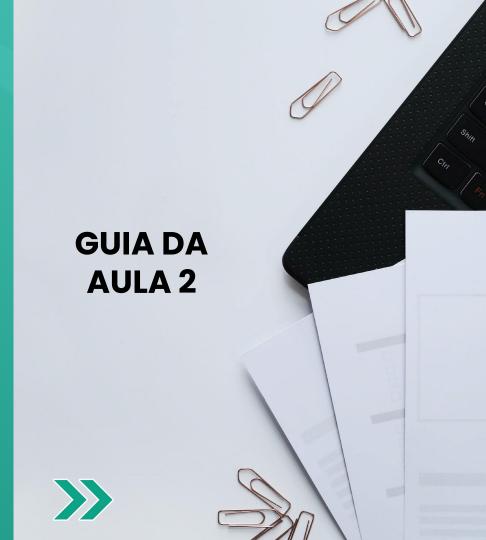


Profissão: Analista de dados





BIG DATA II - PROCESSAMENTO







Faça orientação à coluna

- Introdução
- Apache Parquet
- Apache Arrow



Acompanhe aqui os temas que serão tratados na videoaula







Introdução

Para observar os benefícios que a orientação à coluna traz para o armazenamento de grandes volumes de dados, vamos explorar duas tecnologias orientadas a colunas: o formato de arquivo Apache Parquet (disco) e a estrutura de dados Apache Arrow memória). Vamos também compará-las com seus pares orientados à linha, como arquivos do tipo csv e o pacote Python Pandas.

Como exemplo, vamos utilizar os dados de crimes da cidade de Chicago, Estados Unidos da América, em 2014. Os dados estão armazenados em um arquivo no formato csv de aproximadamente 50MB e foram extraídos do Kaggle.





```
!wget https://raw.githubusercontent.com/\
andre-marcos-perez/ebac-course-utils/\
main/dataset/crime.csv -q -0
crime.csv
```

Vamos criar um DataFrame Pandas com os dados.

```
In []:
    import pandas as pd
    filename = './crime'
    df = pd.read_csv(f'./{filename}.csv')

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





Vamos então conferir alguns metadados do DataFrame.

```
In []: df.shape
In []: df.info()
```

Por fim, vamos realizar uma agregação para futura comparação. Nela, vamos contar a frequência de ocorrência dos crimes agrupados localidades da cidade (coluna Location Description)

```
In [ ]: agg_df = df['Location Description'].value_counts()
In [ ]: agg_df
```





Apache Parquet

O Apache Parquet é o formato de arquivo orientado à coluna mais utilizado no ecossistema de *big data*. Sua documentação pode ser encontrada no *link* https://parquet.apache.org/.

Entre suas funcionalidades, podemos destacar:

- indexação por coluna (processamento);
- tipagem por coluna (processamento e armazenamento);
- compressão por coluna (armazenamento).

A interoperabilidade com o pacote Python Pandas é alcançada atrás do uso de estruturas de dados orientadas à coluna, como o Apache Arrow.





Exemplos:

• Salvar um Pandas DataFrame para um arquivo Apache Parquet:

```
In [ ]:
    df.to_parquet('./crime.parquet', engine='pyarrow')
```

Salvar um Pandas DataFrame para um arquivo Apache Parquet comprimido:

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





Vamos utilizar o método getsize do pacote nativo os para estimar o tamanho dos arquivos na memória persistente (ROM/SSD):

```
In []:
    import os
    extensions = ['csv', 'parquet', 'parquet.gzip']

    for extension in extensions:
        size =
        os.path.getsize(f'{filename}.{extension}')
        size_mb = round(size / 1024 / 1024, 2)

        print(f'{extension}: {size_mb} MB')
```





Apache Arrow

O Apache Arrow é uma estrutura de dados orientada à coluna muito utilizada no ecossistema de big data. Sua documentação pode ser encontrada no link https://arrow.apache.org/. É equivalente ao Apache Parquet, mas em memória, como listas, dicionários e objetos Python. O pacote Python PyArrow permite a criação e manipulação das estruturas de dados do Apache Arrow.

Para encontrar a documentação do PyArrow acesse o link https://arrow.apache.org/docs/python/install.html.

```
In [ ]:
!pip install pyarrow==7.0.0
```





O PyArrow trabalha com uma estrutura de dados orientada a coluna conhecida como table (tabela), similar aos DataFrames Pandas.

```
from pyarrow import
         csv import pyarrow as
         pa import pandas as pd
         filename = './crime'
         table =
         csv.read csv(f'{filename}.csv') df =
         pd.read csv(f'./{filename}.csv')
In [ ]:
         table.shape
         table
```





A similaridade com o Pandas fica evidente quando realizamos operações de agregação.

Pandas

```
In []: agg_df = df['Location Description'].value_counts()
In []: agg_df
```

PyArrow





Vamos utilizar o método getsizeof do pacote nativo sys para estimar o tamanho dos objetos na memória de trabalho (RAM):

```
In []:
    import sys

    objects = [{'pandas': df}, {'pyarrow': table}]

    for obj_dict in objects:
        for id, obj in obj_dict.items():

        size = sys.getsizeof(obj)
        size_mb = round(size / 1024 / 1024, 2)

        print(f'{id}: {size_mb} MB')
```

Vemos que o objeto gerado pelo PyArrow (table) é aproximadamente 3 vezes menor que o objeto (dataframe) utilizado no Pandas.