Avaliação Final

November 16, 2020

1 MBA em Ciência de Dados

2 Análise de Dados com Base em Processamento Massivo em Paralelo

2.1 Avaliação Final

Material Produzido por: >Profa. Dra. Cristina Dutra de Aguiar Ciferri >André Perez >Guilherme Muzzi da Rocha >Jadson José Monteiro Oliveira >João Pedro de Carvalho Castro >Leonardo Mauro Pereira Moraes >Piero Lima Capelo

CEMEAI - ICMC/USP São Carlos

A avaliação final contém 7 questões, as quais estão espalhadas ao longo do texto. Por favor, procurem por Questão para encontrar a especificação das questões. Também é possível localizar as questões utilizando o menu de navegação. O *notebook* contém a constelação de fatos da BI Solutions que deve ser utilizada para responder às questões e também toda a obtenção dos dados e a respectiva geração dos DataFrames e das visões temporárias.

Desejamos uma boa avaliação!

#1 Constelação de Fatos da BI Solutions

A aplicação de *data warehousing* da BI Solutions utiliza como base uma contelação de fatos, conforme descrita a seguir.

Tabelas de dimensão

- data (dataPK, dataCompleta, dataDia, dataMes, dataBimestre, dataTrimestre, dataSemestre, dataAno)
- funcionario (funcPK, funcMatricula, funcNome, funcSexo, funcDataNascimento, funcDiaNascimento, funcMesNascimento, funcAnoNascimento, funcCidade, funcEstadoNome, funcEstadoSigla, funcRegiaoNome, funcRegiaoSigla, funcPaisNome, funcPaisSigla)
- equipe (equipePK, equipeNome, filialNome, filialCidade, filialEstadoNome, filialEstadoSigla, filialRegiaoNome, filialRegiaoSigla, filialPaisNome, filialPaisSigla)
- cargo (cargoPK, cargoNome, cargoRegimeTrabalho, cargoEscolaridadeMinima, cargoNivel)
- cliente (clientePK, clienteNomeFantasia, clienteSetor, clienteCidade, clienteEstadoNome, clienteEstadoSigla, clienteRegiaoNome, clienteRegiaoSigla, clientePaisNome, clientePaisSigla)

Tabelas de fatos - pagamento (dataPK, funcPK, equipePK, cargoPK, salario, quantidadeLancamentos) - negociacao (dataPK, equipePK, clientePK, receita, quantidadeNegociacoes)

#2 Obtenção dos Dados da BI Solutions

2.2 2.1 Baixando o Módulo wget

Para baixar os dados referentes ao esquema relacional da constelação de fatos da BI Solutions, é utilizado o módulo **wget**. O comando a seguir realiza a instalação desse módulo.

[ ]:
```
#instalando o módulo wget
# %%capture
!pip install -q wget
!mkdir data
```

2.3 2.2 Obtenção dos Dados das Tabelas de Dimensão

Os comandos a seguir baixam os dados que povoam as tabelas de dimensão.

[ ]:
```
#baixando os dados das tabelas de dimensão
import wget

url = "https://raw.githubusercontent.com/GuiMuzziUSP/Data_Mart_BI_Solutions/
 ↪main/data.csv"
wget.download(url, "data/data.csv")

url = "https://raw.githubusercontent.com/GuiMuzziUSP/Data_Mart_BI_Solutions/
 ↪main/funcionario.csv"
wget.download(url, "data/funcionario.csv")

url = "https://raw.githubusercontent.com/GuiMuzziUSP/Data_Mart_BI_Solutions/
 ↪main/equipe.csv"
wget.download(url, "data/equipe.csv")

url = "https://raw.githubusercontent.com/GuiMuzziUSP/Data_Mart_BI_Solutions/
 ↪main/cargo.csv"
wget.download(url, "data/cargo.csv")

url = "https://raw.githubusercontent.com/GuiMuzziUSP/Data_Mart_BI_Solutions/
 ↪main/cliente.csv"
wget.download(url, "data/cliente.csv")
```

2.4 2.3 Obtenção dos Dados Tabelas de Fatos

Os comandos a seguir baixam os dados que povoam as tabelas de fatos.

[ ]:
```
#baixando os dados das tabelas de fatos
url = "https://raw.githubusercontent.com/GuiMuzziUSP/Data_Mart_BI_Solutions/
 ↪main/pagamento.csv"
```

```
wget.download(url, "data/pagamento.csv")

url = "https://raw.githubusercontent.com/GuiMuzziUSP/Data_Mart_BI_Solutions/
 ↪main/negociacao.csv"
wget.download(url, "data/negociacao.csv")
```

3 3 Apache Spark Cluster

3.1 3.1 Instalação

Neste *notebook* é criado um *cluster* Spark composto apenas por um **nó mestre**. Ou seja, o *cluster* não possui um ou mais **nós de trabalho** e o **gerenciador de cluster**. Nessa configuração, as tarefas (*tasks*) são realizadas no próprio *driver* localizado no **nó mestre**.

Para que o cluster possa ser criado, primeiramente é instalado o Java Runtime Environment (JRE) versão 8.

[ ]:
```
#instalando Java Runtime Environment (JRE) versão 8
%%capture
!apt-get remove openjdk*
!apt-get update --fix-missing
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
```

Na sequência, é feito o *download* do Apache Spark versão 3.0.0.

[ ]:
```
#baixando Apache Spark versão 3.0.0
%%capture
!wget -q https://archive.apache.org/dist/spark/spark-3.0.0/spark-3.0.
 ↪0-bin-hadoop2.7.tgz
!tar xf spark-3.0.0-bin-hadoop2.7.tgz && rm spark-3.0.0-bin-hadoop2.7.tgz
```

Na sequência, são configuradas as variáveis de ambiente JAVA_HOME e SPARK_HOME. Isto permite que tanto o Java quanto o Spark possam ser encontrados.

[ ]:
```
import os
#configurando a variável de ambiente JAVA_HOME
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
#configurando a variável de ambiente SPARK_HOME
os.environ["SPARK_HOME"] = "/content/spark-3.0.0-bin-hadoop2.7"
```

Por fim, são instalados dois pacotes da linguagem de programação Python, cujas funcionalidades são descritas a seguir.

