Wikipedia Data Analysis

Christian Palko • 02/05/2021

Question 1: Summary

Goal:

Determine which English wikipedia article got the most traffic on January 20, 2021.

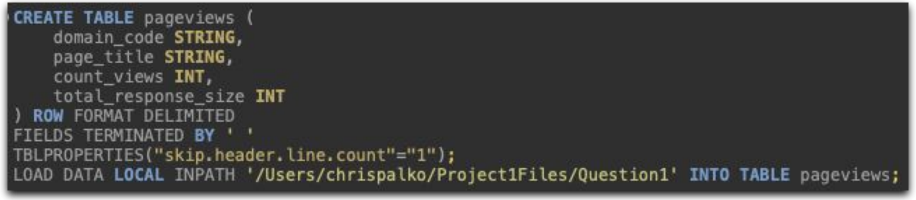
Key Assumptions Made:

- All English ".wikipedia.org" articles in the prepared data were correctly assigned a domain code of "en" or "en.m".
- Spider, bot, and illegitimate traffic was sufficiently filtered out from the prepared data.
- Non-".wikipedia.org" domains are irrelevant (Wikinews, Wikiquotes, etc.) and should not be included, as they contain same-named page titles which are different in content, and are orders of magnitude less popular.

Question 1: Technical Process

Step 1

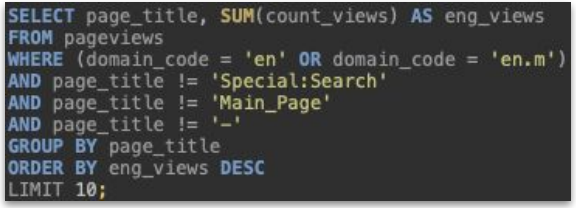
- Create a managed Hive table to fit the given data. That data is stored and then loaded, via local storage, into the table:



```
CREATE TABLE pageviews (
    domain_code STRING,
    page_title STRING,
    count_views INT,
    total_response_size INT
) ROW FORMAT DELIMITED
FIELDS TERMINATED BY ' '
TBLPROPERTIES("skip.header.line.count"="1");
LOAD DATA LOCAL INPATH '/Users/chrispalko/Project1Files/Question1' INTO TABLE pageviews;
```

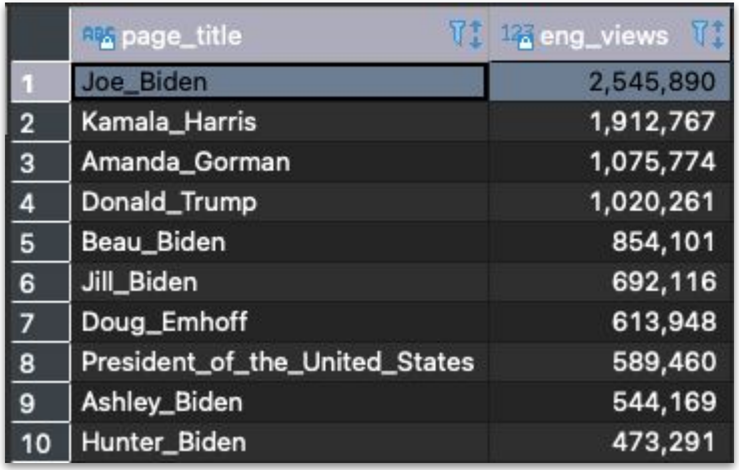
Step 2

- Use a select statement which groups together results of the same page title, sums the view counts, and excludes irrelevant results:



```
SELECT page_title, SUM(count_views) AS eng_views
FROM pageviews
WHERE (domain_code = 'en' OR domain_code = 'en.m')
AND page_title != 'Special:Search'
AND page_title != 'Main_Page'
AND page_title != '-'
GROUP BY page_title
ORDER BY eng_views DESC
LIMIT 10;
```

Question 1: Result



	page_title	eng_views
1	Joe_Biden	2,545,890
2	Kamala_Harris	1,912,767
3	Amanda_Gorman	1,075,774
4	Donald_Trump	1,020,261
5	Beau_Biden	854,101
6	Jill_Biden	692,116
7	Doug_Emhoff	613,948
8	President_of_the_United_States	589,460
9	Ashley_Biden	544,169
10	Hunter_Biden	473,291

Conclusion:

“Joe Biden” received the most traffic by far, reaching 2,545,890 views on his English Wikipedia page.

Question 2: Summary

Goal:

Determine what English wikipedia article has the largest fraction of its readers follow an internal link to another wikipedia article

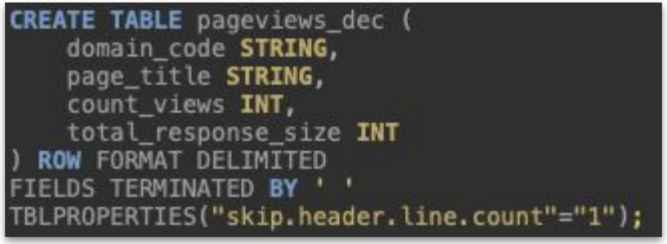
Key Assumptions Made:

- Spider, bot, and illegitimate traffic was sufficiently filtered out from the prepared data.
- The heavily sampled results, as a necessity due to hardware limits, are sufficient to represent the entire dataset (Dec. 15th vs. Dec. 1st-31st) with some compensatory measures.
- Users did not click links multiple times.

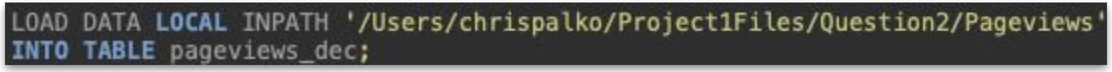
Question 2: Technical Process

Step 1

- Create a December pageviews table, then load in all pageview data for a full DAY (December 15th, 2020) to represent the MONTH



```
CREATE TABLE pageviews_dec (
    domain_code STRING,
    page_title STRING,
    count_views INT,
    total_response_size INT
) ROW FORMAT DELIMITED
FIELDS TERMINATED BY ' '
TBLPROPERTIES("skip.header.line.count"="1");
```

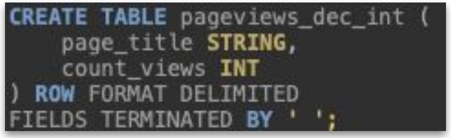


```
LOAD DATA LOCAL INPATH '/Users/chrispalko/Project1Files/Question2/Pageviews'
INTO TABLE pageviews_dec;
```

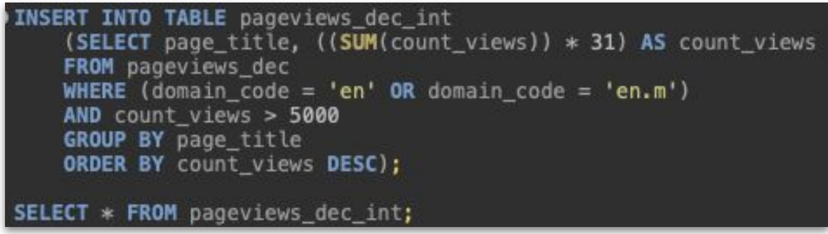
Question 2: Technical Process

Step 2

- Create an intermediate table with only relevant columns (page_title, count_views) from pageviews_dec. Insert pageviews_dec data into pageviews_dec_int.
- In the insert:
    - Filter out non-English domain results.
    - Filter out pages with less than 5000 views to cut down anomalies.
    - Multiply views per page by 31 to simulate the full month vs 1 day (hardware limitation of a single-node cluster; the full month is 100s of GBs).



```
CREATE TABLE pageviews_dec_int (
    page_title STRING,
    count_views INT
) ROW FORMAT DELIMITED
FIELDS TERMINATED BY ' ';
```

