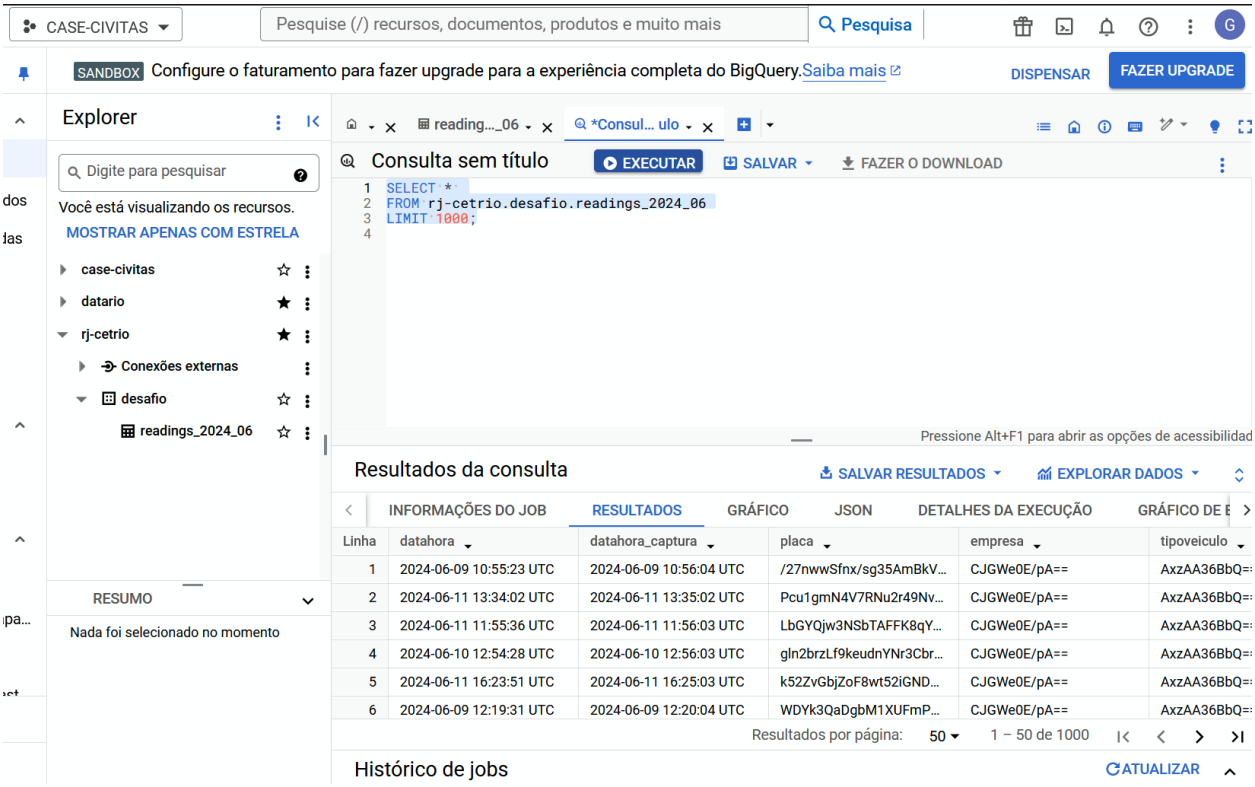
Case - CIVITAS

Análise Exploratória dos Dados (EDA)

Para análise exploratória, normalmente meu processo envolve ter um primeiro contato com os dados da tabela, executando o seguinte comando.

