

Metric Analysis

Sql File Link :

https://drive.google.com/file/d/15G-aXqCKovpYa4LzktjNOfEnY-eswVYp/view?usp=drive_link

Project Description:

The project deals with investigating metrics of a product through the analysis of user engagement, growth, retention, etc. We have been provided with data regarding the sign up, logins, and other engagement of users with the product as well as relevant details regarding users.

Data 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.

Approach: The data given was imported into MySQL workbench using infile command because of the data size. It was then analysed using MySQL queries. MySQL Workbench was used to run various queries and observe the outputs.

Tech-Stack Used: MySQL Workbench 8.0.33 : To Execute the Sql queries.

Result: This project has been rather difficult for me. Initially, it was difficult to load the data. Some of the questions required a lot more search and study than previous projects. However, I finally have an understanding regarding various metric analysis that can be applied to track the growth of a service product.

SQL Queries

```
use metric_spike;  
select * from users;
```

```
alter table users  
modify created_at datetime Not Null;
```

```
CREATE TABLE IF NOT EXISTS `events` (  
  `user_id` bigint(20) NOT NULL,  
  `occured_at` datetime NOT NULL,  
  `event_type` varchar(60) NOT NULL,  
  `event_name` varchar(60) NOT NULL,  
  `location` varchar(60) NOT NULL,  
  `device` varchar(60) NOT NULL,  
  `user_type` bigint(20) NOT NULL  
);
```

```
CREATE TABLE IF NOT EXISTS `email_events` (  
  `user_id` bigint(20) NOT NULL,  
  `occured_at` datetime NOT NULL,  
  `action` varchar(60) NOT NULL,  
  `user_type` bigint(20) NOT NULL  
);
```

