ETL

July 14, 2023

1 Carga de dados ETL - Credit Card Transactions

O armazenamento de dados em um SGBD é uma etapa essencial para o uso no contexto de uma empresa. Um dado quando armazenado, também deve estar disponível para acesso das áreas e times interessados. Pensando no contexto mais tradicional, para que um dado seja aplicado em análises, é necessário que ele seja coletado e armazenado.

Dependendo de como os dados estão organizados, uma alternativa para esse armazenamento é o uso de um banco de dados relacional. Durante preparação dos dados e a insersação no banco, deve-se levar em consideração o relacionamento entre tabelas e a redudância de dados, pois existe um custo atrelado ao armazenamento e consulta. Esse é um trabalho essêncial nas etapas de engenharia de dados, mais especificamente armazenamento e dispobilidade de dados.

1.1 Arquivos Utilizados

Os dados que serão utilizados para carga ETL podem ser acessados clicando aqui, e foram gerados a partir de uma simulação realizada pela IBM de transações de cartões. A vantagem do uso desses dados para o objetivo especificado é a possibilidade de diminuição de criação de tabelas intermediárias, para otimizar o espaço de armazenamento. Além há a necessidade de tratamento básico em algumas variáveis, que é justamente um trabalho necessário no processo de ETL para disponibilização do dado para consumo.

Os dados estão disponibilizados sob a licensa Apache 2.0, e o arquivo LICENSE pode acessado clicando aqui, conforme descrito pelos termos da licença.

1.2 Objetivo

O escopo aqui delmitado é a criação de um banco de dados e carga de dados em um processo ETL, utilizando apenas ferramentas básicas de manipulação de dados e o connector do SGBD em questão. Deverá ser considerado o custo de armazenamento e consultas. Além disso, serão realizadas otimizações na formo como os dados serão armazenados, incluindo informações de metadados, que em um ambiente real possuem valor na organização dos dados e auxilio na realização de consultas de áreas interessadas.

2 Preparação do Ambiente de Trabalho

[1]:
```
import pandas as pd
import numpy as np
from zipfile import ZipFile
import os
import mariadb
import time
from sklearn.preprocessing import LabelEncoder
```

[2]:
```
from platform import python_version
print(python_version())
```

```
3.11.3
```

[3]:
```
import watermark.watermark as watermark
%load_ext watermark
```

[4]:
```
%watermark --iversions
```

```
numpy  : 1.24.4
mariadb: 1.1.7
pandas : 2.0.3
```

2.1 ETL - Extract, Transform, Load

[5]:
```
with ZipFile("archive.zip", "r") as zip_ref:
    zip_ref.extractall(os.getcwd() + "/dataset")
```

[6]:
```
transactions = pd.read_csv('./dataset/credit_card_transactions-ibm_v2.csv')
cards = pd.read_csv('./dataset/sd254_cards.csv')
users = pd.read_csv('./dataset/sd254_users.csv')
```

[7]:
```
tabelas = [transactions, cards, users]

for i in tabelas:
    i.columns = i.columns.str.replace(' ', '').str.replace('?', '').str.
 ↪replace('-', '')
```

Além do ajuste no nome de colunas presentes, pode ser interessante a geração de novas tabelas, utilizando a tecnica de encoding e até mesmo outras tabelas, com a separação das informações presentes na combinação das 3 tabelas. Isso tem como intuito minimizar a quantidade de espaço necessário para armazenamento dos dados em um SGBD, e aumentar a velocidade com que consultas são executadas. Em contrapartida, a leitura e interpretação das tabelas podem se tornar mais complexas.

Além disso, é necessário a realização de ajustes em colunas que representam valores monetários, pois estão representadas como texto, utilizando o simbolo de $.

