

Project8 Problem Solving SQL with Various Company Cases

Case 1 – Technology Company Case: Finding the Most Popular Client_Id Among Users Using Video and Voice Calls

Task

Find and select the most popular client_id based on a count of the number of users who have at least 50% of their events from the following list:

- video call received
- video call sent
- voice call received
- voice call sent

Data

Table name = fact_events

id:	int
time_id:	datetime
user_id:	varchar
customer_id:	varchar
client_id:	varchar
event_type:	varchar
event_id:	int

Sample data for table fact_events:

id	time_id	user_id	customer_id	client_id	event_type	event_id
1	2020-02-28	3668-QPYBK	Sendit	desktop	message sent	3
2	2020-02-28	7892-POOKP	Connectix	mobile	file received	2
3	2020-04-03	9763-GRSKD	Zoomit	desktop	video call received	7
4	2020-04-02	9763-GRSKD	Connectix	desktop	video call received	7
5	2020-02-06	9237-HQITU	Sendit	desktop	video call received	7

Solution

MS SQL

```
with user_criteria_percentage as (  
    select  
        user_id,  
        100 * sum (CASE WHEN event_type = 'video call received'  
                    or event_type = 'video call sent'  
                    or event_type = 'voice call received'  
                    or event_type = 'voice call sent'  
                    THEN 1  
                    ELSE 0  
                END )/count(*) as criteria_percentage  
    from fact_events  
    group by user_id)  
  
select client_id,  
       count(client_id)  
    from fact_events  
   where user_id in (select user_id from user_criteria_percentage  
                     where criteria_percentage >= 50)  
   group by client_id
```

100 %

Results Messages

	client_id	(No column name)
1	desktop	11
2	mobile	3

- first thing to do in this solution is to get the percentage for each user_id. I'm using count with conditional, in query the method is using SUM function with CASE WHEN with BOOLEAN result. And I store this in CTE.
- Next, after we got the percentage, we can filter user_id with WHERE clause. Store this in subquery
- Last, I take the client_id and count of it and filter it by subquery we created on previous step

Case 2 – Online Retailing Company Case: Monthly Percentage Difference

Task

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.

Data

Table name = sf_transactions

id:	int
created_at:	datetime
value:	int
purchase_id:	int

Sample data for table sf_transactions

id	created_at	value	purchase_id
1	2019-01-01	172692	43
2	2019-01-05	177194	36
3	2019-01-09	109513	30
4	2019-01-13	164911	30
5	2019-01-17	198872	39

Solution

```
WITH t1 as(
  SELECT
    FORMAT(created_at, 'yyyy-MM') month_date,
    CAST(SUM(value) as DECIMAL(10,2)) this_month_revenue,
    CAST(LAG(SUM(value),1,0) over (order by FORMAT(created_at, 'yyyy-MM'))as DECIMAL(10,2)) last_month_revenue
  from
    sf_transactions
  group by FORMAT(created_at, 'yyyy-MM')
)

select
  month_date,
  CAST(100*(this_month_revenue-last_month_revenue)/NULLIF(last_month_revenue,0) AS DECIMAL (10,2))
from t1
```

Results		Messages
	month_date	(No column name)
1	2019-01	NULL
2	2019-02	-28.56
3	2019-03	23.35
4	2019-04	-13.84
5	2019-05	13.49
6	2019-06	-2.78
7	2019-07	-6.00
8	2019-08	28.36
9	2019-09	-4.97
10	2019-10	-12.68

Case 3 – Education Institution Case: Consecutive days learning

Task

Company conducted a days of learning SQL contest. The start date of the contest was March 01, 2016 and the end date was March 15, 2016.

Write a query to print total number of unique hackers who made at least submission each day (starting on the first day of the contest), and find the hacker_id and name of the hacker who made maximum number of submissions each day. If more than one such hacker has a maximum number of submissions, print the lowest hacker_id. The query should print this information for each day of the contest, sorted by the date.