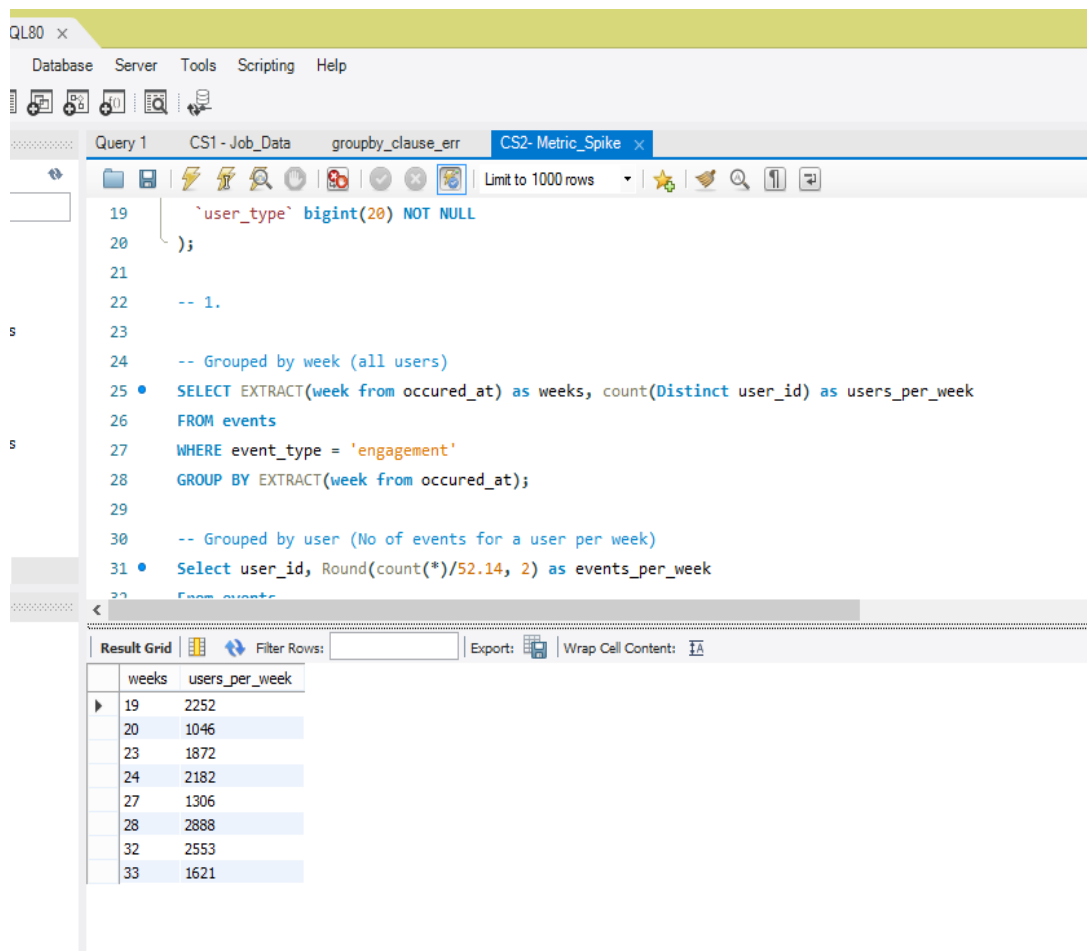
-- A. User Engagement

Measuring the activeness of users on a weekly basis.

-- Grouped by week (all users)

```
SELECT EXTRACT(week from occurred_at) as weeks, count(Distinct user_id) as  
users_per_week  
FROM events  
WHERE event_type = 'engagement'  
GROUP BY EXTRACT(week from occurred_at);
```

Filters engagement data based on week.



The screenshot shows a SQL IDE window with a query editor and a result grid. The query editor contains the following SQL code:

```
