OPERATION ANALYTICS AND INVESTIGATING METRIC SPIKE

Presented by: Ankita Taneja

trainity

OVERVIEW

01 Project Description

02 Software Used

03 Case Study 1

04 Tasks 1

05 Results 1

06 Case Study 2

07 Tasks 2

08 Results 2

09 Drive Link

10 Plagirism Report

01 PROJECT DESCRIPTION

Operational Analytics is an essential process concentrated on examining a company's complete operations. This examination aids in pinpointing areas that require enhancement within the organization. As a Data Analyst, you will collaborate with diverse teams, including operations, support, and marketing, to help them extract meaningful insights from the data they gather. A pivotal element of Operational Analytics is the investigation of metric fluctuations. This entails comprehending and clarifying abrupt variations in critical metrics, such as a decline in daily user engagement or a decrease in sales.

As a Data Critic, addressing these inquiries daily is vital, making it important to know how to explore these metric fluctuations. In this design, you will assume the position of Lead Data Critic at a company similar to Microsoft. You will receive various datasets and tables, and your responsibility will be to extract insights from this data to respond to inquiries raised by different departments within the organization. Your objective is to leverage your advanced SQL skills to analyze the data and offer valuable insights that can aid in enhancing the company's operations and comprehending sudden shifts in key metrics.

02 SOFTWARE USED

01

Canva
Graphic Design & Video Editor

Canva is an online graphic design tool that is free to use. You can utilize it to design social media posts, presentations, posters, videos, logos, and much more.

02

MySQL Workbench serves as an integrated visual solution for database architects, developers, and database administrators. It is compatible with Windows, Linux, and Mac OS X.

03

Explore information from around the globe, encompassing websites, pictures, videos, and additional content.

04

Google Drive is a cloud-based service developed by Google. Users can save their files on Google Drive, allowing them to store their data online.

03 CASE STUDY 1

JOB DATA ANALYSIS

Table name: Job_data

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).

Create Database jd

Create Table JOB_DATA

```sql
-- case study 1 - create job data table
create database jd;
use jd;

create table job_data(
    ds date,
    job_id int not null,
    actor_id int not null,
    event varchar(10) not null,
    language varchar(10) not null,
    time_spent int not null,
    org char(2)
    );
```

Output

Action Output

#	Time	Action	Message
1	17:36:51	create database jd	1 row(s) affected
2	17:36:51	use jd	0 row(s) affected
3	17:36:51	create table job_data(ds date, job_id int not null, actor_id int not null, event varchar(10) not null,...	0 row(s) affected

Insert Values into Table JOB_DATA

Display entered values of table

```
1  insert into job_data(ds, job_id, actor_id, event, language, time_spent, org)
2  values('2020-11-15',9,109,'decision','arabic',35,'S'),
3  ('2020-11-16',10,110,'skip','arabic',20,'L'),
4  ('2020-11-17',11,111,'decision','arabic',46,'P'),
5  ('2020-11-19',12,112,'transfer','arabic',38,'L'),
6  ('2020-11-30',13,113,'transfer','arabic',51,'A');
```

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

Date	Joint_Job_Id	ttshr	Jr_phr_pday
2020-11-15	1	0.01	102.86
2020-11-16	1	0.01	180.00
2020-11-17	1	0.01	78.26
2020-11-19	1	0.01	94.74
2020-11-30	1	0.01	70.59

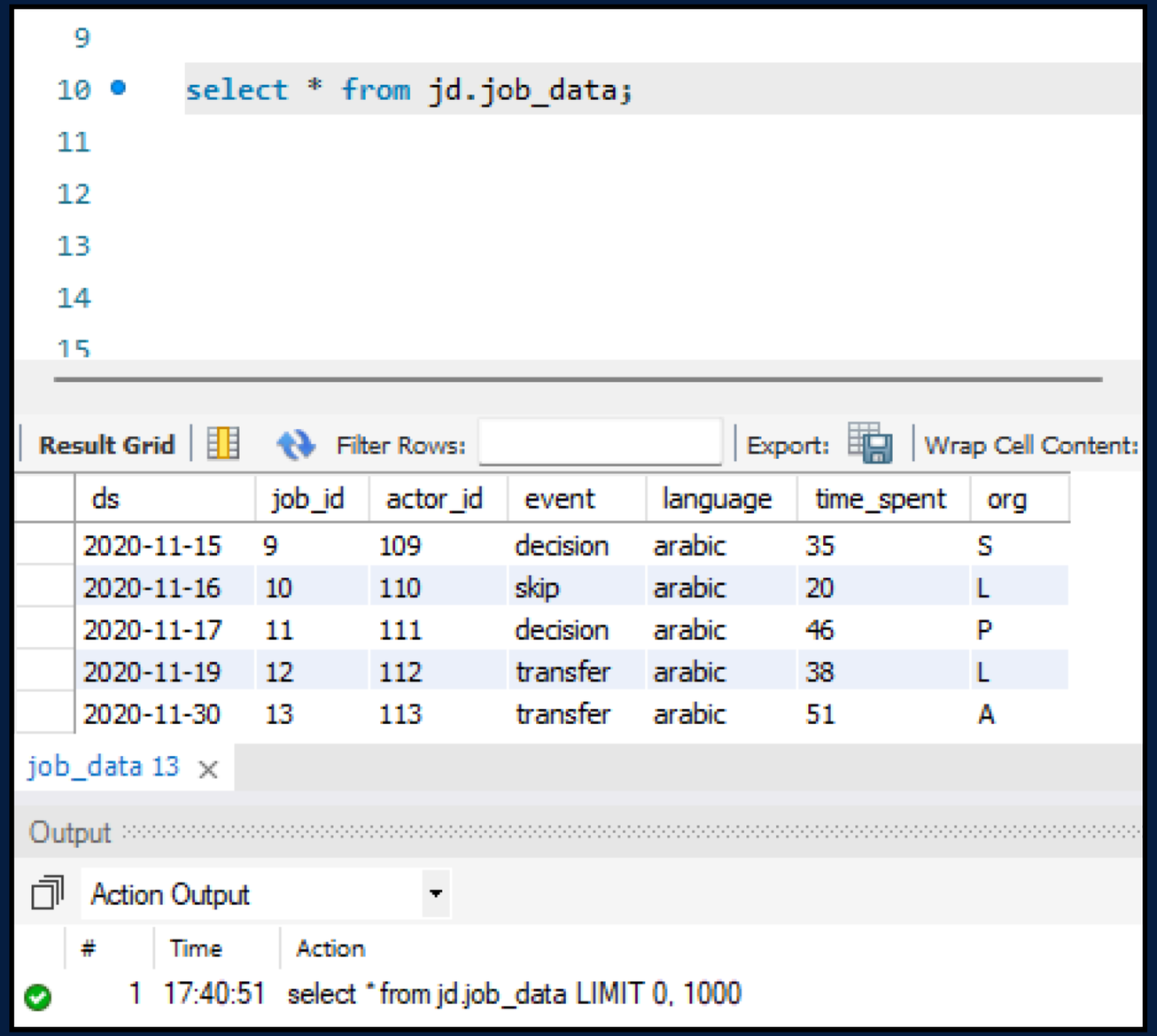
Result 11

Output

Action Output

#	Time	Action	Message
1	17:36:20	insert into job_data(ds, job_id, actor_id, event, language, time_spent, org) values('2020-11-15',9,109,'d...	5 row(s) affe
2	17:36:20	SELECT ds AS Date, COUNT(job_id) As Joint_Job_Id, round((Sum(time_spent)/3600), 2) as ttshr,...	5 row(s) retu

Fetching data from the table



04

TASKS

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.

