CA675 Assignment 01. **Large dataset Analysis**

Student ID: 18212693
Name: Minkun Kim
Email: minkun.kim4@mail.dcu.ie

"Stack Exchange is a network of question and answer websites on diverse topics in many different fields, each site covering a specific topic, where questions, answers, and users are subject to a reputation award process. The sites are modeled after Stack Overflow, a forum for computer programming questions that was the original site in this network."
Stack Exchange Data Explorer (SEDE) https://data.stackexchange.com/stackoverflow/query/new

[Task 1] Data Acquisition:

We are required to acquire the **top 200,000 posts by viewcount** from the Stack Exchange site. Problem is that we can only download 50.000 records at a time.

> We should run at least 4 to 5 queries in total to obtain 200,000 posts. The first thing to figure out would be the range of values in "ViewCount" field that constitutes the top 200,000 posts. After a series of attempts to find the lower bound value in "ViewCount" that accommodates the 200,000 data, we discover that the values in "ViewCount" greater than 28574 give us 200,001 records, thus it is safe to say that we can get at least 200,000 data records by offering 28574 as our lower bound in "ViewCount".

```
:  select count(*) from posts where posts.ViewCount > 28574
```

- This gives **200,001**

> Next, given that we can only obtain 50,000 records at a time, we can break down the the whole range of "ViewCount greater than 28574" into at least 4 parts. And each of which has 50,000 records. In order to sort this out simple, we arrange them in a descending order.

For TOP 25%

```
:  select top 50000 * from posts where posts.ViewCount > 28574 order by posts.ViewCount DESC
```

For Next 25%

```
:  select top 50000 * from posts where posts.ViewCount <= 87000 order by posts.ViewCount DESC
```

For Next 25%

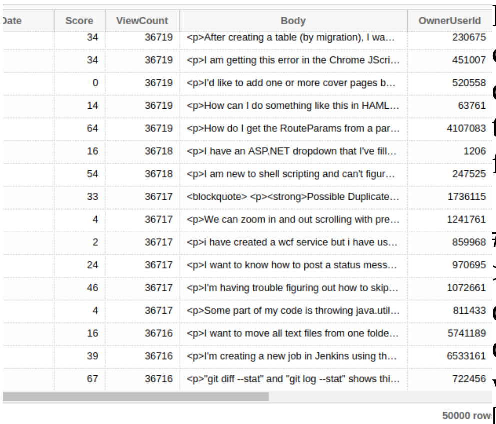
```
:  select top 50000 * from posts where posts.ViewCount <= 51208 order by posts.ViewCount DESC
```

For Next 25%

```
:  select top 50000 * from posts where posts.ViewCount <= 36716 order by posts.ViewCount DESC
```

In case of duplicates or unexpected errors,

```
:  select top 50 * from posts where posts.ViewCount <= 28678 order by posts.ViewCount DESC
```



Date	Score	ViewCount	Body	OwnerUserId
	34	36719	<p>After creating a table (by migration), I wa...	230675
	34	36719	<p>I am getting this error in the Chrome JScri...	451007
	0	36719	<p>I'd like to add one or more cover pages b...	520558
	14	36719	<p>How can I do something like this in HAML...	63761
	64	36719	<p>How do I get the RouteParams from a par...	4107083
	16	36718	<p>I have an ASP.NET dropdown that I've fill...	1206
	54	36718	<p>I am new to shell scripting and can't figur...	247525
	33	36717	<blockquote> <p><strong>Possible Duplicate...	1736115
	4	36717	<p>We can zoom in and out scrolling with pre...	1241761
	2	36717	<p>i have created a wcf service but i have us...	859968
	24	36717	<p>I want to know how to post a status mess...	970695
	46	36717	<p>I'm having trouble figuring out how to skip...	1072661
	4	36717	<p>Some part of my code is throwing java.util...	811433
	16	36716	<p>I want to move all text files from one folde...	5741189
	39	36716	<p>I'm creating a new job in Jenkins using th...	6533161
	67	36716	<p>"git diff --stat" and "git log --stat" shows thi...	722456

