MBA em Ciência de Dados

Análise de Dados com Base em Processamento Massivo em Paralelo

Avaliação Final

Aluno: Benicio Ramos Magalhães

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

2.1 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.

In [1]:
```
#comentado, pois a instalação já foi realizada

#instalando o módulo wget

#%%capture
#!pip install -q wget
#!mkdir data
```

2.2 Obtenção dos Dados das Tabelas de Dimensão

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

In [2]:
```
#comentado, pois banco já foi baixado

#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")
```

2.3 Obtenção dos Dados Tabelas de Fatos

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

In [3]:
```
#comentado, pois banco já foi baixado

#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 Apache Spark Cluster

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.

In [4]:
```
#comentado, pois instalação já foi realizada


#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.

In [5]:
```
#comentado, pois instalação já foi realizada


#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.

In [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"] = "/content/spark-3.0.0-bin-hadoop2.7"

import os

#import modificado para ambiente windows
os.environ['SPARK_HOME'] = 'D:\\Program\\spark-3.0.0-bin-hadoop2.7'
os.environ['PYSPARK_DRIVER_PYTHON_OPTS']='notebook'
```

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.

In [7]:
```
#comentado, pois instalação já foi realizada

#%%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 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.

In [8]:
```
#importando o módulo findspark
import findspark
#carregando a variávels 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.

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

#estava tomando erro no JAVA_HOME que não estava configurado.
#Dê preferencia para instalar em um local sem espaço em branco no nome do diretório em ambiente window
s.
spark = SparkSession.builder.appName("pyspark-notebook").master("local[*]").getOrCreate()
```

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 Criação dos DataFrames

In [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=",")
```

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.

In [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", "dataA
no"]
colunas_equipe = ["equipePK"]
colunas_funcionario = ["funcPK", "funcDiaNascimento", "funcMesNascimento", "funcAnoNascimento"]
colunas_negociacao = ["equipePK", "clientePK", "dataPK", "quantidadeNegociacoes"]
colunas_pagamento = ["funcPK", "equipePK", "dataPK", "cargoPK", "quantidadeLancamentos"]
```

In [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.

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

In [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()))
```

4.3 Criação de Visões Temporárias

In [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")
```

In [16]:
```
#visualizando um exemplo com 5 linhas do df para facilitar na criação das queries, conhecendo melhor a
 estrutura de dados
cargo.show(5)
```

```
+-------+--------------------+------------------+--------------------+-----------------------+------
----+
|cargoPK|           cargoNome|cargoRegimeTrabalho|cargoJornadaTrabalho|cargoEscolaridadeMinima|cargoN
ivel|
+-------+--------------------+------------------+--------------------+-----------------------+------
----+
|      1|PROGRAMADOR DE SI...|         TEMPORARIO|                 20H|                  MEDIO|    JU
NIOR|
|      2|PROGRAMADOR DE SI...|         TEMPORARIO|                 20H|               SUPERIOR|     P
LENO|
|      3|PROGRAMADOR DE SI...|         TEMPORARIO|                 20H|                    POS|    SE
NIOR|
|      4|PROGRAMADOR DE SI...|         TEMPORARIO|                 40H|                  MEDIO|    JU
NIOR|
|      5|PROGRAMADOR DE SI...|         TEMPORARIO|                 40H|               SUPERIOR|     P
LENO|
+-------+--------------------+------------------+--------------------+-----------------------+------
----+
only showing top 5 rows
```

In [17]:
```
#visualizando um exemplo com 5 linhas do df para facilitar na criação das queries, conhecendo melhor a
 estrutura de dados
cliente.show(5)
```

