



escola
britânica de
artes criativas
& tecnologia

Profissão: Analista de dados



BIG DATA II – PROCESSAMENTO



GUIA DA AULA 3



Implemente o particionamento



Acompanhe aqui
os temas que
serão tratados
na videoaula

Introdução

Dados

AWS S3

AWS Athena



Introdução

Para observar os benefícios que o particionamento trás para o armazenamento de grandes volumes de dados, vamos explorar as técnicas de particionamento na *cloud* da AWS, utilizando os serviços AWS S3 e AWS Athena, e o seu efeito combinado com a orientação a coluna através do `Apache Parquet`.



Dados

Vamos criar a coluna `reference_date` a partir da coluna `Date` no formato YYYY-MM-DD e entender se ela será uma boa coluna de partição.

```
In [ ]: import pandas as pd

        filename = './crime'

        df = pd.read_csv(f'./{filename}.csv')
```

```
In [ ]: df.head()
```



```

In [ ]: from datetime import datetime

df['reference_date'] = df['Date'].apply(
    lambda date:
        datetime.strptime(date.split(sep=' ')[0], '%m/%d/%Y') .
        \ strftime('%Y-%m-%d')
)
  
```

```

In [ ]: df.tail()
  
```

Uma coluna que separa os dados em grupos bem distribuídos é uma boa candidata a uma coluna de partição. Vamos contar as ocorrências de crimes (logo, linhas) em cada um dos dias da coluna `reference_date` recém criada.



```

In [ ]: agg_df = pd.DataFrame(
          df['reference_date'].value_counts(
            )
        ).sort_index().reset_index()
agg_df = agg_df.rename(columns={'reference_date':
                                'amount'}) agg_df = agg_df.rename(columns={'index':

```

```

In [ ]: agg_df.tail()

```

```

In [ ]: import seaborn as sns

with sns.axes_style('whitegrid'):

    chart = sns.barplot(x='reference_date', y='amount',
                        data=agg_df) chart.set(xticklabels=[])
    chart.set(
        title='Frequency of Crime per Day (Chicago,
              2014)', xlabel='Date (values ommited)',
        ylabel='Absolute Frequency'
    );
    chart.figure.set_size_inches(w=40/2.54, h=15/2.54)

```



Observa-se que a coluna `reference_date` de fato divide os dados em grupos equilibrados. Sendo assim, vamos salvar o `Dataframe` Pandas em arquivos comprimidos no formato `Parquet` particionados pela coluna `reference_date`.

```

In [ ]: df.to_parquet(
        './crime',
        engine='pyarrow',
        compression='gzip'
        ,
        partition_cols='reference_date'
        )
  
```



Vamos também salvar o `Dataframe` Pandas no formato `csv` para garantir que os arquivos de ambas as abordagens possuem a coluna `reference_date`.

```
In [ ]: df.to_csv('./crime_enriched.csv', sep=',', index=False)
```



AWS S3

Na AWS, vamos criar os recursos tanto para o arquivo no formato `csv` quanto para os arquivos no formato `parquet`.

- CSV

Vamos criar os recursos na AWS:

1. `Bucket` no `AWS S3` para armazenar o arquivo.

- Parquet

Vamos começar criando os recursos na AWS:

1. `Bucket` no `AWS S3` para armazenar os arquivos e suas partições;
2. Usuário no `AWS S3` para fazer o *upload* dos arquivos e suas partições.



Então, vamos inserir as credenciais no Python.

```
In [ ]: from getpass import getpass

aws_access_key_id = getpass()
```

```
In [ ]: from getpass import getpass

aws_secret_access_key = getpass()
```

E instalar o pacote Boto3, o SDK Python da AWS.

```
In [ ]: !pip install boto3
```



Por fim, vamos criar o nosso cliente e fazer o *upload* das partições.

```

In [ ]: import boto3

client =
    boto3.client( 's3',
    aws_access_key_id=aws_access_key_id,
    aws_secret_access_key=aws_secret_access_ke
    y
    )
  
```



In []:

```

import os
BUCKET =

'modulo-42-ebac-parquet' i = 0

for root, dirs, files in
    os.walk('./crime'): elapsed =
        f'{round(100*i/365, 2)} %' print(elapsed)
    for file in files:
        path = os.path.join(root, file)
        bucket_path =
            '/'.join(path.split(sep='/')[2:])
        client.upload_file(path, BUCKET, bucket_path)
    i = i + 1

```



AWS Athena

Na AWS, vamos criar os recursos tanto para o arquivo no formato `csv` quanto para os arquivos no formato `parquet`.