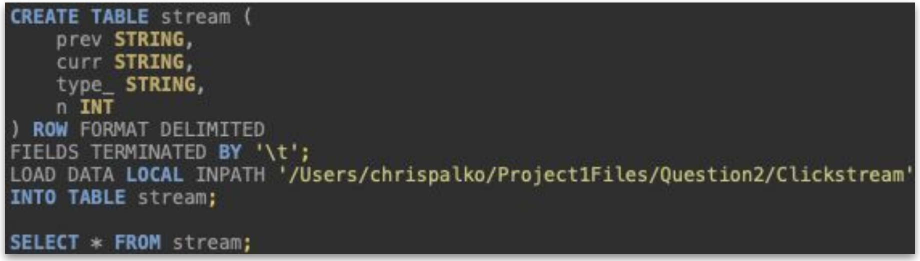


```
INSERT INTO TABLE pageviews_dec_int
    (SELECT page_title, ((SUM(count_views)) * 31) AS count_views
    FROM pageviews_dec
    WHERE (domain_code = 'en' OR domain_code = 'en.m')
    AND count_views > 5000
    GROUP BY page_title
    ORDER BY count_views DESC);

SELECT * FROM pageviews_dec_int;
```

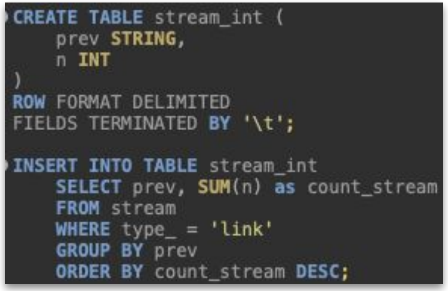
Question 2: Technical Process

Step 3

- Create a clickstream table, loading all English clickstreams from the month of December.
- Create an intermediate table to reduce workload in the coming join. Insert data from the initial clickstream data filtering out non-internal-link clickstream occurrences. Group by previous (prev) article.



```
CREATE TABLE stream (
    prev STRING,
    curr STRING,
    type_ STRING,
    n INT
) ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t';
LOAD DATA LOCAL INPATH '/Users/chrispalko/Project1Files/Question2/Clickstream'
INTO TABLE stream;

SELECT * FROM stream;
```

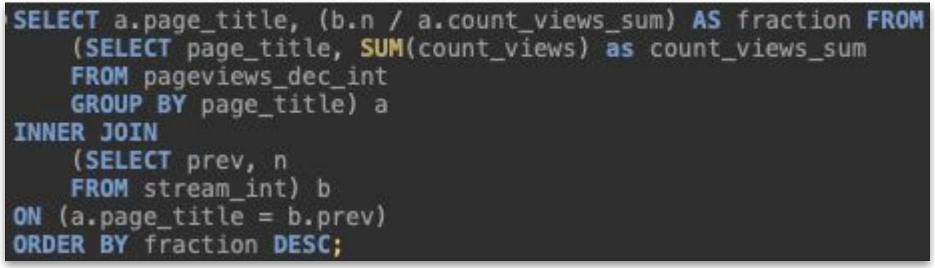


```
CREATE TABLE stream_int (
    prev STRING,
    n INT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t';

INSERT INTO TABLE stream_int
    SELECT prev, SUM(n) as count_stream
    FROM stream
    WHERE type_ = 'link'
    GROUP BY prev
    ORDER BY count_stream DESC;
```

Question 2: Technical Process

Step 4

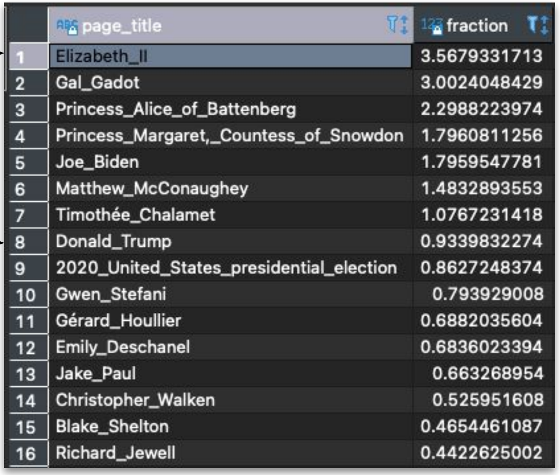
- Select the page title and the (number of clickstreams)/(number of view-counts) per page to get a rough result.



```
SELECT a.page_title, (b.n / a.count_views_sum) AS fraction FROM
    (SELECT page_title, SUM(count_views) as count_views_sum
    FROM pageviews_dec_int
    GROUP BY page_title) a
INNER JOIN
    (SELECT prev, n
    FROM stream_int) b
ON (a.page_title = b.prev)
ORDER BY fraction DESC;
```

Question 2: Result

or



	page_title	fraction
1	Elizabeth_II	3.5679331713
2	Gal_Gadot	3.0024048429
3	Princess_Alice_of_Battenberg	2.2988223974
4	Princess_Margaret,_Countess_of_Snowdon	1.7960811256
5	Joe_Biden	1.7959547781
6	Matthew_McConaughey	1.4832893553
7	Timothée_Chalamet	1.0767231418
8	Donald_Trump	0.9339832274
9	2020_United_States_presidential_election	0.8627248374
10	Gwen_Stefani	0.793929008
11	Gérard_Houllier	0.6882035604
12	Emily_Deschanel	0.6836023394
13	Jake_Paul	0.663268954
14	Christopher_Walken	0.525951608
15	Blake_Shelton	0.4654461087
16	Richard_Jewell	0.4422625002

Conclusion:

"Elizabeth_II" has the largest fraction of its readers follow an internal link to another wikipedia article.
"Donald_Trump" has the largest *realistic/possible* fraction, however.

More notes on assumptions:
The data produced will be *randomly* inaccurate due to spikes in specific pages on various days throughout the month. Due to the vast nature of the data combined with only a single-node cluster heavily limiting processing and storage capability, several results are calculated as over 100%.

This indicates that something spiked that page's views, but on a day which was not sampled, causing it to incur more occurrences of clickstreams than is representative.

Question 3: Summary

Goal:

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

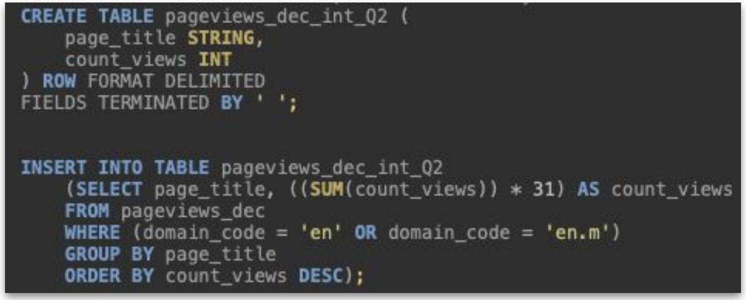
Key Assumptions Made:

- Any given user does not click a link multiple times.
- Users click out of pages at a similar rate that they are successfully referred to other pages.
- The heavily sampled page-view data is sufficient to represent the overall month.