```
+---------+-------------------+-------------------+-------------+-----------------+------------------
+-----------------+------------------+---------------+----------------+
|clientePK|clienteNomeFantasia|       clienteSetor|clienteCidade|clienteEstadoNome|clienteEstadoSigla
|clienteRegiaoNome|clienteRegiaoSigla|clientePaisNome|clientePaisSigla|
+---------+-------------------+-------------------+-------------+-----------------+------------------
+-----------------+------------------+---------------+----------------+
|        1|           VIA FOOD|BEBIDAS E ALIMENTOS|    SAO PAULO|        SAO PAULO|                SP
|          SUDESTE|                SE|         BRASIL|              BR|
|        2|          VIA PIZZA|BEBIDAS E ALIMENTOS|    SAO PAULO|        SAO PAULO|                SP
|          SUDESTE|                SE|         BRASIL|              BR|
|        3|           VIA JAPA|BEBIDAS E ALIMENTOS|    SAO PAULO|        SAO PAULO|                SP
|          SUDESTE|                SE|         BRASIL|              BR|
|        4|            VIA VEG|BEBIDAS E ALIMENTOS|    SAO PAULO|        SAO PAULO|                SP
|          SUDESTE|                SE|         BRASIL|              BR|
|        5|          VIA DRINK|BEBIDAS E ALIMENTOS|   SAO CARLOS|        SAO PAULO|                SP
|          SUDESTE|                SE|         BRASIL|              BR|
+---------+-------------------+-------------------+-------------+-----------------+------------------
+-----------------+------------------+---------------+----------------+
only showing top 5 rows
```

In [18]:
```
#visualizando um exemplo com 5 linhas do df para facilitar na criação das queries, conhecendo melhor a
 estrutura de dados
data.show(5)
```

```
+------+------------+-------+-------+------------+-------------+------------+-------+
|dataPk|dataCompleta|dataDia|dataMes|dataBimestre|dataTrimestre|dataSemestre|dataAno|
+------+------------+-------+-------+------------+-------------+------------+-------+
|     1|    1/1/2016|      1|      1|           1|            1|           1|   2016|
|     2|    2/1/2016|      2|      1|           1|            1|           1|   2016|
|     3|    3/1/2016|      3|      1|           1|            1|           1|   2016|
|     4|    4/1/2016|      4|      1|           1|            1|           1|   2016|
|     5|    5/1/2016|      5|      1|           1|            1|           1|   2016|
+------+------------+-------+-------+------------+-------------+------------+-------+
only showing top 5 rows
```

In [19]:
```
#visualizando um exemplo com 5 linhas do df para facilitar na criação das queries, conhecendo melhor a
 estrutura de dados
equipe.show(5)
```

```
+--------+-------------+--------------------+--------------+------------------+-----------------+----
------------+-----------------+--------------+---------------+
|equipePK|   equipeNome|          filialNome|  filialCidade|  filialEstadoNome|filialEstadoSigla|fili
alRegiaoNome|filialRegiaoSigla|filialPaisNome|filialPaisSigla|
+--------+-------------+--------------------+--------------+------------------+-----------------+----
------------+-----------------+--------------+---------------+
|       1|APP - DESKTOP|SAO PAULO - AV. P...|     SAO PAULO|         SAO PAULO|               SP|
     SUDESTE|               SE|        BRASIL|             BR|
|       2|APP - DESKTOP|RIO DE JANEIRO - ...|RIO DE JANEIRO|    RIO DE JANEIRO|               RJ|
     SUDESTE|               SE|        BRASIL|             BR|
|       3|          WEB|SAO PAULO - AV. P...|     SAO PAULO|         SAO PAULO|               SP|
     SUDESTE|               SE|        BRASIL|             BR|
|       4|          WEB|RIO DE JANEIRO - ...|RIO DE JANEIRO|    RIO DE JANEIRO|               RJ|
     SUDESTE|               SE|        BRASIL|             BR|
|       5|          WEB|CAMPO GRANDE - CE...|  CAMPO GRANDE|MATO GROSSO DO SUL|               MS|
CENTRO-OESTE|               CO|        BRASIL|             BR|
+--------+-------------+--------------------+--------------+------------------+-----------------+----
------------+-----------------+--------------+---------------+
only showing top 5 rows
```

In [20]:
```
#visualizando um exemplo com 5 linhas do df para facilitar na criação das queries, conhecendo melhor a
 estrutura de dados
funcionario.show(5)
```

