Pipeline ETL usando Python

Este código busca exemplificar um pipeline ETL (Extract, Transform, Load) usando Python. Mantive o formato .ipynb (notebook) para facilitar o acompanhamento da rotina por iniciantes. Ele está dividido em três etapas:

1) Extração de múltiplos dados via webscraping; 2) Transformação dos dados; 3) Carregar os dados já estrurados para um database;

Os principais comandos automatizados:

- Checa os arquivos na pasta referência e seleciona as datas que já possuem dados;
- Baixa as datas que não possuem dados (formato txt mas zipados);
- Extrair os arquivos zip e depois os deleta;
- Carrega os arquivos txt, transforma em parquet e deleta os txt;
- Corrige o formatos de dados (str, float e datetime) e exclui colunas redundantes;
- Salva os arquivos e faz um teste de integridade dos dados entre tabelas;
- Faz a primeira carga num banco de dados SQL local;
- Faz uma segunda carga (incremental) no banco de ados criado.

Contexto:

ETL é a sigla para Extract, Transform, Load, que em português pode ser traduzido como Extração, Transformação e Carga. ETL é um processo fundamental na engenharia de dados e amplamente utilizado em ambientes de Business Intelligence (BI) e Data Warehousing para consolidar dados de várias fontes em um único repositório, facilitando a análise e a tomada de decisões.

---

1. Extração

1.1. Bibliotecas

```
import os
import wget
import httplib2
import shutil
import zipfile
import pyarrow
import numpy as np
import pandas as pd
import datetime
import sqlite3
import mplfinance as mpf
import plotly.graph_objects as go
```

1.2. Fontes de dados (data sources)

As fontes de dados são o ponto de partida de um pipeline ETL. Estas podem ser diversas e incluir bancos de dados, armazenamento em nuvem, logs de aplicações, APIs externas e mais.

Neste estudo iremos extrair:

- dados de negociações diárias da B3. Esses dados "tick by tick" mostram cada negociação realizada, incluindo quantidade, preço e agentes envolvidos. A B3 disponibiliza gratuitamente esses dados histórico dos últimos 20 dias úteis.

https://www.b3.com.br/pt_br/market-data-e-indices/servicos-de-dados/market-data/cotacoes/cotacoes/

- Glossário sobre as informações contidas nesses dados:

https://www.b3.com.br/data/files/14/42/28/31/FEC4A8103234E0A8AC094EA8/Glossario_NegociosListados_PT.pdf

- dados dos participantes das negociações B3

https://sistemaswebb3-listados.b3.com.br/participantsPage/

1.3. Extração

A extração é a primeira etapa do processo ETL. Ela envolve a obtenção de dados das diversas fontes e armazena em um local centralizado para processamento. Os métodos de extração variam, incluindo processamento em lote (intervalos programados) ou streaming em tempo real

Função download_b3_tick_by_tick

1. Checar quais os arquivos já foram baixados;
2. Fazer o filtro das datas do mês
3. Checar se a data existe no site
4. Se a data existe, baixar o arquivo .zip
5. Deletar a pasta ".cache" que é criada na execução desta função.

```
def download_b3_tick_by_tick(start=datetime.datetime(2024,1,1)):

    folder_ref = os.getcwd()
    files = os.listdir(folder_ref)
    downloaded_files_zip = [i for i in files if i.endswith('.zip')]
    downloaded_dates = [i.replace('B3_tick_', '').replace('.zip', '')
for i in downloaded_files_zip]
    downloaded_dates = pd.to_datetime(downloaded_dates)

    date_range = pd.date_range(start.date(),
datetime.datetime.now().date())
    date_range_to_download =
date_range[~date_range.isin(downloaded_dates)]

    for i in date_range_to_download:
```

```
        try:
            market_date = i.strftime('%Y-%m-%d')
            url =
f'https://arquivos.b3.com.br/apinegocios/tickercsv/{market_date}'
            h = httplib2.Http('.cache')
            response, content =
h.request(f'https://arquivos.b3.com.br/apinegocios/tickercsv/{market_d
ate}', headers={'Connection':'keep-alive'})
            if len(content)>0:
                response = wget.download(url,
str('B3_tick_'+market_date+'.zip'))

        except:
            None

    shutil.rmtree('.cache')
```