19 `user_type` bigint(20) NOT NULL  
20 );  
21  
22 -- 1.  
23  
24 -- Grouped by week (all users)  
25 • SELECT EXTRACT(week from occurred_at) as weeks, count(Distinct user_id) as users_per_week  
26 FROM events  
27 WHERE event_type = 'engagement'  
28 GROUP BY EXTRACT(week from occurred_at);  
29  
30 -- Grouped by user (No of events for a user per week)  
31 • Select user_id, Round(count(*)/52.14, 2) as events_per_week  
32  
33
```

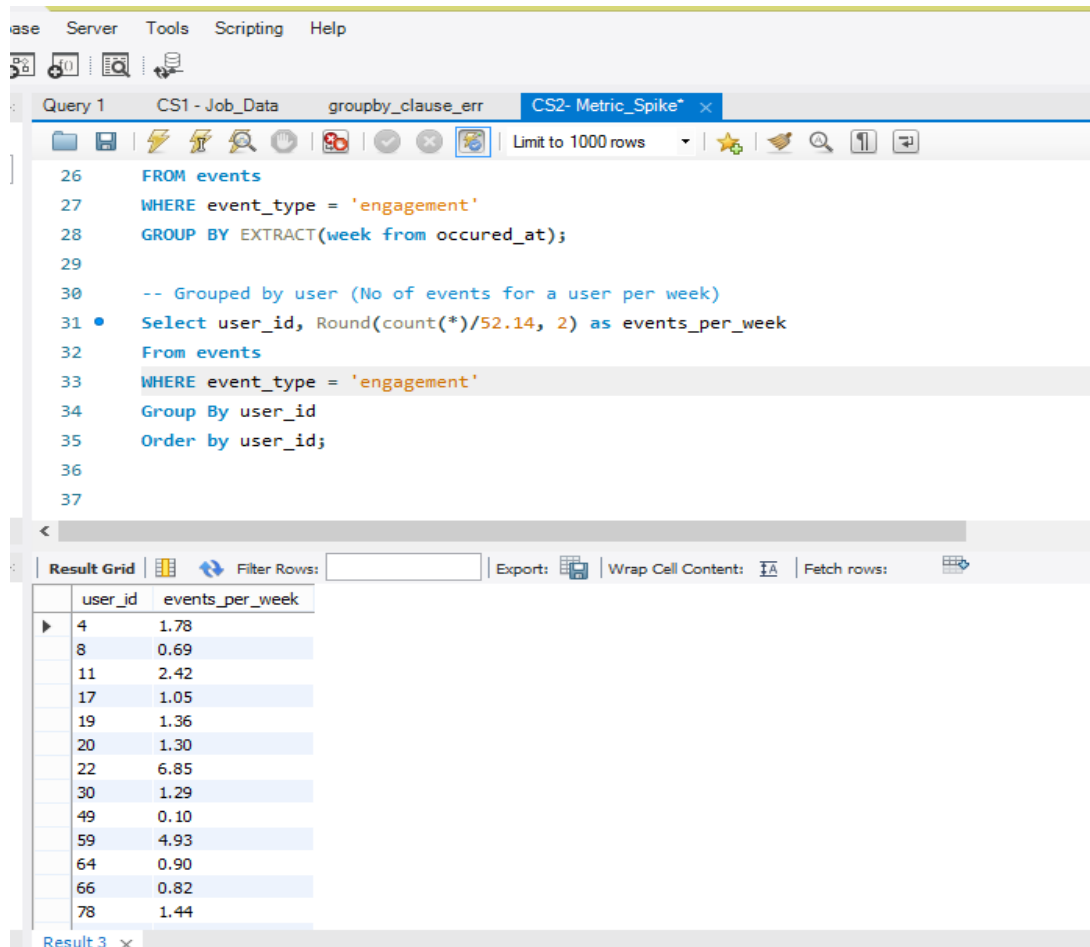
The result grid shows the following data:

weeks	users_per_week
19	2252
20	1046
23	1872
24	2182
27	1306
28	2888
32	2553
33	1621

-- Grouped by user (No of events for a user per week)

```
Select user_id, Round(count(*)/52.14, 2) as events_per_week
From events
WHERE event_type = 'engagement'
Group By user_id
Order by user_id;
```

Filters engagement data based on user.



The screenshot shows a SQL IDE interface with a query editor and a results grid. The query editor contains the following SQL code:

```
26 FROM events
27 WHERE event_type = 'engagement'
28 GROUP BY EXTRACT(week from occurred_at);
29
30 -- Grouped by user (No of events for a user per week)
31 • Select user_id, Round(count(*)/52.14, 2) as events_per_week
32 From events
33 WHERE event_type = 'engagement'
34 Group By user_id
35 Order by user_id;
36
37
```

The results grid displays the following data:

	user_id	events_per_week
▶	4	1.78
	8	0.69
	11	2.42
	17	1.05
	19	1.36
	20	1.30
	22	6.85
	30	1.29
	49	0.10
	59	4.93
	64	0.90
	66	0.82
	78	1.44

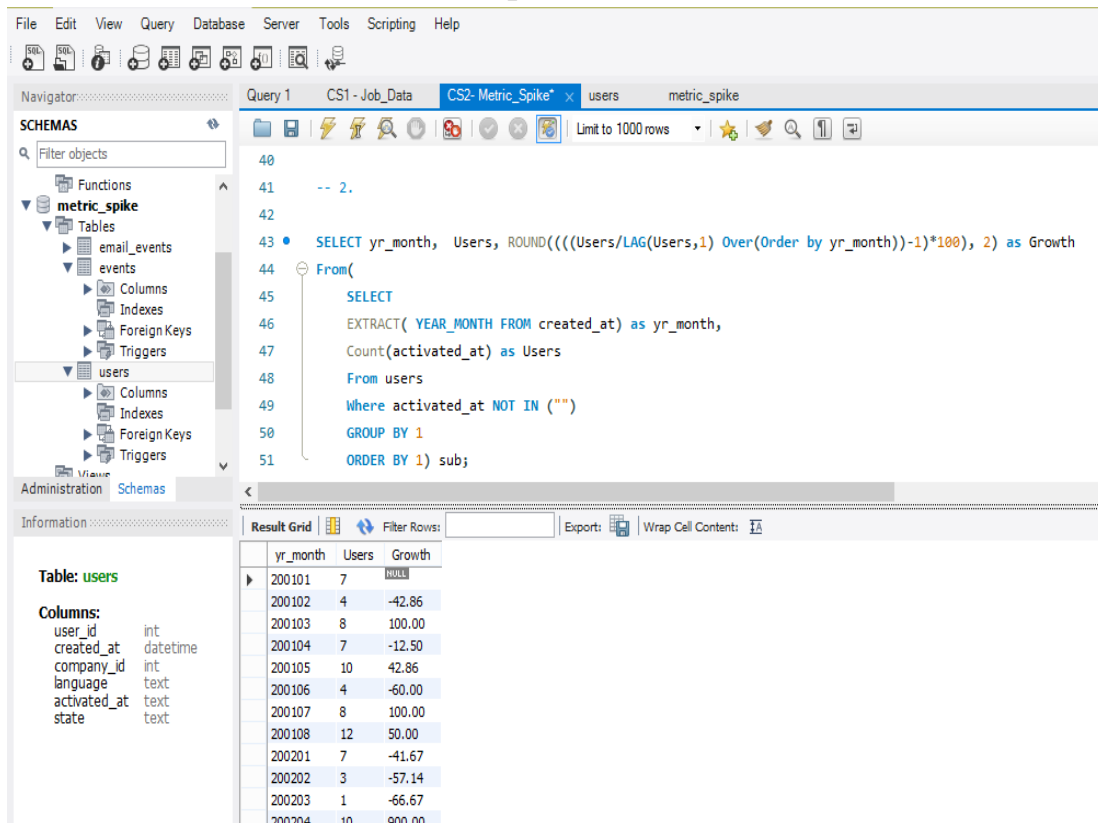
-- 2. User Growth

Analyzing the growth of users over time for product.

```
SELECT yr_month, Users, ROUND((((Users/LAG(Users,1) Over(Order by
yr_month)))-1)*100), 2) as Growth
From(
    SELECT
    EXTRACT( YEAR_MONTH FROM created_at) as yr_month,
    Count(activated_at) as Users
    From users
    Where activated_at NOT IN ("")
    GROUP BY 1
    ORDER BY 1) sub;
```

Uses Year_Month to group users by year-month. Uses Window function lag to compare current months's user count to previous month's.

Formula : ((current_month_users/previous month users) -1)*100



The screenshot shows a database management tool interface. The left sidebar displays a tree view of the database schema, including tables like 'email_events', 'events', 'users', and 'metric_spike'. The main window shows a SQL query in a text editor, which is the same query as provided in the previous blocks. Below the query editor, the 'Result Grid' displays the output of the query. The results are organized into columns: 'yr_month', 'Users', and 'Growth'. The 'Growth' column shows percentage values for each month, with some values being null for the first month of each year.

yr_month	Users	Growth
200101	7	NULL
200102	4	-42.86
200103	8	100.00
200104	7	-12.50
200105	10	42.86
200106	4	-60.00
200107	8	100.00
200108	12	50.00
200201	7	-41.67
200202	3	-57.14
200203	1	-66.67
200204	10	900.00

-- 3. Yearly Retention

Analyzing the retention of users on a yearly basis after signing up for a product.

