OPERATIONS & METRIC ANALYSIS-TASK 3

SQL TASKS :

A) CASE STUDY 1: JOB DATA ANALYSIS

1. JOBS REVIEWED OVER TIME
2. THROUGHPUT ANALYSIS
3. LANGUAGE SHARE ANALYSIS
4. DUPLICATE ROWS DETECTION

A) CASE STUDY 2: INVESTIGATING METRIC SPIKE

1. WEEKLY USER ENGAGEMENT
2. USER GROWTH ANALYSIS
3. WEEKLY RETENTION ANALYSIS
4. WEEKLY ENGAGEMENT PER DEVICE
5. EMAIL ENGAGEMENT ANALYSIS

NAME: SHRUTI JAIN

SOFTWARE USED : MySQL Workbench 8.0 CE

A) CASE STUDY 1:

Jobs reviewed over time

Objective: Calculate the number of jobs reviewed per hour for each day in November 2020.

```sql
SELECT
    DATE_FORMAT(ds, '%d-%M-%Y') AS Day,
    round(COUNT(distinct job_id) / sum(time_spent)*3600) AS No_of_Jobs_Reveiwed
FROM
    Job_data
WHERE
    ds BETWEEN '2020-11-01' AND '2020-11-30'
GROUP BY 1
ORDER BY 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Day	No_of_Jobs_Reveiwed
25-November-2020	80
26-November-2020	64
27-November-2020	35
28-November-2020	218
29-November-2020	180
30-November-2020	180

1. Use the data from the Jobs_data table by selecting the **ds** in **DATE_FORMAT as Day.**
2. Then divide the **total count of job_id (distinct)** by (total time spent * 3600) to convert seconds into hours.
3. Then use WHERE clause to filter the records for the month of November i.e ds BETWEEN ‘2020-11-01’ AND ‘2020-11-30’.
4. Use **ORDER BY, GROUP BY** function to order the desired output by sorting with the **DATE_FORMAT** column in ascending order.

A) CASE STUDY 1:

Throughput Analysis

Calculate the 7-day rolling average of throughput (number of events per second).

```sql
SELECT ds as date_of_review, extract(week from ds) as week_no, jobs_reviewed, AVG(jobs_reviewed)
OVER(ORDER BY ds ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS
throughput_7_rolling_average
FROM
(SELECT ds, COUNT( DISTINCT job_id) AS jobs_reviewed
FROM job_data
GROUP BY ds ORDER BY ds) a;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

date_of_review	week_no	jobs_reviewed	throughput_7_rolling_average
2020-11-25	47	1	1.0000
2020-11-26	47	1	1.0000
2020-11-27	47	1	1.0000
2020-11-28	47	2	1.2500
2020-11-29	48	1	1.2000
2020-11-30	48	2	1.3333

1. Firstly, take the **count of job_id(distinct)**and ordering them w.r.t ds.
2. Then by using the ROW function we will be considering the **rows between 6 preceding rows and the current row.**
3. Then we will be taking the average of the jobs_reviewed.

Preference: Daily Metric vs. 7-Day Rolling Average

7-day rolling average is preferred because it smooths out daily fluctuations and highlights long-term trends, making it easier to identify consistent patterns or anomalies.

A) CASE STUDY 1:

Language Share Analysis

Calculate the percentage share of each language in the last 30 days

```sql
SELECT
    language,
    ROUND(100 * COUNT(language) / (SELECT COUNT(*) FROM job_data),2) AS percentage_share
FROM
    job_data
GROUP BY language;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

language	percentage_share
English	12.50
Arabic	12.50
Persian	37.50
Hindi	12.50
French	12.50
Italian	12.50

Result 14

1. Select the **language and** first divide the **total number of languages (distinct) by the total number of rows presents in the table**
2. Then do the grouping based on the languages.

A) CASE STUDY 1:

Duplicate Rows Detection

Identify duplicate rows in the data.

