Fase Preliminar

Instalação de bibliotecas

In [ ]:
```
# Caso o algoritmo não execute devido a bibliotecas faltantes, descomente
# as linhas abaixo e instale as dependências.

# %pip install sqlalchemy
# %pip install numpy
# %pip install pandas
```

Importação de bibliotecas

In [ ]:
```
import numpy as np
import pandas as pd
import sqlalchemy as sql
import urllib.parse
from sqlalchemy import create_engine
```

Conexão com a base de dados

In [ ]:
```
password = urllib.parse.quote_plus('D3@bGh664%$1VHv*')
engine = create_engine('mysql://candidate_user:' + password + '@dhauz-instance.cut
```

In [ ]:
```
# Função que executa a query desejada
def executarQuery(query):
  connection = engine.connect()
  try:
    # Executa a query e cria um pandas DataFrame com o resultado
    resultado = pd.DataFrame(engine.execute(query).fetchall())
    connection.close()
    return resultado
  except Exception as e:
    # Lidando com erros
    connection.close()
    print(str(e))
```

Exploração inicial dos dados

In [ ]:
```
# Tabela raw_transactions_table
query = 'SELECT * FROM db_hiring_test.raw_transactions_table;'
transactions_inicial = executarQuery(query)
transactions_inicial.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6176 entries, 0 to 6175
Data columns (total 7 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   IdTransaction       6176 non-null   object
 1   AddressOrigin       6176 non-null   object
 2   AddressDestination  6176 non-null   object
 3   TotalSent           6176 non-null   object
 4   Status              6176 non-null   object
 5   SentDate            6176 non-null   object
 6   ImportDate          6176 non-null   object
dtypes: object(7)
memory usage: 337.9+ KB
```

In [ ]:

```
# Analisando valores únicos da variável ImportDate
print('Valores únicos da variável ImportDate: {0}'.format(transactions_inicial.Imp
# Analisando valores únicos da variável Status
print('Valores únicos da variável Status: {0}'.format(transactions_inicial.Status.
# Checando NAs
transactions_inicial = transactions_inicial.replace('',np.nan)
print('Quantidade de valores NA: {}'.format(transactions_inicial.isnull().values.s
```

```
Valores únicos da variável ImportDate: ['2021-01-31 23:59:59' '2021-02-05 23:59:5
9']
Valores únicos da variável Status: ['Confirmed' 'Denied' 'Pending']
Quantidade de valores NA: 5
```

In [ ]:

```
transactions_inicial.TotalSent = transactions_inicial.TotalSent.str.replace('(','-
transactions_inicial.TotalSent = transactions_inicial.TotalSent.str.replace(')',''
transactions_inicial.TotalSent = transactions_inicial.TotalSent.str.replace(',',''
transactions_inicial.TotalSent = transactions_inicial.TotalSent.astype('float')
transactions_inicial.head()
```

```
C:\Users\MATHEU~1\AppData\Local\Temp/ipykernel_13784/3419425904.py:1: FutureWarnin
g: The default value of regex will change from True to False in a future version.
In addition, single character regular expressions will *not* be treated as literal
strings when regex=True.
  transactions_inicial.TotalSent = transactions_inicial.TotalSent.str.replace
('(','-')
C:\Users\MATHEU~1\AppData\Local\Temp/ipykernel_13784/3419425904.py:2: FutureWarnin
g: The default value of regex will change from True to False in a future version.
In addition, single character regular expressions will *not* be treated as literal
strings when regex=True.
  transactions_inicial.TotalSent = transactions_inicial.TotalSent.str.replace
(')','')
```

Out[ ]:

	IdTransaction	AddressOrigin	AddressDestination	TotalSent	Status	SentDate	ImportDate
0	ID1002	A-77	A-49	293659.0	Confirmed	2021-01-08 13:34:04	2021-01-31 23:59:59
1	ID2014	A-24	A-58	542285.0	Confirmed	2021-01-17 13:34:04	2021-01-31 23:59:59
2	ID1092	A-15	A-20	57493.0	Confirmed	2021-01-03 03:07:57	2021-01-31 23:59:59
3	ID1603	A-84	A-59	883745.0	Confirmed	2021-01-02 06:36:39	2021-01-31 23:59:59
4	ID253	A-86	A-44	194591.0	Confirmed	2021-01-14 20:22:08	2021-01-31 23:59:59

