Operation Analytics and Investigating Metric Spike

Description: Operational Analytics is a crucial process that involves analyzing a company's end-to-end operations. This analysis helps identify areas for improvement within the company. As a Data Analyst, you'll work closely with various teams, such as operations, support, and marketing, helping them derive valuable insights from the data they collect.

One of the key aspects of Operational Analytics is investigating metric spikes. This involves understanding and explaining sudden changes in key metrics, such as a dip in daily user engagement or a drop in sales. As a Data Analyst, you'll need to answer these questions daily, making it crucial to understand how to investigate these metric spikes.

JANUARY 26

Trainity
Pavithra K R

CASE STUDY 1

Case Study 1: Job Data Analysis

You will be working with a table named job_data with the following columns:

- **job_id: Unique identifier of jobs**
- **actor_id: Unique identifier of actor**
- **event: The type of event (decision/skip/transfer).**
- **language: The Language of the content**
- **time_spent: Time spent to review the job in seconds.**
- **org: The Organization of the actor**
- **ds: The date in the format yyyy/mm/dd (stored as text).**

Tasks:

A. ***Jobs Reviewed Over Time:***
   - ***Objective: Calculate the number of jobs reviewed per hour for each day in November 2020.***
   - ***Your Task: Write an SQL query to calculate the number of jobs reviewed per hour for each day in November 2020.***
   -

select ds, count(*) as no_of_jobs, sum(time_spent)/3600 as hrs from job_data group by ds order by ds;

ds	no_of_jobs	hrs
2020-11-25	1	0.0125
2020-11-26	1	0.0156
2020-11-27	1	0.0289
2020-11-28	2	0.0092
2020-11-29	1	0.0056
2020-11-30	2	0.0111

B. ***Throughput Analysis:***
   - ***Objective: Calculate the 7-day rolling average of throughput (number of events per second).***
   - ***Your Task: Write an SQL query to calculate the 7-day rolling average of throughput. Additionally, explain whether you prefer using the daily metric or the 7-day rolling average for throughput, and why.***

```
with abc as(
select ds,
count(actor_id) as no_of_events,
sum(time_spent) as tot_time,
round(count(actor_id)/sum(time_spent),3) as tps from job_data group by ds)
select ds as review_date, no_of_events, tot_time,
tps as throughput_per_sec,
round(avg(tps) over(order by ds),3) as rolling_avg from abc order by ds;
```

A 7-day rolling average is preferred over daily metrics. Rolling average is preferred for analyzing the data for a specific time interval to find trends in the data.

review_date	no_of_events	tot_time	throughput_per_sec	rolling_avg
2020-11-25	1	45	0.022	0.022
2020-11-26	1	56	0.018	0.020
2020-11-27	1	104	0.010	0.017
2020-11-28	2	33	0.061	0.028
2020-11-29	1	20	0.050	0.032
2020-11-30	2	40	0.050	0.035

C. ***Language Share Analysis:***

- ***Objective: Calculate the percentage share of each language in the last 30 days.***
- ***Your Task: Write an SQL query to calculate the percentage share of each language over the last 30 days.***

```
with abc as (select count(*) as tot_count from job_data)
select language,count(language) as lang_count, (count(language)/tot_count)*100 as
percent_share
from job_data,abc
group by language,tot_count
order by percent_share desc;
```

language	lang_count	percent_share
Persian	3	37.5000
English	1	12.5000
Arabic	1	12.5000
Hindi	1	12.5000
French	1	12.5000
Italian	1	12.5000

D. ***Duplicate Rows Detection:***
   - ***Objective: Identify duplicate rows in the data.***
   - ***Your Task: Write an SQL query to display duplicate rows from the job_data table.***

select ds,job_id,actor_id,event,language,time_spent,org,count(*) as dup_count from job_data group by ds,job_id,actor_id,event,language,time_spent,org having count(*)>1;

We have no duplicates in the table.

CASE STUDY 2

Case Study 2: Investigating Metric Spike

You will be working with three tables:

- **users: Contains one row per user, with descriptive information about that user's account.**
- **events: Contains one row per event, where an event is an action that a user has taken (e.g., login, messaging, search).**
- **email_events: Contains events specific to the sending of emails.**

Tasks:

A. ***Weekly User Engagement:***
   - ***Objective: Measure the activeness of users on a weekly basis.***
   - ***Your Task: Write an SQL query to calculate the weekly user engagement.***

```
WITH abc AS (
   SELECT
      COUNT(*) AS c1,
      user_id,
      YEAR(occurred_at) AS yr1,
      WEEK(occurred_at) AS wk1
   FROM email_events
   GROUP BY user_id, yr1, wk1
),
def AS (
   SELECT
      COUNT(*) AS c2,
      user_id,
      YEAR(occurred_at) AS yr2,
      WEEK(occurred_at) AS wk2
   FROM events
   GROUP BY user_id, yr2, wk2
),
left_joined AS (
   SELECT
      abc.yr1 AS yr3,
      abc.wk1 AS wk3,
      COALESCE(abc.c1, 0) + COALESCE(def.c2, 0) AS count,
      abc.user_id AS u_id
   FROM abc
   LEFT JOIN def
   ON abc.user_id = def.user_id
     AND abc.yr1 = def.yr2
     AND abc.wk1 = def.wk2
),
right_joined AS (
   SELECT
      def.yr2 AS yr3,
```

```
        def.wk2 AS wk3,
        COALESCE(abc.c1, 0) + COALESCE(def.c2, 0) AS count,
        def.user_id AS u_id
    FROM def
    LEFT JOIN abc
    ON def.user_id = abc.user_id
      AND def.yr2 = abc.yr1
      AND def.wk2 = abc.wk1
)
SELECT
    u_id,
    yr3 AS year,
    wk3 AS week,
    count
FROM left_joined
UNION ALL
SELECT
    u_id,
    yr3 AS year,
    wk3 AS week,
    count
FROM right_joined
ORDER BY count DESC;
```

