Desafio CIVITAS - EMD

Caroline Sodré

Apresentação:

O desafio consiste em utilizar os dados disponibilizados em uma tabela BigQuery que contém leituras de radar do município do Rio de Janeiro, a fim de identificar possíveis placas clonadas. Para isso, devemos realizar uma análise exploratória com o objetivo de entender melhor a natureza e disposição dos dados, localizando inconsistências. Além disso, é necessário desenvolver uma query SQL que identifique e retorne possíveis placas clonadas.

A tabela disponibilizada tem o seguinte esquema:

Coluna	Tipo	Descrição
datahora	TIMESTAMP	Data e hora da leitura
datahora_captura	TIMESTAMP	Data e hora da captura pelo radar
placa	BYTES	Placa do veículo capturado
empresa	STRING	Empresa do veículo
tipoveiculo	STRING	Tipo do veículo
velocidade	INTEGER	Velocidade do veículo
camera_numero	STRING	Número identificador da câmera
camera_latitude	FLOAT	Latitude da câmera

camera_longitude	FLOAT	Longitude da câmera

Antes de decidirmos a melhor maneira de resolver o problema proposto, nos voltaremos a entender melhor os dados disponibilizados.

Análise Exploratória:

1. Visualizando a tabela original:

O primeiro passo será visualizarmos os dados. Para isso, utilizamos a query a seguir, extraindo da tabela “rj-cetrio.desafio.readings_2024_06”:

```
-- Visualizando a tabela original

SELECT *
FROM
  `rj-cetrio.desafio.readings_2024_06`;
```

O que nos retorna:

datahora	datahora_captura	placa	empresa	tipoveiculo
2024-06-03 16:58:07 UTC	2024-06-03 18:08:04 UTC	Q9k+K3avu5cF2QWgFLbVncY=	BRASCONTROL	ciclomotor
2024-06-07 00:53:57 UTC	2024-06-07 00:55:49 UTC	ljD08aSJizf0ZYxVLg/QJxg=	BRASCONTROL	automovel
2024-06-03 19:54:53 UTC	2024-06-03 19:57:40 UTC	pV8ksnS6IKU2Ew4U93yodYc=	BRASCONTROL	automovel
2024-06-05 20:05:07 UTC	2024-06-05 20:11:13 UTC	4m4HLsNjb+59bnEnZmx5yP4=	BRASCONTROL	automovel
2024-06-05 14:49:23 UTC	2024-06-05 14:50:01 UTC	Mi0AFZpmgcfp9/Ldz1GiPSQ=	CONSILUX	automovel
2024-06-07 10:12:47 UTC	2024-06-07 10:13:01 UTC	Dqu3jeT/JvNQ1vDeqtwFpEE=	CONSILUX	automovel
2024-06-09 22:30:01 UTC	2024-06-09 22:31:01 UTC	Rpf3MVkAQSBXGvAief0QjIc=	CONSILUX	automovel
2024-06-04 16:12:59 UTC	2024-06-04 16:14:01 UTC	knJqa0/MyiKmFXVRtJPjkZs=	CONSILUX	automovel
2024-06-10 13:33:41 UTC	2024-06-10 13:34:01 UTC	wuEA4oMC7z3hsy1lEcXlD1s=	CONSILUX	automovel
2024-06-09 23:29:58 UTC	2024-06-09 23:31:01 UTC	/MhCxgXp76/lO6eiwGUqGns=	CONSILUX	automovel

Results per page: 50 1 – 50 of 36358536

velocidade	camera_numero	camera_latitude	camera_longitude
5	690 - 1	-22.881748001	-43.294197001
86	718 - 2	-22.958334001	-43.197136001
86	719 - 2	-22.861619001	-43.267878001
11	757 - 3	-22.954242001	-43.193024001
72	520121111	-22.8971169	-43.18339028
65	520121113	-22.8971169	-43.18339028
67	520121122	-22.8969913	-43.18318469
68	520121123	-22.8969913	-43.18318469
66	520241122	-22.887337	-43.224028
73	520291124	-22.843854	-43.250147

Results per page: 50 1 – 50 of 36358536

Olhando apenas as dez primeiras linhas, os dados parecem estar conforme o esperado, mas, para termos certeza, teremos que realizar algumas contagens.

