



# SQL IMPRESSIONADOR

Apostila completa de SQL.





# SQL

## Módulo SQL SERVER



# SQ<sub>L</sub>

Banco de Dados e SQL





## TEMA 1

O que é um dado?

O que é um dado?

O que é um dado?



# O que é um dado?

Um dado é uma informação que permite chegar ao conhecimento de algo. Estamos rodeados de dados.

Observe a imagem ao lado: concluímos que são bolas de futebol, certo? Logo, o primeiro dado que temos delas é que são utilizadas para jogar futebol. Além disso, temos outras informações sobre elas. Todas têm um tamanho, há uma determinada quantidade delas, provavelmente elas têm um custo associado, um tempo de uso, etc.

Qualquer objeto possui informações associadas. Nós, por exemplo, temos um nome, uma data de nascimento, um peso, altura e assim vai. Todas essas informações são dados.

Por esses exemplos iniciais, chegamos a uma conclusão: existem diferentes tipos de dados.



# O que é um dado?

Em resumo, um dado é uma informação sobre alguma coisa. Em uma empresa, os dados cumprem um papel tão importante quanto qualquer outro bem.

Porém, para extrair o melhor de um dado, não basta tê-lo, é preciso organizá-lo para então entendê-lo.

“

**Os dados são o novo petróleo.**

# O que é um dado?

## Os dados estão em todo lugar

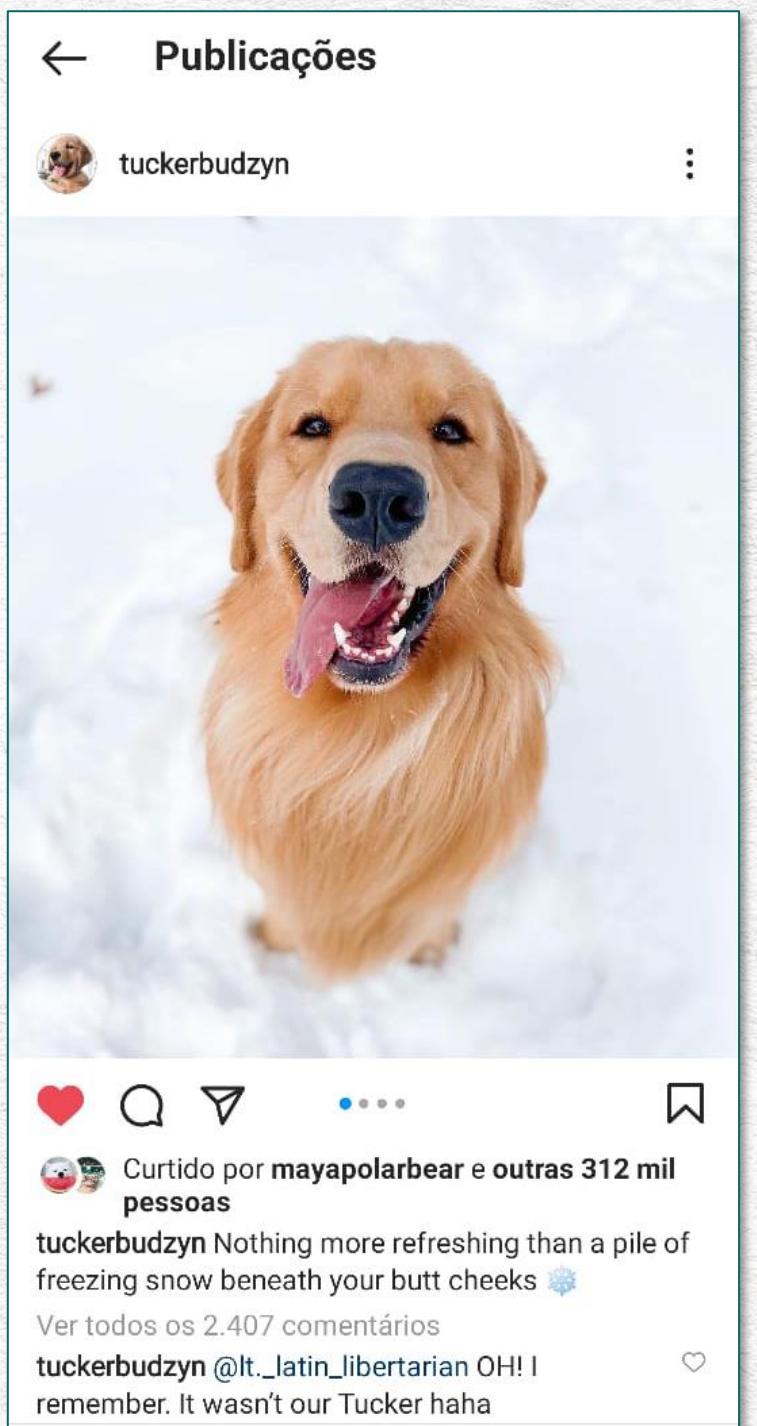
Imagina que você curta uma determinada foto no Instagram, como a foto mostrada ao lado.

Pra você, pra mim e para qualquer outro usuário comum do aplicativo se trata de uma mera curtida em uma foto...

... Porém, para o Instagram, essa curtida vale ouro.

Uma vez que você curte a publicação ao lado, o Instagram passa a ter um dado muito importante sobre você: **você gosta de cachorros**. Com esse dado, ou seja, com essa informação, o Instagram vai começar a mostrar a você mais fotos de cachorros. E se você continuar curtindo essas fotos, o Instagram vai mostrar mais e mais fotos de cachorros para você, até chegar a um ponto de você se perguntar “uau, como o Instagram sabe que eu gosto de cachorros?”

Só que não é mágica, é apenas o Instagram armazenando esses dados, entendendo um padrão e tomando uma ação a partir disso.



# O que é um dado?

Todas as empresas possuem um grande volume de informações/dados que precisam ser organizados e armazenados.

Vamos tomar como exemplo a situação anterior do Instagram. O aplicativo possui mais de 100 milhões de usuários. Imagine a quantidade de dados que precisam ser armazenados:

- Perfis
- Fotos
- Vídeos
- Mensagens
- Etc..

A partir dessas informações, o Instagram vai entender quais são as suas preferências como usuário, o que você gosta ou não, o que deve mostrar a você ou não e como prender a sua atenção, a fim de atingir um objetivo simples: gerar o maior engajamento possível.

## Não é só o Instagram que vive de dados

Não é só o Instagram que tem o desafio diário de manipular tantos dados. Qualquer empresa precisa manipular diariamente centenas, milhares ou até milhões de dados.

É claro que o volume de dados vai variar de acordo com o tamanho da empresa, mas de forma geral, toda empresa precisa ter os dados organizados e centralizados.



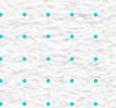


## TEMA 2

O que é um banco de dados?

O que é um banco de dados?

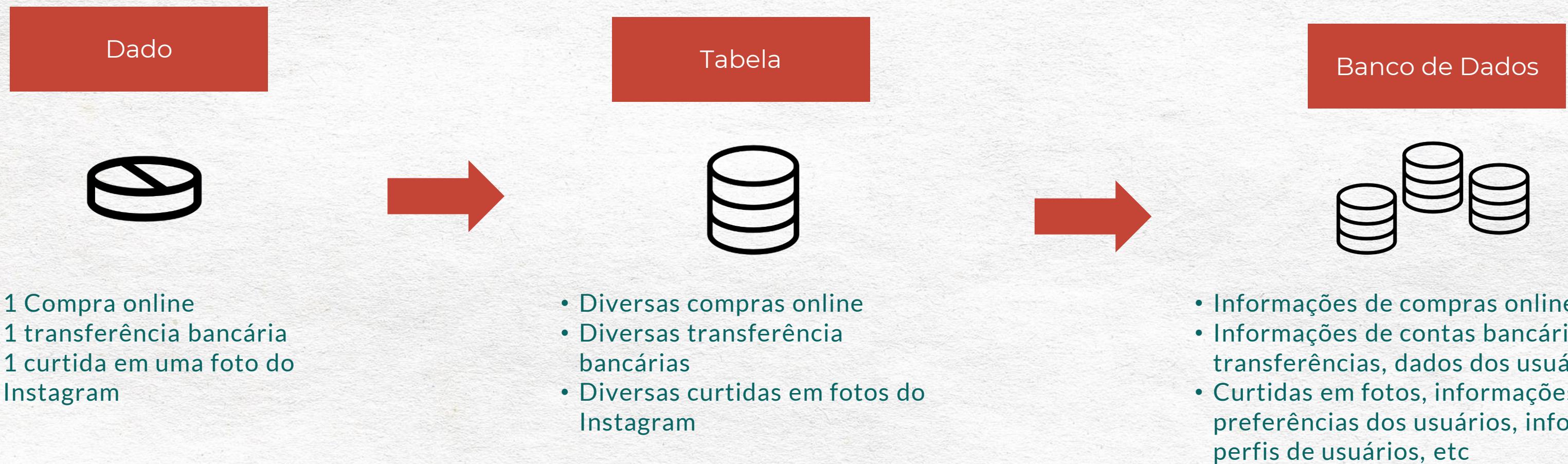
O que é um banco de dados?



# O que é um banco de dados?

Como vimos anteriormente, um dado é uma informação que nos permite chegar ao conhecimento de algo. Um dado isolado por si só já tem um grande valor. Em conjunto com outros dados então, ele é mais valioso ainda.

Vamos então tentar entender o que é um Banco de Dados de uma maneira bem simples. Imagine a imagem abaixo. Um dado carrega consigo alguma informação. Quando temos vários dados, precisamos organizar em uma tabela. E quando temos várias tabelas, temos um banco de dados.



# O que é um banco de dados?

Com os dados bem organizados e estruturados, o Instagram será capaz de melhorar a experiência de cada usuário do aplicativo, mostrando temas mais relevantes de acordo com as preferências de cada usuário.

## Dado

O registro de 1 curtida feita pelo usuário.



Data	Usuário	Curtidas	Categoria
10/05/2021	Joãozinho	1	Cachorro

## Tabela

Lugar onde serão reunidos todas as curtidas dos usuários.



Data	Usuário	Curtidas	Categoria
10/05/2021	Joãozinho	1	Cachorro
10/05/2021	Mariazinha	3	Gato
11/05/2021	Joãozinho	5	Cachorro
13/05/2021	Mariazinha	10	Panda
15/05/2021	Joãozinho	6	Cachorro

## Banco de Dados

Conjunto de várias tabelas que se relacionam, com informações de curtidas, usuários e perfis.

Data	Usuário	Curtidas	Categoria
10/05/2021	Joãozinho	1	Cachorro
10/05/2021	Mariazinha	3	Gato
11/05/2021	Joãozinho	5	Cachorro
13/05/2021	Mariazinha	10	Panda
15/05/2021	Joãozinho	6	Cachorro

Usuário	Idade	Cidade
Joãozinho	21	Rio de Janeiro
Mariazinha	25	São Paulo

Categoria	Perfis
Cachorro	30
Gato	10
Panda	5

# O que é um banco de dados?

Um outro exemplo seria um banco de dados da Nubank. Com ele, a empresa é capaz de entender o perfil de consumo de cada usuário e definir quais terão prioridade em um possível aumento de limite.

## Dado

O registro de 1 compra feita pelo usuário.

Data	Usuário	Loja	Valor
10/05/2021	Joãozinho	Ifood	R\$ 150,00



## Tabela

Lugar onde serão reunidos todos os dados.

Data	Usuário	Loja	Valor
10/05/2021	Joãozinho	Ifood	R\$ 150,00
10/05/2021	Mariazinha	Uber	R\$ 10,30
11/05/2021	Joãozinho	Magalu	R\$ 5.000,00
13/05/2021	Mariazinha	Uber	R\$ 8,90
15/05/2021	Joãozinho	Rapi	R\$ 120,00



## Banco de Dados

Conjunto de várias tabelas que se relacionam, com informações de compras e usuários.

Data	Usuário	Compra	Valor
10/05/2021	Joãozinho	Ifood	R\$ 150,00
10/05/2021	Mariazinha	Uber	R\$ 10,30
11/05/2021	Joãozinho	Magalu	R\$ 5.000,00
13/05/2021	Mariazinha	Uber	R\$ 8,90
15/05/2021	Joãozinho	Rapi	R\$ 120,00

Usuário	Idade	Limite
Joãozinho	21	R\$ 7.000,00
Mariazinha	25	R\$ 6.500,00

Loja	Vendas
Ifood	1000
Uber	560
Magalu	5
Rapi	800

# O que é um banco de dados?

“

Bancos de dados são conjuntos de tabelas, com alguma relação entre si, com dados sobre pessoas, lugares ou coisas.

