NoSQL Assignment 3

Deepkumar Patel
IMT2021011

Ricky Ratnani
IMT2021030

Shrey Salaria
IMT2021087

April 2024

Problem 1

Subproblem A

We must find the top ten frequently occurring predicates in the YAGO dataset. We follow the following steps: load the dataset, group it by predicate, generate predicate counts, order them in descending order, and limit the result to the top 10 predicates. The code can be found in top-10 pig file in the submission. see figure 1 We also wrote a python script to verify the answers and the answers matched. It took about 31 seconds to run.

```
-- YAGO Top Ten Predicates Analysis Script

-- step 1: load the tago dataset
yago_dataset = LOAD '../../data/yago.tsv' USING PigStorage(' ') AS (subject:chararray, predicate:chararray, object:chararray);

-- step 2: group the data by predicate
grouped_by_predicate = GROUP yago_dataset BY predicate;

-- step 3: count the occurences of each predicate
predicate_counts = FOREACH grouped_by_predicate GENERATE group AS predicate, COUNT(yago_dataset) AS count;

-- step 4: order by count and limit to top 10
ordered_predicates = ORDER predicate_counts by count DESC;
top_10_predicates = LIMIT ordered_predicates 10;

-- step 5: the results in a file
STORE top_10_predicates INTO 'part-a-result';;
```

Figure 1: Problem 1A

Subproblem B

We must identify all the given names of people associated with more than one LivesIn predicate from the YAGO dataset. We follow the following steps: load the dataset, filter the dataset with livesIn predicate, group by the subject, and find the ones with a count greater than one, now filter out the dataset which has the predicate hasGivenName and apply the join operation with the previous

result. see figure 2. We also wrote a python script to verify the answers and the answers matched. It took about 38 seconds to run

```
-- Yago Problem 1b

-- step 1: load the tago dataset
yago_dataset = LOAD '../../data/yago.tsv' USING PigStorage(' ') AS (subject:chararray, predicate:chararray, object:chararray);

-- step 2: filter the dataset for livesin
filter_livesin = FILTER yago_dataset BY predicate == '<livesIn>';
grouped_livesin = GROUP filter_livesin BY subject;
count_livesin = FOREACH grouped_livesin GENERATE group AS subject, COUNT(filter_livesin) as count;
more_than_one_livesin = FILTER count_livesin BY count > 1;

-- step 3: filter the dataset for hasgivenname
filter_hasgivenname = FILTER yago_dataset BY predicate == '<hasGivenName>';

-- step 4: join analysis
join_result = JOIN filter_hasgivenname BY subject, more_than_one_livesin BY subject;
final_result = FOREACH join_result GENERATE filter_hasgivenname::object;

-- step 4: store result
STORE final_result INTO 'part-b-result';;

```

Figure 2: Problem 1B Pig Script

Problem 2

We take inspiration from the source code of the implementation of the COUNT aggregation function in PIG, we slightly modify the implementation of the COUNT and take the subject-predicate pair as part of the input tuple along with the data bag. We iterate over the tuples in the data bag and check if the current subject predicate pair matches to the input pair and increment a counter variable accordingly.

We define the `class MyUDF extends EvalFunc<Long> implements Algebraic` just like COUNT.

The implementation of the UDF can be found in the files, we can look at the usage code in figure 3. One can also find the jar file in the submission. While trying to make the jar file, we iteratively found out which library we needed and included them, in the end, we needed the jar files of Apache Pig, Apache Hadoop, and Apache Commons. It took about 32 seconds to run.

```
-- USER DEFINED FUNCTIon

-- step 1: load the tago dataset
yago_dataset = LOAD '../data/yago.tsv' USING PigStorage(' ') AS (subject:chararray, predicate:chararray, object:chararray);
-- yago_dataset = LIMIT yago_dataset 100;

-- step 2: group all the data together
yago_all = GROUP yago_dataset ALL;

-- step 3: get the user defined function
REGISTER 2.jar;
count = FOREACH yago_all GENERATE group as yago_all, MyUDF(yago_dataset, '<Ron_Mann>', '<directed>') AS count;
STORE count INTO 'group-all-udf-trial-16';
```

Figure 3: Problem 2 Pig Script

Problem 3

Subproblem A

Objectives:

- Load the YAGO dataset
- Find out the top three frequently occurring predicates in the YAGO dataset using operators available in the HiveQL.