Question 3: Technical Process

Step 1

- Create a new intermediate page-views table which doesn't filter out pages with <5000 view-counts, unlike in the previous question. The reasoning is that the likelihood of one specific article being subject to an unusual spike in views on a specific day is low.
- Filter out non-English views when inserting data from the primary pageviews table, and order by view-count, descending, to save resources on future queries.

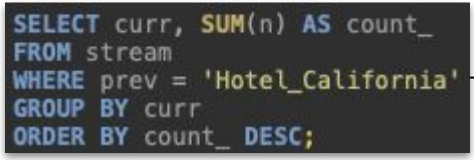


```
CREATE TABLE pageviews_dec_int_Q2 (
    page_title STRING,
    count_views INT
) ROW FORMAT DELIMITED
FIELDS TERMINATED BY ' ';


INSERT INTO TABLE pageviews_dec_int_Q2
    (SELECT page_title, ((SUM(count_views)) * 31) AS count_views
    FROM pageviews_dec
    WHERE (domain_code = 'en' OR domain_code = 'en.m')
    GROUP BY page_title
    ORDER BY count_views DESC);
```

Question 3: Technical Process

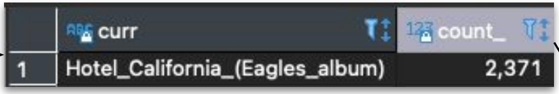
Step 2

- Find the numerator and the denominator, then join the results to view the fraction calculated.

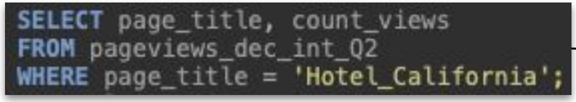


```sql
SELECT curr, SUM(n) AS count_
FROM stream
WHERE prev = 'Hotel_California'
GROUP BY curr
ORDER BY count_ DESC;
```

Numerator:

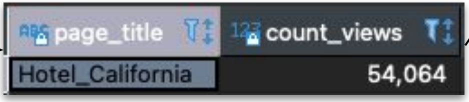


	curr	count_
1	Hotel_California_(Eagles_album)	2,371



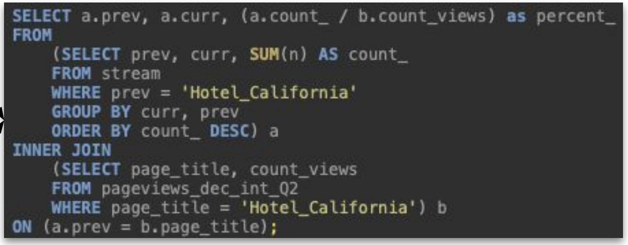
```sql
SELECT page_title, count_views
FROM pageviews_dec_int_Q2
WHERE page_title = 'Hotel_California';
```

Denominator:



page_title	count_views
Hotel_California	54,064

Join



```sql
SELECT a.prev, a.curr, (a.count_ / b.count_views) as percent_
FROM
    (SELECT prev, curr, SUM(n) AS count_
    FROM stream
    WHERE prev = 'Hotel_California'
    GROUP BY curr, prev
    ORDER BY count_ DESC) a
INNER JOIN
    (SELECT page_title, count_views
    FROM pageviews_dec_int_Q2
    WHERE page_title = 'Hotel_California') b
ON (a.prev = b.page_title);
```

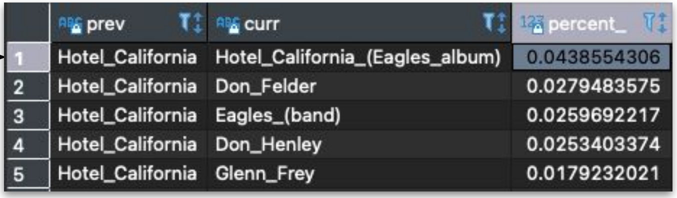
Question 3: Technical Process

Step 3

- View the joined results.
- Use the same join statement, but replace the WHERE clause condition with the top result, two times, and filter looped results.

Joined Results:

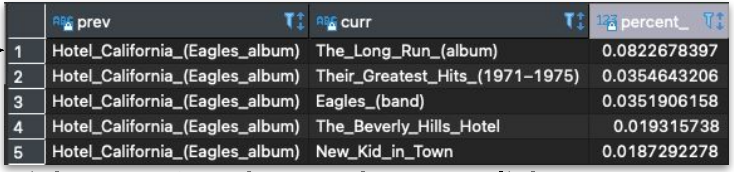
Starting Point: "Hotel_California"



	prev	curr	percent_
1	Hotel_California	Hotel_California_(Eagles_album)	0.0438554306
2	Hotel_California	Don_Felder	0.0279483575
3	Hotel_California	Eagles_(band)	0.0259692217
4	Hotel_California	Don_Henley	0.0253403374
5	Hotel_California	Glenn_Frey	0.0179232021

Highest % of readers clicked: "Hotel_California_(Eagles_album)"

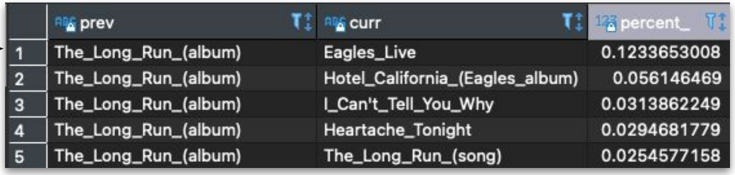
"Hotel_California_(Eagles_album)"



	prev	curr	percent_
1	Hotel_California_(Eagles_album)	The_Long_Run_(album)	0.0822678397
2	Hotel_California_(Eagles_album)	Their_Greatest_Hits_(1971–1975)	0.0354643206
3	Hotel_California_(Eagles_album)	Eagles_(band)	0.0351906158
4	Hotel_California_(Eagles_album)	The_Beverly_Hills_Hotel	0.019315738
5	Hotel_California_(Eagles_album)	New_Kid_in_Town	0.0187292278

Highest % of readers on the same clickstream likely clicked: "The_Long_Run_(album)"

"The_Long_Run_(album)"



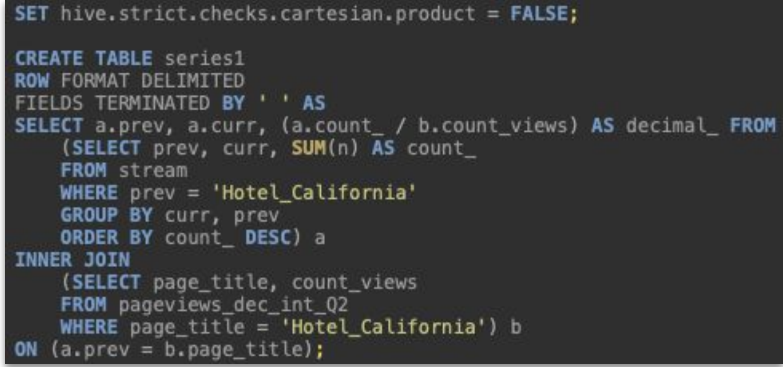
	prev	curr	percent_
1	The_Long_Run_(album)	Eagles_Live	0.1233653008
2	The_Long_Run_(album)	Hotel_California_(Eagles_album)	0.056146469
3	The_Long_Run_(album)	I_Can't_Tell_You_Why	0.0313862249
4	The_Long_Run_(album)	Heartache_Tonight	0.0294681779
5	The_Long_Run_(album)	The_Long_Run_(song)	0.0254577158