Algumas conclusões sobre a base raw_transactions_table:

- Variável ImportDate com duas datas de importação de dados;
- Variável Status com 3 valores: "Denied", "Pending" and "Confirmed";
- Variável TotalSent é do tipo String e há valores entre parênteses que estou considerando como sendo números negativos. É preciso convertê-la para número antes de efetuar operações.
- Há 5 valores ausentes na base

In [ ]:

```
# Tabela raw_transactions_fee
query = 'SELECT * FROM db_hiring_test.raw_transactions_fee;'
fees = executarQuery(query)
fees.head(6)
```

Out[ ]:

	range-start	range-end	fee-percentage
0	0.00	160000.00	10.00
1	160000.01	340000.00	8.00
2	340000.01	500000.00	6.00
3	500000.01	670000.00	5.00
4	670000.01	833000.00	4.00
5	833000.01	99000000.00	2.00

Fase 1

Questão 1

A função `questao_um_f1()` resolve a questão 1. As operações realizadas foram o agrupamento dos dados pela variável AddressOrigin e a contagem de quantas transações cada carteira enviou. Selecionei o identificador da carteira de origem e a contagem de operações para mostrar o resultado. Nesta questão, não foi feito o filtro pela variável Status,

contabilizando todas as transações independentemente de suas confirmações, e também valores ausentes não foram tratados.

In [ ]:
```
def questao_um_f1():
  query = 'SELECT AddressOrigin, max(Volume) as Volume ' + \
            'FROM (' + \
                  'SELECT AddressOrigin, COUNT(*) as Volume ' + \
                  'FROM db_hiring_test.raw_transactions_table ' + \
                  'GROUP BY AddressOrigin' + \
            ') AS groupAddressOrigin;'
  return executarQuery(query)

resposta = questao_um_f1()
print('A carteira com o maior volume de transações enviadas foi a {0}, com {1} tra
      .format(resposta.AddressOrigin[0],resposta.Volume[0]))
print('Resposta obtida a partir da query:')
resposta
```

```
A carteira com o maior volume de transações enviadas foi a A-77, com 90 transações
enviadas.

Resposta obtida a partir da query:
```

Out[ ]:

	AddressOrigin	Volume
0	A-77	90

Questão 2

A função `questao_dois_f1()` resolve a questão 2. As operações realizadas foram o agrupamento dos dados pelo dia do mês extraído da coluna SentDate e a contagem de quantas transações foram feitas em cada dia. Selecionei o dia do mês e a contagem de operações para mostrar o resultado. Nesta questão, não foi feito o filtro pela variável Status, contabilizando todas as transações independentemente de suas confirmações, e também valores ausentes não foram tratados.

In [ ]:
```
def questao_dois_f1():
  query = 'SELECT SentDay, max(Volume) AS Volume ' + \
          'FROM (' + \
          'SELECT DAY(SentDate) AS SentDay, COUNT(*) AS Volume ' + \
          'FROM db_hiring_test.raw_transactions_table ' + \
          'GROUP BY SentDay ' + \
          ') AS groupSentDay;'
  return executarQuery(query)


resposta = questao_dois_f1()
print('O dia do mês com o maior volume de transações enviadas foi o dia {0}, com {
      .format(resposta.SentDay[0],resposta.Volume[0]))
print('Resposta obtida a partir da query:')
resposta
```

```
O dia do mês com o maior volume de transações enviadas foi o dia 8, com 288 transa
ções enviadas.

Resposta obtida a partir da query:
```

Out[ ]:

	SentDay	Volume
0	8	288

Questão 3

A função `questao_tres_f1()` resolve a questão 3. As operações realizadas foram o agrupamento dos dados pelo dia da semana de cada transação extraído da coluna SentDate e a contagem de quantas transações foram feitas em cada dia. Selecionei o dia da semana e a contagem de operações para mostrar o resultado. Nesta questão, não foi feito o filtro pela variável Status, contabilizando todas as transações independentemente de suas confirmações, e também valores ausentes não foram tratados.