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.

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.

DUPLICATE ROWS DETECTION

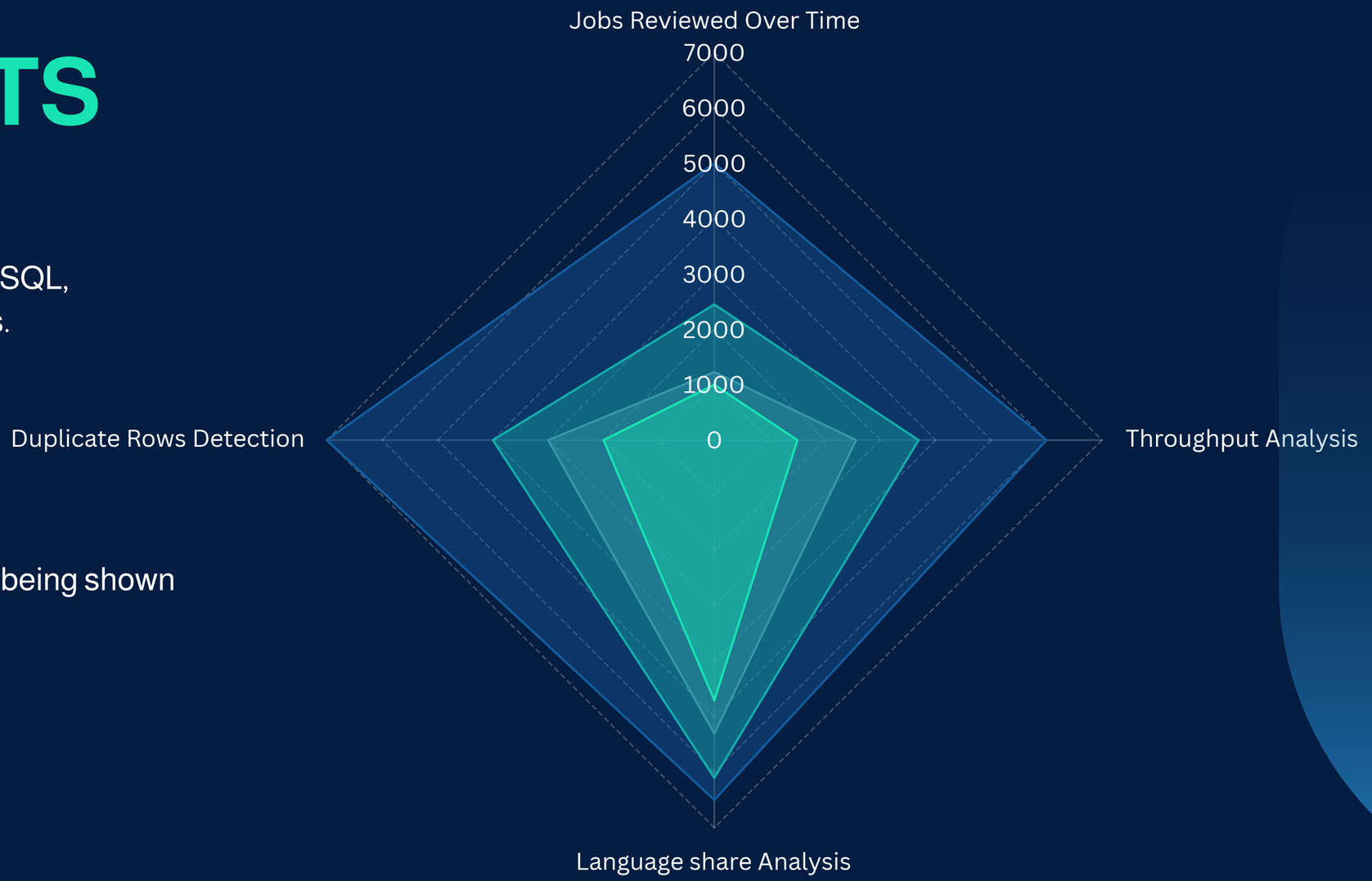
- Objective: Identify duplicate rows in the data.
- Your Task: Write an SQL query to display duplicate rows from the job_data table.

05

RESULTS

- For every task, there is a query in MY SQL, screenshots shared in next few slides.

- For every task, efforts and time (ms) being shown with rough figures on the right side.



5.1

JOBS REVIEWED OVER TIME

Observation::

0.01 Jobs reviewed per hour for each day in the month of November 2020.

The highest job reviewed on 2020-11-16 with 180 per hour.

```
use jd;
SELECT ds as date,
count(job_id) as jjob_id,
round((Sum(time_spent)/3600),2) as ttshr,
ROUND((count(job_id)/(Sum(time_spent)/3600)),2) as jr_phr_pday
from job_data where ds between '2020-11-01' and '2020-11-30'
group by ds order by ds;
```

```
1    -- case study 1 task 1 job reviewed over time
2    use jd;
3    SELECT ds as date, count(job_id) as jjob_id, round((Sum(time_spent)/3600),2) as ttshr,
4    ROUND((count(job_id)/(Sum(time_spent)/3600)),2) as jr_phr_pday from job_data
5    where ds between '2020-11-01' and '2020-11-30'group by ds order by ds;
```

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

date	jjob_id	ttshr	jr_phr_pday
2020-11-15	1	0.01	102.86
2020-11-16	1	0.01	180.00
2020-11-17	1	0.01	78.26
2020-11-19	1	0.01	94.74
2020-11-30	1	0.01	70.59

Result 2

Output

Action Output

#	Time	Action	Message
1	12:47:03	use jd	0 row(s) affected
2	12:47:03	SELECT ds as date, count(job_id) as jjob_id, round((Sum(time_spent)/3600),2) as ttshr, ROUND((coun...	5 row(s) returned

5.2 THROUGHPUT ANALYSIS

Observation:

Daily Average Throughput is varying from 0.01 to 0.08

Weekly Average Throughput is 0.03 events per second..

```sql
use jd;

SELECT ROUND(COUNT(event)/SUM(time_spent),2)
as throughput from job_data;

select ds as dates, ROUND(COUNT(event)/SUM(time_spent),2) as daythroughput
from job_data group by ds order by ds;
```

```sql
-- case study 1 task 2 throughput analysis
use j
SELECt ROUND(COUNT(event)/(SUM(time_spent)),2) as week_throughput
from job_data;

SELECt ds as dates, ROUND(COUNT(event)/(SUM(time_spent)),2) as day_throughput
from job_data group by ds order by ds;
```

Execute the statement under the keyboard cursor

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

dates	day_throughput
2019-01-25	0.06
2019-01-26	0.01
2019-01-27	0.03
2019-01-28	0.05
2019-01-29	0.05
2019-01-30	0.08
2020-11-15	0.03
2020-11-16	0.05
2020-11-17	0.02
2020-11-19	0.03
2020-11-30	0.02

Result Grid | Filter Rows:

week_throughput
0.03

Result 15

Output

Action Output

#	Time	Action	Message
1	17:57:21	SELECt ds as dates, ROUND(COUNT(event)/(SUM(time_spent)),2) as day_throughput from job_data ...	Error Code: 1054. Unknown column 'da' in 'group statement'
2	17:57:33	SELECt ds as dates, ROUND(COUNT(event)/(SUM(time_spent)),2) as day_throughput from job_data ...	11 row(s) returned

Final Observations:

In general, the 7-daytime rolling mean is superior for throughput analysis. A steady production on a trend over time is generated by rolling average throughput, which balances out daily deviations. Short-term events can have an impact on daily measurements, which can vary greatly.

The rolling average, on the other hand, provides a more accurate picture of total performance.