```
+------+-------------+-------------+--------+-----------------+-----------------+-----------------+-
----------------+-----------+--------------+---------------+--------------+---------------+----------
--+-------------+
|funcPK|funcMatricula|     funcNome|funcSexo|funcDataNascimento|funcDiaNascimento|funcMesNascimento|f
uncAnoNascimento| funcCidade|funcEstadoNome|funcEstadoSigla|funcRegiaoNome|funcRegiaoSigla|funcPaisNo
me|funcPaisSigla|
+------+-------------+-------------+--------+-----------------+-----------------+-----------------+-
----------------+-----------+--------------+---------------+--------------+---------------+----------
--+-------------+
|     1|          M-1|ALINE ALMEIDA|       F|          1/1/1990|                1|                1|
            1990|  SAO PAULO|     SAO PAULO|             SP|       SUDESTE|             SE|      BRAS
IL|           BR|
|     2|          M-2|  ARAO ALVES|       M|          2/2/1990|                2|                2|
            1990|   CAMPINAS|     SAO PAULO|             SP|       SUDESTE|             SE|      BRAS
IL|           BR|
|     3|          M-3| ARON ANDRADE|       M|          3/3/1990|                3|                3|
            1990|     SANTOS|     SAO PAULO|             SP|       SUDESTE|             SE|      BRAS
IL|           BR|
|     4|          M-4|  ADA BARBOSA|       F|          4/4/1990|                4|                4|
            1990|SANTO ANDRE|     SAO PAULO|             SP|       SUDESTE|             SE|      BRAS
IL|           BR|
|     5|          M-5|ABADE BATISTA|       M|          5/5/1990|                5|                5|
            1990| PIRACICABA|     SAO PAULO|             SP|       SUDESTE|             SE|      BRAS
IL|           BR|
+------+-------------+-------------+--------+-----------------+-----------------+-----------------+-
----------------+-----------+--------------+---------------+--------------+---------------+----------
--+-------------+
only showing top 5 rows
```

In [21]:
```
#visualizando um exemplo com 5 linhas do df para facilitar na criação das queries, conhecendo melhor a
 estrutura de dados
negociacao.show(5)
```

```
+--------+---------+------+--------+---------------------+
|equipePK|clientePK|dataPK| receita|quantidadeNegociacoes|
+--------+---------+------+--------+---------------------+
|       2|        9|    22|11564.75|                    1|
|       2|       24|    11| 17990.5|                    1|
|       2|       28|    21| 16335.9|                    1|
|       1|       30|    23| 8495.55|                    1|
|       2|       43|    30|24748.75|                    1|
+--------+---------+------+--------+---------------------+
only showing top 5 rows
```

In [22]:
```
#visualizando um exemplo com 5 linhas do df para facilitar na criação das queries, conhecendo melhor a
 estrutura de dados
pagamento.show(5)
```

```
+------+--------+------+-------+-------+---------------------+
|funcPK|equipePK|dataPK|cargoPK|salario|quantidadeLancamentos|
+------+--------+------+-------+-------+---------------------+
|   147|       2|     5|     64|1559.94|                    1|
|   124|       2|     5|    329|8102.77|                    1|
|   175|       1|     5|    328|2532.51|                    1|
|   171|       1|     5|    245| 7882.7|                    1|
|   148|       2|     5|     65|4404.59|                    1|
+------+--------+------+-------+-------+---------------------+
only showing top 5 rows
```

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 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 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 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 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 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 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 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 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, salário 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 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.

Questão 1

(valor: 1,0) Liste, para cada `dataAno` e para cada sexo do funcionário, 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.

Resposta da Questão 1

In [23]:
```
# Resposta da Questão 1

#----------------------------------------CÓDIGO COMENTADO----------------------------------------
#
# 01. selecionando dataAno como ANO, funcSexo como SEXO e a média do salário com arredondamento de 2 casas decimais como MEDIASALARIO
# 02. especificando as relações de data com pagamento contendo mesma dataPK
# 03. especificando as relações de funcionario com pagamento contendo mesmo funcPK
# 04. agrupando os resultados por ANO e SEXO
# 05. ordenando o ANO e SEXO de forma ascendente
# 06. apresentando os 20 primeiros resultados sem truncamento das strings
#------------------------------------------------------------------------------------------------

consultaSQL = """
SELECT dataAno AS `ANO`, funcSexo AS `SEXO`, ROUND(AVG(salario),2) AS `MEDIASALARIO`
FROM data JOIN pagamento ON (data.dataPK = pagamento.dataPK)
          JOIN funcionario ON (funcionario.funcPK = pagamento.funcPK)
GROUP BY ANO, SEXO
ORDER BY ANO, SEXO ASC
"""
spark.sql(consultaSQL).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   |7789.65     |
|2019|M   |7789.65     |
|2020|F   |7585.33     |
|2020|M   |7789.65     |
+----+----+------------+
```

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.

Resposta da Questão 2

In [24]:
```
# Resposta da Questão 2