1.1. Visualizar as primeiras linhas da tabela

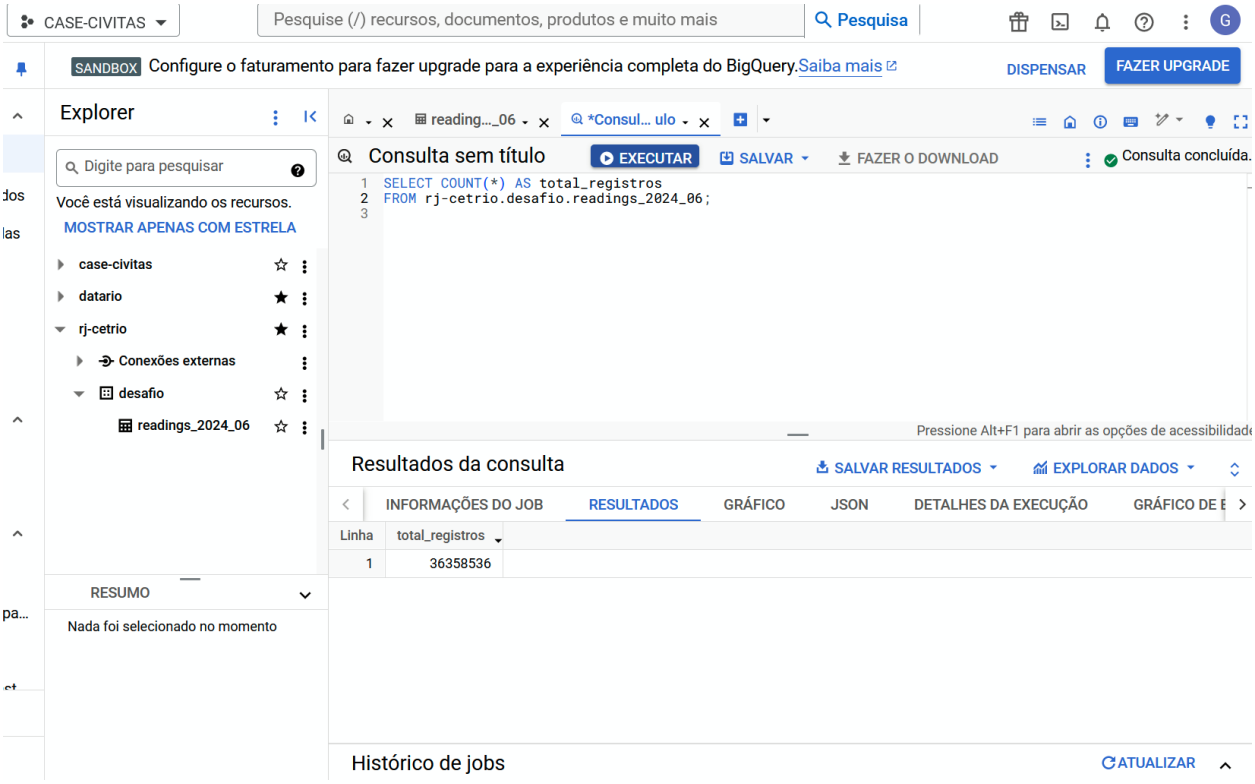
```
SELECT *
FROM rj-cetrio.desafio.readings_2024_06
LIMIT 1000;
```



Aqui eu já consigo ver a formatação de cada coluna e como os dados estão apresentados. Depois, sigo para entender a dimensão da tabela com a seguinte query.

1.2. Contagem total de registros

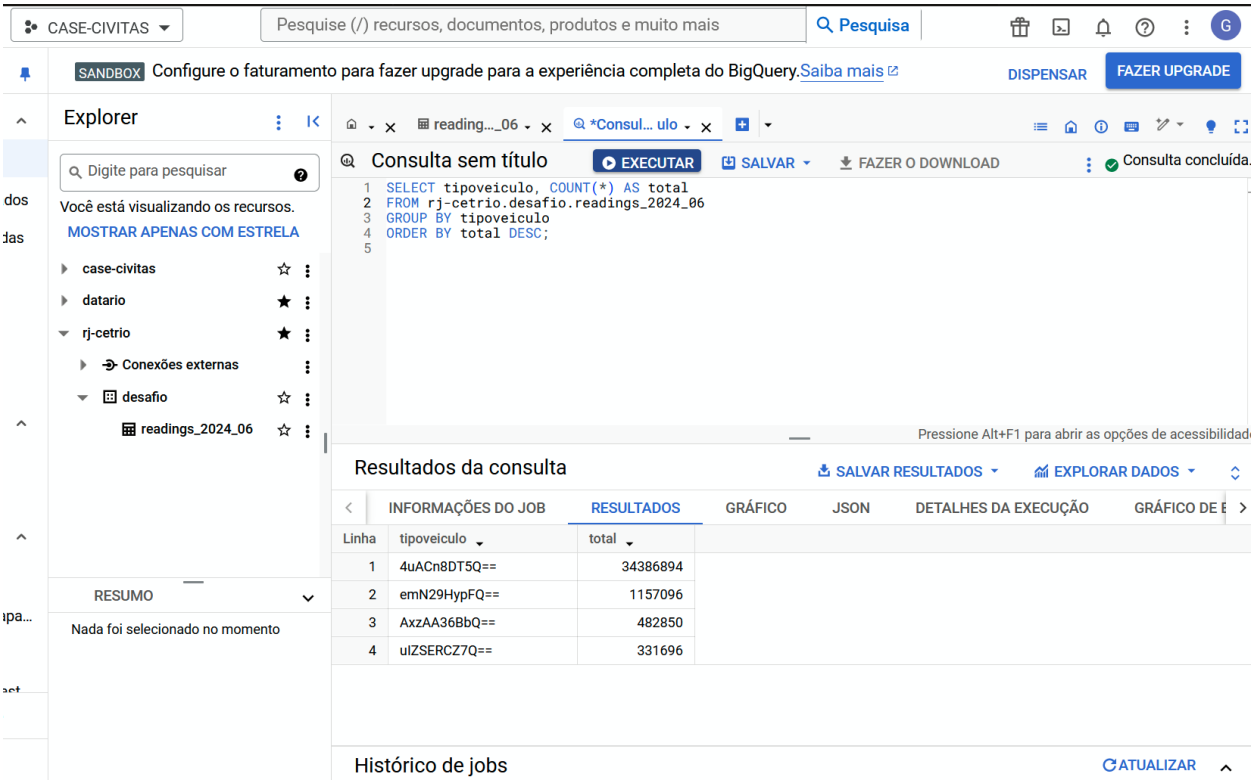
```
SELECT COUNT(*) AS total_registros
FROM rj-cetrio.desafio.readings_2024_06;
```