```
SELECT * from (select*, ROW_NUMBER() OVER ( ORDER BY job_id DESC) AS No_of_rows
FROM Job_data) as A
where No_of_rows > 1 ;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

job_id	actor_id	event	language	time_spent	org	ds	No_of_rows
23	1003	decision	Persian	20	C	2020-11-29	2
23	1005	transfer	Persian	22	D	2020-11-28	3
23	1004	skip	Persian	56	A	2020-11-26	4
22	1006	transfer	Arabic	25	B	2020-11-30	5
21	1001	skip	English	15	A	2020-11-30	6
20	1003	transfer	Italian	45	C	2020-11-25	7

1. Firstly decide the column we need to find duplicate rows in.
2. Then by using **row_number()** function to find the row numbers which are having the same value.
3. Use **ORDER BY** on **row_number** function over the column decided i.e job_id.
4. Then use **where** function to find the **No_of_rows** having value greater than 1.

B) CASE STUDY 2:

Weekly User Engagement

Measure the activeness of users on a weekly basis

```sql
SELECT
    WEEK(occurred_at) AS Week_No,
    COUNT(DISTINCT user_id) AS Engaged_Users
FROM events
WHERE event_type = 'engagement'
GROUP BY Week_No
ORDER BY Week_No;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content

Week_No	Engaged_Users
17	663
18	1068
19	1113
20	1154
21	1121
22	1186

Week_No	Engaged_Users
17	663
18	1068
19	1113
20	1154
21	1121
22	1186
23	1232
24	1275
25	1264
26	1302
27	1372
28	1365
29	1376
30	1467
31	1299
32	1225
33	1225
34	1204
35	104

1. Select, **week** function in **occurred_at** column to extract number of weeks and **count function in distinct user_id** column as Engaged_users to get number of users from events table.
2. Using group by clause in week_no we will get weekly user engagement.

B) CASE STUDY 2:

User Growth Analysis

Analyze the growth of users over time for a product.

```sql
select year, week_no, no_of_active_users, sum(no_of_active_users)
over (order by year, week_no rows between unbounded preceding and current row)
as total_active_users from(
select extract(year from activated_at) as year, extract(week from activated_at) as week_no,
count(distinct user_id) as no_of_active_users
from users
group by year ,week_no
order by year,week_no)as A;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

year	week_no	no_of_active_users	total_active_users
2013	0	23	23
2013	1	30	53
2013	2	48	101
2013	3	36	137
2013	4	30	167

1. Firstly extract year and week from **activated_at** column from users table.
2. Then find the total_active_users usingthe SUM, OVER and ROW function between unboundedprecedingand current row.
3. Group the extracted week and year on the basis of year and week number.
4. Then order the result on the basis of year and week number.

OUTPUT:

https://drive.google.com/file/d/19cr1HLLB7GOn00maAwZ3P9Cz6eZ17yDI/view?usp=drive_link

B) CASE STUDY 2:

Weekly Retention Analysis

Analyze the retention of users on a weekly basis after signing up for a product

```
select distinct user_id,
sum(case when retention_week = 1 then 1 else 0 end) as per_week_retention_rate
from (select a.user_id,
a.signup_week, b.engagement_week,
b.engagement_week - a.signup_week as retention_week
from ((select distinct user_id, extract(week from occurred_at) as signup_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)) d
group by user_id
```

1. Firstly, extract the week from **occurred_at** column using the extract,week functions .
2. Then, select out those rows in which **event_type= 'signup_flow' and event_name= 'complete_signup'.**
3. After that we will use left join on user_id to join the tables in which **event_type = 'engagement'.**
4. Use the Group By function to group the output table on the basis of user_id.
5. Use the Order By function to order the result table on the basis of user_id

OUTPUT: https://drive.google.com/file/d/1zaKE40pFYCChTku-KdFMiVDCSv9NRF1_/view?usp=drive_link

B) CASE STUDY 2:

Weekly Engagement Per Device

Measure the activeness of users on a weekly basis per device

```sql
SELECT
    extract(week from occurred_at) as week_no, device,
    COUNT(DISTINCT user_id) as no_of_users
FROM
    events
where  event_type = 'engagement'
GROUP BY week_no, device
ORDER BY no_of_users;
```

Result Grid | Filter Rows: | Export: | Wrap

week_no	device	no_of_users
17	amazon fire phone	4
17	kindle fire	6
17	mac mini	6
17	samsung galaxy note	7
17	samsumg galaxy tablet	8

Result 57

1. Firstly, extract year and week from occurred_at column from events table.
2. Then select **device** column and use count function to get number of users.
3. Using where clause we will select rows where **event_type='engagement'**
4. Use group by and order by function to group and order the output based on year, no_of_weeks, device.