Highest % of readers on the same clickstream likely clicked: "Eagles_Live"

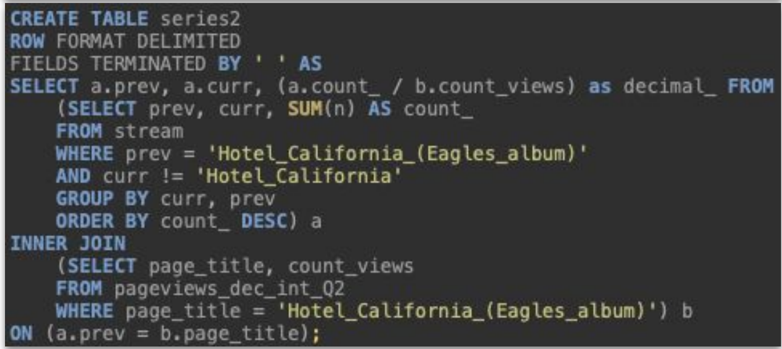
Question 3: Technical Process

Step 4

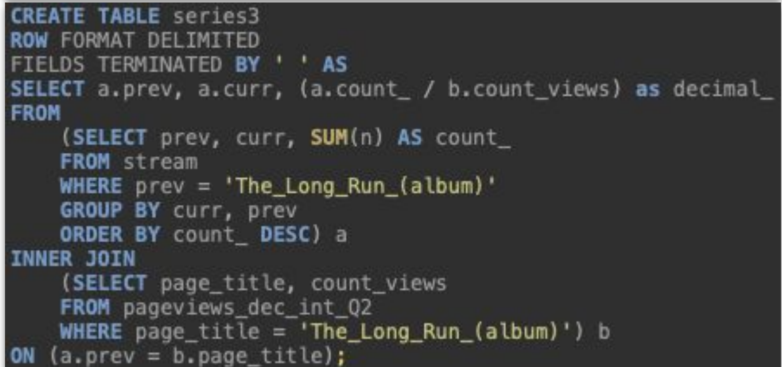
- To avoid excessive subqueries and save resources, utilize “create table as” on the previous select statements with inner joins.
- Disable strict checks for cartesian products so that it’s possible to create a table as a query with joined subqueries.



```
SET hive.strict.checks.cartesian.product = FALSE;

CREATE TABLE series1
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ' ' AS
SELECT a.prev, a.curr, (a.count_ / b.count_views) AS decimal_ FROM
    (SELECT prev, curr, SUM(n) AS count_
    FROM stream
    WHERE prev = 'Hotel_California'
    GROUP BY curr, prev
    ORDER BY count_ DESC) a
INNER JOIN
    (SELECT page_title, count_views
    FROM pageviews_dec_int_Q2
    WHERE page_title = 'Hotel_California') b
ON (a.prev = b.page_title);
```



```
CREATE TABLE series2
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ' ' AS
SELECT a.prev, a.curr, (a.count_ / b.count_views) as decimal_ FROM
    (SELECT prev, curr, SUM(n) AS count_
    FROM stream
    WHERE prev = 'Hotel_California_(Eagles_album)'
    AND curr != 'Hotel_California'
    GROUP BY curr, prev
    ORDER BY count_ DESC) a
INNER JOIN
    (SELECT page_title, count_views
    FROM pageviews_dec_int_Q2
    WHERE page_title = 'Hotel_California_(Eagles_album)') b
ON (a.prev = b.page_title);
```

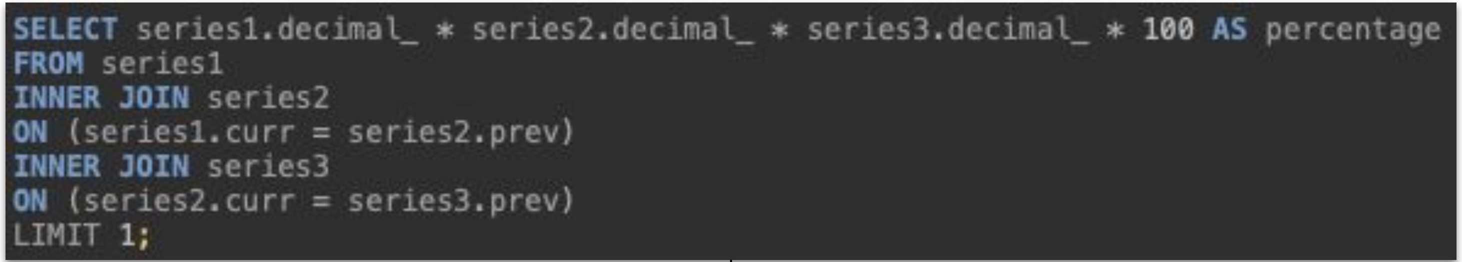


```
CREATE TABLE series3
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ' ' AS
SELECT a.prev, a.curr, (a.count_ / b.count_views) as decimal_
FROM
    (SELECT prev, curr, SUM(n) AS count_
    FROM stream
    WHERE prev = 'The_Long_Run_(album)'
    GROUP BY curr, prev
    ORDER BY count_ DESC) a
INNER JOIN
    (SELECT page_title, count_views
    FROM pageviews_dec_int_Q2
    WHERE page_title = 'The_Long_Run_(album)') b
ON (a.prev = b.page_title);
```

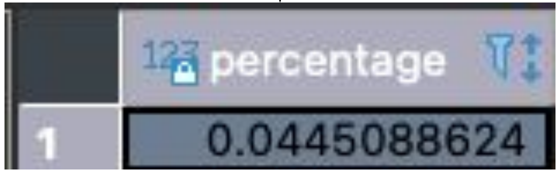
Question 3: Technical Process

Step 5

- Join the series tables and multiply their decimal results (x100) to calculate the percentage of the amount of viewers remaining after the whole clickstream series. This is the clickstream of Wikipedia articles, starting with Hotel California, followed by three clicks, which kept the largest fraction of its readers.



```
SELECT series1.decimal_ * series2.decimal_ * series3.decimal_ * 100 AS percentage
FROM series1
INNER JOIN series2
ON (series1.curr = series2.prev)
INNER JOIN series3
ON (series2.curr = series3.prev)
LIMIT 1;
```



```
   percentage
1  0.0445088624
```

Question 3: Result

Conclusion:

The series of articles, starting at Hotel_California, which kept the largest fraction of its readers clicking internal links, is **Hotel_California -> Hotel_California_(Eagles_album) -> The_Long_Run_(album) -> Eagles_Live**. **0.045%** of viewers of Hotel_California remain after this series of three clicks.

Additional comments on assumptions:

Without having user/session info, it cannot be determined how many times a user clicks a link from one page, and we cannot know a clickstream beyond a single prev, curr pair.

We cannot know if a specific user initially started at Hotel_California or its album page, and we cannot know whether they went back and forth, and how many times if so. We just know how many times the prev, curr pairs occurred.

We might assume the user will not click back to the link they were just on, *at least* as often as they would click to a different link they haven't been to, especially so considering the common practice of opening a link in a new tab. This line of reasoning was used to reach the conclusion.

Question 4: Summary

Goal:

Find an English wikipedia article that is relatively more popular in the U.S. than elsewhere.