> **Pacote findspark:** Usado para ler a variável de ambiente SPARK_HOME e armazenar seu valor na variável dinâmica de ambiente PYTHONPATH. Como resultado, Python pode encontrar a instalação do Spark.
>
> **Pacote pyspark:** PySpark é a API do Python para Spark. Ela possibilita o uso de Python, considerando que o *framework* Apache Spark encontra-se desenvolvido na linguagem de programação Scala.

[ ]:
```
%%capture
#instalando o pacote findspark
!pip install -q findspark==1.4.2
#instalando o pacote pyspark
!pip install -q pyspark==3.0.0
```

3.2 3.2 Conexão

PySpark não é adicionado ao *sys.path* por padrão. Isso significa que não é possível importá-lo, pois o interpretador da linguagem Python não sabe onde encontrá-lo.

Para resolver esse aspecto, é necessário instalar o módulo `findspark`. Esse módulo mostra onde PySpark está localizado. Os comandos a seguir têm essa finalidade.

[1]:
```
#importando o módulo findspark
import findspark
#carregando a variáveis SPARK_HOME na variável dinâmica PYTHONPATH
findspark.init()
```

Depois de configurados os pacotes e módulos e inicializadas as variáveis de ambiente, é possível iniciar o uso do Spark na aplicação de `data warehousing`. Para tanto, é necessário importar o comando `SparkSession` do módulo `pyspark.sql`. São utilizados os seguintes conceitos:

- `SparkSession`: permite a criação de `DataFrames`. Como resultado, as tabelas relacionais podem ser manipuladas por meio de `DataFrames` e é possível realizar consultas OLAP por meio de comandos SQL.
- `builder`: cria uma instância de SparkSession.
- `appName`: define um nome para a aplicação, o qual pode ser visto na interface de usuário web do Spark.
- `master`: define onde está o nó mestre do *cluster*. Como a aplicação é executada localmente e não em um *cluster*, indica-se isso pela *string* `local` seguida do parâmetro `[*]`. Ou seja, define-se que apenas núcleos locais são utilizados.
- `getOrCreate`: cria uma SparkSession. Caso ela já exista, retorna a instância existente.

Observação: A lista completa de todos os parâmetros que podem ser utilizados na inicialização do *cluster* pode ser encontrada neste link.

[2]:
```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("pyspark-notebook").master("local[*]").
 ↪getOrCreate()
```

4 4 Geração dos DataFrames em Spark da BI Solutions

Um `DataFrame` em Spark é equivalente a uma tabela relacional. Portanto, um `DataFrame` possui um esquema, uma ou mais linhas (ou tuplas) e uma ou mais colunas (ou atributos).

4.1 4.1 Criação dos DataFrames

[3]:
```
#criando os DataFrames em Spark
cargo = spark.read.csv(path="data/cargo.csv", header=True, sep=",")
cliente = spark.read.csv(path="data/cliente.csv", header=True, sep=",")
data = spark.read.csv(path="data/data.csv", header=True, sep=",")
equipe = spark.read.csv(path="data/equipe.csv", header=True, sep=",")
funcionario = spark.read.csv(path="data/funcionario.csv", header=True, sep=",")
negociacao = spark.read.csv(path="data/negociacao.csv", header=True, sep=",")
pagamento = spark.read.csv(path="data/pagamento.csv", header=True, sep=",")
```

4.2 4.2 Atualização dos Tipos de Dados

Nos comandos a seguir, primeiro são identificados quais colunas de quais `DataFrames` devem ser do tipo de dado inteiro. Na sequência, ocorre a conversão. Por fim, são exibidos os esquemas dos `DataFrames`, possibilitando visualizar a mudança de tipo de dados das colunas especificadas.

[4]:
```
# identificando quais colunas de quais DataFrames devem ser do tipo de dado␣
↪inteiro
colunas_cargo = ["cargoPK"]
colunas_cliente = ["clientePK"]
colunas_data = ["dataPk", "dataDia", "dataMes", "dataBimestre",␣
↪"dataTrimestre", "dataSemestre", "dataAno"]
colunas_equipe = ["equipePK"]
colunas_funcionario = ["funcPK", "funcDiaNascimento", "funcMesNascimento",␣
↪"funcAnoNascimento"]
colunas_negociacao = ["equipePK", "clientePK", "dataPK",␣
↪"quantidadeNegociacoes"]
colunas_pagamento = ["funcPK", "equipePK", "dataPK", "cargoPK",␣
↪"quantidadeLancamentos"]
```

[5]:
```
# importando o tipo de dado desejado
from pyspark.sql.types import IntegerType


# atualizando o tipo de dado das colunas especificadas
# substituindo as colunas já existentes

for coluna in colunas_cargo:
    cargo = cargo.withColumn(coluna, cargo[coluna].cast(IntegerType()))

for coluna in colunas_cliente:
    cliente = cliente.withColumn(coluna, cliente[coluna].cast(IntegerType()))

for coluna in colunas_data:
    data = data.withColumn(coluna, data[coluna].cast(IntegerType()))

for coluna in colunas_equipe:
```

```
    equipe = equipe.withColumn(coluna, equipe[coluna].cast(IntegerType()))

for coluna in colunas_funcionario:
    funcionario = funcionario.withColumn(coluna, funcionario[coluna].
 ↪cast(IntegerType()))

for coluna in colunas_negociacao:
    negociacao = negociacao.withColumn(coluna, negociacao[coluna].
 ↪cast(IntegerType()))

for coluna in colunas_pagamento:
    pagamento = pagamento.withColumn(coluna, pagamento[coluna].
 ↪cast(IntegerType()))
```

Nos comandos a seguir, primeiro são identificados quais colunas de quais `DataFrames` devem ser do tipo de dado número de ponto flutuante. Na sequência, ocorre a conversão. Por fim, são exibidos os esquemas dos `DataFrames`, possibilitando visualizar a mudança de tipo de dados das colunas especificadas.

