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)

I. 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:

Driver Entity:

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

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:

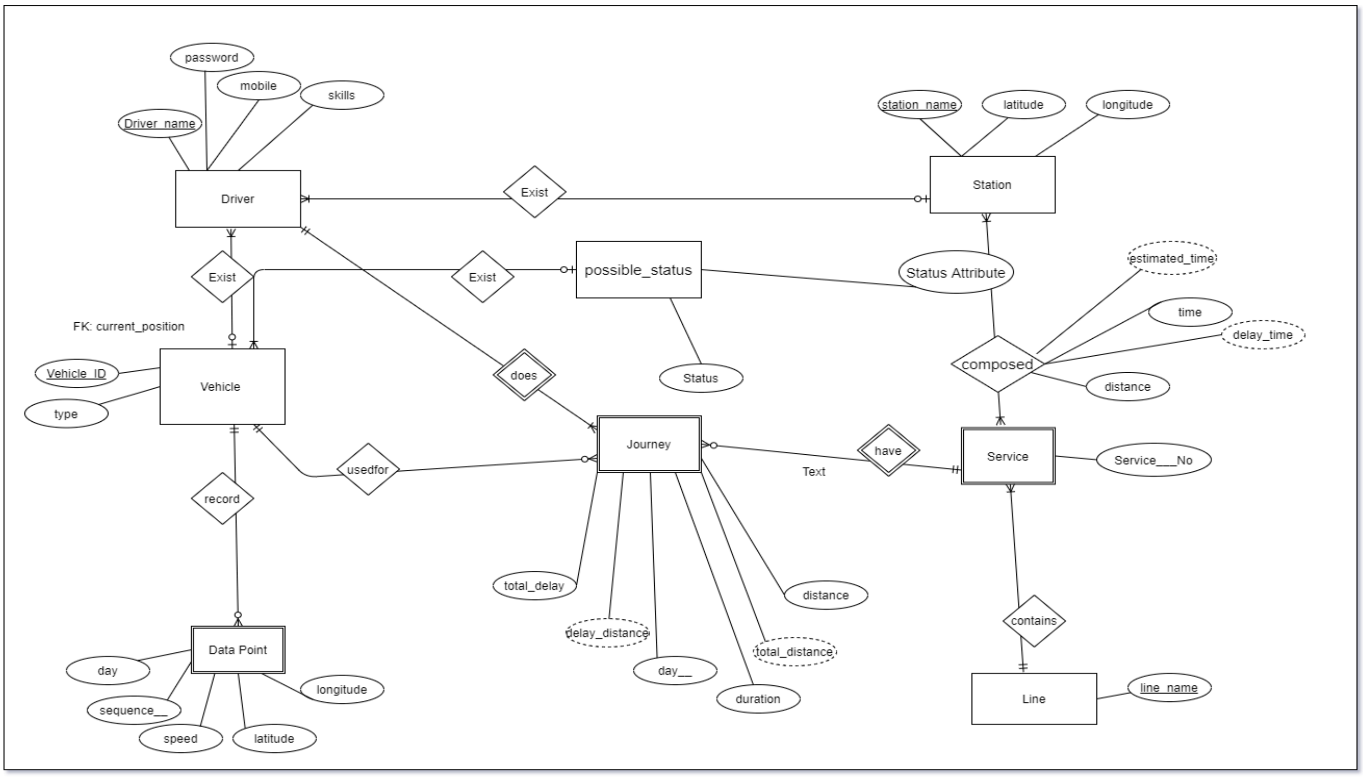
Field name	Definition
Line	Name of the line (**Primary Key**)
Service_no	Service number (**Primary Key**)
latitude	Geographic coordinates in current time of operational vehicle: Latitude
longitude	Geographic coordinates in current time of operational vehicle: Longitude
Speed	Speed of train in a specific 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



