Operation Analytics and Investigating Metric Spike

Advanced SQL

Project Description:

Operation Analytics for Enhanced Business Performance

As the Data Analyst Lead at Microsoft, I am responsible for conducting comprehensive analysis of the end-to-end operations of the company. Operation Analytics plays a crucial role in identifying areas for improvement and guiding strategic decision-making. By closely collaborating with various teams such as operations, support, and marketing, I leverage the data they collect to derive valuable insights.

The primary objective of Operation Analytics is to enable the company to enhance its overall performance and predict its future growth or decline. Through this analysis, I aim to achieve better automation, foster better understanding among cross-functional teams, and optimize workflows for increased efficiency.

One significant aspect of my role is investigating metric spikes. As a Data Analyst, I possess the expertise to understand and help others comprehend why certain metrics may experience fluctuations. For instance, I address questions such as: Why is there a decrease in daily engagement? What factors have led to a decline in sales? Answering such critical inquiries on a daily basis is vital, and investigating metric spikes is an essential component of Operation Analytics.

In my position, I am provided with diverse data sets and tables, which serve as the foundation for deriving meaningful insights. By analyzing these datasets, I can uncover valuable patterns, trends, and correlations that contribute to a deeper understanding of the company's performance. This enables me to provide actionable recommendations to different departments based on data-driven insights.

Ultimately, my role as the Data Analyst Lead at Microsoft entails utilizing Operation Analytics to empower the company to make informed decisions, optimize processes, and drive continuous improvement. By harnessing the power of data, i can steer the company towards sustained growth and success.

Objectives/ Approach:

Identify Improvement Areas: Analyze end-to-end operations to identify bottlenecks, inefficiencies, and areas for improvement. Provide actionable recommendations to optimize processes, reduce costs, and enhance overall operational performance.

Metric Investigation: Investigate metric spikes and fluctuations to understand their underlying causes. Provide explanations for changes in metrics such as daily engagement, sales, or customer satisfaction. Identify actionable steps to address any negative impacts and leverage positive trends.

Cross-Functional Collaboration: Facilitate effective collaboration and information sharing between different departments by providing insights and analytics support. Foster a data-driven culture and encourage teams to leverage analytics in their decision-making processes.

Automation and Workflow Efficiency: Identify opportunities for automation and process optimization to streamline workflows, improve productivity, and reduce manual efforts. Implement data-driven solutions to enhance operational efficiency and resource allocation.

Performance Monitoring and Reporting: Establish a system for ongoing monitoring and reporting of key metrics and KPIs. Develop regular reports and dashboards that provide real-time insights into operational performance. Enable stakeholders to track progress, identify deviations, and take timely corrective actions.

Tech-Stack Used:

In the Operation Analytics project, I utilize both MySQL and PostgreSQL as our chosen database management systems. MySQL, an open-source RDBMS, provides a robust platform for storing and organizing structured operational data, enabling seamless integration with various systems and efficient querying using SQL. On the other hand, PostgreSQL, renowned for its advanced features and extensibility, empowers us to perform complex analytics, leverage flexible data modeling capabilities, and ensure the security of our operational data. By combining the strengths of MySQL and PostgreSQL, I create a comprehensive tech-stack that supports efficient data storage, retrieval, analysis, and reporting for our operation analytics needs.

Understanding database:

Using ‘job_data’ database:

#	Time	Action	Message
1	17:06:31	use job_data	0 row(s) affected

Creating 'job_data' table

```
create table job_data(
job_id int,
actor_id int,
event varchar(50),
language varchar(50),
time_spent int,
org char(1),
ds date);

select * from job_data;
```

```
insert into job_data (ds, job_id, actor_id, event, language, time_spent, org) values
('2020-11-30', 21, 1001, 'skip', 'English', 15, 'A'),
('2020-11-30', 22, 1006, 'transfer', 'Arabic', 25, 'B'),
('2020-11-29', 23, 1003, 'decision', 'Persian', 20, 'C'),
('2020-11-28', 23, 1005,'transfer', 'Persian', 22, 'D'),
('2020-11-28', 25, 1002, 'decision', 'Hindi', 11, 'B'),
('2020-11-27', 11, 1007, 'decision', 'French', 104, 'D'),
('2020-11-26', 23, 1004, 'skip', 'Persian', 56, 'A'),
('2020-11-25', 20, 1003, 'transfer', 'Italian', 45, 'C');
```

Job data table: 8 rows and 7 columns.

job_id	actor_id	event	language	time_spent	org	ds
21	1001	skip	English	15	A	2020-11-30
22	1006	transfer	Arabic	25	B	2020-11-30
23	1003	decision	Persian	20	C	2020-11-29
23	1005	transfer	Persian	22	D	2020-11-28
25	1002	decision	Hindi	11	B	2020-11-28
11	1007	decision	French	104	D	2020-11-27
23	1004	skip	Persian	56	A	2020-11-26
20	1003	transfer	Italian	45	C	2020-11-25

Report:

Case Study 1 (Job Data):