[6]:
```
# identificando quais colunas de quais DataFrames devem ser do tipo de dado␣
 ↪número de ponto flutuante
colunas_negociacao = ["receita"]
colunas_pagamento = ["salario"]
```

[7]:
```
# importando o tipo de dado desejado
from pyspark.sql.types import FloatType


# atualizando o tipo de dado das colunas especificadas
# substituindo as colunas já existentes

for coluna in colunas_negociacao:
    negociacao = negociacao.withColumn(coluna, negociacao[coluna].
 ↪cast(FloatType()))

for coluna in colunas_pagamento:
    pagamento = pagamento.withColumn(coluna, pagamento[coluna].
 ↪cast(FloatType()))
```

4.3 4.3 Criação de Visões Temporárias

[8]:
```
#criando as visões temporárias
cargo.createOrReplaceTempView("cargo")
cliente.createOrReplaceTempView("cliente")
data.createOrReplaceTempView("data")
equipe.createOrReplaceTempView("equipe")
funcionario.createOrReplaceTempView("funcionario")
```

```
negociacao.createOrReplaceTempView("negociacao")
pagamento.createOrReplaceTempView("pagamento")
```

5 5 Instruções Importantes sobre a Avaliação

Esta avaliação é composta por 7 questões referentes a diferentes consultas OLAP. O valor de cada questão encontra-se especificado juntamente com a definição da questão.

5.1 5.1 Especificação das Consultas

As consultas OLAP devem ser respondidas de acordo com o solicitado em cada questão. As seguintes solicitações podem ser feitas:

- Resolva a questão especificando a consulta OLAP na **linguagem SQL**. Neste caso, a consulta deve ser respondida usando os conceitos apresentados na Aula 07 da disciplina. Ou seja, a consulta deve ser respondida usando a linguagem SQL textual e o método `spark.sql()`. Não é possível usar os demais métodos do módulo `pyspark.sql` para especificar a consulta, com exceção do método `show()` para listar o resultado da consulta.
- Resolva a questão especificando a consulta OLAP usando os **métodos de pyspark.sql**. Neste caso, a consulta deve ser respondida usando os conceitos apresentados na Aula 08 da disciplina. Ou seja, a consulta deve ser respondida usando os demais métodos do módulo `pyspark.sql`. Não é possível usar o método `spark.sql()` para especificar a consulta.

Caso a consulta seja especificada de forma diferente do que foi solicitado, a resposta não será considerada, mesmo que ela esteja correta.

5.2 5.2 Ordem das Colunas e das Linhas

A resolução das questões deve seguir estritamente as especificações definidas em cada consulta. Isto significa que:

- As **colunas** solicitadas devem ser exibidas exatamente na mesma ordem que a definida na questão. Note que todas as colunas a serem exibidas como resposta da consulta, bem como a ordem na qual elas devem aparecer são sempre definidas na questão.
- As **linhas** retornadas como respostas devem ser exibidas exatamente na mesma ordem que a definida na questão. Note que a ordem na qual as linhas devem aparecer são sempre definidas na questão.
- Os **nomes das colunas** renomeadas devem seguir estritamente os nomes definidos na questão. Para evitar possíveis erros, os nomes das colunas renomeadas não possuem acentos e espaços em branco, além de serem escritos utilizando apenas letras maiúsculas. Note que os nomes das colunas renomeadas são sempre definidos na questão.

Essas orientações devem ser seguidas uma vez que a correção da avaliação será realizada de forma automática. Caso a consulta retorne resultados de forma diferente do que foi solicitado, a resposta não será considerada, mesmo que ela esteja correta.

5.3 5.3 Listagem das Respostas das Consultas

A resposta de cada consulta deve ser listada usando o método `show()`. Nenhum outro método pode ser utilizado com essa finalidade.

Devem ser listadas apenas as `20` primeiras linhas de resposta de cada consulta. Adicionalmente, devem ser listadas *strings* com tamanho maior do que 20 caracteres, ou seja, o parâmetro `truncate` do método `show()` deve ser inicializado como `false`.

Portanto, a listagem das respostas deve ser feita utilizando o método `show()` como especificado a seguir.

- Quando a consulta OLAP for especificada usando a **linguagem SQL**. Utilize o comando `spark.sql(consultaSQL).show(20,truncate=False)` para exibir o resultado da consulta.
- Quando a consulta OLAP for especificada usando os demais **métodos de pyspark.sql**. Utilize o comando `nomeDoDataFrame.show(20,truncate=False)` para exibir o resultado da consulta.

Por padrão, o método `show()` exibe as `20` primeiras linhas. Mesmo assim, defina o valor `20` como parâmetro.

5.4 5.4 Arredondamento dos Dados

Deve ser realizado o arredondamento dos dados todas as vezes que uma função de agregação for aplicada às medidas numéricas `salario` da tabela de dimensão `pagamento` e `receita` da tabela de dimensão `negociacao`.

O arredondamento deve ser realizado usando a função `round()` na linguagem SQL e o método `round()` em `pyspark.sql` e deve arredondar os dados até duas casas decimais. Por exemplo, podem ser produzidos resultados da forma `112233.4` e `112233.44`.

Portanto, o arredondamento dos dados deve ser feito como especificado a seguir.

- Quando a consulta OLAP for especificada usando a **linguagem SQL**. Utilize a função `ROUND(funçãoDeAgregação,2)` para arredondar o dado até duas casas decimais.
- Quando a consulta OLAP for especificada usando os demais **métodos de pyspark.sql**. Utilize o método `round(funçãoDeAgregação,2)` para arredondar o dado até duas casas decimais.

5.5 5.5 Comentários Explicativos

Devem ser colocados comentários no código que expliquem o passo a passo da resolução da questão. Os comentários explicativos devem ser realizados como especificado a seguir.

- Quando a consulta OLAP for especificada usando a **linguagem SQL**. Utilize `#` para colocar comentários gerais (conforme explicado para os demais métodos de `pyspark.sql`) ou utilize `--` para colocar comentários no comando SQL. Por exemplo:

```
-- na cláusula SELECT são listadas as colunas a serem exibidas
SELECT funcNome
-- na cláusula FROM são especificadas as relações temporárias
FROM funcionario
-- na cláusula WHERE são definidas as condições de seleção
```

```
WHERE funcPK = 1
```

- Quando a consulta OLAP for especificada usando os demais **métodos de pyspark.sql**. Utilize # para colocar comentário. Por exemplo:

```
# no comando a seguir, é aplicado o método select() sobre o DataFrame
# para listar as colunas a serem exibidas, depois é aplicado o método
# filter() para listar as condições de seleção
```

5.6 5.6 Indentação e Organização

As consultas e os comandos que respondem às questões dessa avaliação devem ser escritos de forma indentada. Em caso de dúvida, observem os *notebooks* da Aula 07 e da Aula 08 e verifiquem como as consultas e os comandos foram indentados.

