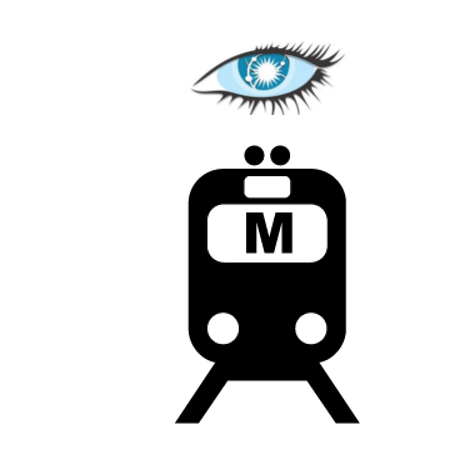
**The design and the implementation of the distributed data management layer for the metro mobile application**

****

**Project by : Jukanna Nityanand (4557016)**

**Mani Kanth seelam (4555036)**

Project Summary

Conceptual Schema

Workload Schema

* Administrator workload:
* Driver workload
* Allocation workload
* Time Table workload

Logical Schema

CQL Implementation

1. Project Summary

The reference domain concerns a Metro public transportation service. In this context, we are interested in a mobile application that works as a data recorder for vehicles. The application uses mobile data connections to send information to servers in real time. Data are stored and are not managed as a stream. The information includes measurements such as the location and speed of the vehicles. Typical workloads mix together write and read operations. The aim of this project is the design and the implementation of the distributed data management layer for the mobile application. To this aim, we choose Cassandra as reference technology and we rely on CQL for workload implementation. Our mobile application needs workload both read and write operations. Cassandra provides very good performance for both kinds of operations.

The mobile application needs a database containing data about drivers, railway vehicles, time table, availability of drivers and vehicles, and current position and speed of vehicles (Data Point).

The deployment of the application and the test database should involve a cluster of machines. The infrastructure team has agreed to the following Availability requirements:

A Strong Availability for 100% of the data must be provided when one node is down.

* The product team has agreed to the following consistency requirements:

• Reading driver and vehicle data must be strongly consistent;

• Reading Data Point and other data may be eventually consistent

We will start by designing the conceptual schema of this problem, then we will identify its workload, and based on the results of this two steps and Cassandra specification we will design the logical schema using query-driven design. After designing the database layer, we are going to generate a database based on a real information of **Melbourne metro**, then implement the required workload using CQL

**Conceptual Schema**

To identify the problem and its specifications, we started by creating an ERD schema using the following entities:

|  |  |
| --- | --- |
| Field Name | Definition |
| Driver\_Name | Unique identifier for each driver (**Primary Key**) |
| Password | Driver’s password |
| Current\_Position | Current position of driver: in station (name of station), in train (vehicle ID), outside (not available) |
| Skill | Driver’s skills according type of vehicle |
| Mobile | Driver’s mobile phone |

**Driver Entity:**

**Railway\_Vehicle Entity​:**

|  |  |
| --- | --- |
| Field name | Definition |
| Vehicle\_id | Unique identifier for each vehicle (**Primary Key**) |
| Status | Status of vehicle which may be: station\_name (available), in\_use (travel according services), maintenance, out\_of\_order |
| Type | Type of vehicle |

**Station Entity:**

|  |  |
| --- | --- |
| Field name | Definition |
| Station\_Name | Unique identifier for each station (**Primary Key**) |
| Position | Station geographic coordinates (User Defined Type of Latitude & Longitude) |

**Data\_Point Entity:**

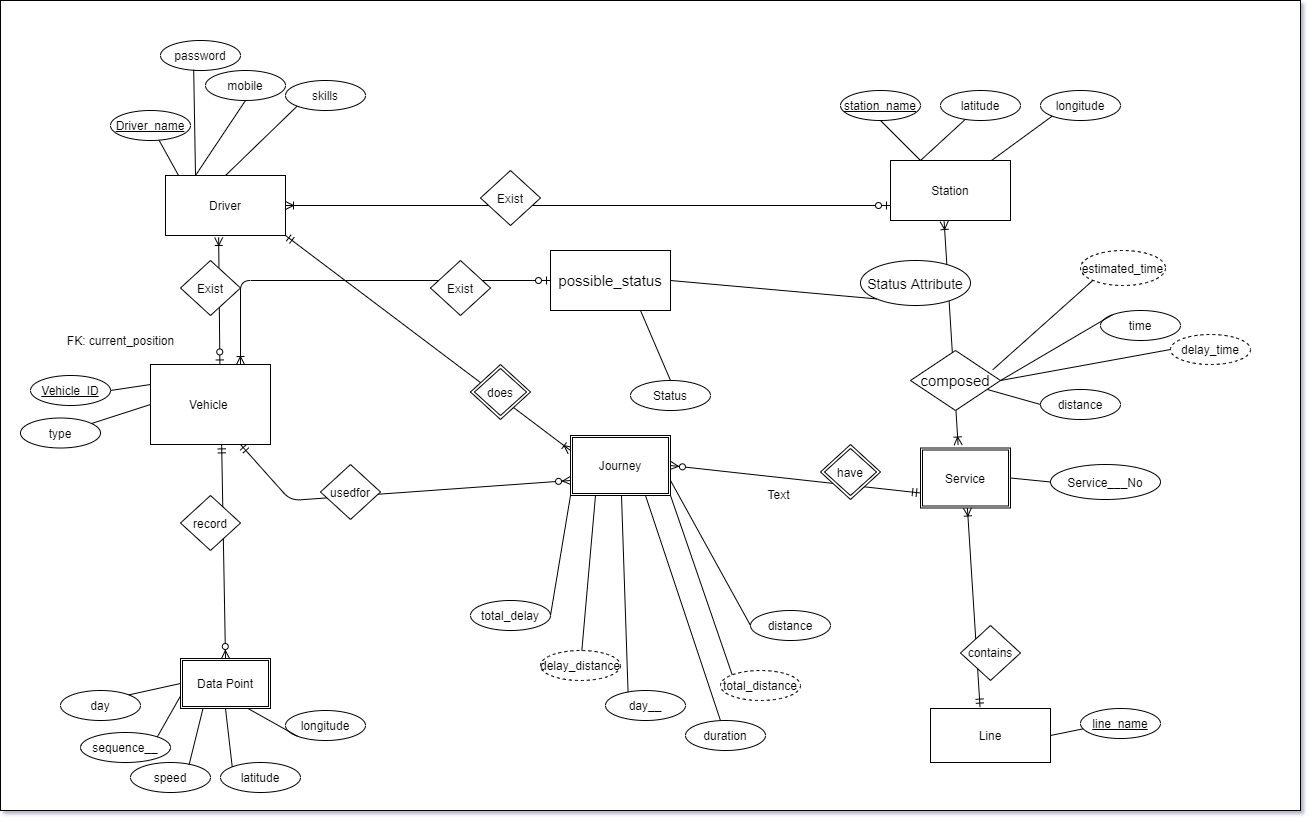
|  |  |
| --- | --- |
| Field name | Definition |
| Line | Name of the line (**Primary Key**) |
| Service\_no | Service number (**Primary Key**) |
| latitude | Geographic coordinates in current time of operational vehicle: Latitude |
| longitude | Geographic coordinates in current time of operational vehicle: Longitude |
| Speed | Speed of train in a specific temporal point |
| Date | Current date |
| time | Current time |