-- 31 possible first login years

```
SELECT first_year AS "Year_No",
SUM(CASE WHEN year_number = 0 THEN 1 ELSE 0 END) AS "Year 0",
SUM(CASE WHEN year_number = 1 THEN 1 ELSE 0 END) AS "Year 1",
SUM(CASE WHEN year_number = 2 THEN 1 ELSE 0 END) AS "Year 2",
SUM(CASE WHEN year_number = 3 THEN 1 ELSE 0 END) AS "Year 3",
SUM(CASE WHEN year_number = 4 THEN 1 ELSE 0 END) AS "Year 4",
SUM(CASE WHEN year_number = 5 THEN 1 ELSE 0 END) AS "Year 5",
SUM(CASE WHEN year_number = 6 THEN 1 ELSE 0 END) AS "Year 6",
SUM(CASE WHEN year_number = 7 THEN 1 ELSE 0 END) AS "Year 7",
SUM(CASE WHEN year_number = 8 THEN 1 ELSE 0 END) AS "Year 8",
SUM(CASE WHEN year_number = 9 THEN 1 ELSE 0 END) AS "Year 9",
SUM(CASE WHEN year_number = 10 THEN 1 ELSE 0 END) AS "Year 10",
SUM(CASE WHEN year_number = 11 THEN 1 ELSE 0 END) AS "Year 11",
SUM(CASE WHEN year_number = 12 THEN 1 ELSE 0 END) AS "Year 12",
SUM(CASE WHEN year_number = 13 THEN 1 ELSE 0 END) AS "Year 13",
SUM(CASE WHEN year_number = 14 THEN 1 ELSE 0 END) AS "Year 14",
SUM(CASE WHEN year_number = 15 THEN 1 ELSE 0 END) AS "Year 15",
SUM(CASE WHEN year_number = 16 THEN 1 ELSE 0 END) AS "Year 16",
SUM(CASE WHEN year_number = 17 THEN 1 ELSE 0 END) AS "Year 17",
SUM(CASE WHEN year_number = 18 THEN 1 ELSE 0 END) AS "Year 18",
SUM(CASE WHEN year_number = 19 THEN 1 ELSE 0 END) AS "Year 19",
SUM(CASE WHEN year_number = 20 THEN 1 ELSE 0 END) AS "Year 20",
SUM(CASE WHEN year_number = 21 THEN 1 ELSE 0 END) AS "Year 21",
SUM(CASE WHEN year_number = 22 THEN 1 ELSE 0 END) AS "Year 22",
SUM(CASE WHEN year_number = 23 THEN 1 ELSE 0 END) AS "Year 23",
SUM(CASE WHEN year_number = 24 THEN 1 ELSE 0 END) AS "Year 24",
SUM(CASE WHEN year_number = 25 THEN 1 ELSE 0 END) AS "Year 25",
SUM(CASE WHEN year_number = 26 THEN 1 ELSE 0 END) AS "Year 26",
SUM(CASE WHEN year_number = 27 THEN 1 ELSE 0 END) AS "Year 27",
SUM(CASE WHEN year_number = 28 THEN 1 ELSE 0 END) AS "Year 28",
SUM(CASE WHEN year_number = 29 THEN 1 ELSE 0 END) AS "Year 29",
SUM(CASE WHEN year_number = 30 THEN 1 ELSE 0 END) AS "Year 30",
```

```

SUM(CASE WHEN year_number = 31 THEN 1 ELSE 0 END) AS "Year 31"
FROM (
SELECT logins.user_id, logins.login_year, login1.first_year, (logins.login_year -
login1.first_year) AS year_number
FROM
    (SELECT user_id, EXTRACT(Year FROM occurred_at) AS login_year
    From events
    GROUP BY 1, 2
) logins,
    (SELECT user_id, MIN(EXTRACT(Year FROM occurred_at)) AS
first_year
    From events
    GROUP BY 1
) login1
WHERE logins.user_id = login1.user_id
) sub
GROUP BY first_year
ORDER BY first_year;