50000 row

[figure 1]. the last few records on the range of ViewCount <= 51208

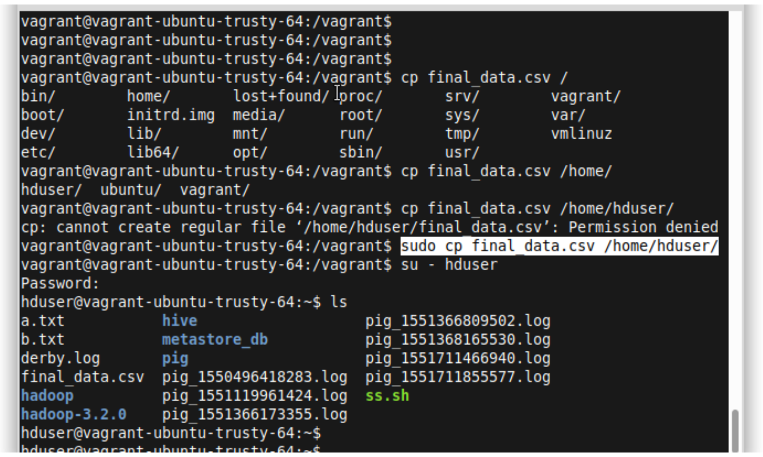
From the last few records fetched from each query, we can obtain the upper bound in "ViewCount" to use for the next query. In this screen shot, for example, we can see 36716 is the ViewCount value that can be utilized as the upper bound for next query.

# Data Cleaning with Python

> One acquiring the full dataset we need, we can do a little data cleaning with Python. For example, the removal of duplicates, or merging dataset can be done in Python. In PIG, we will pick up the left off on the rest of data cleaning task.

[Task 2] Data Cleaning with PIG:

Extract, transform and load the data as applicable.



```
vagrant@vagrant-ubuntu-trusty-64:/vagrant$
vagrant@vagrant-ubuntu-trusty-64:/vagrant$
vagrant@vagrant-ubuntu-trusty-64:/vagrant$
vagrant@vagrant-ubuntu-trusty-64:/vagrant$ cp final_data.csv /
bin/        home/       lost+found/ proc/       srv/        vagrant/
boot/       initrd.img  media/      root/       sys/        var/
dev/        lib/        mnt/        run/        tmp/        vmlinuz
etc/        lib64/      opt/        sbin/       usr/
vagrant@vagrant-ubuntu-trusty-64:/vagrant$ cp final_data.csv /home/
hduser/  ubuntu/  vagrant/
vagrant@vagrant-ubuntu-trusty-64:/vagrant$ cp final_data.csv /home/hduser/
cp: cannot create regular file ‘/home/hduser/final_data.csv’: Permission denied
vagrant@vagrant-ubuntu-trusty-64:/vagrant$ sudo cp final_data.csv /home/hduser/
vagrant@vagrant-ubuntu-trusty-64:/vagrant$ su - hduser
Password:
hduser@vagrant-ubuntu-trusty-64:~$ ls
a.txt           hive                    pig_1551366809502.log
b.txt           metastore_db            pig_1551368165530.log
derby.log       pig                     pig_1551711466940.log
final_data.csv  pig_1550496418283.log  pig_1551711855577.log
hadoop          pig_1551119961424.log  ss.sh
hadoop-3.2.0    pig_1551366173355.log
hduser@vagrant-ubuntu-trusty-64:~$
hduser@vagrant-ubuntu-trusty-64:~$
```

> First, we allow our virtual machine ‘vagrant’ to access our dataset by locating it on the shared folder between our local machine and ‘vagrant’. Then we copy this dataset from our ‘vagrant’ virtual machin home directory onto Hadoop User home directory.