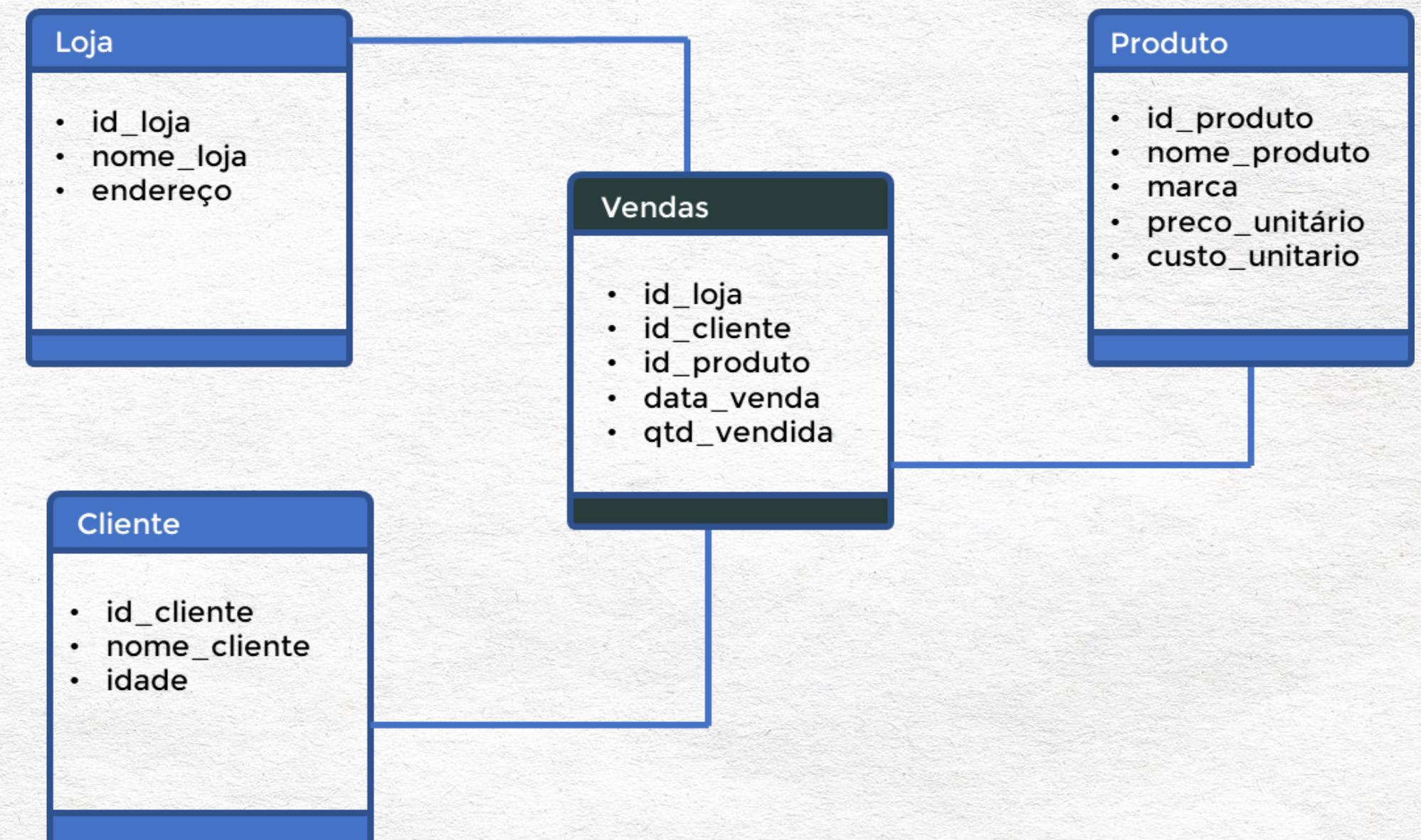
Estes dados organizados permitem a compreensão de um determinado fenômeno na empresa, seja a preferência dos usuários em uma rede social, seja o perfil de consumo em um aplicativo de transações financeiras.

# O que é um banco de dados?

O desenho esquemático de um banco de dados é algo como o mostrado ao lado. Diversas tabelas, com diferentes informações sobre um negócio, e que possuem algum tipo de relação.

A esse banco de dados damos o nome de **RELACIONAL**.

Bancos de dados relacionais serão o foco do nosso curso, até por serem o tipo de bancos de dados mais comumente encontrados no mercado.





## TEMA 3

O que é o SQL?

O que é o SQL?

O que é o SQL?



# O que é o SQL?

Como vimos, dados são o coração de qualquer negócio. Uma empresa como o Facebook precisa manipular, consultar e armazenar bilhões de informações dos usuários, como publicações, fotos, stories, mensagens privadas, etc.

Para que isso seja possível, é necessário algum sistema de banco de dados para manipular todas essas informações.

E quem vai permitir que a gente consiga buscar todos esses dados em bancos de dados, adicionar novos dados, excluir esses dados e atualizar esses dados é o **SQL**.

O **SQL** é uma linguagem padrão para trabalhar com bancos de dados relacionais. Será através do SQL que seremos capazes de consultar e manipular os dados dos nossos bancos de dados.

SQL significa: “**Structured Query Language**”. Se trata de uma linguagem de programação que permite a manipulação dos dados em um banco de dados.

# O que é o SQL?

O SQL é uma linguagem que muitos profissionais acabam precisando aprender: seja quem usa Excel de forma pesada e acaba migrando as informações para um Banco de Dados, seja um cientista de dados que usa o Python para agregar os dados das diferentes fontes de informação.

“

SQL significa “**Structured Query Language**”. Se trata de uma linguagem de programação utilizada para armazenar, consultar, adicionar e excluir informações em um banco de dados.

# O que é o SQL?

Imagine que você tem uma tabela em um banco de dados com as vendas de alguns produtos (pode até ser feita uma analogia a uma planilha do Excel).

id	produto	data_venda	valor
1	televisão	2021-03-20	1500
2	computador	2021-03-22	2300
3	celular	2021-03-25	800
4	ps4	2021-03-28	3100
5	tablet	2021-03-28	650

Se você quer buscar todas as vendas com um valor maior que 1000 reais, o código SQL que você deve usar é o seguinte:

```
SELECT * FROM tabela_vendas WHERE valor > 1000
```

O resultado do código anterior será uma nova tabela, mostrando apenas os produtos onde o valor de venda é maior que 1000 reais.

id	produto	data_venda	valor
1	televisão	2021-03-20	1500
2	computador	2021-03-22	2300
4	ps4	2021-03-28	3100



Sem perceber, o que acabamos de fazer anteriormente foi um exemplo de uma **query**, ou seja, uma **consulta** ao banco de dados, usando um código SQL para isso.

Relembrando: SQL significa “Structured Query Language”, traduzindo, “linguagem de consulta estruturada”.

## TEMA 4

S i s t e m a s   d e   B a n c o   d e   D a d o s ?

S i s t e m a s   d e   B a n c o   d e   D a d o s ?

S i s t e m a s   d e   B a n c o   d e   D a d o s ?



# Sistemas de Gerenciamento de Bancos de Dados

Agora que já sabemos o que é o SQL, surge a seguinte dúvida: o que é um Sistema de Gerenciamento de Banco de Dados, ou SGBD?

Um SGBD permite ao desenvolvedor trabalhar com diferentes tabelas de um banco de dados através de uma interface. Essa interface seria basicamente um programa que nos permite fazer a leitura de tabelas de um banco de dados e utilizar o SQL para manipular esses dados, tudo de uma maneira bem visual e *user-friendly*.

Um SGBD é composto essencialmente por 2 partes:



um **servidor**, onde vamos conseguir armazenar os nossos bancos de dados.



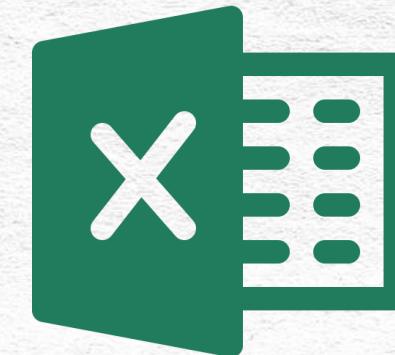
uma **interface** amigável que nos permite escrever os códigos em SQL para acessar os bancos de dados.

# Sistemas de Gerenciamento de Bancos de Dados

# E o que dizer sobre o Excel e o Access?

Em uma visão bem simplista, podemos considerar o Excel como uma simplificação de um SGBD. Por meio dele, conseguimos criar diferentes planilhas e tabelas, relacionando-as entre si por meio de fórmulas.

O Access já seria um programa com mais similaridades a um SGBD, porém com muitas limitações também, pensando em uma empresa que lida com alto volume de dados e que necessita uma maior segurança de armazenamento.



The screenshot shows the Microsoft Access 2016 interface with the 'Access2016\_SampleDatabase : Database...' window open. The ribbon menu is visible at the top, showing tabs for File, Home, Create, External Data, Database Tools, Fields, Table, Tell me..., and Merced Flores. The 'Home' tab is selected. On the left, the 'All Access Objects' navigation pane lists tables like 'Categories', 'Customers' (which is currently selected and highlighted in red), 'Menu Items', 'Order Items', 'Orders Table', 'Orders: December', 'Products Table', 'Sales Unit', and queries like 'Cakes & Pies Sold' and 'Cookies Sold'. The main workspace displays the 'Customers' table data in a grid format. The table has columns: ID, First Name, Last Name, and Street Ad. The data includes 13 rows of customer information. The 'Find' toolbar is visible above the grid, and the status bar at the bottom shows 'Record: 14 < 1 of 200 > No Filter Search'.

ID	First Name	Last Name	Street Ad
1	Tracey	Beckham	7 East Walker
2	Lucinda	George	789 Brewer St.
3	Jerrod	Smith	211 St. George
4	Brett	Newkirk	47 Hillsborough
5	Chloe	Jones	23 Solo Ln.
6	Quinton	Boyd	4 Cypress Cr.
7	Alex	Hinton	1011 Hodge Lr
8	Nisha	Hall	123 Huntington
9	Hillary	Clayton	2516 Newman
10	Kiara	Williams	9014 Miller Ln
11	Katy	Jones	456 Denver Rd
12	Beatrix	Joslin	85 North West
13	Mariah	Allen	12 Jupe

# Sistemas de Gerenciamento de Bancos de Dados

Existem alguns SGBDs para Bancos de Dados Relacionais que são muito utilizados por grandes empresas. Abaixo, temos os 4 principais programas para SGBDs.

É importante que fique claro que **todos esses SGBDs** utilizam o SQL como linguagem de programação.



## MySQL

É um SGBD relacional de código aberto, usado na maioria das aplicações gratuitas para gerir suas bases de dados.

A interface de código utilizada é o MySQL Workbench.

## Oracle

A Oracle é uma das maiores empresas de tecnologia do mundo.

O SGBD da Oracle é focado em empresas de médio e grande porte.

A interface de código utilizada é o SQL Developer.

## SQL Server

O SQL Server é o SGBD criado pela Microsoft, também para bancos de dados relacionais.

A interface de código utilizada é o SSMS.

## PostgreSQL

O PostgreSQL é um SGBD relacional, criado em 1989 e ainda um dos mais utilizados no mundo.

A interface de código utilizada é o pgAdmin.

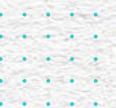


## TEMA 5

SQL vs MySQL, SQL Server,  
Oracle e PostgreSQL?

SQL vs MySQL, SQL Server,  
Oracle e PostgreSQL?

SQL vs MySQL, SQL Server,  
Oracle e PostgreSQL?