Função unzip_files_delete

1. Checar os arquivos dentro da pasta;
2. Para cada arquivo da pasta, se terminar em ".zip", extrair;
3. Para cada arquivo da pasta, se terminar em ".zip", deletar;

```
def unzip_files_delete():

    folder_ref = os.getcwd()
    files = os.listdir(folder_ref)

    for i in files:
        if i.endswith('.zip'):
            with zipfile.ZipFile(i, "r") as zip_ref:
                zip_ref.extractall(folder_ref)

    for i in files:
        if i.endswith('.zip'):
            os.remove(i)
```

Ler o arquivo csv dos participantes B3 (deve estar na pasta)

```
codigo_participantes_B3 = pd.read_csv("participantDownload.csv",
sep=';')
```

Executar as duas funções criadas

```
download_b3_tick_by_tick(start=datetime.datetime(2024,7,1))
unzip_files_delete()
```

2. Transformação

Os dados brutos obtidos na extração raramente são adequados para análise. A fase de transformação envolve a limpeza, enriquecimento e estruturação dos dados para torná-los utilizáveis para o propósito pretendido. Isso pode incluir tarefas como validação de dados, conversão de tipos de dados, desduplicação e agregação de dados de diferentes fontes.

2.1. Transformar os dados txt em parquet (economia de espaço no drive > 70%)

Função convert_to_parquet:

1. Listar os arquivos txt (dados negociações diárias) na pasta;
2. Carregar cada arquivo txt como dataframe;
3. Exportar no formato parquet, otimizando o nome do arquivo para formato Ano/Mês/Dia;
4. Deletar os arquivos txt;

```
folder_ref = os.getcwd()
files = os.listdir(folder_ref)
files_txt = [i for i in files if i.endswith('.txt')]

for i in files_txt:
    df = pd.read_csv(i, sep=";")
    df.to_parquet('raw_'+str(i[6:10] + '_' + i[3:5] + '_' + i[0:2] +
'_' + 'tick_B3.parquet'))

for i in files_txt:
    os.remove(i)
```

2.2. Exploração dos "problemas" nos dados

Esta etapa usualmente não entra no pipeline, já que o engenheiro de dados já explorou os potenciais problemas dos dados para fazer a limpeza, enriquecimento e estruturação dos dados.

Mas para melhor entendimento, vamos começar lendo um dos arquivos de negociações diárias para conhecer um pouco sobre os dados

2.2.1. Dados de negociações diárias

```
df_tick_dia1 = pd.read_parquet('raw_2024_07_04_tick_B3.parquet',
engine='pyarrow')
df_tick_dia1
```

```
        DataReferencia CodigoInstrumento  AcaoAtualizacao PrecoNegocio
\
0           2024-07-04            TF583R                0       10,000

1           2024-07-04            WSPU24                0     5598,250

2           2024-07-04            BGIN24                0      235,050
```

```
3           2024-07-04            CCMN24                  0       56,300
4           2024-07-04            CCMN24                  0       56,140
...                ...               ...                ...          ...
7867012     2024-07-04            INDQ24                  0   127510,000
7867013     2024-07-04            INDQ24                  0   127510,000
7867014     2024-07-04            INDQ24                  0   127510,000
7867015     2024-07-04            INDQ24                  0   127510,000
7867016     2024-07-04            INDQ24                  0   127510,000

         QuantidadeNegociada  HoraFechamento
CodigoIdentificadorNegocio  \
0                      10000        30136837
10
1                          1        90000007
10
2                          2        90000463
50
3                          1        90000517
230
4                          1        90000547
240
...                      ...             ...                          .
..
7867012                    5       183118570
94990
7867013                    5       183118570
95000
7867014                    5       183118570
95010
7867015                    5       183118570
95020
7867016                    5       183118570
95030

         TipoSessaoPregao DataNegocio  CodigoParticipanteComprador  \
0                       1  2024-07-04                        100.0
1                       1  2024-07-04                          3.0
2                       1  2024-07-04                          NaN
3                       1  2024-07-04                          NaN
4                       1  2024-07-04                          NaN
...                   ...         ...                          ...
```

```
7867012                   1  2024-07-04                              88.0
7867013                   1  2024-07-04                              88.0
7867014                   1  2024-07-04                              83.0
7867015                   1  2024-07-04                             308.0
7867016                   1  2024-07-04                             308.0

         CodigoParticipanteVendedor
0                             100.0
1                               3.0
2                               NaN
3                               NaN
4                               NaN
...                             ...
7867012                       107.0
7867013                       107.0
7867014                       107.0
7867015                       107.0
7867016                        93.0

[7867017 rows x 11 columns]
```