Vemos também que a tabela apresenta mais de 36 milhões de linhas.

2. Contagem de Null:

A primeira contagem que faremos identifica quantas entradas “null” existem em cada coluna:

```
-- Realizando a contagem de Nulls de cada coluna

SELECT
    COUNTIF(datahora IS NULL) AS datahora_null,
    COUNTIF(datahora_captura IS NULL) AS datahora_captura_null,
    COUNTIF(placa IS NULL) AS placa_null,
    COUNTIF(empresa IS NULL) AS empresa_null,
    COUNTIF(tipoveiculo IS NULL) AS tipoveiculo_null,
    COUNTIF(velocidade IS NULL) AS velocidade_null,
    COUNTIF(camera_numero IS NULL) AS camera_numero_null,
    COUNTIF(camera_latitude IS NULL) AS camera_latitude_null,
    COUNTIF(camera_longitude IS NULL) AS camera_longitude_null
FROM
  `rj-cetrio.desafio.readings_2024_06`;
```

Retornando:

datahora_null	datahora_captura_null	placa_null	empresa_null	tipoveiculo_null	velocidade_null	camera_numero_null	camera_latitude_null	camera_longitude_null
0	1816325	0	0	0	0	0	0	0

Nota-se, portanto, que a coluna “datahora_captura”, que contém a data e hora da captura pelo radar, apresenta muitas entradas null (cerca de 5%). Poderíamos adotar alguma estratégia para tratar esse problema, mas, como temos acesso à informação da data e hora da leitura (coluna “datahora”), decidimos apenas não utilizar esse dado em nossa análise.

3. Entendendo os tipos de veículos:

Analisaremos agora os tipos de veículos presentes na tabela. Para isso, utilizaremos a query:

```
-- Análise dos tipos de Veículo

SELECT
  tipoveiculo,
  COUNT(*) AS total_tipoveiculo
FROM
  `rj-cetrio.desafio.readings_2024_06`
GROUP BY tipoveiculo;
```

Cujo resultado é:

tipoveiculo	total_tipoveiculo
automovel	34386894
ciclomotor	1157096
caminhao	331696
onibus	482850

Assim, sabemos agora que estamos lidando com dados de quatro tipos de veículos diferentes, sendo o carro o mais recorrente.

4. Análise dos tipos de Empresa:

Com uma query semelhante, conseguimos ter acesso a lista de empresas existentes:

```
-- Análise dos tipos de Empresa

SELECT
  empresa,
  COUNT(*) AS total_empresa
FROM
  `rj-cetrio.desafio.readings_2024_06`
GROUP BY empresa;
```

Onde temos:

empresa	total_empresa
BRASCONTROL	3166599
PERKONS	8717449
CONSILUX	24474488

Ou seja, existem leituras de três empresas diferentes na tabela, sendo a CONSILUX a mais recorrente (cerca de 67%).

5. Checando os diferentes tipos de número de câmera:

Para checarmos os tipos de número de câmera presentes na nossa tabela, faremos uma contagem das vezes em que cada entrada se repete:

```sql
-- Análise dos valores de camera_numero

SELECT
  camera_numero,
  COUNT(*) AS count_camera_numero
FROM
  `rj-cetrio.desafio.readings_2024_06`
GROUP BY
  camera_numero;
```

O que nos retorna:

Row	camera_numero	count_camera_nume
1	685 - 4	12353
2	686 - 3	14517
3	695 - 1	7660
4	718 - 2	43913
5	719 - 1	24970
6	731 - 2	6465
7	736 - 1	53116
8	741 - 1	5078
9	747 - 1	38787
10	748 - 3	11409
11	750 - 3	9228

Results per page: 50 1 – 50 of 1421

Com isso, vemos que existem 1421 câmeras diferentes, cada uma capturando em média em torno de 25.500 registros.