A. Number of jobs reviewed: Amount of jobs reviewed over time.

My task: Calculate the number of jobs reviewed per hour per day for November 2020?

```
#A
select ds, count(job_id),sum(time_spent),
(count(job_id)*60*60/sum(time_spent)) as "Jobs Reviewed per Hour"
from job_data
where ds between '2020-11-01' and '2020-11-30'
group by ds
order by ds;
```

ds	count(job_id)	sum(time_spent)	Jobs Reviewed per Hour
2020-11-25	1	45	80.0000
2020-11-26	1	56	64.2857
2020-11-27	1	104	34.6154
2020-11-28	2	33	218.1818
2020-11-29	1	20	180.0000
2020-11-30	2	40	180.0000

The SQL code performs a query on the "job_data" table to calculate the number of jobs reviewed per hour for each day within the date range from November 1, 2020, to November 30, 2020. Here's an explanation of the code:

SELECT statement:

ds: This selects the "ds" column from the "job_data" table.

COUNT(job_id): This counts the number of occurrences of the "job_id" column.

SUM(time_spent): This calculates the sum of the values in the "time_spent" column.

Expression:

(COUNT(job_id) * 60 * 60 / SUM(time_spent)): This expression calculates the jobs reviewed per hour by dividing the count of job_ids by the sum of time_spent, and then multiplying it by 60 (minutes) and 60 (seconds) to convert it to jobs per hour.

FROM clause:

job_data: This specifies the table from which the data is retrieved.

WHERE clause:

ds BETWEEN '2020-11-01' AND '2020-11-30': This filters the data based on the "ds" column, selecting only the rows where the "ds" value falls within the specified date range.

GROUP BY clause:

ds: This groups the data by the "ds" column, which represents the day.

ORDER BY clause:

ds: This orders the result set in ascending order based on the "ds" column.

B. Throughput: It is the no. of events happening per second.

My task: Let's say the above metric is called throughput. Calculate 7 day rolling average of throughput? For throughput, do you prefer daily metric or 7-day rolling and why?

```
#B
select count(event),sum(time_spent),
round((count(event)/sum(time_spent)),3) as "7 day rolling throughput "
from job_data;
```

	count(event)	sum(time_spent)	7 day rolling throughput
▶	8	298	0.027

This is 7 day rolling average of throughput.

```
select ds, round(count(event)/sum(time_spent), 3) as "Daily matric throughput"
from job_data
group by ds
order by ds;
```

	ds	Daily matric throughput
▶	2020-11-25	0.022
	2020-11-26	0.018
	2020-11-27	0.010
	2020-11-28	0.061
	2020-11-29	0.050
	2020-11-30	0.050

This is daily metric throughput.

Regarding the preference between daily metric and 7-day rolling average for throughput, it depends on the specific context and objectives of the analysis. Here are considerations for both options:

Daily Metric:

Advantage: Daily metric provides a granular view of throughput on a day-to-day basis. It helps in identifying short-term trends, patterns, and variations in the number of events happening per second.

Use case: If you need to monitor and analyze the throughput in real-time, detect sudden changes, or assess daily performance fluctuations, the daily metric is more suitable.

7-day Rolling Average:

Advantage: The 7-day rolling average provides a smoothed representation of throughput, reducing the impact of short-term fluctuations and offering a more stable trend over a longer period.

Use case: If your focus is on capturing the overall trend and identifying long-term changes in throughput, the 7-day rolling average is preferable. It helps to smoothen out noise or anomalies caused by daily variations and provides a more reliable measure of performance over time.

The choice between daily metric and 7-day rolling average depends on the specific requirements above mentioned.

The SQL code performs a query on the "job_data" table to calculate the 7-day rolling throughput. Here's an explanation of the code:

SELECT statement:

COUNT(event): This counts the number of occurrences of the "event" column.

SUM(time_spent): This calculates the sum of the values in the "time_spent" column.

Expression:

ROUND((COUNT(event) / SUM(time_spent)), 3): This expression calculates the 7-day rolling throughput by dividing the count of events by the sum of time_spent. The result is rounded to three decimal places.

FROM clause:

job_data: This specifies the table from which the data is retrieved.

The code calculates the count of events and the sum of time_spent for all records in the "job_data" table. It then calculates the 7-day rolling throughput by dividing the count of events by the sum of time_spent. The result is rounded to three decimal places.

The SQL code performs a query on the "job_data" table to calculate the daily metric throughput. Here's an explanation of the code:

SELECT statement:

ds: This selects the "ds" column from the "job_data" table.

ROUND(COUNT(event) / SUM(time_spent), 3): This expression calculates the daily metric throughput by dividing the count of events by the sum of time_spent. The result is rounded to three decimal places.

FROM clause:

job_data: This specifies the table from which the data is retrieved.

GROUP BY clause:

ds: This groups the data by the "ds" column, which represents the day.

ORDER BY clause:

ds: This orders the result set in ascending order based on the "ds" column.

The code calculates the count of events and the sum of time_spent for each day in the "job_data" table. It then calculates the daily metric throughput by dividing the count of events by the sum of time_spent. The result is rounded to three decimal places.

