P1 Presentation

By: Alan Liang

Overview of Process

1. Insert filtered data in table

2. Combine tables with filtered data (aggregate data if need to)

3. Do necessary computations in the table with filtered data

4. Combine results (columns) into one single table

5. Display the results with limit or where

1. Which English Wikipedia article got the most traffic on October 20th?

DATA USED

pageviews-20201020-000000.gz -

pageviews-20201020-230000.gz

HIVE QUERIES USED TO ANSWER THE QUESTION

First we set up our database and create a table that takes in pageview data from Oct containing only domain codes ‘en’ and ‘en.m’.

```
1) CREATE DATABASE PAGEVIEW_DB;

2) USE PAGEVIEW_DB;

3) CREATE TABLE OCT_PAGEVIEWS (DOMAIN_CODE STRING, PAGE_TITLE STRING, NUM_OF_REQUESTS INT, RESPONSE_BYTES INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ' ';

4) CREATE TABLE EN_PAGEVIEWS (PAGE_TITLE STRING, NUM_OF_REQUESTS INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ' ';
```

(cont in next page...)

1. Which English Wikipedia article got the most traffic on October 20th?

HIVE QUERIES USED TO ANSWER THE QUESTION (cont.)

5) LOAD DATA LOCAL INPATH '/home/aliang30/pageviews-20201020/*' INTO TABLE OCT_PAGEVIEWS;

6) INSERT INTO EN_PAGEVIEWS (SELECT PAGE_TITLE, NUM_OF_REQUESTS FROM OCT_PAGEVIEWS WHERE DOMAIN_CODE='en');

7) INSERT INTO EN_PAGEVIEWS (SELECT PAGE_TITLE, NUM_OF_REQUESTS FROM OCT_PAGEVIEWS WHERE DOMAIN_CODE='en.m');

(Now that we have our table with only English wiki articles, we want to aggregate the num_of_requests with similar page_titles.

And then we display our results

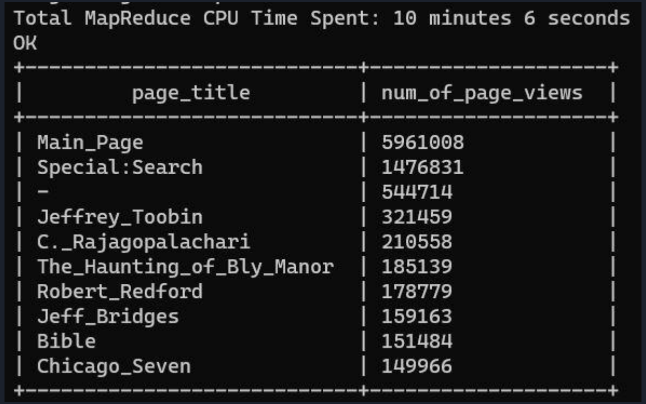
8) SELECT PAGE_TITLE,

SUM (NUM_OF_REQUESTS) AS num_of_page_views FROM EN_PAGEVIEWS

GROUP BY PAGE_TITLE

ORDER BY SUM(NUM_OF_REQUESTS) DESC

LIMIT 10;



```
Total MapReduce CPU Time Spent: 10 minutes 6 seconds
OK
+-----------------------------+----------------------+
|         page_title          | num_of_page_views    |
+-----------------------------+----------------------+
| Main_Page                   | 5961008              |
| Special:Search              | 1476831              |
| -                           | 544714               |
| Jeffrey_Toobin              | 321459               |
| C._Rajagopalachari          | 210558               |
| The_Haunting_of_Bly_Manor   | 185139               |
| Robert_Redford              | 178779               |
| Jeff_Bridges                | 159163               |
| Bible                       | 151484               |
| Chicago_Seven               | 149966               |
+-----------------------------+----------------------+
```

2. What English Wikipedia article has the largest fraction of its readers follow an internal link to another wikipedia article?

DATA USED

We will be using September's data

pageviews-20200901-000000.gz - pageviews-20200930-230000.gz

clickstream-enwiki-2020-09.tsv

COMPUTATION USED

Percentage of readers follow an internal link = (Total internal_link clicks in article) * 100/Total pageviews)

2. What English Wikipedia article has the largest fraction of its readers follow an internal link to another wikipedia article?

HIVE QUERIES USED TO ANSWER THE QUESTION

First we set up our database and create a table that takes in clickstream data from Sept containing only internal link type.

