Documentation

Problem description

If you choose your data source to be a stream from the Meetup website, you need to implement the system which provides the following REST API:

Category A (precomputed reports):

1) Return the statistics with the number of newly created events per each country for the last 6 full hours, excluding the last hour.
The report should be in a format :
[{country1: number1}, {country2: number2},..].
Example of the response:
{“time_start”: “15:00”,
“time_end”: “21:00”,
“statistics”: [{“US”: 1543}, {“France” : 899}, {“Germany” : 923}, ...]}.

2) Return the statistics containing the information about which groups posted the events at each US state in the last 3 full hours, excluding the last hour. The names of the states should be full, not only the state code.
Example of the response:

```
        {“time_start”: “15:00”,
          “time_end”: “18:00”,
          “statistics”: [{“California” : [“Happy Programmers Group”,
                                                        “Emmy’s Bookclub”]},
                         {“Nevada”: [“Las Vegas Warriors”, “Desert Bikers”]}, …]}
```

3) The most popular topic of the events for each country posted in the last 6 hours, excluding the last hour. The popularity is calculated based on the number of occurrences topic has amongst all the topics in all the events created in that country during the specified period.
Example of the response:

```
        {“time_start”: “15:00”,
          “time_end”: “21:00”,
        “statistics”: [{“France” : {“Baking croissants”: 88}},
                       {“Germany”: {“Brewing beer”:71}, …]}
```

Category B (ad-hoc queries):

1) Return the list of all the countries for which the events were created.
2) Return the list of the cities for the specified country where at least one event was created.
3) Given the event id, return the following details:
 a) event name
 b) event time
 c) the list of the topics
 d) the group name
 e) the city and the country of the event
4) Return the list of the groups which have created events in the specified city. It should contain the following details:
 a) City name
 b) Group name
 c) Group id
5) Return all the events that were created by the specified group (group id will be the input parameter). Each event in the list should have the format as in the API #3.

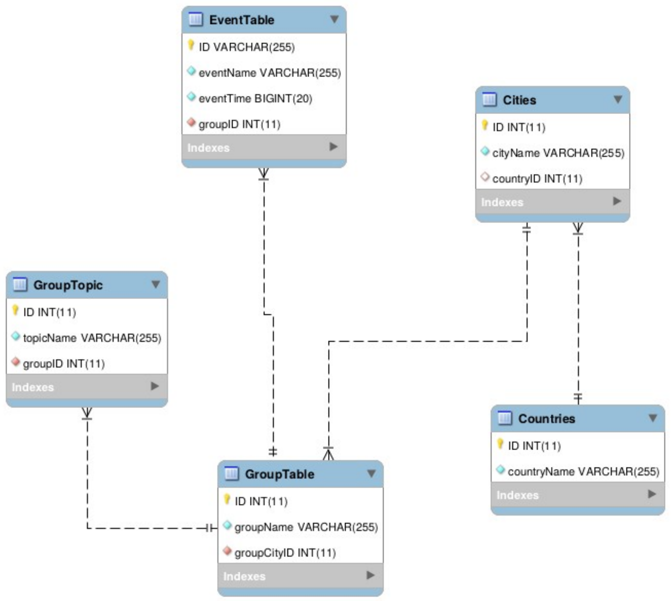
Database

Why SQL?

SQL was chosen because

1. we have many relations
2. we have many different requests and holding all that information together will raise a lot of different memory issues.
3. simple API which can be easily connected to any version of servers.
4. this database is the most familiar.

Our design



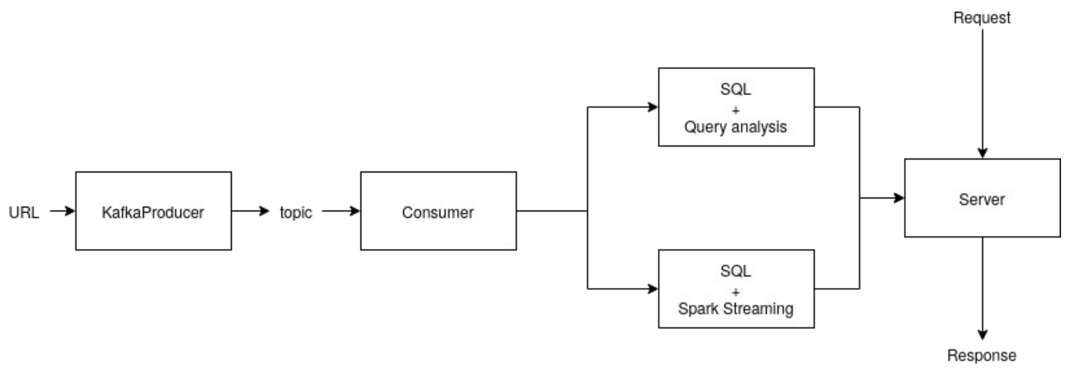
This picture shown our actual model of database. There are also three more tables but they have no relation to this tables and they would be descibed afterwards.

