

AVALIACAO_FINAL_KATIA

November 26, 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!

3 1 Constelação de Fatos da BI Solutions

A aplicação de *data warehousing* da BI Solutions utiliza como base uma constelação 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)

4 2 Obtenção dos Dados da BI Solutions

4.1 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.

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

4.2 2.2 Obtenção dos Dados das Tabelas de Dimensão

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

```
[2]: #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]: 'data/cliente.csv'
```

4.3 2.3 Obtenção dos Dados Tabelas de Fatos

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

```
[3]: #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]: 'data/negociacao.csv'
```

5 3 Apache Spark Cluster

5.1 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.

```
[4]: #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.

```
[5]: #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.

```
[6]: 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"] = "./spark-3.0.0-bin-hadoop2.7"
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.

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

5.2 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.

```
[8]: #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](#).

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

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

6 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).

6.1 4.1 Criação dos DataFrames

```
[10]: #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=",")
```

6.2 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.

```
[11]: # 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"]
```

```
[12]: # 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.

```

[13]: # identificando quais colunas de quais DataFrames devem ser do tipo de dado
    ↳ número de ponto flutuante
colunas_negociacao = ["receita"]
colunas_pagamento = ["salario"]

```

```

[14]: # 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()))

```

6.3 4.3 Criação de Visões Temporárias

```
[15]: #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")
```

7 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.

7.1 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.

7.2 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.

7.3 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.

7.4 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.

7.5 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
```

7.6 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.

7.7 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.

8 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.

8.1 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.

8.1.1 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.

8.1.2 Resposta da Questão 1

```
[16]: # Resposta da Questão 1
query = """
SELECT
    dataAno as ANO
,   funcSexo as SEXO
,   ROUND(MEAN(salario),2) AS MEDIASALARIO

FROM
    funcionario

    JOIN pagamento
    ON pagamento.funcPK = funcionario.funcPK

    JOIN data
    ON data.dataPK = pagamento.dataPK

GROUP BY ANO, SEXO
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	
+-----+-----+-----+-----+			

8.1.3 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`.

8.1.4 Resposta da Questão 2

```
[17]: # IMPORTACAO DA BIBLIOTECA
from pyspark.sql.functions import round

df = funcionario\
    .join(pagamento, ["funcPK"], "inner")\
    .join(data, ["dataPK"], "inner")

# AGRUPAMENTO DOS VALORES
df = df\
    .groupBy("dataAno", "funcSexo", "funcRegiaoNome")\
    .avg("salario")

df = df\
    .withColumn("MEDIASALARIO", round("avg(salario)",2))

# ALTERAÇÃO DO NOME DAS COLUNAS
df = df\
    .withColumnRenamed("dataAno", "ANO")\
    .withColumnRenamed("funcSexo", "SEXO")\
    .withColumnRenamed("funcRegiaoNome", "REGIAO")
```

```
# SELECAO DAS COLUNAS NECESSARIAS
# ORDENACAO DOS RESULTADOS
df = df\
    .select(["ANO", "SEXO", "REGIAO", "MEDIASALARIO"])\
    .orderBy("ANO", "SEXO", "REGIAO")

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
```

8.2 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.

8.2.1 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.

8.2.2 Resposta da Questão 3

```
[18]: # Resposta da Questão 3
query = """
SELECT
    dataAno AS ANO
,    (2020 - funcAnoNascimento) AS IDADE
,    funcRegiaoNome AS REGIAO
,    ROUND(SUM(salario), 2) AS TOTALSALARIO

FROM
    funcionario

    JOIN pagamento
    ON pagamento.funcPK = funcionario.funcPK

    JOIN data
    ON data.dataPK = pagamento.dataPK

WHERE
    funcionario.funcSexo = 'F'
AND funcionario.funcAnoNascimento BETWEEN 1970 AND 1990
AND (
    funcionario.funcRegiaoNome IN ('SUDESTE', 'NORDESTE')
    OR
    funcionario.funcRegiaoSigla IN ('SE', 'NE')
)

GROUP BY
    ANO, IDADE, REGIAO

ORDER BY
    ANO, IDADE, REGIAO
"""

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

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

2016 30	SUDESTE	342165.96	
2016 47	SUDESTE	172363.32	
2017 30	NORDESTE	21567.36	
2017 30	SUDESTE	1330655.03	
2017 46	SUDESTE	53737.44	
2017 47	SUDESTE	196181.64	
2018 30	NORDESTE	70918.68	
2018 30	SUDESTE	1768666.66	
2018 35	SUDESTE	113653.8	
2018 46	SUDESTE	53737.44	
2018 47	SUDESTE	196181.64	
2019 30	NORDESTE	416759.88	
2019 30	SUDESTE	1927991.98	
2019 34	SUDESTE	59745.6	
2019 35	SUDESTE	113653.8	
2019 46	SUDESTE	53737.44	
2019 47	SUDESTE	196181.64	
2020 30	NORDESTE	416759.88	
2020 30	SUDESTE	1927991.98	
2020 34	SUDESTE	59745.6	

+-----+-----+-----+-----+
only showing top 20 rows

8.2.3 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`.