Com relação à organização, é necessário que as respostas às questões sejam localizadas aonde especificado no *notebook*. Por favor, procurem por "Resposta da Questão" para encontrar o local no qual as respostas devem ser especificadas. Também é possível localizar o local das respostas utilizando o menu de navegação.

5.7 5.7 Critério de Avaliação

Na correção da avaliação, serão ponderados os seguintes aspectos:

- Corretude da execução das consultas OLAP.
- Atendimento às especificações definidas nas seções 5.1, 5.2, 5.3 e 5.4.
- Atendimento às especificações da sintaxe das cláusulas e dos métodos utilizados para resolver cada questão.
- Qualidade da documentação entregue, de acordo com as especificações definidas nas seções 5.5 e 5.6.

6 6 Questões

O mercado de trabalho brasileiro usualmente mostra que as mulheres ainda não possuem o mesmo reconhecimento que os homens. Existem diversas pesquisas que mostram que as mulheres ganham menos que homens em todos os cargos, áreas de atuação e níveis de escolaridade. Além disso, mulheres ainda são minoria quando consideradas posições nos principais cargos de gestão. Adicionalmente, existem estudos que indicam que a participação feminina no mercado de trabalho brasileiro aumenta a produtividade.

O objetivo da avaliação é investigar se existe disparidade entre o sexo feminino e masculino na BI Solutions. São considerados os seguintes aspectos nas análises a serem realizadas: temporalidade e regionalidade.

Os resultados obtidos na avaliação poderão ser posteriormente utilizados para definir estratégias que a BI Solutions deve executar para resolver essa disparidade, caso necessário.

6.1 6.1 Visão Comparativa Relacionada aos Sexos

O objetivo das análises desta seção é obter uma visão relacionada aos sexos, por meio da comparação da média dos salários recebidos por mulheres e homens. Podem ser realizadas diferentes análises, sendo que duas delas são solicitadas a seguir.

6.1.1 Questão 1

(valor: 1,0) Liste, para cada `dataAno` e para cada sexo do funcionário, a média dos salários. Arredonde a média dos salários para até duas casas decimais. Devem ser exibidas as colunas na ordem e com os nomes especificados a seguir: "ANO", "SEXO" e "MEDIASALARIO". Ordene as linhas exibidas primeiro por ano e depois por sexo, todos em ordem ascendente. Liste as primeiras 20 linhas da resposta, sem truncamento das *strings*.

Resolva a questão especificando a consulta OLAP na linguagem SQL.

6.1.2 Resposta da Questão 1

[9]:
```
# Resposta da Questão 1
query = """
-- na cláusula SELECT são listadas as colunas a serem exibidas na resposta
SELECT
    dataAno AS ANO,
    funcSexo AS SEXO,
    ROUND(MEAN(salario),2) AS MEDIASALARIO
-- na cláusula FROM são especificadas as relações temporárias realizadas
-- entre data, pagamento e funcionario
FROM data JOIN pagamento ON data.dataPK = pagamento.dataPK
          JOIN funcionario ON funcionario.funcPK = pagamento.funcPK
-- na cláusula GROUP BY são especificadas as relações de agrupamento
GROUP BY ANO, SEXO
-- na cláusula ORDER BY são especificadas as relações de ordenação
ORDER BY ANO, SEXO
"""

spark.sql(query).show(20, truncate=False)
```

```
+----+----+------------+
|ANO |SEXO|MEDIASALARIO|
+----+----+------------+
|2016|F   |9169.65     |
|2016|M   |6906.46     |
|2017|F   |8336.19     |
|2017|M   |7131.79     |
|2018|F   |8334.27     |
|2018|M   |7605.94     |
|2019|F   |7585.33     |
|2019|M   |7789.65     |
|2020|F   |7585.33     |
```

```
|2020|M   |7789.65     |
+----+----+------------+
```

6.1.3 Questão 2

(valor: 1,0) Liste todas as agregações que podem ser geradas a partir da média dos salários dos funcionários por `dataAno` por `funcSexo` por `funcRegiaoNome`. Arredonde a média dos salários para até duas casas decimais. Devem ser exibidas as colunas na ordem e com os nomes especificados a seguir: "ANO", "SEXO", "REGIAO", "MEDIASALARIO". Ordene as linhas exibidas primeiro por ano, depois por sexo, depois por nome da região, todos em ordem ascendente. Liste as primeiras 20 linhas da resposta, sem truncamento das *strings*.

Resolva a questão especificando a consulta OLAP usando os métodos de pyspark.sql.

6.1.4 Resposta da Questão 2

[10]:
```
# Resposta da Questão 2

# Import do pyspark.sql.functions para manipulação correta do Dataframe
from pyspark.sql.functions import round

# no comando a seguir, é aplicado o método join() sobre os DataFrames
#     data,
#     pagamento (dataPK como condição) e
#     funcionario (funcPK como condição)
# para obtenção das informações corretas do problema proposto
df = data\
   .join(pagamento, ["dataPK"], "inner")\
   .join(funcionario, ["funcPK"], "inner")

# no comando a seguir, é aplicado o método groupBy() sobre o DataFrame
# para aplicar a operação de avg() na coluna de salario
df = df\
   .groupBy("dataAno", "funcSexo", "funcRegiaoNome")\
   .avg("salario")

# posteriormente o resultado de avg() é arredondado com o método
# round() na coluna MEDIASALARIO
df = df\
   .withColumn("MEDIASALARIO", round("avg(salario)",2))

# no comando a seguir, é aplicado o método withColumnRenamed() sobre
# o DataFrame para listar as colunas de acordo com o requisitado nas
# condições de seleção
df = df\
   .withColumnRenamed("dataAno", "ANO")\
   .withColumnRenamed("funcSexo", "SEXO")\
```

```
   .withColumnRenamed("funcRegiaoNome", "REGIAO")

# no comando a seguir, é aplicado o método select() sobre o DataFrame
# para listar as colunas a serem exibidas, depois é aplicado o método
# orderBy() para ordernar os resultados na condição especificada
df = df\
   .select(["ANO", "SEXO", "REGIAO", "MEDIASALARIO"])\
   .orderBy("ANO", "SEXO", "REGIAO")

# Os primeiros 20 resultados das operação são exibidos pelo método show()
# e sem truncamento das strings
df.show(20, truncate=False)
```

```
+----+----+--------+------------+
|ANO |SEXO|REGIAO  |MEDIASALARIO|
+----+----+--------+------------+
|2016|F   |SUDESTE |8587.4      |
|2016|F   |SUL     |14992.14    |
|2016|M   |NORDESTE|9351.25     |
|2016|M   |SUDESTE |5903.65     |
|2016|M   |SUL     |14343.41    |
|2017|F   |NORDESTE|1797.28     |
|2017|F   |SUDESTE |8575.92     |
|2017|F   |SUL     |8837.75     |
|2017|M   |NORDESTE|8484.34     |
|2017|M   |SUDESTE |6763.65     |
|2017|M   |SUL     |9169.53     |
|2018|F   |NORDESTE|5917.48     |
|2018|F   |SUDESTE |8333.57     |
|2018|F   |SUL     |9546.04     |
|2018|M   |NORDESTE|6920.48     |
|2018|M   |SUDESTE |7345.4      |
|2018|M   |SUL     |9540.1      |
|2019|F   |NORDESTE|7762.09     |
|2019|F   |SUDESTE |7329.74     |
|2019|F   |SUL     |8857.84     |
+----+----+--------+------------+
only showing top 20 rows
```