# SQL vs. MySQL, SQL Server, Oracle e PostgreSQL

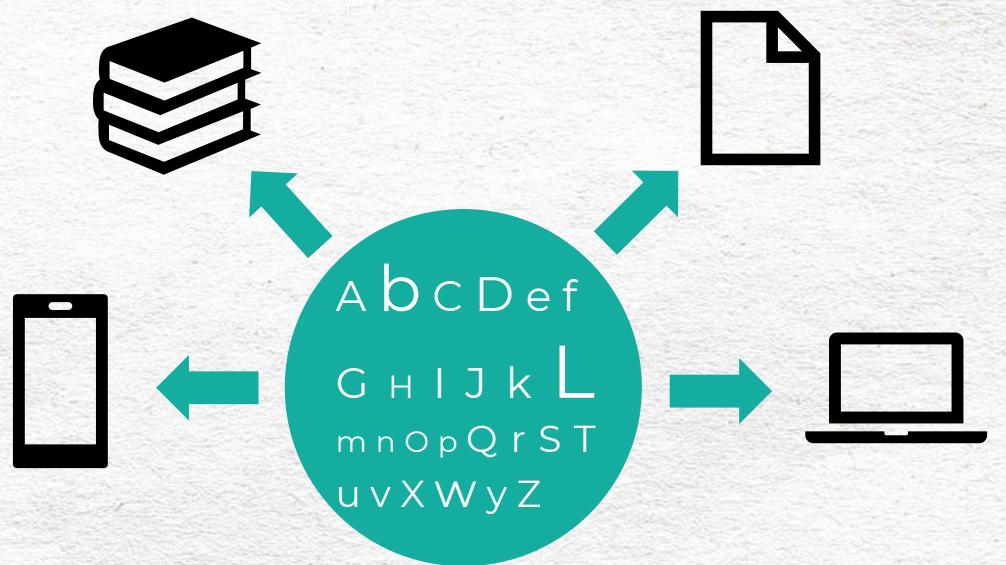
Uma das dúvidas mais comuns de quem começa a estudar SQL é: o que eu devo aprender? SQL ou MySQL? SQL ou SQL Server?

**A resposta é: esse questionamento não faz sentido!**

Para você entender a diferença entre estes termos, imagine que você queira aprender a se comunicar com outras pessoas. Você vai concordar que o alfabeto é muito importante para esse objetivo, certo? Mas de nada serve o alfabeto se não tivermos um local onde podemos combinar essas letras e formar palavras que permitam a comunicação.

Por isso, além do alfabeto, precisamos também de um lugar para escrever as palavras: pode ser uma folha de papel, um caderno, o nosso celular, um computador, e assim vai.

**A nossa decisão então não será entre aprender o alfabeto ou usar uma folha de papel, ou aprender o alfabeto ou usar um caderno. O alfabeto será importante em qualquer situação: ele é a base da nossa comunicação.**



# SQL vs. MySQL, SQL Server, Oracle e PostgreSQL

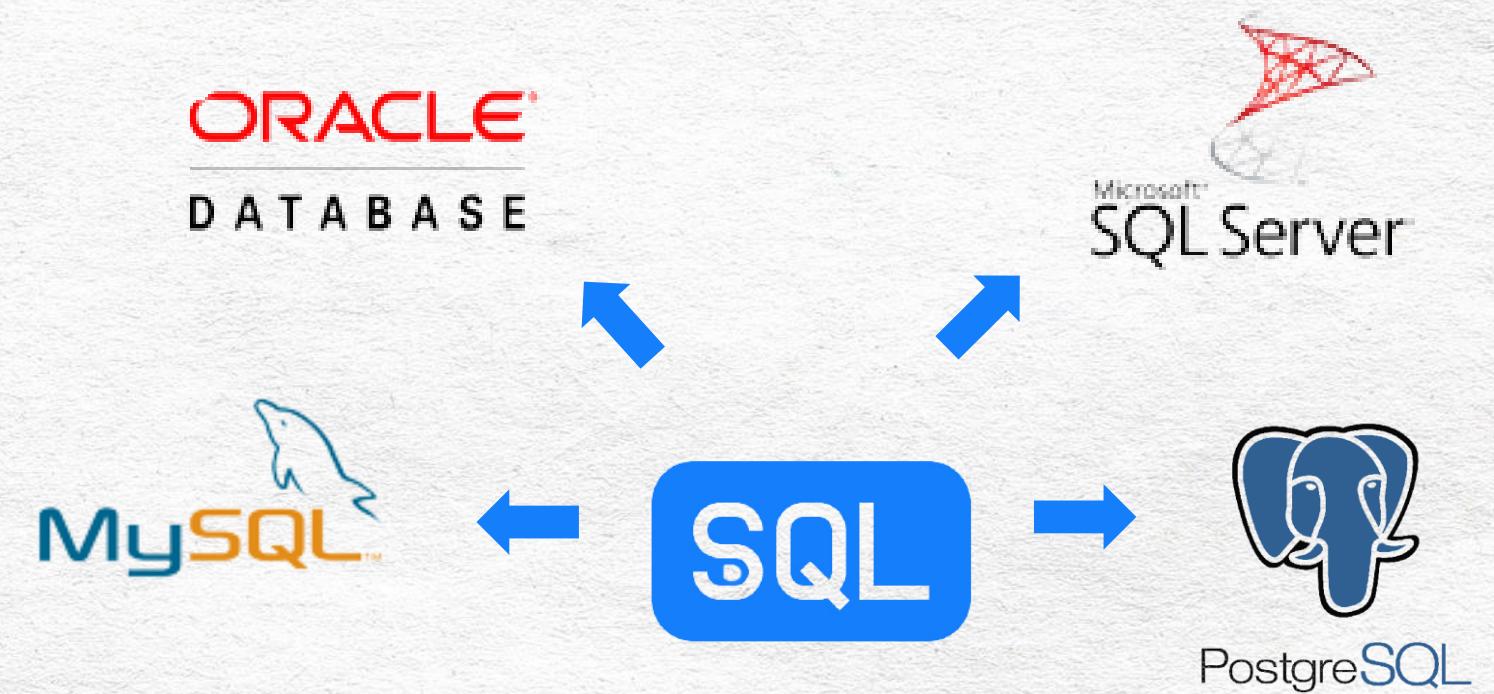
Fazendo um paralelo com o SQL, o SQL é como se fosse o nosso alfabeto, e os diferentes SGBDs serão os lugares onde conseguiremos escrever esses códigos em SQL para acessar nossos bancos de dados.

Por isso, a pergunta correta não é: aprender SQL ou MySQL? E sim, qual sistema de gerenciamento de banco de dados usar: MySQL, SQL Server, Oracle ou PostgreSQL? O SQL você vai precisar aprender em qualquer um dos casos.



O SQL é a linguagem de programação usada para manipular os bancos de dados.

Já o MySQL, SQL Server, Oracle Database e PostgreSQL são sistemas de bancos de dados, um ambiente onde vamos utilizar o SQL.



# TEMA 6

SQL, T-SQL, PL-SQL e pgSQL

SQL, T-SQL, PL-SQL e pgSQL

SQL, T-SQL, PL-SQL e pgSQL



# SQL, T-SQL, PL-SQL e PL-pgSQL: do que se tratam?

O SQL é a linguagem de consulta usada para comunicação com dados em um banco de dados. É usado como a linguagem padrão em todos os sistemas de gerenciamento de banco de dados relacional.

No entanto, o SQL possui algumas vertentes que dependem de cada sistema de gerenciamento de banco de dados: cada SGBD incorpora à linguagem algumas funcionalidades e recursos extras.

Apesar de algumas diferenças, a base para todas as demais variantes do SQL seguem uma lógica semelhante, mas vale a pena entender que elas existem e que, caso você ouça por ai, você entenda minimamente sobre cada uma delas.

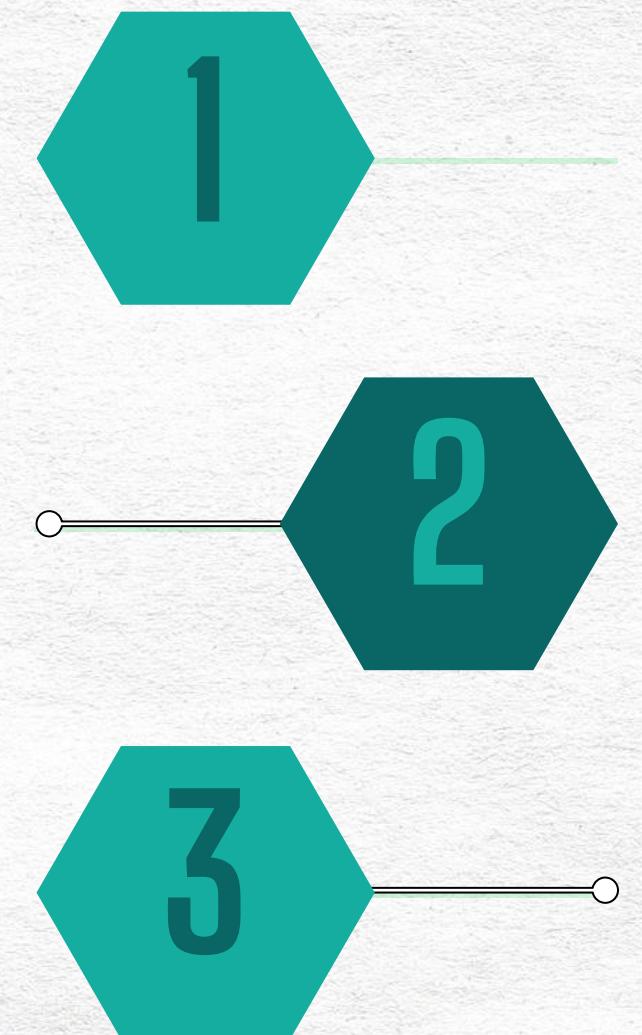
# SQL, T-SQL, PL-SQL e PL-pgSQL: do que se tratam?

Abaixo, um resumo de cada uma dessas variações.

## PL-SQL

PL-SQL ou Procedural Language/SQL é outra forma estendida de SQL que é usada pela Oracle para seu banco de dados.

A principal diferença entre o T-SQL e o PL-SQL é a maneira como eles lidam com variáveis, procedimentos armazenados e funções integradas. T-SQL também é considerado mais fácil e simples de entender, enquanto PL-SQL incorpora maior complexidade.



## T-SQL

TSQL, ou T-SQL (abreviação de Transaction-SQL), é uma versão aprimorada do SQL que contém algumas extensões. O TSQL foi originalmente desenvolvido pela Sybase e agora é propriedade da Microsoft.

O T-SQL adiciona alguns recursos avançados ao SQL para torná-lo mais poderoso, como variáveis declaradas, controle de transação, tratamento de erros e exceções, operações de string, processamento de data e hora, etc.

## PL-pgSQL

PL-PgSQL é uma linguagem procedural específica do PostgreSQL baseada em SQL. Semelhante ao T-SQL e PL-SQL, ele adiciona alguns recursos avançados ao SQL, como loops, variáveis, tratamento de erros e exceções, etc.





## TEMA 7

Por que aprender SQL?

Por que aprender SQL?

Por que aprender SQL?



# Por que aprender SQL?

O SQL possui aplicações em diferentes áreas e profissões. O profissional que tiver conhecimento não só em ferramentas como o Excel, Power BI ou Python, mas também em SQL, terá muitas oportunidades de atuação.

O motivo para aprender SQL é simples: garantir uma maior versatilidade no conhecimento e ser um profissional diferenciado e requisitado no mercado.

A seguir, listamos algumas áreas que precisam manipular um alto volume de dados, e consequentemente precisarão de sistemas de bancos de dados para auxiliar neste trabalho.

# Por que aprender SQL?

## Setor financeiro

Apps de banco, sistemas de pagamento e empresas de investimento

Imagine todo o volume de dados por trás de todas as operações financeiras que fazemos no dia a dia: abertura de contas, pagamentos, transferências, depósitos, investimentos variados, etc.

Para que tudo funcione perfeitamente, é necessário que as empresas tenham um controle dos dados através de sistemas de bancos de dados.



## Redes sociais

Facebook, Instagram, LinkedIn, TikTok

Redes sociais são mais um exemplo de um local onde encontramos um alto volume de dados. Elas armazenam bilhões de dados de usuários, localizações, preferências, fotos, etc.

A partir desses dados, os aplicativos entendem o comportamento de cada usuário e buscam proporcionar a melhor experiência possível para cada um.