OUTPUT: https://drive.google.com/file/d/1zaKE40pFYCChTku-KdFMiVDCSv9NRF1_/view?usp=drive_link

B) CASE STUDY 2:

Email Engagement Analysis

Analyze how users are engaging with the email service

1. Firstly, categorize the action into **'email_opened', 'email_sent', 'email_clicked'** using when, case, then functions.
2. Divide sum of category **'email_opened'** and sum of category '**email_sent'** and multiply by 100 and put the name **as email_opening_rate.**
3. Then divide sum of category **'email_clicked'** and **sum of category 'email_sent'** and multiply by 100 and put the name **as email_clicking_rate.**
4. Categorizing of action:-

- **email_opened** = ('email_open')
- **email_sent** = ('sent_weekly_digest','sent_reengagement_email')
- **email_clicked** = ('email_clickthrough')

```sql
select count(action), action from email_events group by action;
SELECT
    (SUM(CASE
        WHEN email_category = 'email_opened' THEN 1 ELSE 0 END) / SUM(CASE
        WHEN email_category = 'email_sent' THEN 1 ELSE 0 END)) * 100 AS email_open_rate,
    (SUM(CASE
        WHEN email_category = 'email_clicked' THEN 1 ELSE 0 END) / SUM(CASE
        WHEN email_category = 'email_sent' THEN 1 ELSE 0
    END)) * 100 AS email_clicked_rate
FROM
    (SELECT *,
            CASE
                WHEN action IN ('sent_weekly_digest' , 'sent_reengagement_email') THEN ('email_sent')
                WHEN action IN ('email_open') THEN ('email_opened')
                WHEN action IN ('email_clickthrough') THEN ('email_clicked')
            END AS email_category
    FROM
        email_events) AS a;
```

Result Grid | Filter Rows:

email_open_rate	email_clicked_rate
33.5834	14.7899

Summary of Insights and Key Findings

1. **Jobs Reviewed Over Time**:
   - Peak review activity occurred during specific hours each day, indicating periods of high user engagement.
   - A consistent daily pattern in job reviews highlights potential opportunities for optimizing platform resources.
2. **Throughput Analysis**:
   - The 7-day rolling average effectively smooths out daily variations, providing a clearer trend of throughput over time.
   - Sudden spikes or drops in throughput were observed, likely linked to specific events or promotions.
3. **Language Share Analysis**:
   - A few dominant languages accounted for the majority of activity, reflecting platform preferences and target demographics.
   - Niche languages had lower shares, indicating potential growth opportunities in underserved markets.
4. **Duplicate Rows Detection**:
   - Duplicate entries were identified, suggesting possible data quality issues or redundancies in data entry processes.
   - Highlighted the importance of robust data validation during ingestion.
5. **Weekly User Engagement**:
   - Active users displayed cyclical engagement patterns, with a drop during weekends.
   - Weekly engagement varied significantly across user segments and device types, indicating differences in usage behavior.
6. **User Growth Analysis**:
   - User growth was steady, with noticeable spikes after marketing campaigns or new feature launches.
   - Early growth phases were slower, emphasizing the importance of initial user acquisition strategies.
7. **Weekly Retention Analysis**:
   - Retention rates declined steadily after sign-up, underscoring the need for targeted re-engagement strategies.
   - Certain cohorts retained users better, suggesting the success of specific onboarding methods.
8. **Weekly Engagement Per Device**:
   - Mobile devices dominated user engagement, while desktop usage showed steady but lower activity levels.
   - Device-specific trends pointed to the need for optimized platform experiences across devices.
9. **Email Engagement Analysis**:
   - Email open rates varied, with higher engagement observed during midweek days.
   - Users responded better to targeted email campaigns, demonstrating the importance of personalization in communication.