Topic: Operation Analytics and Investigating Metric Spike

Advanced SQL

Description:

This project focuses on analyzing operational efficiency and identifying investment metric spikes by leveraging four distinct data sources: users , job_data, email_events, events and user interaction data. Advanced SQL queries and techniques were employed to extract, transform, and analyze this data from large-scale relational databases.

- ***Case Study 1: Job Data Analysis***

job_data table 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)

Table Name : job_data

ds	job_id	actor_id	event	language	time_spent	org
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
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

job_data 1 ×

A) Jobs Reviewed Over Time:

- Objective: Calculate the number of jobs reviewed per hour for each day in November 2020.
- Task : Write an SQL query to calculate the number of jobs reviewed per hour for each day in November 2020.

Code:

```
select

ds as review_date,

time_spent as review_hour,

count(job_id) as job_reviewed

from job_data

where ds>= '2020-11-01' AND ds <= '2020-11-30'

group by review_date ,review_hour

order by review_date , review_hour;
```

Output:

	review_date	review_hour	job_reviewed
▶	2020-11-25	45	2
	2020-11-26	56	2
	2020-11-27	104	2
	2020-11-28	11	2
	2020-11-28	22	2
	2020-11-29	20	2
	2020-11-30	15	2
	2020-11-30	25	2

B) Throughput Analysis:

- Objective:

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

- 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

Code:

```
with daily_throughput as (
select ds as review_date,
count(job_id) as jobs_reviewed
from job_data
group by ds
)
select review_date,
jobs_reviewed,
avg(jobs_reviewed) over(
```

```
order by review_date

rows between 6 preceding and current row

) as rolling_avg_7_days

from daily_throughput

order by review_date;
```

Output:

review_date	jobs_reviewed	rolling_avg_7_days
2020-11-25	2	2.0000
2020-11-26	2	2.0000
2020-11-27	2	2.0000
2020-11-28	4	2.5000
2020-11-29	2	2.4000
2020-11-30	4	2.6667

Preference: 7-Day Rolling Average Over Daily Metrics for Throughput

I prefer using the **7-day rolling average** over daily throughput metrics for the following reasons:

1. **Reduces Daily Fluctuations:**
   The 7-day rolling average smooths out natural day-to-day variations, making it easier to observe meaningful performance trends without being misled by short-term noise.
2. **Enhances Trend Visibility:**
   Rolling averages help highlight underlying trends in throughput, minimizing the visual and analytical impact of sudden spikes or dips that may not reflect the long-term operational state.
3. **Filters Out Anomalies:**
   Daily metrics can be heavily influenced by one-off events such as holidays, outages, or data inconsistencies. A rolling average softens the effects of these anomalies, providing a more representative view of ongoing operations.
4. **Improves Strategic Insights:**
   For decision-making and monitoring key performance indicators, a rolling average offers

a more stable and reliable measure of operational efficiency, which leads to better-informed business actions.

C) Language Share Analysis:

- Objective:

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

- Task:

Write an SQL query to calculate the percentage share of each language over the last 30 days.

Code:

```
WITH last_30_days_jobs AS (
SELECT
language,
COUNT(*) AS jobs_count
FROM job_data
WHERE ds BETWEEN '2020-11-01' AND '2020-11-30'
GROUP BY language
),
total_jobs AS (
SELECT SUM(jobs_count) AS total_jobs_count
FROM last_30_days_jobs
)
SELECT l.language,
l.jobs_count,
ROUND((l.jobs_count / t.total_jobs_count) * 100, 2) AS percentage_share
FROM
last_30_days_jobs l,
total_jobs t
```

```
ORDER BY
percentage_share DESC;
```

Output:

language	jobs_count	percentage_share
Persian	6	37.50
English	2	12.50
Arabic	2	12.50
Hindi	2	12.50
French	2	12.50
Italian	2	12.50

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.

Code:

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

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

Output:

ds	job_id	actor_id	event	language	time_spent	org
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

- ***There is 8 Duplicate Rows***

- ***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.