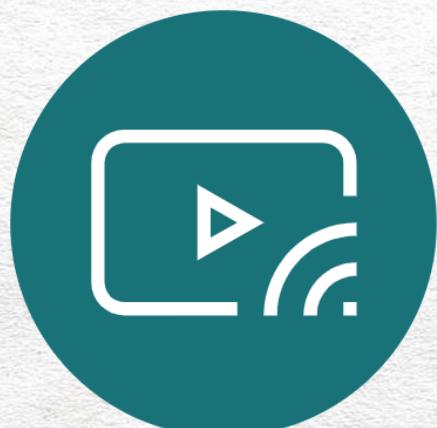


## Aplicativos de música e Streaming

Spotify, Netflix, Globoplay, Amazon Prime

Os Bancos de Dados auxiliam no armazenamento de informações de milhares de bibliotecas, álbuns, músicas, artistas, filmes, séries e gêneros e utilizam esses dados para cruzar com as informações de preferências dos usuários.

Todo o trabalho de pesquisa e sugestão de músicas, séries e filmes tem por trás um trabalho complexo de manipulação e cruzamento de dados.



## Redes de pesquisa

Youtube e Google

Na mesma linha das redes sociais, os mecanismos de busca na internet também se utilizam de uma grande quantidade de dados para sugerir vídeos, postagens, sites, etc, de acordo com as buscas e comportamento na web.

# Por que aprender SQL?

## Data science

Data Science, ou Ciência de Dados, é uma área voltada para o estudo e análise de dados econômicos, financeiros e sociais, com o objetivo de chegar ao conhecimento de algo. Já é de se esperar que essa é uma das áreas que mais necessita de um bom gerenciamento dos dados.



## Setor de energia

O setor de energia é uma área com muitas oportunidades no mercado, envolvendo empresas de geração, transmissão, distribuição e comercialização de energia. Imagine o volume de dados que este setor precisa manipular. Sem bancos de dados bem estruturados, não seria algo viável.



## Business Intelligence e Análise de Dados

A base para se trabalhar com BI e criar relatórios, gráficos e dashboards que auxiliam na tomada de decisão de um negócio é ter os dados do negócio bem estruturado.



## Marketing

Profissionais da área de marketing precisam manipular dados sobre seus clientes, como jornada, histórico de compras, interesse no produto. Dependendo do tamanho/alcance do negócio ou da empresa, será necessário trabalhar com um alto volume de dados.





## TEMA 8

Profissionais na área de BD

Profissionais na área de BD

Profissionais na área de BD



# Profissionais em bancos de dados

Um profissional com conhecimento em Bancos de Dados pode atuar em diferentes segmentos. Abaixo, listamos as principais áreas de atuação. Observe que há uma grande variedade para que trabalha com dados.



# SQL

Instalação do SQL Server,  
SSMS e conhecendo o  
Banco de Dados



## TEMA 9

Instalando o SQL Server

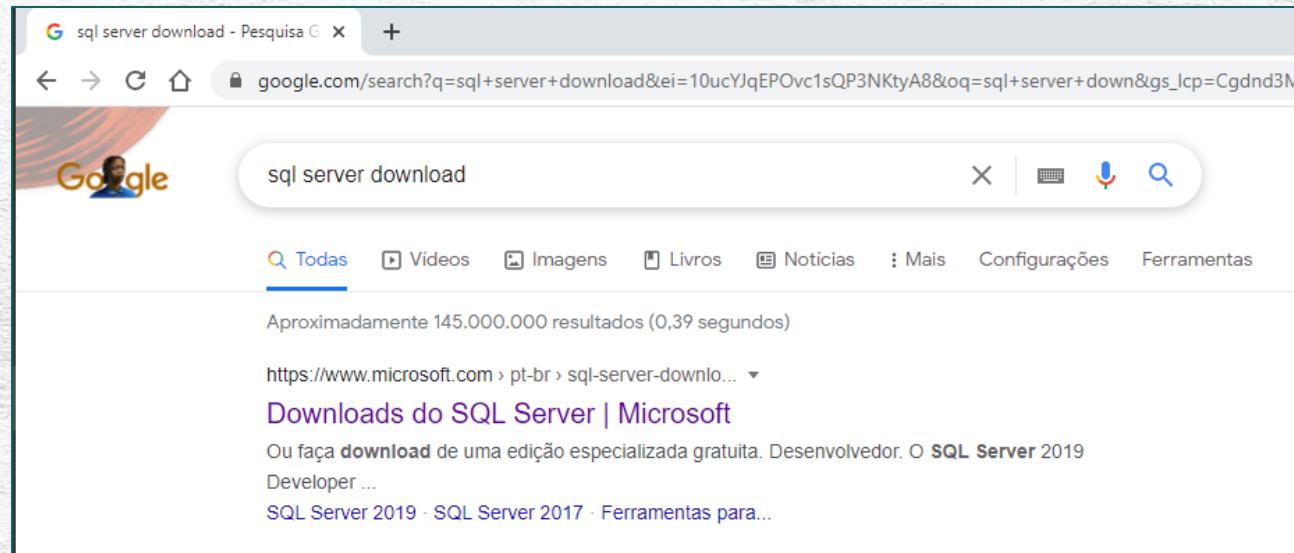
Instalando o SQL Server

Instalando o SQL Server

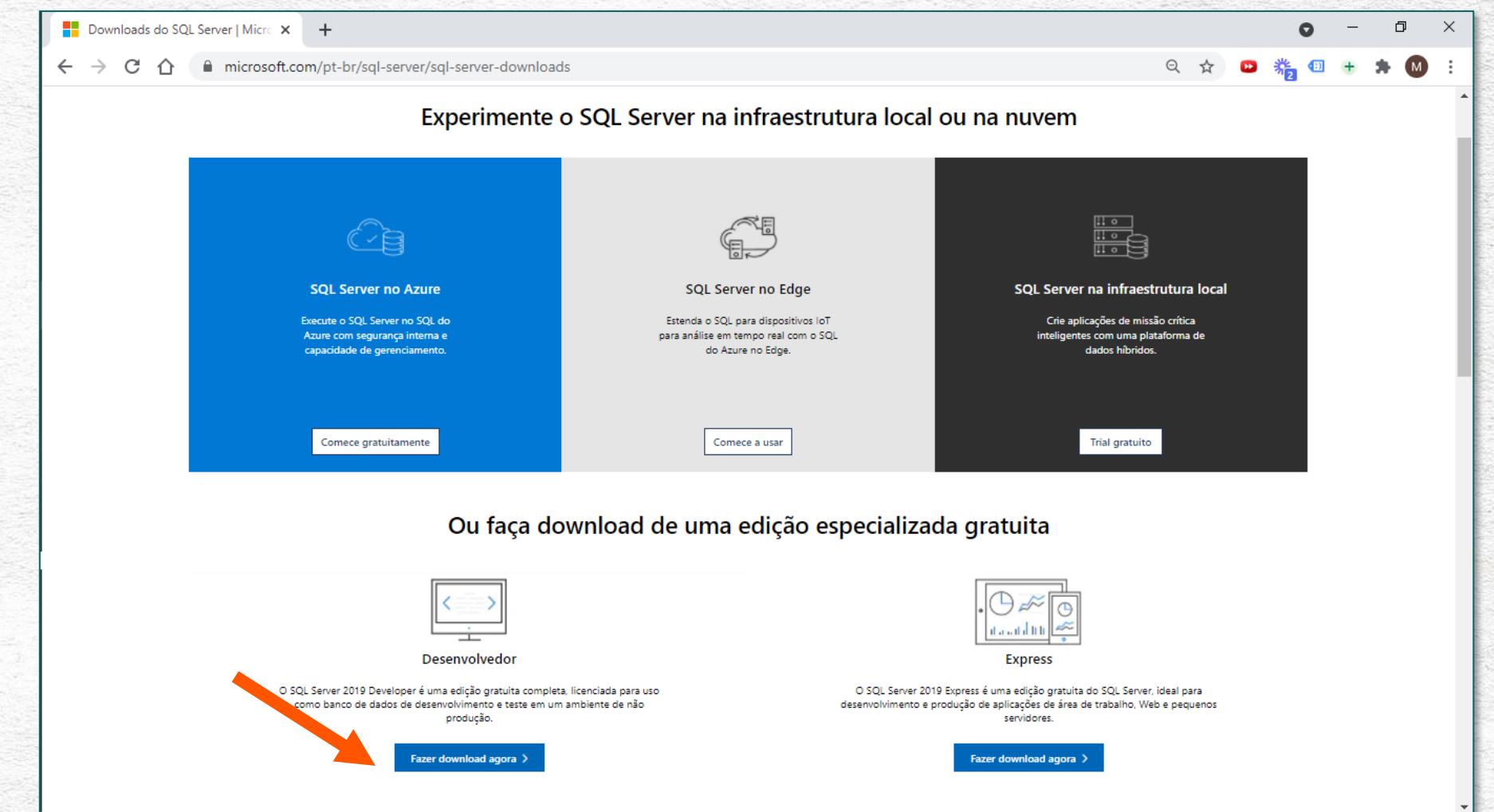


# Instalando o SQL Server

Para fazer o download do SQL Server, basta procurar no Google por “sql server download”. Clique no primeiro link que aparecer.

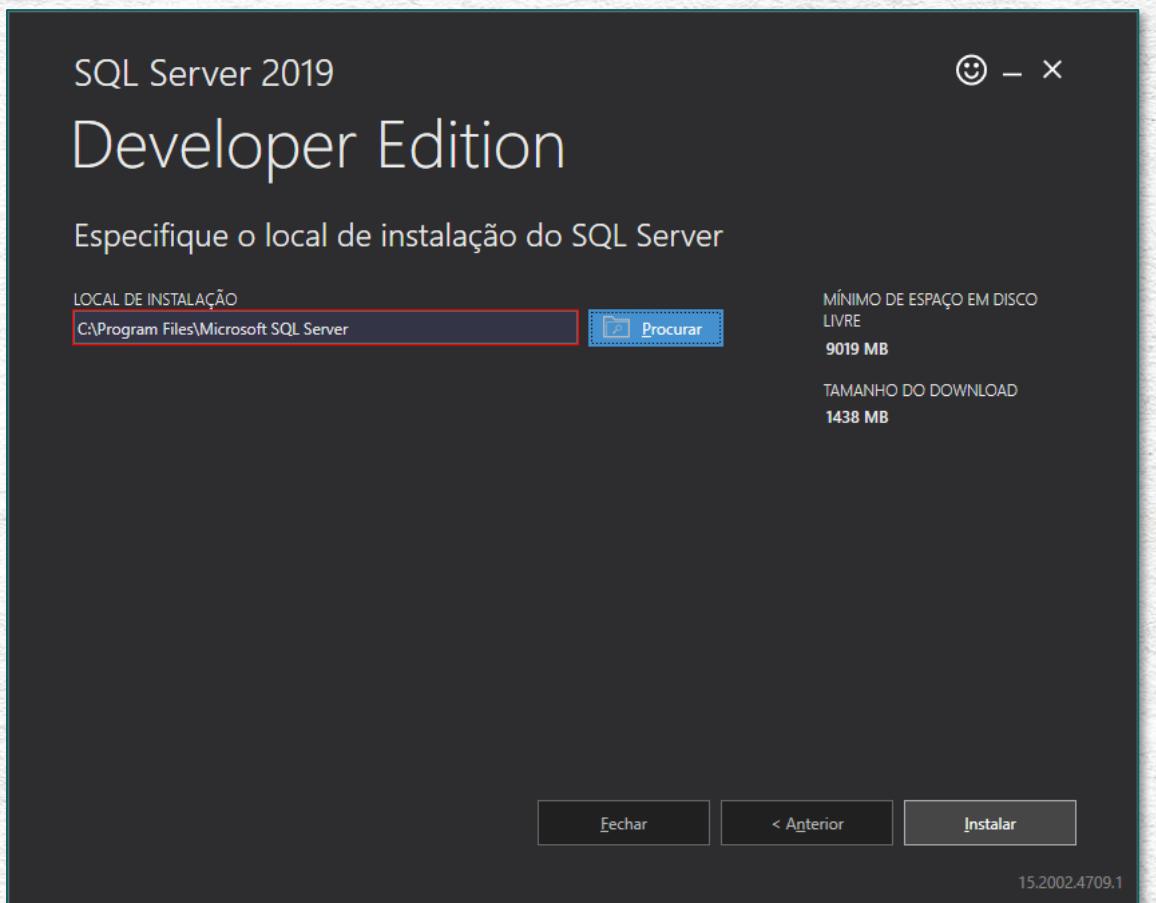
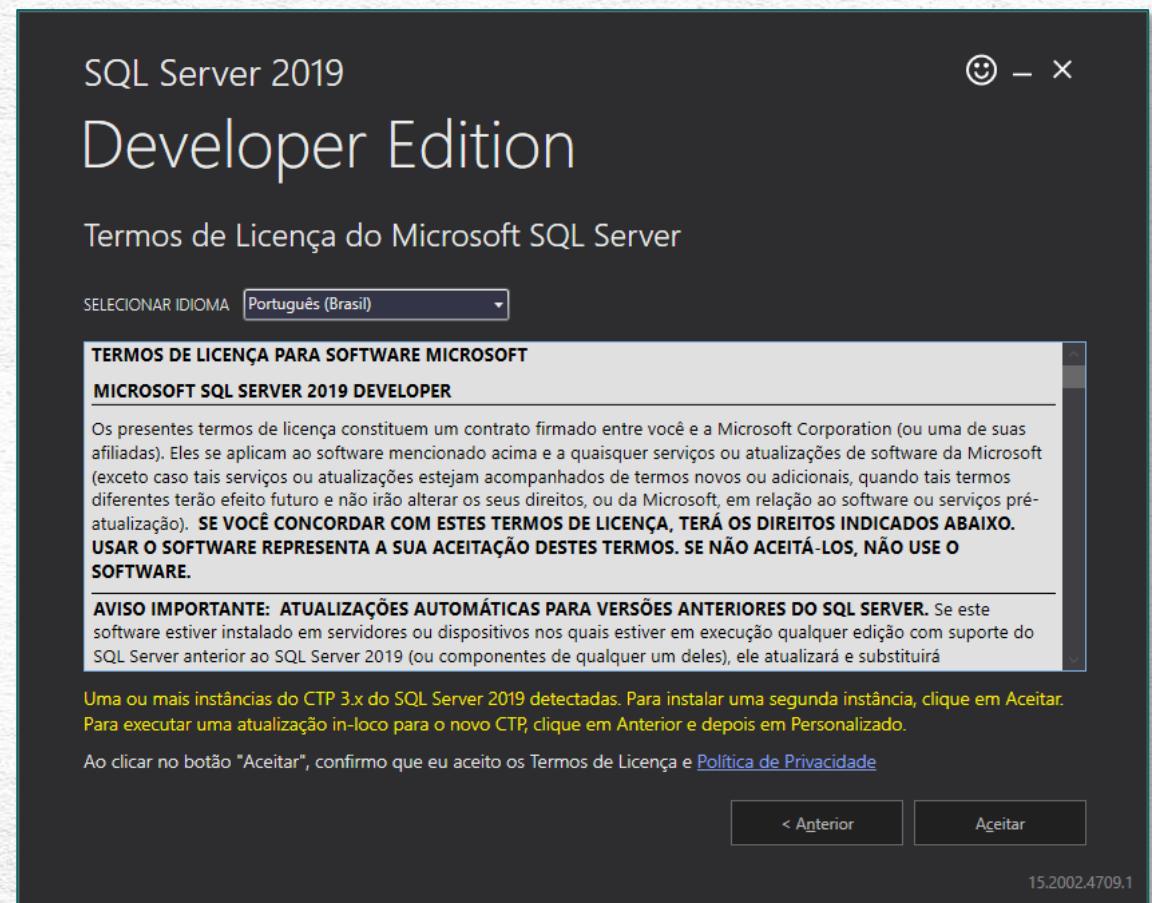