Expected output from sample:

```
2016-03-01 4 20703 Angela
2016-03-02 2 79722 Michael
2016-03-03 2 20703 Angela
2016-03-04 2 20703 Angela
2016-03-05 1 36396 Frank
2016-03-06 1 20703 Angela
```

Data

Table1 = Hackers

Column	Type
hacker_id	Integer
name	String

hacker_id	name
15758	Rose
20703	Angela
36396	Frank
38289	Patrick
44065	Lisa
53473	Kimberly
62529	Bonnie
79722	Michael

Table2 = Submissions

Column	Type
submission_date	Date
submission_id	Integer
hacker_id	Integer
score	Integer

submission_date	submission_id	hacker_id	score
2016-03-01	8494	20703	0
2016-03-01	22403	53473	15
2016-03-01	23965	79722	60
2016-03-01	30173	36396	70
2016-03-02	34928	20703	0
2016-03-02	38740	15758	60
2016-03-02	42769	79722	25
2016-03-02	44364	79722	60
2016-03-03	45440	20703	0
2016-03-03	49050	36396	70
2016-03-03	50273	79722	5
2016-03-04	50344	20703	0
2016-03-04	51360	44065	90
2016-03-04	54404	53473	65
2016-03-04	61533	79722	45

Solution

```
SQLQuery1.sql - A...VOBOOK\renal (66))* ✕
with DailyCountbyHacker as( select submission_date, hacker_id,
COUNT(*) OVER (partition by submission_date, hacker_id) AS HackerSubmissionCntByDay
from submissions),

rolling as (
SELECT *,
COUNT(*) OVER(PARTITION BY hacker_id ORDER BY submission_date ) AS HackerRollingCnt
FROM DailyCountbyHacker
group by hacker_id,submission_date,HackerSubmissionCntByDay),

rolling2 as (
SELECT *, DATEDIFF(day,'2016-03-01',submission_date)+1 as days
FROM Rolling),

rolling_filtered as (select * from rolling2
where HackerRollingCnt = days),

ranks as(
SELECT *,
ROW_NUMBER() OVER (PARTITION BY submission_date ORDER BY HackerSubmissionCntByDay DESC, hacker_id) AS RowNo
FROM rolling),
```

```

ranks_filtered as (
    SELECT *
    FROM Ranks
    where RowNo=1
),

countperday as(
select *,count(hackerssubmissioncntbyday) over (partition by submission_date) as counthackerperday
from rolling_filtered),

final as (
select r.submission_date, c.counthackerperday,r.hacker_id,
row_number () over (partition by r.submission_date order by r.submission_date) rownumfinal from
ranks_filtered r join countperday c on (r.submission_date=c.submission_date)
)

select submission_date,counthackerperday, h.hacker_id,name
from final f join hackers h
on (f.hacker_id=h.hacker_id)
where rownumfinal = 1
order by submission_date

```

Case 4 – Skin Care Online Store Case: Product Without Sales

Task

Given the product and invoice details for products at a skin care store, find all the products that were not sold. For each such product, display its SKU and product name. Order the result by SKU, ascending

Data

product				
id	sku	product_name	product_description	current_p
1	330120	Game Of Thrones - URBAN DECAY	Game Of Thrones Eyeshadow Palette	65
2	330121	Advanced Night Repair - ESTÉE LAUDER	Advanced Night Repair Synchronized Recovery Complex II	98
3	330122	Rose Deep Hydration - FRESH	Rose Deep Hydration Facial Toner	45
4	330123	Pore-Perfecting Moisturizer - TATCHA	Pore-Perfecting Moisturizer & Cleanser Duo	25
5	330124	Capture Youth - DIOR	Capture Youth Serum Collection	95
6	330125	Slice of Glow - GLOW RECIPE	Slice of Glow Set	45
7	330126	Healthv Skin - KIEHL'S SINCE 1851	Healthv Skin Squad	68

Case 5 – Facebook Case: Popularity Percentage

Task

Find the popularity percentage for each user on Meta/Facebook. The popularity percentage is defined as the total number of friends the user has divided by the total number of users on the platform, then converted into a percentage by multiplying by 100.