#importando a função round para poder fazer o arredondamento dos valores agregados
from pyspark.sql.functions import round

#----------------------------------------CÓDIGO COMENTADO----------------------------------------
#
# 01. utilizando dados de funcionario
# 02. junto com dados de pagamento pela chave primária funcPK
# 03. junto com dados de data pela chave primária dataPK
# 04. selecionando os dados requisitados dataAno, funcSexo, funcRegiaoNome e salario
# 05. listando todas as agregações com cube usando as dataAno, funcSexo, funcRegiaoNome e agregando pela média dos salários.
#     O método cube serve para criar subtotais para todas as combinações de atributos definidos na lista de agrupamento
# 06. arredondando média dos salários para duas casas decimais
# 07. renomeando dataAno para ANO
# 08. renomeando funcSexo para SEXO
# 09. renomeando funcRegiaoNome para REGIAO
# 10. renomeando avg(salario) para MEDIASALARIO
# 11. ordenando por dataAno, funcSexo e funcRegiaoNome, todos ascendentes
# 12. apresentando os 20 primeiros resultados sem truncamento das strings
#------------------------------------------------------------------------------------------------

funcionario\
    .join(pagamento, on='funcPK')\
    .join(data, on='dataPK')\
    .select('dataAno','funcSexo','funcRegiaoNome','salario')\
    .cube('dataAno','funcSexo','funcRegiaoNome').avg('salario')\
    .withColumn('avg(salario)',round('avg(salario)',2))\
    .withColumnRenamed('dataAno','ANO')\
    .withColumnRenamed('funcSexo','SEXO')\
    .withColumnRenamed('funcRegiaoNome','REGIAO')\
    .withColumnRenamed('avg(salario)','MEDIASALARIO')\
    .orderBy('dataAno','funcSexo','funcRegiaoNome', ascending=True)\
    .show(20,truncate=False)
```

```
+----+----+--------+------------+
|ANO |SEXO|REGIAO  |MEDIASALARIO|
+----+----+--------+------------+
|null|null|null    |7672.31     |
|null|null|NORDESTE|7151.73     |
|null|null|SUDESTE |7446.14     |
|null|null|SUL     |9397.76     |
|null|F   |null    |7948.58     |
|null|F   |NORDESTE|7043.42     |
|null|F   |SUDESTE |7821.46     |
|null|F   |SUL     |9282.97     |
|null|M   |null    |7581.58     |
|null|M   |NORDESTE|7183.82     |
|null|M   |SUDESTE |7322.23     |
|null|M   |SUL     |9437.12     |
|2016|null|null    |7404.36     |
|2016|null|NORDESTE|9351.25     |
|2016|null|SUDESTE |6562.64     |
|2016|null|SUL     |14505.59    |
|2016|F   |null    |9169.65     |
|2016|F   |SUDESTE |8587.4      |
|2016|F   |SUL     |14932.14    |
|2016|M   |null    |6906.46     |
+----+----+--------+------------+
only showing top 20 rows
```

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.

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.

Resposta da Questão 3

In [43]:
```
# Resposta da Questão 3

#---------------------------------------------------CÓDIGO COMENTADO---------------------------------------------------#
# 01. selecionando dataAno como ANO
#     aplicando um select para extrair o ano atual e subtraindo de funcAnoNascimento para definir a IDADE
#     selecionando funcRegiaoNome como REGIAO
#     realizando a soma dos salarios arredondando para 2 casas decimais como TOTALSALARIO
# 02. especificando as relações de data com pagamento contendo mesma dataPK
# 03. especificando as relações de funcionario com pagamento contendo mesmo funcPK
# 04. aplicando condição para funcionárias apenas do sexo feminino
# 05. que nasceram entre 1970 (inclusive) e 1990 (inclusive) aplicando a função BETWEEN
# 06. que vivem no SUDESTE ou no NORDESTE (tanto pelo nome da região quanto pela sigla)
# 07. agrupando os dados por ANO, IDADE e REGIAO
# 08. ordenando por ANO, IDADE e REGIAO de forma ascendente
# 09. apresentando os 20 primeiros resultados sem truncamento das strings
#----------------------------------------------------------------------------------------------------------------------#