8.2.4 Resposta da Questão 4

```
[19]: from pyspark.sql.functions import round

df = data\
    .join(negociacao, ["dataPK"], "inner")\
    .join(equipe, ["equipePK"], "inner")\
    .join(cliente, ["clientePK"], "inner")

df = df\
```

```

.filter(df.equipePK.isin([1, 3, 5]) & df.clienteCidade.like("%SAO CARLOS%"))

df = df\
    .groupBy("dataAno", "equipeNome", "filialNome", "clienteSetor")\
    .sum("receita")

df = df\
    .withColumn("TOTALRECEITA", round("sum(receita)",2))

df = df\
    .withColumnRenamed("dataAno", "ANO")\
    .withColumnRenamed("equipeNome", "NOMEEQUIPE")\
    .withColumnRenamed("filialNome", "NOMEFILIAL")\
    .withColumnRenamed("clienteSetor", "SETORCLIENTE")

df = df\
    .select(["ANO", "NOMEEQUIPE", "NOMEFILIAL", "SETORCLIENTE",
    ↪ "TOTALRECEITA"])\
    .orderBy("ANO", "NOMEEQUIPE", "NOMEFILIAL", "SETORCLIENTE")

df.show(20, truncate=False)

```

ANO	NOMEEQUIPE	NOMEFILIAL	SETORCLIENTE	TOTALRECEITA
2016	APP - DESKTOP	SAO PAULO - AV. PAULISTA	BEBIDAS E ALIMENTOS	82203.8
2016	APP - DESKTOP	SAO PAULO - AV. PAULISTA	VESTUARIO	71010.0
2017	APP - DESKTOP	SAO PAULO - AV. PAULISTA	BEBIDAS E ALIMENTOS	11256.35
2017	APP - DESKTOP	SAO PAULO - AV. PAULISTA	VESTUARIO	103029.5
2017	WEB	CAMPO GRANDE - CENTRO	TECNOLOGIA	57953.4
2017	WEB	CAMPO GRANDE - CENTRO	VESTUARIO	12275.45
2017	WEB	SAO PAULO - AV. PAULISTA	TECNOLOGIA	3602.75
2017	WEB	SAO PAULO - AV. PAULISTA	VESTUARIO	37813.15
2018	APP - DESKTOP	SAO PAULO - AV. PAULISTA	BEBIDAS E ALIMENTOS	12383.8
2018	APP - DESKTOP	SAO PAULO - AV. PAULISTA	VESTUARIO	76043.15
2018	WEB	CAMPO GRANDE - CENTRO	TECNOLOGIA	26722.1
2018	WEB	CAMPO GRANDE - CENTRO	VESTUARIO	970.1
2018	WEB	SAO PAULO - AV. PAULISTA	TECNOLOGIA	31092.85
2018	WEB	SAO PAULO - AV. PAULISTA	VESTUARIO	16979.95
2019	APP - DESKTOP	SAO PAULO - AV. PAULISTA	BEBIDAS E ALIMENTOS	37909.9
2019	APP - DESKTOP	SAO PAULO - AV. PAULISTA	VESTUARIO	32699.5
2019	WEB	CAMPO GRANDE - CENTRO	TECNOLOGIA	8521.45
2019	WEB	CAMPO GRANDE - CENTRO	VESTUARIO	514.35
2019	WEB	SAO PAULO - AV. PAULISTA	TECNOLOGIA	16136.5
2019	WEB	SAO PAULO - AV. PAULISTA	VESTUARIO	1201.05

only showing top 20 rows

8.3 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.

8.3.1 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*.

8.3.2 Questão 5

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

8.3.3 Resposta da Questão 5

```
[20]: query = """
SELECT
    SALARIO_FUNCIONARIA.ANO
,   SALARIO_FUNCIONARIA.TOTALSALARIO
,   RECEITA_EQUIPE.TOTALRECEITA

FROM
    (
        SELECT
            dataAno AS ANO
        ,   ROUND(SUM(salario), 2) AS TOTALSALARIO
        FROM
            funcionario

        JOIN pagamento
        ON pagamento.funcPK = funcionario.funcPK

        JOIN data
        ON data.dataPK = pagamento.dataPK

        WHERE
```



```

        funcionario.funcSexo = 'F'
    AND (funcionario.funcEstadoNome = 'RIO DE JANEIRO'
        OR funcionario.funcEstadoSigla = 'RJ')

    GROUP BY ANO

    ORDER BY ANO
) AS SALARIO_FUNCIONARIA,

(
    SELECT
        dataAno AS ANO
    ,   ROUND(SUM(receita), 2) AS TOTALRECEITA
    FROM
        equipe

        JOIN negociacao
        ON negociacao.equipePK = equipe.equipePK

        JOIN data
        ON data.dataPK = negociacao.dataPK

    WHERE

        equipe.filialEstadoNome = 'RIO DE JANEIRO'
    OR  equipe.filialEstadoSigla = 'RJ'

    GROUP BY ANO
    ORDER BY ANO
) AS RECEITA_EQUIPE

WHERE
    SALARIO_FUNCIONARIA.ANO = RECEITA_EQUIPE.ANO

ORDER BY SALARIO_FUNCIONARIA.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   |