Em seguida, procure pela opção “Desenvolvedor” assim que aparecer a janela ao lado.



# Instalando o SQL Server

Agora é só escolher a opção ‘Básica’ e aceitar todos os termos até instalar o programa.





# TEMA 10

Instalando o SSMS

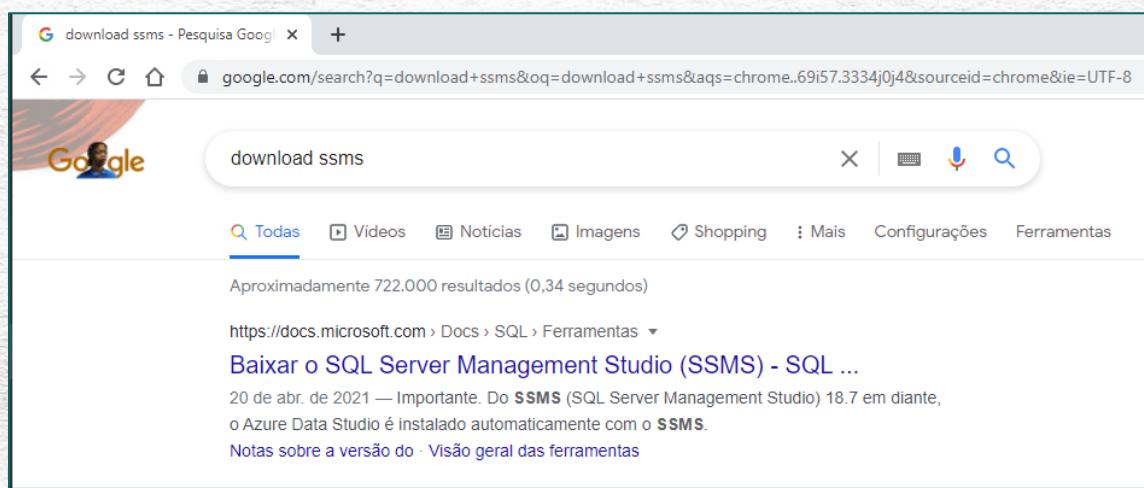
Instalando o SSMS

Instalando o SSMS



# Instalando o SQL Server Management Studio (SSMS)

Para fazer o download do SSMS, basta procurar no Google por “sql server download”. Clique no primeiro link que aparecer.



Em seguida, procure pela opção ao lado assim que aparecer a janela ao lado. Aceite os termos da instalação, não é necessário nenhuma configuração diferente do padrão sugerido.

Baixar o SQL Server Management Studio (SSMS) é um ambiente integrado para gerenciar qualquer infraestrutura de SQL, do SQL Server para o Banco de Dados SQL do Azure. O SSMS fornece ferramentas para configurar, monitorar e administrar instâncias do SQL Server e bancos de dados. Use o SSMS para implantar, monitorar e atualizar os componentes da camada de dados usados pelos seus aplicativos, além de criar consultas e scripts.

Use o SSMS para consultar, criar e gerenciar seus bancos de dados e data warehouses, independentemente de onde estiverem – no computador local ou na nuvem.

## Baixar o SSMS

**Baixar o SSMS (SQL Server Management Studio) 18.9.1**

SSMS 18.9.1 é a versão mais recente em GA (disponibilidade geral). Se você tiver uma versão anterior em GA do SSMS 18 instalada, a instalação do SSMS 18.9.1 atualizará o produto para a versão 18.9.1.

- Número da versão: 18.9.1
- Número de build: 15.0.18384.0
- Data de lançamento: 20 de abril de 2021



# TEMA 11

Download do BD Contoso

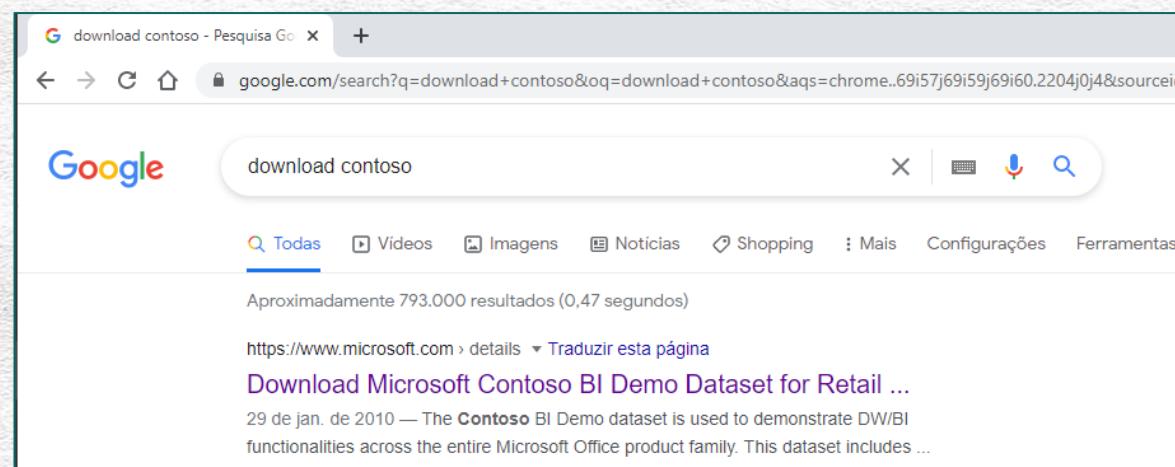
Download do BD Contoso

Download do BD Contoso



# Download do BD Contoso e abrindo no SSMS

Para fazer o download do BD Contoso, basta procurar no Google por “download contoso”. Clique no primeiro link que aparecer.



Na janela ao lado, selecione a opção em inglês e depois clique em download. As bases de dados da Contoso não possuem uma versão em português.

O arquivo a ser baixado deve ser o ContosoBIdemoBAK.exe

Microsoft Contoso BI Demo Dataset for Retail Industry

Important! Selecting a language below will dynamically change the complete page content to that language.

Select Language: English

A fictitious retail demo dataset used for presenting Microsoft Business Intelligence products.

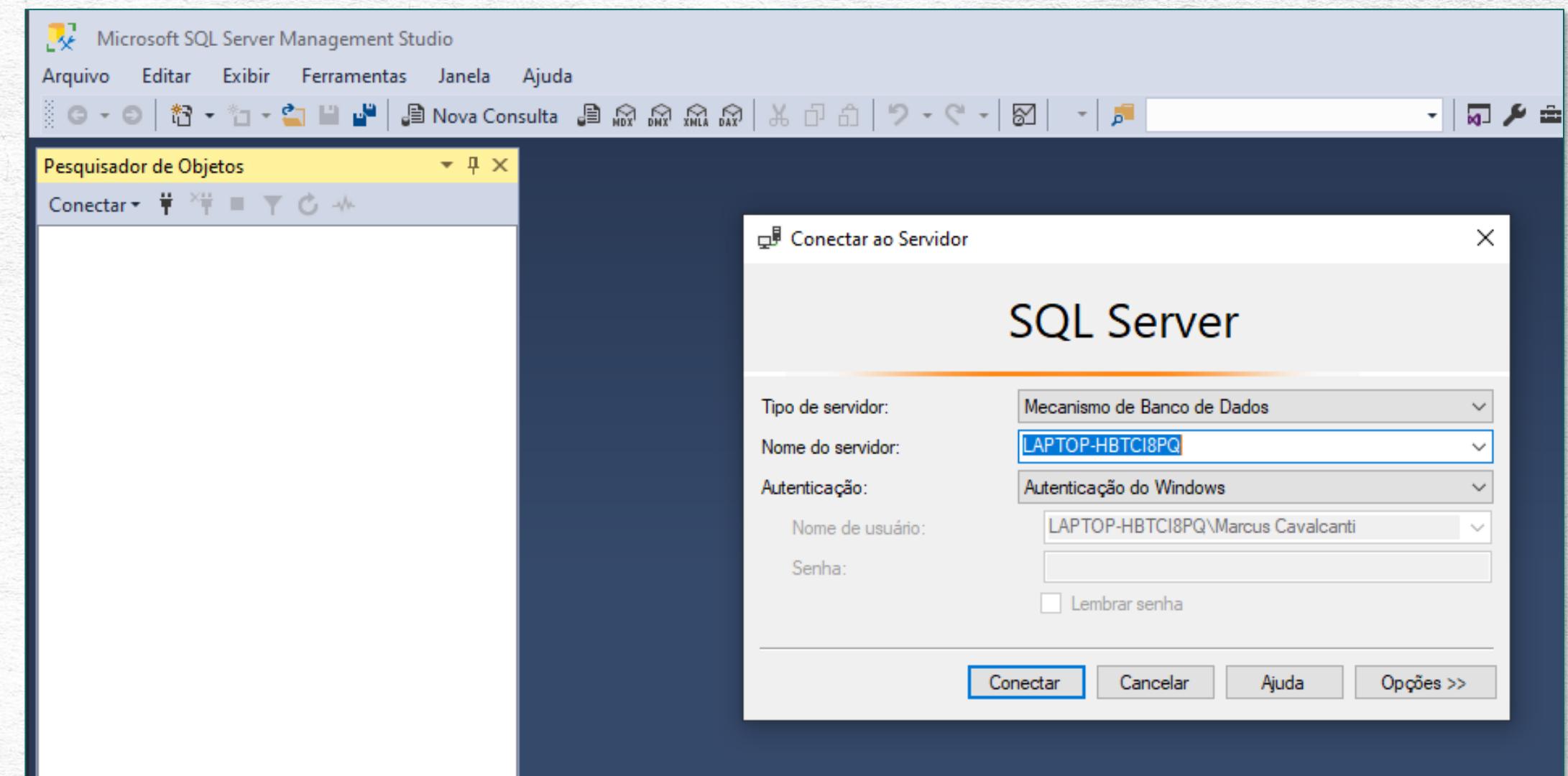
Choose the download you want

File Name	Size
<input type="checkbox"/> ContosoBIdemoBAK.exe	626.5 MB
<input type="checkbox"/> ContosoBIdemoABF.exe	433.4 MB