**Service Entity:**

|  |  |
| --- | --- |
| Field name | Definition |
| Line | The name of line (**Primary Key**) |
| No\_service | Identifier of a service (**Primary Key**) |
| Station\_name | Can be departure/arrival/middle station |
| time | Arrival time for arrival station, and departure for other |
| Estimated\_Time | Predicted time for time. |
| Distance | distance between this station and the departure station of the line |

**Journey Entity:**

|  |  |
| --- | --- |
| Field name | Definition |
| line | Name of the line that operates (**Primary Key**) |
| Service\_no | Identifier of the Service of this travel (**Primary Key**) |
| Driver\_name | Name of the driver who is driving train (**Primary Key**) |
| Vehicle\_id | Identifier of Vehicle of this travel (**Primary Key**) |
| date | Present date |
| delay | Change between the scheduled time actual time |
| distance | Distance travelled |

****

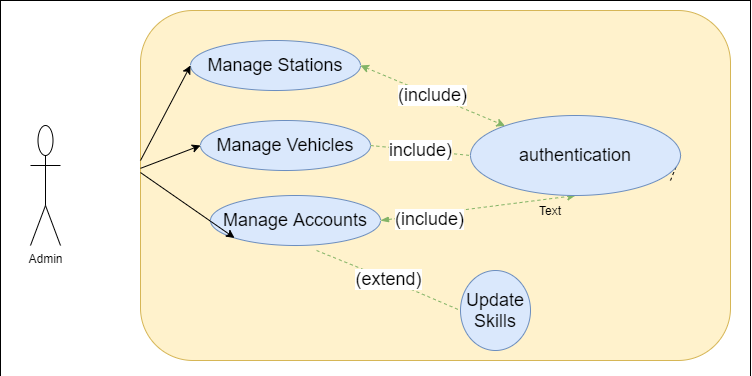
conceptual schema

Workload Schema

workload and split its actions that can be executed in application layer from those that should be executed in database layer. we are going to discuss in details the following parts:

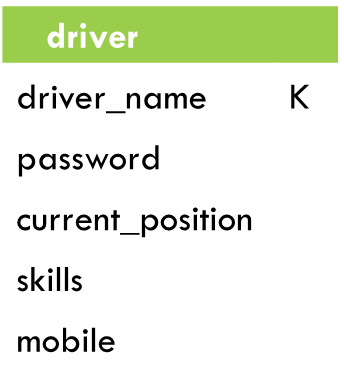
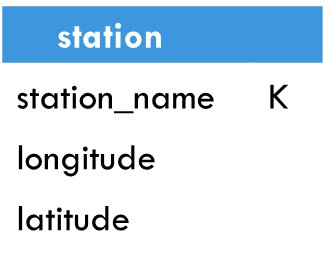
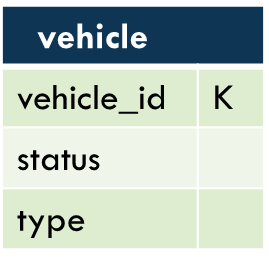
* Administrator workload
* Driver workload
* Allocation of drivers and vehicle workload
* Update time table workload

**Administrator workload use case diagram**



In the table below we split the required queries for this use case into class, each class present the queries of a table, in the objective of designing the logical schema of this use case:

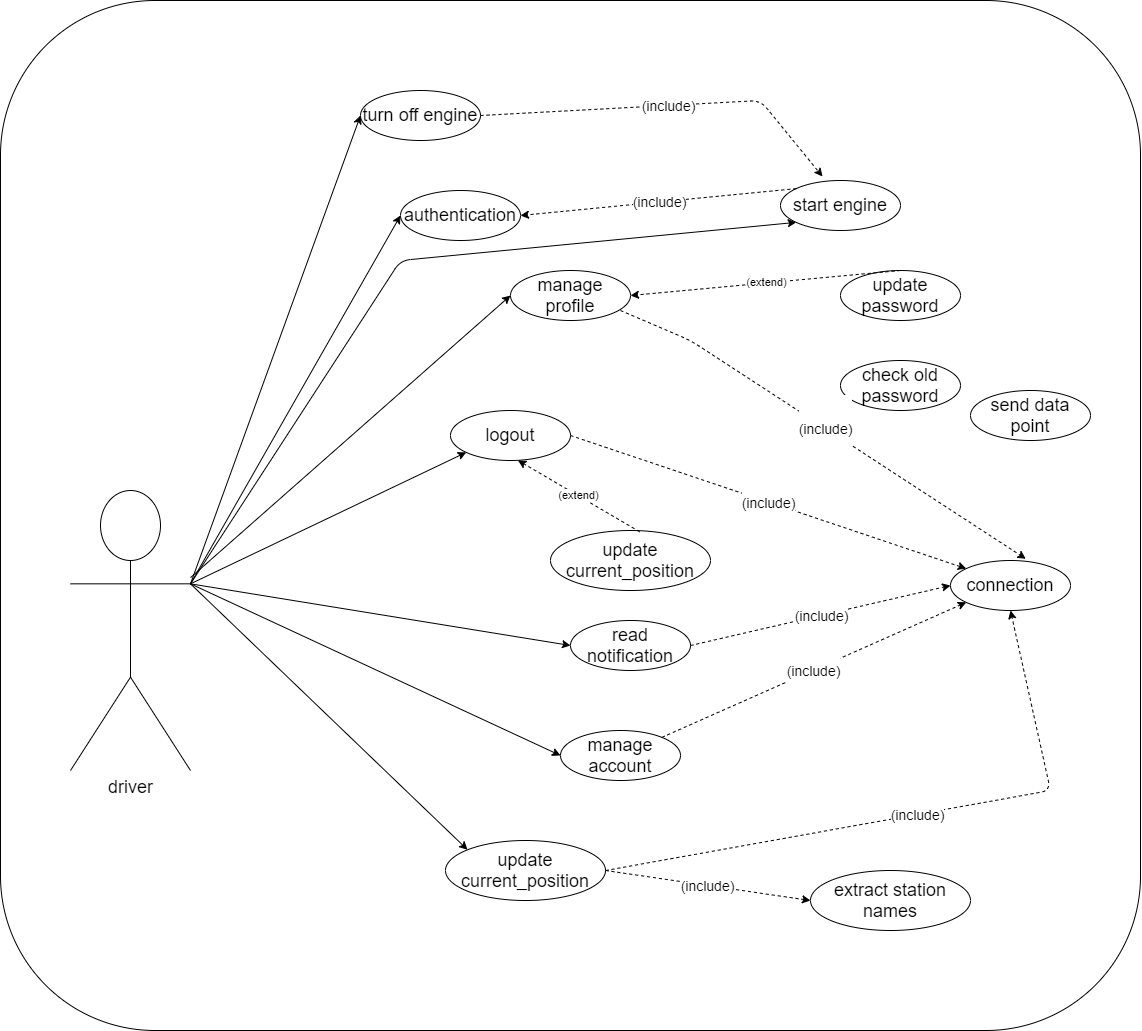
|  |  |
| --- | --- |
| **Class** | **Query** |
| Queries A for Station | Insert/Delete/Update a Station |
| Queries B for Vehicle | Insert/Delete/Update a Vehicle |
| Queries C for Driver | Insert/Delete/Update a driver and his skills |

Logical schema first version

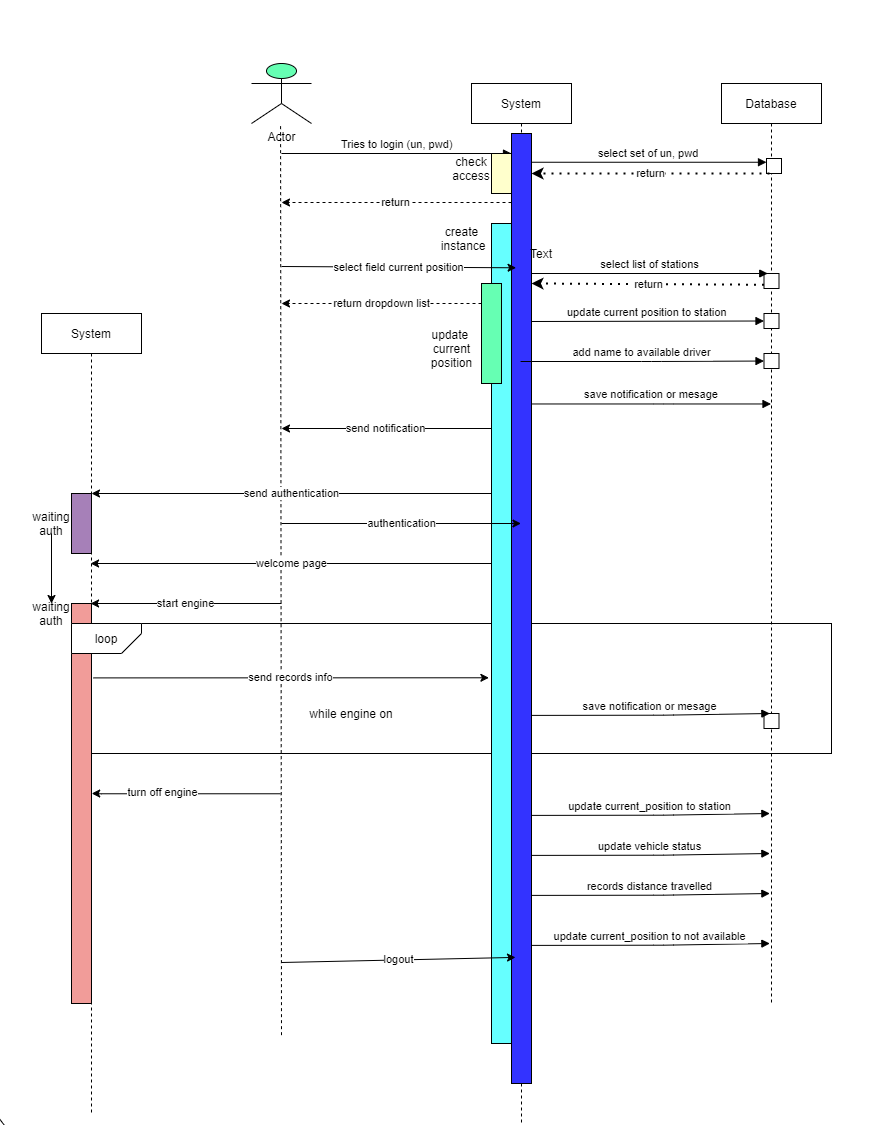
**Driver workload**

Another workload for drivers and mobile device(vehicle), is explained in the use case bellow:



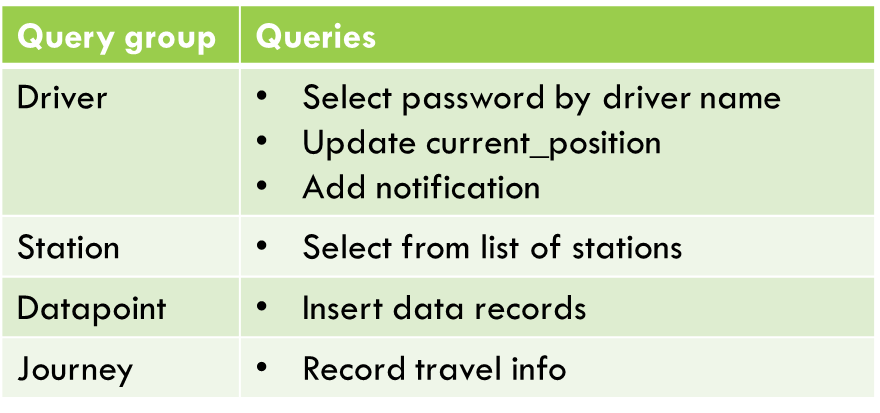
driver use case

This use case follows a suitable order to be executed, this order is presented by a sequence diagram, that will help us to identify the needed queries for our database layer.

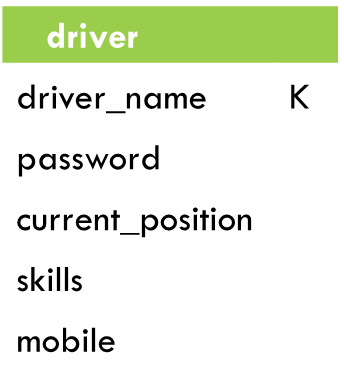
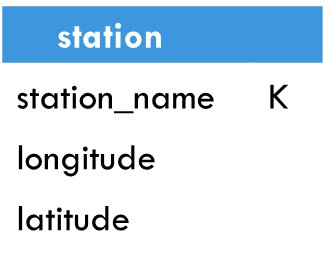
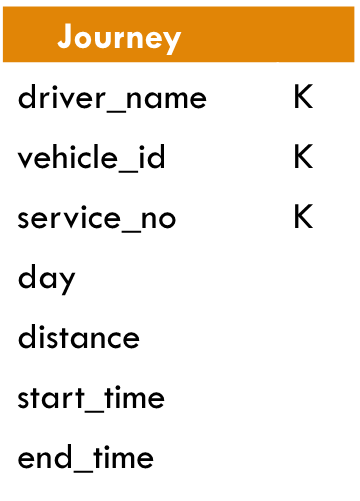
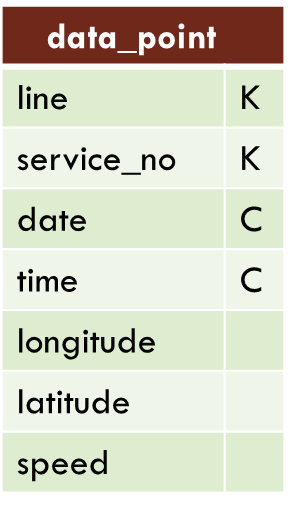


sequence diagram for driver actions

All changes between database component and system, are described in function of queries in the table below, and grouped into class. each class present one table of the second version of logical schema:

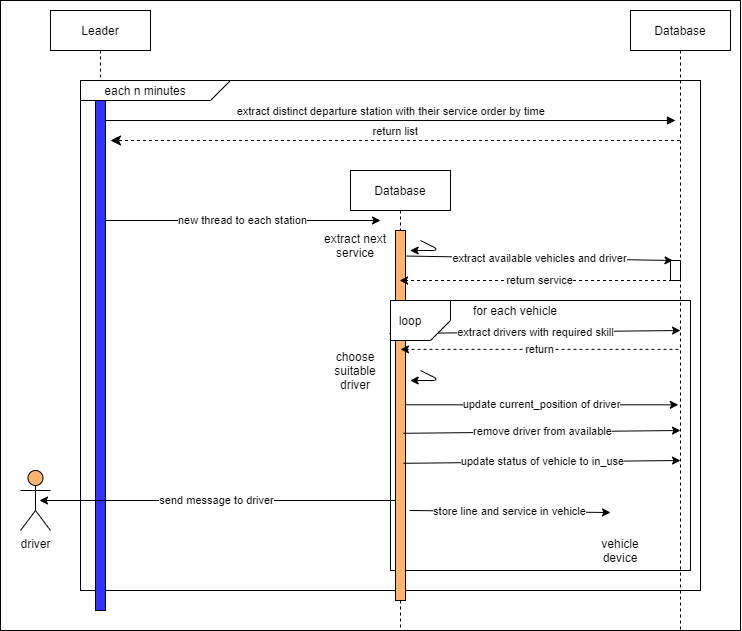


The second version of our logical schema for this use cases

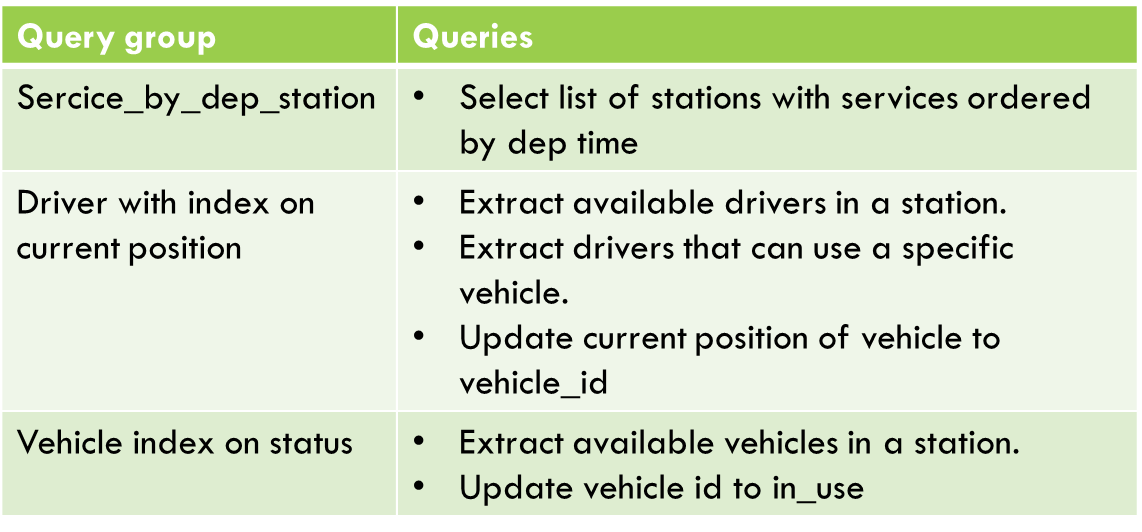
**Allocation workload**

The allocation of devices and drivers is a periodic process, executed by the system in a distributed way. the diagram below describes the sequence of this operation’s task, and split the database layer from the application layer:

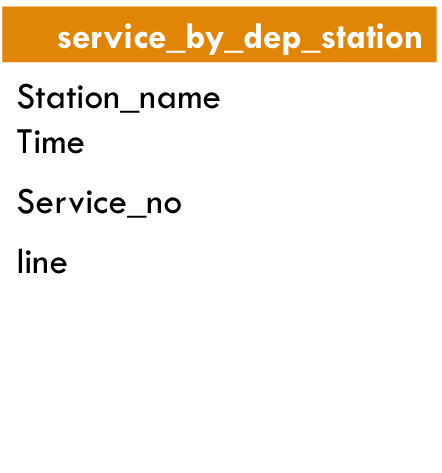
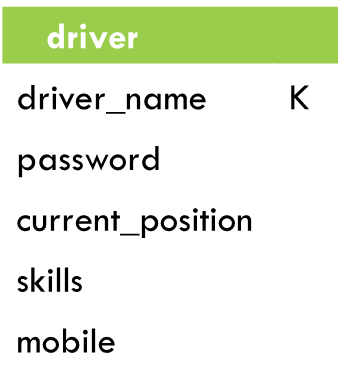
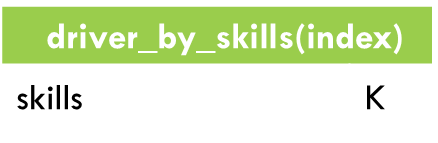