1) CREATE DATABASE CLICKSTREAM_DB;

2) USE CLICKSTREAM_DB;

3) CREATE TABLE CS_TABLE (PREV STRING, CURR STRING, TYPE STRING, NUM_OF_CLICK_THROUGHS INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';

4) CREATE TABLE INTERNAL_DATA (PREV STRING, NUM_OF_CLICK_THROUGHS INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';

5) LOAD DATA LOCAL INPATH '/home/aliang30/clickstream-enwiki-2020-09.tsv' INTO TABLE CS_TABLE;

6) INSERT INTO INTERNAL_DATA (SELECT PREV, NUM_OF_CLICK_THROUGHS FROM CS_TABLE WHERE TYPE='link');

Now that we have the data that contains only internal links, we must aggregate the num of internal link clicks with similar page_titles and insert new data into a new table

7) CREATE TABLE UPDATED_INTERNAL_DATA (PREV STRING, NUM_OF_CLICK_THROUGHS INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';

8) INSERT INTO UPDATED_INTERNAL_DATA (SELECT PREV, SUM (NUM_OF_CLICK_THROUGHS) AS NUM_OF_INTERNAL FROM EN_PAGEVIEWS GROUP BY PAGE_TITLE);

2. What English Wikipedia article has the largest fraction of its readers follow an internal link to another wikipedia article?

HIVE QUERIES USED TO ANSWER THE QUESTION (cont))

Now we need to create a new table that takes in pageview data from Sept containing only domain codes ‘en’ and ‘en.m’.

```
7) CREATE TABLE SEPT_PAGEVIEWS (DOMAIN_CODE STRING, PAGE_TITLE STRING, NUM_OF_REQUESTS INT, RESPONSE_BYTES INT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ' ';

8) CREATE TABLE EN_PAGEVIEWS (PAGE_TITLE STRING, NUM_OF_REQUESTS INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY '
';

9) LOAD DATA LOCAL INPATH '/home/aliang30/pageviews-20200920/*' INTO TABLE SEPT_PAGEVIEWS;

10) INSERT INTO EN_PAGEVIEWS (SELECT PAGE_TITLE, NUM_OF_REQUESTS FROM SEPT_PAGEVIEWS WHERE DOMAIN_CODE='en');

11)  INSERT INTO EN_PAGEVIEWS (SELECT PAGE_TITLE, NUM_OF_REQUESTS FROM SEPT_PAGEVIEWS WHERE DOMAIN_CODE='en.m');
```

We aggregate the num_of_requests with similar page_titles and insert new data into a seperate table

```
12) CREATE TABLE UPDATED_EN_PAGEVIEWS (PAGE_TITLE STRING, NUM_OF_REQUESTS INT) ROW FORMAT DELIMITED FIELDS
TERMINATED BY ' ';

13) INSERT INTO UPDATED_EN_PAGEVIEWS (SELECT PAGE_TITLE, SUM (NUM_OF_REQUESTS) AS NUM_OF_PAGE_VIEWS FROM
EN_PAGEVIEWS GROUP BY PAGE_TITLE);
```

2. What English Wikipedia article has the largest fraction of its readers follow an internal link to another wikipedia article?

HIVE QUERIES USED TO ANSWER THE QUESTION (cont)

Now that we have our tables that contain 1) aggregated num_of_requests and 2) aggregated num_of_click_throughs, we will use an inner join to return the rows that have matching page_title.

```
14) CREATE TABLE Q2_RESULT AS

SELECT UPDATED_EN_PAGEVIEWS.PAGE_TITLE, UPDATED_EN_PAGEVIEWS.NUM_OF_PAGE_VIEWS ,
UPDATED_INTERNAL_DATA.NUM_OF_INTERNAL

FROM UPDATED_EN_PAGEVIEWS

INNER JOIN UPDATED_INTERNAL_DATA. ON UPDATED_EN_PAGEVIEWS.PAGE_TITLE = UPDATED_INTERNAL_DATA.PREV;
```

Now we want to do our computation: (Total internal_link clicks in article Link total * 100/Total pageview) and store that in the PERCENTAGE_OF_READERS_FOLLOW_A_LINK column.

```
15) SELECT PAGE_TITLE, NUM_OF_PAGE_VIEWS, NUM_OF_INTERNAL,

CAST(NUM_OF_INTERNAL * 100.0 / NUM_OF_PAGE_VIEWS AS DECIMAL) AS PERCENTAGE_OF_READERS_FOLLOW_A_LINK

FROM Q2_RESULT

ORDER BY PERCENTAGE_OF_READERS_FOLLOW_A_LINK DESC

LIMIT 10;
```

3. What series of Wikipedia articles, starting with Hotel California, keeps the largest fraction of its readers clicking on internal links?