6.2 6.2 Visão Específica da Atuação Feminina

O objetivo das análises desta seção é obter uma visão direcionada especificamente à atuação feminina, considerando aspectos individuais referentes a salários e receitas. Podem ser realizadas diferentes análises, sendo que duas delas são solicitadas a seguir.

6.2.1 Questão 3

(valor 1,5) Liste, para cada `dataAno`, a soma dos salários das funcionárias do sexo feminino que nasceram entre os anos de 1970 (inclusive) e 1990 (inclusive) e que moram na região "SUDESTE" ("SE") ou "NORDESTE" ("NE"). Arredonde a soma dos salários para até duas casas decimais. Devem ser exibidas as colunas na ordem e com os nomes especificados a seguir: "ANO", "IDADE", "REGIAO" e "TOTALSALARIO". Ano corresponde ao atributo `dataAno` da tabela de dimensão `data`, idade corresponde ao cálculo feito considerando o ano atual de 2020 e o atributo `funcAnoNascimento` da tabela de dimensão `funcionario`, região corresponde ao atributo `funcRegiaoNome` da tabela de dimensão `funcionario`. Ordene as linhas exibidas primeiro por ano, depois por idade e depois por região, todos em ordem ascendente. Liste as primeiras 20 linhas da resposta, sem truncamento das *strings*.

Resolva a questão especificando a consulta OLAP na linguagem SQL.

6.2.2 Resposta da Questão 3

[11]:
```
# Resposta da Questão 3
query = """
-- na cláusula SELECT são listadas as colunas a serem exibidas na resposta
SELECT
    dataAno AS ANO,
    (2020 - funcAnoNascimento) AS IDADE,
    funcRegiaoNome AS REGIAO,
    ROUND(MEAN(salario),2) AS TOTALSALARIO
-- na cláusula FROM são especificadas as relações temporárias realizadas
-- entre data, pagamento e funcionario
FROM data JOIN pagamento ON data.dataPK = pagamento.dataPK
          JOIN funcionario ON funcionario.funcPK = pagamento.funcPK
-- na cláusula WHERE são definidas as condições de seleção:
--     funcionárias do sexo feminino
--     que nasceram entre os anos de 1970 (inclusive) e 1990 (inclusive)
--     e que moram na região "SUDESTE" ("SE") ou "NORDESTE" ("NE")
WHERE funcSexo = 'F'
      AND (funcAnoNascimento >= 1970)
      AND (funcAnoNascimento <= 1990)
      AND (funcRegiaoNome IN ('SUDESTE', 'NORDESTE') OR funcRegiaoSigla IN␣
 ↪('SE', 'NE'))
-- na cláusula GROUP BY são especificadas as relações de agrupamento
GROUP BY ANO, IDADE, REGIAO
-- na cláusula ORDER BY são especificadas as relações de ordenação
ORDER BY ANO, IDADE, REGIAO
"""

spark.sql(query).show(20, truncate=False)
```

```
+----+-----+--------+------------+
|ANO |IDADE|REGIAO  |TOTALSALARIO|
+----+-----+--------+------------+
```

```
|2016|30   |SUDESTE |7128.46     |
|2016|47   |SUDESTE |14363.61    |
|2017|30   |NORDESTE|1797.28     |
|2017|30   |SUDESTE |9240.66     |
|2017|46   |SUDESTE |4478.12     |
|2017|47   |SUDESTE |8174.24     |
|2018|30   |NORDESTE|2954.94     |
|2018|30   |SUDESTE |9211.81     |
|2018|35   |SUDESTE |9471.15     |
|2018|46   |SUDESTE |4478.12     |
|2018|47   |SUDESTE |8174.24     |
|2019|30   |NORDESTE|6946.0      |
|2019|30   |SUDESTE |7303.0      |
|2019|34   |SUDESTE |4978.8      |
|2019|35   |SUDESTE |9471.15     |
|2019|46   |SUDESTE |4478.12     |
|2019|47   |SUDESTE |8174.24     |
|2020|30   |NORDESTE|6946.0      |
|2020|30   |SUDESTE |7303.0      |
|2020|34   |SUDESTE |4978.8      |
+----+-----+--------+------------+
only showing top 20 rows
```

6.2.3 Questão 4

(valor 1,5) Considere que as equipes cujos valores de `equipePK` são iguais a `1, 3 e 5` possuem a maior quantidade de funcionárias do sexo feminino. Liste, para cada `dataAno`, a soma das receitas recebidas por essas equipes, o nome da equipe, o nome da filial e o setor do cliente, considerando apenas os clientes localizados na cidade de "SAO CARLOS". Arredonde a média dos salários para até duas casas decimais. Devem ser exibidas as colunas na ordem e com os nomes especificados a seguir: "ANO", "NOMEEQUIPE", "NOMEFILIAL", "SETORCLIENTE", "TOTALRECEITA". Ordene as linhas exibidas primeiro por ano, depois por nome da equipe, depois por nome da filial e depois por setor do cliente, todos em ordem ascendente. Liste as primeiras 20 linhas da resposta, sem truncamento das *strings*.

Resolva a questão especificando a consulta usando os métodos de pyspark.sql.