In order to find significant and persistent trends in data, a 7-daytime rolling average is an additional and reliable way to measure throughput.

5.3

LANGUAGE SHARE ANALYSIS

Observation:

The Arabic language with 17.24 percentage is spoken over other languages with same percentage of 10.34 with toal number of 29 counts.

```sql
SELECT language,
round(100* count(*)/total, 2) as percentage,
jd.total from job_data
cross join (select count(*) as total from job_data) as jd
group by language,jd.total;
```

```sql
1   -- case study 1 task 3 SQL query to calculate the percentage
2   -- share of each language over the last 30 days
3   SELECT language,
4   round(100* count(*)/total, 2) as percentage,
5   jd.total from job_data
6   cross join (select count(*) as total from job_data) as jd
7   group by language,jd.total;
```

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

language	percentage	total
Hindi	10.34	29
English	10.34	29
Punjabi	10.34	29
Haryanvi	10.34	29
Bengali	10.34	29
urdu	10.34	29
french	10.34	29
italian	10.34	29
arabic	17.24	29

Result 3

Output

Action Output

#	Time	Action	Message
1	12:47:03	use jd	0 row(s) affected
2	12:47:03	SELECT ds as date, count(job_id) as jjob_id, round((Sum(time_spent)/3600),2) as ttshr, ROUND((cou...	5 row(s) returned
3	13:31:25	SELECT language, round(100* count(*)/total, 2) as percentage, jd.total from job_data cross join (selec...	9 row(s) returned

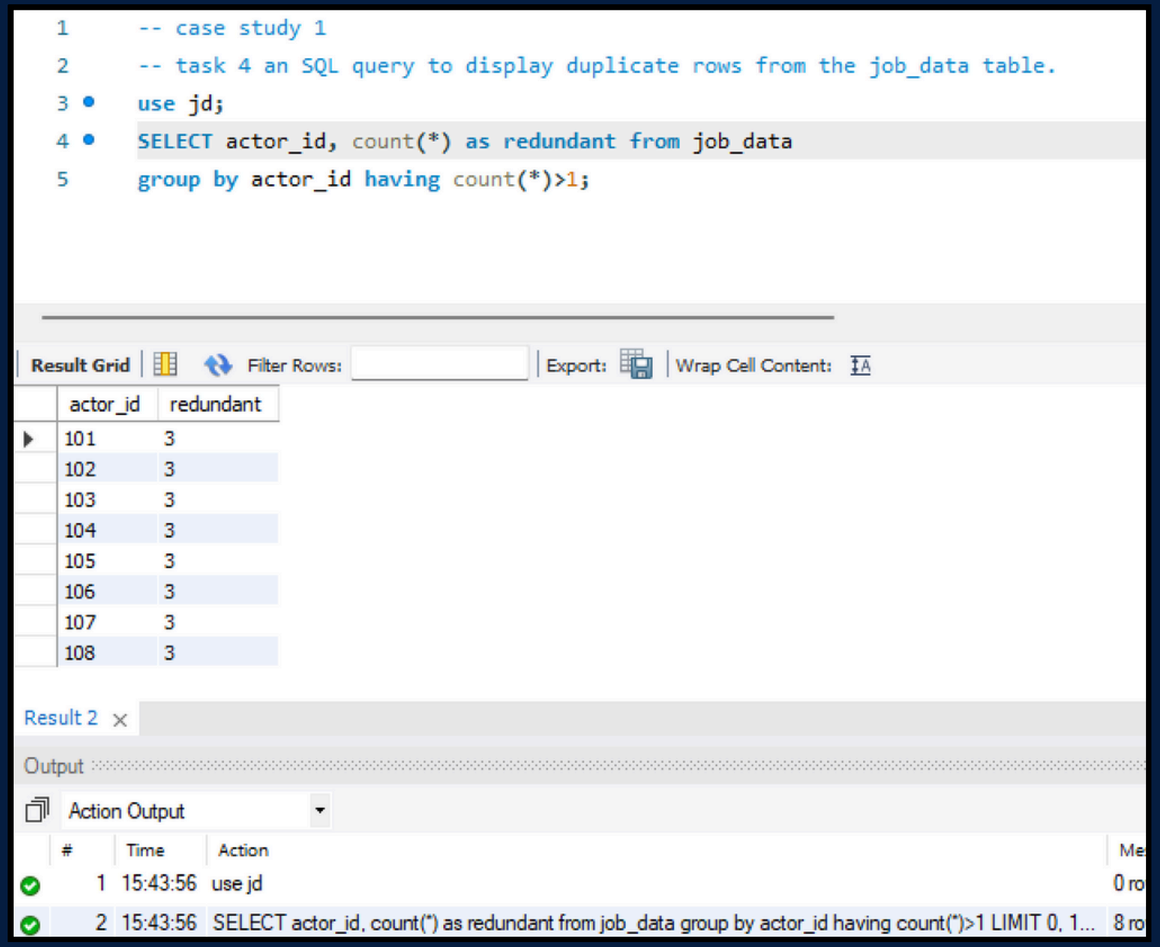
5.4 DUPLICATE ROWS DETECTION

Observation:

The redundant data with actor_id 101 to 108 has been shown at right side from the table job_data.

Next page consists of the non-redundant data from the table job_data to verify the whole fetched data from the table.

```
use jd;
SELECT actor_id, count(*) as redundant from job_data
group by actor_id having count(*)>1;
```



Data Fetching:

The non-redundant data from the table job_data has been shown to the right of the page.

```
select * from job data;
```

```
6 •  select * from job data;
```

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

ds	job_id	actor_id	event	language	time_spent	org
2019-01-26	7	107	skip	french	89	A
2019-01-25	8	108	transfer	italian	16	K
2019-01-30	1	101	skip	Hindi	10	A
2019-01-30	2	102	transfer	English	15	B
2019-01-29	3	103	decision	Punjabi	20	C
2019-01-28	4	104	transfer	Haryanvi	22	Y
2019-01-27	5	105	decision	Bengali	25	J
2019-01-27	6	106	decision	urdu	50	Y
2019-01-26	7	107	skip	french	89	A
2019-01-25	8	108	transfer	italian	16	K
2019-01-30	1	101	skip	Hindi	10	A
2019-01-30	2	102	transfer	English	15	B
2019-01-29	3	103	decision	Punjabi	20	C
2019-01-28	4	104	transfer	Haryanvi	22	Y
2019-01-27	5	105	decision	Bengali	25	J
2019-01-27	6	106	decision	urdu	50	Y
2019-01-26	7	107	skip	french	89	A
2019-01-25	8	108	transfer	italian	16	K
2020-11-15	9	109	decision	arabic	35	S
2020-11-16	10	110	skip	arabic	20	L
2020-11-17	11	111	decision	arabic	46	P
2020-11-19	12	112	transfer	arabic	38	L
2020-11-30	13	113	transfer	arabic	51	A

06

CASE STUDY 2

Investigating Metric Spike

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.

Create table users.

```
1   -- case study 2 Investigating Metric Spike
2   Show databases;
3   use jd;
4   create table users(
5   user_id int,
6   created_at varchar(100),
7   company_id int,
8   language varchar(50),
9   activated_at varchar(100),
10  state varchar(50));
```

Automatic context help is disabled. Use the toolbar t manually get help for the current caret position or t toggle automatic help.