Aqui podemos continuar explorando os dados por coluna. Por exemplo, fazer uma contagem por tipo de veículo para termos um melhor entendimento dos dados apresentados.

1.3. Contagem de registros por tipo de veículo

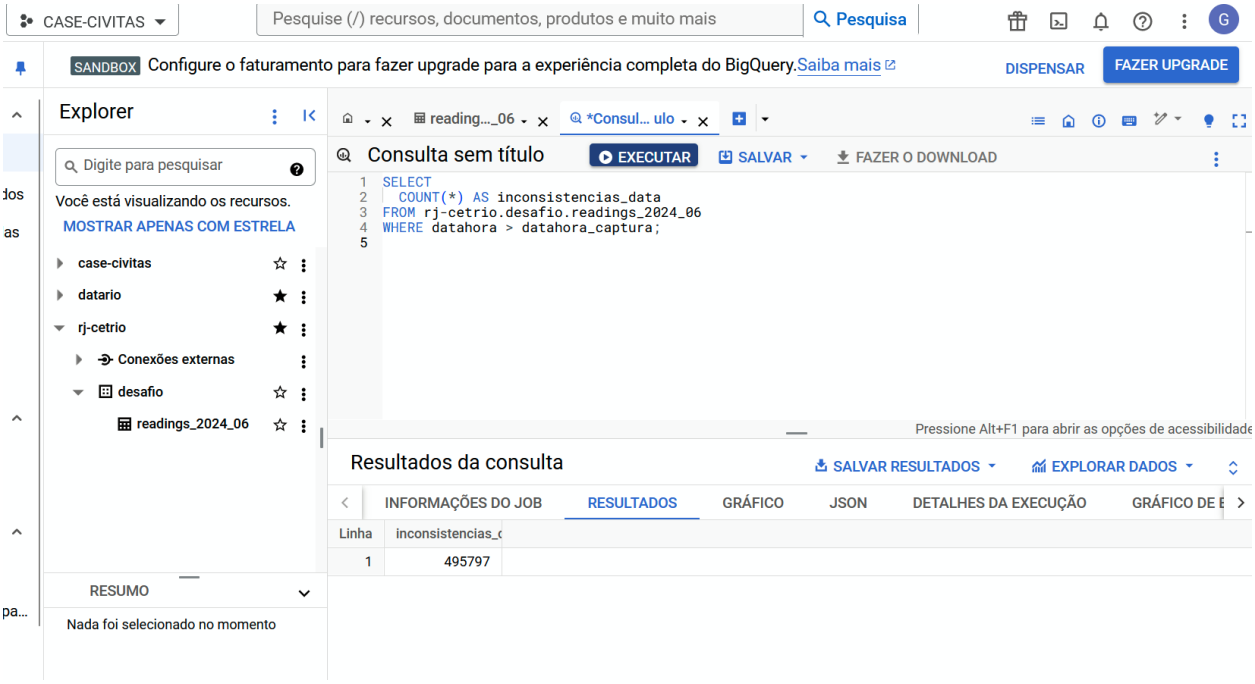
```
SELECT tipoveiculo, COUNT(*) AS total
FROM rj-cetrio.desafio.readings_2024_06
GROUP BY tipoveiculo
ORDER BY total DESC;
```



Outra possibilidade de análise para identificar inconsistências nos dados é comparar a data_hora com a data de captura. Através da query abaixo, podemos identificar o número de casos a serem tratados nesta situação. Neste caso específico, por se tratar de 495 mil casos, não recomendo a exclusão do modelo. Possivelmente exista um cenário onde a data_hora possa ser maior que a data de captura, que eu não esteja considerando.

