# CA675 - Assignment 1

Minh Nhat Vu
DCU student ID: 20214265
Email: minh.vu3@mail.dcu.ie

Link Github: https://github.com/NhatVu/DCU-CA675-Assignment1

## Task 1

Dataset is obtained from https://data.stackexchange.com/. Because of the restriction, the site is only allowed to get 50000 records per query. To get enough 200,000 records, the query must be done multiple times.

The first query will count the number of results to records. When it nearly reaches 50000, I will retrieve the result.

Sample counting query:

```
select count(*)
from posts
where posts.ViewCount <= 80000 and posts.ViewCount > 57000;
```

Then, using the query below to get the data

```
select *
from posts
where posts.ViewCount <= 80000 and posts.ViewCount > 57000;
```

The last query will exceed 200,000 a bit. I use pandas to remove the abundance.
For my dataset, I divide ViewCount into chunks that list below

```
+Ifn, 128k, 80k, 57k, 44k, 41k
```

**Preprocessing data:**

Because data contains multiple lines or commas in a column. This leads to errors when importing data to the Hive table. We need to remove all lines in columns, then use OpenCSVSerde to parse CSV files correctly. But the disadvantage when using OpenCSVSerde is it converts all columns to the String field.

Check file *Task1_preprocess.py* to know more about code for removing line breaks. One important note is that different OSs use different line break characters.

**Dataset description:**

There are many fields for this dataset. For the sake of this assignment, I will care about some relevant fields. It includes OwnerUserID, Score, Title, Body

Otherwise, there are many interesting fields like ViewCount, PostTypeId (the question is 1 and answer is 2), tags, answer count, comment count

Note: OwnerUserId is empty because use has been disabled account.

# Task 2 & 3

In this task, I will use Hive. These tasks look like ad hoc queries. Furthermore, it's easier when processing structured data with HiveQL language.

Link code: https://github.com/NhatVu/DCU-CA675-Assignment1/blob/main/code/Task2%263.sql

### 2.2.1. The top 10 posts by score

Step descriptions: sort score by desc, then select posts based on that sorted order.

Query:

```
select id, cast(score as int), title
from posts
order by cast(score as int) desc
limit 10;
```

### 2.2.2. The top 10 users by total post score

Step description: First, group by OwnerUserId, then calculate the sum of the score for each group. Next, sort by total score. Finally, query based on that sorted order.
Query:

```
select OwnerUserId, sum(cast(score as int)) as s
from posts
where OwnerUserId != ""
group by OwnerUserId
order by s desc
limit 10;
```

### 2.2.3. The number of distinct users, who used the word “cloud” in one of their Posts

Task description:

This required finding the word “cloud” in both title and body. The original form looks like (OwnerUserId, title + body).
Then, I split title + body into word by space and comma. I have another table like (OwnerUserId, word).
Finally, I count distinct OwnerUserId who has the word “cloud”.

Query:

```
select count(distinct(OwnerUserId))
from posts
lateral view explode(split(concat(title," ", body), ' |,')) lateralTable as
word
where word = "cloud";
```

## Task 4

Link Github: https://github.com/NhatVu/DCU-CA675-Assignment1/blob/main/code/Task4.sql

First, we need to calculate TF-IDF. Input has form: (OwnerUserId, title + body)

We need to split title and body to word to achieve this schema: (OwnerUserId, word)

```
create or replace view flatten_word
as
select cast(OwnerUserId as int) as OwnerUserId, trim(word) as word
from posts
lateral view explode(split(concat(title, " ", body), ' |,')) lateralTable
as word
where word != "";
```

Next, we calculate TF and save it to view tf. For simplicity, I choose log normalization. I will create tf view like (*OwnerUserId, word, termFreq)*

```
create or replace view tf
as
select OwnerUserId, word, log(10, count(*) + 1) as termFreq
from flatten_word
group by OwnerUserId, word;
```

