played	matches_wins	matches_losses
Aus	2	1	1
Eng	2	1	1
India	2	2	0
NZ	1	1	0
SA	1	0	1
SL	2	0	2

SCRIPT FOR CREATING TABLE:

```
create table icc_world_cup
(
Team_1 Varchar(20),
Team_2 Varchar(20),
Winner Varchar(20)
);
INSERT INTO icc_world_cup values('India','SL','India');
INSERT INTO icc_world_cup values('SL','Aus','Aus');
```

```
INSERT INTO icc_world_cup values('SA','Eng','Eng');
INSERT INTO icc_world_cup values('Eng','NZ','NZ');
INSERT INTO icc_world_cup values('Aus','India','India');

select * from icc_world_cup;
```

For more such questions visit Ankit Sir's Youtube channel by clicking here.
For more such questions explained in written visit my LnkedIn profile by clicking here.

09 January 2023 11:14

Problem Statement: Write a query that'll identify returning active users. A returning active user is a user that has made a second purchase within 7 days of any other of their purchases. Output a list of user_ids of these returning active users.

amazon_transactions

Field	Type
id	int
user_id	int
item	varchar
created_at	datetime
revenue	int

Sample Table :

id	user_id	item	created_at	revenue
1	109	milk	03-03-2020	123
20	100	banana	18-03-2020	599
26	109	bread	22-03-2020	432
29	100	milk	29-03-2020	410
40	109	bread	02-03-2020	362
67	113	biscuit	21-03-2020	278
74	100	banana	13-03-2020	175
95	100	bread	07-03-2020	410

Mental Approach:

1. For finding users who bought the next product within 7 days of buying a product we need to find the difference between these dates.
2. We need to find the difference of two consecutive buying dates (dates must be sorted in ascending order) for the same user_id. (In SQL we will do this by using the LEAD window function within the DATEDIFF Function to find no day difference)
3. After getting the difference we will now need to filter for the difference that is less than 7 days. (In SQL we will do it by using a subquery to find the difference and filtering out in outer query)
4. There are more than 2 products bought consecutively so to get unique users we need to choose unique records of user_id. (In SQL we do this using DISTINCT)

Query:

```sql
SELECT DISTINCT user_id returning_active_users
FROM (
      SELECT user_id
            ,DATEDIFF(DAY,created_at,LEAD(created_at)
             OVER (PARTITION BY user_id ORDER BY created_at)) diff
      FROM amazon_transactions
     ) difference
WHERE diff <7;
```

Subquery Sample output:

user_id	diff
100	6
100	5
100	11
100	
109	1
109	19
109	
113	

Final Sample Output:

returning_active_users
100
109

Actual Output:

returning_active_users
100
103
105
109
110
111
112
114
117
120
122
128
129
130
131
133
141
143
150

05 January 2023 09:45

Problem Statement: Write a query to find which gender gives a higher average review score when writing reviews as guests. Use the from_type column to identify guest reviews. Output the gender and their average review score.

airbnb_reviews

FIELD	TYPE
from_user	int
to_user	int
from_type	varchar
to_type	varchar
review_score	int

airbnb_guests

FIELD	TYPE
guest_id	int
nationality	varchar
gender	varchar
age	int

Mental Approach

1. Joining the tables.
2. Finding the review_score by filtering through from_user for guest users only .(in SQL we do it using WHERE clause)
3. Now for both gender we need to find the Average of review_score. (In SQL we do it using GROUP BY clause)

```sql
SELECT g.gender
      ,AVG(r.review_score) avg_review_score
FROM airbnb_reviews r
LEFT JOIN airbnb_guests g
ON r.from_user=g.guest_id
WHERE r.from_type = 'guest'
GROUP BY g.gender;
```

Output

gender	avg_review_score
M	5.526
F	5.009

08 January 2023 07:49

SUBSTRING(), RIGHT() , LEFT() Functions

SUBSTRING(): This function returns part of a character, binary, text, or image expression in SQL Server.

SYNTAX:

```
SUBSTRING(expression,start,length)
```

Expression: It explicit input or column name from a table.
Start: It is the position from which position you want to extract.
Length: It is for how many characters you want to extract from the expression.

RIGHT(): This function returns specified number of characters from the right side of given expression.

SYNTAX:

```
RIGHT(expression,number of characters)
```

LEFT(): This function returns specified number of characters from the left side of given expression.

SYNTAX:

```
LEFT(expression,number of characters)
```

Problem Statement: Query the *Name* of any student in **STUDENTS** who scored higher than *Marks*. Order your output by the *last three characters* of each name. If two or more students both have names ending in the same last three characters (i.e.: Bobby, Robby, etc.), secondary sort them by ascending *ID*.

Input Format

The **STUDENTS** table is described as follows:

Column	Type
ID	Integer
Name	String
Marks	Integer

The *Name* column only contains uppercase (A-Z) and lowercase (a-z) letters.

Sample Input

ID	Name	Marks
1	Ashley	81
2	Samantha	75
4	Julia	76
3	Belvet	84

Sample Output

Ashley
Julia
Belvet

Explanation

Only Ashley, Julia, and Belvet have *Marks* >75 . If you look at the last three characters of each of their names, there are no duplicates and 'ley' < 'lia' < 'vet'.

Query:

```sql
SELECT NAME
FROM STUDENTS
WHERE MARKS>75
ORDER BY RIGHT(NAME,3),ID;
```

17 January 2023 18:40

PROBLEM STATEMENT: Given the users purchase history write a query to print users who have done purchase on more than 1 day and products purchased on a given day are never repeated on any other day.

Bonus points if you solve it without using a subquery or self-join.

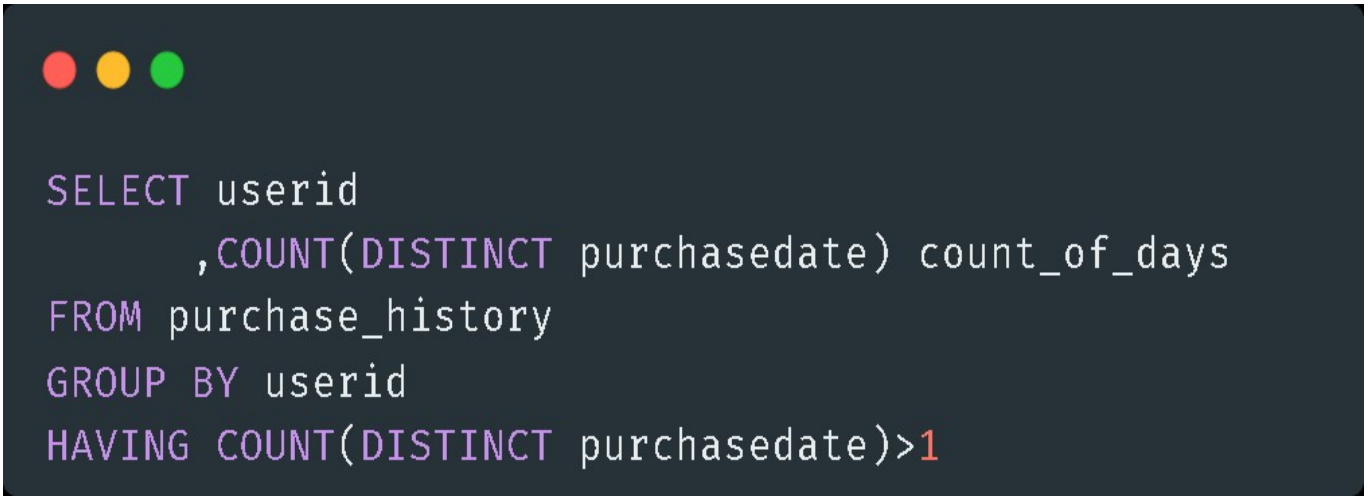
userid	productid	purchasedate
1	1	23-01-2012
1	2	23-01-2012
1	3	25-01-2012
2	1	23-01-2012
2	2	23-01-2012
2	2	25-01-2012
2	4	25-01-2012
3	4	23-01-2012
3	1	23-01-2012
4	1	23-01-2012
4	2	25-01-2012

MENTAL APPROACH:

1. Find the users who have purchased on different dates.
2. After this now we need to find the users who have purchased a product that was not bought on any other day by that particular user.

STEPWISE SOLUTION

1. To get the users who have bought on different days we are using the HAVING clause so that we can filter out those users who have bought only on one particular day.



```
SELECT userid
      ,COUNT(DISTINCT purchasedate) count_of_days
FROM purchase_history
GROUP BY userid
HAVING COUNT(DISTINCT purchasedate)>1
```

Output:

userid	count_of_days
1	2
2	2
4	2

2. Now for getting non-repeated products purchased by the user we are using AND clause with the HAVING clause for extra filtering.
   Here, we are comparing the COUNT of distinct productid (it will give the count of distinct products bought by a particular user) and COUNT of productid (it will give a count of all products bought by a user). If both matches that means all products are distinct else the product is repeated on any day by that particular user and thus will be filtered out.

```
SELECT userid
      ,COUNT(DISTINCT purchasedate) count_of_days
      ,COUNT(DISTINCT productid) no_distinct_products
      ,COUNT(productid) no_products
FROM purchase_history
GROUP BY userid
HAVING COUNT(DISTINCT purchasedate)>1
AND COUNT(DISTINCT productid) = COUNT(productid)
```

Output:

userid	count_of_days	no_distinct_products	no_products
1	2	3	3
4	2	2	2

FINAL QUERY AS PER REQUIREMENT:

```
SELECT userid
FROM purchase_history
GROUP BY userid
HAVING COUNT(DISTINCT purchasedate)>1
AND COUNT(DISTINCT productid) = COUNT(productid)
```

OUTPUT:

userid
1
4

PROBLEM STATEMENT: Given a table of purchases by date, calculate the month-over-month percentage change in revenue. The output should include the year-month date (YYYY-MM) and percentage change, rounded to the 2nd decimal point, and sorted from the beginning of the year to the end of the year.
The percentage change column will be populated from the 2nd month forward and can be calculated as ((this month's revenue - last month's revenue) / last month's revenue)*100.