consultaSQL = """
SELECT dataAno AS `ANO`, ((SELECT EXTRACT (YEAR FROM CURRENT_DATE)) - funcAnoNascimento) AS `IDADE`, funcRegiaoNome AS `REGIAO`, ROUND(SUM(salario),2) AS `TOTALSALARIO`
FROM data JOIN pagamento ON (data.dataPK = pagamento.dataPK)
          JOIN funcionario ON (funcionario.funcPK = pagamento.funcPK)
WHERE funcSexo = 'F'
      AND funcAnoNascimento BETWEEN 1970 AND 1990
      AND (funcRegiaoNome = 'SUDESTE' OR funcRegiaoNome = 'NORDESTE' OR funcRegiaoSigla = 'SE' OR funcRegiaoSigla = 'NE')
GROUP BY ANO, IDADE, REGIAO
ORDER BY ANO ASC, IDADE ASC, REGIAO ASC
"""
spark.sql(consultaSQL).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
```

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.

Resposta da Questão 4

In [42]:
```
# Reposta da Questão 4

#---------------------------------------------------CÓDIGO COMENTADO---------------------------------------------------#
# 01. utilizando dados de negociacao
# 02. juntando com dados de equipe pela chave primária equipePK
# 03. juntando com dados de data pela chave primária dataPK
# 04. juntando com dados de cliente pela chave primária clientePK
# 05. aplicando condição para equipePK sendo 1, 3  e 5 e clientes da cidade de SAO CARLOS
# 06. selecionando os dados requisitados dataAno, equipeNome, filialNome, clienteSetor e receita
# 07. agrupando por dataAno, equipeNome, filialNome, clienteSetor e agregando pela soma das receitas
# 08. arredondando a soma da receitas para 2 casas decimais
# 09. renomeando dataAno para ANO
# 10. renomeando equipeNome para NOMEEQUIPE
# 11. renomeando filialNome para NOMEFILIAL
# 12. renomeando clienteSetor para SETORCLIENTE
# 13. renomeando sum(receita) para TOTALRECEITA
# 14. ordenando por ANO, NOMEEQUIPE, NOMEFILIAL e SETORCLIENTE de forma ascendente
# 15. apresentando os 20 primeiros resultados sem truncamento das strings
#----------------------------------------------------------------------------------------------------------------------#

negociacao\
    .join(equipe, on='equipePK')\
    .join(data, on='dataPK')\
    .join(cliente, on='clientePK')\
    .where('(equipePK = 1 OR equipePK = 3 OR equipePK = 5) AND clienteCidade = "SAO CARLOS"')\
    .select('dataAno', 'equipeNome', 'filialNome', 'clienteSetor', 'receita')\
    .groupBy('dataAno', 'equipeNome', 'filialNome', 'clienteSetor').sum('receita')\
    .withColumn('sum(receita)',round('sum(receita)',2))\
    .withColumnRenamed('dataAno','ANO')\
    .withColumnRenamed('equipeNome','NOMEEQUIPE')\
    .withColumnRenamed('filialNome','NOMEFILIAL')\
    .withColumnRenamed('clienteSetor','SETORCLIENTE')\
    .withColumnRenamed('sum(receita)','TOTALRECEITA')\
    .orderBy('ANO','NOMEEQUIPE','NOMEFILIAL', 'SETORCLIENTE',ascending=True)\
    .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     |
|2016|WEB          |SAO PAULO - AV. PAULISTA|BEBIDAS E ALIMENTOS|11256.35    |
|2016|WEB          |SAO PAULO - AV. PAULISTA|VESTUARIO          |103029.5    |
|2017|APP - DESKTOP|SAO PAULO - AV. PAULISTA|TECNOLOGIA         |57953.4     |
|2017|APP - DESKTOP|SAO PAULO - AV. PAULISTA|VESTUARIO          |12275.45    |
|2017|WEB          |CAMPO GRANDE - CENTRO   |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       |
|2019|APP - DESKTOP|SAO PAULO - AV. PAULISTA|TECNOLOGIA         |31092.85    |
|2019|APP - DESKTOP|SAO PAULO - AV. PAULISTA|VESTUARIO          |16979.95    |
|2019|WEB          |CAMPO GRANDE - CENTRO   |BEBIDAS E ALIMENTOS|37909.9     |
|2019|WEB          |CAMPO GRANDE - CENTRO   |VESTUARIO          |32699.5     |
|2019|WEB          |SAO PAULO - AV. PAULISTA|TECNOLOGIA         |8521.45     |
|2019|WEB          |SAO PAULO - AV. PAULISTA|VESTUARIO          |514.35      |
|2020|APP - DESKTOP|SAO PAULO - AV. PAULISTA|TECNOLOGIA         |16136.5     |
|2020|WEB          |SAO PAULO - AV. PAULISTA|VESTUARIO          |1201.05     |
+----+-------------+------------------------+-------------------+------------+
only showing top 20 rows
```

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.

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*.

Questão 5

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

Resposta da Questão 5

In [64]:
```
# Resposta da Questão 5