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

Database
ig_clone
information_schema
jd
mysql
performance_schema
sys

Result Grid | Form Editor

Result 5 | Read Only | Context Help | Snippets

Output

Action Output

#	Time	Action	Message	Duration / Fetch
1	15:43:56	use jd	0 row(s) affected	0.000 sec
2	15:43:56	SELECT actor_id, count(*) as redundant from job_data group by actor_id having count(*)>1 LIMIT 0, 1...	8 row(s) returned	0.000 sec / 0.000 sec
3	15:47:26	select * from job_data LIMIT 0, 1000	29 row(s) returned	0.000 sec / 0.000 sec
4	18:29:00	Show databases	6 row(s) returned	0.016 sec / 0.000 sec
5	18:29:00	use jd	0 row(s) affected	0.000 sec
6	18:29:00	select * from users LIMIT 0, 1000	Error Code: 1146. Table 'jd.users' doesn't exist	0.000 sec
7	18:33:27	Show databases	6 row(s) returned	0.000 sec / 0.000 sec
8	18:33:27	use jd	0 row(s) affected	0.000 sec
9	18:33:27	create table users(user_id int, created_at varchar(100), company_id int, language varchar(50), activat...	0 row(s) affected	0.015 sec

Load data to table users >

```
13 • SHOW VARIABLES LIKE 'SECURE_FILE_PRIV';
14   -- select @@secure_file_priv;
15 • LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/users.csv"
16   into table users
17   fields terminated by ','
18   enclosed by '"'
19   lines terminated by '\n'
20   ignore 1 rows;
```

Output

Action Output

#	Time	Action	Message
1	15:45:12	LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/users.csv" into table users...	9381 row(s) affect

```
25 • select * from users;
```

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

user_id	created_at	state	activated_at	company_id	language
0	01-01-2013 20:59	active	01-01-2013 21:01	5737	english
3	01-01-2013 18:40	active	01-01-2013 18:42	2800	german
4	01-01-2013 14:37	active	01-01-2013 14:39	5110	indian
6	01-01-2013 18:37	active	01-01-2013 18:38	11699	english
7	01-01-2013 16:19	active	01-01-2013 16:20	4765	french

users 6

Output

Action Output

#	Time	Action	Mess
3	16:44:45	create table users(user_id int, created_at varchar(100), state varchar(50), activated_at varchar(10...	0 row
4	16:44:50	select * from users LIMIT 0, 1000	0 row
5	16:44:59	SHOW VARIABLES LIKE 'SECURE_FILE_PRIV'	1 row
6	16:45:19	LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/users.csv" into table us...	Error
7	16:46:19	SHOW SESSION VARIABLES LIKE 'lower_case_table_names'	OK
8	16:46:19	SHOW DATABASES	OK
9	16:47:11	SHOW SESSION VARIABLES LIKE 'lower_case_table_names'	OK
10	16:47:11	SHOW COLUMNS FROM `jd`.`users`	OK
11	16:47:29	PREPARE stmt FROM 'INSERT INTO `jd`.`users` (`user_id`,`created_at`,`company_id`,`language`,`ac...	OK
12	16:47:50	DEALLOCATE PREPARE stmt	OK
13	16:50:02	select * from users LIMIT 0, 1000	1000

< Fetch data to table users

Alter the data
&
Update the data

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

user_id	company_id	language	activated_at	state	created_at
0	5737	english	01-01-2013 21:01	active	2013-01-01 20:59:00
3	2800	german	01-01-2013 18:42	active	2013-01-01 18:40:00
4	5110	indian	01-01-2013 14:39	active	2013-01-01 14:37:00
6	11699	english	01-01-2013 18:38	active	2013-01-01 18:37:00
7	4765	french	01-01-2013 16:20	active	2013-01-01 16:19:00
8	2698	french	01-01-2013 04:40	active	2013-01-01 04:38:00

✓	1	11:35:35	ALTER TABLE users add column temp_created_at datetime
✓	2	11:35:39	Alter table users drop column created_at
✓	3	11:35:42	alter table users change column temp_created_at created_at DATETIME

✓	27	17:26:34	ALTER TABLE users add column temp_created_at datetime	0 row(s) affected R
✓	28	17:26:37	update users set temp_created_at = str_to_date(created_at,'%d-%M-%Y, %H:%i')	0 row(s) affected R
✓	29	17:26:56	select * from users LIMIT 0, 1000	1000 row(s) returne
✓	30	17:29:18	Alter table users drop column created_at	0 row(s) affected R
✓	31	17:29:25	alter table users change column temp_created_at created_at DATETIME	0 row(s) affected R
✓	32	17:29:32	select * from users LIMIT 0, 1000	1000 row(s) returne

```
13 •  select * from users;
14
15 •  SHOW VARIABLES LIKE 'SECURE_FILE_PRIV';
16 •  select @@secure_file_priv;
17 •  LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/users.csv"
18    into table users
19    fields terminated by ','
20    enclosed by '"'
21    lines terminated by '\n'
22    ignore 1 rows;
23
24 •  ALTER TABLE users add column temp_created_at datetime;
25 •  set SQL_SAFE_UPDATES = 0;
26 •  update users SET temp_created_at=str_to_date(created_at, '%d-%m-%y %H:%i');
27 •  set SQL_SAFE_UPDATES = 1;
28 •  Alter table users drop column created_at;
29 •  alter table users change column temp_created_at created_at DATETIME;
```