Output each user along with their popularity percentage. Order records in ascending order by user id.

The 'user1' and 'user2' column are pairs of friends.

Data

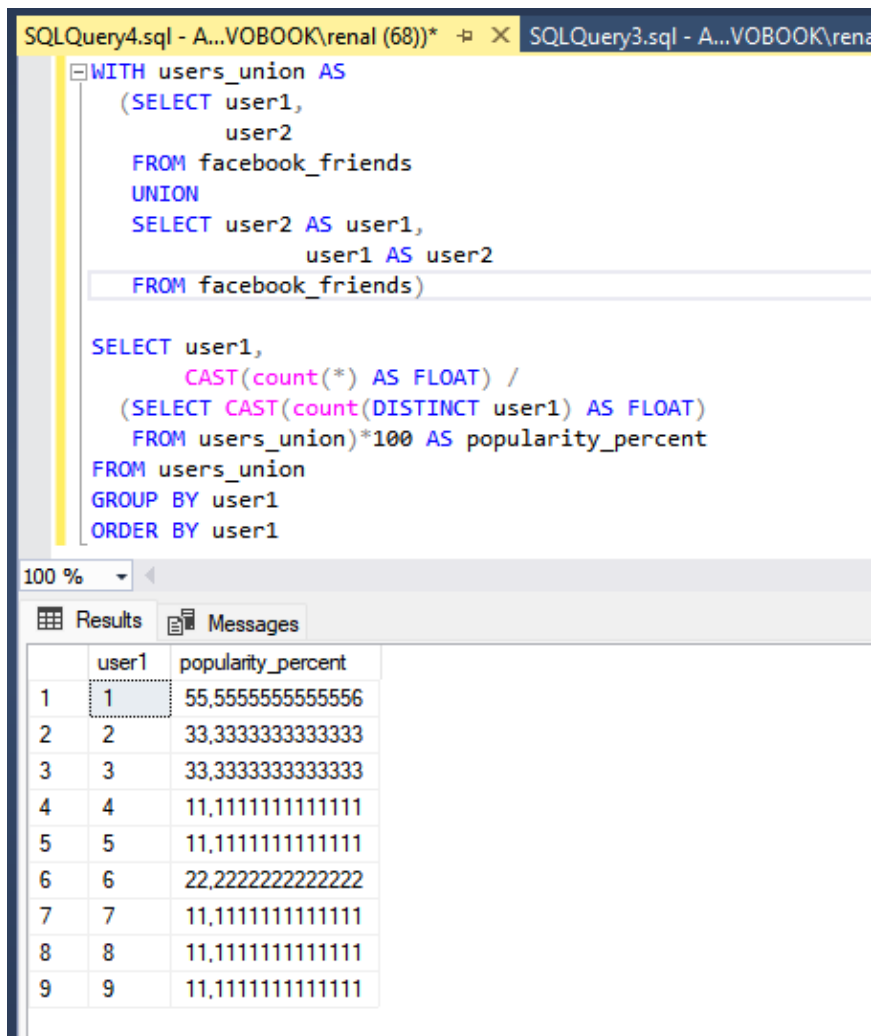
Table name = facebook_friends

user1: int

user2: int

user1	user2
2	1
1	3
4	1
1	5
1	6

Solution



The screenshot shows a SQL Query Editor window with two tabs: 'SQLQuery4.sql' and 'SQLQuery3.sql'. The active tab 'SQLQuery4.sql' contains the following SQL query:

```
WITH users_union AS
(
    SELECT user1,
           user2
    FROM facebook_friends
    UNION
    SELECT user2 AS user1,
           user1 AS user2
    FROM facebook_friends
)

SELECT user1,
       CAST(count(*) AS FLOAT) /
       (SELECT CAST(count(DISTINCT user1) AS FLOAT)
        FROM users_union)*100 AS popularity_percent
FROM users_union
GROUP BY user1
ORDER BY user1
```

Below the query editor, the 'Results' tab is selected, displaying a table with 9 rows and 2 columns: 'user1' and 'popularity_percent'.

	user1	popularity_percent
1	1	55,55555555555556
2	2	33,33333333333333
3	3	33,33333333333333
4	4	11,11111111111111
5	5	11,11111111111111
6	6	22,22222222222222
7	7	11,11111111111111
8	8	11,11111111111111
9	9	11,11111111111111

Case 6 – Amazon Case: Finding User Purchases

Task

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.

