Assignment 3 - Organizing Tweets
W205.1 - STORING AND RETRIEVING DATA
Vineet Gangwar

For each database category of Key/Value, NoSQL, and Relational, describe a systems architecture that contains:

- *Your implementation model of how data is actually organized (e.g. a schema, collection structure, etc.).*
- *The process necessary to store the information into your implementation model.*
- *Pseudo-code / procedures that describe how you would answer each of the questions.*

Table of Contents

---

1 https://www.sqlite.org/lang_datefunc.html
2 https://www.sqlite.org/datatype3.html

Relational Sqlite - Systems Architecture

Implementation model of how data is organized

I have used Sqlite as the relational database. Sqlite can store data in either disk or memory. For the purpose of this assignment I stored on disk.
Data is stored in two tables - tweets and hashtags.
Schema of **tweets** table:

Field Name	Data Type	Description
tweetKey	INTEGER	Ranging from 0 to 766. It is unique for every tweet in both the input json files
username	TEXT	User name from tweet['user']['screen_name']
datetimeStamp	TEXT	Date time stamp from tweet['created_at'] converted into Sqlite TimeString format[1] of *YYYY-MM-DD***T***HH:MM:SS*

Schema of **hashtags** table:

Field Name	Data Type	Description
tweetKey	INTEGER	Used as a foreign key to link to tweets table. This means that tweetKey is not an unique field in this table
hashtag	TEXT	Hashtags from tweet['entities']['hashtags']. Every row in this table as a single hashtag
datetimeStamp	TEXT	Date time stamp from tweet['created_at'] converted into Sqlite TimeString format[1] of *YYYY-MM-DD***T***HH:MM:SS*

Process to store information into the implementation model

I am using a two phase approach to store information into the implementation model:
Phase 1:

1. Read all the tweets from the two input json files and store them as a list of tweets
2. Iterate through the list and perform the following steps:
    a. Create a unique identifier for each tweet. This is just the position of the tweet in the list created above
    b. Extract the username from tweet['user']['screen_name']
    c. Extract timestamp from tweet['created_at'] and convert it into Sqlite TimeString format[1]
    d. Create a tuple of the above three fields for insertion into tweets table
    e. Extract the list of hashtags for each tweet from tweet['entities']['hashtags']
    f. Create a tuple using the fields from steps (a), (c) and (e) above for insertion into hashtags table
    g. Create a dictionary of the tuples created in steps (d) and (f) above
    h. Yield the dictionary created in step (g) above to phase 2.

1 https://www.sqlite.org/lang_datefunc.html
2 https://www.sqlite.org/datatype3.html

Phase 2:

1. Use Sqlite cursor to directly insert the tuple created in step (d) above to the tweets table
2. Iterate through the hashtags list created in step(f) above and perform the following steps:
    a. Create a tuple with the following elements:
        i. tweetKey as created in step(a) of phase 1
        ii. datetimeStamp as created in step(c) of phase 1
        iii. A single hashtag from the list
    b. Insert the tuple into the hashtags table

Pseudo-code / procedures to answer each of the questions

Most tweeted - (between 9:00 +1:00 and 16:00 +1:00 on both the days)

The most tweeted is obtained by an SQL query that has the following:

1. SELECT username from the tweets table, then
2. count GROUP BY to obtain the number of tweets per user, then
3. ORDER BY DESC to order the list, then
4. LIMIT the output to 1 to get the username that tweeted the most

The WHERE clause of the SELECT statement contains the date and time restriction. Since Sqlite does not have a date data type[2] so I performed date calculations by converting dates into epoch time using the function strftime()

Output from the data:

Mosted Tweeted

xmlprague	*Number of tweets:47*

Top 10 hashtags used - (between 9:00 +1:00 and 16:00 +1:00 on both the days)

The top 10 hashtags is obtained by an SQL query that has the following:

1. SELECT hashtag from the hashtags table, then
2. count GROUP BY to obtain the number of hashtags per hashtag, then
3. ORDER BY DESC to order the list, then
4. LIMIT the output to 10 to get the top 10 most used hashtags