Table : **sf_transactions**

FIELD	TYPE
id	int
created_at	datetime
value	int
purchased_id	int

MENTAL APPROACH:

1. Change the created_at to the required format and group them. We will be given year and month specific dates.

2. Using the above formula, determine the month-over-month percentage change in revenue.

QUERY:

```
WITH cte AS (
select FORMAT(CAST(created_at AS DATE),'yyyy-MM') year_month
      ,LAG(SUM(value)) OVER(ORDER BY FORMAT(CAST(created_at AS DATE),'yyyy-MM'))
       previous_month_revenue
      ,SUM(value) revenue
FROM sf_transactions
GROUP BY FORMAT(CAST(created_at AS DATE),'yyyy-MM')
)
SELECT year_month
      ,CAST(SUM((revenue-previous_month_revenue)*100.0/previous_month_revenue) AS
          DECIMAL(15,2)) revenue_diff_pct
FROM cte
GROUP BY year_month
```

QUERY EXPLANATION:

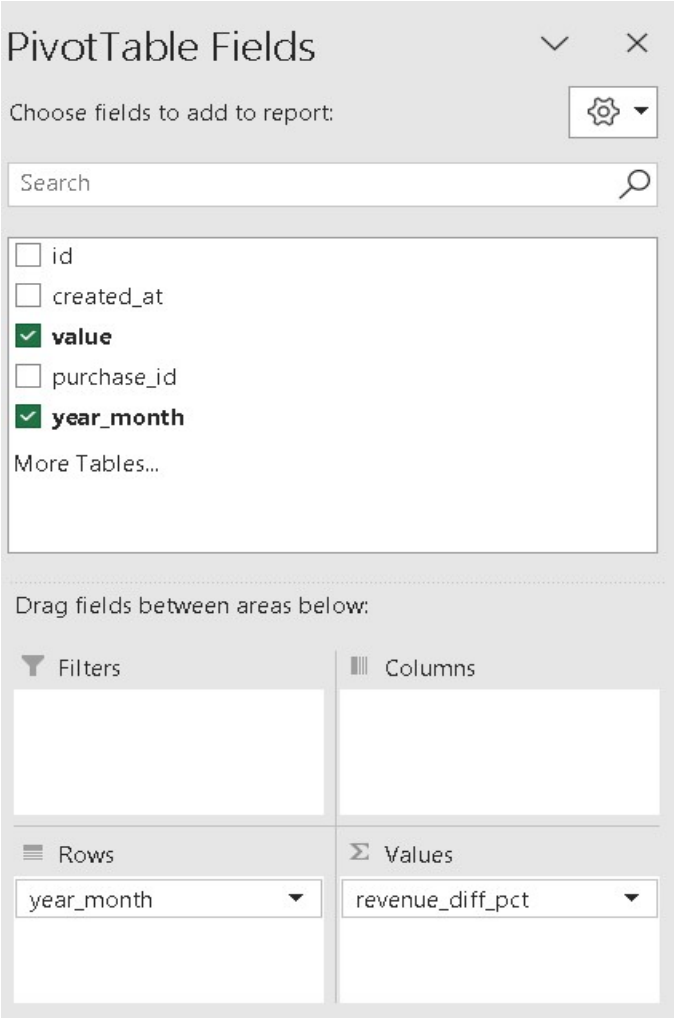
1. With CTE, we change the format of the create at date column to YYYY-MM and retrieve the prior month's salary using the LAG and SUM functions.
   We're using the SUM function here to get the total revenue for that month.

2. With the SELECT statement following the CTE, we are simply utilizing a formula to determine the month-over-month % change in revenue for all months and rounding it to two decimal points.
   In this case, we're using SUM in our calculation since we want to group them by year_month.

SOLVING IT IN EXCEL

STEPS:

1. To get the YYYY-MM date format from the created_at column, first add a column called year month.
2. Now, create a pivot table using the entire table.
3. Place the newly created year_month column in the rows field.
4. Put the revenue in values field. (It will automatically add them up based on year_month.)
5. However, under the value field settings, change the name to revenue_diff_pct in the custom name field, and then go to the show value as tab.
6. Select % Difference From in the show value as tab, with the base field as year month and the base item as (previous).
   This will tell us the revenue percentage change month over month.



FINAL OUTPUT:

year_month	revenue_diff_pct
2019-01	
2019-02	-28.56%
2019-03	23.35%
2019-04	-13.84%
2019-05	13.49%
2019-06	-2.78%
2019-07	-6.00%
2019-08	28.36%
2019-09	-4.97%
2019-10	-12.68%
2019-11	1.71%
2019-12	-2.11%

PROBLEM STATEMENT: Find how many orders are made by new customers and repeat customers get output as order_date, new_customer_count, repeat_customer_count

Customer_orders Table:

order_id	customer_id	order_date	order_amount
1	100	01-01-2022	2000
2	200	01-01-2022	2500
3	300	01-01-2022	2100
4	100	02-01-2022	2000
5	400	02-01-2022	2200
6	500	02-01-2022	2700
7	100	03-01-2022	3000
8	400	03-01-2022	1000
9	600	03-01-2022	3000

MENTAL APPROACH:

1. At first we need to find the first order date for all the customers for each date. It will help us to find the number of new customers.
2. Now as we know the first order date for each customer for each date. We can easily find the number of repeat customers for each date with the help of this.
   NOTE: We will compare the order date of each customer to its first order date so that we can confirm whether it is a repeat order or the customer has ordered for the first time.

QUERY:

```
WITH first_order_date_cte AS (
SELECT order_date
      ,customer_id
      ,MIN(order_date) OVER (PARTITION BY customer_id) first_order_date
FROM customer_orders
)
SELECT order_date
      ,SUM(CASE WHEN order_date=first_order_date THEN 1 ELSE 0 END) new_customers
      ,SUM(CASE WHEN order_date◇first_order_date THEN 1 ELSE 0 END) repeat_customers
FROM first_order_date_cte
GROUP BY order_date
```

QUERY EXPLANATION:

1. With the help of CTE we are finding the minimum order date for each customer. This will help us to find the first order date for each customers.

order_date	customer_id	first_order_date
01-01-2022	100	01-01-2022
02-01-2022	100	01-01-2022
03-01-2022	100	01-01-2022
01-01-2022	200	01-01-2022
01-01-2022	300	01-01-2022
02-01-2022	400	02-01-2022
03-01-2022	400	02-01-2022
02-01-2022	500	02-01-2022
03-01-2022	600	03-01-2022

2. Now with SELECT statement after CTE we are using SUM along with CASE WHEN statement. Here CASE WHEN alone will give following result:

order_date	new_customers	repeat_customers
01-01-2022	1	0
02-01-2022	0	1
03-01-2022	0	1
01-01-2022	1	0
01-01-2022	1	0
02-01-2022	1	0
03-01-2022	0	1
02-01-2022	1	0
03-01-2022	1	0

When order date and first order date are same that means that customer is new customer. So, we will flag it with 1 else with 0 and add all of them using SUM. Thus it will give count of new customers.

Similarly when order date and first order date are not same that means that customer is repeat customer. So, we will flag 1 when this is met else with 0 and add all of them. It will give count of repeat customers.

FINAL OUTPUT:

order_date	new_customers	repeat_customers
01-01-2022	3	0
02-01-2022	2	1
03-01-2022	1	2

ANKIT BANSAL Sir's APPROACH:

QUERY:

```
WITH first_order_date_cte AS (
SELECT customer_id
      ,MIN(order_date) first_order_date
FROM customer_orders
GROUP BY customer_id
)
SELECT c.order_date
      ,SUM(CASE WHEN c.order_date=f.first_order_date THEN 1 ELSE 0 END) new_customers
      ,SUM(CASE WHEN c.order_date<>f.first_order_date THEN 1 ELSE 0 END) repeat_customers
FROM customer_orders c
INNER JOIN first_order_date_cte f
ON c.customer_id=f.customer_id
GROUP BY c.order_date
```

QUERY EXPLANATION:

1. In CTE first order date for each customer is extracted.

customer_id	first_order_date
100	01-01-2022
200	01-01-2022
300	01-01-2022
400	02-01-2022
500	02-01-2022
600	03-01-2022

2. With SELECT statement after CTE we are flagging 1 and 0 with CASE WHEN.
   Here, we have used join because in CTE we don't have an order date. So to get count of new and repeat customers for each order date we need the order date.

order_date	new_customers	repeat_customers
01-01-2022	1	0
02-01-2022	0	1
03-01-2022	0	1
01-01-2022	1	0
01-01-2022	1	0
02-01-2022	1	0
03-01-2022	0	1
02-01-2022	1	0
03-01-2022	1	0

3. Now to count new and repeat customers for each date we will SUM the flagged numbers.

FINAL OUTPUT:

order_date	new_customers	repeat_customers
01-01-2022	3	0
02-01-2022	2	1
03-01-2022	1	2

For more such interview questions visit Ankit Bansal Sir's Channel by clicking here.
For more such written explanations of interview questions visit my LinkedIn by clicking here.

25 January 2023 11:59

PROBLEM STATEMENT: Amber's conglomerate corporation just acquired some new companies. Each of the companies follows this hierarchy:

Founder
⇓
Lead Manager
⇓
Senior Manager
⇓
Manager
⇓
Employee

Given the table schemas below, write a query to print the *company_code*, *founder* name, total number of *lead* managers, total number of *senior* managers, total number of *managers*, and total number of *employees*. Order your output by ascending *company_code*.

Note:

- The tables may contain duplicate records.
- The *company_code* is string, so the sorting should not be **numeric**. For example, if the *company_codes* are *C_1*, *C_2*, and *C_10*, then the ascending *company_codes* will be *C_1*, *C_10*, and *C_2*.