6. Análise dos valores de Latitude:

Para entendermos melhor a faixa de valores presentes de Latitude na tabela, executaremos a query:

```sql
-- Analise dos valores mínimos e máximos de Latitude

SELECT
  MIN(camera_latitude) AS camera_latitude_min,
  MAX(camera_latitude) AS camera_latitude_max,
FROM
  `rj-cetrio.desafio.readings_2024_06`;
```

Que nos retorna os valores mínimo e máximo de Latitude:

Row	camera_latitude_min	camera_latitude_max
1	-23.8597222	0.0

O valor mínimo parece estar de acordo com o esperado, mas o valor máximo provavelmente é um erro. Retiraremos os zeros da consulta e repetiremos a análise:

```
SELECT
  MIN(camera_latitude) AS camera_latitude_min,
  MAX(camera_latitude) AS camera_latitude_max,
FROM
  `rj-cetrio.desafio.readings_2024_06`
WHERE
  camera_latitude != 0;
```

Cujo o resultado é:

Row	camera_latitude_min	camera_latitude_max
1	-23.8597222	-22.4850965

Agora os valores parecem mais plausíveis - retiraremos os valores zerados de latitude nas análises futuras.

7. Estudo dos valores de Longitude:

Para estudarmos os valores presentes de longitude, realizaremos uma consulta parecida com a do caso anterior:

```
-- Analise dos valores mínimos e máximos de Longitude

SELECT
  MIN(camera_longitude) AS camera_longitude_min,
  MAX(camera_longitude) AS camera_longitude_max,
FROM
  `rj-cetrio.desafio.readings_2024_06`;
```

Onde obtemos:

Row	camera_longitude_m	camera_longitude_m
1	-43.690229	43.334218

Como no caso da latitude, o valor mínimo parece plausível, mas o valor máximo possivelmente é um erro. Refaçamos a consulta retirando todos os valores maiores que zero:

```sql
SELECT
  MIN(camera_longitude) AS camera_longitude_min,
  MAX(camera_longitude) AS camera_longitude_max,
FROM
  `rj-cetrio.desafio.readings_2024_06`
WHERE
  camera_longitude < 0;
```

Cujo resultado é:

Row	camera_longitude_m	camera_longitude_m
1	-43.690229	-43.1223252

Agora a faixa de valores de longitude parece correta - retiraremos os valores de longitude maiores ou iguais à zero nas análises futuras.

8. Análise dos valores de velocidade:

Mais uma vez, utilizaremos as funções MIN e MAX:

```sql
-- Análise do range de Velocidade

SELECT
  MIN(velocidade) AS velocidade_min,
  MAX(velocidade) AS velocidade_max,
FROM
  `rj-cetrio.desafio.readings_2024_06`;
```

Nos retornando:

Row	velocidade_min	velocidade_max
1	0	255

Aqui, temos um valor mínimo zerado, o que parece ser um erro, e um valor máximo de 255 km/hr. Antes de decidirmos o que fazer, realizaremos uma contagem de quantas leituras com velocidade acima de 150 km/hr existem em nossa base:

```sql
-- Contanto quantos veículos foram registrados com velocidade acima de 150

SELECT COUNT(*) AS velocidade_maior_150
FROM
  `rj-cetrio.desafio.readings_2024_06`
WHERE
 velocidade > 150;
```

O que nos retorna:

Row	velocidade_maior_15
1	5049

Ou seja, apenas 0,01% dos casos. Para determinar se essas leituras são ou não, de fato, erros, uma análise mais profunda avaliando o local e hora parece ser necessária. Como os registros mal compõem 0.01% das entradas, parece seguro mantê-los em nossa análise. Retiraremos então apenas os valores de velocidade zerados.

9. Checando as repetições de cada placa:

Para isso, adotaremos uma abordagem parecida com a já feita anteriormente, e realizaremos a seguinte contagem:

```
-- Análise os tipos de placas e quantas vezes repetem

SELECT
  placa,
  COUNT(*) AS total_placa
FROM
  `rj-cetrio.desafio.readings_2024_06`
GROUP BY
  placa;
```

Cujo resultado é:

Row	placa	total_placa
1	apZgsqFvb45Tic2UWh9Nak0=	68
2	WH52rFnBnjKcl8uLR+jsvRw=	129
3	JNaGNfQQFRalLte6jZIcsYs=	48
4	Cb/bBkEv+Nd7dO2gLBkyuWI=	32
5	3N3oXrHDDIWljMGc4+R+8hE=	101
6	97QCPSfAtTNvjlYGG5cL2jY=	22
7	AC3Z3oeqFsnFnaoJlJ4wNx8=	54
8	OczQfL8AyxG3QG88IhAzYmY=	33
9	mcKxNIgdIw4JzjQGwsofifw=	86
10	JuvWf1nwEvQRtJJZkAz7MR8=	19

Results per page: 50 1 – 50 of 7984610

Ou seja, há um total de 7984610 placas em nossa tabela, configurando uma média de 22 leituras por placa.