As you can see there are many different relation and this schema is also good because most of requests from part B even don't use full database but only few related tables.

Script for creation of this database can be found in *create_schema.sql*

System design

First let’s take a look on diagram which represents main part of the system and then we will describe each of them.



General describtion

We selected *lambda architecture* for this task, because in problem description you can find out that we actually need two different part: one for fast responses and it is using all available information and second one is analysing before writing to database (database we would descibe soon)

KafkaProducer + Consumer

We selected Kafka as modified version of queue for this task. Main advantages of using it here are:

1. It can continously read data from stream and by default can clean itself.
2. The queue which is generated with **Consumer** can be easily splited to Spark Stream and also to general SQL writter.
3. Also good plus was simple java integration.

How it works?

We have topic called storage where all information from url stream comes. After that we read that topic and rather analyse it using Spark Streaming or write straight to database.

Related classes

ConsumerMain.java
GeneratorMain.java

SQL + Query analysis

What does it means?
In this block we write straight to database and after that by using queries we select all data that we need.
This part contains in classes:
ConsumerMain.java
DatabaseWriter.java
And large part of directory spring. Main are:
CityRepository.java
CountryRepository.java
EventRepository.java
GroupRepository.java

Queries examples:

SELECT cities.ID,cities.cityName FROM cities left join countries on cities.countryID = countries.ID where countryName=?1

SELECT eventtable.ID, eventtable.eventName, eventtable.eventTime, grouptable.groupName, cities.cityName,countries.countryName,x.topics FROM (eventtable left join grouptable on eventtable.groupID = grouptable.ID left join cities on grouptable.groupCityID = cities.ID left join countries on cities.countryID = countries.ID left join (select grouptopic.groupID, group_concat(grouptopic.topicName) as topics from grouptopic group by grouptopic.groupID) x on grouptable.ID = x.groupID) where eventtable.ID = ?1

Why we choose this design?

This part must answer as fast as possible so there is no need to analyse something here (SQL can hold everything for you). This is one more reason why we selected this database.

SQL + Spark

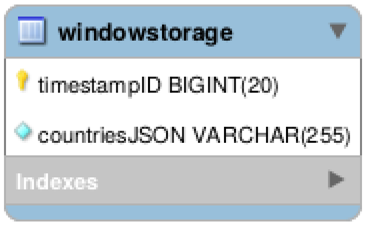
Why Spark?

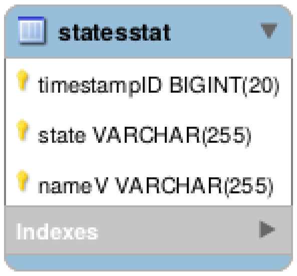
Main purpose of using Spark was availability of window function. All tasks from part A actually can be described in terms of window function. Also such function can perform different data manipulation which can make further analysis faster. And one more thing is that spark connect to database only a few times each 10 min so all other time can be used to write ahead so that won't make large queue.

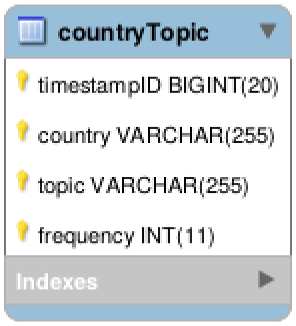
As it was said,Spark has 10 min interval for all window function. It was made for more accurate results and also we can't make it smaller because it will generate too big amount of data and it will take too long to write it down.

Databases for this case

Here were used three additional tables which were descibed earlier.







They have no relation but stored in SQL because each hour the request would change.

Server

For server we used *Spark Boot.* Main purposes of using it were:

1. Integration with many different modules by default.
2. Clear interface.
3. Easy to test and design.
4. Easy to read and understand