In [ ]:

```
def questao_tres_f1():
  query = 'SELECT WeekDayNumber, max(Volume) AS Volume ' + \
          'FROM ( ' + \
          'SELECT DAYOFWEEK(SentDate) AS WeekDayNumber, COUNT(*) AS Volume ' + \
          'FROM db_hiring_test.raw_transactions_table ' + \
          'GROUP BY WeekDayNumber ' + \
          ') AS groupWeekDay;'
  return executarQuery(query)

# Objeto que define os dias da semana utilizados como padrão da função DAYOFWEEK()
# Fonte: https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html#func
diasSemana = {
  1: 'Domingo',
  2: 'Segunda-Feira',
  3: 'Terça-Feira',
  4: 'Quarta-Feira',
  5: 'Quinta-Feira',
  6: 'Sexta-Feira',
  7: 'Sábado'
}

resposta = questao_tres_f1()
numeroDia = resposta.WeekDayNumber[0]
print('O dia da semana com o maior volume de transações enviadas foi {0}, com {1}
      .format(diasSemana.get(numeroDia),resposta.Volume[0]))
print('Resposta obtida a partir da query:')
resposta
```

```
O dia da semana com o maior volume de transações enviadas foi Sexta-Feira, com 978
transações enviadas.

Resposta obtida a partir da query:
```

Out[ ]:

	WeekDayNumber	Volume
0	6	978

Questão 4

Para esta questão, foi necessário investigar a base de dados para encontrar valores fora da conformidade da base de dados. Como na exploração de dados inicial constatou-se que todas as variáveis estão no formato String e que os valores nulos estão presentes nas colunas AddressOrigin, AddressDestination e TotalSent, a primeira etapa da busca por dados atípicos baseou-se na filtragem da base por valores que fossem iguais a uma string vazia ('') nessas colunas, definida na função `buscarStringVazia()` .

In [ ]:
```
# Query que busca strings vazias
def buscarStringVazia():
  query = 'SELECT IdTransaction, AddressOrigin, AddressDestination, TotalSent ' +
          'FROM db_hiring_test.raw_transactions_table ' + \
          'WHERE AddressOrigin = "" OR AddressDestination = "" OR TotalSent = ""'
  return executarQuery(query)

stringsVazias = buscarStringVazia()
stringsVazias.head()
```

Out[ ]:

	IdTransaction	AddressOrigin	AddressDestination	TotalSent
0	ID3138	A-57		157,964.00
1	ID3046	A-93	A-80	
2	ID3017		A-66	380,569.00
3	ID3153	A-74	A-69	
4	ID3169	A-69	A-28	

In [ ]:
```
# Como esperado, a query retornou uma tabela com 5 linhas, cada uma com um valor n
stringsVazias.shape
```

Out[ ]:
```
(5, 4)
```

A segunda etapa de investigação da base contou com a ordenação da coluna TotalSent em diferentes ordens para verificar a presença de caracteres especiais, uma vez que a variável é do tipo String. A função `buscarCaracteres()` realiza o procedimento mencionado.

In [ ]:
```
def buscarCaracteres():
  query = 'SELECT IdTransaction, AddressOrigin, AddressDestination, TotalSent ' +
          'FROM db_hiring_test.raw_transactions_table ' + \
          'WHERE AddressOrigin != "" AND AddressDestination != "" AND TotalSent !=
          'ORDER BY TotalSent ASC'
  ordemCrescente = executarQuery(query)

  query = 'SELECT IdTransaction, AddressOrigin, AddressDestination, TotalSent ' +
          'FROM db_hiring_test.raw_transactions_table ' + \
          'WHERE AddressOrigin != "" AND AddressDestination != "" AND TotalSent !=
          'ORDER BY TotalSent DESC'
  ordemDecrescente = executarQuery(query)

  return [ordemCrescente, ordemDecrescente]

ordemCrescente, ordemDecrescente = buscarCaracteres()
print('Ordenação crescente: \n')
print(ordemCrescente.head(10))
print('\n Ordenação decrescente: \n')
print(ordemDecrescente.head())
```

```
Ordenação crescente:

  IdTransaction AddressOrigin AddressDestination     TotalSent
0        ID3075          A-91                A-92      (453.00)
1        ID3002          A-87                A-66   (56,213.00)
2        ID1237          A-67                A-60     1,520.00
3        ID1237          A-67                A-60     1,520.00
4        ID2871          A-87                A-15    10,043.00
5        ID2871          A-87                A-15    10,043.00
6         ID893          A-40                A-14    10,620.00
7         ID893          A-40                A-14    10,620.00
8         ID266          A-50                A-84   100,156.00
9         ID266          A-50                A-84   100,156.00

 Ordenação decrescente:

  IdTransaction AddressOrigin AddressDestination     TotalSent
0           ID8          A-78                A-34   999,565.00
1           ID8          A-78                A-34   999,565.00
2        ID2794          A-95                A-18   999,367.00
3        ID2794          A-95                A-18   999,367.00
4        ID2085          A-59                A-28   999,235.00
```