Data

Table name = amazon_transactions

id:	int
user_id:	int
item:	varchar
created_at:	datetime
revenue:	int

id	user_id	item	created_at	revenue
1	109	milk	2020-03-03	123
2	139	biscuit	2020-03-18	421
3	120	milk	2020-03-18	176
4	108	banana	2020-03-18	862
5	130	milk	2020-03-28	333

Solution

```
1 select distinct a1.user_id
2 from amazon_transactions as a1 join amazon_transactions as a2
3 on
4     a1.user_id = a2.user_id
5     and
6     a1.id <> a2.id
7     and
8     a1.created_at >= a2.created_at
9 where datediff(a1.created_at, a2.created_at) <= 7
```

I will explain every step how SELF JOIN filter the table for further understanding:

t1					t2				
id	user_id	item	created_at	revenue	id	user_id	item	created_at	revenue
20	100	banana	#####	599	20	100	banana	#####	599
29	100	milk	#####	410	29	100	milk	#####	410
74	100	banana	#####	175	74	100	banana	#####	175
95	100	bread	#####	410	95	100	bread	#####	410
27	101	milk	#####	449	27	101	milk	#####	449
65	101	milk	#####	740	65	101	milk	#####	740
24	102	bread	#####	325	24	102	bread	#####	325
6	103	bread	#####	862	6	103	bread	#####	862
22	103	milk	#####	290	22	103	milk	#####	290
32	104	biscuit	#####	957	32	104	biscuit	#####	957
58	105	bread	#####	562	58	105	bread	#####	562
75	105	banana	#####	815	75	105	banana	#####	815
82	105	banana	#####	972	82	105	banana	#####	972
96	106	milk	#####	379	96	106	milk	#####	379

join on t1.user_id=t2.user_id									
id	user_id	item	created_at	revenue	id	user_id	item	created_at	revenue
95	100	bread	#####	410	20	100	banana	#####	599
74	100	banana	#####	175	20	100	banana	#####	599
29	100	milk	#####	410	20	100	banana	#####	599
20	100	banana	#####	599	20	100	banana	#####	599
95	100	bread	#####	410	29	100	milk	#####	410
74	100	banana	#####	175	29	100	milk	#####	410
29	100	milk	#####	410	29	100	milk	#####	410
20	100	banana	#####	599	29	100	milk	#####	410
95	100	bread	#####	410	74	100	banana	#####	175
74	100	banana	#####	175	74	100	banana	#####	175
29	100	milk	#####	410	74	100	banana	#####	175
20	100	banana	#####	599	74	100	banana	#####	175
95	100	bread	#####	410	95	100	bread	#####	410
74	100	banana	#####	175	95	100	bread	#####	410
29	100	milk	#####	410	95	100	bread	#####	410
20	100	banana	#####	599	95	100	bread	#####	410
65	101	milk	#####	740	27	101	milk	#####	449
27	101	milk	#####	449	27	101	milk	#####	449

t1.id <> t2.id									
id	user_id	item	created_at	revenue	id	user_id	item	created_at	revenue
95	100	bread	#####	410	20	100	banana	#####	599
74	100	banana	#####	175	20	100	banana	#####	599
29	100	milk	#####	410	20	100	banana	#####	599
20	100	banana	#####	599	20	100	banana	#####	599
95	100	bread	#####	410	29	100	milk	#####	410
74	100	banana	#####	175	29	100	milk	#####	410
29	100	milk	#####	410	29	100	milk	#####	410
20	100	banana	#####	599	29	100	milk	#####	410
95	100	bread	#####	410	74	100	banana	#####	175
74	100	banana	#####	175	74	100	banana	#####	175
29	100	milk	#####	410	74	100	banana	#####	175
20	100	banana	#####	599	74	100	banana	#####	175
95	100	bread	#####	410	95	100	bread	#####	410
74	100	banana	#####	175	95	100	bread	#####	410
29	100	milk	#####	410	95	100	bread	#####	410
20	100	banana	#####	599	95	100	bread	#####	410