- CSV

Vamos criar os recursos na AWS:

1. Tabela no `AWS Athena` Apontando para o arquivo.



```

CREATE EXTERNAL TABLE `crime_csv` (
  `index` bigint,
  `id` string,
  `case number` string,
  `date` string,
  `block` string,
  `iucr` string,
  `primary type` string,
  `description` string,
  `location` description string,
  `arrest` string,
  `domestic` string,
  `beat` string,
  `district` string,
  `ward` string,
  `community area` string,
  `fbi code` string,
  `latitude` string,
  `longitude` string,
  `reference_date` string)

```



```

ROW FORMAT SERDE
  'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
  'separatorChar' = ',',
  'quoteChar' = '\"',
  'escapeChar' = '\\'
)
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
,
LOCATION
  's3://modulo-42-ebac-csv/'
TBLPROPERTIES (
  "skip.header.line.count"="1")

```



- Parquet

Vamos criar os recursos na AWS:

1. Tabela no **AWS Athena** apontando para os arquivos e suas partições.
2. Carregar partições.

```

CREATE EXTERNAL TABLE `crime_parquet` (
  `index` bigint,
  `id` bigint,
  `case` number` string,
  `date` string,
  `block` string,
  `iucr` string,
  `primary type` string,
  `description` string,
  `location` description` string,
  `arrest` boolean,
  `domestic` boolean,

```



```

    `beat` bigint,
    `district` bigint,
    `ward` double,
    `community area` double,
    `fbi code` string,
    `latitude` double,
    `longitude` double)
PARTITIONED BY (
  `reference_date` string)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION
  's3://modulo-42-ebac-parquet/'

MSCK REPAIR TABLE `crime_parquet`;

```



Por fim, vamos executar um conjunto de consultas SQL em ambas as tabelas e observar a quantidade de dados escaneados.

- Efeito da orientação à coluna:

```

SELECT "location description", COUNT(1) as "amount"
FROM crime_csv
GROUP BY 1 ORDER
BY 2 DESC;

```

```

SELECT "location description", COUNT(1) as "amount"
FROM crime_parquet
GROUP BY 1
ORDER BY 2 DESC;

```



A consulta escaneou 47.34 MB para a tabela `crime_csv`, que é o mesmo tamanho do arquivo, logo um *full scan*. Já para a tabela `crime_parquet`, a consulta escaneou 0.44 MB. Ou seja, a tabela com o dado orientado à coluna escaneou 108 vezes menos dados para a consulta SQL que seu par em `csv`.

- Efeito do particionamento:

```

SELECT *
FROM crime_csv
WHERE CAST(reference_date as DATE) BETWEEN DATE '2014-12-01' and
DATE '2014-12-31'
  
```

```

SELECT *
FROM crime_parquet
WHERE CAST(reference_date as DATE) BETWEEN DATE '2014-12-01' and
DATE '2014-12-31'
  
```



A consulta escaneou 47.34 MB para a tabela `crime_csv`, que é o mesmo tamanho do arquivo, logo um *full scan*. Já para a tabela `crime_parquet`, a consulta escaneou 1.00 MB. Ou seja, a tabela com o dado particionado escaneou 47.34 vezes menos dados para a consulta SQL que seu par em csv.

- Efeito da orientação a coluna e do particionamento:

```

SELECT "location description" , COUNT(1) as "amount"
FROM crime_csv
WHERE CAST(reference_date as DATE) BETWEEN DATE '2014-12-01' and
DATE '2014-12-31'
GROUP BY 1
ORDER BY 2 DESC
  
```



```

SELECT "location description", COUNT(1) as "amount"
FROM crime_parquet
WHERE CAST(reference_date as DATE) BETWEEN DATE '2014-12-01' and
DATE '2014-12-31'
GROUP BY 1
ORDER BY 2 DESC

```

A consulta escaneou 47.34 MB para a tabela `crime.csv`, que é o mesmo tamanho do arquivo, logo um *full scan*. Já para a tabela `crime_parquet`, a consulta escaneou 0.04 MB. Ou seja, a tabela com o dado particionado e orientado à coluna escaneou 1183.5 vezes menos dados para a consulta SQL que seu par em `csv`.