1.4. Verificação de inconsistências nas datas

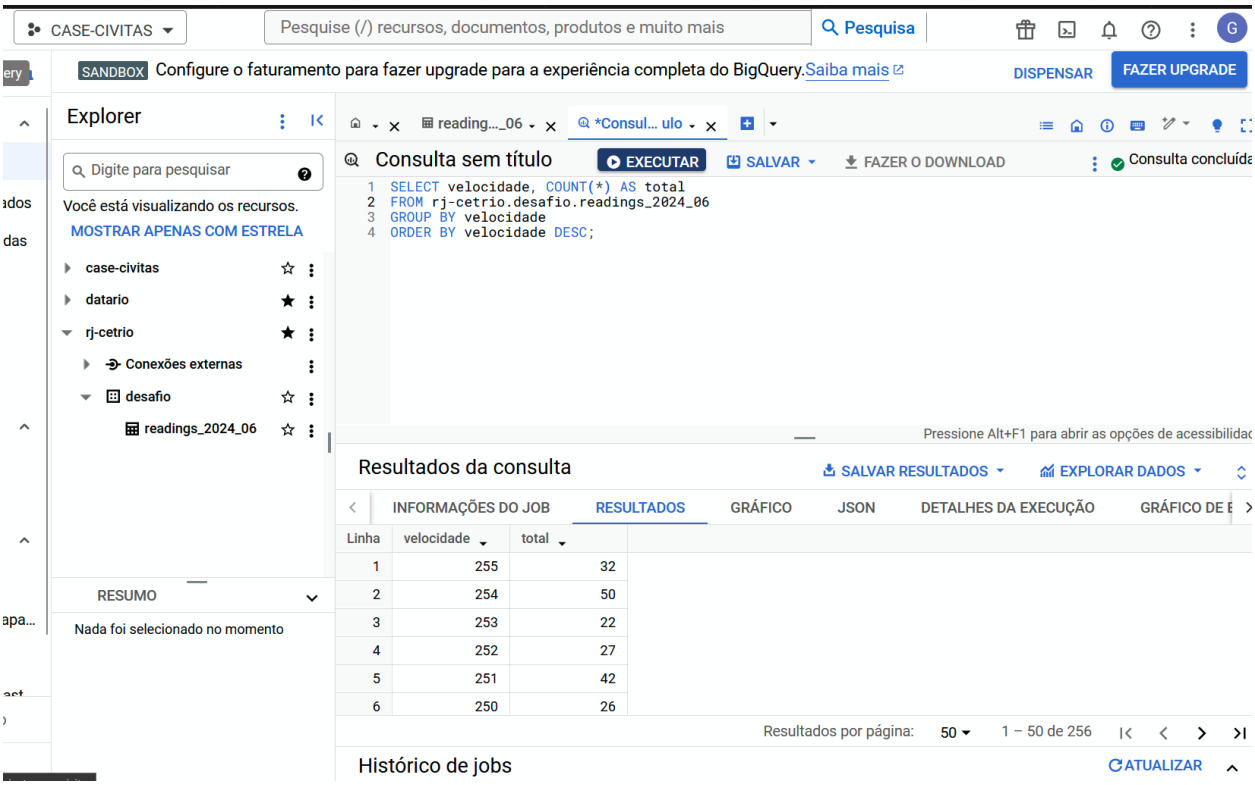
```
SELECT
  COUNT(*) AS inconsistencias_data
FROM rj-cetrio.desafio.readings_2024_06
WHERE datahora > datahora_captura;
```



Além disso, podemos buscar variáveis numéricas e executar análises de distribuição para identificar padrões. Pessoalmente, esse tipo de análise eu costumo fazer em Python para aproveitar as bibliotecas de plotagem de dados, mas em SQL se faz da seguinte forma.