Output

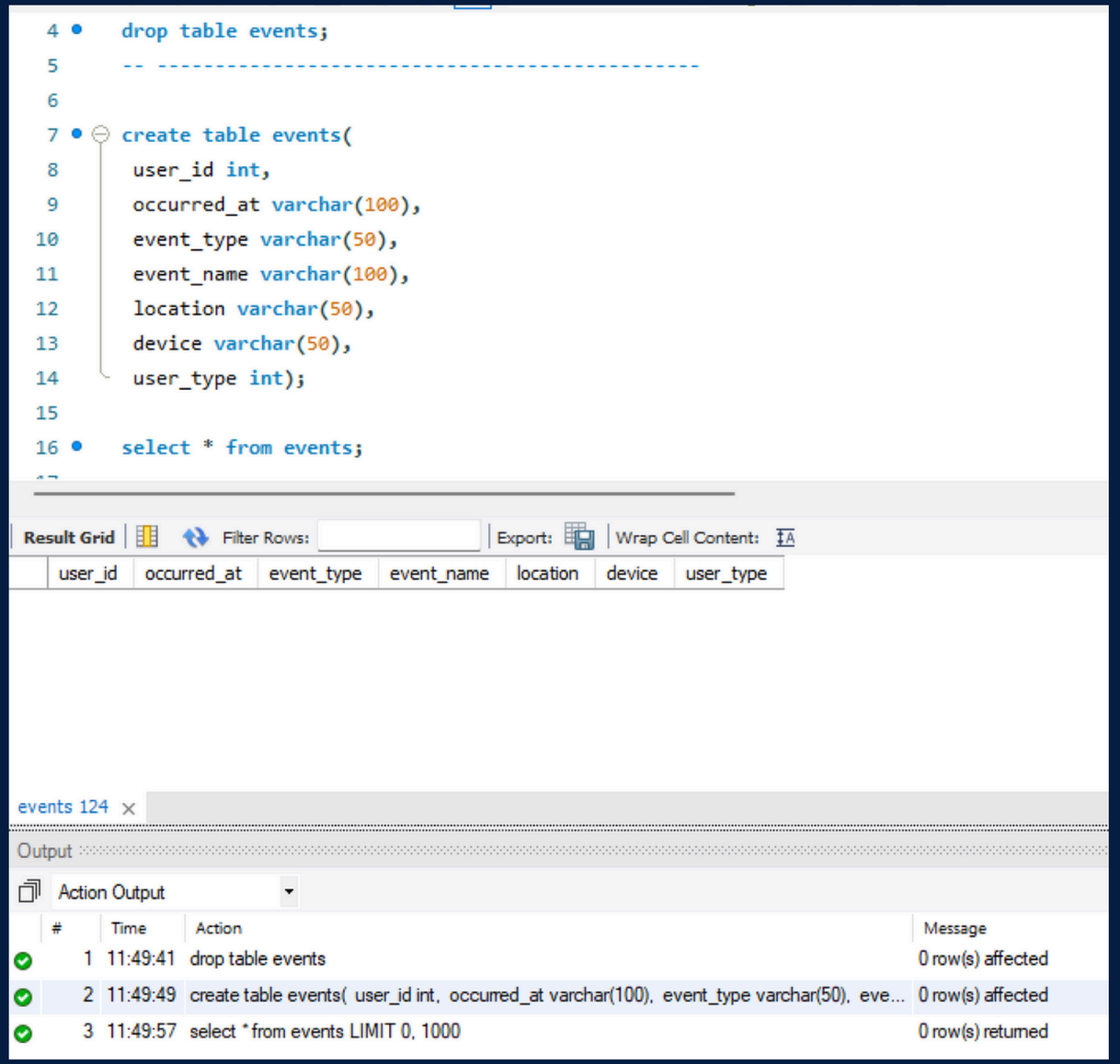
Action Output

	#	Time	Action	Message
✓	28	23:27:25	use jd	0 row(s) affected
✓	29	23:27:25	drop table users	0 row(s) affected
✓	30	23:27:25	create table users(user_id i...	0 row(s) affected
✓	31	23:27:26	select * from users LIMIT 0, ...	0 row(s) returned
✓	32	23:27:26	SHOW VARIABLES LIKE 'S...	1 row(s) returned
✓	33	23:27:26	select @@secure_file_priv ...	1 row(s) returned
✓	34	23:27:26	LOAD DATA INFILE "C:/Pr...	9381 row(s) affected Records: 9381 Deleted: 0 Skipped: 0 Warnings: 0
✓	35	23:28:15	LOAD DATA INFILE "C:/Pr...	9381 row(s) affected Records: 9381 Deleted: 0 Skipped: 0 Warnings: 0

Drop table events.

Create table events.

select data from table events.



Load data to table events >

Fetch data to table events >

```
18 •  SHOW VARIABLES LIKE 'SECURE_FILE_PRIV';
19 •  select @@secure_file_priv;
20 •  LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/events.csv"
21    into table events
22    fields terminated by ','
23    enclosed by '"'
24    lines terminated by '\n'
25    ignore 1 rows;
26 •  select * from events;
```

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

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

events 128

Output

Action Output

#	Time	Action	Message
1	11:55:04	SHOW VARIABLES LIKE 'SECURE_FILE_PRIV'	1 row(s) returned
2	11:55:08	select @@secure_file_priv LIMIT 0, 1000	1 row(s) returned
3	11:55:15	LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/events.csv" i...	325255 row(s) affected Re
4	11:55:27	select * from events LIMIT 0, 1000	1000 row(s) returned

```
26 •  select * from events;
27 •  ALTER TABLE events add column temp_occurred_at datetime;
28 •  set sql_safe_updates=0;
29 •  update events set temp_occurred_at = str_to_date(occurred_at,'%d-%M-%Y, %H:%i');
30
31 •  Alter table events drop column occurred_at;
32 •  alter table events change column temp_occurred_at occurred_at DATETIME;
```

Output

Action Output

#	Time	Action	Message
1	11:59:46	set sql_safe_updates=0	0 row(s) affected
2	12:00:20	Alter table events drop column occurred_at	0 row(s) affected Rec
3	12:00:23	alter table events change column temp_occurred_at occurred_at DATETIME	0 row(s) affected Rec

< alter data to table events

```sql
select * from events;
ALTER TABLE events add column temp_occurred_at datetime;
set sql_safe_updates=0;
update events set temp_occurred_at = str_to_date(occurred_at,'%d-%M-%Y, %H:%i');

Alter table events drop column occurred_at;
alter table events change column temp_occurred_at occurred_at DATETIME;
```

Output

Action Output

#	Time	Action	Message
1	14:50:33	select * from events LIMIT 0, 1000	1000 row(s) retur
2	14:50:37	ALTER TABLE events add column temp_occurred_at datetime	0 row(s) affected
3	14:50:41	set sql_safe_updates=0	0 row(s) affected
4	14:50:44	update events set temp_occurred_at = str_to_date(occurred_at,"%d-%M-%Y, %H:%i")	0 row(s) affected
5	14:50:51	Alter table events drop column occurred_at	0 row(s) affected
6	14:50:55	alter table events change column temp_occurred_at occurred_at DATETIME	0 row(s) affected

Create table email_events.

```sql
create table email_events(
 user_id int,
 occurred_at varchar(100),
 action varchar(50),
 user_type int);

select * from email_events;
```

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

user_id	occurred_at	action	user_type

email_events 1

Output

Action Output

#	Time	Action	Message
1	12:21:21	create table email_events(user_id int, occur...	0 row(s) affected
2	12:21:25	select * from email_events LIMIT 0, 1000	0 row(s) returned

Load data to table events >

Fetch data to table events >

alter data to table events

```
24 •  ALTER TABLE email_events add column temp_occurred_at datetime;
25 •  set sql_safe_updates=0;
26 •  update events set temp_occurred_at = str_to_date(occurred_at,'%d-%M-%Y, %H:%i');
27
28 •  Alter table email_events drop column occurred_at;
29 •  alter table email_events change column temp_occurred_at occurred_at DATETIME;
```

Output

Action Output

#	Time	Action	Message
1	14:58:23	ALTER TABLE email_events add column temp_occurred_at datetime	0 row(s) affected Records: 0 Duplicat
2	14:58:26	set sql_safe_updates=0	0 row(s) affected
3	14:58:30	Alter table email_events drop column occurred_at	0 row(s) affected Records: 0 Duplicat
4	14:58:34	alter table email_events change column temp_occurred_at occurred_at DATETIME	0 row(s) affected Records: 0 Duplicat

```
16 •  select @@secure_file_priv;
17 •  LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/email_events.csv
18    into table email_events
19    fields terminated by ','
20    enclosed by '"'
21    lines terminated by '\n'
22    ignore 1 rows;
23 •  select * from email_events;
```

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

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
0	15-07-2014 09:30	sent_weekly_digest	1

email_events 16

Output

Action Output

#	Time	Action	Mes
1	14:56:18	select * from email_events LIMIT 0, 1000	0 ro
2	14:56:27	SHOW VARIABLES LIKE 'SECURE_FILE_PRIV'	1 ro
3	14:56:39	LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/email_events.csv" into ta...	903
4	14:56:48	select * from email_events LIMIT 0, 1000	100
5	14:56:54	select * from email_events LIMIT 0, 1000	100

07

TASKS

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.

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.

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.