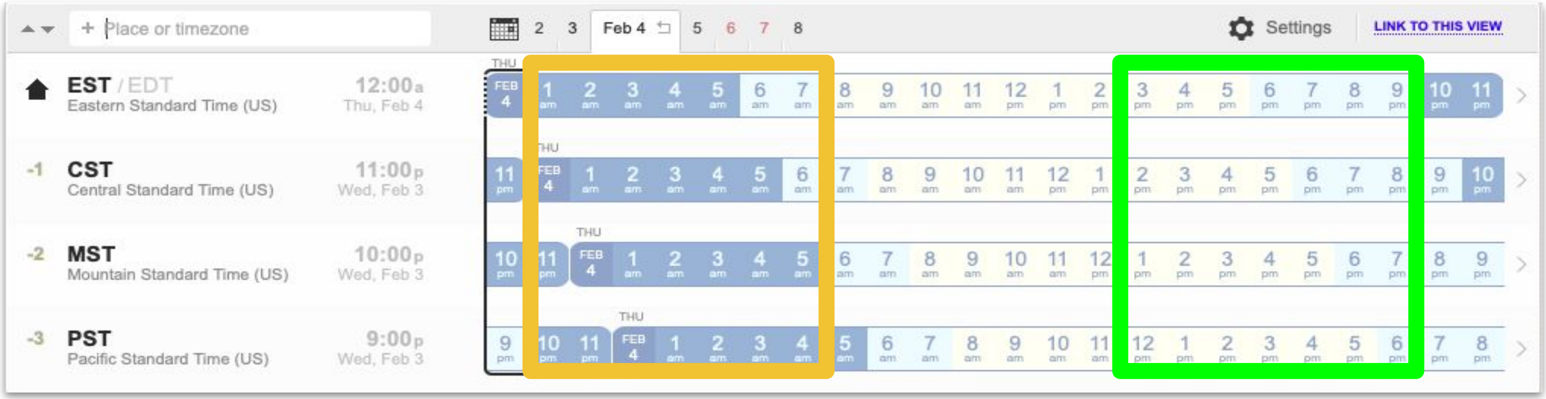
Key Assumptions Made:

- “Elsewhere” vs. “U.S.” exhibit similar amounts of internet traffic.
- Extrapolating page view info without attribution to the region it’s been created from, and instead only attribution to time of day and domain, will be sufficient.

Question 4: Technical Process

Step 1

- Determine a time range which most appropriately represents active and inactive hours for the majority of timezones in the U.S. I selected 3pm to 9pm EST to represent generally active hours, and 1 am to 7 am EST to represent generally inactive hours.



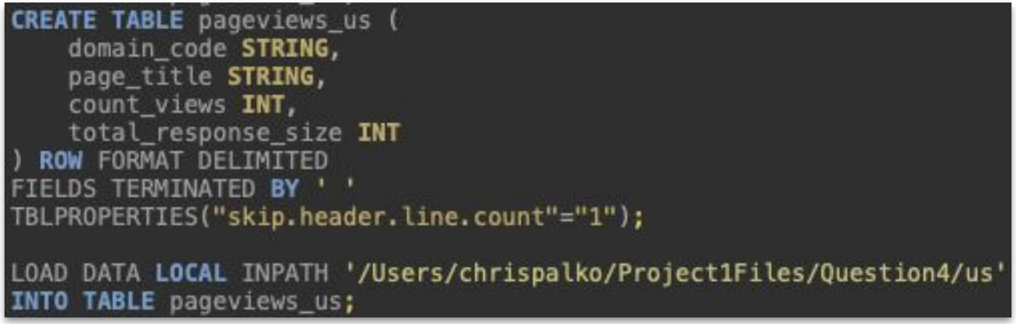
	EST / EDT Eastern Standard Time (US)	12:00a Thu, Feb 4	THU FEB 4	1 am	2 am	3 am	4 am	5 am	6 am	7 am	8 am	9 am	10 am	11 am	12 pm	1 pm	2 pm	3 pm	4 pm	5 pm	6 pm	7 pm	8 pm	9 pm	10 pm	11 pm
-1	CST Central Standard Time (US)	11:00p Wed, Feb 3	11 pm	THU FEB 4	1 am	2 am	3 am	4 am	5 am	6 am	7 am	8 am	9 am	10 am	11 am	12 pm	1 pm	2 pm	3 pm	4 pm	5 pm	6 pm	7 pm	8 pm	9 pm	10 pm
-2	MST Mountain Standard Time (US)	10:00p Wed, Feb 3	10 pm	11 pm	THU FEB 4	1 am	2 am	3 am	4 am	5 am	6 am	7 am	8 am	9 am	10 am	11 am	12 pm	1 pm	2 pm	3 pm	4 pm	5 pm	6 pm	7 pm	8 pm	9 pm
-3	PST Pacific Standard Time (US)	9:00p Wed, Feb 3	9 pm	10 pm	11 pm	THU FEB 4	1 am	2 am	3 am	4 am	5 am	6 am	7 am	8 am	9 am	10 am	11 am	12 pm	1 pm	2 pm	3 pm	4 pm	5 pm	6 pm	7 pm	8 pm

Source: Worldtimebuddy.com

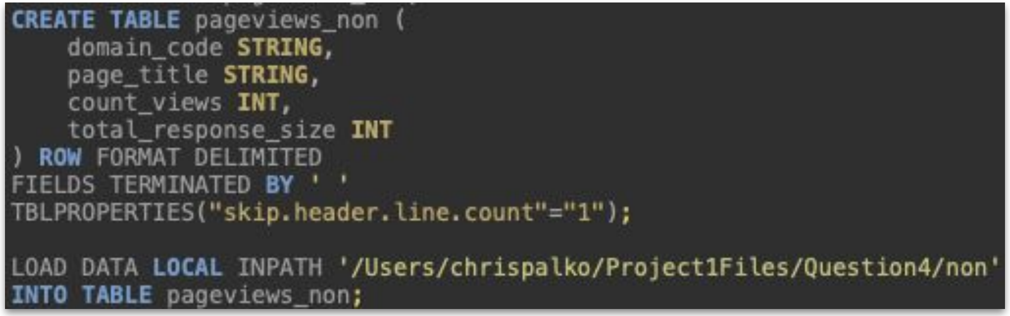
Question 4: Technical Process

Step 2

- Convert the previous ranges to UTC, since the Wikipedia data is recorded as such. 8 pm - 2 am UTC represents data more likely to be generated from the U.S. 5am-12pm UTC represents data more likely to be generated by elsewhere.
- Create two page-views tables representing those datasets and insert the files according to the hours correlated by region.



```
CREATE TABLE pageviews_us (
    domain_code STRING,
    page_title STRING,
    count_views INT,
    total_response_size INT
) ROW FORMAT DELIMITED
FIELDS TERMINATED BY ' '
TBLPROPERTIES("skip.header.line.count"="1");

LOAD DATA LOCAL INPATH '/Users/chrispalko/Project1Files/Question4/us'
INTO TABLE pageviews_us;
```

