

Project: User Engagement & Retention Analysis

"Analysed login patterns, inactive users, and high-value customers to drive retention strategies"

Overview

The User Login Activity Analysis System is a database-driven project designed to manage and analyse user login activities for a business application. The system tracks user information and their login sessions, enabling the business to derive insights into user engagement, session trends, and activity patterns. The project involves two primary tables: users and logins, with SQL queries developed to address specific business requirements such as identifying inactive users, analysing quarterly session trends, and recognizing top-performing users based on session scores and consistency.

Database Schema:

```
CREATE TABLE users (  
    USER_ID INT PRIMARY KEY,  
    USER_NAME VARCHAR(20) NOT NULL,  
    USER_STATUS VARCHAR(20) NOT NULL  
);  
  
CREATE TABLE logins (  
    USER_ID INT,  
    LOGIN_TIMESTAMP DATETIME NOT NULL,  
    SESSION_ID INT PRIMARY KEY,  
    SESSION_SCORE INT,  
    FOREIGN KEY (USER_ID) REFERENCES USERS(USER_ID)  
);
```

Key Business Questions & Analysis

- Inactive Users (No Logins in Past 5 Months)
 - Retrieve users who have not logged in since January 28, 2024.
- Quarterly User & Session Analysis
 - Calculate the number of unique users and total sessions per quarter, ordered from newest to oldest.
 - Return: Quarter start date, user count, session count.
- Users Active in January 2024 but Not in November 2023
 - Identify users who logged in January 2024 but had no logins in November 2023.

- Quarterly Session Growth Rate
 - Extend the query from Question 2 to include:
 - Session count
 - Previous quarter's session count
 - Percentage change in sessions
 - Return: Quarter start date, session count, previous session count, % change.
- Daily Top Performers (Highest Session Score per Day)
 - For each day, find the user with the highest session score.
 - Return: Date, username, max score.
- Most Consistent Users (Logged in Every Day Since First Login)
 - Identify users who logged in every single day since their first login (assume no breaks).
 - Return: User ID.
- Dates with Zero Logins
 - Find dates where no logins occurred at all.
 - Return: Login date (with zero activity).

Expected Outcomes

- Identification of inactive users for re-engagement campaigns.
- Insights into quarterly user & session trends for business planning.
- Recognition of top-performing users based on session scores.
- Detection of platform downtime (dates with no logins).
- Understanding user retention patterns (consistent vs. sporadic logins).

Technologies & Methods Used

- SQL (Window functions, subqueries, aggregations, date filtering)
- Time-Series Analysis (Quarterly trends, daily activity)
- Data-Driven Decision Making (User engagement strategies, retention analysis)

1. Management wants to see all the users that did not login in past 5 months

```
select max(LOGIN_TIMESTAMP) from logins --2024-06-28 --2024-01-28
```

--WAY 1

```
select USER_ID, MAX(LOGIN_TIMESTAMP) as MAX_DATE
from logins
group by USER_ID
having max(LOGIN_TIMESTAMP) < '2024-01-28'
```

--WAY 2

```
select distinct USER_ID
from logins
where USER_ID not in(select USER_ID
                      from logins
                      where LOGIN_TIMESTAMP > '2024-01-28')
```

Results Messages		
	USER_ID	MAX_DATE
1	1	2024-01-10 07:45:00.000
2	2	2024-01-25 09:30:00.000
3	3	2024-01-25 09:30:00.000
4	4	2023-11-25 09:30:00.000
5	5	2024-01-15 11:00:00.000

2. For the business units quarterly analysis, calculate how many users and how many sessions were at each quarter, order by quarter from newest to oldest.

return first day of the quarter, user count, session count

assumption: considered quarter only irrespective of year

```
select * from logins
```

```
select DATEPART(QUARTER, LOGIN_TIMESTAMP) as QUATER_NO
      , min(LOGIN_TIMESTAMP) as QTR_FIRST_LOGIN
      , count(distinct USER_ID) as USER_COUNT
      , sum(SESSION_SCORE) as SESSION_COUNT
      --, DATETRUNC(QUARTER, min(LOGIN_TIMESTAMP)) as FIRST_QUATER_DATE
from logins
group by DATEPART(QUARTER, LOGIN_TIMESTAMP)
```

Results Messages				
	QUATER_NO	QTR_FIRST_LOGIN	USER_COUNT	SESSION_COUNT
1	1	2024-01-10 07:45:00.000	4	342
2	2	2024-04-12 08:00:00.000	5	695
3	3	2023-07-15 09:30:00.000	5	420
4	4	2023-10-12 08:30:00.000	6	567

3. Display user id that log-in in January 2024 and did not login in on November 2023

--january 2024 1 2 3 5

--novemeber 2023 2 4 6 7

```
select USER_ID
from logins
where MONTH(LOGIN_TIMESTAMP)=1 and YEAR(LOGIN_TIMESTAMP)=2024
and user_id not in (select USER_ID
                    from logins
                    where MONTH(LOGIN_TIMESTAMP)=11 and
YEAR(LOGIN_TIMESTAMP)=2023
                    group by USER_ID)
group by USER_ID
```