Creating the schema for YAGO dataset and loading the data:

```
CREATE TABLE yago_table (
    subject STRING,
    predicate STRING,
    object STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ' '
LOCATION '/my_hive_tables/yago_table';
```

Figure 4: yago schema create

```
LOAD DATA LOCAL INPATH '/path_to_yago_dataset/yago_full_clean.tsv' INTO TABLE yago_table;
```

Figure 5: yago load data

Each row of the provided yago dataset contains three values. Namely subject, predicate and object all of which are strings. These vales are seperated from a single white space.So in figure 4 the delimeter is set as a single white space. And at the very end the location of the directory where the table is to be created in the hadoop dfs is provided.

Getting the top 3 most frequently used predicates

```
SELECT predicate, COUNT(*)
FROM yago_table
GROUP BY predicate
ORDER BY COUNT(*) DESC
LIMIT 3;
```

(a) Query for top 3 predicates

predicate	_c1
<isCitizenOf>	2141725
<hasFamilyName>	2002574
<hasGender>	1972842

(b) Top 3 predicates

Figure 6: Getting Top 3 predicates

To select the top 3 most frequently used predicated we first select the predicate and count(*) from the yago table and group it by the predicate. Once we group it by the predicates we can order by the count(*). Here the count(*) will provide the frequency for each predicate as we have already grouped by the predicate. Then finally to get the top 3 we order it in descending order and then limit the output rows by 3 using LIMIT 3.

Subproblem B

Problem statement :

Identify all the given-names (i.e., the object values of the hasGivenName predicate) of persons who are associated with more than one livesIn predicates from the YAGO dataset using the relational operators (join, grouping, etc.) in the HiveQL

```
SELECT names.object
FROM (SELECT object FROM yago_table WHERE predicate="<hasGivenName>") as names
JOIN (SELECT subject from yago_table WHERE predicate="<livesIn>" GROUP BY subject HAVING count(*)>1) as temp
ON(temp.subject == names.object);
```

Figure 7: part b query

Description :

As given in the problem statement we will need the persons who are associated with more than one livesIn predicate. So for this we can just set a condition in WHERE clause that predicate = ⟨livesIn⟩ and then group by the subject. This will result in a table of all subjects which are associated with livesIn predicate. But as we want only those subjects which are associated with livesIn predicate more than once we will use the condition of count greater than 1 in HAVING clause.

Now we have a table of all subjects which are associated with livesIn predicate more than once.

Now finally we want a table of all objects values of hasGivenName predicate who are associated with more than one livesIn predicate. So for this we will first get the table of all the objects from the yago table whose preidicate is hasGivenName and then simply take the inner join of our two tables with the condition being subject of first table equals object of the second table we talked about.

So for getting the table of all the objects from the yago table whose preidicate is hasGivenName we can simply select the objects and keep the condition in WHERE clause that predicate = ⟨hasGivenName⟩.

Finally we take the inner join of them as discussed above. This inner join is shown by word JOIN in the query. Thus after this we have the final table that was asked in the problem statement.

Problem 4

Partitioning and Bucketing

Data is stored in separate partitions based on the specified partition key (which is the predicate in this case). This helps in organizing and accessing data efficiently.
Within each partition, data is further divided into buckets based on the clustering key (which is the subject in this case), which enables more targeted and optimized join operations.
The use of partitioning and bucketing can significantly improve query performance, especially for join operations, as it reduces the amount of data that needs to be scanned and processed.

Following is the output obtained on running the commands in hivepart1.txt on yago table ⇒

```
Total MapReduce CPU Time Spent: 13 minutes 17 seconds 110 msec
OK
Time taken: 1230.987 seconds
```

Figure 8: Output

Partitioning but not Bucketing

Data is still stored in separate partitions based on the partition key.
Here, there is no additional bucketing structure within each partition.
Partitioning can still provide benefits in terms of data organization and partition pruning, but the lack of bucketing may lead to less optimized join operations as compared to the previous case.

Following is the output obtained on running the commands in hivepart2.txt on yago table⇒

```
Total MapReduce CPU Time Spent: 2 minutes 12 seconds 910 msec
OK
Time taken: 122.882 seconds
```

Figure 9: Output

Neither Partitioning nor Bucketing

Data is stored without any specific partitioning or bucketing, leading to a flat storage structure.
Without partitioning or bucketing, queries need to scan the entire dataset, resulting in higher resource consumption and longer query execution times, especially for join operations.
As a result, this approach is generally less efficient for large-scale datasets.

Following is the output obtained on running the commands in hivepart3.txt on yago table⇒

```
Time taken: 38.656 seconds, Fetched: 67028 row(s)
```

Figure 10: Output