#---------------------------------------------------CÓDIGO COMENTADO---------------------------------------------------#
#    Neste caso vamos usar a operação drill-across que compara medidas numéricas de tabelas de fatos diferentes
#    utilizando uma dimensão em comum (no caso data).
# 01. selecionando dataAnoP (dataAno já consolidada na subquery de salário) como ANO
#     selecionando a soma dos salarios arredondando com 2 casas como TOTALSALARIO
#     selecionando a soma das receitas arredondando com 2 casas como TOTALRECEITA
#    Vamos executar dois sub-selects no comando FROM. O primeiro responsável pelos dados de salário e o segundo de receita
# 02. selecionando dataAno de data e soma dos salarios de funcionario
# 03. especificando as relações de pagamento com data contendo mesma dataPK
# 04. especificando as relações de funcionario com pagamento contendo mesmo funcPK
# 05. aplicando condição para funcionárias apenas do sexo feminino
# 06. aplicando condição para funcionárias do RIO DE JANEIRO (filtrando tanto pelo nome do estado quanto pela sigla)
# 07. agrupando os dados por dataAno
# 08. aplicando os resultados do sub-select como dataAnoP para dataAno e salario como o total do salário das funcionárias
# 09. cláusula para fazer a junção com outro sub-select, agora aplicado nos dados de receita
# 10. selecionando dataAno de data e soma das receitas de negociação
# 11. especificando as relações de data com negociação contendo mesma dataPK
# 12. especificando as relações de equipe com negociação contendo mesmo equipePK
# 13. aplicando condição para equipes do RIO DE JANEIRO (filtrando tanto pelo nome do estado quanto pela sigla)
# 14. agrupando os dados por dataAno
# 15. aplicando os resultados do sub-select como dataAnoN para dataAno e receita como o total do receita das equipes
# 16. aplicando condição para juntar a dimensão em comum, dataAnoP do salário igual ao dataAnoN da receita
# 17. agrupando por data, no caso dataAnoP
# 18. ordenando por data, no caso dataAnoP de forma ascendente
# 19. apresentando os 20 primeiros resultados sem truncamento das strings
#----------------------------------------------------------------------------------------------------------------------#

consultaSQL = """
SELECT dataAnoP AS `ANO`, ROUND(SUM(salario),2) AS `TOTALSALARIO`, ROUND(SUM(receita),2) AS `TOTALRECEITA`
FROM ( SELECT dataAno, SUM(salario)
       FROM pagamento JOIN data        ON (pagamento.dataPK = data.dataPK)
                      JOIN funcionario ON (funcionario.funcPK = pagamento.funcPK)
        WHERE funcSexo = 'F'
              AND (funcEstadoNome = 'RIO DE JANEIRO' OR funcEstadoSigla = 'RJ')
        GROUP BY dataAno
      ) AS sal(dataAnoP, salario)
      JOIN
      ( SELECT dataAno, SUM(receita)
        FROM data JOIN negociacao ON (data.dataPK = negociacao.dataPK)
                  JOIN equipe     ON (equipe.equipePK = negociacao.equipePK)
        WHERE (filialEstadoNome = 'RIO DE JANEIRO' OR filialEstadoSigla = 'RJ')
        GROUP BY dataAno
      ) AS rec(dataAnoN, receita)
WHERE dataAnoP = dataAnoN
GROUP BY dataAnoP
ORDER BY dataAnoP ASC
"""
spark.sql(consultaSQL).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   |
+----+------------+------------+
```

Questão 6

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

Resposta da Questão 6

In [70]:
```
# Resposta da Questão 6