Note que temos quase 8 milhões de linhas e 11 colunas! Explorar algumas características e potenciais problemas nos dados

2.2.1.1. Problema: tipo dos dados no dataframe (Dtype)

```
df_tick_dia1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7867017 entries, 0 to 7867016
Data columns (total 11 columns):
 #   Column                       Dtype  
---  ------                       -----  
 0   DataReferencia               object 
 1   CodigoInstrumento            object 
 2   AcaoAtualizacao              int64  
 3   PrecoNegocio                 object 
 4   QuantidadeNegociada          int64  
 5   HoraFechamento               int64  
 6   CodigoIdentificadorNegocio   int64  
 7   TipoSessaoPregao             int64  
 8   DataNegocio                  object 
 9   CodigoParticipanteComprador  float64
 10  CodigoParticipanteVendedor   float64
dtypes: float64(2), int64(5), object(4)
memory usage: 660.2+ MB
```

2.2.1.2. Problema: coluna "PrecoNegocio" com divisor decimal como "vírgula"

```
df_tick_dia1.head()
```

```
  DataReferencia CodigoInstrumento  AcaoAtualizacao PrecoNegocio  \
0     2024-07-04            TF583R                0       10,000
1     2024-07-04            WSPU24                0     5598,250
2     2024-07-04            BGIN24                0      235,050
3     2024-07-04            CCMN24                0       56,300
4     2024-07-04            CCMN24                0       56,140

   QuantidadeNegociada  HoraFechamento  CodigoIdentificadorNegocio  \
0                10000        30136837                          10
1                    1        90000007                          10
2                    2        90000463                          50
3                    1        90000517                         230
4                    1        90000547                         240

   TipoSessaoPregao DataNegocio  CodigoParticipanteComprador  \
0                 1  2024-07-04                        100.0
1                 1  2024-07-04                          3.0
2                 1  2024-07-04                          NaN
3                 1  2024-07-04                          NaN
4                 1  2024-07-04                          NaN

   CodigoParticipanteVendedor
0                       100.0
1                         3.0
2                         NaN
3                         NaN
4                         NaN
```

2.2.1.3. Problema: colunas com informações redundantes ou pouco úteis

A coluna "AcaoAtualizacao" informa se o negócio foi cancelado. O valor default é 0, sendo 2 para negócio cancelado. Veja no próximo output que são raros os negócios cancelados, mas de qualquer forma excluiremos as linhas com AcaoAtualizacao=2 na etapa de limpeza

```
df_tick_dia1[df_tick_dia1.AcaoAtualizacao!=0]

        DataReferencia CodigoInstrumento  AcaoAtualizacao PrecoNegocio
\
2250003     2024-07-04            DI1F26                2       11,340

3094794     2024-07-04          IBOVG130                2      255,000

4972047     2024-07-04            DI1F28                2       11,890

5994595     2024-07-04         DIFF26F27                2       12,130

6004891     2024-07-04         DIFF25F26                2       11,690

7200710     2024-07-04          SBSPT817                2        3,570
```

```
7756636     2024-07-04          RDORA370                  2        0,750


         QuantidadeNegociada  HoraFechamento
CodigoIdentificadorNegocio  \
2250003                 2790       104952817
102410
3094794                  320       112412370
10
4972047                 1000       132240687
80090
5994595                 1900       144042400
10
6004891                  685       144146357
10
7200710                160000       161838593
90
7756636                820000       172137407
10

         TipoSessaoPregao DataNegocio  CodigoParticipanteComprador  \
2250003                 1  2024-07-04                          NaN
3094794                 1  2024-07-04                          NaN
4972047                 1  2024-07-04                          NaN
5994595                 1  2024-07-04                          NaN
6004891                 1  2024-07-04                          NaN
7200710                 1  2024-07-04                          NaN
7756636                 1  2024-07-04                          NaN

         CodigoParticipanteVendedor
2250003                         NaN
3094794                         NaN
4972047                         NaN
5994595                         NaN
6004891                         NaN
7200710                         NaN
7756636                         NaN
```