```

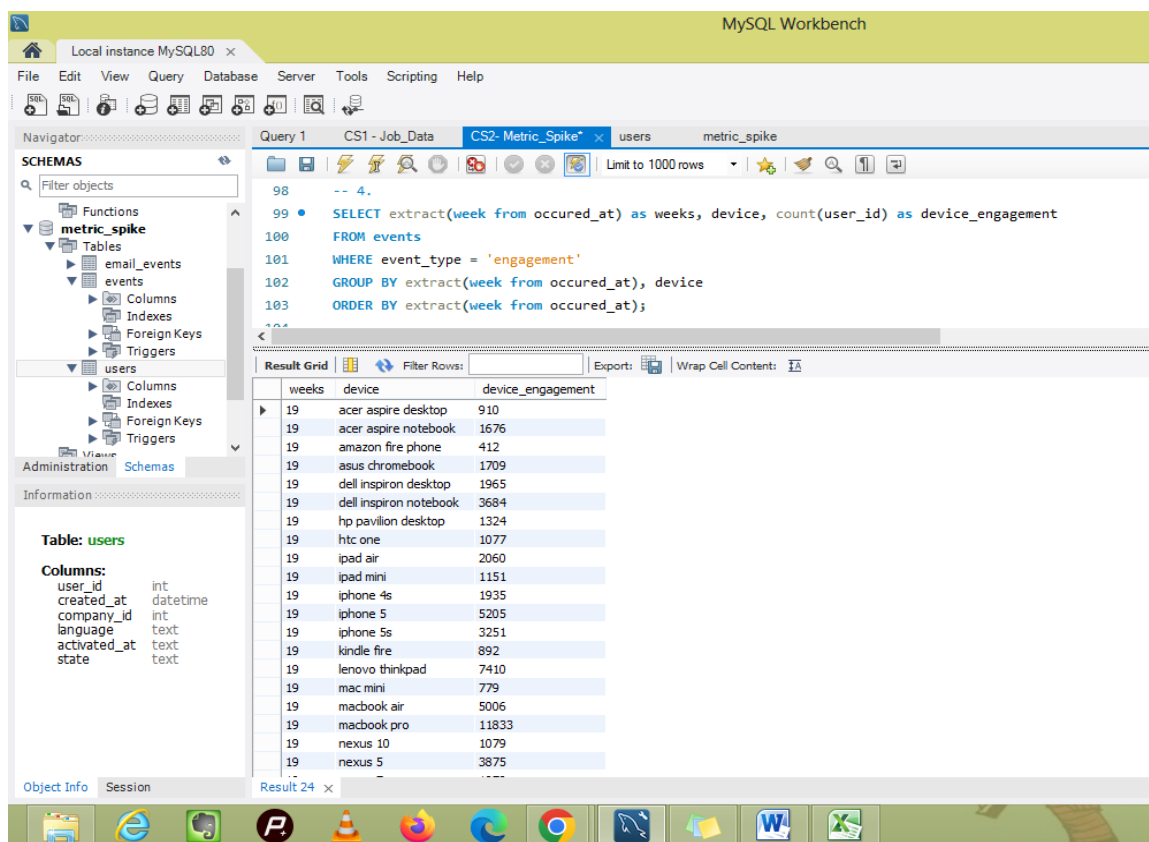
Year_No	Year 0	Year 1	Year 2	Year 3	Year 4	Year 5	Year 6	Year 7	Year 8	Year 9	Year 10	Year 11	Year 12	Year 13	Year 14	Year 15	Year 16	Year 17	Year 18	Year 19
2001	1210	299	285	270	357	299	308	320	293	233	251	257	270	260	272	264	233	233	264	268
2002	883	150	202	178	253	208	178	201	160	183	159	176	136	151	191	129	156	152	180	162
2003	575	116	107	112	140	103	90	103	96	111	109	90	98	92	92	87	88	100	76	96
2004	556	85	97	87	140	99	78	90	75	77	87	68	92	58	99	75	58	84	74	92
2005	422	63	61	64	119	41	53	75	59	56	60	64	41	52	62	58	51	61	66	56
2006	363	37	37	35	90	48	44	51	49	43	41	35	41	38	53	44	39	43	33	33
2007	381	36	48	25	117	50	44	50	49	49	34	51	46	41	42	41	38	35	35	27
2008	310	28	24	17	82	29	30	38	34	30	23	26	23	17	24	18	22	19	20	18
2009	298	15	20	21	95	29	23	41	24	26	26	28	27	20	21	20	18	25	23	24
2010	211	15	17	8	53	12	21	11	14	14	10	11	16	10	12	5	15	6	7	13
2011	313	11	17	17	83	33	15	30	19	24	22	26	21	10	18	18	9	17	12	20
2012	250	16	10	13	82	20	16	26	25	21	14	12	12	14	16	19	20	17	10	8
2013	250	8	6	9	77	21	16	21	12	18	10	7	10	14	18	9	10	4	2	0
2014	287	12	11	6	83	23	18	21	19	18	14	11	16	14	10	19	14	6	0	0
2015	242	11	6	9	102	11	19	23	11	18	11	9	12	13	23	11	7	0	0	0
2016	246	2	11	13	75	21	15	23	12	10	11	15	8	10	8	5	0	0	0	0
2017	163	6	2	4	54	12	10	6	8	7	4	4	8	9	5	0	0	0	0	0
2018	260	14	12	11	86	19	12	20	14	10	17	15	19	7	0	0	0	0	0	0

The table logins collect the logins of users and record the year of login. The table login1 collects the first login of a user. The case-else is used to group users as per their first login year to create a cohort.

-- 4. Weekly Engagement per device

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

```
SELECT extract(week from occurred_at) as weeks, device, count(user_id) as device_engagement
FROM events
WHERE event_type = 'engagement'
GROUP BY extract(week from occurred_at), device
ORDER BY extract(week from occurred_at);
```



The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' tree with 'metric_spike' and 'users' tables. The main window shows a SQL query in the 'Query Editor' and its results in the 'Result Grid'.