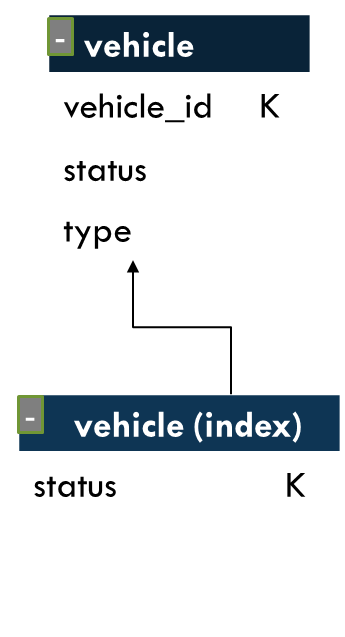
sequence diagram of allocation

Based on the changes between database layer and system layer, we described here



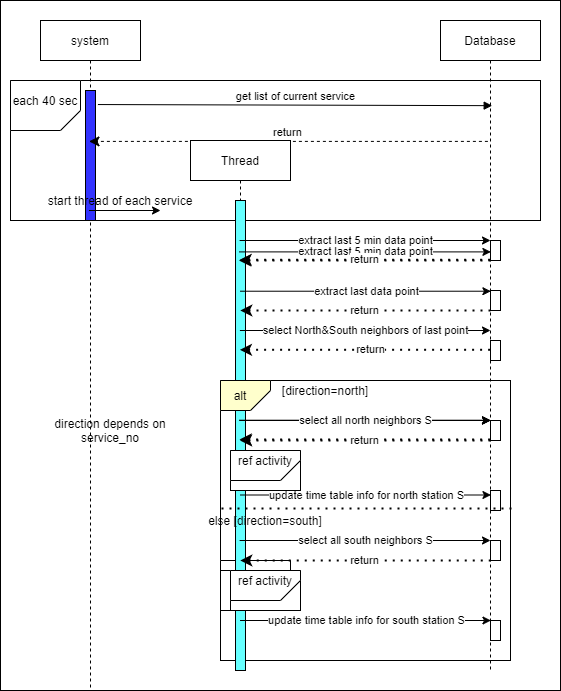
Third version of logical schema



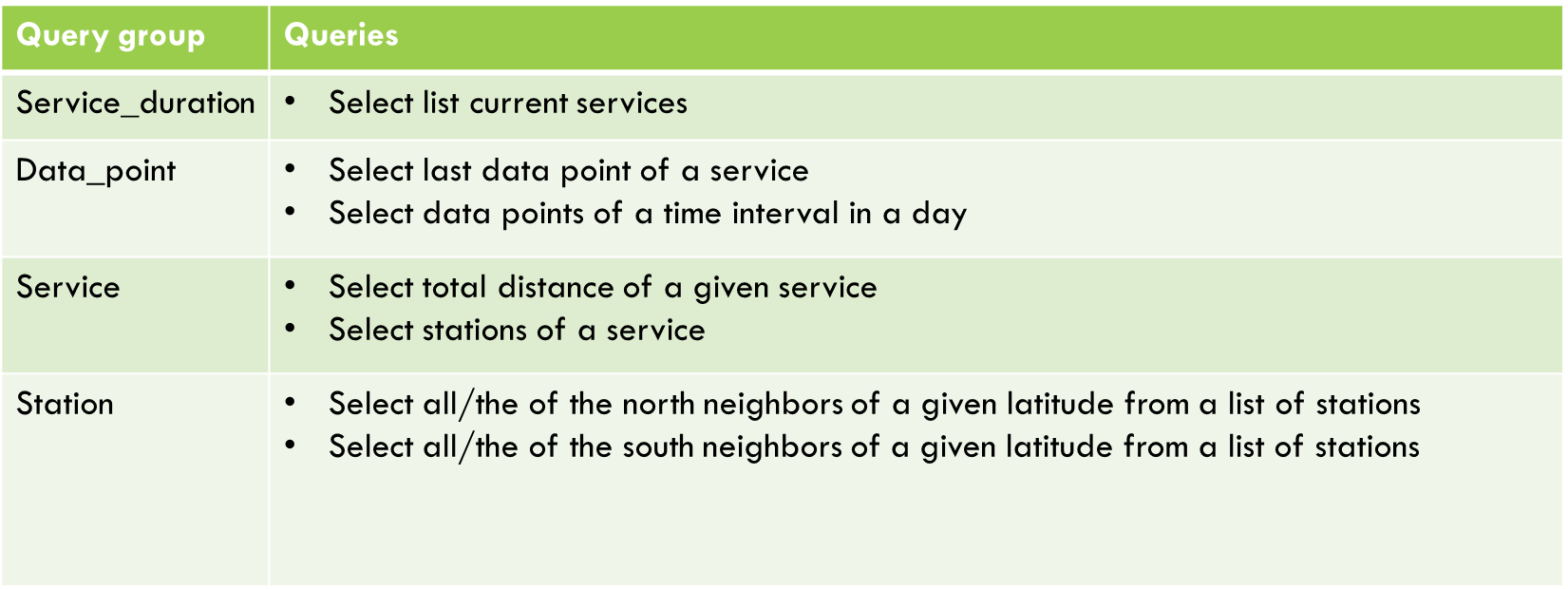
**Time Table workload**

Each unit of time (30 second), the time tables in stations should be uploaded to display the departure time of services in real time. The diagram below presents the sequence of instruction in this operation, and separate the application workload from database workload:

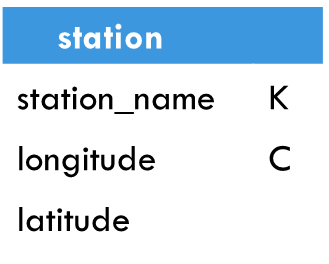
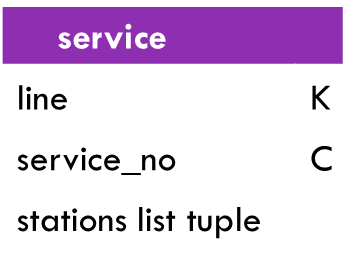
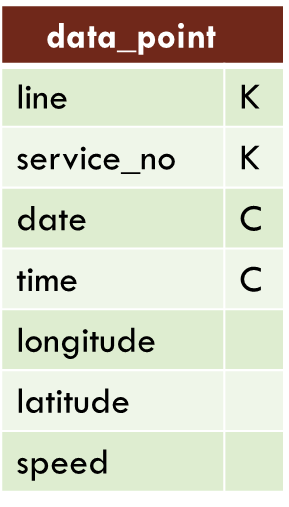
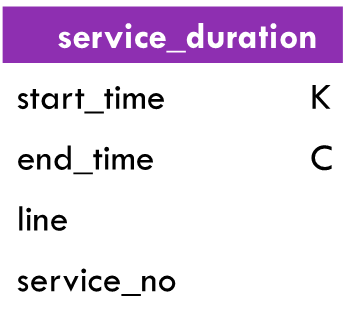


sequence diagram of time table update

Based on instructions sent to database component, we added the new queries to our database layer



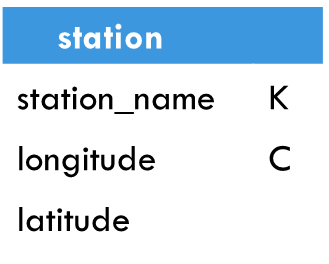
last version of logical schema

Logical Schema

Based on the conceptual schema and the required workloads, the global logical schema schema for database layer after adding other queries for analysing the data base

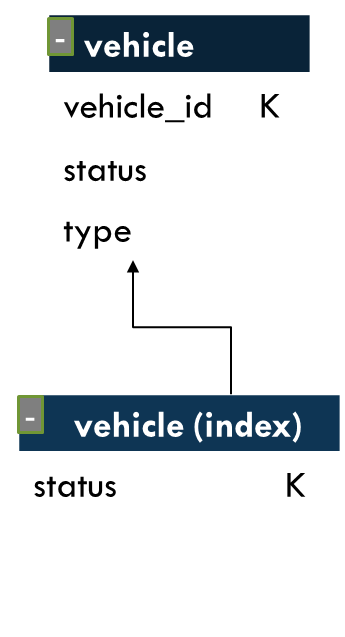
Queries A (1,2,3)



Queries A:

1. Insert station by admin
2. Remove a station by admin
3. Select list of Station

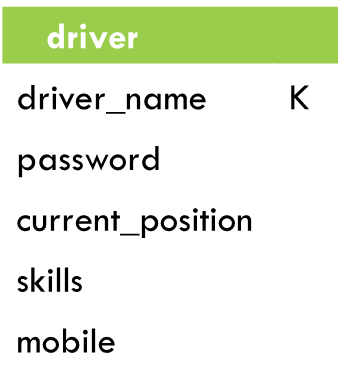
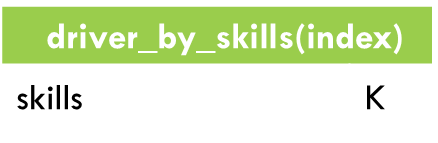
Queries B (1,2,3,4,5)