Observe abaixo que as colunas "DataReferencia" e "DataNegocio" diferem mt pouco, apenas para ativos de commodities e em poucos casos. Remover essa redundância mantendo apenas "DataReferencia". Outras colunas como "CodigoIdentificadorNegocio" e "TipoSessaoPregao" não agregarão informações úteis ao nossas análises e serão removidas

```
print('"DataReferencia" difere da "DataNegocio" em apenas',
(df_tick_dia1.DataReferencia != df_tick_dia1.DataNegocio).sum(),
'linhas')
set(df_tick_dia1[df_tick_dia1.DataReferencia !=
df_tick_dia1.DataNegocio]['CodigoInstrumento'])
```

```
"DataReferencia" difere da "DataNegocio" em apenas 279 linhas

{'BGIF25',
 'BGIN24',
 'BGIQ24',
 'BGIU24',
 'BGIV24',
 'BGIX24',
 'BGIZ24',
 'CCMN24',
 'CCMU24',
 'CCMX24',
 'ETHN24',
 'FRP1',
 'ICFU24',
 'ICFZ24'}
```

2.2.1.4. Problema: coluna "HoraFechamento" em HHMMSSNNN

Dados como inteiros e assim, horários antes das 10h possui um número a menos. Os milisegundos também não trazem informações úteis para esse estudo

```
df_tick_dia1['HoraFechamento']

0           30136837
1           90000007
2           90000463
3           90000517
4           90000547
             ...
7867012    183118570
7867013    183118570
7867014    183118570
7867015    183118570
7867016    183118570
Name: HoraFechamento, Length: 7867017, dtype: int64
```

2.2.2. Dados dos participantes

Nesses dados, estamos buscando apenas as colunas "Nome" e "Código". O nome indica o participante B3 e o código faz referência aos dados de negociações (colunas "CodigoParticipanteComprador", "CodigoParticipanteVendedor")

Note que um mesmo agente tem vários códigos, inclusive dentro de uma mesma célula. Precisaremos apenas dos participantes com Perfil "PARTICIPANTE DE NEGOCIAÇÃO PLENO"

```
codigo_participantes_B3
```

```
                  CNPJ
Nome  \
0                  191                                  BANCO DO BRASIL
S/A
1                  191                                  BANCO DO BRASIL
S/A
2                  191                                  BANCO DO BRASIL
S/A
3                  191                                  BANCO DO BRASIL
S/A
4                  191                                  BANCO DO BRASIL
S/A
...                ...                                                .
..
28952  97711801000105  953 FICFIM CREDITO PRIVADO INVESTIMENTO NO
EXT...
28953  97837181000147                                          DEXCO
S.A
28954  97929213000134  WHG SISTEMA II FUNDO DE INVESTIMENTO
MULTIMERC...
28955  98102924000101                          DELTASUL UTILIDADES
LTDA
28956  99999999999999
ICECCP

                Site                   Perfil  \
0      www.bb.com.br  ADMINISTRADOR iMERCADO
1      www.bb.com.br      AGENTE DE CUSTÓDIA
2      www.bb.com.br      CUSTODIANTE BALCÃO
3      www.bb.com.br   EMITENTE DE GARANTIAS
4      www.bb.com.br            ESCRITURADOR
...              ...                     ...
28952            NaN         SEGMENTO BALCAO
28953            NaN         SEGMENTO BALCAO
28954            NaN         SEGMENTO BALCAO
28955            NaN         SEGMENTO BALCAO
28956            NaN         SEGMENTO BALCAO

                                                  Código
0                                             1124 - 2659
1      254 - 705 - 1123 - 1124 - 1126 - 1345 - 2659 -...
2                                              254 - 1123
3                                                   10001
4                                                    6098
...                                                   ...
28952                                          13753.00-7
28953                                          26320.40-3
28954                                          13565.00-6
28955                                          42836.40-7
28956                            77777777-7 - 88889.88-9
```

```
[28957 rows x 5 columns]
```

2.3. Limpeza e enriquecimento dos dados

2.3.1. Dados de negociações

Função change_dtypes:

1. Coluna "DataReferencia" para datetime;
2. Substituir NaN nas colunas "CodigoParticipanteComprador" e "CodigoParticipanteVendedor" para 0;
3. Colunas "CodigoParticipanteComprador" e "CodigoParticipanteVendedor" para int (remover casas decimais) e depois para str;
4. Coluna "PrecoNegocio" para float;

```
df_tick_dia1
```

```
        DataReferencia CodigoInstrumento  AcaoAtualizacao PrecoNegocio
\
0           2024-07-04             TF583R                0       10,000

1           2024-07-04             WSPU24                0     5598,250

2           2024-07-04             BGIN24                0      235,050

3           2024-07-04             CCMN24                0       56,300

4           2024-07-04             CCMN24                0       56,140

...                ...                ...              ...          ...

7867012     2024-07-04             INDQ24                0   127510,000

7867013     2024-07-04             INDQ24                0   127510,000

7867014     2024-07-04             INDQ24                0   127510,000

7867015     2024-07-04             INDQ24                0   127510,000

7867016     2024-07-04             INDQ24                0   127510,000


         QuantidadeNegociada  HoraFechamento
CodigoIdentificadorNegocio  \
0                      10000        30136837
10
1                          1        90000007
10
2                          2        90000463
50
```

```
3                            1        90000517
230
4                            1        90000547
240
...                        ...             ...                            .
..
7867012                      5       183118570
94990
7867013                      5       183118570
95000
7867014                      5       183118570
95010
7867015                      5       183118570
95020
7867016                      5       183118570
95030

         TipoSessaoPregao DataNegocio  CodigoParticipanteComprador  \
0                       1  2024-07-04                        100.0
1                       1  2024-07-04                          3.0
2                       1  2024-07-04                          NaN
3                       1  2024-07-04                          NaN
4                       1  2024-07-04                          NaN
...                   ...         ...                          ...
7867012                 1  2024-07-04                         88.0
7867013                 1  2024-07-04                         88.0
7867014                 1  2024-07-04                         83.0
7867015                 1  2024-07-04                        308.0
7867016                 1  2024-07-04                        308.0

         CodigoParticipanteVendedor
0                             100.0
1                               3.0
2                               NaN
3                               NaN
4                               NaN
...                             ...
7867012                       107.0
7867013                       107.0
7867014                       107.0
7867015                       107.0
7867016                        93.0

[7867017 rows x 11 columns]
```

```python
def change_dtypes(df):
    df['DataReferencia'] = pd.to_datetime(df['DataReferencia'])
    df[['CodigoParticipanteComprador','CodigoParticipanteVendedor']] =
df[['CodigoParticipanteComprador','CodigoParticipanteVendedor']].filln
a(0)
```

```
    df[['CodigoParticipanteComprador','CodigoParticipanteVendedor']] =
df[['CodigoParticipanteComprador','CodigoParticipanteVendedor']].astyp
e('int').astype('str')
    df['PrecoNegocio'] = df.PrecoNegocio.str.replace(",",
".").astype('float')
```

Função change_HoraFechamento:

1. Tranforma coluna "HoraFechamento" em str e completa com zero no inicio para que todos tenham 9 caracteres;
2. Remove os últimos 3 caracteres (milisegundos)
3. Tranforma em time (H:M:S)

```
def change_HoraFechamento(df):
    df['HoraFechamento'] =
df['HoraFechamento'].astype(str).str.zfill(9)
    df['HoraFechamento'] = df['HoraFechamento'].str[:-3]
    df['HoraFechamento'] = pd.to_datetime(df['HoraFechamento'],
format='%H%M%S').dt.time
```

Função remove_useless_data:

1. Filtrar linhas que representam negócios cancelados ("AcaoAtualizacao"=2);
2. Manter apenas as colunas desejadas:
   - "DataReferencia",
   - "CodigoInstrumento",
   - "PrecoNegocio",
   - "QuantidadeNegociada",
   - "HoraFechamento",
   - "CodigoParticipanteComprador",
   - "CodigoParticipanteVendedor"

```
def remove_useless_data(df):
    df = df[df.AcaoAtualizacao!=2]
    df = df[['DataReferencia', 'CodigoInstrumento', 'PrecoNegocio',
             'QuantidadeNegociada', 'HoraFechamento',
             'CodigoParticipanteComprador',
'CodigoParticipanteVendedor']]
    return df
```