10. Entendendo o intervalo temporal do problema:

Finalmente, passaremos à análise do intervalo temporal do problema. Para isso, separaremos os dados da coluna “datahora” em ano, mês, dia, hora, minuto e segundo:

```
-- Criando uma View com os dados transformados

CREATE OR REPLACE VIEW temporary_table.vw_dados_new AS
SELECT
  *,
  EXTRACT(YEAR FROM TIMESTAMP(datahora)) AS ano,
  EXTRACT(MONTH FROM TIMESTAMP(datahora)) AS mes,
  EXTRACT(DAY FROM TIMESTAMP(datahora)) AS dia,
  EXTRACT(HOUR FROM TIMESTAMP(datahora)) AS horas,
  EXTRACT(MINUTE FROM TIMESTAMP(datahora)) AS minutos,
  EXTRACT(SECOND FROM TIMESTAMP(datahora)) AS segundos
FROM
  `rj-cetrio.desafio.readings_2024_06`;
```

Realizaremos contagens de máximo e mínimo, similares às já apresentadas, para cada uma dessas variáveis com o intuito de checar se há alguma inconsistência. As querys podem ser encontradas no arquivo “Contagem_Placas.sql”.

A partir dos resultados, concluímos que os dados da tabela se encontram entre o período de 3 a 10 de junho de 2024.

Além disso, foi possível constatar que nenhum valor inesperado foi identificado, portanto, seguiremos para o próximo passo.

Construindo uma solução para o problema:

Agora que compreendemos melhor os dados que temos em mãos, podemos nos concentrar na construção de uma solução para o problema proposto.

Essencialmente, o problema se resume a encontrar uma relação entre o tempo decorrido e a distância percorrida por cada placa em registros subsequentes, escolhendo um limite realisticamente cabível para a realização do trajeto - o que nos leva a velocidade média.

Com a primeira parte resolvida, basta escolhermos um limite de velocidade média no qual, acima dele, classificaremos as leituras como a de possíveis placas clonadas. Considerando que os dados são do município do Rio de Janeiro, cujo tráfego é dinâmico, parece cabível assumirmos um limite de 150 km/hr.

Agora que estruturamos uma possível solução para o problema proposto, devemos entender quais são os passos necessários para implementá-la.

Primeiro, devemos remover os casos em que velocidade e camera_latitude são nulos, além dos em que camera_longitude são maiores ou iguais à zero.

Com isso, temos:

```sql
-- Criando uma view apenas com as colunas relevantes para o problema, retirando os valores nulos de
-- latitude, longitude e velocidade

CREATE OR REPLACE VIEW temporary_table.auxiliary_dataset AS
SELECT
  placa,
  datahora,
  camera_latitude,
  camera_longitude,
FROM
  `rj-cetrio.desafio.readings_2024_06`
WHERE
  -- Removendo os regristros em que camera_latitude e velocidade são nulos, e camera_longitude é positivo ou nulo
  camera_latitude != 0
  AND camera_longitude < 0
  AND velocidade != 0;
```

Checando a view “auxiliary_dataset”, temos:

Row	placa	datahora	camera_latitude	camera_longitude
1	ahoVeE5DXP+aVe0O3DOeBnI=	2024-06-06 18:04:15 UTC	-22.91843	-43.365114
2	9Bzru/uGH88RLQ9iQGF/9tU=	2024-06-08 17:24:17 UTC	-22.963997	-43.394165
3	KybwuEmjXMkidkLnmlomrzY=	2024-06-05 10:07:09 UTC	-22.989424	-43.417077
4	uQV8O/divcl3jVV7p9/aH1I=	2024-06-10 12:18:56 UTC	-22.902954	-43.249439
5	6KZY9yLaiy+4w3JCU/vxJBg=	2024-06-03 23:15:52 UTC	-22.9202612	-43.2593741
6	W4iLjdkTuATbS37kMRdzkU4=	2024-06-06 21:44:23 UTC	-22.9321811	-43.5745980555…
7	GDi3SYVSa0AWQOuYSi2/Lxg=	2024-06-08 02:24:08 UTC	-23.003303	-43.324678
8	UG30/gg6BFIDuQbMdSRNy4M=	2024-06-03 19:41:10 UTC	-22.95997	-43.388318
9	BavIzJ30pxoJ6cXTGDAuVXw=	2024-06-09 23:08:41 UTC	-22.887477	-43.345052
10	eW8GFTkJxFxCHEj1jfTbDbI=	2024-06-05 16:16:26 UTC	-23.001696	-43.328769
11	AI4l2GfNLxrkplq7Ez6H+/M=	2024-06-04 20:53:24 UTC	-22.90638889	-43.19388889
12	lOcyO9KfDUdB+KlX1jY11jA=	2024-06-05 00:38:20 UTC	-22.882317	-43.522371
13	rdJdiuBinlYMIzsn7hLlamo=	2024-06-08 21:02:22 UTC	-22.94903	-43.339187
14	955I5PxTY8EOVTHv0FmmIIY=	2024-06-09 13:56:09 UTC	-22.8701569444…	-43.5545261111…
15	TMpJ+gvL4AfUhf2tctnYN+A=	2024-06-07 15:31:38 UTC	-22.963587	-43.394159
16	VL56Ndfdbr+nNxiUm3FYqXk=	2024-06-09 13:12:56 UTC	-22.967531	-43.397387

Agora, podemos calcular o tempo decorrido e a distância percorrida por cada placa em registros subsequentes. Para isso, faremos uso das funções abaixo.

1. ST_GeogPoint:

Essa função nos permite criar um ponto a partir dos parâmetros especificados de longitude FLOAT64 (em graus, negativa a oeste do Meridiano de Greenwich, positiva a leste) e latitude (em graus, positiva ao norte do Equador, negativa ao sul) e retorna esse ponto em um valor de “GEOGRAPHY”. [1]

Portanto, ela é fundamental para manipulação e análise de dados geoespaciais em bancos de dados, permitindo a criação precisa de pontos geográficos com base em suas coordenadas de latitude e longitude, além da utilização desses pontos para cálculos diversos.

2. TIMESTAMP_DIFF:

Obtém o número de fronteiras de unidade entre dois valores de TIMESTAMP em uma granularidade de tempo específica (microsecond: microsegundo; millisecond: milissegundo; second: segundo; minute: minuto; hour: hora e day: dia) [2]. Utilizaremos a granularidade em segundos para obter um resultado mais acurado.

3. ST_Distance

Retorna a menor distância em metros entre duas GEOGRAFIAS não vazias.

O parâmetro opcional use_spheroid determina como esta função calcula a distância. Se use_spheroid é FALSE, a função calcula a distância na superfície de uma esfera perfeita. Se use_spheroid é TRUE, a função calcula a distância levando em consideração a forma elipsoidal da Terra [3]. Utilizaremos o parâmetro use_spheroid como TRUE.

Com essa função, conseguimos realizar os cálculos de distância de forma mais simples, sem ter que recorrer a outras opções como, por exemplo, implementar a Fórmula de Haversine.