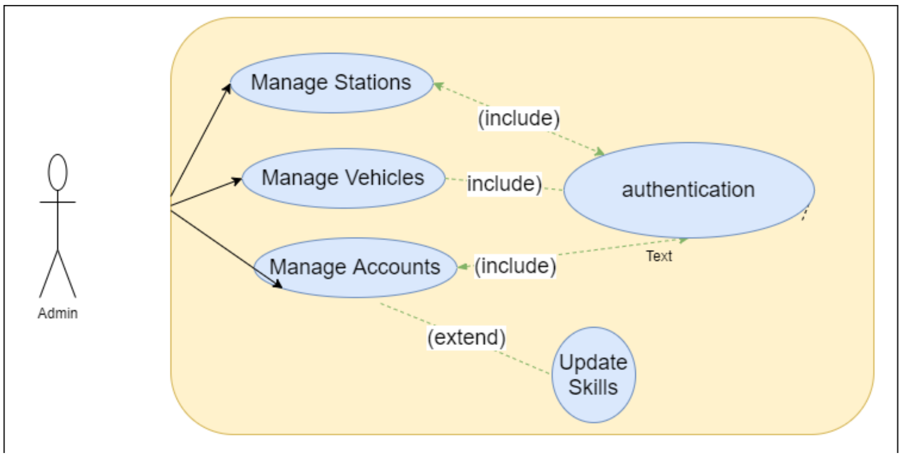
conceptual schema

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

driver	
driver_name	K
password	
current_position	
skills	
mobile	

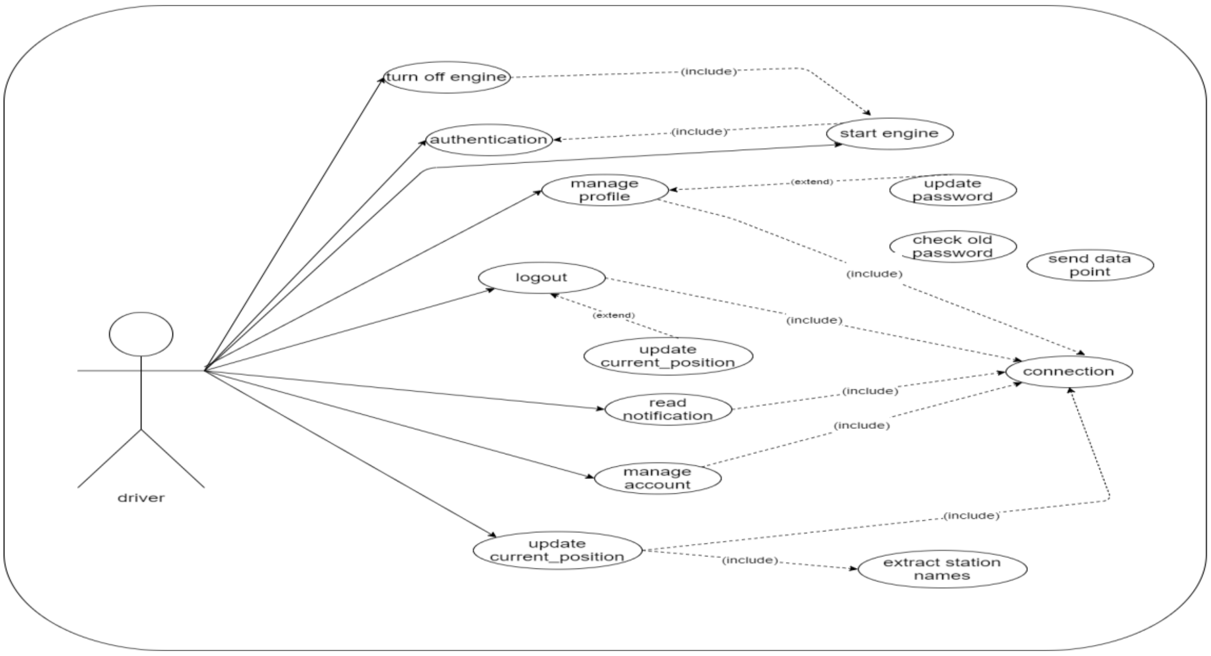
station	
station_name	K
longitude	
latitude	

