Table of Contents

[Artifacts 2](#_Toc44870046)

[Generating – Parquet data files 2](#_Toc44870047)

[Creation scripts 2](#_Toc44870048)

[Records table 2](#_Toc44870049)

[Sensors table 2](#_Toc44870050)

[Queries 3](#_Toc44870051)

[General report 3](#_Toc44870052)

[Report for license plate YUYRB78292 3](#_Toc44870053)

[Reasoning 4](#_Toc44870054)

[Schema 4](#_Toc44870055)

[Query 6](#_Toc44870056)

[Cost estimation 7](#_Toc44870057)

[Alternative approaches 9](#_Toc44870058)



# Artifacts

## Generating – Parquet data files

I built script which convert the data from CSV / JSON(GZIP) to Parquet (GZIP) format (I also created README.md which describes the usage), In addition attached converted files (Directory - parquet\_converted\_files\_20200705-203559):

*Python3* ./converter --src ./converter/data --dest ./ --bucket <bucket\_name >

Will generate the following directory structure as follow (In S3 and locally):

A picture containing building

Description automatically generated

## Creation scripts

### Records table

*CREATE EXTERNAL TABLE* IF *NOT EXISTS* records (  
 `LicensePlate` string,  
 `Sensor` *int*,  
 `Time` string  
)  
STORED *AS* PARQUET  
LOCATION 's3://idc-ex2/records/'  
tblproperties ("parquet.compress"="gzip")

### Sensors table

*CREATE EXTERNAL TABLE* IF *NOT EXISTS* sensors (  
 `Sensor` *int*,  
 `Pricing` *double*)  
STORED *AS* PARQUET  
LOCATION 's3://idc-ex2/sensors/'  
tblproperties ("parquet.compress"="gzip"

## Queries

### General report

*SELECT* LicensePlate *as* "License plate",  
*SUM*(Pricing) *as* "Total Cost",  
DATE\_FORMAT(date\_parse(*Time*,'%Y-%m-%d %h:%i:%s'),'%m') *AS* "Month",  
DATE\_FORMAT(date\_parse(*Time*,'%Y-%m-%d %h:%i:%s'),'%Y') *AS* "Year",  
*count*(\*) *as* "Number of tolls"  
*FROM* records  
*JOIN* sensors *ON* records.Sensor=sensors.Sensor  
*GROUP BY* LicensePlate,  
DATE\_FORMAT(date\_parse(*Time*,'%Y-%m-%d %h:%i:%s'),'%Y'),  
DATE\_FORMAT(date\_parse(*Time*,'%Y-%m-%d %h:%i:%s'),'%m')

Output:

A screenshot of a cell phone

Description automatically generated

### Report for license plate YUYRB78292

*WITH* report *AS* (  
 *SELECT* LicensePlate *as* "License plate",  
 *SUM*(Pricing) *as* "Total Cost",  
 DATE\_FORMAT(date\_parse(*Time*,'%Y-%m-%d %h:%i:%s'),'%m') *AS* "Month",  
 DATE\_FORMAT(date\_parse(*Time*,'%Y-%m-%d %h:%i:%s'),'%Y') *AS* "Year",  
 *count*(\*) *as* "Number of tolls"  
 *FROM* records  
 *JOIN* sensors *ON* records.Sensor=sensors.Sensor  
 *GROUP BY* LicensePlate,  
 DATE\_FORMAT(date\_parse(*Time*,'%Y-%m-%d %h:%i:%s'),'%Y'),  
 DATE\_FORMAT(date\_parse(*Time*,'%Y-%m-%d %h:%i:%s'),'%m')  
 )  
  
*select \*  
FROM* report  
*WHERE* "License plate"='YUYRB78292';

Output:

A screenshot of a cell phone

Description automatically generated

# Reasoning

## Schema

First I chose to split the data to 2 different directories in order to query only the data I need in specific query, Therefor I split the data to 2 directories as follow:

1. Records – Directory which contains all cars records.
2. Sensors – Sensors pricing table.

When choosing the schema I considered the following objectives:

1. Cost:
   1. S3 storage cost.
   2. Athena query cost.
2. Write:
   1. S3 – execution time.
   2. Athena – No option for writing data.
3. Read:
   1. S3 – execution time.
   2. Athena – execution time.

The following file formats supported by Athena:

A screenshot of a cell phone

Description automatically generated

I chose to focus on 2 file format which considered to be optimized to BIG-data handling:

* Parquet.
* Avro.