6.2.4 Resposta da Questão 4

[12]:
```
# Reposta da Questão 4

# Import do pyspark.sql.functions para manipulação correta do Dataframe
from pyspark.sql.functions import round

# no comando a seguir, é aplicado o método join() sobre os DataFrames
#     data,
#     negociacao (dataPK como condição),
```

```
#     equipe (equipePK como condição) e
#     cliente (clientePK como condição)
# para obtenção das informações corretas do problema proposto
df = data\
   .join(negociacao, ["dataPK"], "inner")\
   .join(equipe, ["equipePK"], "inner")\
   .join(cliente, ["clientePK"], "inner")

# no comando a seguir, é aplicado o método filter() sobre o DataFrame
# para considerar as equipes cujos valores de equipePK são iguais a 1, 3 e 5
# pois possuem a maior quantidade de funcionárias do sexo feminino
# e filtrar os clientes localizados na cidade de "SAO CARLOS"
df = df\
   .filter(df.equipePK.isin([1, 3, 5]) & df.clienteCidade.like("%SAO CARLOS%"))

# no comando a seguir, é aplicado o método groupBy() sobre o DataFrame
# para aplicar a operação de avg() na coluna de receita
df = df\
   .groupBy("dataAno", "equipeNome", "filialNome", "clienteSetor")\
   .avg("receita")

# posteriormente o resultado de avg() é arredondado com o método
# round() na coluna TOTALRECEITA
df = df\
   .withColumn("TOTALRECEITA", round("avg(receita)",2))

# no comando a seguir, é aplicado o método withColumnRenamed() sobre
# o DataFrame para listar as colunas de acordo com o requisitado nas
# condições de seleção
df = df\
   .withColumnRenamed("dataAno", "ANO")\
   .withColumnRenamed("equipeNome", "NOMEEQUIPE")\
   .withColumnRenamed("filialNome", "NOMEFILIAL")\
   .withColumnRenamed("clienteSetor", "SETORCLIENTE")

# no comando a seguir, é aplicado o método select() sobre o DataFrame
# para listar as colunas a serem exibidas, depois é aplicado o método
# orderBy() para ordernar os resultados na condição especificada
df = df\
   .select(["ANO", "NOMEEQUIPE", "NOMEFILIAL", "SETORCLIENTE", "TOTALRECEITA"])\
   .orderBy("ANO", "NOMEEQUIPE", "NOMEFILIAL", "SETORCLIENTE")

# Os primeiros 20 resultados das operação são exibidos pelo método show()
# e sem truncamento das strings
df.show(20, truncate=False)
```

```
+----+------------+----------------------+------------------+------------+
|ANO |NOMEEQUIPE  |NOMEFILIAL            |SETORCLIENTE      |TOTALRECEITA|
```

```
+----+------------+----------------------+------------------+------------+
|2016|APP - DESKTOP|SAO PAULO - AV. PAULISTA|BEBIDAS E ALIMENTOS|41101.9     |
|2016|APP - DESKTOP|SAO PAULO - AV. PAULISTA|VESTUARIO          |17752.5     |
|2017|APP - DESKTOP|SAO PAULO - AV. PAULISTA|BEBIDAS E ALIMENTOS|5628.17     |
|2017|APP - DESKTOP|SAO PAULO - AV. PAULISTA|VESTUARIO          |20605.9     |
|2017|WEB          |CAMPO GRANDE - CENTRO   |TECNOLOGIA         |19317.8     |
|2017|WEB          |CAMPO GRANDE - CENTRO   |VESTUARIO          |4091.82     |
|2017|WEB          |SAO PAULO - AV. PAULISTA|TECNOLOGIA         |1200.92     |
|2017|WEB          |SAO PAULO - AV. PAULISTA|VESTUARIO          |12604.38    |
|2018|APP - DESKTOP|SAO PAULO - AV. PAULISTA|BEBIDAS E ALIMENTOS|12383.8     |
|2018|APP - DESKTOP|SAO PAULO - AV. PAULISTA|VESTUARIO          |38021.57    |
|2018|WEB          |CAMPO GRANDE - CENTRO   |TECNOLOGIA         |26722.1     |
|2018|WEB          |CAMPO GRANDE - CENTRO   |VESTUARIO          |970.1       |
|2018|WEB          |SAO PAULO - AV. PAULISTA|TECNOLOGIA         |10364.28    |
|2018|WEB          |SAO PAULO - AV. PAULISTA|VESTUARIO          |8489.97     |
|2019|APP - DESKTOP|SAO PAULO - AV. PAULISTA|BEBIDAS E ALIMENTOS|12636.63    |
|2019|APP - DESKTOP|SAO PAULO - AV. PAULISTA|VESTUARIO          |10899.83    |
|2019|WEB          |CAMPO GRANDE - CENTRO   |TECNOLOGIA         |4260.73     |
|2019|WEB          |CAMPO GRANDE - CENTRO   |VESTUARIO          |514.35      |
|2019|WEB          |SAO PAULO - AV. PAULISTA|TECNOLOGIA         |5378.83     |
|2019|WEB          |SAO PAULO - AV. PAULISTA|VESTUARIO          |1201.05     |
+----+------------+----------------------+------------------+------------+
only showing top 20 rows
```

6.3 6.3 Visão Geral da Atuação Feminina

O objetivo das análises desta seção é obter uma visão direcionada especificamente à atuação feminina, considerando aspectos conjuntos referentes a salários e receitas. Podem ser realizadas diferentes análises, dentre as quais destaca-se a análise base descrita a seguir.

6.3.1 Análise Base

Liste, para cada `dataAno`, a soma dos salários das funcionárias de sexo feminino que moram no estado do "RIO DE JANEIRO" ("RJ") e as somas das receitas recebidas pelas equipes localizadas no estado do "RIO DE JANEIRO" ("RJ"). O estado no qual as funcionárias moram pode ser identificado pelos atributos `funcEstadoNome` ou `funcEstadoSigla` da tabela de dimensão `funcionario`, enquanto que o estado nos quais as equipes estão localizadas pode ser identificado pelos atributos `filialEstadoNome` ou `filialEstadoSigla` da tabela de dimensão `equipe`. Arredonde a soma dos salários e a soma das receitas para até duas casas decimais. Devem ser exibidas as colunas na ordem e com os nomes especificados a seguir: "ANO", "TOTALSALARIO", "TOTALRECEITA". Ordene as linhas exibidas por ano em ordem ascendente. Liste as primeiras 20 linhas da resposta, sem truncamento das *strings*.