[figure 2]. vagrant and hadoop user terminal

> Then we load our dataset into PIG. Our main task in PIG is the data cleaning. We simply copy our dataset from Hadoop-User-home where our PIG is located.

```
:  copyFromLocal mydata.csv user/hduser
```

> Load our dataset in PIG, specifying each data type.

```
:  mydata = LOAD 'final_data.csv' using PigStorage(',') AS (Index: int, Id:int, PostTypeId:int, AcceptedAnswerId:int, ParentId:int, CreationDate:datetime, DeletionDate:datetime, Score:int, ViewCount:int, OwnerUserId:int, OwnerDisplayName:chararray, LastEditorUserId:int, LastEditorDisplayName:chararray, LastEditDate:datetime, LastActivityDate:datetime, Title:chararray, Tags:chararray, AnswerCount:int, CommentCount:int, FavoriteCount:int, ClosedDate:datetime, CommunityOwnedDate:datetime);
```

> Pick up fields that we need, and generate a new table.

```
:  A = FOREACH mydata GENERATE Id, Score, ViewCount, OwnerUserId, OwnerDisplayName, Title, Tags;
```

> Create a new file folder to save this cleaned up data. Here our cleaned up dataset is stored in this folder - newdata

```
:  STORE A INTO 'newdata' using PigStorage(',');
```

> Copy this new file folder into Hadoop User home directory.

```
:  copyToLocal newdata /home/hduser/
```

> Move to Hadoop User directory and finally pick up and copy our dataset to vagrant location in the local machine. Now we can see our cleaned dataset “part-m-00000” in the shared folder between our local machine and ‘vagrant’. Next we copy our cleaned dataset to Hadoop User home directory to allow HIVE to access it. I changed its name as “data.csv”.

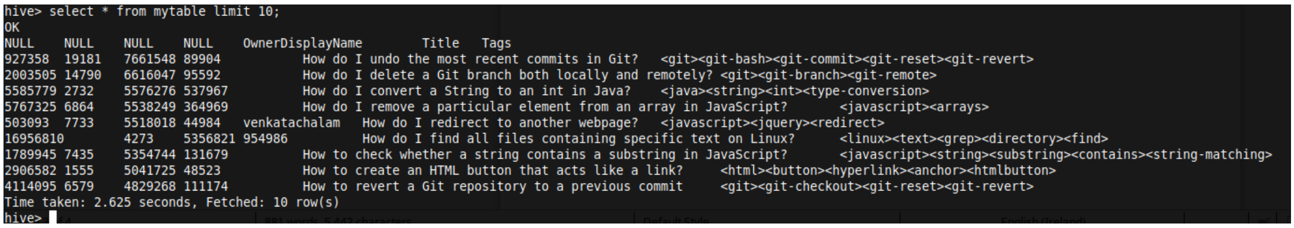
```
:  cd newdata
:  sudo cp part-m-00000 /vagrant
:  sudo cp part-m-00000 /home/hduser/data.csv
```

[Task 3] Querying with HIVE:

1. The top 10 posts by score
2. The top 10 users by score
3. The number of distinct users, who used the word ‘hadoop’ in one of their posts

> Now we access our dataset in HIVE environment. First we need to create the empty table (I named it as mytable) to fill in , using our dataset. Then we can import our dataset ‘data.csv’ and overwrite the table.

: hive>> create external table if not exists **mytable** (Id int, Score int, ViewCount int, OwnerUserId int, OwnerDisplayName string, Title string, Tags string)
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ',';
: hive>> load data local inpath 'data.csv' overwrite into table **mytable**;
: hive>> select * from **mytable** limit 10;



```
hive> select * from mytable limit 10;
OK
NULL    NULL    NULL    NULL    OwnerDisplayName        Title   Tags
927358  19181   7661548 89904           How do I undo the most recent commits in Git?   <git><git-bash><git-commit><git-reset><git-revert>
2003505 14790   6616047 95592           How do I delete a Git branch both locally and remotely? <git><git-branch><git-remote>
5585779 2732    5576276 537967          How do I convert a String to an int in Java?    <java><string><int><type-conversion>
5767325 6864    5538249 364969          How do I remove a particular element from an array in JavaScript?       <javascript><arrays>
503093  7733    5518018 44984   venkatachalam   How do I redirect to another webpage?   <javascript><jquery><redirect>
16956810        4273    5356821 954986          How do I find all files containing specific text on Linux?      <linux><text><grep><directory><find>
1789945 7435    5354744 131679          How to check whether a string contains a substring in JavaScript?       <javascript><string><substring><contains><string-matching>
2906582 1555    5041725 48523           How to create an HTML button that acts like a link?     <html><button><hyperlink><anchor><htmlbutton>
4114095 6579    4829268 111174          How to revert a Git repository to a previous commit     <git><git-checkout><git-reset><git-revert>
Time taken: 2.625 seconds, Fetched: 10 row(s)
hive>
```