Função cleaning_tick_data_parquet:

1. Carregar todos os arquivos parquet da pasta como dataframe;
2. Executar a função "change_dtypes";
3. Executar a função "change_HoraFechamento";
4. Executar a função "remove_useless_data" que retorna um novo dataframe;
5. Salva um novo arquivo parquet com todas as modificaçoes, renomeado sem o ínicio "raw_";

6. Deleta o arquivo parquet de dados brutos

```
def cleaning_tick_data_parquet():
    folder_ref = os.getcwd()
    files = os.listdir(folder_ref)
    files_parquet = [i for i in files if i.endswith('.parquet') &
i.startswith('raw_')]

    for i in files_parquet:
        df = pd.read_parquet(i, engine='pyarrow')
        change_dtypes(df)
        change_HoraFechamento(df)
        cleaned_df = remove_useless_data(df)
        cleaned_df.to_parquet(i[4:])
        os.remove(i)

cleaning_tick_data_parquet()
```

2.3.2. Dados dos participantes

Função cleaning_participantes:

1. Filtrar "PARTICIPANTE DE NEGOCIAÇÃO PLENO" na coluna "Perfil";
2. Manter colunas "Nome" e "Código";
3. Manter apenas o primeiro código do participante (poucos mantêm dois códigos que estão separados por -);
4. Incluir última linha com Nome = "NÃO IDENTIFICADO" e Código = '0'
5. Renomear coluna "Código" para "Participante_index"
6. Resetar o índice

```
def cleaning_participantes(df):
    df = df[df.Perfil=='PARTICIPANTE DE NEGOCIAÇÃO PLENO']
    df = df[['Nome', 'Código']]
    cleaned_index = codigo_participantes_B3['Código'].str.split(' -',
n=1, expand=True)[0]
    df['Código'] = cleaned_index
    df.loc[len(df)] = ["NAO IDENTIFICADO", '0']
    df = df.rename(columns={"Código": "Participante_index"})
    df = df.reset_index(drop=True)
    df.to_parquet('cleaned_participantes_index.parquet')

cleaning_participantes(codigo_participantes_B3)
```

2.4. Testes de integridade, validação e agregação entre os databases

É importante garantir que os dados estejam realmente prontos para serem carregados. Esses testes são críticos para garantir que os dados que serão carregados estejam disponíveis para análise de maneira eficiente e confiável. Por exemplo, vamos testar se um conjunto de dados de negociação diária conseguem se relacionar com o conjunto de dados dos participantes B3

```
participantes_B3 =
pd.read_parquet('cleaned_participantes_index.parquet',
engine='pyarrow')
df_tick_dia1 = pd.read_parquet('2024_07_04_tick_B3.parquet',
engine='pyarrow')

merged_df = df_tick_dia1[['DataReferencia',
                          'CodigoInstrumento',

'CodigoParticipanteComprador']].merge(participantes_B3,

how='left',

left_on='CodigoParticipanteComprador',

right_on='Participante_index')

merged_df
```

```
        DataReferencia CodigoInstrumento
CodigoParticipanteComprador  \
0           2024-07-04            TF583R                            100

1           2024-07-04            WSPU24                              3

2           2024-07-04            BGIN24                              0

3           2024-07-04            CCMN24                              0

4           2024-07-04            CCMN24                              0

...                ...               ...                            ...

7867005     2024-07-04            INDQ24                             88

7867006     2024-07-04            INDQ24                             88

7867007     2024-07-04            INDQ24                             83

7867008     2024-07-04            INDQ24                            308

7867009     2024-07-04            INDQ24                            308


                                                      Nome
Participante_index
0                                                      NaN
NaN
1                                XP INVESTIMENTOS CCTVM S/A
3
2                                          NAO IDENTIFICADO
```

```
0
3                                          NAO IDENTIFICADO
0
4                                          NAO IDENTIFICADO
0
...                                                      ...
...
7867005  CM CAPITAL MARKETS CORRETORA DE CÂMBIO  TÍTULO...
88
7867006  CM CAPITAL MARKETS CORRETORA DE CÂMBIO  TÍTULO...
88
7867007                                    MASTER S/A CCTVM
83
7867008          CLEAR CORRETORA - GRUPO XP E VALORES MOBI
308
7867009          CLEAR CORRETORA - GRUPO XP E VALORES MOBI
308

[7867010 rows x 5 columns]
```

3. Load (carga)

A etapa Load (carga) no processo ETL é a fase final onde os dados transformados serão carregados para o destino final. Esse destino pode ser um data warehouse, um data lake, um banco de dados específico, ou qualquer outra estrutura de armazenamento de dados destinada a análise e relatórios.