Creating the tables from given datasets case study 2

- ***table 1 :users***

```
create table users (
user_id int,
created_at varchar(100),
```

company_id int,

language varchar (50),

activated_at varchar(100),

state varchar(50));

- ***Import data***

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/users.csv'

INTO TABLE events

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 LINES;

table users:

user_id	created_at	company_id	language	activated_at	state
0	01-01-2013 20:59	5737	english	01-01-2013 21:01	active
3	01-01-2013 18:40	2800	german	01-01-2013 18:42	active
4	01-01-2013 14:37	5110	indian	01-01-2013 14:39	active
6	01-01-2013 18:37	11699	english	01-01-2013 18:38	active
7	01-01-2013 16:19	4765	french	01-01-2013 16:20	active
8	01-01-2013 04:38	2698	french	01-01-2013 04:40	active
11	01-01-2013 08:07	3745	english	01-01-2013 08:09	active
13	02-01-2013 12:27	4025	english	02-01-2013 12:29	active
15	02-01-2013 15:39	4259	english	02-01-2013 15:41	active
17	02-01-2013 10:56	5025	japanese	02-01-2013 10:57	active
19	02-01-2013 09:54	326	english	02-01-2013 09:55	active
20	02-01-2013 09:41	7	italian	02-01-2013 09:43	active
21	02-01-2013 09:29	2606	english	02-01-2013 09:30	active

users 10

- ***table 2: events***

create table events (

```
user_id int,
occurred_at varchar(100),
event_type varchar(50),
event_name varchar(100),
location varchar(50),
device varchar(50),
user_type int ) ;
```

- ***Import data***

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

event table:

user_id	occurred_at	event_type	event_name	location	device	user_type
10522	02-05-2014 11:02	engagement	login	Japan	dell inspiron notebook	3
10522	02-05-2014 11:02	engagement	home_page	Japan	dell inspiron notebook	3
10522	02-05-2014 11:03	engagement	like_message	Japan	dell inspiron notebook	3
10522	02-05-2014 11:04	engagement	view_inbox	Japan	dell inspiron notebook	3
10522	02-05-2014 11:03	engagement	search_run	Japan	dell inspiron notebook	3
10522	02-05-2014 11:03	engagement	search_run	Japan	dell inspiron notebook	3
10612	01-05-2014 09:59	engagement	login	Netherlands	iphone 5	1
10612	01-05-2014 10:00	engagement	like_message	Netherlands	iphone 5	1
10612	01-05-2014 10:00	engagement	send_message	Netherlands	iphone 5	1
10612	01-05-2014 10:01	engagement	home_page	Netherlands	iphone 5	1
10612	01-05-2014 10:01	engagement	like_message	Netherlands	iphone 5	1
10612	01-05-2014 10:02	engagement	home_page	Netherlands	iphone 5	1
10612	01-05-2014 10:02	engagement	view_inbox	Netherlands	iphone 5	1

events 9

- ***table 3: email_events***

```
create table email_events (
```

```
user_id int,
occurred_at varchar(100),
action varchar(100),
user_type int);
```

- ***Import Data***

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

email_events table:

user_id	occurred_at	action	user_type
0	06-05-2014 09:30	sent_weekly_digest	1
0	13-05-2014 09:30	sent_weekly_digest	1
0	20-05-2014 09:30	sent_weekly_digest	1
0	27-05-2014 09:30	sent_weekly_digest	1
0	03-06-2014 09:30	sent_weekly_digest	1
0	03-06-2014 09:30	email_open	1
0	10-06-2014 09:30	sent_weekly_digest	1
0	10-06-2014 09:30	email_open	1
0	17-06-2014 09:30	sent_weekly_digest	1
0	17-06-2014 09:30	email_open	1
0	24-06-2014 09:30	sent_weekly_digest	1
0	01-07-2014 09:30	sent_weekly_digest	1
0	08-07-2014 09:30	sent_weekly_digest	1

A) Weekly User Engagement:

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

Code:

```
SELECT
  EXTRACT(YEAR FROM STR_TO_DATE(occurred_at, '%Y-%m-%d')) AS year_num,
  EXTRACT(WEEK FROM STR_TO_DATE(occurred_at, '%Y-%m-%d')) AS week_num,
  COUNT(DISTINCT user_id) AS no_of_users
FROM events
WHERE event_type='engagement'
GROUP BY year_num, week_num
ORDER BY year_num, week_num;
```

Output:

year_num	week_num	no_of_users
2001	20	293
2001	24	86
2001	28	354
2001	33	430
2002	20	358
2002	24	285
2002	28	329
2002	33	189
2003	20	145
2003	24	345
2003	29	385
2003	33	111
2004	20	79

Result 12

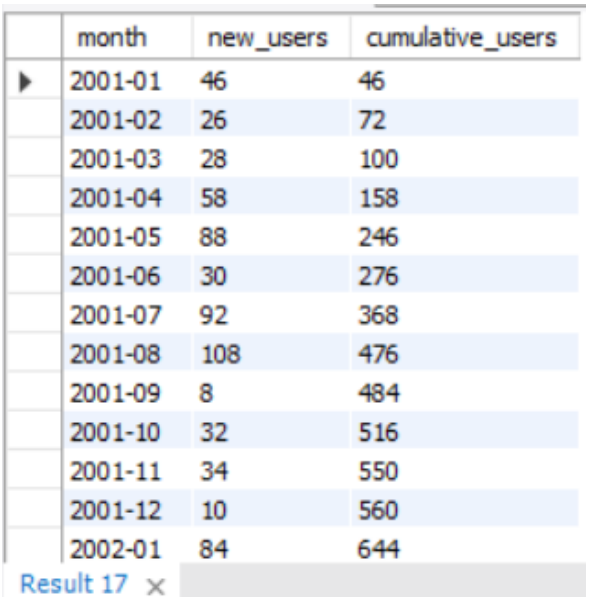
B) User Growth Analysis:

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

Code:

```
WITH monthly_user_growth AS (
 SELECT
  DATE_FORMAT(STR_TO_DATE(created_at, '%Y-%m-%d'), '%Y-%m') As month, -- Extract year and month
  COUNT(user_id) AS new_users -- Count new users per month
  FROM users
  GROUP BY DATE_FORMAT(STR_TO_DATE(created_at, '%Y-%m-%d'), '%Y-%m')
)
SELECT
   month,
   new_users,
   SUM(new_users) OVER (ORDER BY month) AS cumulative_users -- Cumulative sum of users over time
FROM monthly_user_growth
ORDER BY month;
```

Output:



month	new_users	cumulative_users
2001-01	46	46
2001-02	26	72
2001-03	28	100
2001-04	58	158
2001-05	88	246
2001-06	30	276
2001-07	92	368
2001-08	108	476
2001-09	8	484
2001-10	32	516
2001-11	34	550
2001-12	10	560
2002-01	84	644

Result 17

C) Weekly Retention Analysis:

- Objective:

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

- *Task*:

  Write an SQL query to calculate the weekly retention of users based on their

  sign-up cohort.

Code:

```
WITH user_sign_up_weeks AS (
  -- Extract the year-week of user sign-up
  SELECT
    user_id,
    DATE_FORMAT(STR_TO_DATE(created_at, '%d-%m-%Y %H:%i:%s'), '%Y-%u') AS sign_up_week
  FROM users
),
user_activity_weeks AS (
  -- Extract the year-week of user activity from events and email_events
  SELECT
    user_id,
    DATE_FORMAT(STR_TO_DATE(occurred_at, '%d-%m-%Y %H:%i:%s'), '%Y-%u') AS activity_week
  FROM events
  UNION ALL
  SELECT
    user_id,
    DATE_FORMAT(STR_TO_DATE(occurred_at, '%d-%m-%Y %H:%i:%s'), '%Y-%u') AS activity_week
  FROM email_events
```

```
),
user_weekly_retention AS (
  -- Calculate the number of weeks after sign-up that the user was active
  SELECT
    u.user_id,
    u.sign_up_week,
    a.activity_week,
    TIMESTAMPDIFF(
      WEEK,
      STR_TO_DATE(CONCAT(u.sign_up_week, '-1'), '%Y-%u-%w'),
      STR_TO_DATE(CONCAT(a.activity_week, '-1'), '%Y-%u-%w')
    ) AS weeks_after_signup
  FROM user_sign_up_weeks u
  JOIN user_activity_weeks a ON u.user_id = a.user_id
  WHERE a.activity_week >= u.sign_up_week -- Only consider activities on or after signup week
)
SELECT
  sign_up_week,
  weeks_after_signup,
  COUNT(DISTINCT user_id) AS retained_users
FROM user_weekly_retention
GROUP BY sign_up_week, weeks_after_signup
ORDER BY sign_up_week, weeks_after_signup;
```