C. Percentage share of each language: Share of each language for different contents.

My task: Calculate the percentage share of each language in the last 30 days?

```
#C
select language, round(count(*)*100/8,3) as percentage
from job_data
group by language;
```

language	percentage
English	12.500
Arabic	12.500
Persian	37.500
Hindi	12.500
French	12.500
Italian	12.500

The SQL code performs a query on the "job_data" table to calculate the percentage distribution of languages. Here's an explanation of the code:

SELECT statement:

language: This selects the "language" column from the "job_data" table.

ROUND(COUNT() * 100 / 8, 3): This expression calculates the percentage distribution by dividing the count of rows (represented by COUNT()) by the total number of rows considered (in this case, 8). The result is multiplied by 100 to obtain a percentage value. It is then rounded to three decimal places.

FROM clause:

job_data: This specifies the table from which the data is retrieved.

GROUP BY clause:

language: This groups the data by the "language" column.

The code calculates the count of occurrences for each language in the "job_data" table. It then calculates the percentage distribution by dividing the count of rows by the total number of rows (8 in this case) and multiplying it by 100. The result is rounded to three decimal places.

D. Duplicate rows: Rows that have the same value present in them.

My task: Let's say you see some duplicate rows in the data. How will you display duplicates from the table?

```
select job_id, count(job_id) as repeated
from job_data
group by job_id
having count(job_id)>1;
```

job_id	repeated
23	3

```sql
select actor_id, count(actor_id) as repeated
from job_data
group by actor_id
having count(actor_id)>1;
```

	actor_id	repeated
▶	1003	2

Like this we can able to do for other variables in table job_data.

The SQL code performs a query on the "job_data" table to calculate the percentage distribution of languages. Here's an explanation of the code:

SELECT statement:

language: This selects the "language" column from the "job_data" table.

ROUND(COUNT() * 100 / 8, 3): This expression calculates the percentage distribution by dividing the count of rows (represented by COUNT()) by the total number of rows considered (in this case, 8). The result is multiplied by 100 to obtain a percentage value. It is then rounded to three decimal places.

FROM clause:

job_data: This specifies the table from which the data is retrieved.

GROUP BY clause:

language: This groups the data by the "language" column.

The code calculates the count of occurrences for each language in the "job_data" table. It then calculates the percentage distribution by dividing the count of rows by the total number of rows (8 in this case) and multiplying it by 100. The result is rounded to three decimal places.

Case Study 2 (Investigating metric spike)

Creating and understanding tables:

Table-1: users

I faced some difficult in import table1 users table in MySQL, so used PostgreSQL.

```sql
create table users(
user_id int,
created_at timestamp,
company_id int,
language varchar(50),
activated_at timestamp,
state varchar(50));
```

```sql
COPY users(user_id, created_at, company_id, language, activated_at, state)
FROM 'C:\Program Files\PostgreSQL\15\Table-1 users'
DELIMITER ','
CSV HEADER;

select * from users;
```

There are 6 columns and 19066 rows in this table:

	user_id integer	created_at timestamp without time zone	company_id integer	language character varying (50)	activated_at timestamp without time zone	state character varying (50)
1	0	2013-01-01 20:59:39	5737	english	2013-01-01 21:01:07	active
2	1	2013-01-01 13:07:46	28	english	[null]	pending
3	2	2013-01-01 10:59:05	51	english	[null]	pending
4	3	2013-01-01 18:40:36	2800	german	2013-01-01 18:42:02	active
5	4	2013-01-01 14:37:51	5110	indian	2013-01-01 14:39:05	active
6	5	2013-01-01 13:39:51	2463	spanish	[null]	pending
7	6	2013-01-01 18:37:27	11699	english	2013-01-01 18:38:45	active
8	7	2013-01-01 16:19:01	4765	french	2013-01-01 16:20:28	active

Table-2: events

```sql
create table events(
user_id int,
occurred_at timestamp,
event_type varchar(50),
event_name varchar(50),
location varchar(50),
device varchar(150),
user_type varchar(50));

select *from events;

LOAD DATA INFILE 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\Table-2 events.csv'
INTO TABLE events
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 LINES;
```

There are 7 columns and 340832 rows in tis table:

user_id	occurred_at	event_type	event_name	location	device	user_type
10522	2014-05-02 11:02:39	engagement	login	Japan	dell inspiron notebook	3.0
10522	2014-05-02 11:02:53	engagement	home_page	Japan	dell inspiron notebook	3.0
10522	2014-05-02 11:03:28	engagement	like_message	Japan	dell inspiron notebook	3.0
10522	2014-05-02 11:04:09	engagement	view_inbox	Japan	dell inspiron notebook	3.0
10522	2014-05-02 11:03:16	engagement	search_run	Japan	dell inspiron notebook	3.0
10522	2014-05-02 11:03:43	engagement	search_run	Japan	dell inspiron notebook	3.0
10612	2014-05-01 09:59:46	engagement	login	Netherlands	iphone 5	1.0
10612	2014-05-01 10:00:18	engagement	like_message	Netherlands	iphone 5	1.0
10612	2014-05-01 10:00:53	engagement	send_message	Netherlands	iphone 5	1.0
10612	2014-05-01 10:01:24	engagement	home_page	Netherlands	iphone 5	1.0