Comparing the objectives of the 2 files formats:

1. Cost – As we can see in the following artifacts of the same data using GZIP, We will get smaller size when using Parquet/Avro (The screenshot is Parquet(GZIP)):



Therefor we will achieve:

1. S3 – lower cost using Parquet/Avro storing/updating the data.
2. Athena – lower cost using Parquet/Avro due to fact that it charged on size of the data which scans (compressed or not).
3. Write – In any case for updating the data:
   1. Updating existing file – We will need to download the file -> update it -> upload it, In those case we will achieve faster execution when the size of the data is reduced as possible due to internet speed , therefore we will achieve better performance using Avro/Parquet as I showed before the size of those formats is smaller than json.
   2. Creating new data – In this scenario we will only need to upload the new file, So again we will achieve better performance as we reduce the file size, so we will choose Avro/Parquet.

\*\*\* Avro known for the speed of its writing, However its not our concern currently.

1. Read – Parquet known for its reading performance compare to all other file formats, the task requires mostly reading a lot of data when the data scale up, Therefor for achieving better reading we will choose Parquet.

A screenshot of a cell phone

Description automatically generated

<https://www.datanami.com/2018/05/16/big-data-file-formats-demystified/>

In conclusion, I will choose parquet file format for performing this monthly task.

## Query

*-- Choosing LicensePlate as first column  
 SELECT* LicensePlate *as* "License plate",  
*-- Total cost calculation based on grouping by License Plate, Month, Year with same values  
 SUM*(Pricing) *as* "Total Cost",  
*-- Month columnt* DATE\_FORMAT(date\_parse(*Time*,'%Y-%m-%d %h:%i:%s'),'%m') *AS* "Month",  
*-- Year column* DATE\_FORMAT(date\_parse(*Time*,'%Y-%m-%d %h:%i:%s'),'%Y') *AS* "Year",  
*-- Number of tolls based on grouping by License Plate, Month, Year with same values  
 count*(*\**) *as* "Number of tolls"  
 *FROM* records  
*-- Joining sensors price to each recode for allowing calculation  
 JOIN* sensors *ON* records.Sensor=sensors.Sensor  
*-- Group rules as specified above  
 GROUP BY* LicensePlate,  
 DATE\_FORMAT(date\_parse(*Time*,'%Y-%m-%d %h:%i:%s'),'%Y'),  
 DATE\_FORMAT(date\_parse(*Time*,'%Y-%m-%d %h:%i:%s'),'%m')

In line comment – explaining each expression in the Query.

# Cost estimation

Given the following:

* Sensors = 15,000
* Records every day = 50,000
* Days in month = 30

A screenshot of a cell phone

Description automatically generated**Size estimation:**

Calculating record size by given data:

Number of records: **317881**

When the data size is:



Therefor estimated size of each record is:

A screenshot of a cell phone

Description automatically generatedCalculating sensor size by given data:

Number of sensors: **171**

When the data size is: **3KB**



Therefor estimated size of each record is:

**Cost calculation:**

Assuming we have 1 month of data, Data size which will be scanned is:

Athena

Assuming the following pricing:

A screenshot of a cell phone

Description automatically generated

Athena charged by size data which scanned, Therefor for specific month the cost will be:

S3

Assume the following S3-Standard pricing:

A screenshot of a cell phone

Description automatically generated

In total assuming every month we are deleting old data it will cost 7.322836$ per month.

# Alternative approaches

Preparing cluster of instances which store the data or single instance which manage the data, For example creating RDS instance with MySQL DB, Storing and updating the data in the instance.

**Using Athena**

Advantages

* Server less.
* Much lower price – EBS/EFS used in RDS is much more expensive than S3 storage.
* Support of wide range of data formats – out of the box.
* Fast queries – without the need to ETL (Extract transform load data).
* Good UI.
* IAM – can be managed in AWS consoles.
* Ease to create.

Dis-advantages

* Can’t write back to S3 (INSERT SQL command).
* Not all DLLs functions supported.
* Work only on external tables.
* Size limitation in Amount of tables for example.

**Using RDS**

Advantages

* Able to make everything in your own way and configuration:
  + Functions you created.
  + Security enforcement.
* Able to insert data to existing tables without replacing many records.
* Could be faster due to more accessible storage (Like storage in RAM).

Dis-advantages

* Much more expensive – EBS/EFS is much more expensive than most S3 storage classes.
* Require specific steps to create (Not out of the box solution).