Input Format

The following tables contain company data:

- *Company:* The *company_code* is the code of the company and *founder* is the founder of the company.

Column	Type
company_code	String
founder	String

- *Lead_Manager:* The *lead_manager_code* is the code of the lead manager, and the *company_code* is the code of the working company.

Column	Type
lead_manager_code	String
company_code	String

- *Senior_Manager:* The *senior_manager_code* is the code of the senior manager, the *lead_manager_code* is the code of its lead manager, and the *company_code* is the code of the working company.

Column	Type
senior_manager_code	String
lead_manager_code	String
company_code	String

- *Manager:* The *manager_code* is the code of the manager, the *senior_manager_code* is the code of its senior manager, the *lead_manager_code* is the code of its lead manager, and the *company_code* is the code of the working company.

Column	Type
manager_code	String
senior_manager_code	String
lead_manager_code	String
company_code	String

- *Employee:* The *employee_code* is the code of the employee, the *manager_code* is the code of its manager, the *senior_manager_code* is the code of its senior manager, the *lead_manager_code* is the code of its lead manager, and the *company_code* is the code of the working company.

Column	Type
employee_code	String
manager_code	String
senior_manager_code	String
lead_manager_code	String
company_code	String

Sample Input

Company Table:

company_code	founder
C1	Monika

company_code	founder
C1	Monika
C2	Samantha

Lead_Manager Table:

lead_manager_code	company_code
LM1	C1
LM2	C2

Senior_Manager Table:

senior_manager_code	lead_manager_code	company_code
SM1	LM1	C1
SM2	LM1	C1
SM3	LM2	C2

Manager Table:

manager_code	senior_manager_code	lead_manager_code	company_code
M1	SM1	LM1	C1
M2	SM3	LM2	C2
M3	SM3	LM2	C2

Employee Table:

employee_code	manager_code	senior_manager_code	lead_manager_code	company_code
E1	M1	SM1	LM1	C1
E2	M1	SM1	LM1	C1
E3	M2	SM3	LM2	C2
E4	M3	SM3	LM2	C2

Sample Output

C1 Monika 1 2 1 2
C2 Samantha 1 1 2 2

Explanation

In company *C1*, the only lead manager is *LM1*. There are two senior managers, *SM1* and *SM2*, under *LM1*. There is one manager, *M1*, under senior manager *SM1*. There are two employees, *E1* and *E2*, under manager *M1*.

In company *C2*, the only lead manager is *LM2*. There is one senior manager, *SM3*, under *LM2*. There are two managers, *M2* and *M3*, under senior manager *SM3*. There is one employee, *E3*, under manager *M2*, and another employee, *E4*, under manager, *M3*.

MENTAL APPROACH:

1. As we want to find total number of lead managers, total number of senior managers, total number of managers, and total number of employees for each company so we will find count of all the designations from their respective table.

QUERY:

```
SELECT c.company_code
      ,c.founder
      ,COUNT(DISTINCT l.lead_manager_code) no_lead_manager
      ,COUNT(DISTINCT s.senior_manager_code) no_senior_manager
      ,COUNT(DISTINCT m.manager_code) no_manager
      ,COUNT(DISTINCT e.employee_code) no_employee
FROM company c
INNER JOIN lead_manager l
ON c.company_code=l.company_code
INNER JOIN senior_manager s
ON l.lead_manager_code=s.lead_manager_code
INNER JOIN manager m
ON s.senior_manager_code=m.senior_manager_code
INNER JOIN employee e
ON m.manager_code=e.manager_code
GROUP BY c.company_code, c.founder
ORDER BY c.company_code
```

QUERY EXPLANATION:

1. We are joining all the tables and counting the required fields.

   Here, we have used distinct in count because same manager can be manager for multiple employees and similarly same senior manager can be senior manager for multiple managers and likewise lead manager also.

   NOTE: We could have used the employee table directly over here for getting the required fields but in a real scenario, it would not be the appropriate way. It is because there may be new employees who have not been assigned to any manager. Thus we are joining all tables and then finding the required fields.
2. We are grouping by company code because we want the required field for each company
3. At last we are ordering on the basis of ascending order of company code.

SAMPLE OUTPUT:

company_code	founder	no_lead_manager	no_senior_manager	no_manager	no_employee
C1	Angela	1	2	5	13
C10	Earl	1	1	2	3
C100	Aaron	1	2	4	10
C11	Robert	1	1	1	1
C12	Amy	1	2	6	14
C13	Pamela	1	2	5	14
C14	Maria	1	1	3	5
C15	Joe	1	1	2	3
C16	Linda	1	1	3	5
C17	Melissa	1	2	3	7

12 January 2023 17:09

Problem Statement: Pivot the *Occupation* column in **OCCUPATIONS** so that each *Name* is sorted alphabetically and displayed underneath its corresponding *Occupation*. The output column headers should be *Doctor*, *Professor*, *Singer*, and *Actor*, respectively.

Note: Print **NULL** when there are no more names corresponding to an occupation.

Input Format

The **OCCUPATIONS** table is described as follows:

Column	Type
Name	String
Occupation	String

Occupation will only contain one of the following values: **Doctor**, **Professor**, **Singer** or **Actor**.

Sample Input

Name	Occupation
Samantha	Doctor
Julia	Actor
Maria	Actor
Meera	Singer
Ashely	Professor
Ketty	Professor
Christeen	Professor
Jane	Actor
Jenny	Doctor
Priya	Singer

Sample Output

Doctor	Professor	Singer	Actor
Jenny	Ashley	Meera	Jane
Samantha	Christeen	Priya	Julia
NULL	Ketty	NULL	Maria

Explanation

1. The first column is an alphabetically ordered list of Doctor names.
2. The second column is an alphabetically ordered list of Professor names.
3. The third column is an alphabetically ordered list of Singer names.
4. The fourth column is an alphabetically ordered list of Actor names.
5. The empty cell data for columns with less than the maximum number of names per occupation (in this case, the Professor and Actor columns) are filled with **NULL** values.

SOLUTION

Mental Approach:

1. Basically at first we will make unique column for each distinct occupation that is available in occupation column.
2. We will traverse for the particular Name that belongs to the particular Occupation.
3. Now we will put that Names for their corresponding Occupations one by one.

Query for creating different occupation columns with Name:

```
SELECT CASE WHEN Occupation='Doctor' THEN Name END Doctor
      ,CASE WHEN Occupation='Professor' THEN Name END Professor
      ,CASE WHEN Occupation='Singer' THEN Name END Singer
      ,CASE WHEN Occupation='Actor' THEN Name END Actor
FROM Occupations
```

Output we get:

Doctor	Professor	Singer	Actor
NULL	Ashley	NULL	NULL
NULL	NULL	NULL	Samantha
Julia	NULL	NULL	NULL
NULL	Britney	NULL	NULL
NULL	Maria	NULL	NULL

Query Steps: Creating different columns with CASE WHEN and then inserting the Name values for matching WHEN condition.

Problem: In output, we can see that a single Name is coming in all rows for different occupations but actually a row should contain Names for all columns unless there is no available Name for that particular Occupation.

Solution: So, here we must aggregate the Name so that we can group them together.

Query for give row number for each rows:

```
SELECT Occupation
      ,Name
      ,ROW_NUMBER() OVER (PARTITION BY Occupation ORDER BY Name ) rn
FROM Occupations
```

Output of Subquery:

Occupation	Name	rn
Actor	Eve	1
Actor	Jennifer	2
Actor	Ketty	3
Actor	Samantha	4
Doctor	Aamina	1
Doctor	Julia	2
Doctor	Priya	3
Professor	Ashley	1
Professor	Belvet	2
Professor	Britney	3
Professor	Maria	4
Professor	Meera	5
Professor	Naomi	6
Professor	Priyanka	7
Singer	Christeen	1
Singer	Jane	2
Singer	Jenny	3
Singer	Kristeen	4

1. Here with ROW_NUMBER firstly we are partitioning on the basis of Occupation so that Names for every Occupation get separated.
2. After partitioning, we are providing a row number to each row on the basis of the Name column by providing the Name in the ORDER BY clause within the ROW_NUMBER function.
3. Thus, we see row number on the basis of ascending order of Name for that particular occupation starting with row number 1 for each new occupation.

Final Query:

```
SELECT MAX(CASE WHEN Occupation='Doctor' THEN Name END) Doctor
      ,MAX(CASE WHEN Occupation='Professor' THEN Name END) Professor
      ,MAX(CASE WHEN Occupation='Singer' THEN Name END) Singer
      ,MAX(CASE WHEN Occupation='Actor' THEN Name END) Actor
FROM (SELECT Occupation
            ,Name
            ,ROW_NUMBER() OVER (PARTITION BY Occupation ORDER BY
Name ) rn
      FROM Occupations) subquery
GROUP BY subquery.rn
```

Output:

Doctor	Professor	Singer	Actor
Aamina	Ashley	Christeen	Eve
Julia	Belvet	Jane	Jennifer
Priya	Britney	Jenny	Ketty
NULL	Maria	Kristeen	Samantha
NULL	Meera	NULL	NULL
NULL	Naomi	NULL	NULL
NULL	Priyanka	NULL	NULL

1. In the above query we are using a query of finding row_number as a subquery.
2. Here, we need to aggregate the output as we want the Name with row number 1 to be placed in a single row.
3. In the output we are getting NULL because for a particular Occupation there is no any Name for that particular row number.

BY MANISH KUMAR

PROBLEM STATEMENT: Harry Potter and his friends are at Ollivander's with Ron, finally replacing Charlie's old broken wand. Hermione decides the best way to choose is by determining the minimum number of gold galleons needed to buy each non-evil wand of high power and age. Write a query to print the id, age, coins_needed, and power of the wands that Ron's interested in, sorted in order of descending power. If more than one wand has same power, sort the result in order of descending age.