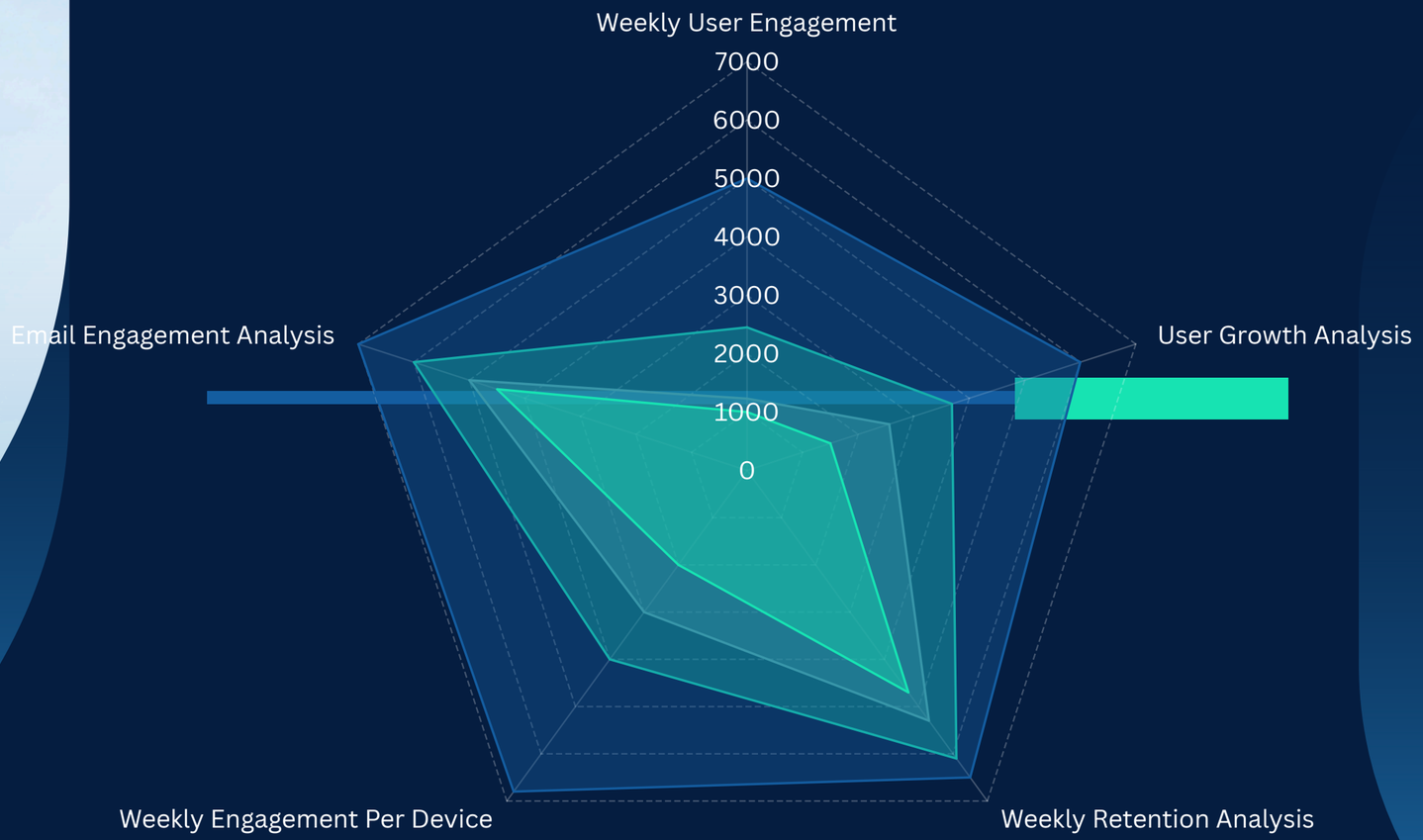
WEEKLY ENGAGEMENT PER DEVICE

- 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.

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.

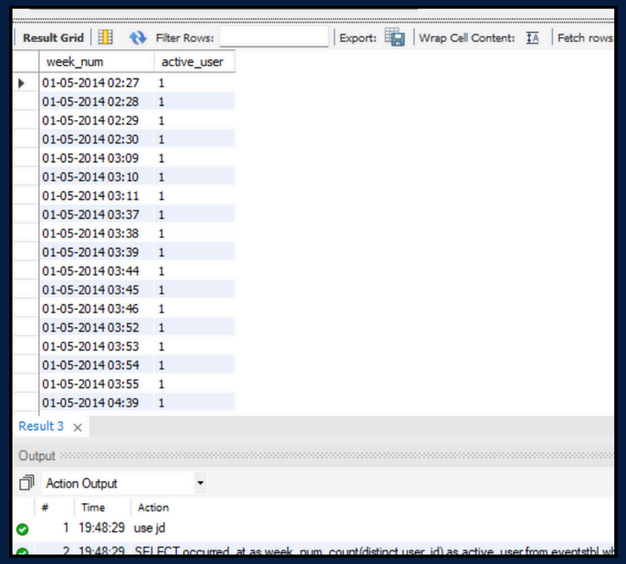
08 RESULTS


Weekly User Engagement
7000
6000
5000
4000
3000
2000
1000
0
User Growth Analysis
Weekly Retention Analysis
Weekly Engagement Per Device
Email Engagement Analysis

8.1 WEEKLY USER ENGAGEMENT

Observation:

Weekly user engagement for active users is 5969 where week number is null as it was distinctly fetched.




```sql
-- Write an SQL query to calculate the weekly user engagement.
use jd;
SELECT
Extract(week from occurred_at) as week_num,
count(distinct user_id) as active_user
from eventstbl where
event_type = 'engagement'
group by week_num order by week_num;
```

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

week_num	active_user
NULL	5969

Result 2

Output

Action Output

#	Time	Action	Message
1	15:36:03	use jd	0 row(s) affected
2	15:36:08	SELECT Extract(week from occurred_at) as week_num, count(distinct user_id) as active_user from ev...	1 row(s) returned

8.2

USER GROWTH ANALYSIS

Observation:

The no of users in count 9381 has cumulative users as 9381.

```sql
with weekly_active_users as(
    select
        Extract(year from created_at) as year_n,
        Extract(week from created_at) as week_n,
        count(distinct user_id) as no_of_users
    from users
    group by year_n, week_n
)
select year_n, week_n, no_of_users,
    sum(no_of_users)
    OVER (order by year_n, week_n)
    AS cusers
from weekly_active_users
order by year_n, week_n;
```

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

ar_n	week_n	no_of_users	cusers
	NULL	9381	9381

```sql
-- case 2 task 2
-- query to calculate the user growth for the product.
with weekly_active_users as(
    select
        Extract(year from created_at) as year_n,
        Extract(week from created_at) as week_n,
        count(distinct user_id) as no_of_users
    from users
    group by year_n, week_n
)
select year_n, week_n, no_of_users,
    sum(no_of_users)
    OVER (order by year_n, week_n)
    AS cusers
from weekly_active_users
order by year_n, week_n;
```

Save the script to a file.