É fundamental entender os métodos de carga. Por exemplo, se for uma carga completa, todos os dados de uma vez no destino, substituindo qualquer dado existente. É usada principalmente para cargas iniciais ou quando a atualização incremental não é viável. Já numa carga incremental, apenas os dados novos ou modificados desde a última carga serão carregados.

3.1. Conectar com um database

Nesse exemplo vamos criar um database SQL local, ou seja, na própria pasta onde estamos executando o código. Dessa forma, não será um database em nuvem (que pode exigir custos de armazenamento e execuções. Essa pasta poderia ser uma pasta compartilhada em intranet por ex., para permitir que outros usuários também explorem os dados.

```
connection = sqlite3.connect('database_negociacoes_B3.db')
cursor = connection.cursor()
```

Observe em sua pasta que um "Data Base File (.db)" foi criado mas tem tamanho de arquivo 0.

3.2. Carga inicial (completa) do database

Nosso database SQL será composto de duas tabelas: participantes_B3 e negociacoes_diarias.

3.2.1. Dados dos participantes

No caso de participantes_B3, é uma tabela que será carregada apenas uma vez, na primeira rotina. Carregaremos o arquivo parquet como dataframe e incluiremos no nosso database_negociacoes_B3. O nome desta tabela no "database_negociacoes_B3".db será "Participantes_B3".

```
df_participantes_B3 =
pd.read_parquet('cleaned_participantes_index.parquet',
engine='pyarrow')
df_participantes_B3.to_sql(name='Participantes_B3', con=connection,
index=False)
```

```
93
```

Veja que agora o arquivo database_negociacoes_B3.db já não tem tamanho 0 mais

3.2.1.1. Testar consultas (queries) ao database criado

Aqui precisamos entender um pouco sobre linguagem SQL. Entretanto as explicações dessa linguagem fogem do nosso escopo Python.

Vamos fazer três consultas rápidas:

1. Obter os primeiros 5 dados da tabela
2. Obter nome do participante que tem o index '16'

```
query = """
SELECT *
FROM Participantes_B3 LIMIT 5
"""

cursor.execute(query)
cursor.fetchall()
```

```
[('BANCO DO BRASIL S/A', '20017'),
 ('BANCO CENTRAL DO BRASIL', '810'),
 ('BANCO B3 S.A.', '500'),
 ('BANCO RABOBANK INTERNATIONAL BRASIL S/A', '50054'),
 ('BANCO COOPERATIVO SICREDI S/A', '50087')]
```

```
query = """
SELECT *
FROM Participantes_B3
WHERE Participante_index = '16'
"""

cursor.execute(query)
cursor.fetchone()
```

```
('J.P. MORGAN CCVM S/A', '16')
```

3.2.2. Dados de negociações

3.2.2.1. Carga completa dados de negociações

Para fins didáticos, vamos imaginar que tivéssemos apenas os dados de negociações dos três primeiros dias da nossa rotina de Extração e Transformação.