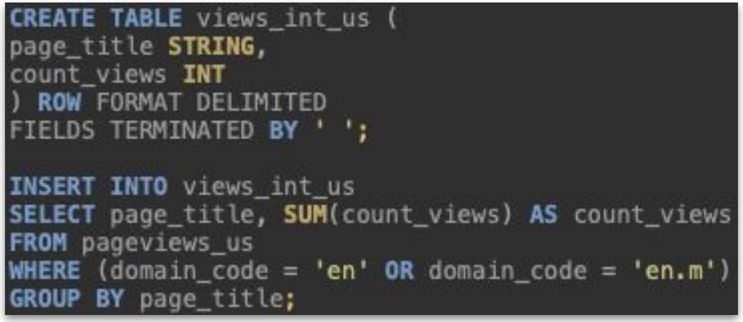


```
CREATE TABLE pageviews_non (
    domain_code STRING,
    page_title STRING,
    count_views INT,
    total_response_size INT
) ROW FORMAT DELIMITED
FIELDS TERMINATED BY ' '
TBLPROPERTIES("skip.header.line.count"="1");

LOAD DATA LOCAL INPATH '/Users/chrispalko/Project1Files/Question4/non'
INTO TABLE pageviews_non;
```

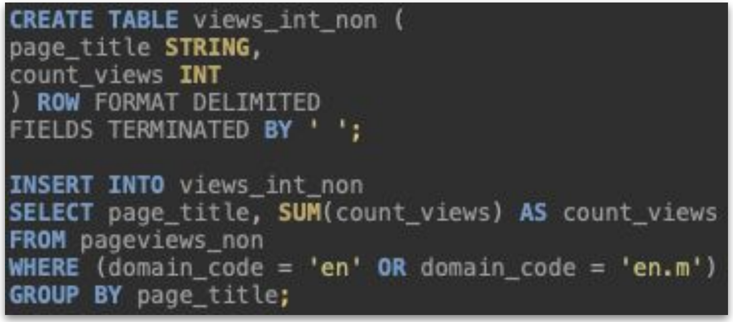
Question 4: Technical Process

Step 3

- Create then populate intermediate tables of the two prior tables.
- In the select statement to fill the table, exclude the now-irrelevant domain code column as well as the total response size column.
- Sum view-counts, filter out non-English domains, and group by page title.



```
CREATE TABLE views_int_us (
page_title STRING,
count_views INT
) ROW FORMAT DELIMITED
FIELDS TERMINATED BY ' ';

INSERT INTO views_int_us
SELECT page_title, SUM(count_views) AS count_views
FROM pageviews_us
WHERE (domain_code = 'en' OR domain_code = 'en.m')
GROUP BY page_title;
```

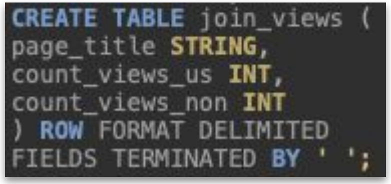


```
CREATE TABLE views_int_non (
page_title STRING,
count_views INT
) ROW FORMAT DELIMITED
FIELDS TERMINATED BY ' ';

INSERT INTO views_int_non
SELECT page_title, SUM(count_views) AS count_views
FROM pageviews_non
WHERE (domain_code = 'en' OR domain_code = 'en.m')
GROUP BY page_title;
```

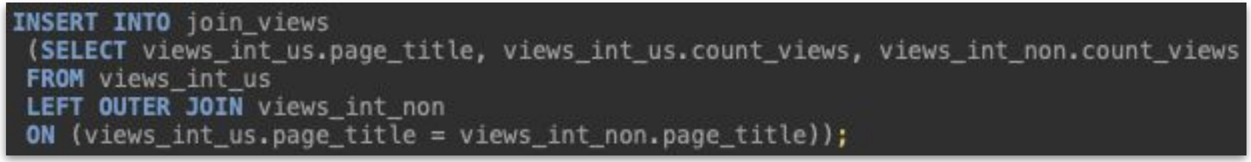
Question 4: Technical Process

Step 4

- Create a join views table to prepare to join the data, which will then be queried from.
- Insert into that table a join between the two intermediate tables on page titles.
- Choose a left outer join specifically because we want all page titles from the U.S. and matching page titles from elsewhere, but page titles that are only viewed elsewhere are irrelevant to the question.



```
CREATE TABLE join_views (
page_title STRING,
count_views_us INT,
count_views_non INT
) ROW FORMAT DELIMITED
FIELDS TERMINATED BY '  ';
```

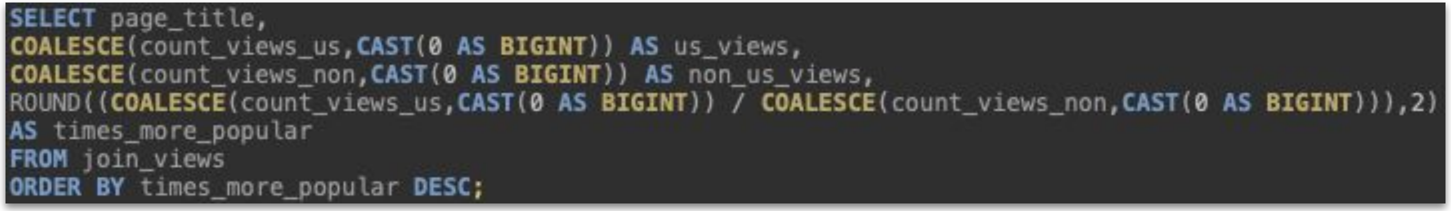


```
INSERT INTO join_views
 (SELECT views_int_us.page_title, views_int_us.count_views, views_int_non.count_views
 FROM views_int_us
 LEFT OUTER JOIN views_int_non
 ON (views_int_us.page_title = views_int_non.page_title));
```

Question 4: Technical Process & Result

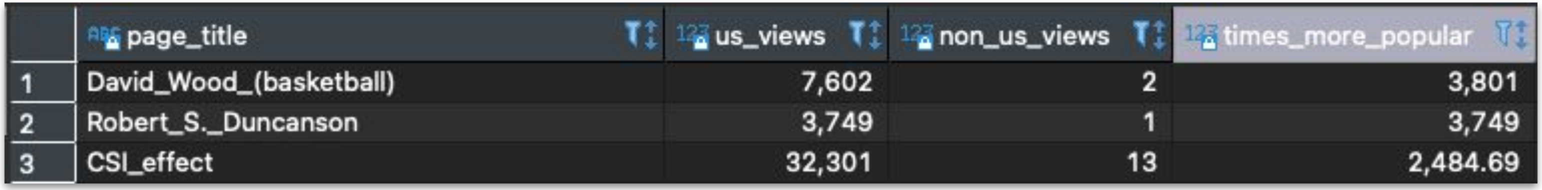
Step 5

- Retrieve the answer by dividing (U.S. views of pages) / (non-U.S. views of those pages), ordering by the highest (i.e. how many times more popular the article is in the simulated U.S. data). Set NULL values caused by the outer join to 0 first using coalesce.



```
SELECT page_title,
COALESCE(count_views_us,CAST(0 AS BIGINT)) AS us_views,
COALESCE(count_views_non,CAST(0 AS BIGINT)) AS non_us_views,
ROUND((COALESCE(count_views_us,CAST(0 AS BIGINT)) / COALESCE(count_views_non,CAST(0 AS BIGINT))),2)
AS times_more_popular
FROM join_views
ORDER BY times_more_popular DESC;
```

Result:



	page_title	us_views	non_us_views	times_more_popular
1	David_Wood_(basketball)	7,602	2	3,801
2	Robert_S._Duncanson	3,749	1	3,749
3	CSI_effect	32,301	13	2,484.69

- David Wood (basketball player) is the top result, being a rough-estimated 3801 times more popular in the U.S. vs. elsewhere.