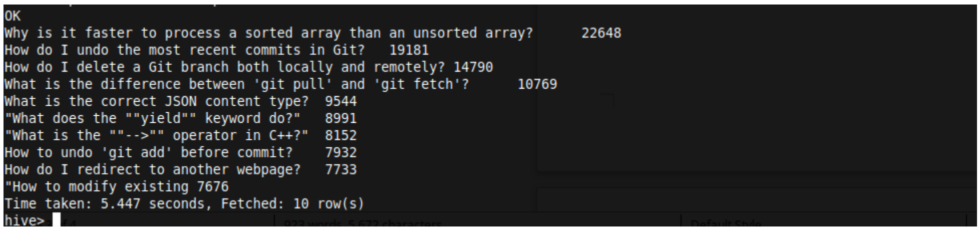
[figure 3]. HIVE terminal

Now, we can answer the questions.

Q1.

> We want to see top 10 posts(referred by title, score) in StackExhange in order by the **score**.

: hive>> select Title, Score from **mytable** order by Score desc limit 10;



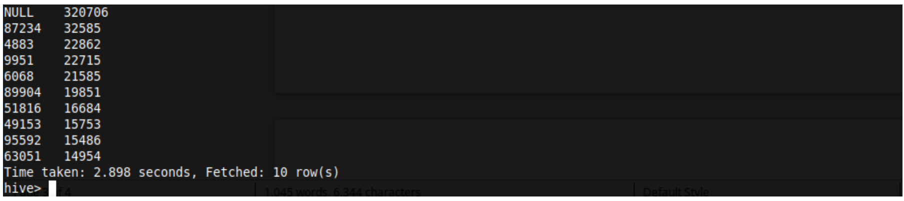
```
OK
Why is it faster to process a sorted array than an unsorted array?      22648
How do I undo the most recent commits in Git?   19181
How do I delete a Git branch both locally and remotely? 14790
What is the difference between 'git pull' and 'git fetch'?      10769
What is the correct JSON content type?  9544
"What does the ""yield"" keyword do?"   8991
"What is the ""-->"" operator in C++?"  8152
How to undo 'git add' before commit?    7932
How do I redirect to another webpage?   7733
"How to modify existing 7676
Time taken: 5.447 seconds, Fetched: 10 row(s)
hive>
```

[figure 4]. HIVE terminal

Q2.

> We try to make two approaches because ‘user’ can be defined in two ways – **UserID**, **UserName**. The first is tried with respect to **UserId** and the second is tried with respect to **UserName**. We need to sort users by the total score, and this implies an aggregate function such as SUM() & group by(). Thus it makes more sense to create a temporary table that has two fields – A: UserID, B: the output from SUM(score) & group by(UserID) – in order to capture the top 10 users by scores.