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

year_n	week_n	no_of_users	cusers
NULL	NULL	9381	9381

Result 4

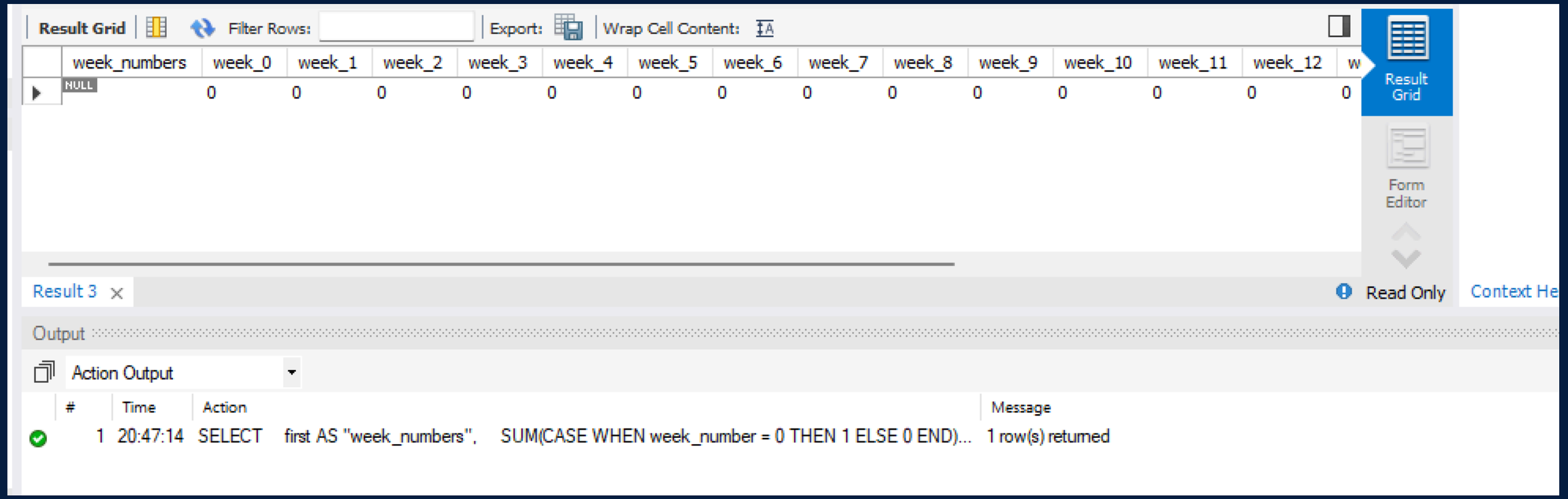
Output

Action Output

#	Time	Action	Message
1	20:10:32	with weekly_active_users as(select Extract(year from created_at) as year_n	Extract(week from c	1 row(s) returned

8.3 WEEKLY RETENTION ANALYSIS

```sql
-- case 2 task 3
-- Write an SQL query to calculate the weekly retention of users based on their sign-up cohort.
SELECT
    first AS "week_numbers",
    SUM(CASE WHEN week_number = 0 THEN 1 ELSE 0 END) AS "week_0",
    SUM(CASE WHEN week_number = 1 THEN 1 ELSE 0 END) AS "week_1",
    SUM(CASE WHEN week_number = 2 THEN 1 ELSE 0 END) AS "week_2",
    SUM(CASE WHEN week_number = 3 THEN 1 ELSE 0 END) AS "week_3",
    SUM(CASE WHEN week_number = 4 THEN 1 ELSE 0 END) AS "week_4",
    SUM(CASE WHEN week_number = 5 THEN 1 ELSE 0 END) AS "week_5",
    SUM(CASE WHEN week_number = 6 THEN 1 ELSE 0 END) AS "week_6",
    SUM(CASE WHEN week_number = 7 THEN 1 ELSE 0 END) AS "week_7",
    SUM(CASE WHEN week_number = 8 THEN 1 ELSE 0 END) AS "week_8",
    SUM(CASE WHEN week_number = 9 THEN 1 ELSE 0 END) AS "week_9",
    SUM(CASE WHEN week_number = 10 THEN 1 ELSE 0 END) AS "week_10",
    SUM(CASE WHEN week_number = 11 THEN 1 ELSE 0 END) AS "week_11",
    SUM(CASE WHEN week_number = 12 THEN 1 ELSE 0 END) AS "week_12",
    SUM(CASE WHEN week_number = 13 THEN 1 ELSE 0 END) AS "week_13",
    SUM(CASE WHEN week_number = 14 THEN 1 ELSE 0 END) AS "week_14",
    SUM(CASE WHEN week_number = 15 THEN 1 ELSE 0 END) AS "week_15",
    SUM(CASE WHEN week_number = 16 THEN 1 ELSE 0 END) AS "week_16",
    SUM(CASE WHEN week_number = 17 THEN 1 ELSE 0 END) AS "week_17",
    SUM(CASE WHEN week_number = 18 THEN 1 ELSE 0 END) AS "week_18"
FROM (
    SELECT
        m.user_id,
        m.login_week,
        n.first,
        m.login_week - n.first AS week_number
    FROM (
        SELECT
            user_id,
            EXTRACT(WEEK FROM occurred_at) AS login_week
        FROM eventstbl
        GROUP BY user_id, login_week
    ) m
    JOIN (
        SELECT
            user_id,
            MIN(EXTRACT(WEEK FROM occurred_at)) AS first
        FROM eventstbl
        GROUP BY user_id
    ) n
    ON m.user_id = n.user_id
) sub
GROUP BY first
ORDER BY first;
```



Observation:

The week wise representation is been shown in the table.

8.4 WEEKLY ENGAGEMENT PER DEVICE

Observation:

The weekly engagement of devices and the count of users engaged are shown.

As the table shows, the devices with respect to user id count.

```
3  SELECT
4      EXTRACT(WEEK FROM occurred_at) AS week,
5      EXTRACT(YEAR FROM occurred_at) AS year,
6      device,
7      COUNT(DISTINCT user_id) AS count
8  FROM     eventstbl
9  WHERE     event_type = 'engagement'
10 GROUP BY     week, year, device
11 ORDER BY     week, year, device;
```

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

week	year	device	count
NULL	NULL	acer aspire desktop	165
NULL	NULL	acer aspire notebook	300
NULL	NULL	amazon fire phone	76
NULL	NULL	asus chromebook	293
NULL	NULL	dell inspiron desktop	301
NULL	NULL	dell inspiron notebook	562
NULL	NULL	hp pavilion desktop	283
NULL	NULL	htc one	160

Result 2

Output

Action Output

#	Time	Action		Message
1	15:41:01	SELECT	EXTRACT(WEEK FROM occurred_at) AS week,	EXTRACT(YEAR FROM occurred_at... 26 row(s) returned

8.5

EMAIL ENGAGEMENT ANALYSIS

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
16693	06-08-2014 11:13	sent_weekly_digest	1
0	03-06-2014 09:30	sent_weekly_digest	1
16693	06-08-2014 11:14	email_open	1
0	10-06-2014 09:30	sent_weekly_digest	1
16693	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
16693	06-08-2014 11:15	sent_weekly_digest	1
0	01-07-2014 09:30	sent_weekly_digest	1
0	08-07-2014 09:30	sent_weekly_digest	1
16693	06-08-2014 11:17	engagement	1
0	29-07-2014 09:30	sent_weekly_digest	1
0	29-07-2014 09:30	email_open	1
16693	10-08-2014 16:49	sent_weekly_digest	1
0	12-08-2014 09:30	sent_weekly_digest	1
0	19-08-2014 09:30	email_open	1
0	26-08-2014 09:30	sent_weekly_digest	1
16693	10-08-2014 16:50	sent_weekly_digest	3

email_events 7

```sql
-- case 2 task 5
-- Write an SQL query to calculate the email engagement metrics.

select * from email_events;
select count(distinct user_id),
count(distinct (action)) as act
from email_events;
```

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

count(distinct user_id)	act
5008	5

Result 6

Output

Action Output

#	Time	Action	Message
1	17:10:02	select count(distinct user_id), count(distinct (action)) as act from email_events LIMIT 0, 1000	1 row(s) returned
2	17:10:10	select * from email_events LIMIT 0, 1000	1000 row(s) returned
3	17:10:15	select count(distinct user_id), count(distinct (action)) as act from email_events LIMIT 0, 1000	1 row(s) returned