HIVE QUERIES USED TO ANSWER THE QUESTION

Similar to question 2 but we will have to gather all of the articles “Hotel_California” refers to and see which one has the highest number of internal click_throughs

```
1) CREATE TABLE INTERNAL_DATA2 (PREV STRING,  CURR STRING, NUM_OF_CLICK_THROUGHS INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
```

We’ll use the CS_TABLE from previous question and gather all of the articles starting with “Hotel_California” as well as type “link” and insert them in UPDATED_INTERNAL_DATA2.

```
2) INSERT INTO INTERNAL_DATA2

SELECT PREV, CURR, NUM_OF_CLICK_THROUGHS FROM CS_TABLE

WHERE PREV = “Hotel_California” AND

TYPE = “link”

ORDER BY PREV DESC;
```

3. What series of Wikipedia articles, starting with Hotel California, keeps the largest fraction of its readers clicking on internal links?

HIVE QUERIES USED TO ANSWER THE QUESTION (cont)

Now that we have the data that contains only internal links, we must aggregate the num of internal link clicks with similar curr page_titles and insert new data into a new table

```
3) CREATE TABLE Q3_RESULTS (PREV STRING,  CURR STRING, NUM_OF_CLICK_THROUGHS INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
```

```
4) INSERT INTO Q3_RESULTS (SELECT PREV, CURR, SUM (NUM_OF_CLICK_THROUGHS) AS NUM_OF_INTERNAL

FROM INTERNAL_DATA2 GROUP BY CURR);
```

Display the results of the table and order them by the num of internal clicks

```
5) SELECT * FROM Q3_RESULTS ORDER BY NUM_OF_INTERNAL LIMIT 10;
```

4. Find an example of an English Wikipedia article that is relatively more popular in the UK. Find the same for the US and Australia.

Assumptions

1) According to the latest study of global internet traffic by Cisco, the internet is busiest between 9pm and 11pm around the world.
2) Wiki page views traffic time stamps are based on UTC (Coordinated Universal Time)
3) So if UTC is 12:00:00am, time in US eastern is 07:00:00pm, time in US pacific 04:00:00pm, time in US central is 06:00:00pm, time in UK (UTC) is 12:03:00am, and time in Australia (GMT) is 11:00:00am.

US eastern 9PM-11PM	UTC 2AM-4AM
US central 9PM-11PM	UTC 3AM - 5AM
US pacific 9PM-11PM	UTC 5AM - 7AM
UK 9PM-11PM	UTC 9PM - 11PM
Australia 9PM-11PM	UTC 8AM - 10AM

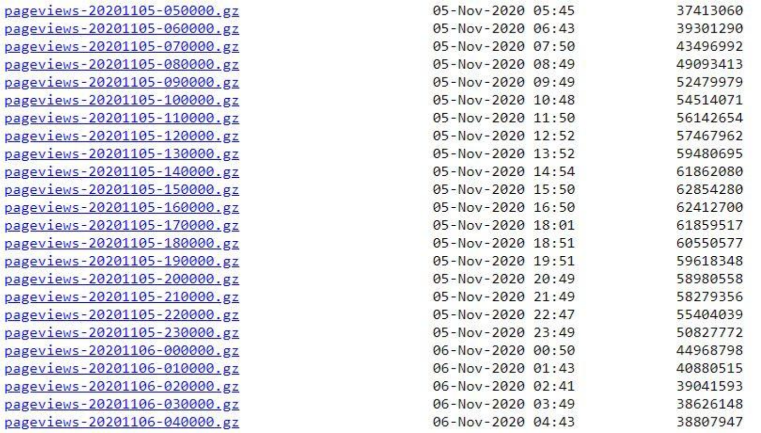
4. Find an example of an English Wikipedia article that is relatively more popular in the UK. Find the same for the US and Australia.