Input Format

The following tables contain data on the wands in Ollivander's inventory:

- Wands: The id is the id of the wand, code is the code of the wand, coins_needed is the total number of gold galleons needed to buy the wand, and power denotes the quality of the wand (the higher the power, the better the wand is).

Column	Type
id	Integer
code	Integer
coins_needed	Integer
power	Integer

- Wands_Property: The code is the code of the wand, age is the age of the wand, and is_evil denotes whether the wand is good for the dark arts. If the value of is_evil is 0, it means that the wand is not evil. The mapping between code and age is one-one, meaning that if there are two pairs, and , then and .

Column	Type
code	Integer
age	Integer
is_evil	Integer

Sample Input

Wands Table:

id	code	coins_needed	power
1	4	3688	8
2	3	9365	3
3	3	7187	10
4	3	734	8
5	1	6020	2
6	2	6773	7
7	3	9873	9
8	3	7721	7
9	1	1647	10
10	4	504	5
11	2	7587	5
12	5	9897	10
13	3	4651	8
14	2	5408	1
15	2	6018	7
16	4	7710	5
17	2	8798	7
18	2	3312	3
19	4	7651	6
20	5	5689	3

Wands_Property Table:

code	age	is_evil
1	45	0
2	40	0
3	4	1
4	20	0
5	17	0

Sample Output

9 45 1647 10
12 17 9897 10
1 20 3688 8
15 40 6018 7
19 20 7651 6
11 40 7587 5
10 20 504 5
18 40 3312 3
20 17 5689 3
5 45 6020 2
14 40 5408 1

EXPLANATION:

The data for wands of *age 45* (code 1):

id	age	coins_needed	power
5	45	6020	2
9	45	1647	10

- The minimum number of galleons needed for **wand(age =45,power=2) =6020**
- The minimum number of galleons needed for **wand(age =45,power=10) =1647**

The data for wands of *age 40* (code 2):

id	age	coins_needed	power
14	40	5408	1
18	40	3312	3
11	40	7587	5
15	40	6018	7
17	40	8798	7
6	40	6773	7

- The minimum number of galleons needed for **wand(age =40,power=1) =5408**
- The minimum number of galleons needed for **wand(age =40,power=3) =3312**
- The minimum number of galleons needed for **wand(age =40,power=5) =7587**
- The minimum number of galleons needed for **wand(age =40,power=7) =6018**

The data for wands of *age 20* (code 4):

id	age	coins_needed	power
10	20	504	5
16	20	7710	5
19	20	7651	6
1	20	3688	8

- The minimum number of galleons needed for **wand(age =20,power=5) =504**
- The minimum number of galleons needed for **wand(age =20,power=6) =7710**
- The minimum number of galleons needed for **wand(age =20,power=8) =3688**

The data for wands of *age 17* (code 5):

id	age	coins_needed	power
20	17	5689	3
12	17	9897	10

- The minimum number of galleons needed for **wand(age =17,power=3) =5689**
- The minimum number of galleons needed for **wand(age =17,power=10) =9897**

QUERY:

```
SELECT id
      ,age
      ,coins_needed
      ,power
FROM (SELECT w.id
            ,wp.age
            ,w.coins_needed
            ,w.power
            ,RANK() OVER (PARTITION BY wp.age,w.power ORDER BY w.coins_needed ASC)
rnk
      FROM wands w
      INNER JOIN wands_property wp
      ON w.code=wp.code
      WHERE wp.is_evil=0
      ) ranking
WHERE rnk=1
ORDER BY power DESC,age DESC
```

QUERY EXPLANATION:

1. We're using SUBQUERY "ranking" to sort the coins_needed in ascending order. for each partition formed on the basis of age and power.
   We ordered in ascending order here because we want the most coins needed (gold galleons) at the top of each partition.

2. Using the outer query, we are now choosing the appropriate columns using a filter condition to obtain records where rank is 1 for all separate partitions.

26 January 2023 11:22

PROBLEM STATEMENT : Write a query to print all *prime numbers* less than or equal to 1000 . Print your result on a single line, and use the ampersand (&) character as your separator (instead of a space).

For example, the output for all prime numbers would be:
2&3&5&7

QUERY:

```
WITH cte AS (
SELECT 2 n
UNION ALL
SELECT n+1 n
FROM cte
WHERE n<1000
)
SELECT STRING_AGG(c1.n,'&')
FROM cte c1
WHERE NOT EXISTS (SELECT n
                  FROM cte c2
                  WHERE c1.n%c2.n=0
                  AND c1.n<>c2.n)
OPTION (MAXRECURSION 1000)
```

QUERY EXPLANATION:

1. With recursive CTE we are creating the series of number from till 1000. In first SELECT statement we are simply getting 2 (because 2 is smallest prime number) and in second SELECT statement we recursively calling the CTE by increasing value of n by 1.
   NOTE: Here, recursive CTE acts like a loop that is it will get executed till the condition is met.
2. Now after CTE we are simply using STRING_AGG function in SELECT statement to get the result as per our requirement.
3. To get the prime number we are using the NOT EXISTS in WHERE condition along with subquery.
4. In subquery we are selecting all n where value of n from c1 is divisible by value of n from another table c2 and value of n from table c1 is not equal to value of n from table c2.
   NOTE: Here, this we are doing so that we can check if number n is divisible by all other numbers except of itself or not.
5. So, here NOT EXISTS will help us to get the result except of the result provided by subquery.

NOTE: AS per different databases there are different syntax for combining the result of different columns in one row. In MS SQL Server we can use STRING_AGG.

OUTPUT:

2&3&5&7&11&13&17&19&23&29&31&37&41&43&47&53&59&61&67&71&73&79&83&89&97&101&103&107&109&113&127&131&137&139&149&151&157&163&167&173&179&181&191&193&197&199&211&223&227&229&233&239&241&251&257&263&269&271&277&281&283&293&307&311&313&317&331&337&347&349&353&359&367&373&379&383&389&397&401&409&419&421&431&433&439&443&449&457&461&463&467&479&487&491&499&503&509&521&523&541&547&557&563&569&571&577&587&593&599&601&607&613&617&619&631&641&643&647&653&659&661&673&677&683&691&701&709&719&727&733&739&743&751&757&761&769&773&787&797&809&811&821&823&827&829&839&853&857&859&863&877&881&883&887&907&911&919&929&937&941&947&953&967&971&977&983&991&997

PROBLEM STATEMENT: A company wants to track the entries of users inside its premises. Users can only log in once a day, but there is a loophole that allows a user to log in with a different email address. We must therefore capture this loophole by finding the number of times a customer visited as well as the floor on which they visited the most. In addition, we must determine what resources they use.

Table name: **entries**

name	address	email	floor	resources
A	Bangalore	A@gmail.com	1	CPU
A	Bangalore	A1@gmail.com	1	CPU
A	Bangalore	A2@gmail.com	2	DESKTOP
B	Bangalore	B@gmail.com	2	DESKTOP
B	Bangalore	B1@gmail.com	2	DESKTOP
B	Bangalore	B2@gmail.com	1	MONITOR

MENTAL APPROACH:

1. Find the distinct names that have entered the premises.
2. Because their email addresses changed, they are on different rows, so you can count their number of visits by counting their name.
3. Count how many times a user visited a particular floor to find out which floor has been visited most often.
4. Now that we want resources used by them, we will go through and find distinct resources that are used by each user and then combine them all into one row.

QUERY:

```sql
WITH floor_visited AS (
SELECT name
      ,floor
      ,COUNT(1) floor_visits_times
      ,RANK() OVER (PARTITION BY name ORDER BY COUNT(1) DESC) rnk
FROM entries
GROUP BY name, floor
),
total_visits AS (
SELECT name
      ,COUNT(1) total_visit
FROM entries
GROUP BY name
),
agg_cte AS (
SELECT name
      ,STRING_AGG(resources,',') resources_used
FROM (SELECT name, resources FROM entries GROUP BY name,resources) sub
GROUP BY name
)
SELECT f.name
      ,t.total_visit
      ,f.floor most_visited_floor
      ,a.resources_used
FROM floor_visited f
INNER JOIN total_visits t
ON f.name=t.name
INNER JOIN agg_cte a
ON t.name=a.name
WHERE f.rnk=1
```

QUERY EXPLANATION:

1. There are three different CTEs: floor_visited, total_visits, and agg_cte.
   By CTE floor_visited, we can determine how many times a floor has been visited by a specific user and rank them accordingly in decreasing order.
   In order to determine how many times a particular user has visited the premises, use the CTE total_visits.
   Using CTE agg_cte, you can combine the different resources used by a particular user.

2. SELECT statement after CTE selects required fields and filters with WHERE condition of rnk=1.
   We are using a WHERE condition here so that we only include records from users who have visited a particular floor the most number of times.

CREATE STATEMENT FOR PRACTICE PURPOSE:

```
create table entries (
name varchar(20),
address varchar(20),
email varchar(20),
floor int,
resources varchar(10)
);

insert into entries values
('A','Bangalore','A@gmail.com',1,'CPU'),
('A','Bangalore','A1@gmail.com',1,'CPU'),
('A','Bangalore','A2@gmail.com',2,'DESKTOP'),
('B','Bangalore','B@gmail.com',2,'DESKTOP'),
('B','Bangalore','B1@gmail.com',2,'DESKTOP'),
('B','Bangalore','B2@gmail.com',1,'MONITOR');
```

NOTE

FOR MORE SUCH QUESTIONS VISIT ANKIT BANSAL Sir's Yotube Channel: Click Here
FOR MORE SUCH WRITTEN EXPLANATION BY ME: Click Here

01 January 2023 09:10

Problem Statement : Query all columns for all American cities in the **CITY** table with populations larger than 100000. The **CountryCode** for America is USA.