: hive>> create table user_table as select **ownerUserId** as A, **SUM(Score)** as B from **mytable** group by **ownerUserId**;
: hive> select * from users_table order by B desc limit 10;

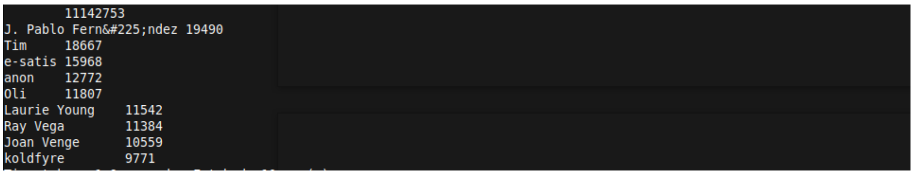


```
NULL    320706
87234   32585
4883    22862
9951    22715
6068    21585
89904   19851
51816   16684
49153   15753
95592   15486
63051   14954
Time taken: 2.898 seconds, Fetched: 10 row(s)
hive>
```

[figure 5]. HIVE terminal

: hive> create table

user_table_2 as select **OwnerDisplayName** as C, **SUM(Score)** as D from **mytable** group by **ownerDisplayName;**
: hive> select * from users_table_2 order by D desc limit 10;

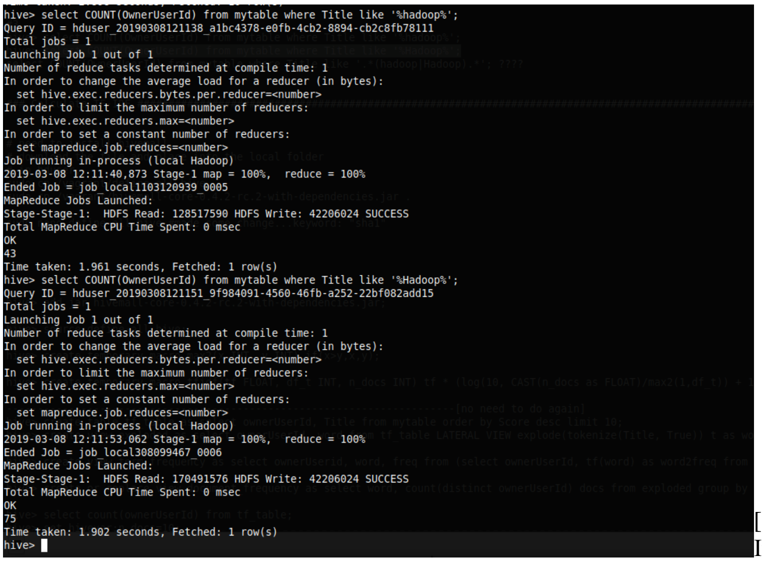


```
        11142753
J. Pablo Fern&#225;ndez 19490
Tim     18667
e-satis 15968
anon    12772
Oli     11807
Laurie Young    11542
Ray Vega        11384
Joan Venge      10559
koldfyre        9771
```

[figure 6]. HIVE terminal

Q3.

> We want to find how many users have ever built posts about HADOOP in Stack Exchange website. Thus another aggragate function COUNT() can be used with respect to word “hadoop” or “Hadoop”.
: hive> select COUNT(**OwnerUserId**) from **mytable** where Title like '%hadoop%';
: hive> select COUNT(**OwnerUserId**) from **mytable** where Title like '%Hadoop%';



```
hive> select COUNT(OwnerUserId) from mytable where Title like '%hadoop%';
Query ID = hduser_20190308121138_a1bc4378-e0fb-4cb2-8894-cb2c8fb78111
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2019-03-08 12:11:40,873 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_local1103120939_0005
MapReduce Jobs Launched:
Stage-Stage-1:  HDFS Read: 128517590 HDFS Write: 42206024 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
43
Time taken: 1.961 seconds, Fetched: 1 row(s)
hive> select COUNT(OwnerUserId) from mytable where Title like '%Hadoop%';
Query ID = hduser_20190308121151_9f984091-4560-46fb-a252-22bf082add15
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2019-03-08 12:11:53,062 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_local308099467_0006
MapReduce Jobs Launched:
Stage-Stage-1:  HDFS Read: 170491576 HDFS Write: 42206024 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
75
Time taken: 1.902 seconds, Fetched: 1 row(s)
hive>
```

[figure 7]. HIVE terminal:
It gives 43 for “hadoop” and 75 for “Hadoop”.