1) We can look at the time posted in https://dumps.wikimedia.org/other/pageviews/2020/2020-11/

2) Assuming that Wiki is running on UTC (Coordinated Universal Time), we can take a .gz from (for example: 06-Nov-2020 01:43) and assume that the English articles contained inside that .gz are relatively more popular in the US.

If from (05-Nov-2020 20:49), the English articles contained in that .gz are more popular in the UK

If from (05-Nov-2020 08:49), the English articles contained in that .gz are more popular in Australia



```
pageviews-20201105-050000.gz                      05-Nov-2020 05:45          37413060
pageviews-20201105-060000.gz                      05-Nov-2020 06:43          39301290
pageviews-20201105-070000.gz                      05-Nov-2020 07:50          43496992
pageviews-20201105-080000.gz                      05-Nov-2020 08:49          49093413
pageviews-20201105-090000.gz                      05-Nov-2020 09:49          52479979
pageviews-20201105-100000.gz                      05-Nov-2020 10:48          54514071
pageviews-20201105-110000.gz                      05-Nov-2020 11:50          56142654
pageviews-20201105-120000.gz                      05-Nov-2020 12:52          57467962
pageviews-20201105-130000.gz                      05-Nov-2020 13:52          59480695
pageviews-20201105-140000.gz                      05-Nov-2020 14:54          61862080
pageviews-20201105-150000.gz                      05-Nov-2020 15:50          62854280
pageviews-20201105-160000.gz                      05-Nov-2020 16:50          62412700
pageviews-20201105-170000.gz                      05-Nov-2020 18:01          61859517
pageviews-20201105-180000.gz                      05-Nov-2020 18:51          60550577
pageviews-20201105-190000.gz                      05-Nov-2020 19:51          59618348
pageviews-20201105-200000.gz                      05-Nov-2020 20:49          58980558
pageviews-20201105-210000.gz                      05-Nov-2020 21:49          58279356
pageviews-20201105-220000.gz                      05-Nov-2020 22:47          55404039
pageviews-20201105-230000.gz                      05-Nov-2020 23:49          50827772
pageviews-20201106-000000.gz                      06-Nov-2020 00:50          44968798
pageviews-20201106-010000.gz                      06-Nov-2020 01:43          40880515
pageviews-20201106-020000.gz                      06-Nov-2020 02:41          39041593
pageviews-20201106-030000.gz                      06-Nov-2020 03:49          38626148
pageviews-20201106-040000.gz                      06-Nov-2020 04:43          38807947
```

5. Analyze how many users will see the average vandalized Wikipedia page before the offending edit is reversed.

SOURCES USED

We will be using September's data

pageviews-20200901-000000.gz - pageviews-20200930-230000.gz

2020-09.enwiki.2020-09.tsv.bz2

HIVE QUERIES USED TO ANSWER THE QUESTION

Create a table that can hold all of the data in "2020-09.enwiki.2020-09.tsv.bz2."

```
1) CREATE TABLE DUMP_DATA (
        wiki_db string,
        event_entity string,
        event_type string,
        ...
        revision_tags array<string>
) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
```

5. Analyze how many users will see the average vandalized Wikipedia page before the offending edit is reversed.

HIVE QUERIES USED TO ANSWER THE QUESTION (cont)

Create tables to insert data and group by

```
2) CREATE TABLE FILTERED_DATA (PAGE_TITLE STRING, revision_seconds_to_identity_revert BIGINT) ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t';
```

```
.3) INSERT INTO FILTERED_DATA (SELECT PAGE_TITLE, revision_seconds_to_identity_revert FROM DUMP_DATA WHERE
EVENT_ENTITY='revision' AND revision_seconds_to_identity_revert > 0.0);
```

```
4) CREATE TABLE UPDATED_FILTERED_DATA (PAGE_TITLE STRING, revision_seconds_to_identity_revert BIGINT) ROW FORMAT
DELIMITED FIELDS TERMINATED BY '\t';
```

```
5) INSERT INTO UPDATED_FILTERED_DATA (SELECT PAGE_TITLE, SUM (revision_seconds_to_identity_revert) FROM FILTERED_DATA
GROUP BY PAGE_TITLE);
```

5. Analyze how many users will see the average vandalized Wikipedia page before the offending edit is reversed.

HIVE QUERIES USED TO ANSWER THE QUESTION (cont)