The **CITY** table is described as follows:

CITY

Field	Type
ID	NUMBER
NAME	VARCHAR2(17)
COUNTRYCODE	VARCHAR2(3)
DISTRICT	VARCHAR2(20)
POPULATION	NUMBER

First Step: For selecting all columns we will select by column names but not using * because it lowers the performance of the code.

```
SELECT ID
      ,NAME
      ,COUNTRYCODE
      ,DISTRICT
      ,POPULATION
FROM CITY
```

This will list the output for all the cities without any filter.

Sample output

ID	NAME	COUNTRYCODE	DISTRICT	POPULATION
6	Rotterdam	NLD	Zuid-Holland	593321
3878	Scottsdale	USA	Arizona	202705
3965	Corona	USA	California	124966
3973	Concord	USA	California	121780
3977	Cedar Rapids	USA	Iowa	120758
3982	Coral Springs	USA	Florida	117549
4054	Fairfield	USA	California	11754
4058	Boulder	USA	Colorado	91238

4054	Fairfield	USA	California	11754
4058	Boulder	USA	Colorado	91238
4061	Fall River	USA	Massachusetts	90555

Second Step: We need to filter above query so that we only see output of American cities.

```
SELECT ID
      ,NAME
      ,COUNTRYCODE
      ,DISTRICT
      ,POPULATION
FROM CITY
WHERE COUNTRYCODE = 'USA';
```

This will give all details for American cities only.

Sample Output

ID	NAME	COUNTRYCODE	DISTRICT	POPULATION
3878	Scottsdale	USA	Arizona	202705
3965	Corona	USA	California	124966
3973	Concord	USA	California	121780
3977	Cedar Rapids	USA	Iowa	120758
3982	Coral Springs	USA	Florida	117549
4054	Fairfield	USA	California	11754
4058	Boulder	USA	Colorado	91238
4061	Fall River	USA	Massachusetts	90555

Third Step: Now we need to filter above query to provide output for those records only which are having population greater than 100000

```
SELECT ID
      ,NAME
      ,COUNTRYCODE
      ,DISTRICT
      ,POPULATION
FROM CITY
WHERE COUNTRYCODE = 'USA'
AND POPULATION > 100000;
```

This query will give use the final output that is all details of American country having population greater than 100000.

Sample Output

ID	NAME	COUNTRYCODE	DISTRICT	POPULATION
3878	Scottsdale	USA	Arizona	202705
3965	Corona	USA	California	124966
3973	Concord	USA	California	121780
3977	Cedar Rapids	USA	Iowa	120758
3982	Coral Springs	USA	Florida	117549

02 January 2023 11:04

Problem Statement : Query the **NAME** field for all American cities in the **CITY** table with populations larger than 120000. The *CountryCode* for America is USA.

The **CITY** table is described as follows:

Field	Type
ID	NUMBER
NAME	VARCHAR2(17)
COUNTRYCODE	VARCHAR2(3)
DISTRICT	VARCHAR2(20)
POPULATION	NUMBER

First Step: For selecting NAME column we will specify NAME column in SELECT statement.

```
SELECT NAME
FROM CITY;
```

This will list all the NAMES of cities irrespective of any particular country.

Sample Output

NAME
Rotterdam
Scottsdale
Corona
Concord
Cedar Rapids
Coral Springs
Fairfield
Boulder
Fall River

Second Step: We need to filterve query so the that we only see output of American cities.

```
SELECT NAME
FROM CITY
WHERE COUNTRYCODE='USA';
```

This will give all NAMES of American cities.

Sample Output

NAME
Scottsdale
Corona
Concord
Cedar Rapids
Coral Springs
Fairfield
Boulder
Fall River

Insight: From this, we can see that Rotterdam is a city of another country so it has been filtered out.

Third Step: Now we need to filter above query to provide output for those records only which are having population greater than 120000

```
SELECT NAME
FROM CITY
WHERE COUNTRYCODE='USA'
AND POPULATION>120000;
```

This query will give use the final output that is NAME of American cities having population greater than 120000.

Sample Output

NAME
Scottsdale
Corona
Concord
Cedar Rapids

Insight: From this, we can see that Population of Coral Springs, Fairfield, Boulder and Fall River are less than 120000 so they are filter out of our final output.

03 January 2023 09:22

Problem Statement: Write a query that calculates the difference between the highest salaries found in the marketing and engineering departments. Output just the absolute difference in salaries.

db_employee

id	int
first_name	varchar
last_name	varchar
salary	int
department_id	int

db_dept

id	int
department	varchar

FIRST APPROACH

1. To find the MAX salary for marketing (dept_id =4) department

```
SELECT MAX(e.salary) max_salary_marketing
FROM db_employee e
LEFT JOIN db_dept d
ON e.department_id=d.id
WHERE d.department = 'marketing'
```

Here we could have directly used **department_id = 4** in the **WHERE** clause without using **JOIN** as well but in real world, it was a chance that there are lot of departments then we would have gotten into difficulty.

2. To find the MAX salary for engineering (dept_id =1) department

```
SELECT MAX(e.salary) max_salary_engineering
FROM db_employee e
LEFT JOIN db_dept d
ON e.department_id=d.id
WHERE d.department = 'engineering'
```

Here also we could have directly used **department_id = 1**.

3. Now we need to find difference of both max_salary_marketing and max_salary_engineering.
   For this we could have done it in different ways.

   We can directly use both **SELECT** statements as **Subquery** and subtract one from other and use **ABS** function to get absolute output which.

   e.g (5-9) = - 4
   e.g **ABS**(5-9) = 4

```
SELECT ABS((SELECT MAX(e.salary)
max_salary_marketing
         FROM db_employee e
         LEFT JOIN db_dept d
         ON e.department_id=d.id
         WHERE d.department = 'marketing')
         -
         (SELECT MAX(e.salary)
max_salary_engineering
          FROM db_employee e
          LEFT JOIN db_dept d
          ON e.department_id=d.id
          WHERE d.department =
'engineering')) difference
```

Output

difference
2400

INSIGHT: This approach will take more time as here it will have to iterate through multiple select statement in subquery and then find max for both and then finally take the difference of them.

SECOND APPROACH

1. Find MAX salary for all departments

```
SELECT d.department
      ,MAX(e.salary) salary
FROM db_employee e
LEFT JOIN db_dept d
ON e.department_id = d.id
GROUP BY d.department;
```

Output:

department	salary
customer care	49926
engineering	45787
human resource	46356
marketing	48187
operation	49488
sales	47657

2. Now we will filter for department for which we need to find the difference of maximum salary. So, we will use **WHERE** clause with **IN**

```
SELECT d.department
      ,MAX(e.salary) salary
FROM db_employee e
LEFT JOIN db_dept d
ON e.department_id = d.id
WHERE d.department IN ('marketing','engineering')
GROUP BY d.department
```

Output:

department	salary
engineering	45787
marketing	48187

3. As we have not got the maximum salary of both the required departments. We can now get difference of MAX and MIN salary from this by using it as a subquery.

```
SELECT MAX(salary)-MIN(salary) salary_difference
FROM (SELECT d.department
            ,MAX(e.salary) salary
      FROM db_employee e
      LEFT JOIN db_dept d
      ON e.department_id = d.id
      WHERE d.department IN ('marketing','engineering')
      GROUP BY d.department) max_salary
```

Output:

salary_difference
2400

03 January 2023 08:16

Problem Statement: Query all columns for a city in **CITY** with the *ID* 1661.
The **CITY** table is described as follows:

Field	Type
ID	NUMBER
NAME	VARCHAR2(17)
COUNTRYCODE	VARCHAR2(3)
DISTRICT	VARCHAR2(20)
POPULATION	NUMBER

First Step: We will select all columns that is required within SELECT statement.

```
SELECT ID
      ,NAME
      ,COUNTRYCODE
      ,DISTRICT
      ,POPULATION
FROM CITY;
```

Sample Output

ID	NAME	COUNTRYCODE	DISTRICT	POPULATION
1350	Dehri	IND	Bihar	94526
1383	Tabriz	IRN	East Azerbaidzan	1191043
1385	Karaj	IRN	Teheran	940968
1508	Bolzano	ITA	Trentino-Alto Adige	97232
1520	Cesena	ITA	Emilia-Romagna	89852
1613	Neyagawa	JPN	Osaka	257315
1630	Ageo	JPN	Saitama	209442
1661	Sayama	JPN	Saitama	162472
1681	Omuta	JPN	Fukuoka	142889
1739	Tokuyama	JPN	Yamaguchi	107078
1793	Novi	Sad	YUG Vojvodina	179626
1857	Kelowna	CAN	British Colombia	89442

Second Step: Now we will filter for ID for which we need all details.

```
SELECT ID
      ,NAME
      ,COUNTRYCODE
      ,DISTRICT
      ,POPULATION
FROM CITY
WHERE ID = 1661;
```

Insight: In this query we have filtered for ID 1661. So it will list record that is having ID as 1661 only.

Sample Output:

ID	NAME	COUNTRYCODE	DISTRICT	POPULATION
1661	Sayama	JPN	Saitama	162472

15 January 2023 11:51

PROBLEM STATEMENT: You are given two tables: *Students* and *Grades*. *Students* contains three columns *ID*, *Name* and *Marks*.

Column	Type
ID	Integer
Name	String
Marks	Integer

Grades contains the following data:

Grade	Min_Mark	Max_Mark
1	0	9
2	10	19
3	20	29
4	30	39
5	40	49
6	50	59
7	60	69
8	70	79
9	80	89
10	90	100

PROBLEM DESCRIPTION:

1. *Ketty* gives *Eve* a task to generate a report containing three columns: *Name*, *Grade* and *Mark*.
2. Ketty doesn't want the NAMES of those students who received a grade lower than 8. The report must be in descending order by grade -- i.e. higher grades are entered first.
3. If there is more than one student with the same grade (8-10) assigned to them, order those particular students by their name alphabetically.
4. Finally, if the grade is lower than 8, use "NULL" as their name and list them by their grades in descending order.
5. If there is more than one student with the same grade (1-7) assigned to them, order those particular students by their marks in ascending order.