1.5. Distribuição da velocidade dos veículos

```
SELECT velocidade, COUNT(*) AS total
FROM rj-cetrio.desafio.readings_2024_06
GROUP BY velocidade
ORDER BY velocidade DESC;
```

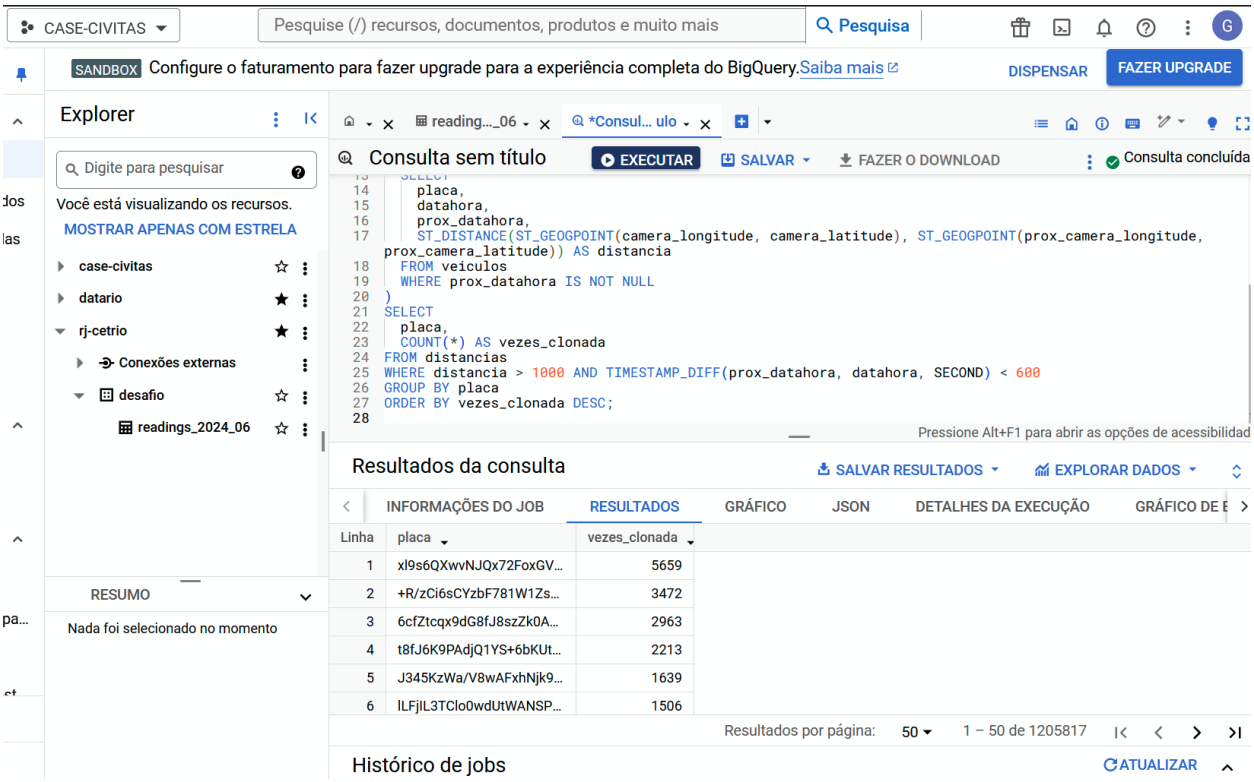


Identificação de Placas Clonadas

Para identificar possíveis placas clonadas, devemos procurar por situações onde a mesma placa foi registrada em locais diferentes (latitude e longitude) com um intervalo de tempo muito curto, o que seria impossível fisicamente para o mesmo veículo.

```
WITH veiculos AS (
  SELECT
    placa,
    datahora,
    camera_latitude,
    camera_longitude,
    LEAD(datahora) OVER (PARTITION BY placa ORDER BY datahor
```

```
a) AS prox_datahora,
    LEAD(camera_latitude) OVER (PARTITION BY placa ORDER BY d
atahora) AS prox_camera_latitude,
    LEAD(camera_longitude) OVER (PARTITION BY placa ORDER BY
datahora) AS prox_camera_longitude
  FROM rj-cetrio.desafio.readings_2024_06
),
distancias AS (
  SELECT
    placa,
    datahora,
    prox_datahora,
    ST_DISTANCE(ST_GEOGPOINT(camera_longitude, camera_latitud
e), ST_GEOGPOINT(prox_camera_longitude, prox_camera_latitud
e)) AS distancia
  FROM veiculos
  WHERE prox_datahora IS NOT NULL
)
SELECT
  placa,
  COUNT(*) AS vezes_clonada
FROM distancias
WHERE distancia > 1000 AND TIMESTAMP_DIFF(prox_datahora, data
hora, SECOND) < 600
GROUP BY placa
ORDER BY vezes_clonada DESC;
```



Nesta abordagem, podemos definir os gatilhos que determinam um comportamento suspeito. Podemos aumentar a distância ou o intervalo de tempo em que uma placa foi vista pelo radar.