```

+----+-----+-----+-----+

8.3.4 Questão 6

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

8.3.5 Resposta da Questão 6

8.3.6 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*.

```
[21]: from pyspark.sql.functions import round
```

```
[22]: receita = equipe\
        .join(negociacao, ['equipePK'], 'inner')\
        .join(data, ["dataPK"], 'inner')

receita = receita\
        .filter(receita.filialEstadoNome.like('RIO DE JANEIRO') |
        ↪receita.filialEstadoSigla.like('RJ'))

receita = receita\
        .groupBy("dataAno")\
        .sum("receita")

receita = receita\
        .withColumn("TOTALRECEITA", round("sum(receita)", 2))\
        .withColumnRenamed("dataAno", "ANO")

receita = receita.select(["ANO", "TOTALRECEITA"]).orderBy("ANO")

receita.show(20, truncate=False)
```

```
+----+-----+
|ANO |TOTALRECEITA|
+----+-----+
|2016|2205042.91  |
```

```
|2017|3484981.8 |
|2018|4741199.75 |
|2019|5100984.61 |
|2020|4192420.2 |
+----+-----+
```

```
[23]: salario = funcionario\
        .join(pagamento, ['funcPK'], 'inner')\
        .join(data, ["dataPK"], 'inner')

salario = salario\
        .filter(salario.funcSexo.like('F') &\
                (salario.funcEstadoNome.like('RIO DE JANEIRO') |\
                 salario.funcEstadoSigla.like('RJ'))))

salario = salario\
        .groupBy("dataAno")\
        .sum("salario")

salario = salario\
        .withColumn("TOTALSALARIO", round("sum(salario)", 2))\
        .withColumnRenamed("dataAno", "ANO")

salario = salario.select(["ANO", "TOTALSALARIO"]).orderBy("ANO")

salario.show(20, truncate=False)
```

```
+----+-----+
|ANO |TOTALSALARIO|
+----+-----+
|2016|30061.2      |
|2017|30061.2      |
|2018|48108.36     |
|2019|70794.36     |
|2020|70794.36     |
+----+-----+
```

```
[24]: # DATAFRAME BASE COM AS JUNCOES DAS BASES DE GERADAS
receita_salario = receita\
        .join(salario, ["ANO"])

# SELECAO DAS COLUNAS NECESSARIAS
# ORDENACAO DOS RESULTADOS
receita_salario = receita_salario\
        .select(["ANO", "TOTALSALARIO", "TOTALRECEITA"])\
```

```
.orderBy("ANO")

receita_salario.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   |
+-----+-----+-----+
```

8.4 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.

8.4.1 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.

8.4.2 Resposta da Questão 7

```
[25]: # Resposta da Questão 7
query = """
SELECT
    SEXO_SALARIO.ANO
,   SEXO_SALARIO.TOTALSALARIOMULHERES
,   SEXO_SALARIO.TOTALSALARIOHOMENS
,   EQUIPE_RECEITA.TOTALRECEITA
FROM
    (
        SELECT
            dataAno AS ANO
        ,   ROUND(SUM(
                CASE
```

```

        WHEN funcionario.funcSexo == 'F' THEN salario
        ELSE 0
    END
), 2) AS TOTALSALARIOMULHERES
,   ROUND(SUM(
        CASE
            WHEN funcionario.funcSexo == 'M' THEN salario
            ELSE 0
        END
    ), 2) AS TOTALSALARIOHOMENS

FROM
    funcionario

    JOIN pagamento
    ON pagamento.funcPK = funcionario.funcPK

    JOIN data
    ON data.dataPK = pagamento.dataPK

GROUP BY ANO
ORDER BY ANO
) AS SEXO_SALARIO,
(
    SELECT
        dataAno AS ANO
        ,   ROUND(SUM(receita), 2) AS TOTALRECEITA

FROM
    equipe

    JOIN negociacao
    ON negociacao.equipePK = equipe.equipePK

    JOIN data
    ON data.dataPK = negociacao.dataPK

GROUP BY ANO
ORDER BY ANO
) AS EQUIPE_RECEITA

WHERE
    SEXO_SALARIO.ANO = EQUIPE_RECEITA.ANO

ORDER BY SEXO_SALARIO.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