Write a query to help Eve.

Sample Input

ID	Name	Marks
1	Julia	88
2	Samantha	68
3	Maria	99
4	Scarlet	78
5	Ashley	63
6	Jane	81

Sample Output

Maria 10 99
Jane 9 81
Julia 9 88
Scarlet 8 78
NULL 7 63
NULL 7 68

Note

Print "NULL" as the name if the grade is less than 8.

Explanation

Consider the following table with the grades assigned to the students:

ID	Name	Marks	Grade
1	Julia	88	9
2	Samantha	68	7
3	Maria	99	10
4	Scarlet	78	8
5	Ashley	63	7
6	Jane	81	9

So, the following students got *8*, *9* or *10* grades:

- *Maria (grade 10)*
- *Jane (grade 9)*
- *Julia (grade 9)*
- *Scarlet (grade 8)*

For practicing this question, go to this link.

SOLUTION

MENTAL APPROACH:

1. Output Name, Grade and Marks from both table by combining them together.
2. We have been provided with minimum and maximum marks for Grades. Thus, we need to search for marks that are present in Students table in Grades table's within which range it lies i.e between the minimum and maximum marks of Grades table.
3. Now we will sort the result on the basis of Grades. If one or more students having same Grades then we will sort them on the basis of their Name.
4. At last we will replace the Student Name with "NULL" for those students who are having Grade less than 8.

QUERY:

```
SELECT CASE WHEN G.Grade <8 THEN "NULL"
            ELSE S.Name
       END Name
      ,G.Grade
      ,S.Marks
FROM Students S
LEFT JOIN Grades G
ON S.Marks BETWEEN Min_Mark AND Max_Mark
ORDER BY G.Grade DESC, S.Name
```

QUERY EXECUTION:

1. First of all FROM and JOIN will be executed and both tables will get combined on the basis of the ON condition.
   NOTE: Here, we have used BETWEEN because Marks in Students table is not present directly in the Grades table but instead it is provided in a range that is minimum and maximum marks.
2. After this our SELECT statement will be executed and then CASE WHEN conditions will be fulfilled. Here, we have used CASE WHEN to replace the students' name with "NULL" where their Grade is less than 8 and get the name of Students where Grade is greater than or equal to 8.
3. Now at last our output will be sorted or ordered on the basis of the ORDER BY clause.

OUTPUT:

Name	Grade	Marks
Britney	10	95
Heraldo	10	94
Julia	10	96
Kristeen	10	100
Stuart	10	99
Amina	9	89
Christene	9	88
Salma	9	81
Samantha	9	87
Scarlet	9	80
Vivek	9	84
Aamina	8	77
Belvet	8	78
Paige	8	74
Priya	8	76
Priyanka	8	77
NULL	7	64
NULL	7	66
NULL	6	55
NULL	4	34

17 January 2023 15:27

PROBLEM STATEMENT: Julia just finished conducting a coding contest, and she needs your help assembling the leaderboard! Write a query to print the respective *hacker_id* and *name* of hackers who achieved full scores for *more than one* challenge. Order your output in descending order by the total number of challenges in which the hacker earned a full score. If more than one hacker received full scores in same number of challenges, then sort them by ascending *hacker_id*.

Input Format

The following tables contain contest data:

- *Hackers:* The *hacker_id* is the id of the hacker, and *name* is the name of the hacker.

Column	Type
hacker_id	Integer
name	String

- *Difficulty:* The *difficult_level* is the level of difficulty of the challenge, and *score* is the score of the challenge for the difficulty level.

Column	Type
difficulty_level	Integer
score	Integer

- *Challenges:* The *challenge_id* is the id of the challenge, the *hacker_id* is the id of the hacker who created the challenge, and *difficulty_level* is the level of difficulty of the challenge.

Column	Type
challenge_id	Integer
hacker_id	Integer
difficulty_level	Integer

- *Submissions:* The *submission_id* is the id of the submission, *hacker_id* is the id of the hacker who made the submission, *challenge_id* is the id of the challenge that the submission belongs to, and *score* is the score of the submission.

Column	Type
submission_id	Integer
hacker_id	Integer
challenge_id	Integer
score	Integer

Sample Input

Hackers Table:

hacker_id	name
5580	Rose
8439	Angela
27205	Frank
52243	Patrick
52348	Lisa
57645	Kimberly
77726	Bonnie
83082	Michael
86870	Todd
90411	Joe

Difficulty Table:

difficulty_level	score
1	20
2	30
3	40
4	60
5	80
6	100
7	120

Challenges Table:

challenge_id	hacker_id	difficulty_level
4810	77726	4
21089	27205	1
36566	5580	7
66730	52243	6
71055	52243	2

Submissions Table:

submission_id	hacker_id	challenge_id	score
68628	77726	36566	30
65300	77726	21089	10
40326	52243	36566	77
8941	27205	4810	4
83554	77726	66730	30
43353	52243	66730	0
55385	52348	71055	20
39784	27205	71055	23
94613	86870	71055	30
45788	52348	36566	0
93058	86870	36566	30
7344	8439	66730	92
2721	8439	4810	36
523	5580	71055	4
49105	52348	66730	0
55877	57645	66730	80
38355	27205	66730	35
3924	8439	36566	80
97397	90411	66730	100
84162	83082	4810	40
97431	90411	71055	30

Sample Output
90411 Joe

Explanation

Hacker *86870* got a score of *30* for challenge *71055* with a difficulty level of *2*, so *86870* earned a full score for this challenge.

Hacker *90411* got a score of *30* for challenge *71055* with a difficulty level of *2*, so *90411* earned a full score for this challenge.

Hacker *90411* got a score of *100* for challenge *66730* with a difficulty level of *6*, so *90411* earned a full score for this challenge.

Only hacker *90411* managed to earn a full score for more than one challenge, so we print the

their *hacker_id* and *name* as space-separated values

SOLUTION

MENTAL APPROACH:

1. Firstly we will combine all the tables one by one together so that we can get our desired output easily and efficiently.

2. Now we will filter for hackers who have scored full scores. (In SQL we will do this by using the WHERE clause)

3. As we want hackers who have submitted more than one challenge, we will filter this out. (Using HAVING Clause)
4. We will n
5. ow order our output on the basis of descending order of the number of challenges solved by hackers.

6. If there are multiple hackers who have solved the same number of challenges then we will order on the basis of ascending order of hacker id.

QUERY:

```
SELECT s.hacker_id
      ,MAX(h.name) name
FROM submissions s
LEFT JOIN challenges c
ON s.challenge_id = c.challenge_id
LEFT JOIN difficulty d
ON c.difficulty_level = d.difficulty_level
LEFT JOIN hackers h
ON s.hacker_id = h.hacker_id
WHERE d.score=s.score
GROUP BY s.hacker_id
HAVING COUNT(s.challenge_id)>1
ORDER BY COUNT(s.challenge_id) DESC, hacker_id ASC
```

QUERY EXECUTION:

1. FROM and JOIN clauses will be executed in the first step.
   NOTE: Here, we are using the submission table as the main table for joining other tables because we are details of challenges submitted by hackers are provided in this table.

2. Filtering on the basis of the WHERE clause will be executed by making sure that score in the submissions table is equal to the score present in the difficulty table. Now we will be left with records of those hackers who have scored full scores in the challenge.
   Here, we have used the score from the difficulty table to compare it with the score from the submissions table because it contains the maximum score for a particular difficulty level.

3. Now GROUP BY will get executed and it will group the above result on the basis of hacker id.

Here, it was done to get distinct hackers as it was repeating because the same hacker has submitted multiple challenges and as well as to filter out with the HAVING clause.

4. After GROUPING now HAVING clause will get executed and it will filter out the records on the basis of the condition provided.
   Here, we are basically filtering so that we can get records of those hackers who have submitted more than one challenge.

5. Now the SELECT query will be executed and we will get hacker_id and hacker name as output.
   NOTE: Here, we are using the aggregate function MAX() for the hacker name because we have to group the records on the basis of the hacker id to get the desired output.
6. At last we are using the ORDER BY clause to order our output.

SAMPLE OUTPUT:

hacker_id	name
27232	Phillip
28614	Willie
15719	Christina
43892	Roy
14246	David
14372	Michelle
18330	Lawrence
26133	Jacqueline
26253	John
30128	Brandon
35583	Norma

20 January 2023 17:35

PROBLEM STATEMENT: Calculate each user's average session time. A session is defined as the time difference between a page_load and page_exit. For simplicity, assume a user has only 1 session per day and if there are multiple of the same events on that day, consider only the latest page_load and earliest page_exit. Output the user_id and their average session time.

FIELD	TYPE
user_id	int
timestamp	datetime
action	varchar

TABLE:

user_id	timestamp	action
0	25-04-2019 13:30	page_load
0	25-04-2019 13:30	page_load
0	25-04-2019 13:30	scroll_down
0	25-04-2019 13:30	scroll_up
0	25-04-2019 13:31	page_exit
1	25-04-2019 13:40	page_load
1	25-04-2019 13:40	scroll_down
1	25-04-2019 13:40	page_exit
2	25-04-2019 13:41	page_load
2	25-04-2019 13:41	scroll_down
2	25-04-2019 13:41	scroll_down
2	25-04-2019 13:41	scroll_up
1	26-04-2019 11:15	page_load
1	26-04-2019 11:15	scroll_down
1	26-04-2019 11:15	scroll_up
1	26-04-2019 11:15	page_exit
0	28-04-2019 14:30	page_load
0	28-04-2019 14:30	page_load
0	28-04-2019 13:30	scroll_down
0	28-04-2019 15:31	page_exit

MENTAL APPROACH:

1. Find the page load time for all user for all different users on different days. As per question we should consider the latest page load time.