[Task 4] Calculate the per-user TF-IDF with HIVE:
Find **Top 10 terms** used for each of the top 10 users by post score

Hivemall: Apache Hivemall is a scalable machine learning library that runs on Apache Hive/Pig/Spark. It provides machine learning functionality as well as **feature engineering functions** through UDFs/UDAFs/UDTFs of Hive. When it comes to feature engineering in the context of information retrieval, Hivemall offers TF-IDF tool.

> We install Hivemall. First, download the following two installation files and place into the local shared folder:
*define-all.hive
*hivemall-core-0.4.2-rc.2-with-dependencies.jar

> Then in Hadoop User home directory, we bring in these two files, and make some changes in ‘define-all.hive’ file that contains Genneral Macros and Statistics functions.
: sudo cp /vagrant/define-all.hive .
: sudo cp /vagrant/hivemall-core-0.4.2-rc.2-with-dependencies.jar .
> In ‘define-all.hive’, via nano editer, we make a change by adding ‘--’ in the front.
--drop temporary function sha1;
--create temporary function sha1 as 'hivemall.ftvec.hashing.Sha1UDF';

> Next, loads all Hivemall functions and define macros used in the TF-IDF computation.

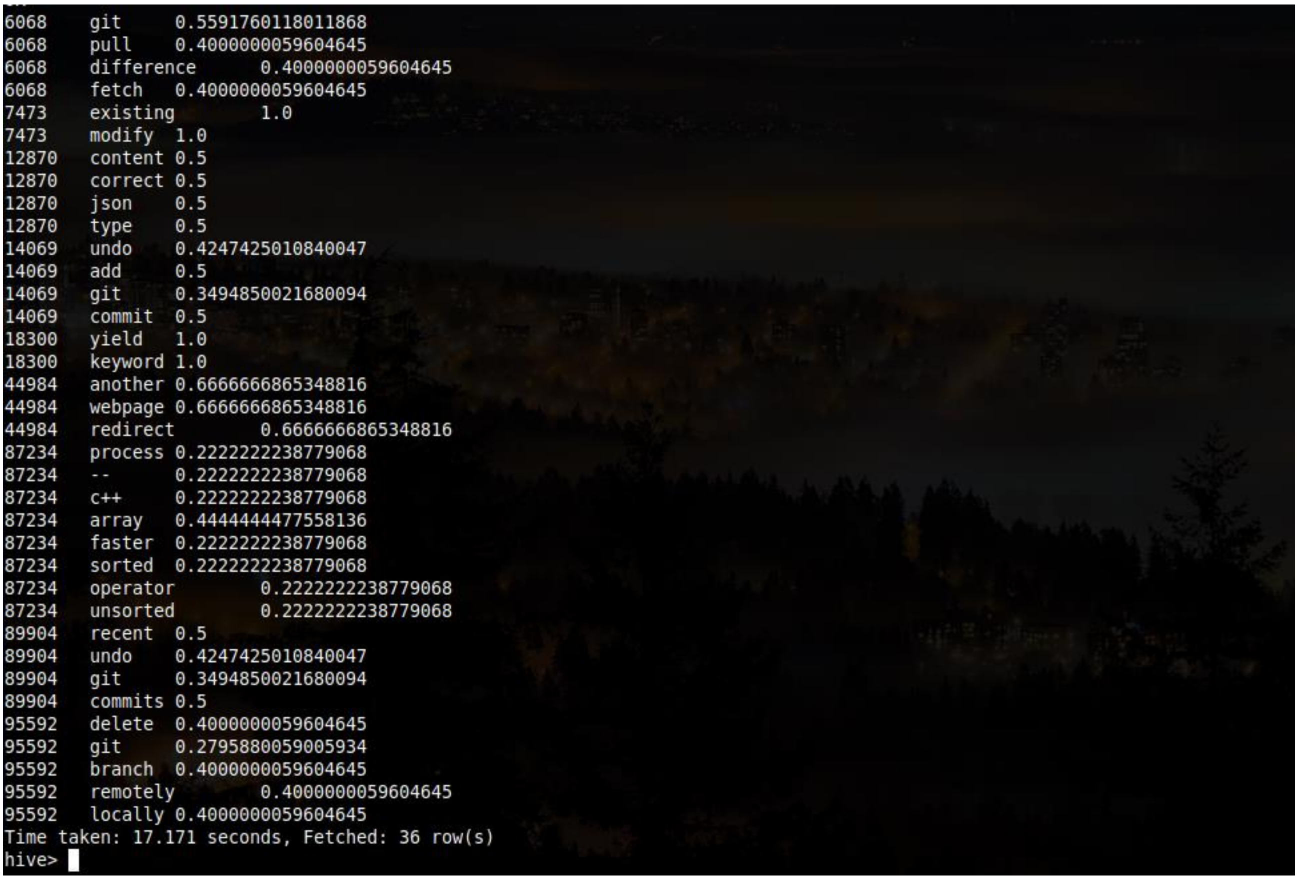
```
:  hive> add jar hivemall-core-0.4.2-rc.2-with-dependencies.jar;
:  hive> source define-all.hive;
:  hive> create temporary macro max2(x INT, y INT) if(x>y,x,y);
:  hive> create temporary macro tfidf(tf FLOAT, df_t INT, n_docs INT) tf * (log(10, CAST(n_docs as FLOAT)/max2(1,df_t)) + 1.0);
```