B. User Growth Analysis:

- *Objective: Analyze the growth of users over time for a product.*
- *Your Task: Write an SQL query to calculate the user growth for the product.*

```
/*new users per week*/
select week(created_at) as week, year(created_at) as year, count(*) as
no_of_new_users from
```

```
users group by week,year order by year;

/*active users*/
select year(occurred_at) as year ,week(occurred_at) week,count(distinct user_id)
no_of_active_users from (
select user_id, occurred_at from email_events
union
select user_id, occurred_at from events) as t1 group by year,week order by year;

/*user growth*/
with abc as (
select week(created_at) as week, year(created_at) as year, count(*) as
no_of_new_users from
users group by week,year order by year)
select week, year, no_of_new_users, no_of_new_users- lag(no_of_new_users,1,0)
over(order by year,week) as change_in_users,
case
when no_of_new_users- lag(no_of_new_users,1,0) over(order by year,week) >0 then
"Increase"
when no_of_new_users- lag(no_of_new_users,1,0) over(order by year,week)<0 then
"Decrease"
else "No change"
end as growth_insight
from abc;
```

C. *Weekly Retention Analysis:*
   - *Objective: Analyze the retention of users on a weekly basis after signing up for a product.*
   - *Your Task: Write an SQL query to calculate the weekly retention of users based on their sign-up cohort.*

```
WITH abc AS (
  SELECT
```

```
    user_id,
    YEAR(created_at) AS s_year,
    WEEK(created_at) AS s_week
  FROM users
),
def AS (
  SELECT
    user_id,
    YEAR(occurred_at) AS a_year,
    WEEK(occurred_at) AS a_week
  FROM (
    SELECT user_id, occurred_at FROM email_events
    UNION
    SELECT user_id, occurred_at FROM events
  ) AS all_events
),
ijk AS (
  SELECT
    c.s_year,
    c.s_week,
    a.a_year,
    a.a_week,
    COUNT(DISTINCT a.user_id) AS retained_users
  FROM abc c
  join def a on c.user_id = a.user_id
 group by c.s_year,
    c.s_week,
    a.a_year,
    a.a_week,
)
SELECT
  s_year,
  s_week,
```

```
    a_year,
    a_week,
    retained_users
FROM ijk
ORDER BY s_year, s_week, a_year, a_week;
```

D. Weekly Engagement Per Device:

- *Objective: Measure the activeness of users on a weekly basis per device.*
- *Your Task: Write an SQL query to calculate the weekly engagement per device.*

```
select year(occurred_at) as year, week(occurred_at) as week, device, count(distinct
user_id) as no_of_users
from events
group by device, week, year
order by week, year;
```

E. Email Engagement Analysis:

- *Objective: Analyze how users are engaging with the email service.*
- *Your Task: Write an SQL query to calculate the email engagement metrics.*

```
/*finds no of users doing a particular action per week*/
select year(occurred_at) as year, week(occurred_at) as week, action, count(distinct
user_id) as no_of_users
from email_events
group by action, week, year
order by week, year;

/*finds email events activity per day*/
select
case
when dayofweek(occurred_at)=0 then "SUN"
```

```
when dayofweek(occurred_at)=1 then "MON"
when dayofweek(occurred_at)=2 then "TUE"
when dayofweek(occurred_at)=3 then "WED"
when dayofweek(occurred_at)=4 then "THU"
when dayofweek(occurred_at)=5 then "FRI"
else "SAT"
end as day_of_week, action, count(distinct user_id) as no_of_users
from email_events
group by action, day_of_week
order by day_of_week;

select user_type, count(user_id) as activities from email_events group by user_type ;

select user_type, year(occurred_at) as yr, week(occurred_at) as wk, count(user_id)
as activities from email_events group by yr,wk,user_type
order by wk,yr;

/*find user with most no of email activities per year*/
select user_id, count(action) as no_act, year(occurred_at) as yr
from email_events
group by user_id,yr
order by no_act desc;

with abc as (
select user_id, count(action) as no_act, year(occurred_at) as yr,
row_number() over (partition by year(occurred_at) order by count(action) desc) as
ranks
from email_events
group by user_id, yr
)
select user_id, no_act, yr, ranks
from abc
order by ranks;
```