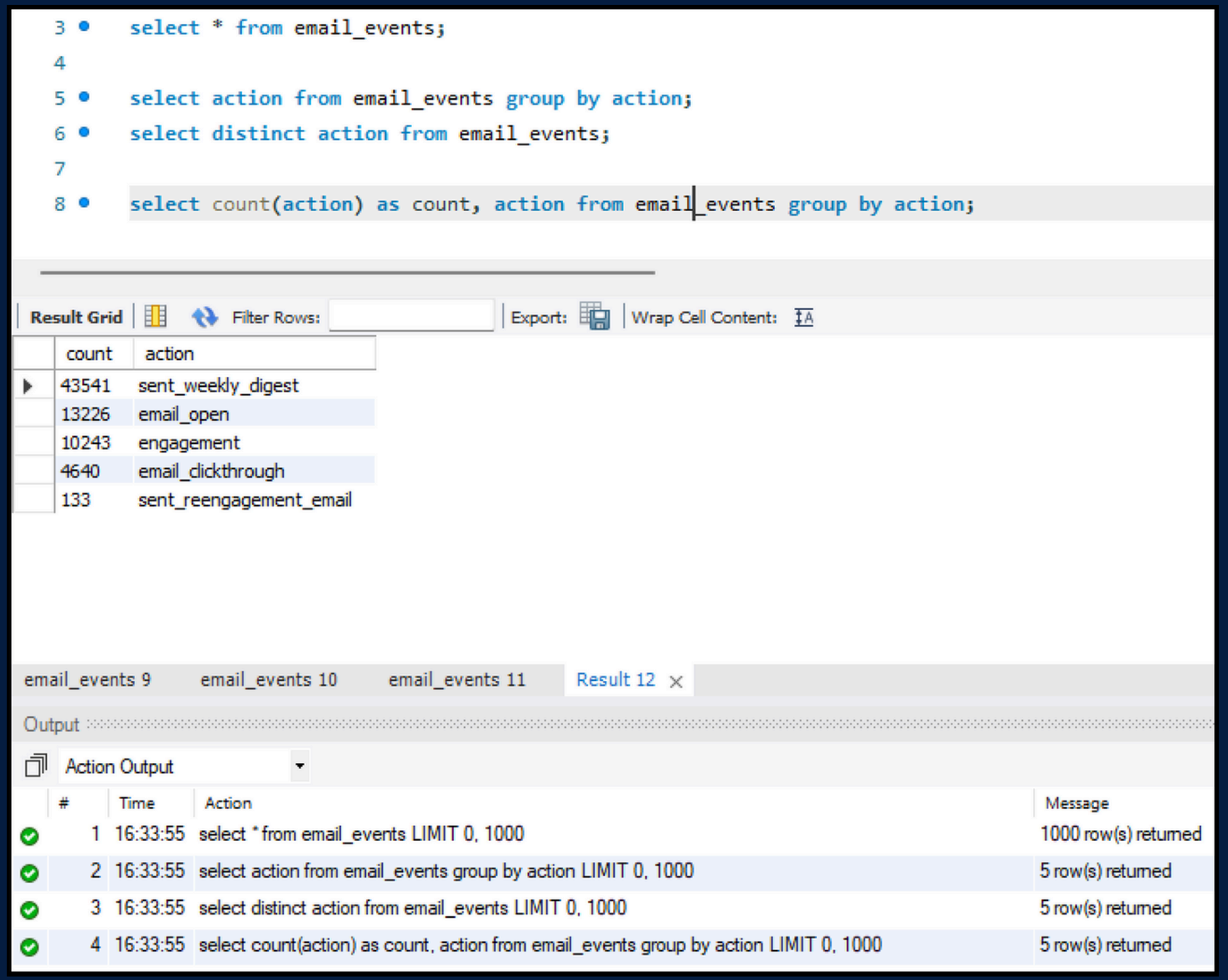
Observation:

Previous slide:

- Fetching the data from table email_events.
- Fetching distinct user_id count and action count from the table.

Current slide:

- fetching the count of action and action from table grouped by action event.



09

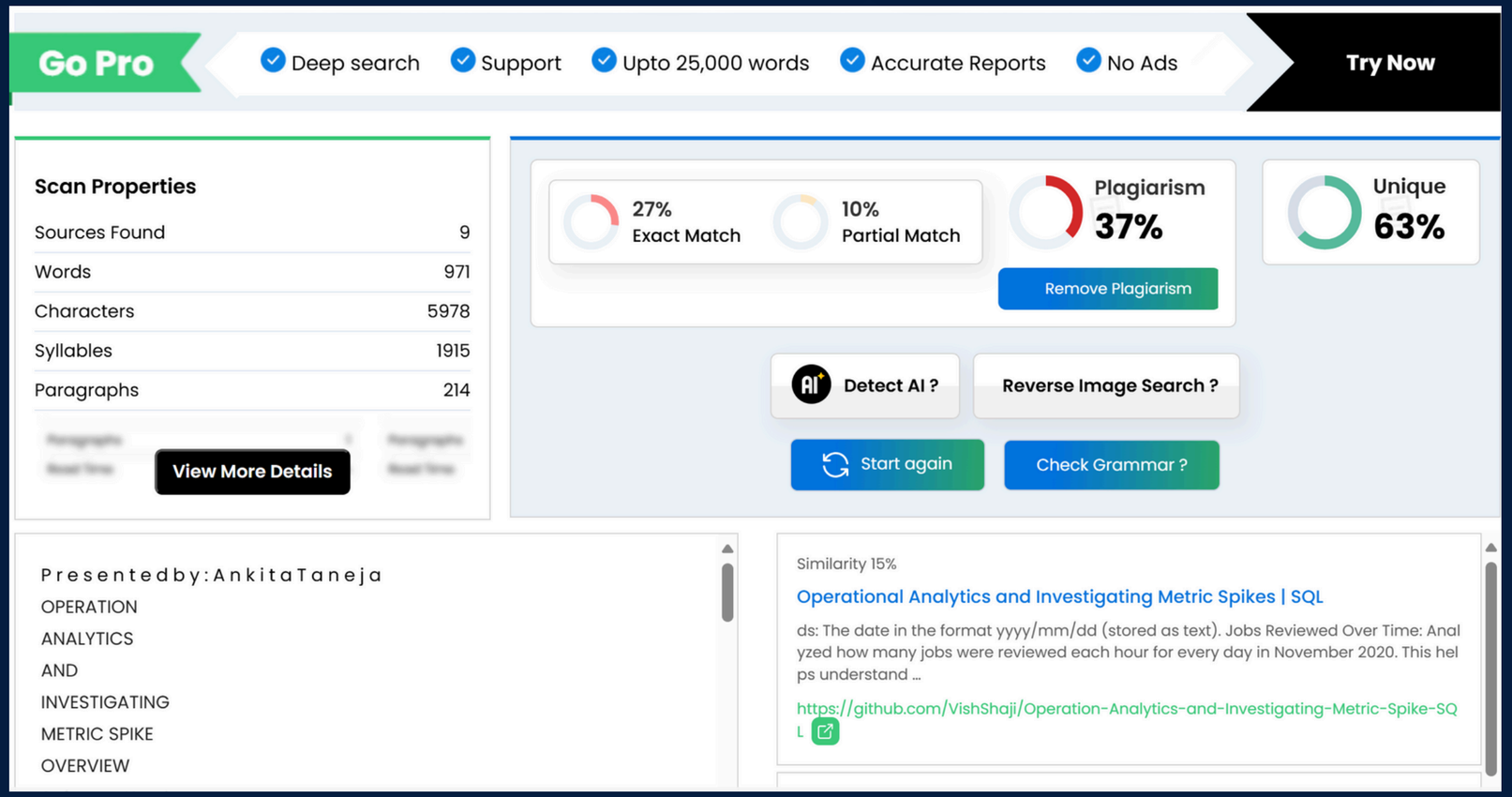
DRIVE LINK

PROJECT 3

https://drive.google.com/file/d/155PapdkkgO9MbUwTaLuJh_wTYGJ8JeHV/view?usp=drive_link

10

PLAGIRISM CHECK



OUR TEAM

ANKITA TANEJA

Presentation Designer

THANK YOU

CONNECT WITH US.

+91-7011334048

Venusgirlatwork@gmail.com