6.3.2 Questão 5

(valor: 1,5) Resolva a "Análise Base" especificando a consulta OLAP na linguagem SQL.

6.3.3 Resposta da Questão 5

[13]:
```
# Resposta da Questão 5
query = """
-- na cláusula SELECT são listadas as colunas a serem exibidas na resposta
SELECT
    funcionarias_salario_ano.ANO,
    TOTALSALARIO,
    TOTALRECEITA
-- na cláusula FROM são especificadas as relações temporárias realizadas
FROM (
    -- a soma dos salários das funcionárias de sexo feminino que moram no␣
 ↪estado do "RIO DE JANEIRO" ("RJ")
    SELECT
        dataAno AS ANO,
        ROUND(SUM(salario), 2) AS TOTALSALARIO
    FROM data JOIN pagamento ON data.dataPK = pagamento.dataPK
              JOIN funcionario ON funcionario.funcPK = pagamento.funcPK
    -- na cláusula WHERE são definidas as condições de seleção:
    --     funcionárias do sexo feminino
    --     que moram no estado do "RIO DE JANEIRO" ("RJ")
    WHERE funcSexo = 'F'
          AND (funcEstadoNome LIKE 'RIO DE JANEIRO' OR funcEstadoSigla LIKE␣
 ↪'RJ')
    GROUP BY ANO
    ORDER BY ANO
    ) AS funcionarias_salario_ano,
    (
    -- somas das receitas recebidas pelas equipes localizadas no estado do "RIO␣
 ↪DE JANEIRO" ("RJ")
    SELECT
        dataAno AS ANO,
        ROUND(SUM(receita), 2) AS TOTALRECEITA
    FROM data JOIN negociacao ON data.dataPK = negociacao.dataPK
              JOIN equipe ON negociacao.equipePK = equipe.equipePK
    -- na cláusula WHERE são definidas as condições de seleção:
    --     equipes localizadas no estado do "RIO DE JANEIRO" ("RJ")
    WHERE (filialEstadoNome LIKE 'RIO DE JANEIRO' OR filialEstadoSigla LIKE␣
 ↪'RJ')
    GROUP BY ANO
    ORDER BY ANO
    ) AS equipes_receita_ano
-- na cláusula WHERE são definidas as condições de seleção
WHERE funcionarias_salario_ano.ANO = equipes_receita_ano.ANO
-- na cláusula ORDER BY são especificadas as relações de ordenação
ORDER BY ANO
"""
```

```
spark.sql(query).show(20, truncate=False)
```

```
+----+-----------+------------+
|ANO |TOTALSALARIO|TOTALRECEITA|
+----+-----------+------------+
|2016|30061.2     |2205042.91  |
|2017|30061.2     |3484981.8   |
|2018|48108.36    |4741199.75  |
|2019|70794.36    |5100984.61  |
|2020|70794.36    |4192420.2   |
+----+-----------+------------+
```

6.3.4 Questão 6

(valor: 1,5) Resolva a "Análise Base" especificando a consulta OLAP usando os métodos de pyspark.sql.

6.3.5 Resposta da Questão 6

6.3.6 Análise Base

Liste, para cada `dataAno`, a soma dos salários das funcionárias de sexo feminino que moram no estado do "RIO DE JANEIRO" ("RJ") e as somas das receitas recebidas pelas equipes localizadas no estado do "RIO DE JANEIRO" ("RJ"). O estado no qual as funcionárias moram pode ser identificado pelos atributos `funcEstadoNome` ou `funcEstadoSigla` da tabela de dimensão `funcionario`, enquanto que o estado nos quais as equipes estão localizadas pode ser identificado pelos atributos `filialEstadoNome` ou `filialEstadoSigla` da tabela de dimensão `equipe`. Arredonde a soma dos salários e a soma das receitas para até duas casas decimais. Devem ser exibidas as colunas na ordem e com os nomes especificados a seguir: "ANO", "TOTALSALARIO", "TOTALRECEITA". Ordene as linhas exibidas por ano em ordem ascendente. Liste as primeiras 20 linhas da resposta, sem truncamento das *strings*.

[14]:
```
# Resposta da Questão 6

# Import do pyspark.sql.functions para manipulação correta do Dataframe
from pyspark.sql.functions import round
```

[15]:
```
#
# Parte 1 - Receitas recebidas pelas equipes localizadas
# no estado do "RIO DE JANEIRO" ("RJ")
#
#
# no comando a seguir, é aplicado o método join() sobre os DataFrames
#     data,
#     negociacao (dataPK como condição),
#     equipe (equipePK como condição) e
```

```
# para obtenção das informações corretas do problema proposto
df_1 = data\
   .join(negociacao, ["dataPK"], "inner")\
   .join(equipe, ["equipePK"], "inner")

# no comando a seguir, é aplicado o método filter() sobre o DataFrame
# para considerar as equipes localizadas no estado do "RIO DE JANEIRO" ("RJ")
df_1 = df_1\
   .filter(df_1.filialEstadoNome.like("RIO DE JANEIRO") | df_1.
 ↪filialEstadoSigla.like("RJ"))

# no comando a seguir, é aplicado o método groupBy() sobre o DataFrame
# para aplicar a operação de sum() na coluna de receita
df_1 = df_1\
   .groupBy("dataAno")\
   .sum("receita")

# posteriormente o resultado de sum() é arredondado com o método
# round() na coluna TOTALRECEITA
df_1 = df_1\
   .withColumn("TOTALRECEITA", round("sum(receita)",2))

# no comando a seguir, é aplicado o método withColumnRenamed() sobre
# o DataFrame para listar as colunas de acordo com o requisitado nas
# condições de seleção
df_1 = df_1\
   .withColumnRenamed("dataAno", "ANO")

# no comando a seguir, é aplicado o método select() sobre o DataFrame
# para listar as colunas a serem exibidas
df_1 = df_1\
   .select(["ANO", "TOTALRECEITA"])
```