vehicle	
vehicle_id	K
status	
type	

Logical schema first version

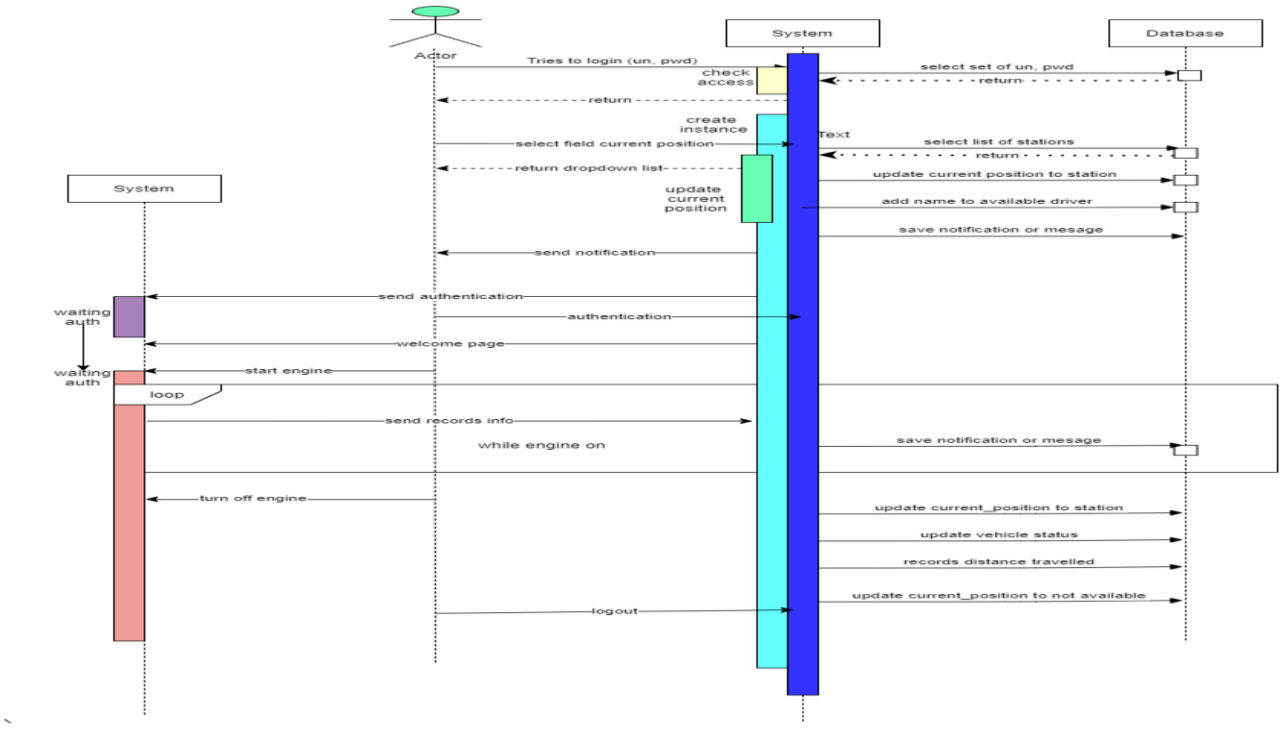
Driver workload

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.

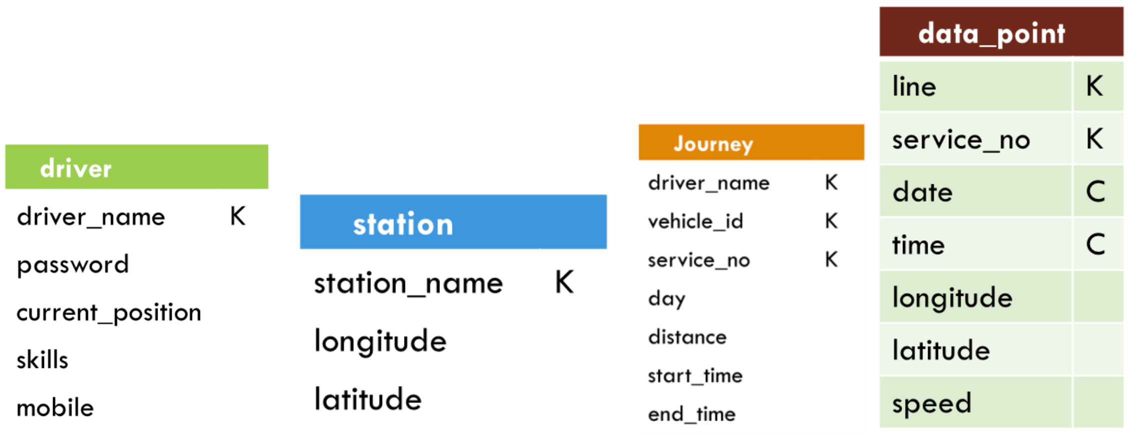


sequence diagram for driver actions

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:

Query group	Queries
Driver	• Select password by driver name • Update current_position • Add notification
Station	• Select from list of stations
Datapoint	• Insert data records
Journey	• Record travel info