The WHERE clause of the SELECT statement contains the date and time restriction. Since Sqlite does not have a date data type[2] so I performed date calculations by converting dates into epoch time using the function strftime()

Output from the data:

Top 10 Hashtags

xmlprague	*Number of tweets:441*
thetransformationsong	*Number of tweets:22*
XMLPrague	*Number of tweets:20*
XProc	*Number of tweets:15*
xproc	*Number of tweets:9*
existdb	*Number of tweets:8*
RDFa	*Number of tweets:7*
BRILLIANT	*Number of tweets:6*
CreditWhereDue	*Number of tweets:6*
FUCKYEAH	*Number of tweets:6*

1 https://www.sqlite.org/lang_datefunc.html
2 https://www.sqlite.org/datatype3.html

Tweets per hour - (between 9:00 +1:00 and 16:00 +1:00 on both the days)

Tweets per hour is obtained by an SQL query that has the following:

1. SELECT the datetimeStamp field from the tweets table. Then extract the date and hour to create a new field called date_hr. e.g. ‘2015-02-14T06:23:57’ will be converted to ‘2015-02-14 08’, then
2. count GROUP BY the new field date_hr

The WHERE clause of the SELECT statement contains the date and time restriction. Since Sqlite does not have a date data type[2] so I performed date calculations by converting dates into epoch time using the function strftime()

Output from the data:

Tweets per Hour

```
2015-02-14 08        Number of tweets:46
2015-02-14 09        Number of tweets:55
2015-02-14 10        Number of tweets:19
2015-02-14 11        Number of tweets:42
2015-02-14 12        Number of tweets:9
2015-02-14 13        Number of tweets:24
2015-02-14 14        Number of tweets:24
2015-02-15 08        Number of tweets:21
2015-02-15 09        Number of tweets:56
2015-02-15 10        Number of tweets:16
2015-02-15 11        Number of tweets:66
2015-02-15 12        Number of tweets:13
2015-02-15 13        Number of tweets:13
2015-02-15 14        Number of tweets:41
```

NoSQL MongoDB - Systems Architecture

Implementation model of how data is organized

For MongoDB, the collection structure will have two collections named tweets and hashtags.

Structure of tweets collection:

{‘tweetKey’:tweetKey, 'username':username, 'datetimeStamp':datetimeStamp}

Structure of hashtags collection:

{‘tweetKey’:tweetKey, ‘hashtag’:hashtag, 'datetimeStamp':datetimeStamp}

Data types:

Field Name	**Data type**	**Description**
tweetKey	INTEGER	Ranging from 0 to 766. It is unique for every tweet in both the input json files
datetimeStamp	DATE	Date time stamp from tweet['created_at'] converted into mongoDB Date data type using ISODate()
username	STRING	User name from tweet['user']['screen_name']

1 https://www.sqlite.org/lang_datefunc.html

2 https://www.sqlite.org/datatype3.html

hashtag	STRING	Hashtags from tweet['entities']['hashtags']. Every row in this table as a single hashtag

Process to store information into the implementation model

This will require a two phase approach to store information into the implementation model:
Phase 1:

1. Read all the tweets from the two input json files and store them as a list of tweets
2. Iterate through the list and perform the following steps:
    a. Create a unique identifier for each tweet. This is just the position of the tweet in the list created above
    b. Extract the username from tweet['user']['screen_name']
    c. Extract timestamp from tweet['created_at'] and convert it into the format that can be input into ISODate(). The format is - "YYYY-MM-DDTHH:MM:SS.SSSZ"
    d. Create a dictionary of the above three fields for insertion into tweets collection
    e. Extract the list of hashtags for each tweet from tweet['entities']['hashtags']
    f. Create a dictionary using the fields from steps (a), (c) and (e) above for insertion into hashtags collection
    g. Create a dictionary of the dictionaries created in steps (d) and (f) above
    h. Yield the dictionary created in step (g) above to phase 2.