Then, I calculate IDF and produce schema: (*word, idf)*

```
create or replace view idf
as
```

```
select word, numberAppearInUser, numberUser, log(10,
numberUser/(numberAppearInUser + 1)) + 1 as idf
from (
select word, count(distinct(OwnerUserId)) as numberAppearInUser
from flatten_word
group by word) as a
cross join (
select count(distinct (OwnerUserId)) as numberUser
from flatten_word) as b;
```

Finally, I combine the tf view and the idf view to have the final result. Because calculating tf idf cost a lot of time, I created a tf idf table to save this result.

```
create external table if not exists tfidf (
 OwnerUserId int,
 word String,
 tfidf double
);
```

```
INSERT OVERWRITE table tfidf
select OwnerUserId, tf.word, tf.termFreq * idf.idf as tfidf
from tf
join idf
on tf.word = idf.word;
```

For getting the top 10 words for each user in the top 10, we use this query. It contains several steps. First, I will filter out all rows that do not belong to the top 10 users. Then, I partition the dataset by OwnerUserId and apply the rank function on it, with descending order by tfidf score. Finally, filter row with rank less than 10.

```
select * from (
select OwnerUserId, word, tfidf, rank() over(partition by OwnerUserId order
by tfidf desc) as rn
from tfidf as T
where T.OwnerUserId in (
 select cast(OwnerUserId as int) from (
 select OwnerUserId, sum(cast(score as int)) as s
from posts
where OwnerUserId != ""
group by OwnerUserId
order by s desc
limit 10
```

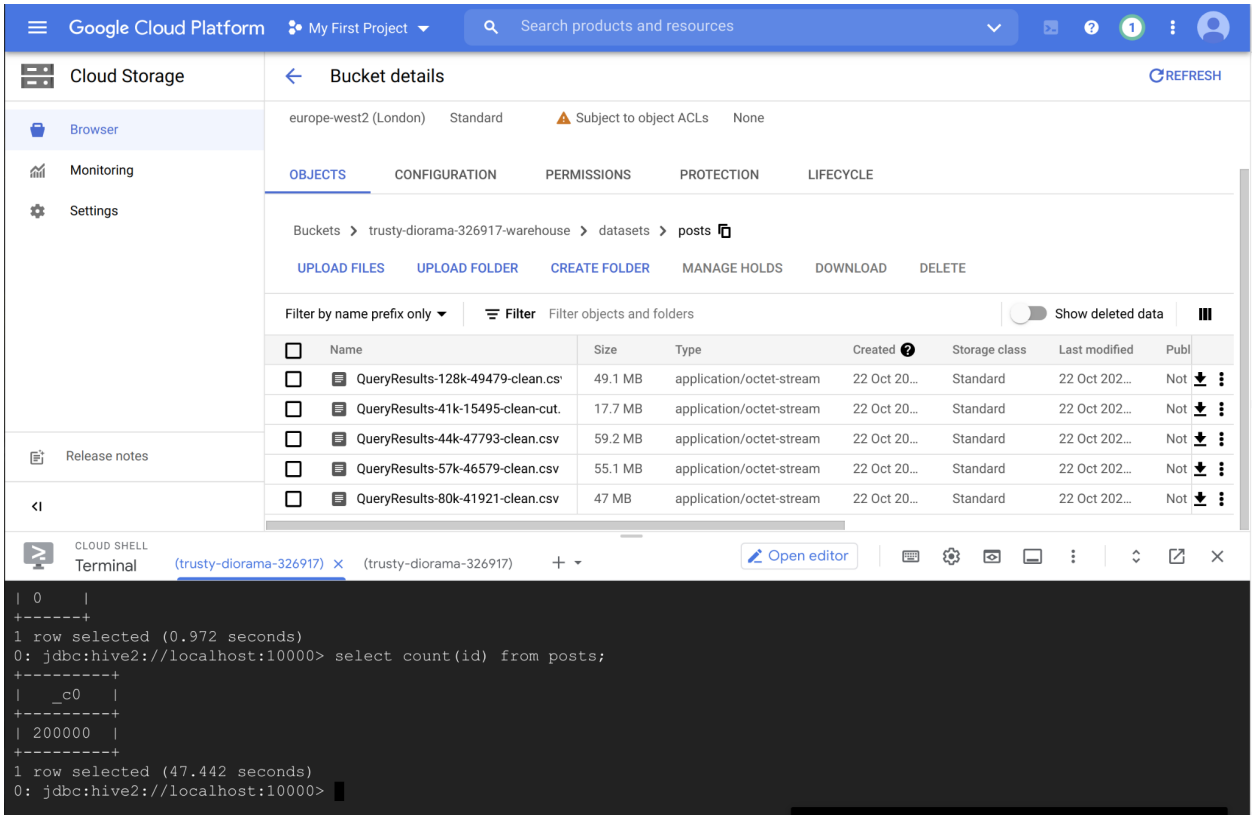
```
) as B
)
) as A
where A.rn <= 10;
```