Table-3: email_events

```
create table email_events(
user_id int,
occurred_at timestamp,
action varchar(50),
user_type int);

LOAD DATA INFILE 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\Table-3 email_events.csv'
INTO TABLE email_events
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 LINES;

select *from email_events;
```

There are 4 columns and 90389 rows in this table:

user_id	occurred_at	action	user_type
0	2014-05-06 09:30:00	sent_weekly_digest	1
0	2014-05-13 09:30:00	sent_weekly_digest	1
0	2014-05-20 09:30:00	sent_weekly_digest	1
0	2014-05-27 09:30:00	sent_weekly_digest	1
0	2014-06-03 09:30:00	sent_weekly_digest	1
0	2014-06-03 09:30:25	email_open	1
0	2014-06-10 09:30:00	sent_weekly_digest	1
0	2014-06-10 09:30:24	email_open	1
0	2014-06-17 09:30:00	sent_weekly_digest	1
0	2014-06-17 09:30:23	email_open	1

Report:

A. User Engagement: To measure the activeness of a user. Measuring if the user finds quality in a product/service.

Your task: Calculate the weekly user engagement?

```
#A User Engagement
select count(distinct user_id) as no_of_user, extract(week from occurred_at) as weeks from events
where event_type = 'engagement'
group by weeks;
```

no_of_user	weeks
663	17
1068	18
1113	19
1154	20
1121	21
1186	22
1232	23
1275	24

1264	25
1302	26
1372	27
1365	28
1376	29
1467	30
1299	31
1225	32
1225	33
1204	34
104	35

from table 2

The SQL code performs a query on the "events" table to calculate the count of distinct user IDs and the week number extracted from the "occurred_at" column for events of type 'engagement'. Here's an explanation of the code:

SELECT statement:

COUNT(DISTINCT user_id) as no_of_user: This calculates the count of distinct user IDs and assigns it the alias "no_of_user".

EXTRACT(WEEK FROM occurred_at) as weeks: This extracts the week number from the "occurred_at" column and assigns it the alias "weeks".

FROM clause:

events: This specifies the table from which the data is retrieved.

WHERE clause:

event_type = 'engagement': This filters the data to include only events with the event_type equal to 'engagement'.

GROUP BY clause:

weeks: This groups the data by the week number extracted from the "occurred_at" column.

The code calculates the count of distinct user IDs for events categorized as 'engagement' in the "events" table. It also extracts the week number from the "occurred_at" column for each event. The results are grouped by week number.

B. User Growth: Amount of users growing over time for a product.

My task: Calculate the user growth for product? (used postgreSQL)

```
#B User Growth
select Years,Weeks, Users,
Users-LAG(Users, 1) OVER (ORDER BY Years)
AS "Growth by comparative weekly",
sum(Users) over(order by Years,Weeks rows between unbounded preceding and current row)
as "Cumulative Growth"
from(
select extract(year from created_at) as Years,
extract(week from created_at) as Weeks,count(activated_at) AS Users
from users
where state='active'
group by Years,Weeks
order by Years, Weeks)a;
```

	years numeric	weeks numeric	users bigint	Growth by comparative weekly bigint	Cumulative Growth numeric
1	2013	1	67	[null]	67
2	2013	2	29	-38	96
3	2013	3	47	18	143
4	2013	4	36	-11	179
5	2013	5	30	-6	209
6	2013	6	48	18	257
7	2013	7	41	-7	298
8	2013	8	39	-2	337
9	2013	9	33	-6	370
10	2013	10	43	10	413
11	2013	11	33	-10	446
12	2013	12	32	-1	478
13	2013	13	33	1	511
14	2013	14	40	7	551
15	2013	15	35	-5	586
16	2013	16	42	7	628
17	2013	17	48	6	676
18	2013	18	48	0	724
19	2013	19	45	-3	769
20	2013	20	55	10	824
21	2013	21	41	-14	865
22	2013	22	49	8	914
23	2013	23	51	2	965
24	2013	24	51	0	1016
25	2013	25	46	-5	1062
26	2013	26	57	11	1119
27	2013	27	57	0	1176
28	2013	28	52	-5	1228
29	2013	29	71	19	1299
30	2013	30	66	-5	1365
31	2013	31	69	3	1434
32	2013	32	66	-3	1500
33	2013	33	73	7	1573
34	2013	34	71	-2	1644
35	2013	35	79	8	1723
36	2013	36	65	-14	1788
37	2013	37	71	6	1859
38	2013	38	84	13	1943
39	2013	39	92	8	2035
40	2013	40	81	-11	2116
41	2013	41	88	7	2204
42	2013	42	74	-14	2278
43	2013	43	97	23	2375
44	2013	44	92	-5	2467
45	2013	45	97	5	2564
46	2013	46	94	-3	2658
47	2013	47	82	-12	2740
48	2013	48	103	21	2843
49	2013	49	96	-7	2939
50	2013	50	117	21	3056
51	2013	51	123	6	3179
52	2013	52	104	-19	3283
53	2014	1	91	-13	3374
54	2014	2	122	31	3496
55	2014	3	112	-10	3608
56	2014	4	113	1	3721
57	2014	5	130	17	3851
58	2014	6	132	2	3983
59	2014	7	135	3	4118
60	2014	8	127	-8	4245
61	2014	9	127	0	4372
62	2014	10	135	8	4507
63	2014	11	152	17	4659
64	2014	12	132	-20	4791
65	2014	13	151	19	4942
66	2014	14	161	10	5103