Reuse the UPDATED_EN_PAGEVIEWS table from earlier that contains the pageview data from Sept with domain_code "en" and "en.m" and inner join the table with UPDATED_FILTERED_DATA (return the rows that have matching page_title and move the results to Q5_RESULTS)

```
6) CREATE TABLE Q5_RESULTS AS

SELECT UPDATED_EN_PAGEVIEWS.PAGE_TITLE, UPDATED_EN_PAGEVIEWS.NUM_OF_PAGE_VIEWS,
UPDATED_FILTERED_DATA.revision_seconds_to_identity_revert

FROM UPDATED_EN_PAGEVIEWS

INNER JOIN UPDATED_FILTERED_DATA ON UPDATED_EN_PAGEVIEWS.PAGE_TITLE = UPDATED_FILTERED_DATA.PAGE_TITLE;
```

Now we want to make our computation: (SUM(NUM_OF_PAGE_VIEWS) / COUNT(PAGE_TITLE) FROM UPDATED_FILTERED_DATA to find the avg number of users that saw the vandalized Wiki page before the edit is reversed.

```
7) SELECT SUM(NUM_OF_PAGE_VIEWS) / COUNT(PAGE_TITLE) FROM Q5_RESULTS;
```

6. What is the total number of English articles visited on 10/20/2020 with titles that start with the letter ‘a’?

SOURCES USED

pageviews-20201020-000000.gz - pageviews-20201020-230000.gz

HIVE QUERIES USED TO ANSWER THE QUESTION

Similar procedure as question 1. First we create a table that takes in pageview data from Oct containing only domain codes ‘en’ and ‘en.m’.

1) USE PAGEVIEW_DB;

2) CREATE TABLE OCT_PAGEVIEWS (DOMAIN_CODE STRING, PAGE_TITLE STRING, NUM_OF_REQUESTS INT, RESPONSE_BYTES INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ' ';

3) CREATE TABLE EN_PAGEVIEWS (PAGE_TITLE STRING, NUM_OF_REQUESTS INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ' ';

4) LOAD DATA LOCAL INPATH '/home/aliang30/pageviews-20201020/*' INTO TABLE OCT_PAGEVIEWS;

5) INSERT INTO EN_PAGEVIEWS (SELECT PAGE_TITLE, NUM_OF_REQUESTS FROM OCT_PAGEVIEWS WHERE DOMAIN_CODE='en');

6) INSERT INTO EN_PAGEVIEWS (SELECT PAGE_TITLE, NUM_OF_REQUESTS FROM OCT_PAGEVIEWS WHERE DOMAIN_CODE='en.m');

6. What is the total number of English articles visited on 10/20/2020 with titles that start with the letter ‘a’?

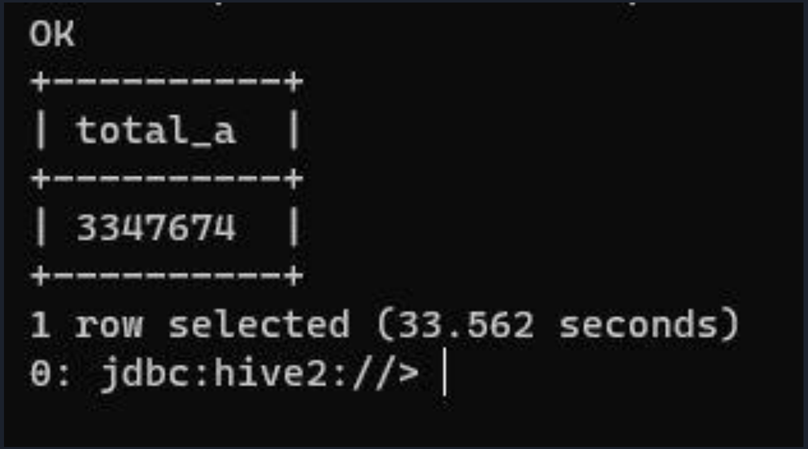
HIVE QUERIES USED TO ANSWER THE QUESTION (cont)

Now that we have our table with only English wiki articles, we can find the total number of articles that start with the letter ‘a’

7) SELECT COUNT(*) AS total_a

FROM EN_PAGEVIEWS

WHERE PAGE_TITLE LIKE 'A%';



```
OK
+-----------+
| total_a  |
+-----------+
| 3347674  |
+-----------+
1 row selected (33.562 seconds)
0: jdbc:hive2://>
```

Github link

https://github.com/AlanLiang1/Project_1.git