Results		Messages	
	USER_ID		
1	1		
2	3		
3	5		

4. Add to the query from question 2 the percentage change in the session form the last quater

return- first day of the quater, session cnt,
session cnt previous, session percentage change.

```
select *, (SESSION_COUNT-CHANGE_LST_QTR)*100/CHANGE_LST_QTR as PERCENTAGE_CHANGE
from(
    select *
    , LAG(SESSION_COUNT,1,SESSION_COUNT) over(order by QUATER_NO) as
CHANGE_LST_QTR
    from(
        select DATEPART(QUARTER, LOGIN_TIMESTAMP) as QUATER_NO
        , min(LOGIN_TIMESTAMP) as QTR_FIRST_LOGIN
        , count(distinct USER_ID) as USER_COUNT
        , COUNT(*) as SESSION_COUNT
        --, DATETRUNC(QUARTER, min(LOGIN_TIMESTAMP)) as
FIRST_QUATER_DATE
        from logins
        group by DATEPART(QUARTER, LOGIN_TIMESTAMP)) A) B
```

Results

Messages

	QUATER_NO	QTR_FIRST_LOGIN	USER_COUNT	SESSION_COUNT	CHANGE_LST_QTR	PERCENTAGE_CHANGE
1	1	2024-01-10 07:45:00.000	4	4	4	0
2	2	2024-04-12 08:00:00.000	5	8	4	100
3	3	2023-07-15 09:30:00.000	5	5	8	-37
4	4	2023-10-12 08:30:00.000	6	7	5	40

5. Display the user that had highest session score(max) for each day
return: date, username, score

```
select LOGIN_DATE
,MAX(SUM_SCORE) as MAXIMUM_SESSION_SCORE
from(
    select USER_ID
    ,cast(LOGIN_TIMESTAMP as date) as LOGIN_DATE
    ,SUM(SESSION_SCORE) as SUM_SCORE
    from logins
    group by USER_ID,cast(LOGIN_TIMESTAMP as date)) A
group by LOGIN_DATE
order by LOGIN_DATE

select *
from(
    select *
    ,ROW_NUMBER() over(partition by DATEE order by SUMSCORE desc) as rn
    from(
        select USER_ID
        ,cast(LOGIN_TIMESTAMP as date) as DATEE
        ,sum(SESSION_SCORE) as SUMSCORE
        from logins
        group by USER_ID, cast(LOGIN_TIMESTAMP as date)) A) B
where rn=1
```

Results Messages		
	LOGIN_DATE	MAXIMUM_SESSION_SCORE
1	2023-07-15	85
2	2023-07-22	90
3	2023-08-10	75
4	2023-08-20	88
5	2023-09-05	82
6	2023-10-12	77
7	2023-11-10	82
8	2023-11-15	80
9	2023-11-18	81
10	2023-11-25	84
11	2023-12-01	84
12	2023-12-15	79
13	2024-01-10	86
14	2024-01-15	78
15	2024-01-25	89
16	2024-04-12	80
17	2024-05-18	82

6. To identify our best users - return the user that had a session on every single day since their first login
(make assumption if needed)
return- userid

```
select *
from(
    select *
    ,DATEDIFF(DAY,MIN_LOGIN_DATE,MAXIMUM_LOGIN_DATE)+1 as DAYS_BETWEEN
    from(
        select USER_ID
        ,MIN(LOGIN_TIMESTAMP) as MIN_LOGIN_DATE
        ,MAX(LOGIN_TIMESTAMP) as MAXIMUM_LOGIN_DATE
        ,COUNT(*) as TOTAL_NO_OF_LOGINS
        from logins
        group by USER_ID) A)B
where DAYS_BETWEEN=TOTAL_NO_OF_LOGINS
```

```
select USER_ID
,MIN(cast(LOGIN_TIMESTAMP as date)) as FIRST_LOGIN
,max(cast(LOGIN_TIMESTAMP as date)) as LAST_LOGIN
,DATEDIFF(DAY,MIN(cast(LOGIN_TIMESTAMP as date))
,max(cast(LOGIN_TIMESTAMP as date)))+1 as DATE_DIFF
,COUNT(USER_ID) as TOTAL_LOGIN
from logins
group by USER_ID
having COUNT(USER_ID)=DATEDIFF(DAY,MIN(cast(LOGIN_TIMESTAMP as
date)),max(cast(LOGIN_TIMESTAMP as date)))+1
```

Results		Messages			
	USER_ID	MIN_LOGIN_DATE	MAXIMUM_LOGIN_DATE	TOTAL_NO_OF_LOGINS	DAYS_BETWEEN
1	10	2024-06-25 15:00:00.000	2024-06-28 15:45:00.000	4	4

7. On what date there were no login at all
--return- login date

```
select min(LOGIN_TIMESTAMP)as MIN_DATE,max(LOGIN_TIMESTAMP)as MAX_DATE
from logins
--2023-07-15
--2024-06-28
```

recursive cte for creating date table from the min date to max date


```
with cte as (
    select cast('2023-07-15' as date) as CAL_DATE
    union all
    select DATEADD(day,1,CAL_DATE) as CAL_DATE
    from cte
    where CAL_DATE<cast('2024-06-28' as date)
)
select * into CAL_TABLE
from cte
option(maxrecursion 500)
```

```

select *
from CAL_TABLE c
left join
    (select cast(LOGIN_TIMESTAMP as date) as REQ_DATE
     from logins) a
    on c.CAL_DATE=a.REQ_DATE
where REQ_DATE is null

```

Results		Messages
	CAL_DATE	REQ_DATE
1	2023-07-16	NULL
2	2023-07-17	NULL
3	2023-07-18	NULL
4	2023-07-19	NULL
5	2023-07-20	NULL
6	2023-07-21	NULL
7	2023-07-23	NULL
8	2023-07-24	NULL
9	2023-07-25	NULL
10	2023-07-26	NULL
11	2023-07-27	NULL
12	2023-07-28	NULL
13	2023-07-29	NULL
14	2023-07-30	NULL
15	2023-07-31	NULL
16	2023-08-01	NULL
17	2023-08-02	NULL

 Query executed successfully.