67	2014	15	166	5	5269
68	2014	16	165	-1	5434
69	2014	17	176	11	5610
70	2014	18	172	-4	5782
71	2014	19	160	-12	5942
72	2014	20	186	26	6128
73	2014	21	177	-9	6305
74	2014	22	186	9	6491
75	2014	23	197	11	6688
76	2014	24	198	1	6886
77	2014	25	222	24	7108
78	2014	26	210	-12	7318
79	2014	27	199	-11	7517
80	2014	28	223	24	7740
81	2014	29	215	-8	7955
82	2014	30	228	13	8183
83	2014	31	234	6	8417
84	2014	32	189	-45	8606
85	2014	33	250	61	8856
86	2014	34	259	9	9115
87	2014	35	266	7	9381

from table1

The SQL code calculates the user growth and cumulative growth over time based on the data from the "users" table. Here's an explanation of the code:

SELECT statement:

extract(year from created_at) as Years: This extracts the year from the "created_at" column and assigns it the alias "Years".

extract(week from created_at) as Weeks: This extracts the week number from the "created_at" column and assigns it the alias "Weeks".

count(activated_at) AS Users: This counts the number of activated users and assigns it the alias "Users".

Users - LAG(Users, 1) OVER (ORDER BY Years): This calculates the growth in users by subtracting the previous week's user count from the current week's user count using the LAG() window function.

sum(Users) over(order by Years,Weeks rows between unbounded preceding and current row) as "Cumulative Growth": This calculates the cumulative growth of users over time using the SUM() window function with a cumulative range from the unbounded preceding row to the current row.

FROM clause:

users: This specifies the table from which the data is retrieved.

WHERE clause:

state = 'active': This filters the data to include only active users.

GROUP BY clause:

Years, Weeks: This groups the data by the extracted year and week.

ORDER BY clause:

Years, Weeks: This orders the result set in ascending order based on the year and week.

The inner subquery calculates the count of active users for each year and week. The outer query then uses window functions to calculate the growth in users compared to the previous week and the cumulative growth of users over time.

C. Weekly Retention: Users getting retained weekly after signing-up for a product.

My task: Calculate the weekly retention of users-sign up cohort?

```
#C Weekly Retention
select retention_week, count(user_id) as no_of_users_retained from(
select a.user_id,
       a.sign_up_week,
       b.engagement_week,
       b.engagement_week - a.sign_up_week as retention_week
from(
(select distinct user_id, extract(week from occurred_at) as sign_up_week
from events
where event_type = 'signup_flow'
and event_name = 'complete_signup')a
left join
(select distinct user_id, extract(week from occurred_at) as engagement_week
from events
where event_type = 'engagement')b
on a.user_id = b.user_id)
where b.engagement_week - a.sign_up_week >= 1)c
group by retention_week;
```

retention_week	no_of_users_retained
1	2499
2	1287
3	834
4	554
5	388
6	300
7	257
8	178
9	158
10	116
11	91
12	75
13	50
14	33
15	25
16	6
17	3

from table2

Retention_week is after how many weeks the users are retained(after sign up the user log in for their use of the product) , no of user retained is the count of users.

The SQL code calculates the retention rate by counting the number of users retained for each retention week. Here's an explanation of the code:

SELECT statement:

retention_week: This selects the calculated retention week.

count(user_id) as no_of_users_retained: This counts the number of distinct user IDs and assigns it the alias "no_of_users_retained".

FROM clause:

(SELECT ...) c: This subquery is used to calculate the retention week and join the signup and engagement events data.

Subquery (a):

SELECT DISTINCT user_id, extract(week from occurred_at) as sign_up_week: This selects the distinct user IDs and extracts the week from the "occurred_at" column for signup events.

FROM events: This specifies the table from which the signup events data is retrieved.

WHERE event_type = 'signup_flow' AND event_name = 'complete_signup': This filters the data to include only events of type 'signup_flow' and with the event_name 'complete_signup'.

Subquery (b):

SELECT DISTINCT user_id, extract(week from occurred_at) as engagement_week: This selects the distinct user IDs and extracts the week from the "occurred_at" column for engagement events.

FROM events: This specifies the table from which the engagement events data is retrieved.

WHERE event_type = 'engagement': This filters the data to include only events of type 'engagement'.