More about functions and other part is API documentation

API documentation

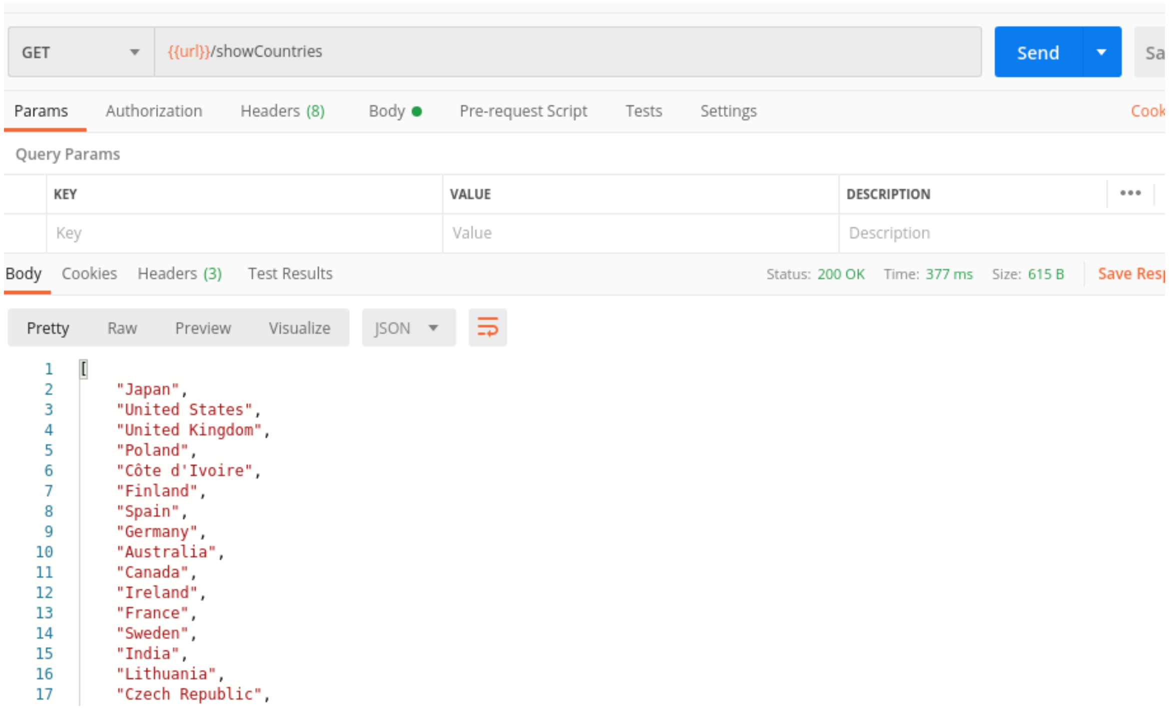
Our API has routes:

1.	/showCountries	Task 1 from part B
2.	/getCities	Task 2 from part B
3.	/getEventByEventID	Task 3 from part B
4.	/getGroup	Task 4 from part B
5.	/getEventByGroupID	Task 5 from part B
6.	/getFirstAnalysis	
7.	/getSecondAnalysis	
8.	/getThirdAnalysis	

/showCountries

Simple GET request.

Example of usage:



{{url}}/showCountries

Status: 200 OK Time: 377 ms Size: 615 B

```
[
    "Japan",
    "United States",
    "United Kingdom",
    "Poland",
    "Côte d'Ivoire",
    "Finland",
    "Spain",
    "Germany",
    "Australia",
    "Canada",
    "Ireland",
    "France",
    "Sweden",
    "India",
    "Lithuania",
    "Czech Republic",
```

full response in countries.json

/getCities

Post request

```
{
"countryName":"Japan"
}
```

countryName with full name of country

Response

```
[
"Tokyo",
"Osaka",
"Sapporo",
"Nagoya"
]
```

full response in file cities.json

/getEventByEventID

Post request with “eventID” and its string value.

```
{
"eventID":"241120167"
}
```

Response

```
{
"eventName": "Home Euchre party at Laura's",
"eventTime": "June 14, 2020 at 1:00 AM",
"groupName": "Fort Wayne Euchre Meetup",
"cityName": "Fort Wayne",
"countryName": "United States",
"topics": [
"Euchre Club",
"Euchre Players",
"Game Night",
"Board Games",
"Gaming",
"Games",
"Card Games",
"Euchre"
],
"id": 241120167
}
```

full response in eventID.json

/getGroup

Post method with requests looks like this

```
{
"cityName":"Tokyo"
}
```

cityName as key and full name as value

Response:

```
[
{
"groupName": "??? Japanese study group @ 80's Cafe",
"cityName": "Tokyo",
"id": 7040132
}...]
```

full response in file groups.json

/getEventByGroupID

Post request

```
{
"groupID":1789394
}
```

groupID and integer value of it.

Response

```
{
"eventName": "Panel: Getting Real with Data Analytics - Acting on Market Risks & Opportunities",
"eventTime": "July 7, 2020 at 7:00 PM",
"groupName": "Big Data LDN Meetup",
"cityName": "London",
"countryName": "United Kingdom",
"topics": [
"Data Management",
"Business Intelligence",
"Data Science",
"Software Development",
"Big Data",
"Cloud Computing",
"Data Analytics",
"Database Professionals",
"NoSQL",
"High Scalability Computing",
"Technology",
"Machine Learning",
"Hadoop",
"Predictive Analytics",
"Open Source"
],
"id": 271159892
}
```

It can be seen in groupID.json

/getFirstAnalysis
/getSecondAnalysis
/getThirdAnalysis

All of this are simple Get requests.
Output can be seen in firstAnalysyis.json, secondAnalysyis.json, thirdAnalysyis.json