The second version of our logical schema for this use cases



driver	
driver_name	K
password	
current_position	
skills	
mobile	

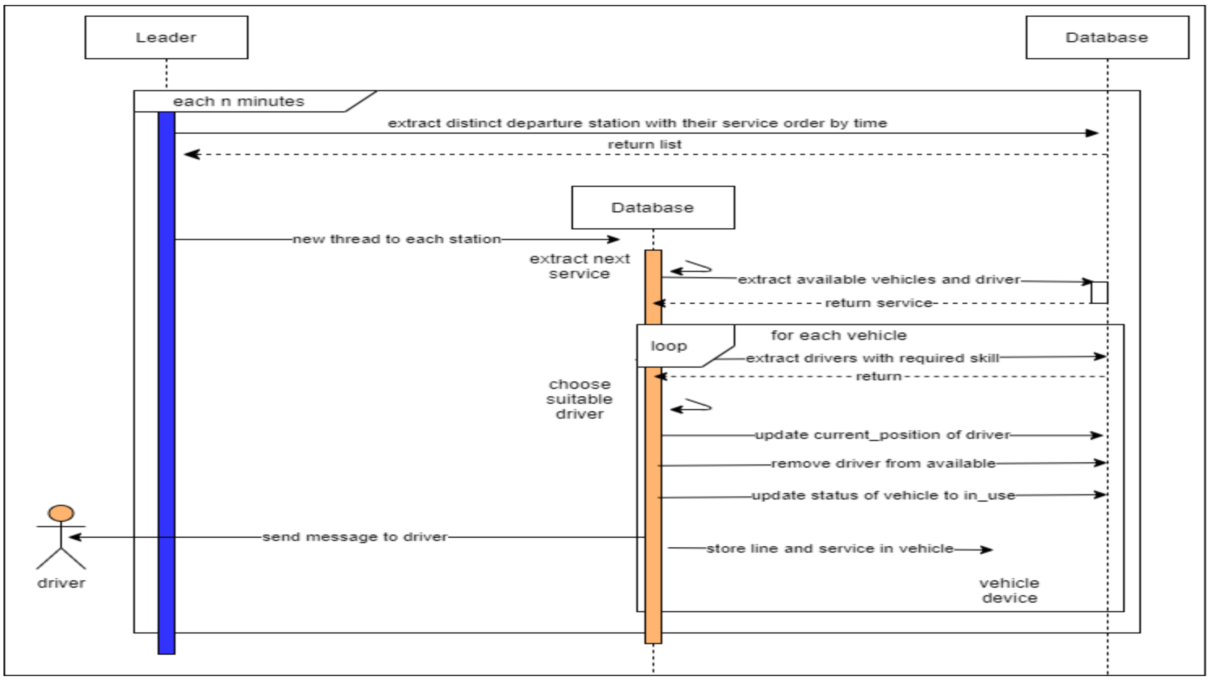
station	
station_name	K
longitude	
latitude	

Journey	
driver_name	K
vehicle_id	K
service_no	K
day	
distance	
start_time	
end_time	

data_point	
line	K
service_no	K
date	C
time	C
longitude	
latitude	
speed	

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:



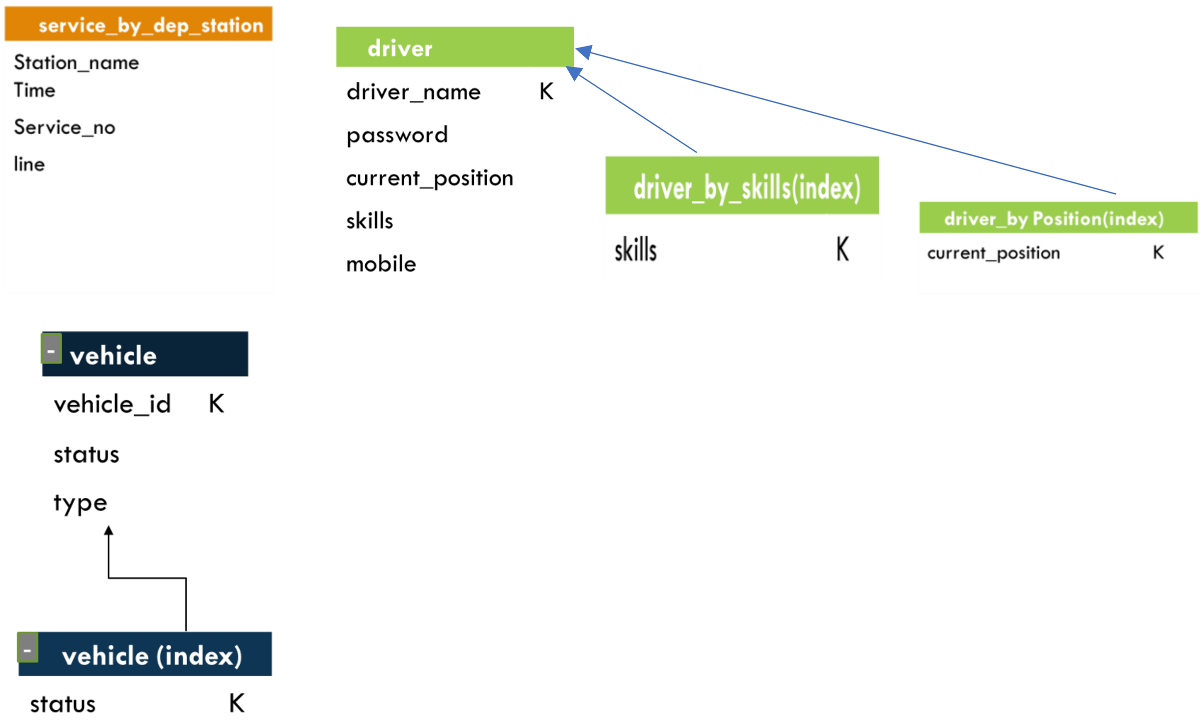
Leader
Database
each n minutes
extract distinct departure station with their service order by time
return list
Database
new thread to each station
extract next service
extract available vehicles and driver
return service
loop
for each vehicle
extract drivers with required skill
return
choose suitable driver
update current_position of driver
remove driver from available
update status of vehicle to in_use
send message to driver
store line and service in vehicle
driver
vehicle device

sequence diagram of allocation

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

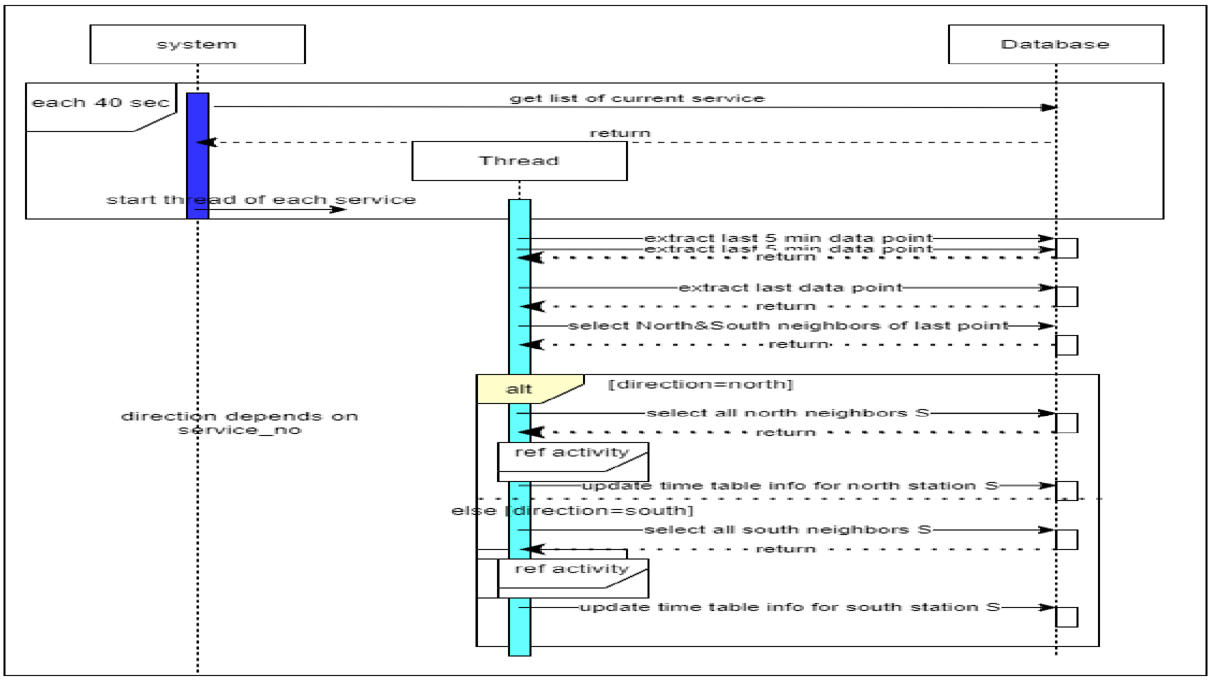
Query group	Queries
Sercice_by_dep_station	• Select list of stations with services ordered by dep time
Driver with index on current position	• Extract available drivers in a station. • Extract drivers that can use a specific vehicle. • Update current position of vehicle to vehicle_id
Vehicle index on status	• Extract available vehicles in a station. • Update vehicle id to in_use