Nestas operações é possível notar que os 2 primeiros valores da ordenação crescente da coluna TotalSent apresentam valores discrepantes com a presença de parênteses. Geralmente, valores entre parênteses se referem a números negativos, porém, em uma tabela com transações financeiras, valores negativos não fazem sentido. Sendo assim, uma função foi criada para concatenar os valores atípicos encontrados, sendo strings vazias ou strings com caracteres especiais, definida por `concatenarOutliers()` .

In [ ]:

```
def concatenarOutliers():
  stringsVazias = buscarStringVazia()
  caracteresEspeciais, _ = buscarCaracteres()

  concat = pd.concat([stringsVazias,caracteresEspeciais[0:2]]) \
  .sort_values('IdTransaction', ascending = True, ignore_index = True)

  return concat

concat = concatenarOutliers()
print('A tabela resultante com os dados que devem ser validados é:')
concat
```

```
A tabela resultante com os dados que devem ser validados é:
```

Out[ ]:

	IdTransaction	AddressOrigin	AddressDestination	TotalSent
0	ID3002	A-87	A-66	(56,213.00)
1	ID3017		A-66	380,569.00
2	ID3046	A-93	A-80	
3	ID3075	A-91	A-92	(453.00)
4	ID3138	A-57		157,964.00
5	ID3153	A-74	A-69	
6	ID3169	A-69	A-28	

A partir da busca dos dados, conclui-se que as transações que precisam ser validadas possuem os respectivos Ids:

- ID3002
- ID3017
- ID3046
- ID3075
- ID3138
- ID3153
- ID3169

Questão 5

Nesta questão, duas queries foram executadas em paralelo. A primeira para agrupar os dados por carteira e calcular o volume de transações que foram enviados a partir da mesma. A segunda para calcular o volume de transações recebidas por carteira. Em seguida, as duas tabelas resultantes são mescladas e o balanço final de cada carteira é calculado, chegando na resposta para o problema.

In [ ]:

```
def questao_cinco_f1():
  queryEnviadas = 'SELECT * ' + \
          'FROM ( ' + \
            'SELECT AddressOrigin AS Address, SUM(ToFloat) Sent ' + \
            'FROM ( ' + \
            'SELECT * ' + \
            'FROM ( ' + \
              'SELECT *, CAST(Replaced AS FLOAT) ToFloat  ' + \
              'FROM ( ' + \
                'SELECT *, REPLACE(REPLACE(REPLACE(REPLACE(TotalSent, "(", "-"), "
                'FROM db_hiring_test.raw_transactions_table ' + \
                'WHERE Status = "Confirmed" ' + \
              ') filtered ' + \
              'HAVING ToFloat != 0 AND AddressOrigin != "" AND AddressDestination
            ') cast ' + \
          ') result ' + \
          'GROUP BY result.AddressOrigin ) as one ' + \
          'ORDER BY one.Address ASC'
  queryRecebidas = 'SELECT * ' + \
          'FROM ( ' + \
            'SELECT AddressDestination AS Address, SUM(ToFloat) Received ' + \
            'FROM ( ' + \
            'SELECT * ' + \
            'FROM ( ' + \
              'SELECT *, CAST(Replaced AS FLOAT) ToFloat  ' + \
              'FROM ( ' + \
                'SELECT *, REPLACE(REPLACE(REPLACE(REPLACE(TotalSent, "(", "-"), "
                'FROM db_hiring_test.raw_transactions_table ' + \
                'WHERE Status = "Confirmed" ' + \
              ') filtered ' + \
              'HAVING ToFloat != 0 AND AddressOrigin != "" AND AddressDestination
            ') cast ' + \
          ') result ' + \
          'GROUP BY result.AddressDestination) as two ' + \
          'ORDER BY two.Address ASC' ;
  enviadas = executarQuery(queryEnviadas)

  recebidas = executarQuery(queryRecebidas)
  return [enviadas,recebidas]

enviadas, recebidas = questao_cinco_f1()
saldo = pd.merge(enviadas, recebidas, how = 'left', on = 'Address')
saldo['Balance'] = saldo.Received - saldo.Sent
```

```
maiorSaldo = saldo.iloc[saldo['Balance'].idxmax()]
print('A carteira com o maior saldo final foi a {}, com um volume de {}. \n'
.format(maiorSaldo.Address,str( maiorSaldo.Balance).replace('.',',')))

print('A tabela resultante com dados do volume de transações por carteira é:')
saldo
```

```
A carteira com o maior saldo final foi a A-30, com um volume de 25204201,0.

A tabela resultante com dados do volume de transações por carteira é:
```