Phase 2:

1. Directly insert the tuple created in step (d) above to the tweets collection
   e.g. db.tweets.insert({'tweetKey':tweetKey, 'username':username, 'datetimeStamp': ISODate(timestamp)})
2. Iterate through the hashtags list created in step(f) above and perform the following steps:
    c. Create a dictionary with the following elements:
        i. tweetKey as created in step(a) of phase 1
        ii. timestamp created in step(c) of phase 1
        iii. A single hashtag from the list
    d. Insert the dictionary into the hashtags table

Pseudo-code / procedures to answer each of the questions

Most tweeted - (between 9:00 +1:00 and 16:00 +1:00 on both the days)

Use aggregate method on tweets collection to answer this question. The method should use the following aggregation pipeline:

Stage Operators	**Remarks**
$match	To filter tweets between 9:00 +1:00 and 16:00 +1:00 on both the days
$group	On $username to get count per username
$sort	To sort descending on count
$limit	To return just the top 1 document

1 https://www.sqlite.org/lang_datefunc.html
2 https://www.sqlite.org/datatype3.html

Top 10 hashtags used - (between 9:00 +1:00 and 16:00 +1:00 on both the days)
Use aggregate method on hashtags collection to answer this question. The method should use the following aggregation pipeline:

Stage Operators	Remarks
$match	To filter tweets between 9:00 +1:00 and 16:00 +1:00 on both the days
$group	On $hashtag to get count per hashtag
$sort	To sort descending on count
$limit	To return just the top 10 documents

Tweets per hour - (between 9:00 +1:00 and 16:00 +1:00 on both the days)
Use aggregate method on tweets collection to answer this question. The method should use the following aggregation pipeline:

Stage Operators	Remarks
$match	To filter tweets between 9:00 +1:00 and 16:00 +1:00 on both the days
$project	To compute a new field using date and just the hour part. e.g. '2015-02-14T06:23:57' will be converted to '2015-02-14 08'
$group	On the newly computed field above to get the count of tweets per hour for each day

Key/Value AWS S3 - Systems Architecture

Implementation model of how data is organized
Data will stored in two sets of keys.
First set of keys will store username information:

Key	Value
s3://myBucket/tweets/1	tweetKey, username, datetimeStamp
s3://myBucket/tweets/2	tweetKey, username, datetimeStamp
s3://myBucket/tweets/3	tweetKey, username, datetimeStamp
...	...

1 https://www.sqlite.org/lang_datefunc.html
2 https://www.sqlite.org/datatype3.html

Second set of keys will store hashtags information

Key	**Value**
s3://myBucket/hashtags/1	tweetKey, hashtag, datetimeStamp
s3://myBucket/hashtags/2	tweetKey, hashtag, datetimeStamp
s3://myBucket/hashtags/3	tweetKey, hashtag, datetimeStamp
...	...

Field Name	**Description**
tweetKey	Ranging from 0 to 766. It is unique for every tweet in both the input json files
datetimeStamp	Date time stamp from tweet['created_at'] converted into epoch time
username	User name from tweet['user']['screen_name']
hashtag	Hashtags from tweet['entities']['hashtags']. Every row in this table as a single hashtag

Process to store information into the implementation model

This will require a two phase approach to store information into the implementation model:

Phase 1:

1. Read all the tweets from the two input json files and store them as a list of tweets
2. Iterate through the list and perform the following steps:
    a. Create a unique identifier for each tweet. This is just the position of the tweet in the list created above
    b. Extract the username from tweet['user']['screen_name']
    c. Extract timestamp from tweet['created_at'] and convert it into epoch time
    d. Create a csv string of the above three fields to used as value to store usernames
    e. Extract the list of hashtags for each tweet from tweet['entities']['hashtags']
    f. Create a list using the fields from steps (a), (c) and (e) above to store hashtags
    g. Create a dictionary of the items created in steps (d) and (f) above
    h. Yield the dictionary created in step (g) above to phase 2.