[8]:
```
le = LabelEncoder()

colunas_encoding = [['UseChip', 'IsFraud'], ['CardBrand', 'CardType',␣
 ↪'HasChip', 'CardonDarkWeb'], ['Gender']]
colunas_cifrao = [['Amount'], ['CreditLimit'], ['PerCapitaIncomeZipcode',␣
 ↪'YearlyIncomePerson', 'TotalDebt']]

for x, i in enumerate(tabelas):
    for v in colunas_cifrao[x]:
        i[v] = i[v].str.replace("$", '').astype('float64')
    for w in colunas_encoding[x]:
        i[w + 'Encoding'] = le.fit_transform(i[w])
```

[9]:
```
cardbrand = cards[['CardBrand', 'CardBrandEncoding']].drop_duplicates().
 ↪reset_index(drop = True)
cardtype = cards[['CardType', 'CardTypeEncoding']].drop_duplicates().
 ↪reset_index(drop = True)
chipcode = transactions[['UseChip', 'UseChipEncoding']].drop_duplicates().
 ↪reset_index(drop = True)
```

Uma vez criada tabelas auxiliares e as colunas de encoding, é conveniente realizar a deleção das variáveis utilizadas nos datasets principais.

A coluna `CurrentAge` na tabela users também pode ser removida, uma vez que em termos de armazenamento não é uma forma conveniente de armazenar uma idade, pois o dado ficará defasado, necessitando de atualização constnate. Além disso, há as colunas `BirthYear` e `BirthMonth` que também podem entregar a mesma informação, sem necessidade de serem atualizadas.

[10]:
```
cards.head(2)
```

[10]:

	User	CARDINDEX	CardBrand	CardType	CardNumber	Expires	CVV	HasChip	\
0	0	0	Visa	Debit	4344676511950444	12/2022	623	YES	
1	0	1	Visa	Debit	4956965974959986	12/2020	393	YES	

	CardsIssued	CreditLimit	AcctOpenDate	YearPINlastChanged	CardonDarkWeb	\
0	2	24295.0	09/2002	2008	No	
1	2	21968.0	04/2014	2014	No	

	CardBrandEncoding	CardTypeEncoding	HasChipEncoding	CardonDarkWebEncoding
0	3	1	1	0
1	3	1	1	0

[11]:
```
users.head(2)
```

[11]:

	Person	CurrentAge	RetirementAge	BirthYear	BirthMonth	Gender	\
0	Hazel Robinson	53	66	1966	11	Female	
1	Sasha Sadr	53	68	1966	12	Female	

```
                  Address  Apartment         City State  Zipcode  Latitude  \
0           462 Rose Lane        NaN     La Verne    CA    91750     34.15
1  3606 Federal Boulevard        NaN  Little Neck    NY    11363     40.76

   Longitude  PerCapitaIncomeZipcode  YearlyIncomePerson  TotalDebt  \
0    -117.76                 29278.0             59696.0   127613.0
1     -73.74                 37891.0             77254.0   191349.0

   FICOScore  NumCreditCards  GenderEncoding
0        787               5               0
1        701               5               0
```

[12]:
```
transactions.head(2)
```

[12]:
```
   User  Card  Year  Month  Day   Time  Amount            UseChip  \
0     0     0  2002      9    1  06:21  134.09  Swipe Transaction
1     0     0  2002      9    1  06:42   38.48  Swipe Transaction

          MerchantName   MerchantCity MerchantState      Zip   MCC Errors  \
0  3527213246127876953       La Verne            CA  91750.0  5300    NaN
1 -727612092139916043  Monterey Park            CA  91754.0  5411    NaN

  IsFraud  UseChipEncoding  IsFraudEncoding
0      No                2                0
1      No                2                0
```

[13]:
```
cards.drop(['CardBrand', 'CardType', 'HasChip', 'CardonDarkWeb'], axis = 1,␣
↪inplace = True)
users.drop(['Gender', 'CurrentAge'], axis = 1, inplace = True)
transactions.drop(['UseChip', 'IsFraud'], axis = 1, inplace = True)
```

Observando as tabelas, de forma mais minuciosa, é possível perceber que a tabela de usuários não possui um identificador comcorrespondência na tabela de transações e cartões. A coluna que poderia ser utilizada para isso seria a coluna `User`, presente nas tabelas cards e transactions. Ainda observando, essa variável pode ser criada, uma vez que os dados foram gerados de forma sequêncial e todos os cartões na tabela **cards** estão de forma sequencial, correspondentes a pessoa da tabela **users**.

A criação de um ID único na tabela `transactions`.

[14]:
```
users = users.assign(User = range(len(users)))
transactions = transactions.assign(transaction_id = range(len(transactions)))
```

Uma vez com os dados devidamente preparados, basta salvar as tabelas e realizar a construção do banco de dados e relacionamento entre as tabelas.