Vamos carregar os arquivos parquets como dataframes, concatenar e incluir no nosso database_negociacoes_B3 com o nome da Tabela "Negociacoes_diarias"

```
negociacoes_dia1 = pd.read_parquet('2024_07_01_tick_B3.parquet',
engine='pyarrow')
negociacoes_dia2 = pd.read_parquet('2024_07_02_tick_B3.parquet',
engine='pyarrow')
negociacoes_dia3 = pd.read_parquet('2024_07_03_tick_B3.parquet',
engine='pyarrow')

negociacoes = pd.concat([negociacoes_dia1, negociacoes_dia2,
negociacoes_dia3], ignore_index=True)
negociacoes
```

```
         DataReferencia CodigoInstrumento  PrecoNegocio
QuantidadeNegociada  \
0            2024-07-01            TF583R         10.00
10000
1            2024-07-01            WSPU24       5537.00
1
2            2024-07-01            WSPU24       5537.00
1
3            2024-07-01            WSPU24       5537.00
1
4            2024-07-01            WSPU24       5538.00
1
...                 ...               ...           ...
...
30868115     2024-07-03            INDQ24     127540.00
5
30868116     2024-07-03            INDQ24     127540.00
5
30868117     2024-07-03            INDQ24     127540.00
5
30868118     2024-07-03            INDQ24     127540.00
5
30868119     2024-07-03            FRCF25          6.58
1300

         HoraFechamento CodigoParticipanteComprador
CodigoParticipanteVendedor
0              03:22:17                         100
100
```

```
1              09:00:00                              3
3
2              09:00:00                              3
3
3              09:00:00                              8
3
4              09:00:00                              8
3
...                 ...                            ...
...
30868115       18:31:25                             88
1099
30868116       18:31:25                             83
3
30868117       18:31:25                             83
88
30868118       18:31:25                            120
114
30868119       18:32:27                             88
88

[30868120 rows x 7 columns]
```

Veja que já estamos falando em 30 milhões de linhas. Quer ficar preso apenas ao Excel? ;)

Agora a carga completa destes dados

```
negociacoes.to_sql(name='Negociacoes_diarias', con=connection,
index=False)
```

```
30868120
```

3.3. Fechar conexão com um database

Como finalizamos a carga, é importante fechar a conexão com nosso Banco de Dados

```
connection.close()
```

3.4. Carga incremental do database

Novamente para fins didáticos, vamos imaginar que executamos novamente a rotina de extração e transformação dos dados de negociações para um quarto dia.

Primeiro vamos reestabeler a conexão Python-Database

```
connection = sqlite3.connect('database_negociacoes_B3.db')
cursor = connection.cursor()
```

Carregar os dados do quarto dia

```
negociacoes_dia4 = pd.read_parquet('2024_07_04_tick_B3.parquet',
engine='pyarrow')
negociacoes_dia4.to_sql(name='Tabela_incremental', con=connection,
index=False)
```

Observe que agora temos 3 tabelas no Database

```
print(pd.read_sql_query("SELECT name FROM sqlite_master WHERE
type='table';", connection))
```

```
                  name
0     Participantes_B3
1  Negociacoes_diarias
2   Tabela_incremental
```

Vamos unir as duas através de queries. Primeiro vamos checar o tamanho das duas tabelas

```
query = """
SELECT count('DataReferencia')
FROM Negociacoes_diarias
"""

cursor.execute(query)
cursor.fetchall()
```

```
[(38735130,)]
```

Note que temos mais de 30 milhões de linhas. Agora mesma query para a Tabela_incremental

```
query = """
SELECT count('DataReferencia')
FROM Tabela_incremental
"""

cursor.execute(query)

df_participantes = cursor.fetchall()
print(df_participantes)
```

```
[(7867010,)]
```

Quase 8 milhões de linhas. Agora uma query para unir as duas tabelas verticalmente

```
query = """
INSERT INTO Negociacoes_diarias
SELECT *
FROM Tabela_incremental;
"""
```

```
cursor.execute(query)
```

```
<sqlite3.Cursor at 0x223a18bafc0>
```

Vemos que as tabelas foram concatenadas

```
query = """
SELECT count('DataReferencia')
FROM Negociacoes_diarias
"""

cursor.execute(query)
cursor.fetchall()
```

```
[(38735130,)]
```

Por fim, podemos remover a Tabela_incremental do nosso database e fechar a conexão

```
print(pd.read_sql_query("SELECT name FROM sqlite_master WHERE
type='table';", connection))
```

```
                  name
0     Participantes_B3
1  Negociacoes_diarias
2   Tabela_incremental
```

```
query = """
DROP TABLE Tabela_incremental
"""

cursor.execute(query)
```

```
<sqlite3.Cursor at 0x223a18b8540>
```

```
print(pd.read_sql_query("SELECT name FROM sqlite_master WHERE
type='table';", connection))
```

```
                  name
0     Participantes_B3
1  Negociacoes_diarias
```

Enviar as modificações e fechar conexão

```
connection.commit()
connection.close()
```