Fazendo uso de tais funções, conseguimos construir a query abaixo:

```
CREATE OR REPLACE VIEW temporary_table.final_view AS
SELECT
  s.placa AS placa,
  s.datahora AS starting_time,
  e.datahora AS end_time,
  -- Criando pontos geográficos usando as coordenadas de latitude e longitude
  ST_GeogPoint(s.camera_longitude, s.camera_latitude) AS starting_point,
  ST_GeogPoint(e.camera_longitude, e.camera_latitude) AS end_point,
  -- Calculando a diferença de tempo levado entre o starting_point e end_point em segundos, usando a função TIMESTAMP_DIFF
  TIMESTAMP_DIFF(e.datahora, s.datahora, SECOND) AS time_taken,
  -- Calculando a distancia em metros entre o starting_point e end_point, usando a função ST_Distance
  ST_Distance(ST_GeogPoint(s.camera_longitude, s.camera_latitude), ST_GeogPoint(e.camera_longitude, e.camera_latitude), true)
  AS travelled_distance,
  -- Calculando a velocidade média (em m/s) a partir da distância percorrida e do tempo levado
  ST_Distance(ST_GeogPoint(s.camera_longitude, s.camera_latitude), ST_GeogPoint(e.camera_longitude, e.camera_latitude), true)
  / TIMESTAMP_DIFF(e.datahora, s.datahora, SECOND) AS average_speed
```

```sql
FROM (
  SELECT
    placa,
    datahora,
    camera_longitude,
    camera_latitude,
    -- Retornando o próximo valor de datahora para a mesma placa, ordenado pela datahora
    LEAD(datahora) OVER(PARTITION BY placa ORDER BY datahora) AS end_time,
    -- Retornando a próxima longitude para a mesma placa, ordenada pela datahora
    LEAD(camera_longitude) OVER(PARTITION BY placa ORDER BY datahora) AS end_longitude,
    -- Retornando a próxima latitude para a mesma placa, ordenada pela datahora
    LEAD(camera_latitude) OVER(PARTITION BY placa ORDER BY datahora) AS end_latitude
  FROM `projeto-placas-clonadas.temporary_table.auxiliary_dataset`
) s
-- Criando uma relação entre registros consecutivos de uma mesma placa de veiculo, com base na ordem temporal (datahora)
JOIN (
  SELECT
    placa,
    datahora,
    camera_longitude,
    camera_latitude
  FROM `projeto-placas-clonadas.temporary_table.auxiliary_dataset`
) e

ON
  s.placa = e.placa
  AND s.end_time = e.datahora
  AND s.end_longitude = e.camera_longitude
  AND s.end_latitude = e.camera_latitude
WHERE
  s.datahora < e.datahora;
```

Com isso, conseguimos criar uma view onde cada linha representa um par de pontos geográficos consecutivos para uma mesma placa de veículo, junto com o tempo decorrido entre esses pontos e a velocidade média calculada com base na distância percorrida e no tempo decorrido:

placa	starting_time	end_time	starting_point	end_point	time_taken	travelled_distance	average_speed
WZCTkRoDLkB/vFyrUDOj094=	2024-06-05 23:32:46 UTC	2024-06-06 20:12:34 UTC	POINT(-43.309212 -22.850707)	POINT(-43.48833333 -23.0194…	74388	26205.77943498…	0.352285038379…
WkflooQgpeEKxENj7gm9X5k=	2024-06-06 21:08:18 UTC	2024-06-06 21:11:50 UTC	POINT(-43.417077 -22.989424)	POINT(-43.398391 -22.973449)	212	2607.848560123…	12.30117245341…
XUyArSxGLVi8pd+2BaBnSeA=	2024-06-05 16:04:34 UTC	2024-06-06 14:36:05 UTC	POINT(-43.3663333 -23.008472)	POINT(-43.226783001 -22.978…	81091	14687.52385609…	0.181123970059…
XooBR/TjvTbo1TIbGFPRQk0=	2024-06-08 19:54:55 UTC	2024-06-08 20:07:28 UTC	POINT(-43.38805556 -23.2338…	POINT(-43.4381894 -23.00725…	753	25618.24238117…	34.02157022732…
YEcNgSk/HWRUfmravJcGJkY=	2024-06-07 21:11:56 UTC	2024-06-08 15:33:07 UTC	POINT(-43.369157 -22.938538)	POINT(-43.338944 -22.95005)	66071	3350.7857219859	0.050714923672…
Yonoe4Vg054gs+hxFIT4xys=	2024-06-06 13:59:35 UTC	2024-06-06 17:34:06 UTC	POINT(-43.5725266666667 -22…	POINT(-43.3712858781407 -22…	12871	22750.56604879…	1.767583408343…
Y7ENNISlWY2J6fKvTpDpbCs=	2024-06-09 20:12:03 UTC	2024-06-09 20:18:42 UTC	POINT(-43.303479 -22.994422)	POINT(-43.285387001 -22.973…	399	2941.295061154…	7.371666819936…
Y/Betncr5nmrXIOwQbBoqdw=	2024-06-04 19:06:07 UTC	2024-06-04 19:54:37 UTC	POINT(-43.398244 -22.972918)	POINT(-43.379452 -22.973064)	2910	1927.054425565…	0.662218015658…
ZNwZaWs5DQBa4HubzGxW2…	2024-06-08 17:14:34 UTC	2024-06-08 17:31:09 UTC	POINT(-43.371769 -22.935662)	POINT(-43.328769 -23.001696)	995	8539.391239972…	8.582302753740…
ZiTqv525uide7cA4VDoJ8xY=	2024-06-09 14:53:52 UTC	2024-06-09 14:55:36 UTC	POINT(-43.409648 -22.981472)	POINT(-43.398391 -22.973449)	104	1456.642180786…	14.00617481525…
Z6j1swlSmFo2Xjk/Q+FqDrU=	2024-06-09 17:34:05 UTC	2024-06-09 17:34:54 UTC	POINT(-43.3572222222222 -22…	POINT(-43.3588888888889 -22…	49	492.0821991294…	10.04249385978…
aMZcOOh6bhUe1S1iB+n93kE=	2024-06-09 10:40:05 UTC	2024-06-09 11:12:37 UTC	POINT(-43.43345 -23.003542)	POINT(-43.4441230555556 -23…	1952	1215.595826771…	0.622743763714…
av+H4pKzw4SBRhxp03kMQe0=	2024-06-09 18:14:56 UTC	2024-06-09 18:15:14 UTC	POINT(-43.19 -22.905)	POINT(-43.18777778 -22.9041…	18	245.9569769827…	13.66427649903…
a5DeQb9gSOvsOBDGQOlmmP…	2024-06-09 15:46:30 UTC	2024-06-09 16:00:01 UTC	POINT(-43.172880001 -22.925…	POINT(-43.1713889 -22.95333…	811	3120.870562459…	3.848175786016…
cAu/ir94iEwAg2bd7dx4X9s=	2024-06-06 20:44:42 UTC	2024-06-06 20:48:42 UTC	POINT(-43.369314 -23.010703)	POINT(-43.3890011 -23.01266…	240	2029.902877237…	8.457928655155…

Finalmente, filtrando pelos casos em que a velocidade média ultrapassa 150 km/hr:

```sql
-- Filtrando apenas os casos com velocidade acimda de 150 km/hr

SELECT
    placa,
    COUNT(*) AS total_placa
FROM
    `projeto-placas-clonadas.temporary_table.final_view`
WHERE
  -- Passando 150 km/hr para m/s
    average_speed > 150/3.6
GROUP BY
    placa;
```

Temos:

Row	placa	total_placa
1	TWC2QsqeouLuzw9LE2Cu3Uc=	25
2	ryPDEU5+dd5NZsLkUAv63ok=	53
3	pDdWl98vQfzF8g5M16G9ctA=	8
4	+R/zCi6sCYzbF781W1ZsGtA=	3000
5	NYFl6bKDxLOKai2vi/soIK4=	16
6	nLnBPRKPK3cvr5AejpIMfT4=	10
7	CioA93OYRUqmJP3/U2DB5KU=	8
8	m7BhTqVn/N8yP1DqmYJH6H...	9
9	CjxkjlfTf6bvsblhrT+bLE4=	27
10	DeZ4W2Ga2PquY3yWxGzrPlE=	9

Results per page: 50 1 – 50 of 167160

Ou seja, mais de 167 mil registros (cerca de 0,4%).

Possíveis melhorias:

Algumas sugestões de melhoria são:

1. Alterar as entradas de velocidade a fim de que comporte números decimais para que tenhamos mais certeza quanto a, por exemplo, as entradas nulas que identificamos;;
2. Automatizar o processo para que a consulta seja feita de forma eficiente a cada novo registro;

Referências:

[1]https://cloud.google.com/bigquery/docs/reference/standard-sql/geography_functions#st_geogpoint

[2]https://cloud.google.com/bigquery/docs/reference/standard-sql/timestamp_functions#timestamp_diff

[3]https://cloud.google.com/bigquery/docs/reference/standard-sql/geography_functions#st_distance