Output:

	sign_up_week	weeks_after_signup	retained_users
	2013-01	75	10
	2013-01	76	10
	2013-01	77	10
	2013-01	78	10
	2013-01	79	10
	2013-01	80	10
	2013-01	81	10
	2013-01	82	10
	2013-01	83	10
	2013-01	84	10
	2013-01	85	10
	2013-01	86	10
	2013-02	68	4

Result 18 ×

D) Weekly Engagement Per Device:

- Objective:

  Measure the activeness of users on a weekly basis per device.

- Task:

  Write an SQL query to calculate the weekly engagement per device.

Code:

```
WITH events_cleaned AS (
  SELECT
    STR_TO_DATE(occurred_at, '%d-%m-%Y %H:%i:%s') AS occurred_dt,
    device
     FROM events
```

```
),
weekly_device_engagement AS (
  SELECT
    DATE_FORMAT(occurred_dt, '%Y-%u') AS week_start, -- Year-Week (calendar weeks)
    device,
    COUNT(*) AS event_count
  FROM events_cleaned
  GROUP BY week_start, device
)
SELECT
  week_start,
  device,
  event_count
FROM weekly_device_engagement
ORDER BY week_start, device;
```

Output:

week_start	device	event_count
2014-18	acer aspire desktop	142
2014-18	acer aspire notebook	430
2014-18	amazon fire phone	168
2014-18	asus chromebook	572
2014-18	dell inspiron desktop	396
2014-18	dell inspiron notebook	1138
2014-18	hp pavilion desktop	282
2014-18	htc one	384
2014-18	ipad air	716
2014-18	ipad mini	458
2014-18	iphone 4s	438
2014-18	iphone 5	1548
2014-18	iphone 5s	1024

Result 19

E) Email Engagement Analysis:

- Objective:

  Analyze how users are engaging with the email service.

- Task:

  Write an SQL query to calculate the email engagement metrics.

Code:

```
WITH email_cleaned AS (
  -- Convert occurred_at to proper DATETIME
```

```
    SELECT
        STR_TO_DATE(occurred_at, '%d-%m-%Y %H:%i:%s') AS occurred_dt,
        action
    FROM email_events
),
weekly_email_engagement AS (
    -- Weekly aggregation by action
    SELECT
        DATE_FORMAT(occurred_dt, '%Y-%u') AS week_start, -- Year-week (Sunday-based weeks)
        action,
        COUNT(*) AS action_count
    FROM email_cleaned
    GROUP BY week_start, action
)
SELECT
    week_start,
    action,
    action_count
FROM weekly_email_engagement
ORDER BY week_start, action;
```

Output:

week_start	action	action_count
2014-19	sent_reengagement_email	328
2014-19	sent_weekly_digest	5204
2014-20	email_dickthrough	958
2014-20	email_open	1942
2014-20	sent_reengagement_email	350
2014-20	sent_weekly_digest	5330
2014-21	email_dickthrough	996
2014-21	email_open	1990
2014-21	sent_reengagement_email	358
2014-21	sent_weekly_digest	5466
2014-22	email_dickthrough	906
2014-22	email_open	2052
2014-22	sent_reengagement_email	358

Result 20