Phase 2:

1. Use the csv string created in step (d) above to create a new key/value pair in S3. The key will look like s3://myBucket/tweets/1. Increment the last number of the key at every insert
2. Iterate through the hashtags list created in step(f) above and perform the following steps:
    a. Create a csv string with the following elements:
        i. tweetKey as created in step(a) of phase 1
        ii. timestamp created in step(c) of phase 1
        iii. A single hashtag from the list

1 https://www.sqlite.org/lang_datefunc.html
2 https://www.sqlite.org/datatype3.html

b. Create a new key/value pair in S3. The key will look like s3://myBucket/hashtags/1. Increment the last number of the key at every insert

Pseudo-code / procedures to answer each of the questions

Most tweeted - (between 9:00 +1:00 and 16:00 +1:00 on both the days)

The most tweeted can obtained by the following:

1. Read all key/value pairs with the prefix S3://myBucket/tweets and store the values in a list.
2. Iterate through the list and discard all items that do not satisfy the time constraints. This can be done by converting the four time constraints into epoch and doing a simple logical operation of less/greater than with the epoch time stored in the list
3. Use python to do the equivalent SQL equivalent of GROUP BY, SORT and LIMIT. These can be achieved in the following way:
    a. Create a list of usernames
    b. Create a unique set of usernames by converting the list of usernames into a set
    c. Use the count method of python list object to find out the number. This is equivalent to GROUP BY of SQL
    d. Then use sorted() to sort in reverse order
    e. Then print the first item
    f. Sample code:

```
uniqueUsers = set(userList)
tempList = list()
for item in uniqueUsers:
    tempList.append([userList.count(item), item])
tempList = sorted(tempList, reverse = True)
print tempList[0]
```

Top 10 hashtags used - (between 9:00 +1:00 and 16:00 +1:00 on both the days)

Top 10 hashtags can obtained by the following:

1. Read all key/value pairs with the prefix S3://myBucket/hashtags and store the values in a list.
2. Iterate through the list and discard all items that do not satisfy the time constraints. This can be done by converting the four time constraints into epoch and doing a simple logical operation of less/greater than with the epoch time stored in the list
3. Use python to do the equivalent SQL equivalent of GROUP BY, SORT and LIMIT. These can be achieved in the following way:
    a. Create a list of hashtags
    b. Create a unique set of hashtags by converting the list of hashtags into a set
    c. Use the count method of python list object to find out the number. This is equivalent to GROUP BY of SQL
    d. Then use sorted() to sort in reverse order
    e. Then print the first item
    f. Sample code:

```
uniqueHashtags = set(hashtagsList)
tempList = list()
for item in uniqueHashtags:
    tempList.append([hashtagsList.count(item), item])
```

1 https://www.sqlite.org/lang_datefunc.html
2 https://www.sqlite.org/datatype3.html

```
tempList = sorted(tempList, reverse = True)
print tempList[0:9]
```

Tweets per hour - (between 9:00 +1:00 and 16:00 +1:00 on both the days)

Tweets per hour can be obtained by the following:

1. Read all key/value pairs with the prefix S3://myBucket/tweets and store the values in a list.
2. Iterate through the list and discard all items that do not satisfy the time constraints. This can be done by converting the four time constraints into epoch and doing a simple logical operation of less/greater than with the epoch time stored in the list
3. Use python to do the SQL equivalent of GROUP BY, SORT and LIMIT. These can be achieved in the following way:
    a. Create a list of a new field that consists of date and hour. This can be done by converting the epoch time stored in the csv string in S3. The new field called DateHR will look like '2015-02-14 10'
    b. Create a unique set of new field created above by converting it into a set
    c. Use the count method of python list object to find out the number. This is equivalent to GROUP BY of SQL
    d. Sample code:

```
uniqueDateHr = set(DateHrList)
tempList = list()
for item in uniqueDateHr:
    tempList.append([DateHrList.count(item), item])
print tempList
```

1 https://www.sqlite.org/lang_datefunc.html
2 https://www.sqlite.org/datatype3.html