[16]:
```
#
# Parte 2 - Soma dos salários das funcionárias de sexo feminino
# que moram no estado do "RIO DE JANEIRO" ("RJ")
#
#
# no comando a seguir, é aplicado o método join() sobre os DataFrames
#     data,
#     pagamento (dataPK como condição) e
#     funcionario (funcPK como condição)
# para obtenção das informações corretas do problema proposto
df_2 = data\
   .join(pagamento, ["dataPK"], "inner")\
   .join(funcionario, ["funcPK"], "inner")
```

```
# no comando a seguir, é aplicado o método filter() sobre o DataFrame
# para considerar as funcionárias de sexo feminino  que moram no
# estado do "RIO DE JANEIRO" ("RJ")
df_2 = df_2\
   .filter(df_2.funcSexo.like("F") & (df_2.funcEstadoNome.like("RIO DE␣
 ↪JANEIRO") | df_2.funcEstadoSigla.like("RJ")))

# no comando a seguir, é aplicado o método groupBy() sobre o DataFrame
# para aplicar a operação de sum() na coluna de salario
df_2 = df_2\
   .groupBy("dataAno")\
   .sum("salario")

# posteriormente o resultado de sum() é arredondado com o método
# round() na coluna MEDIASALARIO
df_2 = df_2\
   .withColumn("TOTALSALARIO", round("sum(salario)",2))

# no comando a seguir, é aplicado o método withColumnRenamed() sobre
# o DataFrame para listar as colunas de acordo com o requisitado nas
# condições de seleção
df_2 = df_2\
   .withColumnRenamed("dataAno", "ANO")

# no comando a seguir, é aplicado o método select() sobre o DataFrame
# para listar as colunas a serem exibidas
df_2 = df_2\
   .select(["ANO", "TOTALSALARIO"])
```

[17]:
```
# no comando a seguir, é aplicado o método join() sobre os DataFrames
#     df_1 (parte 1) e
#     df_2 (parte 2)
df = df_1\
   .join(df_2, ["ANO"])

# no comando a seguir, é aplicado o método select() sobre o DataFrame
# para listar as colunas a serem exibidas, depois é aplicado o método
# orderBy() para ordernar os resultados na condição especificada
df = df\
   .select(["ANO", "TOTALSALARIO", "TOTALRECEITA"])\
   .orderBy("ANO")

# Os primeiros 20 resultados das operação são exibidos pelo método show()
# e sem truncamento das strings
df.show(20, truncate=False)
```

```
+----+------------+------------+
|ANO |TOTALSALARIO|TOTALRECEITA|
```

```
+----+-----------+-----------+
|2016|30061.2    |2205042.91 |
|2017|30061.2    |3484981.8  |
|2018|48108.36   |4741199.75 |
|2019|70794.36   |5100984.61 |
|2020|70794.36   |4192420.2  |
+----+-----------+-----------+
```

6.4 6.4 Visão Comparativa Final

O objetivo da análise desta seção é obter uma visão relacionada aos sexos, por meio da comparação do total anual de gastos em salários para o pagamento das mulheres e dos homens em comparação ao total anual de receitas recebidas.

6.4.1 Questão 7

(valor 2,0) Liste, para cada `dataAno`, a soma dos salários das funcionárias de sexo feminino, a soma dos salários dos funcionários do sexo masculino e as somas das receitas recebidas. Arredonde a soma dos salários e a soma das receitas para até duas casas decimais. Devem ser exibidas as colunas na ordem e com os nomes especificados a seguir: "ANO", "TOTALSALARIOMULHERES", "TOTALSALARIOHOMENS", "TOTALRECEITA". Ordene as linhas exibidas por ano em ordem ascendente. Liste as primeiras 20 linhas da resposta, sem truncamento das *strings*.

Resolva a questão especificando a consulta OLAP na linguagem SQL.

6.4.2 Resposta da Questão 7

[18]:
```sql
# Resposta da Questão 7
query = """
-- na cláusula SELECT são listadas as colunas a serem exibidas na resposta
SELECT
    funcionarias_salario_ano.ANO,
    TOTALSALARIOMULHERES,
    TOTALSALARIOHOMENS,
    TOTALRECEITA
-- na cláusula FROM são especificadas as relações temporárias realizadas
FROM (
    -- a soma dos salários das funcionárias de sexo feminino
    SELECT
        dataAno AS ANO,
        ROUND(SUM(salario), 2) AS TOTALSALARIOMULHERES
    FROM data JOIN pagamento ON data.dataPK = pagamento.dataPK
              JOIN funcionario ON funcionario.funcPK = pagamento.funcPK
    -- na cláusula WHERE são definidas as condições de seleção:
    --     funcionárias do sexo feminino
    WHERE funcSexo = 'F'
    GROUP BY ANO
    ORDER BY ANO
```

```
    ) AS funcionarias_salario_ano,
    (
    -- a soma dos salários dos funcionários de sexo masculino
    SELECT
        dataAno AS ANO,
        ROUND(SUM(salario), 2) AS TOTALSALARIOHOMENS
    FROM data JOIN pagamento ON data.dataPK = pagamento.dataPK
              JOIN funcionario ON funcionario.funcPK = pagamento.funcPK
    -- na cláusula WHERE são definidas as condições de seleção:
    --     funcionárias do sexo masculino
    WHERE funcSexo = 'M'
    GROUP BY ANO
    ORDER BY ANO
    ) AS funcionarios_salario_ano,
    (
    -- somas das receitas recebidas pelas equipes
    SELECT
        dataAno AS ANO,
        ROUND(SUM(receita), 2) AS TOTALRECEITA
    FROM data JOIN negociacao ON data.dataPK = negociacao.dataPK
              JOIN equipe ON negociacao.equipePK = equipe.equipePK
    GROUP BY ANO
    ORDER BY ANO
    ) AS equipes_receita_ano
-- na cláusula WHERE são definidas as condições de seleção
WHERE funcionarias_salario_ano.ANO = funcionarios_salario_ano.ANO
      AND funcionarias_salario_ano.ANO = equipes_receita_ano.ANO
-- na cláusula ORDER BY são especificadas as relações de ordenação
ORDER BY ANO
"""


spark.sql(query).show(20, truncate=False)
```

```
+----+--------------------+------------------+-------------+
|ANO |TOTALSALARIOMULHERES|TOTALSALARIOHOMENS|TOTALRECEITA |
+----+--------------------+------------------+-------------+
|2016|1210393.21          |3232223.89        |4614246.97   |
|2017|2500857.97          |7274421.87        |7200423.35   |
|2018|3800427.49          |1.113509898E7     |1.159353966E7|
|2019|4733247.25          |1.38344192E7      |3.535331833E7|
|2020|4733247.25          |1.38344192E7      |3.022217587E7|
+----+--------------------+------------------+-------------+
```