#---------------------------------------------------CÓDIGO COMENTADO---------------------------------------------------#
#    Neste caso vamos usar a operação drill-across que compara medidas numéricas de tabelas de fatos diferentes
#    utilizando uma dimensão em comum (no caso data). Neste caso, vamos precisar separar as consultas em dois blocos,
#    sendo o primeiro referente aos dados de salario e o segundo referente aos dados de receitas.
# 01. utilizando dados de pagamento e atribuindo ao bloco 1 nomeado de pag
# 02. juntando com dados de data pela chave primária dataPK
# 03. juntando com dados de funcionario pela chave primária funcPK
# 04. aplicando condição para identificar funcionárias do estado do RIO DE JANEIRO (tanto por nome quanto por sigla)
# 05. selecionando os dados de dataAno e salario como resultados do bloco 1
# 06. agrupando os dados por dataAno
# 07. agregando os dados de salário como o total do salário das funcionárias
# 08. utilizando dados de negociacao e atribuindo ao bloco 2 nomeado de neg
# 09. juntando com dados de data pela chave primária dataPK
# 10. juntando com dados de equipe pela chave primária equipePK
# 11. aplicando condição para identificar equipes do estado do RIO DE JANEIRO (tanto por nome quanto por sigla)
# 12. selecionando os dados de dataAno e receita como resultados do bloco 2
# 13. agrupando os dados por dataAno
# 14. agregando os dados de receita como o total da receita das equipes
# 15. utilizando dados do bloco 1 nomeado de pag
# 16. juntando com dados do bloco 2 nomeado de neg pela dimensão em comum entre ambas, no caso, dataAno
# 17. selecionando os dados de dataAno, total de salarios e total de receitas
# 18. arredondando a soma dos salários para 2 casas decimais
# 19. arredondando a soma da receitas para 2 casas decimais
# 20. renomeando dataAno para ANO
# 21. renomeando sum(salario) para TOTALSALARIO
# 22. renomeando sum(receita) para TOTALRECEITA
# 23. ordenando por data, no caso ANO de forma ascendente
# 24. apresentando os 20 primeiros resultados sem truncamento das strings
#----------------------------------------------------------------------------------------------------------------------#

pag = pagamento\
   .join(data, on='dataPK')\
   .join(funcionario, on='funcPK')\
   .where('funcSexo = "F" AND (funcEstadoNome = "RIO DE JANEIRO" OR funcEstadoSigla = "RJ")')\
   .select('dataAno', 'salario')\
   .groupBy('dataAno')\
   .sum('salario')

neg = negociacao\
   .join(data, on='dataPK')\
   .join(equipe, on='equipePK')\
   .where('filialEstadoNome = "RIO DE JANEIRO" OR filialEstadoSigla = "RJ"')\
   .select('dataAno', 'receita')\
   .groupBy('dataAno')\
   .sum("receita")

pag\
   .join(neg, on='dataAno')\
   .select('dataAno', 'sum(salario)', 'sum(receita)')\
   .withColumn('sum(salario)', round('sum(salario)',2))\
   .withColumn('sum(receita)', round('sum(receita)',2))\
   .withColumnRenamed('dataAno', 'ANO')\
   .withColumnRenamed("sum(salario)", 'TOTALSALARIO')\
   .withColumnRenamed("sum(receita)", 'TOTALRECEITA')\
   .orderBy('ANO',ascending=True)\
   .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 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.

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.