# Download do BD Contoso e abrindo no SSMS

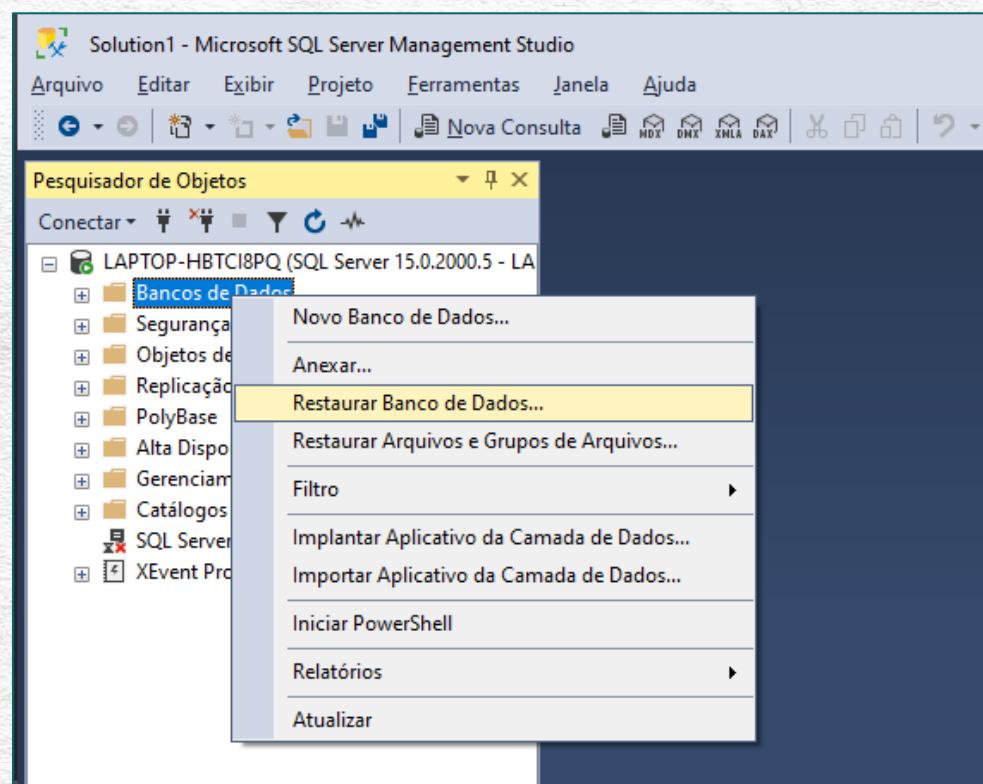
Abra o programa SSMS no seu computador.

Ao aparecer a janela ao lado, clique em **Conectar**. Por enquanto, podemos utilizar as configurações padrão do SQL Server.

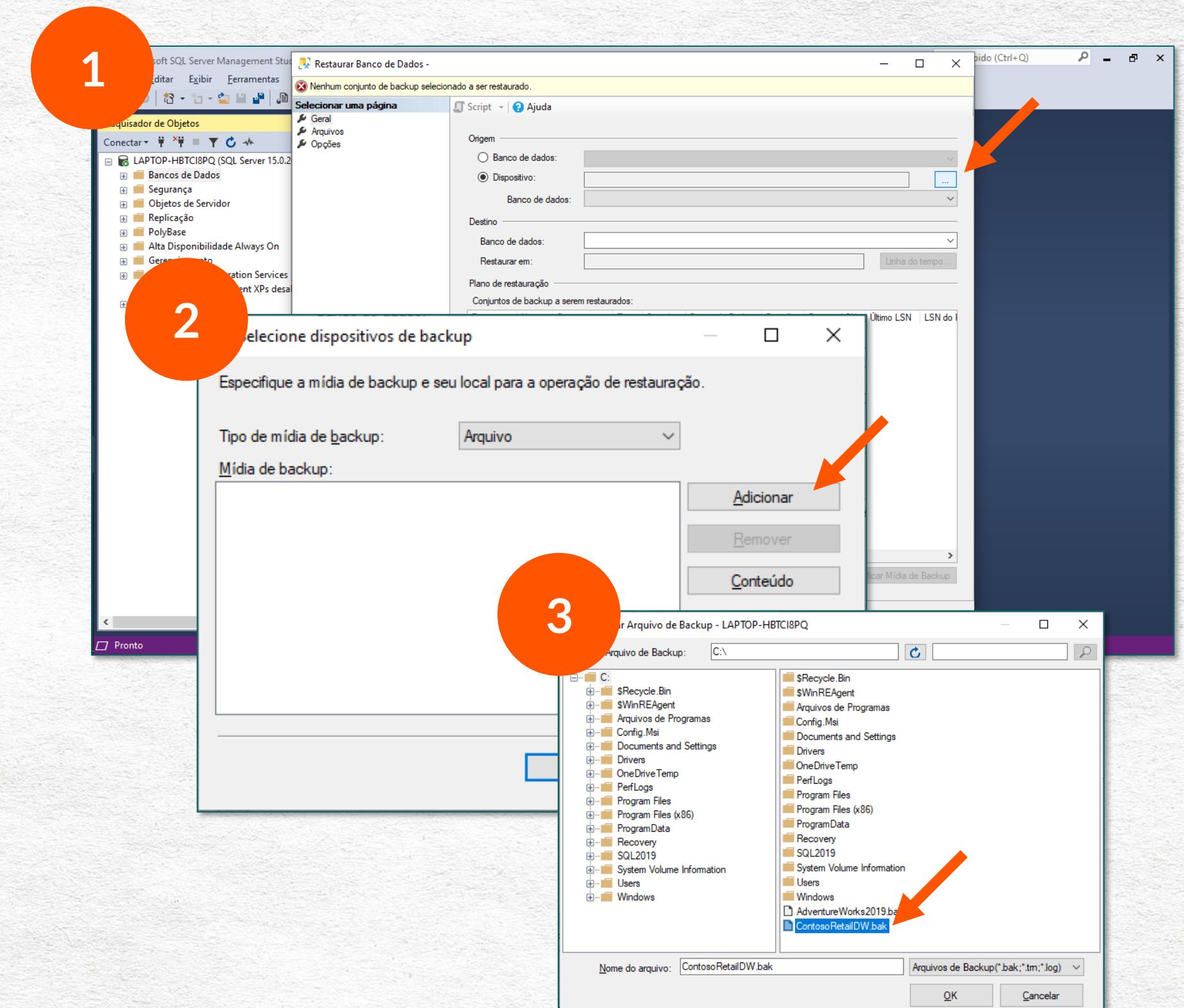


# Download do BD Contoso e abrindo no SSMS

Para abrir o BD Contoso no SSMS, clique com o botão direito em Banco de Dados > Restaurar Banco de Dados...



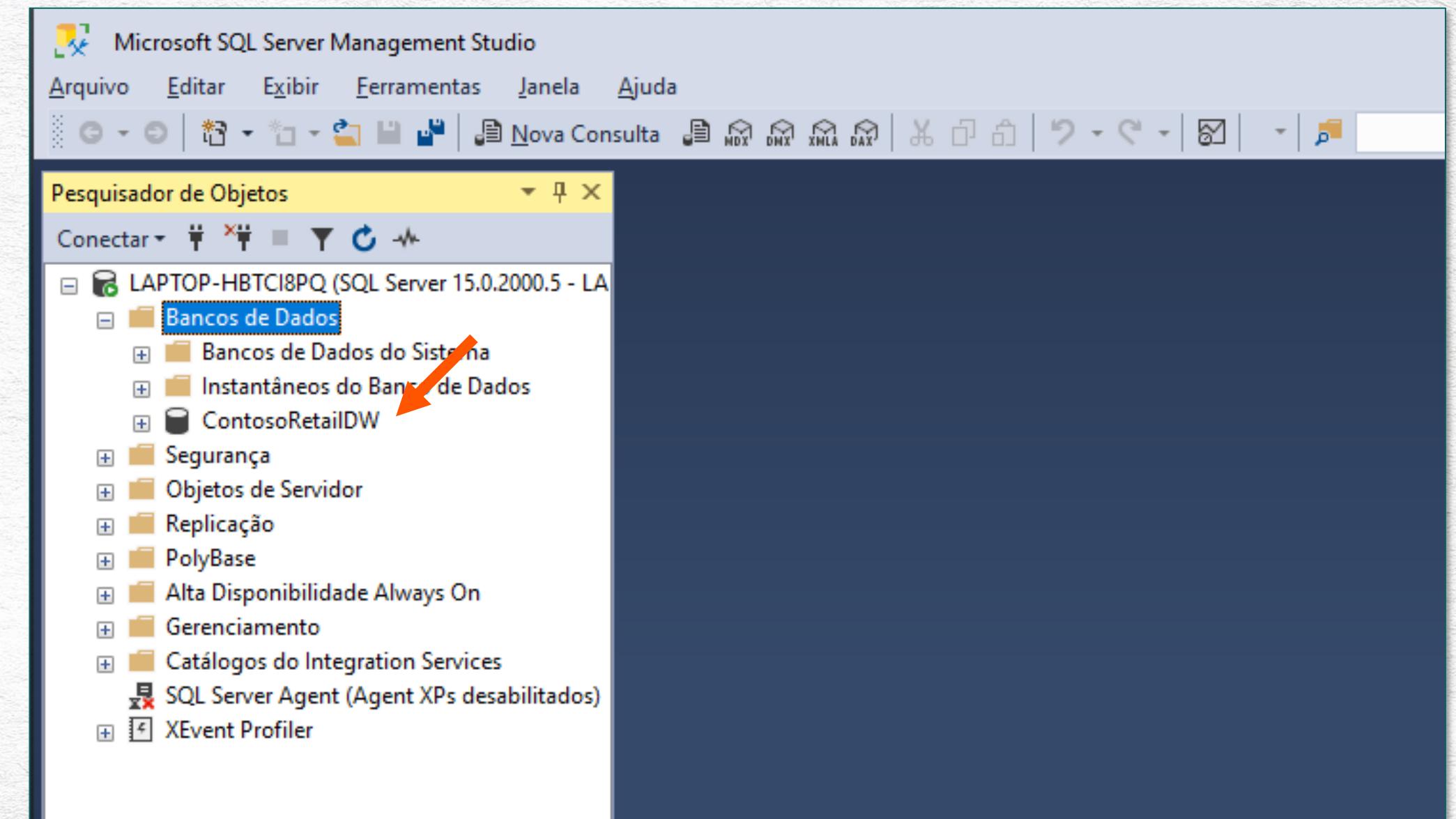
Na janela ao lado, troque a Origem para ‘Dispositivo’ e procure a pasta onde você salvou o arquivo BAK do Contoso. Por simplicidade, recomendamos que salve na unidade C: do seu computador.



Depois é só clicar em Ok até o BD aparecer no SSMS.

# Download do BD Contoso e abrindo no SSMS

Pronto! O BD já foi restaurado e já pode ser acessado no SSMS.



# TEMA 12

Conhecendo o BD Contoso

Conhecendo o BD Contoso

Conhecendo o BD Contoso

# Conhecendo o BD Contoso

## Vamos conhecer um pouco mais sobre a Contoso.

- A empresa Contoso Corporation é uma empresa multinacional (fictícia) com sede em Paris e com escritórios espalhados ao longo de todo o mundo.
  - A empresa é uma organização de fabricação, vendas e suporte, com mais de **100.000 produtos**.
  - Por ser uma empresa multinacional, utiliza um SGBD da Microsoft (SQL Server) para centralizar e manipular o grande volume de dados associado ao negócio.
  - Possui mais de **25 mil funcionários**, espalhados ao longo das dezenas de escritórios no mundo.