LEFT JOIN clause:

ON a.user_id = b.user_id: This joins the signup events (subquery a) with the engagement events (subquery b) based on the user ID.

WHERE clause:

b.engagement_week - a.sign_up_week >= 1: This filters the joined data to include only the users whose engagement week is at least 1 week after their signup week. This ensures that only retained users are considered.

GROUP BY clause:

retention_week: This groups the data by the calculated retention week.

The code calculates the retention week by subtracting the signup week from the engagement week for each user. It then counts the number of users retained for each retention week

After the slit alter in the previous code we can able to see the list of user retained and number of time retained after signing up:

```
#C Weekly Retention
select user_id,count(user_id) as no_of_times_retained,
MIN(retention_week) AS first_retention_week, max(retention_week) AS last_retention_week
FROM (
select a.user_id,
       a.sign_up_week,
       b.engagement_week,
       b.engagement_week - a.sign_up_week as retention_week
from(
(select distinct user_id, extract(week from occurred_at) as sign_up_week
from events
where event_type = 'signup_flow'
and event_name = 'complete_signup')a
left join
(select distinct user_id, extract(week from occurred_at) as engagement_week
from events
where event_type = 'engagement')b
on a.user_id = b.user_id)
where b.engagement_week - a.sign_up_week >= 1) c
GROUP BY user_id;
```

user_id	no_of_times_retained	first_retention_week	last_retention_week
11775	1	1	1
11778	2	4	6
11779	4	1	4
11780	1	1	1
11787	2	1	2
11791	1	1	1
11793	5	1	5
11795	1	1	1
11798	5	1	6
11799	9	1	15
11801	1	1	1
11804	1	1	1
11811	1	1	1
11813	5	4	15
11816	2	7	17
11818	1	1	1
11820	3	1	5

from table2

There are more than 1000 rows in this table, where first_retention_week is after how many weeks the users login and used for first time.

last_retention_week is after how many weeks the user login and user for last time.

D. Weekly Engagement: To measure the activeness of a user. Measuring if the user finds quality in a product/service weekly.

My task: Calculate the weekly engagement per device?

```
#D Weekly Engagement
select
extract(year from occurred_at) as years,
extract(week from occurred_at) as weeks,
device,
count(distinct user_id) as no_of_users
from events
where event_type = 'engagement'
group by years,weeks,device
order by years,weeks;
```

years	weeks	device	no_of_users
2014	17	acer aspire desktop	9
2014	17	acer aspire notebook	20
2014	17	amazon fire phone	4
2014	17	asus chromebook	21
2014	17	dell inspiron desktop	18
2014	17	dell inspiron notebook	46
2014	17	hp pavilion desktop	14
2014	17	htc one	16
2014	17	ipad air	27
2014	17	ipad mini	19
2014	17	iphone 4s	21
2014	17	iphone 5	65
2014	17	iphone 5s	42
2014	17	kindle fire	6
2014	17	lenovo thinkpad	86

from table2

There are 491 rows in this solution.

Alternative code: where I tried and found better solution code

```
select *from events;
select distinct device from events;
select extract(week from occurred_at)as weeks,
count(distinct case when device ='acer aspire desktop' then user_id else null end) as acer_aspire_desktop,
count(distinct case when device ='acer aspire notebook' then user_id else null end) as acer_aspire_notebook,
count(distinct case when device ='amazon fire phone' then user_id else null end) as amazon_fire_phone,
count(distinct case when device ='asus chromebook' then user_id else null end) as asus_chromebook,
count(distinct case when device ='dell inspiron desktop' then user_id else null end) as dell_inspiron_desktop,
count(distinct case when device ='dell inspiron notebook' then user_id else null end) as dell_inspiron_notebook,
count(distinct case when device ='hp pavilion desktop' then user_id else null end) as hp_pavilion_desktop,
count(distinct case when device ='htc one' then user_id else null end) as htc_one,
count(distinct case when device ='ipad air' then user_id else null end) as ipad_air,
count(distinct case when device ='ipad mini' then user_id else null end) as ipad_mini,
count(distinct case when device ='iphone 4s' then user_id else null end) as iphone_4s,
count(distinct case when device ='iphone 5' then user_id else null end) as iphone_5,
count(distinct case when device ='iphone 5s' then user_id else null end) as iphone_5s,
count(distinct case when device ='kindle fire' then user_id else null end) as kindle_fire,
count(distinct case when device ='lenovo thinkpad' then user_id else null end) as lenovo_thinkpad,
count(distinct case when device ='mac mini' then user_id else null end) as mac_mini,
count(distinct case when device ='macbook air' then user_id else null end) as macbook_air,
count(distinct case when device ='macbook pro' then user_id else null end) as macbook_pro,
count(distinct case when device ='nexus 10' then user_id else null end) as nexus_10,
count(distinct case when device ='nexus 5' then user_id else null end) as nexus_5,
count(distinct case when device ='nexus 7' then user_id else null end) as nexus_7,
count(distinct case when device ='nokia lumia 635' then user_id else null end) as nokia_lumia_635,
count(distinct case when device ='samsumg galaxy tablet' then user_id else null end) as samsumg_galaxy_tablet,
count(distinct case when device ='samsung galaxy note' then user_id else null end) as samsung_galaxy_note,
count(distinct case when device ='samsung galaxy s4' then user_id else null end) as samsung_galaxy_s4,
count(distinct case when device ='windows surface' then user_id else null end) as windows_surface
from events
group by weeks;
```

