**A step by step guide to mongo aggregations**

#### retrieveUsersInformation()

results = collection.aggregate(Arrays.asList(

new Document("$group", new Document("\_id","$userID")

.append("value", new Document("$sum", 1))),

new Document("$sort", new Document("value", -1))));

Step 1

new Document("$group", new Document("\_id","$userID")

.append("value", new Document("$sum", 1)))

In this step we aggregate the data present in the mongoDB collection Search by userID and for each document for that user we sum 1 (so we are counting the number of Search for each user).

Step 2

new Document("$sort", new Document("value", -1))));

In this step we sort our users by the number of search done. Since we are more interested in users with higher number of search done we order them in a decreasing order

#### calculateTrendingKeyWords()

//ValueOfKeyword=NumberOfArticles^2\*Occ

results = collection.aggregate(Arrays.asList(

new Document("$match", new Document("date", new Document("$gt",

queryDate))),

new Document("$unwind", "$Keywords"),

new Document("$group", new Document("\_id", "$Keywords.keyword")

.append("Occur", new Document("$sum", "$Keywords.Occ"))

.append("NumberOfArticles", new Document("$sum", 1))),

new Document("$project",new Document("\_id",1)

.append("Value", new Document("$multiply",indexes) )),

new Document("$sort", new Document("Value", -1)),

new Document("$limit",500)));

Step 1

new Document("$match", new Document("date", new Document("$gt",queryDate))),

Since we are interested only in the newest articles to calculate the trading word by the match operand we select only the Articles present in our article collection that have the date greater that the queryDate. The quaryDate variable contains the date of our access with a subtraction of a fixed number of days.

Step 2

new Document("$unwind", "$Keywords"),

Since every article has an array containing all his keywords in the format

1. { "keyword" : <keyword\_name>, "Occ" : <keyword\_number\_of\_occurencies>}

 To perform the aggregation on each keyword for every article we need to obtain a single document for each element of the keywords array (since we need to determinate the number of occurrences of each <keyword\_name> and the number of total article in which the keyword is present)

Step 3

new Document("$group", new Document("\_id", "$Keywords.keyword")

.append("Occur", new Document("$sum", "$Keywords.Occ"))

.append("NumberOfArticles", new Document("$sum", 1))),

In this step we aggregate each keyword in the articles selected by the previous match operation.

We calculate the total number of occurrences of a specific keyword by summing the number of occurrences on each article in which the keyword appears.

We calculate the number of Articles in which the keyword appears by counting in how many documents the keyword is present (since each article has at most one array element with a specific keyword this sum corresponds exactly to the number of article in which the keyword is present)

Step 4

new Document("$project",new Document("\_id",1)

.append("Value", new Document("$multiply",indexes) )),

The we have documents in the format

1. {“\_id”:<keyword\_name>, “Occur”:<total\_number\_of\_occurency\_for\_keyword\_name>, “NumberOfArticle”: ”:<total\_number\_of\_articles\_for\_keyword\_name>}

 We need to project only the fields we need : The <keyword\_name> and his value measured by the formula (NumberOfOccurences\*NumberOfArticles^2).

We specified that the “\_id” field is needed in our projection.

We stored the in indexed the field we have to multiply:

1. ArrayList<String> indexes=new ArrayList<>();
2. indexes.add("$Occur");
3. indexes.add("$NumberOfArticles");
4. indexes.add("$NumberOfArticles");

This is useful so we can call the multiply instruction to calculate the above formula by passing this array of string.

Step 5 and 6

new Document("$sort", new Document("Value", -1)),

new Document("$limit",500)));

We order our result in a decreasing order of value and limit the document to 500 (so we have the 500 most influent word of the period)

results = collection.aggregate(Arrays.asList(

Aggregates.match(and(eq("userID",u.userID),gte("dateRead",queryDate))),

new Document("$group", new Document("\_id", "$filters")

.append("value", new Document("$sum", 1))),

new Document("$sort", new Document("value", -1)),

new Document("$limit", 3)));

#### suggestedArticles(User u)

results = collection.aggregate(Arrays.asList(

Aggregates.match(and(eq("userID",u.userID),gte("dateRead",queryDate))),

new Document("$group", new Document("\_id", "$filters")

.append("value", new Document("$sum", 1))),

new Document("$sort", new Document("value", -1)),

new Document("$limit", 3)));

Step 1

Aggregates.match(and(eq("userID",u.userID),gte("dateRead",queryDate))),

From the Search collection (that contains information about each article read by the user, the filter used by the user to find that specific article and the date of search ) since this query is invoked every time an user log in our application, this query retrieves all the searches related to that user for a specific period.

Step 2

new Document("$group", new Document("\_id", "$filters")

.append("value", new Document("$sum", 1))),

We count how many search have been done with each filter found

Step 3 and 4

new Document("$sort", new Document("value", -1)),

new Document("$limit", 3)));

We sort the found filters in a decreasing order and take only the three most used filters