Resposta da Questão 7

In [78]:
```
# Resposta da Questão 7

#---------------------------------------------------CÓDIGO COMENTADO---------------------------------------------------#
#    Neste caso vamos usar a operação drill-across que compara medidas numéricas de tabelas de fatos diferentes
#    utilizando uma dimensão em comum (no caso data).
# 01. selecionando dataAnoPF (dataAno já consolidada na subquery de salário para sexo feminino) como ANO
#     selecionando a soma dos salarios femininos arredondando com 2 casas como TOTALSALARIOMULHERES
#     selecionando a soma dos salarios masculinos arredondando com 2 casas como TOTALSALARIOHOMENS
#     selecionando a soma das receitas arredondando com 2 casas como TOTALRECEITA
#    Nestes casos, tivemos alguns números sendo apresentados com notação científica, portanto, aplicamos a função CAST
#    AS DECIMAL, para a saída ficar no formato adequado (requerido no exercício). Como parâmetros do DECIMAL passamos o
#    valor 10 (tamanho considerado adequado - número de dígitos permitidos na parte inteira) e 2 (número de casas decimais).
#    Vamos executar três sub-selects no comando FROM. O primeiro responsável pelos dados de salário feminino
#    o segundo responsável pelos dados de salário masculino e o terceiro responsável pelos dados de receita.
# 02. selecionando dataAno de data e soma dos salarios como salarioF (que terá os dados para funcionárias de sexo feminino)
# 03. especificando as relações de pagamento com data contendo mesma dataPK
# 04. especificando as relações de funcionario com pagamento contendo mesmo funcPK
# 05. aplicando condição para funcionárias apenas do sexo feminino
# 06. agrupando os dados por dataAno
# 07. aplicando os resultados do sub-select como dataAnoPF para dataAno e salarioF como o total do salário das mulheres
# 08. cláusula para juntarmos o segundo sub-select
# 09. selecionando dataAno de data e soma dos salarios como salarioM (que terá os dados para funcionárias de sexo masculino)
# 10. especificando as relações de pagamento com data contendo mesma dataPK
# 11. especificando as relações de funcionario com pagamento contendo mesmo funcPK
# 12. aplicando condição para funcionários apenas do sexo masculino
# 13. agrupando os dados por dataAno
# 14. aplicando os resultados do sub-select como dataAnoPM para dataAno e salarioM como o total do salário dos homens
# 15. cláusula para juntarmos o terceiro sub-select
# 16. selecionando dataAno de data e soma das receitas de negociação
# 17. especificando as relações de data com negociação contendo mesma dataPK
# 18. especificando as relações de equipe com negociação contendo mesmo equipePK
# 19. agrupando os dados por dataAno
# 20. aplicando os resultados do sub-select como dataAnoN para dataAno e receita como o total do receita das equipes
# 21. aplicando condição para juntar a dimensão em comum, dataAnoPF igual ao dataAnoN, igual ao dataAnoPM
# 22. agrupando por data, no caso dataAnoPF
# 23. ordenando por data, no caso dataAnoPF de forma ascendente
# 24. apresentando os 20 primeiros resultados sem truncamento das strings
#----------------------------------------------------------------------------------------------------------------------#

consultaSQL = """
SELECT dataAnoPF AS `ANO`, CAST(ROUND(SUM(salarioF),2) AS DECIMAL(10,2)) AS `TOTALSALARIOMULHERES`, CAST(ROUND(SUM(salarioM),2) AS DECIMAL(10,2)) AS `TOTALSALARIOHOMENS`, CAST(ROUND(SUM(receita),2) AS DECIMAL(10,2)) AS `TOTALRECEITA`
FROM  ( SELECT dataAno, SUM(salario) AS `salarioF`
        FROM pagamento JOIN data        ON (pagamento.dataPK = data.dataPK)
                       JOIN funcionario ON (funcionario.funcPK = pagamento.funcPK)
        WHERE funcSexo = 'F'
        GROUP BY dataAno
      ) AS salF(dataAnoPF, salarioF)
      JOIN
      ( SELECT dataAno, SUM(salario) AS `salarioM`
        FROM pagamento JOIN data        ON (pagamento.dataPK = data.dataPK)
                       JOIN funcionario ON (funcionario.funcPK = pagamento.funcPK)
        WHERE funcSexo = 'M'
        GROUP BY dataAno
      ) AS salM (dataAnoPM, salarioM)
      JOIN
      ( SELECT dataAno, SUM(receita)
        FROM data JOIN negociacao ON (data.dataPK = negociacao.dataPK)
                  JOIN equipe     ON (equipe.equipePK = negociacao.equipePK)
        GROUP BY dataAno
      ) AS rec(dataAnoN, receita)
WHERE (dataAnoPF = dataAnoN AND dataAnoPF = dataAnoPM)
GROUP BY dataAnoPF
ORDER BY dataAnoPF ASC
"""
spark.sql(consultaSQL).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          |11135098.98       |11593539.66 |
|2019|4733247.25          |13834419.20       |35353318.33 |
|2020|4733247.25          |13834419.20       |30222175.87 |
+----+--------------------+------------------+------------+
```