# Appendix

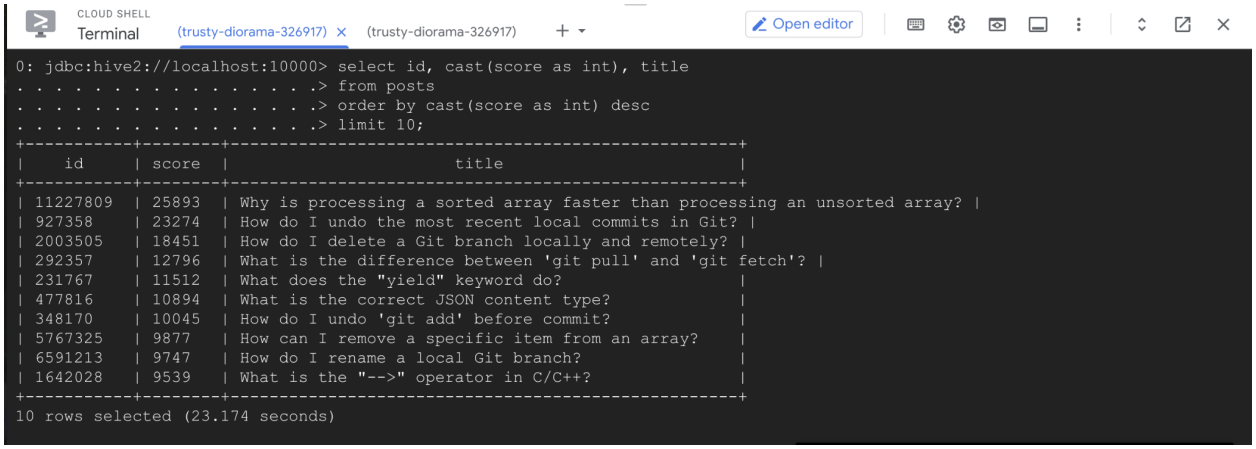
This section provides screenshots for each task above.

## Screenshot task 1

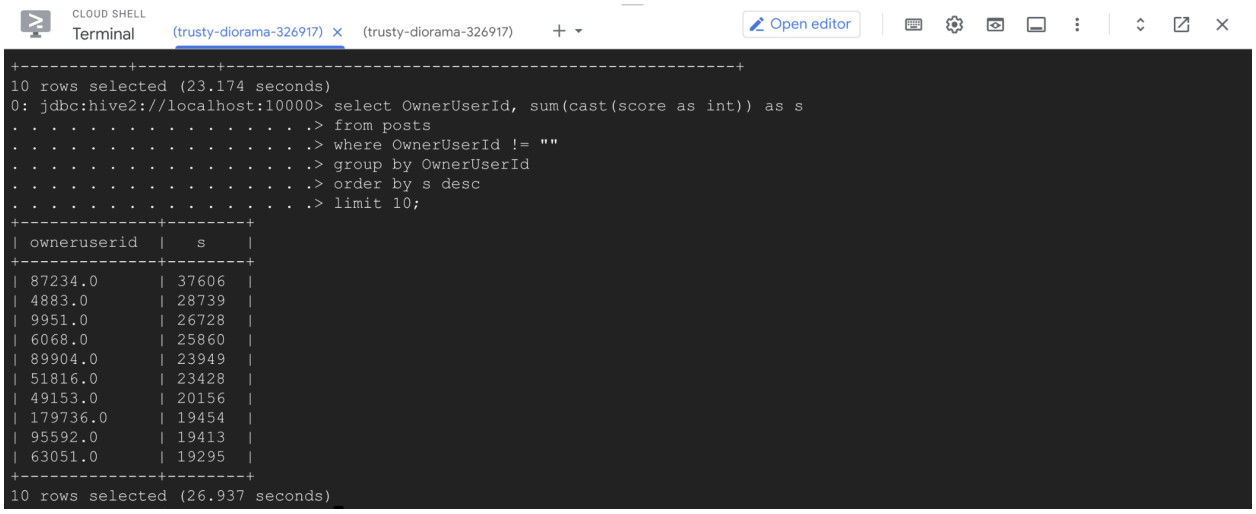
Upload posts dataset to cloud. Then using count the number of records