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:

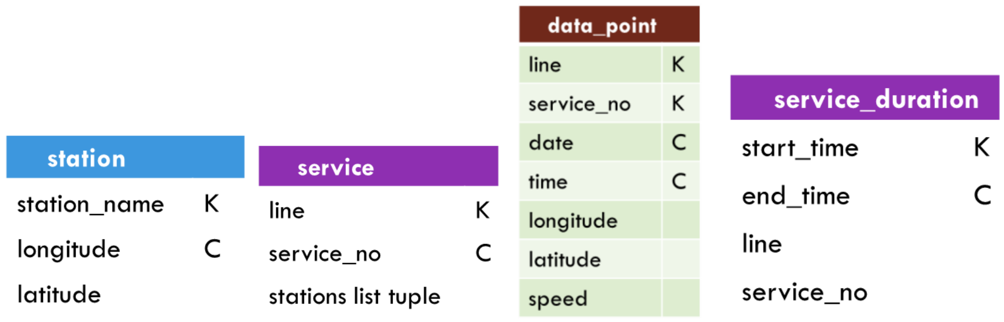

system
Database
each 40 sec
get list of current service
return
Thread
start thread of each service
extract last 5 min data point
extract last 5 min data point
return
extract last data point
return
select North&South neighbors of last point
return
alt
[direction=north]
direction depends on service_no
select all north neighbors S
return
ref activity
update time table info for north station S
else [direction=south]
select all south neighbors S
return
ref activity
update time table info for south station S

sequence diagram of time table update

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

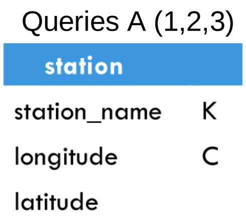
Query group	Queries
Service_duration	• Select list current services
Data_point	• Select last data point of a service • Select data points of a time interval in a day
Service	• Select total distance of a given service • Select stations of a service
Station	• Select all/the of the north neighbors of a given latitude from a list of stations • Select all/the of the south neighbors of a given latitude from a list of stations

last version of logical schema



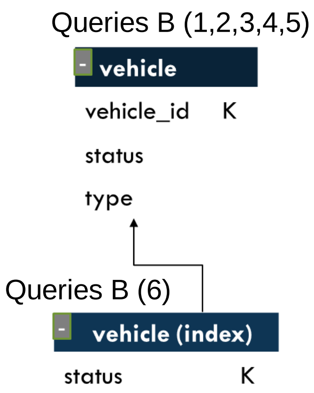
Logical Schema

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

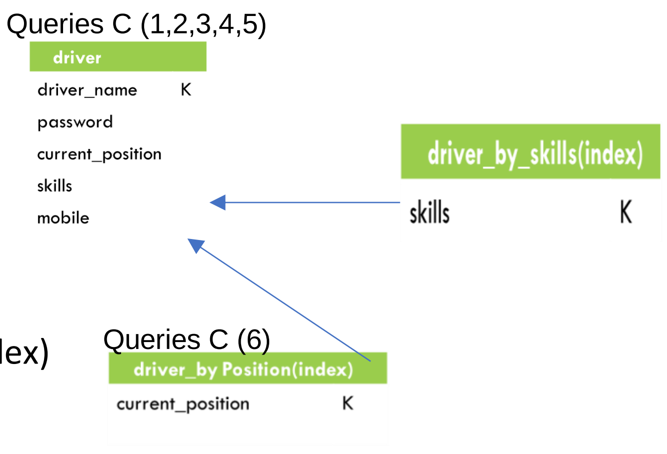


Queries B:

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

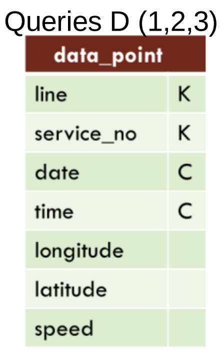
Queries C:

1. 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)



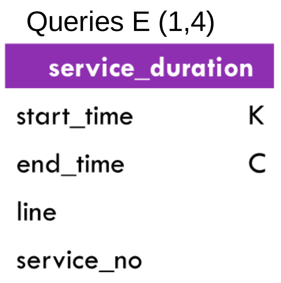
Queries D:

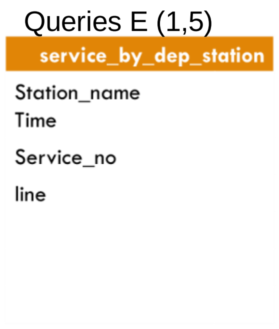
1. Insert data records
2. Select data points for a given Line_service in time interval of day
3. Select last data points for a given Line_service and day

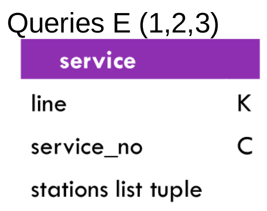


Queries E:

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.

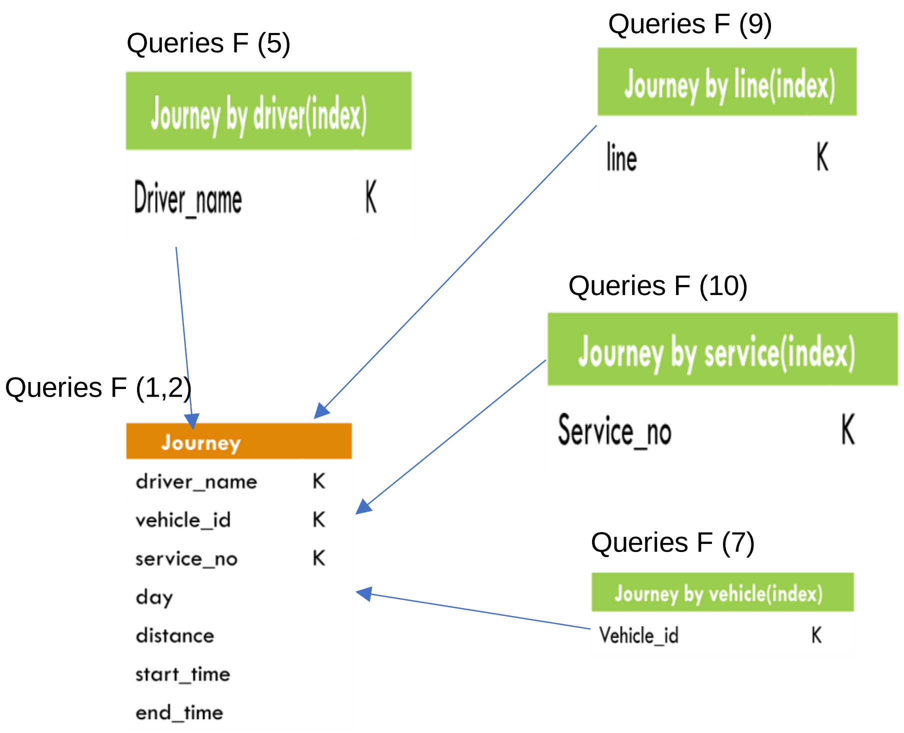






Queries F:

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)



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'); 2. DELETE FROM Driver WHERE driver_name = 'driverName'; 3. UPDATE Driver SET skill = driverSkill WHERE driver_name = 'driverName'; 4. SELECT password FROM Driver WHERE driver_name = 'driverName'; 5. UPDATE Driver SET current_position='current_position' WHERE driver_name= 'driverName'; 6. SELECT driver_name FROM Driver WHERE current_position = 'current_position'; 7. 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); 2. select tuple_distance(lastOfList(distance)) from Service where line='AAAA' and service_no=1; 3. select line,service_no from Service_duration where TOKEN(startTime)<TOKEN(1203) AND endTime>1203 ALLOW FILTERING 4. 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; 5. 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 9. SELECT count (*) from Journey where line='Brignole_Brin' ALLOW FILTERING; 10. SELECT count (*) from Journey where service_no=3 ALLOW FILTERING;