Question 5: Summary

Goal:

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

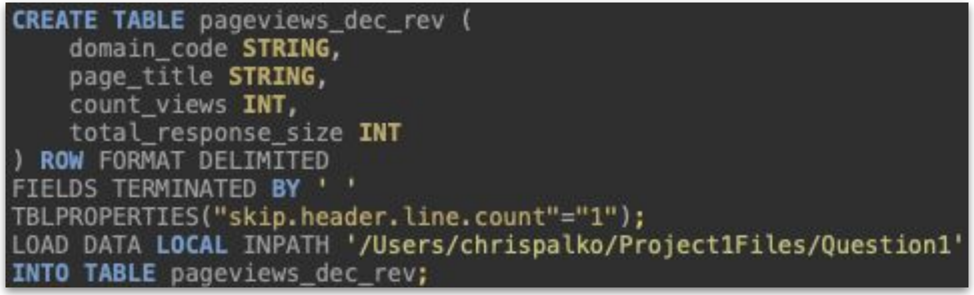
Key Assumptions Made:

- All vandalized articles are caught and reverted.
- All vandalized articles which are caught and reverted have a proper comment left indicating such.
- The sampled data (page-views and revisions) for one day in December 2020 is sufficient to represent all data to a degree.
- Data per day can be accurately extrapolated to data per hour. Seconds to identity reversion can be accurately extrapolated to hours.

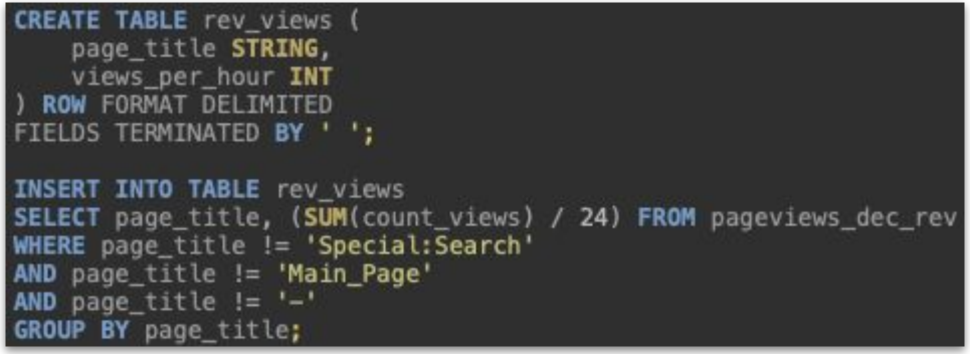
Question 5: Technical Process

Step 1

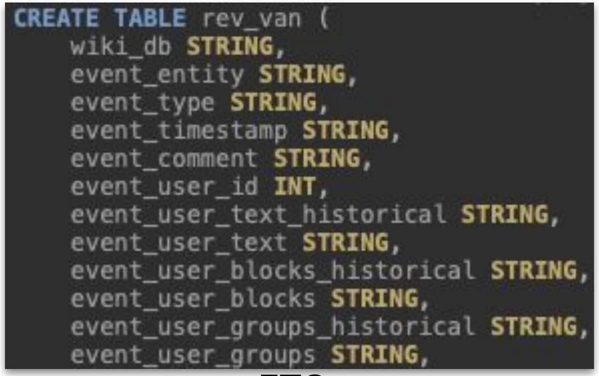
- Create a new page-views table, loading all views from Dec. 15th.
- Create an intermediate table, inserting data filtering out popular non-articles (main page, -, etc.), dividing all views by 24 to estimate views by hour, and grouping by page title.
- Create a table to then load page-revision data into.



```
CREATE TABLE pageviews_dec_rev (
    domain_code STRING,
    page_title STRING,
    count_views INT,
    total_response_size INT
) ROW FORMAT DELIMITED
FIELDS TERMINATED BY ' '
TBLPROPERTIES("skip.header.line.count"="1");
LOAD DATA LOCAL INPATH '/Users/chrispalko/Project1Files/Question1'
INTO TABLE pageviews_dec_rev;
```



```
CREATE TABLE rev_views (
    page_title STRING,
    views_per_hour INT
) ROW FORMAT DELIMITED
FIELDS TERMINATED BY ' ';

INSERT INTO TABLE rev_views
SELECT page_title, (SUM(count_views) / 24) FROM pageviews_dec_rev
WHERE page_title != 'Special:Search'
AND page_title != 'Main_Page'
AND page_title != '-'
GROUP BY page_title;
```



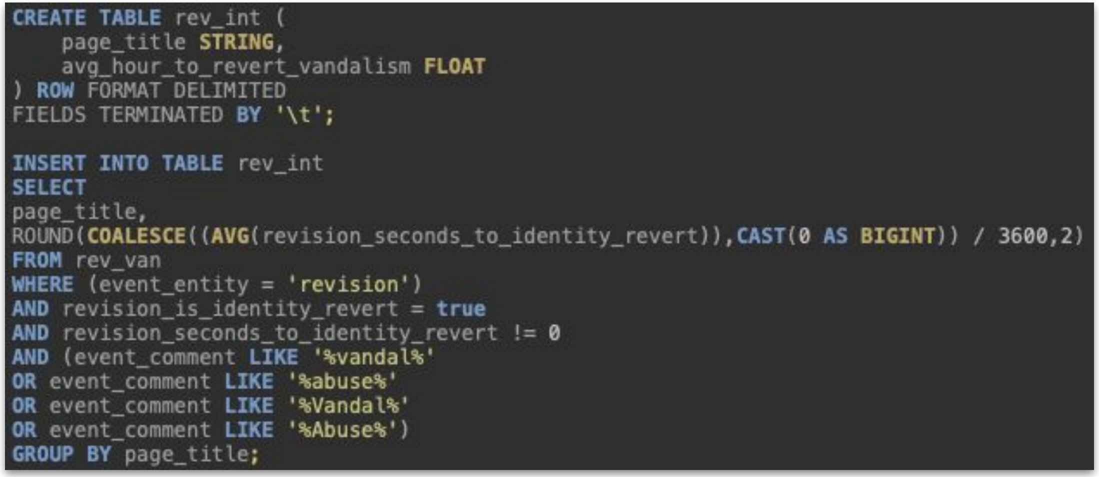
```
CREATE TABLE rev_van (
    wiki_db STRING,
    event_entity STRING,
    event_type STRING,
    event_timestamp STRING,
    event_comment STRING,
    event_user_id INT,
    event_user_text_historical STRING,
    event_user_text STRING,
    event_user_blocks_historical STRING,
    event_user_blocks STRING,
    event_user_groups_historical STRING,
    event_user_groups STRING,
```

ETC.

Question 5: Technical Process

Step 2

- Create an intermediate revisions table, inserting only page title and revision seconds to identity reversion.
- Filter inserted data to only include revision events, where it takes at least 1 second to revert (NULL values are converted to 0 with coalesce, then excluded).