[15]:
```
tabelas = [users, cardbrand, cardtype, chipcode, cards, transactions]
tabelas_nomes = ['users.csv', 'cardbrand.csv', 'cardtype.csv', 'chipcode.csv',␣
↪'cards.csv', 'transactions.csv']
```

```
if not os.path.exists("./datasets_modificados"):
    os.mkdir('./datasets_modificados')

for x, i in enumerate(tabelas):
    i.to_csv('./datasets_modificados/' + tabelas_nomes[x] , index = False)
```

[16]:
```
def data_type_sql(dataframe):
    """Função de determina o formato de dados paras as colunas de uma tabela em␣
 ↪SQL"""

    dict_schema = {coluna : str(tipo_dado) for (coluna, tipo_dado) in␣
 ↪zip(list(dataframe.columns), [x for x in list(dataframe.dtypes)])}
    chaves = list(dict_schema)

    for i in chaves:
        if dict_schema[i] == 'int64':
            if len(dataframe[i].unique()) == 2:
                dict_schema[i] = 'BOOLEAN'
            elif dataframe[i].abs().max() <= 2147483647:
                dict_schema[i] = 'INT'
            else:
                dict_schema[i] = 'BIGINT'
        elif dict_schema[i] == 'float64':
            if dataframe[i].abs().max() <= 3.402823466E+38:
                dict_schema[i] = 'FLOAT'
            else:
                dict_schema[i] = 'DOUBLE'
        else:
            dict_schema[i] = 'VARCHAR(255)'

        lista_chaves = list(dict_schema.keys())
        lista_valores = list(dict_schema.values())
        string_schema = ""

        for x, v in enumerate(lista_chaves):
            string_schema = string_schema + lista_chaves[x] + " " +␣
 ↪lista_valores[x] + ", "
    return string_schema[:-2].split(', ')
```

[17]:
```
# Descrição de cada coluna presente nas tabelas

comments = [['Nome do portador do cartão', 'Idade esperada de aposentadoria',␣
 ↪'Ano de nascimento', 'Mês de nascimento', 'Endereço de Residência',␣
 ↪'Quantidade de apartamentos que possui', 'Cidade de residência',
```

```
            'Estado de Residência', 'Código Postal', 'Distância em Graus da␣
↪Linha do Equador', 'Distância em Graus do Meridiano Greenwich', 'Renda Per␣
↪Capita por Código Postal',
            'Renda Anual', 'Total em Dívida', 'Score de Crédito', 'Número de␣
↪cartões de crédito', 'Código correspondente ao genero, 0 para Feminino e 1␣
↪para Masculino', 'Id correspondente do usuário'],

           ['Bandeira do Cartão', 'Código identificador, correspondente a␣
↪Bandeira do Cartão'],

           ['Tipo de Cartão', 'Código identificador, correspondente ao Tipo de␣
↪Cartão'],

           ['Tipo de transação', 'Código identificador, correspondente ao Tipo␣
↪de Transação'],

           ['Id correspondente do usuário', 'Id correspondente do cartão de␣
↪crédito', 'Número do Cartão de Crédito', 'Data de Expiração', 'Card␣
↪Verification Value','Quantidade de Cartões',
            'Limite de Crédito', 'Data de contratação', 'Ultimo ano de␣
↪Alteração do PIN', 'Bandeira do Cartão, 0 para Amex, 1 Para Discover, 2 para␣
↪Mastercard, 3 para Visa',
            'Tipo do cartão, 0 para Crédito, 1 para Débito e 2 para Débito␣
↪pré-pago', 'Informação se o cartão possui Chip, 1 para Sim e 0 para Não',
            'Informação se o cartão já foi encontrado em vazamentos da␣
↪DarkWeb, 1 para Sim e 0 para não'],

           ['Id correspondente do usuário', 'Id correspondente do cartão de␣
↪crédito', 'Ano da transação', 'Mês da Transação', 'Dia da Transação',␣
↪'Horário da Transação', 'Valor da transação',
            'Nome do estabelecimento', 'Cidade em que a transação foi␣
↪realizada', 'Estado em que a transação foi realizada', 'Código postal do␣
↪local em que a transação foi realizada', 'Merchant Category Code',
            'Motivo do erro da transação', 'Tipo de transação realizada, 0␣
↪para Chip Transaction, 1 para Online Transaction, 2 para Swipe Transaction',
            'Código identificador para se a transação realizada é fraudulenta,␣
↪0 para Não e 1 para Sim', 'Id correspondente da transação']]
```