weeks	acer_aspire_desktop	acer_aspire_notebook	amazon_fire_phone	asus_chromebook	dell_inspiron_desktop	dell_inspiron_notebook	hp_pavilion_desktop	htc_one	ipad_air
17	12	23	4	23	20	48	17	19	29
18	33	41	9	49	63	84	39	20	62
19	26	48	14	32	38	93	47	34	57
20	25	49	12	45	55	93	32	32	63
21	31	49	6	39	46	88	49	29	55
22	27	45	8	55	57	100	44	26	67
23	25	50	16	56	59	112	60	23	49
24	26	44	12	49	63	113	61	22	62
25	30	52	15	44	59	116	60	23	67
26	35	37	15	57	65	101	53	25	63
27	33	54	12	57	58	105	60	30	64
28	36	55	9	56	66	113	60	30	57
29	33	66	16	54	57	123	62	32	59
30	34	66	16	62	60	131	48	32	75
31	35	60	14	71	50	123	56	15	59
32	37	61	12	71	61	116	59	22	53
33	39	50	15	52	43	118	47	23	50
34	37	66	13	54	55	115	43	31	46
35	1	3	1	6	1	10	1	2	1

ipad_mini	iphone_4s	iphone_5	iphone_5s	kindle_fire	lenovo_thinkpad	mac_mini	macbook_air	macbook_pro	nexus_10	nexus_5	nexus_7	nokia_lumia_635
19	27	69	47	6	94	7	61	154	16	45	19	18
37	51	130	76	29	174	15	136	280	34	89	35	35
40	48	123	88	22	204	20	123	296	28	99	47	25
39	63	142	86	24	201	30	134	289	27	110	36	27
29	49	147	86	31	193	20	131	268	29	99	33	30
40	52	135	81	25	199	27	159	280	30	111	54	27
44	62	163	85	26	201	18	137	308	50	100	42	31
41	62	156	95	28	186	30	171	281	41	99	56	40
34	44	152	94	25	220	23	141	311	31	96	53	38
49	58	170	97	27	217	12	150	303	30	100	50	44
42	73	185	95	29	217	19	154	332	41	92	46	34
40	68	164	102	34	240	30	168	335	32	98	40	39
36	71	158	102	38	226	31	173	334	28	94	51	48
40	73	176	119	28	244	25	177	358	41	95	66	38
30	58	159	79	16	233	24	166	360	32	80	42	32
34	48	134	75	14	205	21	145	353	36	78	28	32
34	38	122	73	16	208	33	162	354	27	81	38	33
26	57	117	81	15	224	30	160	332	33	79	40	22
2	6	4	4	3	18	2	10	22	2	4	2	2

samsumg_galaxy_tablet	samsung_galaxy_note	samsung_galaxy_s4	windows_surface
8	8	58	11
15	18	90	13
6	12	103	17
11	19	106	24
7	21	96	21
12	19	120	17
16	17	116	16
11	21	112	25
13	15	106	22
15	10	129	24
17	17	127	33
12	11	132	36
15	19	140	33
9	17	121	25
8	20	104	23
8	13	96	12
14	14	94	17
16	15	107	20
0	1	7	3

from table2

The SQL code retrieves the distinct devices from the "events" table and then calculates the count of unique users for each device based on the weeks extracted from the "occurred_at" column. Here's an explanation of the code:

First query:

SELECT DISTINCT device FROM events: This retrieves the distinct devices from the "events" table.

Second query:

SELECT extract(week from occurred_at) as weeks: This extracts the week number from the "occurred_at" column and assigns it the alias "weeks".

COUNT(DISTINCT CASE WHEN device = '...' THEN user_id ELSE NULL END) as '...': This counts the distinct user IDs for each device using a conditional statement. Replace the '...' with the name of each device.

FROM events: This specifies the table from which the data is retrieved.

GROUP BY weeks: This groups the data by the extracted week number.

The code calculates the count of distinct users for each device based on the weeks. It uses conditional statements to count the user IDs only if the device matches the specified device name.

The output of this query will provide the weeks and the corresponding count of unique users for each device. It allows you to analyze user engagement and activity for different devices over time.