   NOTE: Here, latest page load time means if same user for same day is having different page load time then we should choose one which is the most recent (or we can say maximum time out of all available time)

2. Now similarly find the page exit time for all different users on different days. But here we should consider the which ever is the earliest one (that is minimum time out of all available time).

3. We want session time which is difference of page exit time and page load time divided by the total number of days (here it is basically calculating average)
   Here, we are told to consider that in one day user logins one time only, so it means one day here we consider as one session

4. As we have got page load and page exit time for all users for all different days, now we will calculate the difference of page exit time and page load time for all users for all different days.

5. Now we need to count the number of sessions for each user. (which is basically count number of days a user has logged in)

6. Finally, we will calculate the session time by using step 3 formula for each users.

QUERY:

```sql
WITH timestamp_cte AS (
SELECT user_id
      ,CASE WHEN action = 'page_load' THEN timestamp END page_load
      ,CASE WHEN action = 'page_exit' THEN timestamp END page_exit
      ,DAY(timestamp) day_time
FROM facebook_web_log
WHERE action IN ('page_load','page_exit')
)
,page_load_exit_cte AS (
SELECT user_id
      ,MAX(page_load)   page_load
      ,MIN(page_exit)  page_exit
FROM timestamp_cte
GROUP BY user_id, day_time
)
SELECT user_id
      ,CAST((AVG(DATEDIFF(SECOND,page_load,page_exit)*1.0)) AS NUMERIC(15,1)) average_session_time
FROM page_load_exit_cte
GROUP BY user_id
HAVING AVG(DATEDIFF(SECOND,page_load,page_exit)) <> 0
```

QUERY EXPLANATION:

1. We are using CTE for solving this question. (Subquery can also be used)

2. First CTE that is timestamp_cte we finding the dividing the page load and exit time in different columns. Here, we are removing the other actions apart from page load and exit because they are of no use. At last, we are grouping them on users and day wise.

user_id	page_load	page_exit	day_time
0	25-04-2019 13:30		25
0	25-04-2019 13:30		25
0		25-04-2019 13:31	25
1	25-04-2019 13:40		25
1		25-04-2019 13:40	25
2	25-04-2019 13:41		25
1	26-04-2019 11:15		26
1		26-04-2019 11:15	26
0	28-04-2019 14:30		28
0	28-04-2019 14:30		28
0		28-04-2019 15:31	28

3. Second CTE that is page_load_exit_cte where we are combining both page load and exit time in single row for a single user for one particular day.

user_id	page_load	page_exit
0	25-04-2019 13:30	25-04-2019 13:31
1	25-04-2019 13:40	25-04-2019 13:40
2	25-04-2019 13:41	
1	26-04-2019 11:15	26-04-2019 11:15
0	28-04-2019 14:30	28-04-2019 15:31

4. At last we are extracting user id and average session time from page_load_exit_cte and filtering out those users whose average session time is NULL.

user_id	average_session_time
0	1883.5
1	35

04 January 2023 10:36

```
/*Problem Statement: Query a list of CITY names from STATION for cities
that have an even ID number. Print the results in any order, but exclude
duplicates from the answer.
The STATION table is described as follows:*/

/*First Step : Let us have look at ID and CITY column from STATION
Table*/
SELECT ID
      ,CITY
FROM STATION;

/*Sample Output*/

_____________________
ID   | NAME
_____________________
794 | Kissee Mills
824 | Loma Mar
603 | Sandy Hook
478 | Tipton
619 | Arlington
711 | Turner
839 | Slidell
411 | Negreet
588 | Glencoe
754 | Tipton


/*Second Step: Now we will filter for Name for which ID is an even
number. For this we will use  WHERE clause.*/

SELECT CITY
```

```sql
SELECT CITY
FROM STATION
WHERE ID % 2 =0;
/* % is used for getting reminder, so if reminder is 0 that means it is
divisible by 2 thus a number divisible by 2 means it is even number */

/*Sample Output*/

_____________________
ID   | NAME
_____________________
794 | Kissee Mills
824 | Loma Mar
478 | Tipton
588 | Glencoe
754 | Tipton

/*Third Step: In question we are told not to have duplicate CITY so we
will use DISTINCT in SELECT Statement*/

SELECT DISTINCT CITY
FROM STATION
WHERE ID % 2 = 0;

_____________________
ID   | NAME
_____________________
794 | Kissee Mills
824 | Loma Mar
478 | Tipton
588 | Glencoe
```

05 January 2023 08:26

COUNT() Function

COUNT() function is used to count the number of records in a table.

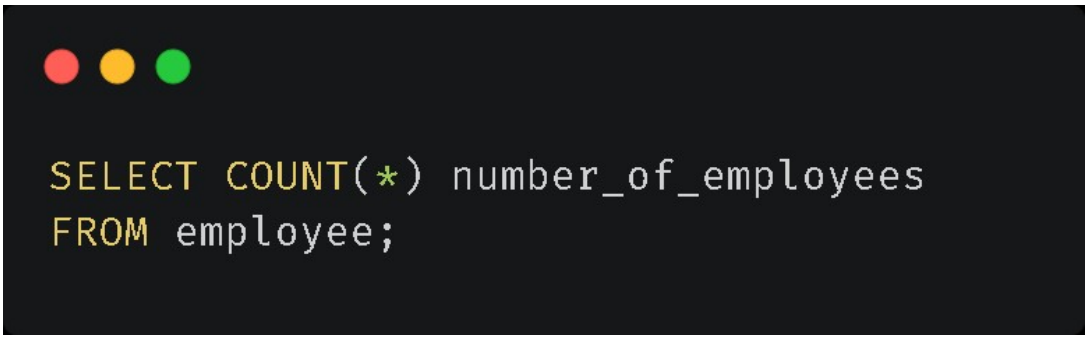
SYNTAX :

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

For example purpose we will use this employee table:

ID	FIRST NAME	LAST NAME	SALARY
1	Prayash	Kumar	29000
2	Ranya	Gupta	41000
3	Anusha		56000
4	Salone	Yadav	32000
5	Ranya		24000

COUNT(*) : It will count all the records present in a table irrespective of NULL values which means it will count all the NULL and NON-NULL values.

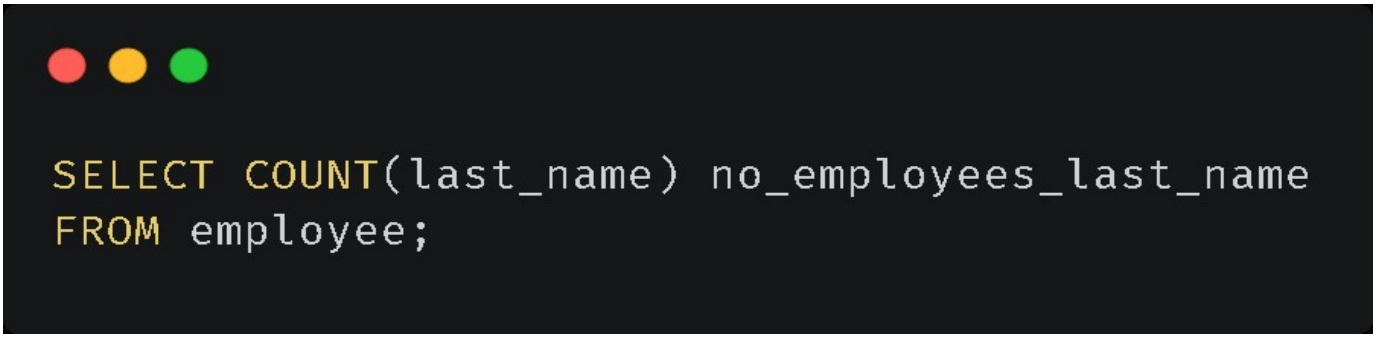


```
SELECT COUNT(*) number_of_employees
FROM employee;
```

Output

number_of_employees
5

COUNT(column_name) : It will count all the NON-NULL records present in a table.



```
SELECT COUNT(last_name) no_employees_last_name
FROM employee;
```

This query will count the number of employees of those who are having last name.

Output

number_of_employees
3

Insight: It is showing 3 as output because it has filtered out the null records and count the non-null records (i.e Kumar, Gupta, Yadav) from the last_name columns. So basically now it is using below table to count the records.

ID	FIRST NAME	LAST NAME	SALARY
1	Prayash	Kumar	29000
2	Ranya	Gupta	41000
4	Salone	Yadav	32000

NOTE : If we want distinct first name from employee table then we will use DISTINCT FIRST NAME inside the COUNT() function.

COUNT(DISTINCT column_name): This will count unique records present in a particular column.

```
SELECT COUNT(DISTINCT first_name) unique_names
FROM employee;
```

Output

number_of_employees
4

Insight: It is showing 4 as output because it is counting only distinct or unique records (i.e Prayash, Ranya, Anusha, Salone) from the firs_name column. It has removed the duplicate record for Ranya. It is using below table to count distinct records.

ID	FIRST NAME	LAST NAME	SALARY
1	Prayash	Kumar	29000
2	Ranya	Gupta	41000
3	Anusha		56000
4	Salone	Yadav	32000

Question based out of COUNT() function from HackerRank.

Problem Statement: Find the difference between the total number of **CITY** entries in the table and the number of distinct **CITY** entries in the table.

The **STATION** table is described as follows:

STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where **LAT_N** is the northern latitude and **LONG_W** is the western longitude.