## Screenshot task 2.2.1



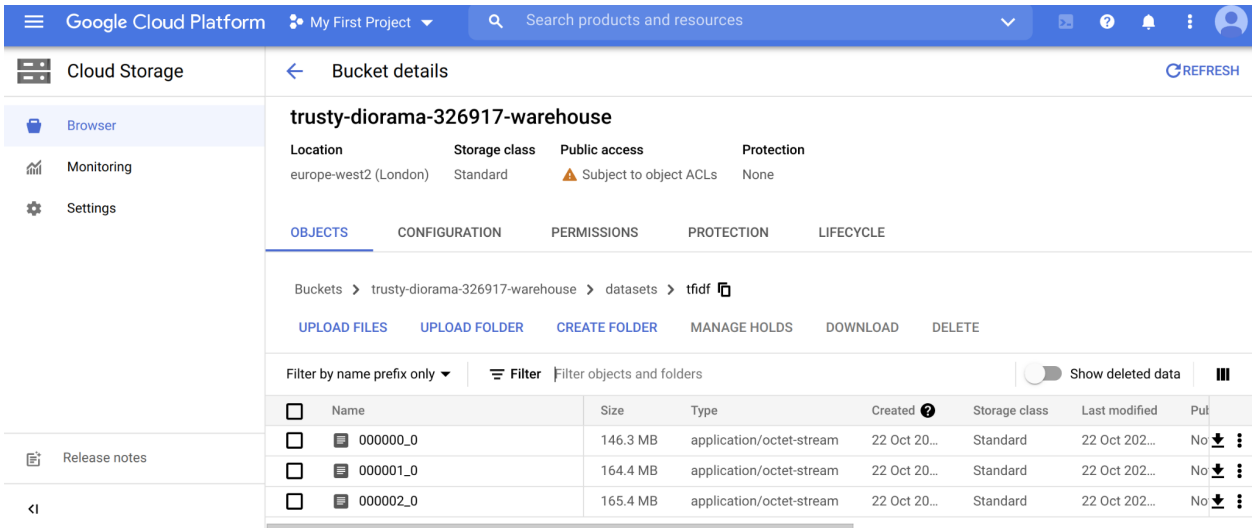
## Screenshot task 2.2.2



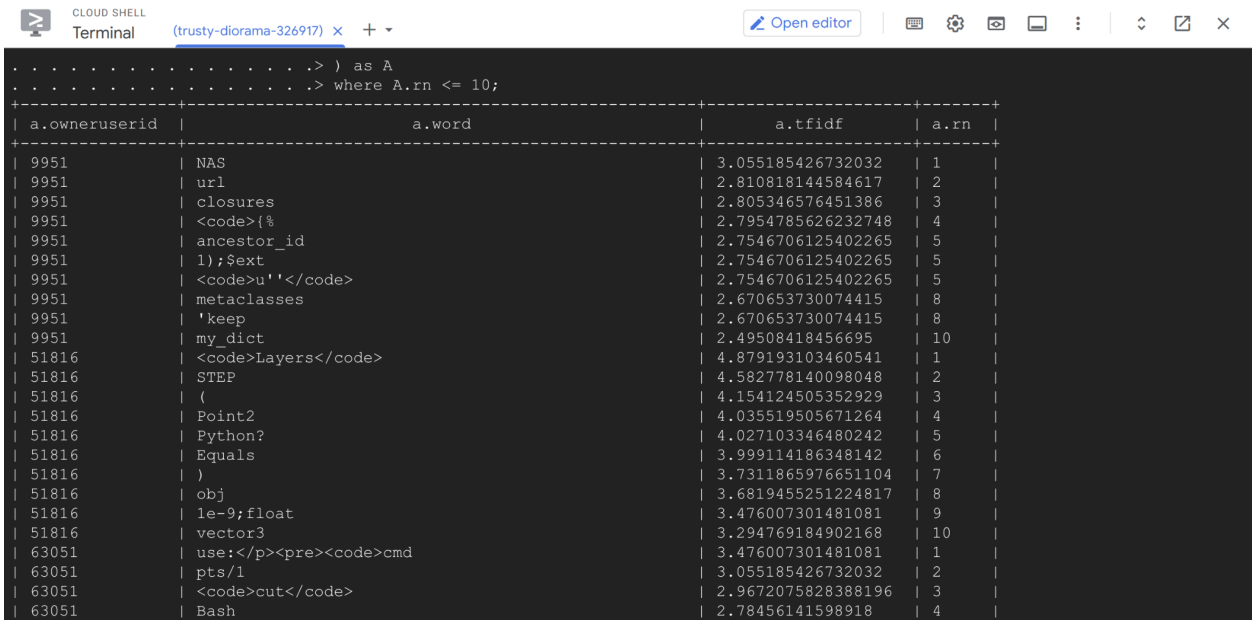

## Screenshot task 2.2.3


## Screenshot task 4

Save the tfidf table on Hadoop.



Result for getting top 10 words for each top 10 OwnerUserId




```
. . . . . .> ) as A
. . . . . .> where A.rn <= 10;
+----------------+-----------------------------------------------------+---------------------+-------+
| a.owneruserid  |                       a.word                        |       a.tfidf       | a.rn  |
+----------------+-----------------------------------------------------+---------------------+-------+
| 9951           | NAS                                                 | 3.055185426732032   | 1     |
| 9951           | url                                                 | 2.810818144584617   | 2     |
| 9951           | closures                                            | 2.805346576451386   | 3     |
| 9951           | <code>{%                                            | 2.7954785626232748  | 4     |
| 9951           | ancestor_id                                         | 2.7546706125402265  | 5     |
| 9951           | 1);$ext                                             | 2.7546706125402265  | 5     |
| 9951           | <code>u''</code>                                    | 2.7546706125402265  | 5     |
| 9951           | metaclasses                                         | 2.670653730074415   | 8     |
| 9951           | 'keep                                               | 2.670653730074415   | 8     |
| 9951           | my_dict                                             | 2.49508418456695    | 10    |
| 51816          | <code>Layers</code>                                 | 4.879193103460541   | 1     |
| 51816          | STEP                                                | 4.582778140098048   | 2     |
| 51816          | (                                                   | 4.154124505352929   | 3     |
| 51816          | Point2                                              | 4.035519505671264   | 4     |
| 51816          | Python?                                             | 4.027103346480242   | 5     |
| 51816          | Equals                                              | 3.999114186348142   | 6     |
| 51816          | )                                                   | 3.7311865976651104  | 7     |
| 51816          | obj                                                 | 3.6819455251224817  | 8     |
| 51816          | 1e-9;float                                          | 3.476007301481081   | 9     |
| 51816          | vector3                                             | 3.294769184902168   | 10    |
| 63051          | use:</p><pre><code>cmd                              | 3.476007301481081   | 1     |
| 63051          | pts/1                                               | 3.055185426732032   | 2     |
| 63051          | <code>cut</code>                                    | 2.9672075828388196  | 3     |
| 63051          | Bash                                                | 2.78456141598918    | 4     |
```