Queries B (6)

Queries B:

1. Insert vehicle by admin
2. remove vehicle by admin
3. Update vehicle by admin
4. Update status
5. Select vehicle type
6. Select available vehicle in given station

****

Queries C (1,2,3,4,5)

Queries C (6)

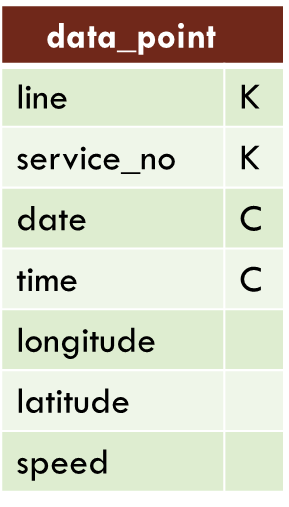
Queries C:

1. Insert driver by admin
2. Remove driver by admin
3. Update driver skills by admin
4. Select password of driver by his name
5. Update current position to selected station
6. Select available driver in a given station(index)
7. Extract the drivers that can use a specific vehicle (index)

Queries D:

1. Insert data records
2. Select data points for a given Line\_service in time interval of day
3. Select last data points for a given Line\_service and day

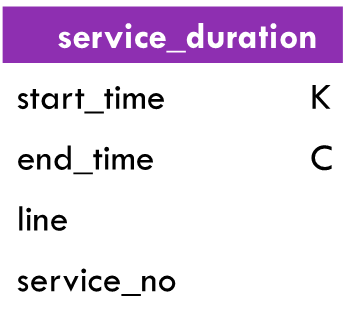
Queries D (1,2,3)

****

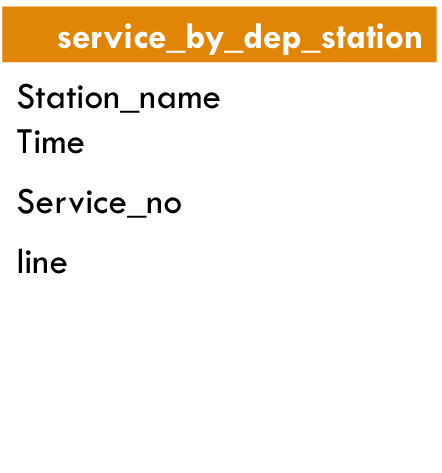
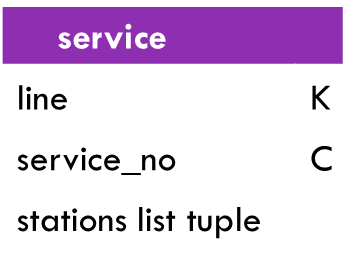
Queries E:

1. Insert new service
2. Select distance of a service
3. Select list of all stations of a service
4. Select current service with current time bigger than departure\_time (distance=0) and small than arrivel/departure\_time(distance !=0) => Service\_duration(Index)
5. Select list of distinct station with their services ordered by time departure.

Queries E (1,4)

****

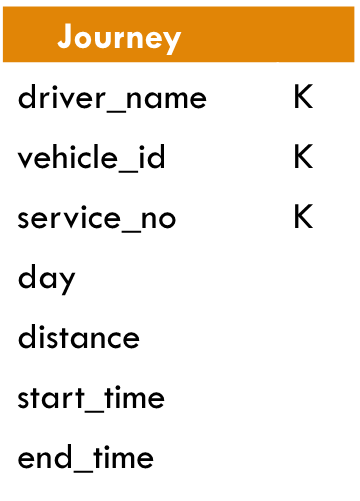
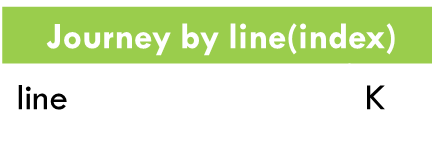
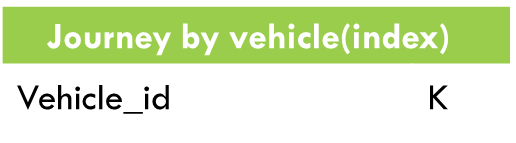
Queries E (1,5)

**** ****

Queries E (1,2,3)

Queries F:

1. 1. Insert new records after selecting (QueryF.2,QueryD.5)
2. Select total number travels What does it mean total travels
3. --
4. --
5. Select total work time of a driver on a day interval index (travel\_by\_driver)
6. --
7. Select total distance for a given vehicle on a day interval index (travel\_by\_vehicle)
8. --
9. Select total travel of a service (travel\_by\_service) 10. Select total travel of a line (travel\_by\_line)
10. Select total travel of a line (travel\_by\_line)

****

Queries F (9)

Queries F (7)

Queries F (1,2)

Queries F (10)

Queries F (5)

**CQL Implementation**

To respect system requirement, all write or read operation on table driver or vehicle should be with a strong consistency and availability using quorum technic.

​>>CONSISTENCY QUORUM;

For operations in other table it should ​provide the highest availability of all the levels with a weak consistency using two closest replica.