Out[ ]:

	Address	Sent	Received	Balance
0	A-1	28936433.0	23943366.0	-4993067.0
1	A-10	35188653.0	33828849.0	-1359804.0
2	A-100	18168719.0	32041161.0	13872442.0
3	A-11	38716466.0	30060514.0	-8655952.0
4	A-12	21637926.0	44150299.0	22512373.0
...	...	...	...	...
95	A-95	31189976.0	38228273.0	7038297.0
96	A-96	21354636.0	18378403.0	-2976233.0
97	A-97	29122858.0	23417154.0	-5705704.0
98	A-98	26443299.0	20737724.0	-5705575.0
99	A-99	46284813.0	25900018.0	-20384795.0

100 rows × 4 columns

Fase 2

Nesta fase, optei por realizar uma busca no banco de dados (removendo valores nulos e atípicos e filtrando pelo status "Confirmed") utilizando a linguagem SQL e manipular os dados utilizando Python para economia de tempo.

In [ ]:

```
# Função que realiza a busca na base. Seu resultado será utilizado para resolver a
def buscarBases():
  queryTransactions = 'SELECT IdTransaction, AddressOrigin, SentDate, ToFloat ' +
  'FROM ( ' + \
    'SELECT *, CAST(Replaced AS FLOAT) ToFloat '  + \
    'FROM (' + \
      'SELECT *, REPLACE(REPLACE(REPLACE(REPLACE(TotalSent, "(", "-"), ")",""),","
      'FROM db_hiring_test.raw_transactions_table ' + \
      'WHERE Status = "Confirmed" ' + \
    ') filtered ' + \
    'HAVING ToFloat != 0 AND AddressOrigin != "" AND AddressDestination != "" ) ta

  queryFees = 'SELECT * FROM db_hiring_test.raw_transactions_fee'

  transactions = executarQuery(queryTransactions)
  fees = executarQuery(queryFees).astype('float')
  return [transactions,fees]

transactions, fees = buscarBases()
```

In [ ]:
```
# Função para calcular a taxa paga pela carteira de origem em cada transação.
# Não é a forma ideal de se calcular esta variável, mas, como a base de dados
# é pequena, foi possível realizar esta operação sem grandes perdas no tempo
# de execução do algoritmo.
transactions['Fees'] = 0
for idx, values in fees.iterrows():
  transactions.loc[(transactions.ToFloat >= values['range-start']) & (transactions

transactions['PaidFees'] = transactions['Fees']/100 * transactions['ToFloat']
transactions['SentDateYear'] = pd.DatetimeIndex(transactions['SentDate']).year
transactions['SentDateMonth'] = pd.DatetimeIndex(transactions['SentDate']).month
transactions = transactions.drop_duplicates().reset_index(drop = True)
print('Tabela resultante: \n')
transactions
```

```
Tabela resultante:
```

Out[ ]:

	IdTransaction	AddressOrigin	SentDate	ToFloat	Fees	PaidFees	SentDateYear	SentDateM
0	ID1002	A-77	2021-01-08 13:34:04	293659.0	8	23492.72	2021	
1	ID2014	A-24	2021-01-17 13:34:04	542285.0	5	27114.25	2021	
2	ID1092	A-15	2021-01-03 03:07:57	57493.0	10	5749.30	2021	
3	ID1603	A-84	2021-01-02 06:36:39	883745.0	2	17674.90	2021	
4	ID253	A-86	2021-01-14 20:22:08	194591.0	8	15567.28	2021	
...	...	...	...	...	...	...	...	
3137	ID3119	A-5	2021-02-01 02:53:12	87570.0	10	8757.00	2021	
3138	ID3118	A-55	2021-02-05 02:53:12	367471.0	6	22048.26	2021	
3139	ID3108	A-72	2021-02-04 01:35:23	973492.0	2	19469.84	2021	
3140	ID3050	A-51	2021-02-02 22:56:28	60621.0	10	6062.10	2021	
3141	ID3102	A-57	2021-02-02 22:56:28	225522.0	8	18041.76	2021	

3142 rows × 8 columns

Questão 1

Nesta questão, foi feita a filtragem da base resultante pelo mês e ano da transação, selecionando os valores iguais a 1 para o mês (janeiro) e iguais a 2021 para o ano. A função que realiza a operação é a `questao_um_f2()` .

In [ ]:
```
def questao_um_f2():
  filtro = transactions[(transactions.SentDateMonth == 1) & (transactions.SentDate
  groupFiltro = filtro.groupby('AddressOrigin', as_index = False).agg({
    'PaidFees': 'sum'
  })
  return groupFiltro.iloc[groupFiltro['PaidFees'].idxmax()]
resposta = questao_um_f2()

print('A carteira com o maior pagamento de taxas em janeiro de 2021 foi a {}, com
  resposta.AddressOrigin, resposta.PaidFees
))
```

```
A carteira com o maior pagamento de taxas em janeiro de 2021 foi a A-99, com um va
lor de 996523.18
```

Questão 2

Esta questão seguiu o mesmo procedimento que a questão anterior, mas filtrando com base no mês de fevereiro de 2021 a partir da função `questao_dois_f2()`

In [ ]:
```
def questao_dois_f2():
  filtro = transactions[(transactions.SentDateMonth == 2) & (transactions.SentDate
  groupFiltro = filtro.groupby('AddressOrigin', as_index = False).agg({
    'PaidFees': 'sum'
  })
  return groupFiltro.iloc[groupFiltro['PaidFees'].idxmax()]
resposta = questao_dois_f2()

print('A carteira com o maior pagamento de taxas em fevereiro de 2021 foi a {}, co
  resposta.AddressOrigin, resposta.PaidFees
))
```

```
A carteira com o maior pagamento de taxas em fevereiro de 2021 foi a A-29, com um
valor de 107830.31
```

Questão 3

Para esta questão, a função `questao_tres_f2()` implementa o filtro da base de dados pela linha onde a variável "PaidFees" chegou ao maior valor e retorna o id da transação para tal linha.

In [ ]:
```
def questao_tres_f2():
  return transactions.iloc[transactions['PaidFees'].idxmax()]['IdTransaction']

resposta = questao_tres_f2()

print('O id da transação com a maior taxa paga foi o {}.'.format(resposta))
```

```
O id da transação com a maior taxa paga foi o ID635.
```

Questão 4

A `função questao_quatro_f2()` implementa o cálculo da média de taxa paga pelas carteiras considerando todas as transações realizadas.

In [ ]:
```
def questao_quatro_f2():
  return transactions.PaidFees.mean()

resposta = questao_quatro_f2()
print('A média das taxas pagas pelas carteiras foi igual a {:.2f}.'.format(respost
```

```
A média das taxas pagas pelas carteiras foi igual a 21733.50.
```

NOTA

Após a finalização das questões, percebi que haviam dados duplicados na base de dados, podendo ter comprometido toda a análise da Fase 1. Infelizmente, porém, não houve tempo hábil para verificação e correção.