Query:

```
-- 4.
SELECT extract(week from occurred_at) as weeks, device, count(user_id) as device_engagement
FROM events
WHERE event_type = 'engagement'
GROUP BY extract(week from occurred_at), device
ORDER BY extract(week from occurred_at);
```

Result Grid:

weeks	device	device_engagement
19	acer aspire desktop	910
19	acer aspire notebook	1676
19	amazon fire phone	412
19	asus chromebook	1709
19	dell inspiron desktop	1965
19	dell inspiron notebook	3684
19	hp pavilion desktop	1324
19	htc one	1077
19	ipad air	2060
19	ipad mini	1151
19	iphone 4s	1935
19	iphone 5	5205
19	iphone 5s	3251
19	kindle fire	892
19	lenovo thinkpad	7410
19	mac mini	779
19	macbook air	5006
19	macbook pro	11833
19	nexus 10	1079
19	nexus 5	3875

The data is filtered on the basis of week and device. The results can be made more presentable using case-else statements.

-- 5. Email Engagement

Analyzing how users are engaging with the email service.

```
Select week,  
Round((weekly_digests/total*100),2) AS "Weekly Digest Rate",  
Round((email_opens/total*100),2) AS "Weekly Digest Rate",  
Round((email_clickthroughs/total*100),2) AS "Weekly Digest Rate",  
Round((reengagement_emails/total*100),2) AS "Weekly Digest Rate"  
FROM (  
SELECT EXTRACT(WEEK FROM occurred_at) as week,  
COUNT(CASE WHEN action = 'sent_weekly_digest' THEN user_id ELSE  
NULL END) as weekly_digests,  
COUNT(CASE WHEN action = 'email_open' THEN user_id ELSE NULL  
END) as email_opens,  
COUNT(CASE WHEN action = 'email_clickthrough' THEN user_id ELSE  
NULL END) as email_clickthroughs,  
COUNT(CASE WHEN action = 'sent_reengagement_email' THEN user_id ELSE  
NULL END) as reengagement_emails,  
COUNT(user_id) AS total  
FROM email_events  
GROUP BY 1 ) sub  
GROUP BY 1  
ORDER BY 1;
```

Inline query is used to fetch the counts of each email event.

MySQL Workbench

Local instance MySQL80 x

File Edit View Query Database Server Tools Scripting Help

Navigator Query 1 CS1-Job_Data CS2-Metric_Spike* x users metric_spike

Limit to 1000 rows

SCHEMAS

Filter objects

Functions

metric_spike

Tables

email_events

events

Columns

Indexes

Foreign Keys

Triggers

users

Columns

Indexes

Foreign Keys

Triggers

Administration Schemas

Information

Table: users

Columns:

user_id int

created_at datetime

company_id int

language text

activated_at text

state text

```
107
108 • Select week,
109 Round((weekly_digests/total*100),2) AS "Weekly Digest Rate",
110 Round((email_opens/total*100),2) AS "Email Open Rate",
111 Round((email_clickthroughs/total*100),2) AS "Email Clickthrough Rate",
112 Round((reengagement_emails/total*100),2) AS "Email Reengagement Rate"
113 FROM (
114 SELECT EXTRACT(WEEK FROM occurred_at) as week,
115 COUNT(CASE WHEN action = 'sent_weekly_digest' THEN user_id ELSE NULL END) as weekly_digests,
116 COUNT(CASE WHEN action = 'email_open' THEN user_id ELSE NULL END) as email_opens,
117 COUNT(CASE WHEN action = 'email_clickthrough' THEN user_id ELSE NULL END) as email_clickthroughs,
118 COUNT(CASE WHEN action = 'sent_reengagement_email' THEN user_id ELSE NULL END) as reengagement_emails,
119 COUNT(user_id) AS total
```

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

	week	Weekly Digest Rate	Email Open Rate	Email Clickthrough Rate	Email Reengagement Rate
▶	17	62.32	21.28	11.39	5.01
	18	63.45	22.24	10.49	3.83
	19	62.16	22.67	11.13	4.04
	20	61.62	22.64	11.43	4.31
	21	63.52	22.82	9.97	3.69
	22	63.59	21.56	10.66	4.19
	23	62.39	22.34	11.18	4.09
	24	61.61	22.92	10.99	4.48
	25	63.77	21.79	10.54	3.90
	26	62.99	22.22	10.61	4.18
	27	62.24	22.49	11.37	3.90
	28	62.92	22.48	10.77	3.83

Object Info Session Result 26 x