For example, if there are three records in the table with **CITY** values 'New York', 'New York', 'Bengalaru', there are 2 different city names: 'New York' and 'Bengalaru'. The query returns 1 , because .

total number of records - number of unique city names = 3 - 2 = 1

Query

```
SELECT COUNT(CITY)-COUNT(DISTINCT CITY) difference
FROM STATION
```

Output

difference
13

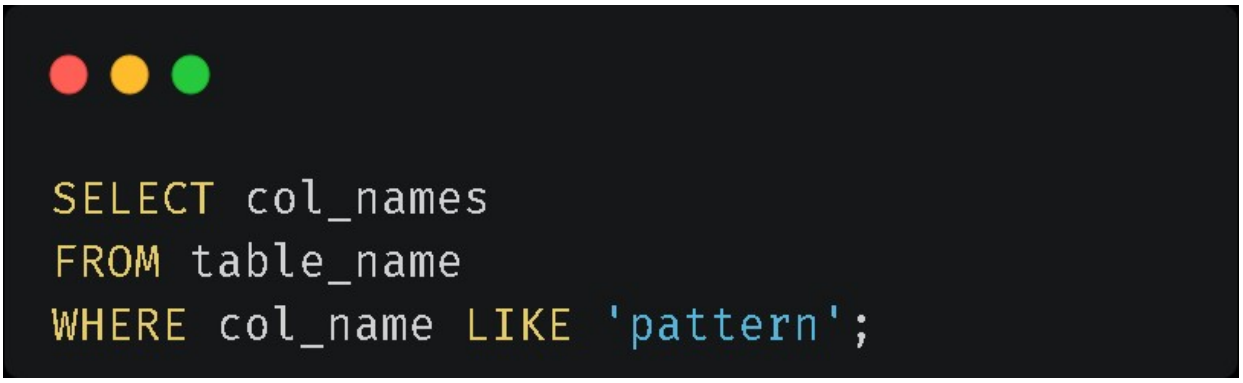
06 January 2023 08:09

LIKE OPERATOR

In SQL like operator is used in WHERE clause to search for any type of pattern within a column.

Like Operator has few wildcards which is listed in table.

SYNTAX:



Wildcard character	Description	Example
%	Any string of zero or more characters.	WHERE title LIKE '%computer%' finds all book titles with the word 'computer' anywhere in the book title.
_ (underscore)	Any single character.	WHERE au_fname LIKE '_ean' finds all four-letter first names that end with ean (Dean, Sean, and so on).
[ ]	Any single character within the specified range ([a-f]) or set ([abcdef]).	WHERE au_lname LIKE '[C-P]arsen' finds author last names ending with arsen and starting with any single character between C and P, for example Carsen, Larsen, Karsen, and so on. In range searches, the characters included in the range may vary depending on the sorting rules of the collation.
[^]	Any single character not within the specified range ([^a-f]) or set ([^abcdef]).	WHERE au_lname LIKE 'de[^l]%' finds all author last names starting with de and where the following letter isn't l.

For more information visit: Mircrosoft Learning Website by clicking here.

Problem Statement: Query the list of *CITY* names starting with vowels (i.e., a, e, i, o, or u) from **STATION**. Your result *cannot* contain duplicates.

Input Format

The **STATION** table is described as follows:

STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where *LAT_N* is the northern latitude and *LONG_W* is the western longitude.

Mental Approach

1. We find all the CITY names first. (in sql we use SELECT clause)
2. Now we will only choose the CITY names which are starting with a,e,i,o,u. (in sql we use WHERE clause with LIKE operator)
3. We now find the distinct CITY name out of above list because we don't want duplicate records. (in sql we do this using DISTINCT in SELECT clause)

Query To SELECT all CITY name

```
SELECT CITY
FROM STATION;
```

Sample Output

NAME
Arlington
Turner
Slidell
Negreet
Glencoe
Alisa
Chignik Lagoon
Pelahatchie
Ellen
Dorrance
Albany
Usain

Final Query

```
SELECT DISTINCT CITY
FROM STATION
WHERE CITY LIKE '[a,e,i,o,u]%';
```

Sample Output

Name
Arlington
Alisa
Ellen
Albany
Usain

What To Study

06 January 2023 09:47

PROBLEM STATEMENT: Query the list of *CITY* names from **STATION** that *do not start* with vowels. Your result cannot contain duplicates.

Input Format

The **STATION** table is described as follows:

STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where *LAT_N* is the northern latitude and *LONG_W* is the western longitude.

SAMPLE TABLE:

ID	CITY	STATE	LAT_N	LONG_W
794	Kissee Mills	MO	140	73
824	Loma Mar	CA	49	131
603	Sandy Hook	CT	72	148
478	Tipton	IN	34	98
619	Arlington	CO	75	93
711	Turner	AR	50	101
839	Slidell	LA	85	152
411	Negreet	LA	99	105

QUERY :

```
SELECT DISTINCT CITY
FROM STATION
WHERE CITY LIKE '[^aeiou]%';
```

SAMPLE OUTPUT :

CITY
Kissee Mills
Loma Mar
Sandy Hook
Tipton
Turner
Slidell
Negreet

INSIGHTS: All names starting with a,e,i,o,u are removed from provided data.

09 January 2023 10:25

Query the *Western Longitude* (*LONG_W*) for the largest *Northern Latitude* (*LAT_N*) in **STATION** that is less than 137.2345 . Round your answer to 4 decimal places.

Input Format

The **STATION** table is described as follows:

STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where *LAT_N* is the northern latitude and *LONG_W* is the western longitude.

Mental Approach:

1. We will search for a maximum value of LAT_N which is less than 137.2345. (In SQL query we will use WHERE clause for it)
2. Now corresponding to that value we will look LONG_W value. (In SQL query we do it using SELECT clause)
3. At last we will round the LONG_W value to 4 decimal places. (In SQL within SELECT clause we round the value using some ROUND functions)

Query:

```
SELECT CAST(LONG_W AS DECIMAL(15,4)) LONG_W
FROM STATION
WHERE LAT_N = (SELECT MAX(LAT_N) FROM STATION
               WHERE LAT_N <137.2345)
```

Output:

LONG_W
117.2465

10 January 2023 13:17

Problem Statement: A median is defined as a number separating the higher half of a data set from the lower half. Query the *median* of the *Northern Latitudes* (*LAT_N*) from **STATION** and round your answer to decimal places.

Input Format

The **STATION** table is described as follows:

STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where *LAT_N* is the northern latitude and *LONG_W* is the western longitude.

Mental Approach:

1. We will sort the LAT_N from smallest to largest
2. Now we will check whether count of LAT_N is odd or even.
3. If it is odd then we will go the middle value that will be our median.
4. If it is even then we need to find the average of middle two values.

Generic Method:

```
WITH rank_cte AS (
SELECT LAT_N
      ,ROW_NUMBER() OVER (ORDER BY LAT_N ASC) rn_asc
      ,ROW_NUMBER() OVER (ORDER BY LAT_N DESC) rn_desc
FROM STATION
)
SELECT CAST(1.0*AVG(LAT_N)AS DECIMAL(15,4)) AS  median
FROM rank_cte
WHERE ABS(rn_asc-rn_desc) <= 1
GROUP BY LAT_N
ORDER BY LAT_N ASC
```

Steps:

1. First Lat_N will be sorted in ascending order.
2. A column with rn_asc will be added which gives row numbers in ascending order.
3. Another column with rn_desc will added which gives row numbers in descending order.
4. This above part is done within CTE.
5. So, after this we will query for the AVG of LAT_N with a condition where absolute difference between rn_asc and rn_desc

   will be less than or equal to 1.

   Here, 5th step is very important. Basically, middle row number for both rn_asc and rn_desc will match if there is an odd number of records.

   Else they will be having difference of either 1 or -1 when there are an even number of records.

Sample Output of Query within CTE:

LAT_N	rn_asc	rn_desc
83.12257286	247	253
83.27433063	248	252
83.49946581	249	251
83.89130493	250	250
83.92116818	251	249
84.78749012	252	248

Overall Output:

median
83.8913

14 January 2023 19:51

Problem Statement: Find the titles of workers that earn the highest salary. Output the highest-paid title or multiple titles that share the highest salary.

worker table

Field	Type
worker_id	int
first_name	varchar
last_name	varchar
salary	int
joining_date	datetime
department	varchar

title table

Field	Type
worker_ref_id	int
worker_title	varchar
affected_from	datetime

Table

worker table

worker_id	first_name	last_name	salary	joining_date	department
1	Monika	Arora	100000	20-02-2014 09:00	HR
2	Niharika	Verma	80000	11-06-2014 09:00	Admin
3	Vishal	Singhal	300000	20-02-2014 09:00	HR
4	Amitah	Singh	500000	20-02-2014 09:00	Admin
5	Vivek	Bhati	500000	11-06-2014 09:00	Admin
6	Vipul	Diwan	200000	11-06-2014 09:00	Account
7	Satish	Kumar	75000	20-01-2014 09:00	Account
8	Geetika	Chauhan	90000	11-04-2014 09:00	Admin

title table

worker_ref_id	worker_title	affected_from
1	Manager	20-02-2016
2	Executive	11-06-2016
8	Executive	11-06-2016
5	Manager	11-06-2016
4	Asst. Manager	11-06-2016
7	Executive	11-06-2016
6	Lead	11-06-2016
3	Lead	11-06-2016

Mental Approach:

1. First combine both table together. (In SQL we do this by using JOINS)
2. Now look for highest salary in the table. (In SQL we do this using aggregate function MAX())
3. Now corresponding to that highest salary search for worker_title. (For this we will just query for worker_title in SELECT statement)

Query:

```
SELECT t.worker_title
FROM worker w
LEFT JOIN title t
ON w.worker_id = t.worker_ref_id
WHERE w.salary = (SELECT MAX(salary) FROM worker);
```

Steps followed by the above query:

1. First of all FROM and JOIN will happen on the basis of worker_id from the worker table and worker_ref_id from the title table.
2. Now it will move to WHERE clause. Here it will filter out and it will only keep those salaries which is the maximum salary of all the workers.
3. In the WHERE clause we are using a subquery to filter for maximum salary.
4. Last step is that the SELECT statement will be executed and it will print the worker_title for those worker/workers which is/are having maximum salary of all.

Output of Subquery that is present in WHERE clause

max_salary
500000

Final Output:

worker_title
Asst. Manager
Manager