> Finally, creating a table To calculate TF-IDF, preparing a relation consists of (docid,word) tuples and do TF-IDF calculation for each docid/word pair.

```
:  hive> create table tf_table as select ownerUserId, Title from mytable order by Score desc limit 10;
:  hive> create view exploded as select ownerUserId, word from tf_table LATERAL VIEW explode(tokenize(Title, True)) t as word where not is_stopword(word);
:  hive> create view term_frequency as select ownerUserid, word, freq from (select ownerUserId, tf(word) as word2freq from exploded group by ownerUserId) t LATERAL VIEW explode(word2freq) t2 as word, freq;
:  hive> create or replace view document_frequency as select word, count(distinct ownerUserId) docs from exploded group by word;
:  hive> select count(ownerUserId) from tf_table;
:  hive> set hivevar:n_docs=10;
:  hive> create or replace view tfidf as select tf.ownerUserId,  tf.word, tfidf(tf.freq, df.docs, ${n_docs}) as tfidf from  term_frequency tf  JOIN document_frequency df ON (tf.word = df.word) order by tfidf desc;
```

> Now we can get the result ! (w.r.t **userID** & **terms** used)

```
:  hive> select * from tfidf;
```



```
6068    git     0.5591760118011868
6068    pull    0.4000000059604645
6068    difference      0.4000000059604645
6068    fetch   0.4000000059604645
7473    existing        1.0
7473    modify  1.0
12870   content 0.5
12870   correct 0.5
12870   json    0.5
12870   type    0.5
14069   undo    0.4247425010840047
14069   add     0.5
14069   git     0.3494850021680094
14069   commit  0.5
18300   yield   1.0
18300   keyword 1.0
44984   another 0.6666666865348816
44984   webpage 0.6666666865348816
44984   redirect        0.6666666865348816
87234   process 0.2222222238779068
87234   --      0.2222222238779068
87234   c++     0.2222222238779068
87234   array   0.4444444477558136
87234   faster  0.2222222238779068
87234   sorted  0.2222222238779068
87234   operator        0.2222222238779068
87234   unsorted        0.2222222238779068
89904   recent  0.5
89904   undo    0.4247425010840047
89904   git     0.3494850021680094
89904   commits 0.5
95592   delete  0.4000000059604645
95592   git     0.2795880059005934
95592   branch  0.4000000059604645
95592   remotely        0.4000000059604645
95592   locally 0.4000000059604645
Time taken: 17.171 seconds, Fetched: 36 row(s)
hive> █
```

[figure 8]. HIVE terminal: it gives what terms are used by each top 10 user.