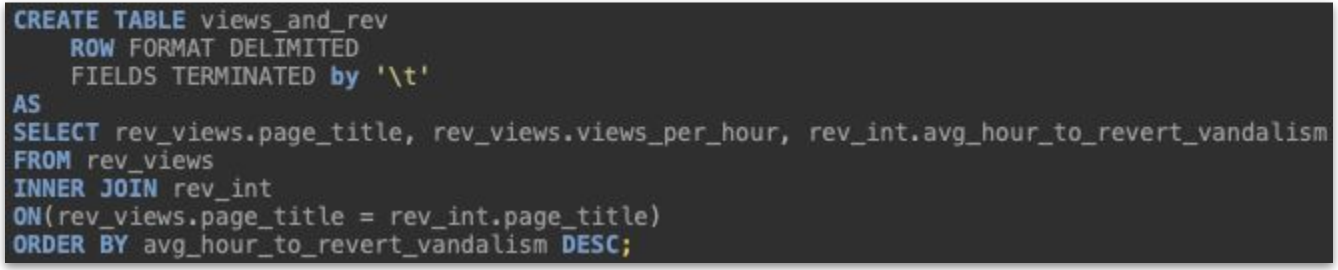
```
CREATE TABLE rev_int (
    page_title STRING,
    avg_hour_to_revert_vandalism FLOAT
) ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t';

INSERT INTO TABLE rev_int
SELECT
page_title,
ROUND(COALESCE((AVG(revision_seconds_to_identity_revert)),CAST(0 AS BIGINT)) / 3600,2)
FROM rev_van
WHERE (event_entity = 'revision')
AND revision_is_identity_revert = true
AND revision_seconds_to_identity_revert != 0
AND (event_comment LIKE '%vandal%'
OR event_comment LIKE '%abuse%'
OR event_comment LIKE '%Vandal%'
OR event_comment LIKE '%Abuse%')
GROUP BY page_title;
```

- Filter where the initial revision was reverted, and where the event comment mentions some form of vandalism or abuse.

Question 5:
Technical Process

Step 3

- Create a table as a select statement which joins the two intermediate tables on page title. This combines “views per hour” from page-views and “average hour to revert vandalism” (both per page) from the revision table to then be on the same table.

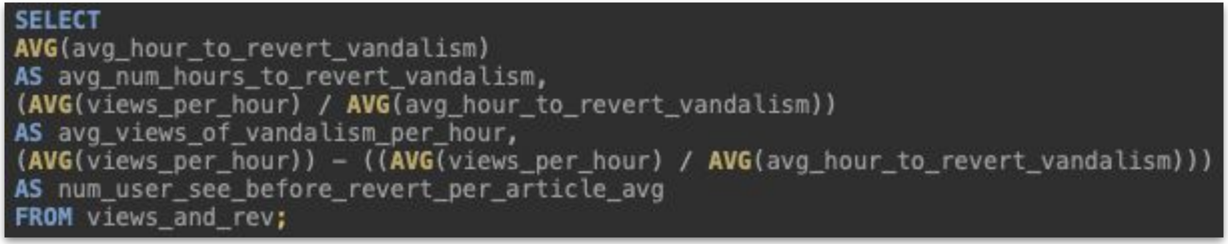


```
CREATE TABLE views_and_rev
    ROW FORMAT DELIMITED
    FIELDS TERMINATED by '\t'
AS
SELECT rev_views.page_title, rev_views.views_per_hour, rev_int.avg_hour_to_revert_vandalism
FROM rev_views
INNER JOIN rev_int
ON(rev_views.page_title = rev_int.page_title)
ORDER BY avg_hour_to_revert_vandalism DESC;
```

Question 5: Technical Process

Step 4

- Select the average hour to revert vandalism from each page, and average that. Select the (average views per hour) / (average hour to revert vandalism) to get average views of vandalism per hour, and lastly, select the (average views per hour) - (average views of vandalism per hour) to answer the primary question.

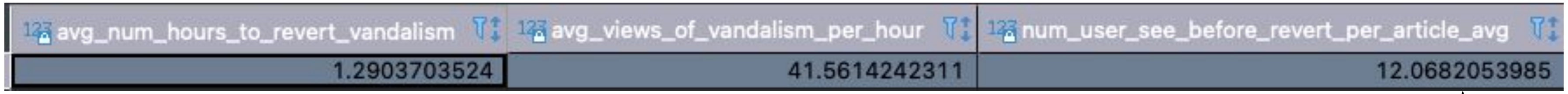


```
SELECT
AVG(avg_hour_to_revert_vandalism)
AS avg_num_hours_to_revert_vandalism,
(AVG(views_per_hour) / AVG(avg_hour_to_revert_vandalism))
AS avg_views_of_vandalism_per_hour,
(AVG(views_per_hour)) - ((AVG(views_per_hour) / AVG(avg_hour_to_revert_vandalism)))
AS num_user_see_before_revert_per_article_avg
FROM views_and_rev;
```

Question 5: Result

Conclusion:

~12.07 users will see the average vandalized wikipedia page before the offending edit is reversed.



avg_num_hours_to_revert_vandalism	avg_views_of_vandalism_per_hour	num_user_see_before_revert_per_article_avg
1.2903703524	41.5614242311	12.0682053985

Question 6: Summary

Goal:

-Determine the non-English domain codes with the top 5 most page-views for January 20th, 2021.

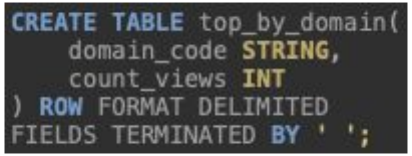
Key Assumption Made:

- Spider, bot, and illegitimate traffic was sufficiently filtered out from the prepared data.

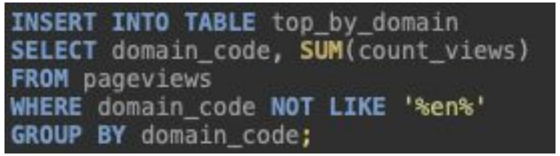
Question 6: Technical Process & Result

Step 1

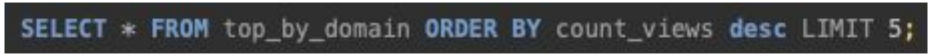
- Create an intermediate table from the original Pageviews table. Sum the views and group by domain code, filtering out non-English domains.



```
CREATE TABLE top_by_domain(
    domain_code STRING,
    count_views INT
) ROW FORMAT DELIMITED
FIELDS TERMINATED BY ' ';
```



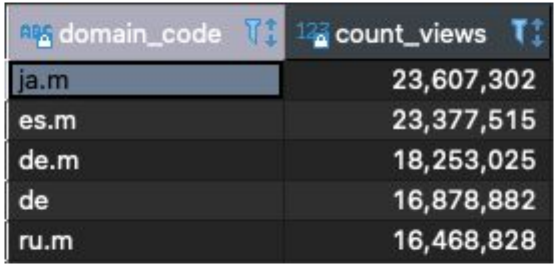
```
INSERT INTO TABLE top_by_domain
SELECT domain_code, SUM(count_views)
FROM pageviews
WHERE domain_code NOT LIKE '%en%'
GROUP BY domain_code;
```



```
SELECT * FROM top_by_domain ORDER BY count_views desc LIMIT 5;
```

Result:

1. Japan (mobile),
2. Spanish (mobile)
3. German (mobile)
4. German
5. Russian (mobile)



domain_code	count_views
ja.m	23,607,302
es.m	23,377,515
de.m	18,253,025
de	16,878,882
ru.m	16,468,828

GitHub Repo:

https://github.com/christian-palko/Wikipedia_Datasets_Analysis

Christian Palko • 02/05/2021