[18]:
```
query_alter = [["ALTER TABLE users ADD PRIMARY KEY (User);"],

               ["ALTER TABLE cardbrand ADD PRIMARY KEY (CardBrandEncoding);"],

               ["ALTER TABLE cardtype ADD PRIMARY KEY (CardTypeEncoding);"],

               ["ALTER TABLE chipcode ADD PRIMARY KEY (UseChipEncoding);"],
```

```
                ["ALTER TABLE cards ADD PRIMARY KEY (User,CARDINDEX);",
                 "ALTER TABLE cards ADD FOREIGN KEY (User) REFERENCES␣
 ↪users(User);",
                 "ALTER TABLE cards ADD FOREIGN KEY (CardBrandEncoding)␣
 ↪REFERENCES cardbrand(CardBrandEncoding);",
                 "ALTER TABLE cards ADD FOREIGN KEY (CardTypeEncoding)␣
 ↪REFERENCES cardtype(CardTypeEncoding);"],

                ["ALTER TABLE transactions ADD PRIMARY KEY (transaction_id);",
                 "ALTER TABLE transactions ADD FOREIGN KEY (User) REFERENCES␣
 ↪users(User);",
                 "ALTER TABLE transactions ADD FOREIGN KEY (User) REFERENCES␣
 ↪users(USER);",
                 "ALTER TABLE transactions ADD FOREIGN KEY (User, Card)␣
 ↪REFERENCES cards(User, CARDINDEX);",
                 "ALTER TABLE transactions ADD FOREIGN KEY (UseChipEncoding)␣
 ↪REFERENCES chipcode(UseChipEncoding);"]]
```

[19]:
```
conn_params = {
    "user" : "root", # Caso tenha criado o banco utilizando os comandos em␣
 ↪docker pode ser substituido por root
    "password" : "your_password", # Senha definida para o usuário no banco de␣
 ↪dados
    "host" : "IP_host", # Endereço IP do banco de dados, caso o banco tenha␣
 ↪sido criado utilizando docker, a porta pode ser vista rodando o comando␣
 ↪docker ps, caso conste --protocol=tcp, a porta definida é 127.0.0.1
    "port" : 3406
}

connection = mariadb.connect(**conn_params)
cursor = connection.cursor()

cursor.execute("CREATE OR REPLACE DATABASE credit_card")
cursor.execute("USE credit_card")
```

[20]:
```
for x, i in enumerate(tabelas_nomes):

    # Definindo o esquema da tabela
    schema_dados = data_type_sql(tabelas[x])

    # Construindo a query para criação da tabela
    query = "CREATE OR REPLACE TABLE {} (".format(i[:-4])

    commentarios_colunas = comments[x]
    for w, y in enumerate(schema_dados):
        query = query + y + " COMMENT '" + commentarios_colunas[w]  + "', "
```

```
    query = query[:-2] + ')'
    cursor.execute(query)

    # Adição das contrains para as tabelas criadas
    for j in query_alter[x]:
        cursor.execute(j)

    # Carregando os arquivos por linha de comando
    query_load = "LOAD DATA LOCAL INFILE '" + './datasets_modificados/' + i +␣
↪"' INTO TABLE " + i[:-4] + " FIELDS TERMINATED BY ',' ENCLOSED BY '\"'␣
↪ESCAPED BY '\\\\' IGNORE 1 LINES"
    cursor.execute(query_load)
    connection.commit()

    print("Arquivo", i, "carregado")
```

```
Arquivo users.csv carregado
Arquivo cardbrand.csv carregado
Arquivo cardtype.csv carregado
Arquivo chipcode.csv carregado
Arquivo cards.csv carregado
Arquivo transactions.csv carregado
```

2.2 Conclusão

Os dados foram devidamente carregados, com tabelas intermediárias, relacionamento e descrições referentes a cada coluna e estão disponíveis para acesso no banco de dados.