# Conhecendo o BD Contoso

O Banco de Dados possui uma série de informações, divididas em diferentes tabelas. As principais são destacadas abaixo:

## dimCustomer

Tabela contendo informações dos **clientes**, tais como: Primeiro Nome, Data de Nascimento, Email, etc.

## dimEmployee

Tabela contendo informações dos **funcionários**, tais como: Primeiro Nome, Cargo, Departamento, etc.

## dimProduct

Tabela contendo informações dos **produtos**, tais como: Nome do Produto, Marca, Cor, etc.

## dimStore

Tabela contendo informações das **lojas**, tais como: Nome, Data de Abertura, Contato, etc.

## factSales

Tabela contendo informações de **vendas**, tais como: Produto vendido, Quantidade, Preço, etc.

## factOnlineSales

Tabela contendo informações de **vendas online**, muito semelhante à tabela factSales.

# TEMA 13

Conhecendo o BD Contoso

Conhecendo o BD Contoso

Conhecendo o BD Contoso

# Como explorar as bases de dados pela primeira vez

Sempre que você for trabalhar em cima de bancos de dados, antes de criar qualquer código SQL, tenha o bom costume de explorar os dados à sua disposição.

A maneira mais simples de fazer isso é clicar com o botão direito em cada tabela e escolher a opção ‘Selecionar 1000 linhas superiores’.

Isso vai te dar uma visão dos dados de cada tabela. Essa prática é importante para você conhecer os dados de uma tabela quando você está trabalhando com Bancos de Dados pela primeira vez.

The screenshot shows the Microsoft SQL Server Management Studio interface. On the left, the Object Explorer displays the database structure of 'ContosoRetailDW'. A right-click context menu is open over the 'dbo.DimCustomer' table, with the option 'Selecionar 1000 Linhas Superiores' highlighted and a red arrow pointing to it. To the right, the Results pane shows the output of the query 'SELECT TOP (1000) [CustomerKey], [GeographyKey], [CustomerLabel], [Title], [FirstName], [MiddleName], [LastName], [NameStyle], [BirthDate], [MaritalStatus], [Suffix], [Gender], [EmailAddress], [YearlyIncome], [TotalChildren], [NumberChildrenAtHome], [Education], [Occupation] FROM [ContosoRetailDW].[dbo].[DimCustomer]'. The results show 11 rows of customer data from the Adventure Works database.

CustomerKey	GeographyKey	CustomerLabel	Title	FirstName	MiddleName	LastName	NameStyle	BirthDate	MaritalStatus	Suffix	Gender	EmailAddress
1	680	11000	NULL	Jon	V	Yang	0	1966-04-08	M	NULL	M	jon24@adventure-works.com
2	692	11001	NULL	Eugene	L	Huang	0	1985-05-14	S	NULL	M	eugene10@adventure-works.com
3	493	11002	NULL	Ruben	NULL	Torres	0	1965-08-12	M	NULL	M	ruben35@adventure-works.com
4	519	11003	NULL	Christy	NULL	Zhu	0	1968-02-15	S	NULL	F	christy12@adventure-works.com
5	706	11004	NULL	Elizabeth	NULL	Johnson	0	1968-08-08	S	NULL	F	elizabeth5@adventure-works.com
6	478	11005	NULL	Julio	NULL	Ruiz	0	1965-08-05	S	NULL	M	julio1@adventure-works.com
7	509	11006	NULL	Janet	G	Alvarez	0	1965-12-06	S	NULL	F	janet9@adventure-works.com
8	568	11007	NULL	Marco	NULL	Mehta	0	1964-05-09	M	NULL	M	marco14@adventure-works.com
9	425	11008	NULL	Rob	NULL	Verhoff	0	1964-07-07	S	NULL	F	rob4@adventure-works.com
10	492	11009	NULL	Shannon	C	Carlson	0	1964-04-01	S	NULL	M	shannon38@adventure-works.com
11	478	11010	NULL	Jacquelyn	C	Suarez	0	1964-02-06	S	NULL	F	jacquelyn20@adventure-works.com

# SQL

## Introdução ao SQL



# O comando SELECT

O **SELECT** é um comando para selecionar dados de uma tabela.

Ao lado, temos a estrutura para selecionar apenas colunas específicas da tabela.

Seleciona uma única coluna:

```
SQLQuery1.sql - LAP...us Cavalcanti (55)* ▾ X
└-- Seleciona a coluna FirstName
    -- da tabela dimCustomer
SELECT FirstName
FROM dimCustomer
```

	FirstName
1	Jon
2	Eugene
3	Ruben
4	Christy
5	Elizabeth
6	Julio
7	Janet
8	Marco
9	Rob
10	Shannon
11	Jacquelyn
12	Curtis

Seleciona mais de uma coluna:

```
SQLQuery1.sql - LAP...us Cavalcanti (55)* ▾ X
└-- Seleciona a coluna
    -- FirstName da tabela dimCustomer
SELECT FirstName, LastName, EmailAddress
FROM dimCustomer
```

	FirstName	LastName	EmailAddress
1	Jon	Yang	jon24@adventure-works.com
2	Eugene	Huang	eugene10@adventure-works.com
3	Ruben	Torres	ruben35@adventure-works.com
4	Christy	Zhu	christy12@adventure-works.com
5	Elizabeth	Johnson	elizabeth5@adventure-works.com
6	Julio	Ruiz	julio1@adventure-works.com
7	Janet	Alvarez	janet9@adventure-works.com
8	Marco	Mehta	marco14@adventure-works.com
9	Rob	Verhoff	rob4@adventure-works.com
10	Shannon	Carlson	shannon38@adventure-works.com
11	Jacquelyn	Suarez	jacquelyn20@adventure-works.com
12	Curtis	Lu	curtis9@adventure-works.com

# O comando SELECT\*

O comando **SELECT \*** permite selecionar TODAS as colunas de uma determinada tabela. Não é muito recomendável para tabelas muito grandes. O objetivo deste comando é ter uma rápida visualização de toda a tabela.

Será melhor usado em conjunto com o comando TOP.

Seleciona toda a tabela:

```
SQLQuery1.sql - LAP...us Cavalcanti (55)* ✎ ×
-- Seleciona todas as linhas e colunas
-- da tabela dimCustomer
SELECT *
FROM dimCustomer
```

	CustomerKey	GeographyKey	CustomerLabel	Title	FirstName	MiddleName	LastName	NameStyle	BirthDate	MaritalStatus	Suffix	Gender	EmailAddress
1	1	680	11000	NULL	Jon	V	Yang	0	1966-04-08	M	NULL	M	jon24@adventure-works.com
2	2	692	11001	NULL	Eugene	L	Huang	0	1965-05-14	S	NULL	M	eugene10@adventure-works.c
3	3	493	11002	NULL	Ruben	NULL	Torres	0	1965-08-12	M	NULL	M	ruben35@adventure-works.co
4	4	519	11003	NULL	Christy	NULL	Zhu	0	1968-02-15	S	NULL	F	christy12@adventure-works.cc
5	5	706	11004	NULL	Elizabeth	NULL	Johnson	0	1968-08-08	S	NULL	F	elizabeth5@adventure-works.c
6	6	478	11005	NULL	Julio	NULL	Ruiz	0	1965-08-05	S	NULL	M	julio1@adventure-works.com
7	7	509	11006	NULL	Janet	G	Alvarez	0	1965-12-06	S	NULL	F	janet9@adventure-works.com
8	8	568	11007	NULL	Marco	NULL	Mehta	0	1964-05-09	M	NULL	M	marco14@adventure-works.co
9	9	425	11008	NULL	Rob	NULL	Verhoff	0	1964-07-07	S	NULL	F	rob4@adventure-works.com
10	10	492	11009	NULL	Shannon	C	Carlson	0	1964-04-01	S	NULL	M	shannon38@adventure-works.
11	11	478	11010	NULL	Jacquelyn	C	Suarez	0	1964-02-06	S	NULL	F	jacquelyn20@adventure-works.v

# O comando SELECT TOP(N)

O comando **SELECT TOP(N)** permite que a gente selecione apenas as N primeiras linhas de uma tabela.

**Selecionar as TOP(N) linhas da tabela:**

```
SQLQuery1.sql - LAP...us Cavalcanti (55)* ✎ X
-- Retorna as 10 primeiras
-- linhas da tabela
SELECT TOP(10) ProductName
FROM dimProduct
```

	ProductName
1	Contoso 512MB MP3 Player E51 Silver
2	Contoso 512MB MP3 Player E51 Blue
3	Contoso 1G MP3 Player E100 White
4	Contoso 2G MP3 Player E200 Silver
5	Contoso 2G MP3 Player E200 Red
6	Contoso 2G MP3 Player E200 Black
7	Contoso 2G MP3 Player E200 Blue
8	Contoso 4G MP3 Player E400 Silver
9	Contoso 4G MP3 Player E400 Black
10	Contoso 4G MP3 Player E400 Green

**Selecionar as TOP(N) % linhas da tabela :**

```
SQLQuery1.sql - LAP...us Cavalcanti (55)* ✎ X
-- Retorna as 10% primeiras linhas da tabela.
-- Se a tabela tem 100 linhas, ele retorna
-- as 10 primeiras
SELECT TOP(10) PERCENT ProductName
FROM dimProduct
```

	ProductName
1	Contoso 512MB MP3 Player E51 Silver
2	Contoso 512MB MP3 Player E51 Blue
3	Contoso 1G MP3 Player E100 White
4	Contoso 2G MP3 Player E200 Silver
5	Contoso 2G MP3 Player E200 Red
6	Contoso 2G MP3 Player E200 Black
7	Contoso 2G MP3 Player E200 Blue
8	Contoso 4G MP3 Player E400 Silver
9	Contoso 4G MP3 Player E400 Black
10	Contoso 4G MP3 Player E400 Green
11	Contoso 4G MP3 Player E400 Orange
12	Contoso 4GB Flash MP3 Player E401 Blue

252 linhas



# O comando SELECT DISTINCT

O comando **SELECT DISTINCT** retorna os valores distintos de uma tabela.

Retorna todas as linhas da tabela:

```
SQLQuery1.sql - LAP...us Cavalcanti (55)* ✎ X
-- Retorna todas as linhas
-- da coluna ColorName
SELECT ColorName
FROM dimProduct
```

	ColorName
1	Silver
2	Blue
3	White
4	Silver
5	Red
6	Black
7	Blue
8	Silver
9	Black
10	Green
11	Orange
12	Blue

Retorna valores  
repetidos!

Retorna apenas as cores únicas da tabela:

```
SQLQuery1.sql - LAP...us Cavalcanti (55)* ✎ X
-- Retorna apenas os valores
-- distintos da coluna ColorName
SELECT DISTINCT ColorName
FROM dimProduct
```

	ColorName
1	Purple
2	Azure
3	Transparent
4	Silver
5	Green
6	Pink
7	Grey
8	Red
9	Brown
10	Gold
11	Black
12	Blue

Nenhum valor  
repetido!

# Comentando o código

Comentários são uma boa prática para garantir um bom entendimento de um código.

Conforme vamos criando consultas cada vez mais complexas, os comentários podem ser muito úteis para ajudar no entendimento do que está sendo feito.

Existem duas formas de comentar códigos:

Na opção 1), utilizamos o hífen duplo para comentar uma única linha de código.

Já na opção 2), utilizamos uma barra seguida de um asterisco para identificar onde começa um comentário, e utilizamos o asterisco seguido de uma barra para identificar onde termina um comentário.

Ao lado, um exemplo de cada situação.