t1.id <> t2.id									
id	user_id	item	created_at	revenue	id	user_id	item	created_at	revenue
95	100	bread	07/03/2020 00.00	410	20	100	banana	18/03/2020 00.00	599
74	100	banana	13/03/2020 00.00	175	20	100	banana	18/03/2020 00.00	599
29	100	milk	29/03/2020 00.00	410	20	100	banana	18/03/2020 00.00	599
95	100	bread	07/03/2020 00.00	410	29	100	milk	29/03/2020 00.00	410
74	100	banana	13/03/2020 00.00	175	29	100	milk	29/03/2020 00.00	410
20	100	banana	18/03/2020 00.00	599	29	100	milk	29/03/2020 00.00	410
95	100	bread	07/03/2020 00.00	410	74	100	banana	13/03/2020 00.00	175
29	100	milk	29/03/2020 00.00	410	74	100	banana	13/03/2020 00.00	175
20	100	banana	18/03/2020 00.00	599	74	100	banana	13/03/2020 00.00	175
74	100	banana	13/03/2020 00.00	175	95	100	bread	07/03/2020 00.00	410
29	100	milk	29/03/2020 00.00	410	95	100	bread	07/03/2020 00.00	410
20	100	banana	18/03/2020 00.00	599	95	100	bread	07/03/2020 00.00	410

t1.created_at >= t2.created_at									
id	user_id	item	created_at	revenue	id	user_id	item	created_at	revenue
95	100	bread	07/03/2020 00.00	410	20	100	banana	18/03/2020 00.00	599
74	100	banana	13/03/2020 00.00	175	20	100	banana	18/03/2020 00.00	599
29	100	milk	29/03/2020 00.00	410	20	100	banana	18/03/2020 00.00	599
95	100	bread	07/03/2020 00.00	410	29	100	milk	29/03/2020 00.00	410
74	100	banana	13/03/2020 00.00	175	29	100	milk	29/03/2020 00.00	410
20	100	banana	18/03/2020 00.00	599	29	100	milk	29/03/2020 00.00	410
95	100	bread	07/03/2020 00.00	410	74	100	banana	13/03/2020 00.00	175
29	100	milk	29/03/2020 00.00	410	74	100	banana	13/03/2020 00.00	175
20	100	banana	18/03/2020 00.00	599	74	100	banana	13/03/2020 00.00	175
74	100	banana	13/03/2020 00.00	175	95	100	bread	07/03/2020 00.00	410
29	100	milk	29/03/2020 00.00	410	95	100	bread	07/03/2020 00.00	410
20	100	banana	18/03/2020 00.00	599	95	100	bread	07/03/2020 00.00	410

t1.created_at >= t2.created_at									
id	user_id	item	created_at	revenue	id	user_id	item	created_at	revenue
29	100	milk	29/03/2020 00.00	410	20	100	banana	18/03/2020 00.00	599
29	100	milk	29/03/2020 00.00	410	74	100	banana	13/03/2020 00.00	175
20	100	banana	18/03/2020 00.00	599	74	100	banana	13/03/2020 00.00	175
74	100	banana	13/03/2020 00.00	175	95	100	bread	07/03/2020 00.00	410
29	100	milk	29/03/2020 00.00	410	95	100	bread	07/03/2020 00.00	410
20	100	banana	18/03/2020 00.00	599	95	100	bread	07/03/2020 00.00	410

Reference Code

Case 1 : SS_2029

Case 2 : SS_10319

Case 3 : HR_ADV

Case 4 : HR_CERT

Case 5 : SS_10284

Case 6 : SS_10322