>>CONSISTENCY TWO;

The implementation of this queries with CQL is :

|  |  |
| --- | --- |
| Queries A on Station Table | |
| To create schema | CREATE TABLE ks\_user39.station(station\_name text PRIMARY KEY, latitude float, longitude float); |
| To answer queries | 1. INSERT INTO Station (station\_name, position) VALUES (stationName, stationLatitude,stationLongitude); 2. DELETE FROM Station WHERE station\_name = 'station\_name'; 3. SELECT station\_name FROM Station |

|  |  |
| --- | --- |
| Queries B on Vehicle | |
| To create schema | CREATE TABLE ks\_user39.vehicle (vehicle\_id text PRIMARY KEY,status text, type text);  CREATE INDEX vehicle\_status\_idx ON ks\_user39.vehicle (status); |
| To answer queries | >>CONSISTENCY QUORUM   1. INSERT INTO Vehicle (vehicle\_id, status, type) VALUES ('vehicle\_id', 'status', 'type'); 2. DELETE FROM Vehicle WHERE vehicle\_id='vehicle\_id'; 3. UPDATE Vehicle SET status = 'status' WHERE vehicle\_id = 'vehicle\_id'; 4. UPDATE Vehicle SET status = 'status' WHERE vehicle\_id = 'vehicle\_id'; 5. SELECT type FROM Vehicle WHERE vehicle\_id='vehicle\_id'; 6. SELECT vehicle\_id FROM Vehicle WHERE status = 'station\_name'; |

|  |  |
| --- | --- |
| Queries C on Driver Table | |
| To create schema | CREATE TABLE ks\_user39.driver (driver\_name text PRIMARY KEY, current\_position text, password text, phone text, skills set<text>);  CREATE INDEX driver\_current\_position ON ks\_user39.driver (current\_position);  CREATE INDEX driver\_by\_skill ON ks\_user39.driver (values(skills)); |
| To answer queries | >>CONSISTENCY QUORUM   1. INSERT INTO Driver (driver\_name, current\_position, password, phone, skills) VALUES   ('driverName','password','current\_position',['skill1','skill2'],'mobile');   1. DELETE FROM Driver WHERE driver\_name = 'driverName'; 2. UPDATE Driver SET skill = driverSkill WHERE driver\_name = 'driverName'; 3. SELECT password FROM Driver WHERE driver\_name = 'driverName'; 4. UPDATE Driver SET current\_position='current\_position' WHERE driver\_name= 'driverName'; 5. SELECT driver\_name FROM Driver WHERE current\_position = 'current\_position'; 6. SELECT driver\_name FROM Driver WHERE skills = 'skill'; |

|  |  |
| --- | --- |
| Queries D on Data\_point Table | |
| To create schema | CREATE TABLE ks\_user39.data\_point (line text, service\_no smallint, date int, time int, latitude float, longitude float, speed int, PRIMARY KEY ((line, service\_no), date, time))  WITH CLUSTERING ORDER BY (date ASC, time ASC) |
| To answer queries | 1. INSERT INTO Data\_point (line, service\_no, date, time, vehicle\_id, position, speed) VALUES ('line', 10, 20190210, 0405, 'vehicle\_id',48.75485, -73.97544, 150); 2. SELECT position,speed from Data\_point where line='line' and service\_no=34 and date=current\_date and time>=current\_time LIMIT 1; 3. SELECT position,speed from Data\_point where line='Brin' and service\_no=34 and date=20180101 and time<=1400 and time>=1330; |

|  |  |
| --- | --- |
| Queries E on Services Table | |
| To create schema | CREATE TABLE ks\_user39.Service\_duration (start\_time int,endtime int,line text,service\_no smallint, PRIMARY KEY (start\_time, endtime)) WITH CLUSTERING ORDER BY (endtime ASC);  CREATE TABLE ks\_user39.Service (line text, service\_no smallint, stations list<frozen<tuple<int, text, float>>>, PRIMARY KEY (line, service\_no)) WITH CLUSTERING ORDER BY (service\_no ASC);  CREATE TABLE ks\_user39.Service\_by\_dep\_station (station\_name text PRIMARY KEY, services list<frozen<tuple<int, int, text>>>);  CREATE OR REPLACE FUNCTION lastOfList (input list<text>) CALLED ON NULL INPUT RETURNS float LANGUAGE java AS 'return Integer.valueOf(input.get(input.size()-1)); ‘;  CREATE OR REPLACE FUNCTION tuple\_distance(input tuple<int,text,float>) CALLED ON NULL INPUT RETURNS float LANGUAGE java AS 'return Integer.valueOf(input.get(input.size()-1)); '; |
| To answer queries | 1. Insert into Service (line,Service\_no,stations) VALUES ('AAAA',1,[{0405,'Zeeland',0},{0406,’Brin’,12});   Insert into Service\_by\_dep\_station (station\_name,time,line,Service\_no) VALUES('Zeeland',0405,'AAAA',1);  Insert into Service\_duration (starttime ,endtime ,line,Service\_no) VALUES(0200,0405,'AAAA',1);   1. select tuple\_distance(lastOfList(distance)) from Service where line='AAAA' and service\_no=1; 2. select line,service\_no from Service\_duration where TOKEN(startTime)<TOKEN(1203) AND endTime>1203 ALLOW FILTERING 3. select station\_name,services from Service\_by\_dep\_station; |

|  |  |
| --- | --- |
| Queries F on Journey Table | |
| To create schema | CREATE TABLE ks\_user39.Journey (line text, service\_no smallint, vehicle\_id int, driver\_name text, date int, delay int, distance float, PRIMARY KEY ((line, service\_no, vehicle\_id, driver\_name), date)) WITH CLUSTERING ORDER BY (date ASC); |
| To answer queries | 1. 1.INSERT INTO Journey (line,service\_no,vehicle\_id,driver\_name,date,delay,distance) Values('Brignole\_Brin',3,'98723','NARGES',20180201,20,35,); 2. SELECT count (\*) from Journey ALLOW FILTERING; 3. SELECT sum(duration) from Journey where driver\_name='NARGES' AND day>20182801   7. SELECT sum(distance) from Journey where vehicle\_id='24352' AND day>20182801 ALLOW FILTERING   1. SELECT count (\*) from Journey where line='Brignole\_Brin' ALLOW FILTERING; 2. SELECT count (\*) from Journey where service\_no=3 ALLOW FILTERING; |