## Taxi Trips Analytics Service

1. **Implementation Taxi Trips Analytics Service**

**Technology Used**

* [JDK 17](%5d(https:/www.azul.com/downloads/?version=java-17-lts&os=linux&package=jdk#zulu))
* [Maven 3](https://maven.apache.org/)
* [Spring Boot 3.3.2](https://github.com/spring-projects/spring-boot/wiki/Spring-Boot-3.3-Release-Notes)
* Mockito and Junit 5 for testing

1. **Project Structure**

Taxi Trips analytics service project is a **multi-module** maven project having a separate core module which contains all service implementations to support analytics operations on trips data. In current project this separate core module is used with rest module. In future, core module can be **imported as library** to be used with other presentation layers like **command line**.

Detail of project structure are as below:

‘**taxi-trips-analytics-service’**

* **‘time-series-db’ module**

This module contains custom implementation of in-memory time series db. Data structure used to implement this is ConcurrentHashMap and [ConcurrentSkipListMap](https://www.baeldung.com/java-concurrent-skip-list-map). This module provides functions like query based on time range, filter operation on queried data, Aggregate operation(Count, Sum, Avg), and group by operation.

**Details on Data Structure used:**

* + **ConcurrentHashMap** is the concurrent version of original hashmap providing thread-safe apis which needs no lock for read access and write locks are segmented. So at a time, ‘time-series-db’ supports concurrent write operation equals to the number of segments.
  + **ConcurrentSkipListMap**: ConcurrentSkipListMap is thread-safe version of TreeMap providing concurrent write access and no locks are taken for read access.

**How is the trips data saved:**

Trips data is saved at year-month-day-hour-timestamp level. ConcurrentHashMap is used to store the data for Year-Month-Day-Hour and thereafter the data is stored at timestamp level (829837913, 913829837 in below example). We are not storing the entire timestamp as key instead we are storing only the difference in timestamps. Timestamp difference is calculated from given date with the start of the hour.

Reason to use ConcurrentHashMap for Year-Month-Day-Hour is bcoz these metrics are all of small ranges E.g. month (1-12), day (1-31), Hour (1-24). We can apply containsKey check on this small range with O(1) time complexity.

JSON example of how the data is stored in DB

{

"2019": {//year, ConcurrentHashMap

"10": { //month, ConcurrentHashMap

"21": { //day, ConcurrentHashMap

"14": { //hour, ConcurrentHashMap

"829837913":{ //timstamp, ConcurrentSkipListMap

[

**TripInfo**(taxiId=1, tripStartTime: 2019-10-21 T14:44:12,

**TripInfo**(…),.,.,

]

}

}

}

},

"2020": { //year

"11": { //month

"22": { //day

"12": { //hour

"913829837": [**TripInfo**(taxiId=2, tripStartTime: 2020-11-22 T12:23:32, **TripInfo**(…),.,.,]

}

}

},

"12": { //month

"18": { //day

"04": { //hour

"382919837": [**TripInfo**(taxiId=3, tripStartTime: 2019-12-18 T04:11:43, **TripInfo**(…),.,.,]

}

},

"23": { //day

"13": { //hour

"298913837": [**TripInfo**(taxiId=4, tripStartTime: 2019-12-23 T13:54:59, **TripInfo**(…),.,.,]

}

}

}

}

}

* ‘**taxi-trips-analytics-domain’ module**

This module contains all the domain classes: TripInfo and CoOrdinate class.

* ‘**taxi-trips-analytics-core’** **module**

This module contains all the service implementations to provide analytics service on Trips data. The core module uses apis provided by above `**time-series-db**` to perform analytics operation on trips data. In future, we can add more apis to provide more analytics services.

* **‘taxi-trips-analytics-rest’ module**

This module contains the rest layer which internally uses ‘**taxi-trips-analytics-core’ module**. This module will contain all business validation to be done.

1. **Extensibility**

* Project structure follows a **multi-module** maven project. It has separate core module which contains all service implementations to support analytics. This core module can be later imported as library and used with other modules like Command Iine, graphQL to support taxi fleet service.
* I have implemented my own **‘time-series-db’** providing thread safe apis (for aggregate, groupBy, time-based query at year, month, day, hour level, additional filter operations) using java concurrent data structures like ConcurrentHashMap and ConcurrentSkipListMap. This module can be enhanced by adding more apis to it and imported as separate module for future use. It’s a completely pure java implementation; no external frameworks like spring have been used in this module.
* With ‘**time-series-db’**,I have made sure concise storage of data, No Redundancy. Even when storing data at timestamp, we are not storing the entire timestamp as key, only the difference in timestamp w.r.t to the start of the hour is stored.
* Docker added to support containerization. Swagger available at http://localhost:8080/swagger-ui/index.html.

1. **Assumption**

* I Have found some corrupted data in provided data sets. So, I have added below validations as per my own judgement. Below are they
  + TripEndTime should be greater than TripStartTime.
  + Trip\_seconds should be greater than zero.
  + Longitude should be between -180 and 180 degree and Latitude should be between -90 and 90 degree.
  + Along with above range validation, to calculated S2cellId for average\_fare\_heatmap, Longitude and Latitude value should not be zero.
* I have only added instruction coverage to be printed at the end of bin/setup as maven does not have trusted plugin to support printing of coverage details to console. Detailed code coverage can be found at [path](http://localhost:63342/taxi-trips-analytics-service/coverage-module/target/site/jacoco-aggregate/index.html?_ijt=nuf46ao2ak8md0nb4861qpqtbt&_ij_reload=RELOAD_ON_SAVE).

1. **What more can be done, if time permits**

* We can make use of parallel stream when doing reduce operation like groupBy and aggregate.
* Functions like pushData and reduce operation can be executed in multiple threads and later return the aggregated result thereby following Divide and Conquer method.

1. **How to run the program**

I have provided the README.md file at the root of the project which has steps to create the jar file and to test, run the application.

Once application is run, we can goto swagger UI: <http://localhost:8000/swagger-ui/index.html> and start testing.

1. **Test Quality (Coverage)**

Line: 81.83% (1879/2296)