E. Email Engagement: Users engaging with the email service.

My task: Calculate the email engagement metrics?

```
#E Email Engagement
select weeks, sent_weekly_digest*100/total as 'Weekly_digest',
email_open*100/total as 'Weekly_email_open',
email_clickthrough*100/total as'Weekly_clickthrough',
sent_reengagement_email*100/total as'Weekly_reengagement'
from(
select extract(week from occurred_at)as weeks,count(user_id)as total,
sum(case when action ='sent_weekly_digest' then 1 else 0 end) as sent_weekly_digest,
sum(case when action ='email_open' then 1 else 0 end) as email_open,
sum(case when action ='email_clickthrough' then 1 else 0 end) as email_clickthrough,
sum(case when action ='sent_reengagement_email' then 1 else 0 end) as sent_reengagement_email
from email_events
group by weeks)a
group by weeks
order by weeks;
```

weeks	Weekly_digest	Weekly_email_open	Weekly_clickthrough	Weekly_reengagement
17	62.3198	21.2766	11.3933	5.0103
18	63.4479	22.2385	10.4852	3.8283
19	62.1647	22.6732	11.1267	4.0355
20	61.6234	22.6381	11.4318	4.3067
21	63.5156	22.8224	9.9707	3.6912
22	63.5867	21.5596	10.6597	4.1940
23	62.3935	22.3353	11.1781	4.0931
24	61.6071	22.9167	10.9921	4.4841
25	63.7701	21.7936	10.5389	3.8974
26	62.9912	22.2243	10.6066	4.1778
27	62.2413	22.4867	11.3715	3.9004
28	62.9203	22.4780	10.7714	3.8302
29	63.9829	21.7136	10.5094	3.7941
30	62.2857	23.2437	10.5882	3.8824
31	65.2728	23.2490	7.6579	3.8203
32	66.5926	22.8469	7.1429	3.4176
33	64.7306	23.1042	7.9058	4.2594
34	64.3349	23.9124	7.6682	4.0845
35	0.0000	32.2835	29.9213	37.7953

from table3

The SQL code calculates the percentage of various email actions (sent_weekly_digest, email_open, email_clickthrough, sent_reengagement_email) compared to the total number of emails for each week. Here's an explanation of the code:

SELECT statement:

weeks: This selects the extracted week number from the "occurred_at" column.

sent_weekly_digest * 100 / total as 'Weekly_digest': This calculates the percentage of sent_weekly_digest emails compared to the total number of emails and assigns it the alias 'Weekly_digest'.

email_open * 100 / total as 'Weekly_email_open': This calculates the percentage of email_open actions compared to the total number of emails and assigns it the alias 'Weekly_email_open'.

email_clickthrough * 100 / total as 'Weekly_clickthrough': This calculates the percentage of email_clickthrough actions compared to the total number of emails and assigns it the alias 'Weekly_clickthrough'.

sent_reengagement_email * 100 / total as 'Weekly_reengagement': This calculates the percentage of sent_reengagement_email actions compared to the total number of emails and assigns it the alias 'Weekly_reengagement'.

FROM clause:

(SELECT ...) a: This subquery calculates the counts of various email actions and the total number of emails for each week.

Subquery (a):

SELECT extract(week from occurred_at) as weeks: This selects the extracted week number from the "occurred_at" column and assigns it the alias 'weeks'.

count(user_id) as total: This counts the number of user IDs and assigns it the alias 'total'.

sum(case when action = 'sent_weekly_digest' then 1 else 0 end) as sent_weekly_digest: This sums the occurrences of 'sent_weekly_digest' actions and assigns it the alias 'sent_weekly_digest'.

sum(case when action = 'email_open' then 1 else 0 end) as email_open: This sums the occurrences of 'email_open' actions and assigns it the alias 'email_open'.

sum(case when action = 'email_clickthrough' then 1 else 0 end) as email_clickthrough: This sums the occurrences of 'email_clickthrough' actions and assigns it the alias 'email_clickthrough'.

sum(case when action = 'sent_reengagement_email' then 1 else 0 end) as sent_reengagement_email: This sums the occurrences of 'sent_reengagement_email' actions and assigns it the alias 'sent_reengagement_email'.

FROM email_events: This specifies the table from which the email events data is retrieved.

GROUP BY weeks: This groups the data by the extracted week number.

Outer GROUP BY clause:

weeks: This groups the data by the week number.

ORDER BY clause:

weeks: This orders the result set in ascending order based on the week number.

The code calculates the percentage of each email action compared to the total number of emails for each week. It uses conditional statements to sum the occurrences of specific actions. The outer query then groups the data by the week number and presents the results in ascending order.

Conclusion:

Operation Analytics plays a crucial role in analyzing the end-to-end operations of a company. As a Data Analyst Lead at Microsoft, my primary responsibility is to work closely with various teams, such as operations, support, and marketing, to derive meaningful insights from the data they collect.

Through Operation Analytics, we can identify areas for improvement, optimize workflows, and enhance cross-functional collaboration. By leveraging data-driven insights, we can make informed decisions that contribute to the overall growth and success of the company.

Additionally, investigating metric spikes is an essential aspect of Operation Analytics. As a Data Analyst, I need to understand and help other teams comprehend fluctuations in key metrics such as daily engagement or sales. By addressing these questions on a daily basis, we can proactively identify and rectify issues, ensuring the company's continued success.

By utilizing various data sets and tables, I can derive valuable insights and provide answers to the critical questions posed by different departments. This enables the organization to make data-driven decisions, enhance automation, and foster better understanding and collaboration among teams.