## (1) O -- comenta linha por linha do código

```
-- Isso é um comentário
-- O código abaixo seleciona a coluna
-- FirstName da tabela
SELECT FirstName
FROM dimCustomer
```

## (2) O /\* ... \*/ comenta mais de 1 linha por vez

```
/* SELECT TOP(10) PERCENT ProductName
   FROM dimProduct*/
```

# AS: Renomeando (aliasing) colunas

Uma coluna de uma tabela pode ser renomeada por meio do comando **AS**. O nome da nova coluna pode ser escrito sem as aspas simples, apenas se for um texto único. Se precisar renomear com um nome composto, utilize as aspas simples.

```
SQLQuery1.sql - LAP...us Cavalcanti (55)* + X
-- Retorna 3 colunas da tabela dimProduct,
-- e renomeia cada uma delas
SELECT
    ProductName AS Produto,
    BrandName AS Marca,
    ColorName AS Cor
    FROM DimProduct
```

```
SQLQuery1.sql - LAP...us Cavalcanti (55)* + X
-- Retorna 3 colunas da tabela dimProduct,
-- e renomeia cada uma delas
SELECT
    ProductName AS 'Nome Produto',
    BrandName AS 'Nome Marca',
    ColorName AS 'Nome Cor'
    FROM DimProduct
```

	Resultado	Mensagens	
	Produto	Marca	Cor
1	Contoso 512MB MP3 Player E51 Silver	Contoso	Silver
2	Contoso 512MB MP3 Player E51 Blue	Contoso	Blue
3	Contoso 1G MP3 Player E100 White	Contoso	White
4	Contoso 2G MP3 Player E200 Silver	Contoso	Silver
5	Contoso 2G MP3 Player E200 Red	Contoso	Red
6	Contoso 2G MP3 Player E200 Black	Contoso	Black
7	Contoso 2G MP3 Player E200 Blue	Contoso	Blue
8	Contoso 4G MP3 Player E400 Silver	Contoso	Silver
9	Contoso 4G MP3 Player E400 Black	Contoso	Black
10	Contoso 4G MP3 Player E400 Green	Contoso	Green
11	Contoso 4G MP3 Player E400 Orange	Contoso	Orange
12	Contoso 4GB Flash MP3 Player E401 Blue	Contoso	Blue

	Nome Produto	Nome Marca	Nome Cor
1	Contoso 512MB MP3 Player E51 Silver	Contoso	Silver
2	Contoso 512MB MP3 Player E51 Blue	Contoso	Blue
3	Contoso 1G MP3 Player E100 White	Contoso	White
4	Contoso 2G MP3 Player E200 Silver	Contoso	Silver
5	Contoso 2G MP3 Player E200 Red	Contoso	Red
6	Contoso 2G MP3 Player E200 Black	Contoso	Black
7	Contoso 2G MP3 Player E200 Blue	Contoso	Blue
8	Contoso 4G MP3 Player E400 Silver	Contoso	Silver
9	Contoso 4G MP3 Player E400 Black	Contoso	Black
10	Contoso 4G MP3 Player E400 Green	Contoso	Green
11	Contoso 4G MP3 Player E400 Orange	Contoso	Orange
12	Contoso 4GB Flash MP3 Player E401 Blue	Contoso	Blue

# EXERCÍCIOS



1

## Questão 1

Você é responsável por controlar os dados de clientes e de produtos da sua empresa. O que você precisará fazer é confirmar se:

- Existem 2.517 produtos cadastrados na base e, se não tiver, você deverá reportar ao seu gestor para saber se existe alguma defasagem no controle dos produtos.
- Até o mês passado, a empresa tinha um total de 19.500 clientes na base de controle. Verifique se esse número aumentou ou reduziu.

2

## Questão 2

Você trabalha no setor de marketing da empresa Contoso e acaba de ter uma ideia de oferecer descontos especiais para os clientes no dia de seus aniversários. Para isso, você vai precisar listar todos os clientes e as suas respectivas datas de nascimento, além de um contato.

- Selecione as colunas: CustomerKey, FirstName, EmailAddress, BirthDate da tabela dimCustomer.
- Renomeie as colunas dessa tabela usando o alias (comando AS).

# EXERCÍCIOS

3

## Questão 3

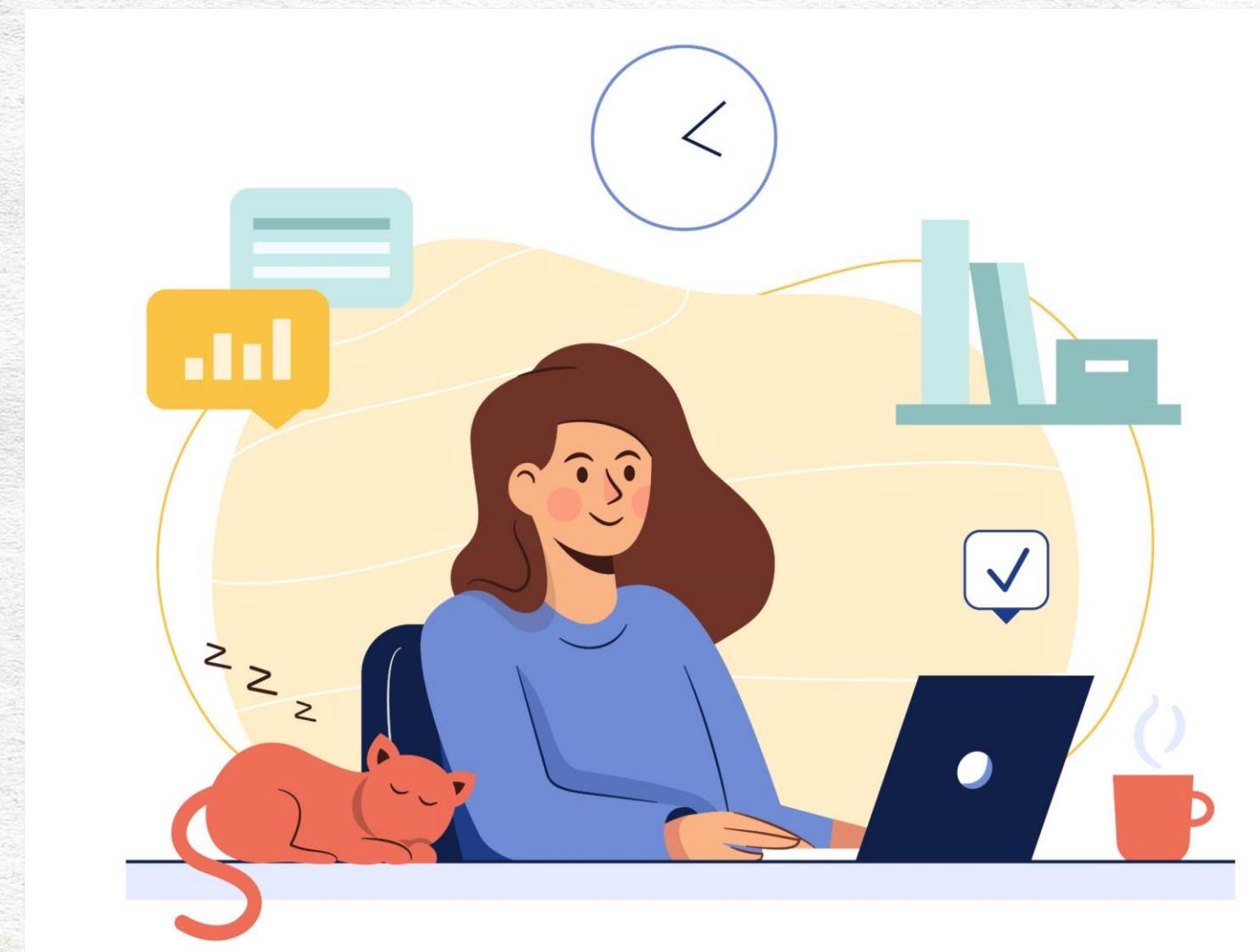
A Contoso está comemorando aniversário de inauguração de 10 anos e pretende fazer uma ação de premiação para os clientes. A empresa quer presentear os primeiros clientes desde a inauguração.

Você foi alocado para levar adiante essa ação. Para isso, você terá que fazer o seguinte:

- a. A Contoso decidiu presentear os primeiros 100 clientes da história com um vale compras de R\$ 10.000. Utilize um comando em SQL para retornar uma tabela com os primeiros 100 primeiros clientes da tabela dimCustomer (selecione todas as colunas).
- b. A Contoso decidiu presentear os primeiros 20% de clientes da história com um vale compras de R\$ 2.000. Utilize um comando em SQL para retornar 10% das linhas da sua tabela dimCustomer (selecione todas as colunas).
- c. Adapte o código do item a) para retornar apenas as 100 primeiras linhas, mas apenas as colunas FirstName, EmailAddress, BirthDate.
- d. Renomeie as colunas anteriores para nomes em português



# EXERCÍCIOS



4

## Questão 4

A empresa Contoso precisa fazer contato com os fornecedores de produtos para repor o estoque. Você é da área de compras e precisa descobrir quem são esses fornecedores.

Utilize um comando em SQL para retornar apenas os nomes dos fornecedores na tabela dimProduct e renomeie essa nova coluna da tabela.

5

## Questão 5

O seu trabalho de investigação não para. Você precisa descobrir se existe algum produto registrado na base de produtos que ainda não tenha sido vendido. Tente chegar nessa informação.

Obs: caso tenha algum produto que ainda não tenha sido vendido, você não precisa descobrir qual é, é suficiente saber se teve ou não algum produto que ainda não foi vendido.

# GABARITOS

1

## Questão 1

Você é responsável por controlar os dados de clientes e de produtos da sua empresa. O que você precisará fazer é confirmar se:

- a. Existem 2.517 produtos cadastrados na base e, se não tiver, você deverá reportar ao seu gestor para saber se existe alguma defasagem no controle dos produtos. **A quantidade de produtos se manteve.**
- b. Até o mês passado, a empresa tinha um total de 19.500 clientes na base de controle. Verifique se esse número aumentou ou reduziu. **A quantidade de clientes reduziu!**

```
--Questão 1
--a)
SELECT * FROM DimProduct
| 2.517 linhas | ✓

--b)
SELECT * FROM DimCustomer
| 18.869 linhas | ✗
```

# GABARITOS

2

## Questão 2

Você trabalha no setor de marketing da empresa Contoso e acaba de ter uma ideia de oferecer descontos especiais para os clientes no dia de seus aniversários. Para isso, você vai precisar listar todos os clientes e as suas respectivas datas de nascimento, além de um contato.

- Selecione as colunas: CustomerKey, FirstName, EmailAddress, BirthDate da tabela dimCustomer.
- Renomeie as colunas dessa tabela usando o alias (comando AS).

The screenshot shows a SQL Server Management Studio window. The top part is a script pane with the following SQL code:

```

SQLQuery9.sql - LAP...us Cavalcanti (54)*  X
--Questão 2
--a) e b)

SELECT
    CustomerKey AS 'ID do Cliente',
    FirstName AS 'Nome',
    EmailAddress AS 'E-mail',
    BirthDate AS 'Data de Nascimento'
FROM
    DimCustomer
  
```

The bottom part is a results grid titled "Resultados" showing 18 rows of data from the DimCustomer table. The columns are: CustomerKey, GeographyKey, CustomerLabel, Title, FirstName, MiddleName, LastName, NameStyle, BirthDate, MaritalStatus, Suffix, Gender, and EmailAddress. The data includes various names like Jon, Eugene, Ruben, Christy, Elizabeth, Julio, Janet, and their respective birthdates and contact information.

	CustomerKey	GeographyKey	CustomerLabel	Title	FirstName	MiddleName	LastName	NameStyle	BirthDate	MaritalStatus	Suffix	Gender	EmailAddress
1	1	680	11000	NULL	Jon	V	Yang	0	1966-04-08	M	NULL	M	jon24@adventure-works.com
2	2	692	11001	NULL	Eugene	L	Huang	0	1965-05-14	S	NULL	M	eugene10@adventure-works.c
3	3	493	11002	NULL	Ruben	NULL	Torres	0	1965-08-12	M	NULL	M	ruben35@adventure-works.cor
4	4	519	11003	NULL	Christy	NULL	Zhu	0	1968-02-15	S	NULL	F	christy12@adventure-works.co
5	5	706	11004	NULL	Elizabeth	NULL	Johnson	0	1968-08-08	S	NULL	F	elizabeth5@adventure-works.c
6	6	478	11005	NULL	Julio	NULL	Ruiz	0	1965-08-05	S	NULL	M	julio1@adventure-works.com
7	7	509	11006	NULL	Janet	G	Alvarez	0	1965-12-06	S	NULL	F	janet9@adventure-works.com

Consulta executada com êxito. | LAPTOP-HBTCl8PQ (15.0 RTM) | LAPTOP-HBTCl8PQ\Marcus... | ContosoRetailDW | 00:00:06 | 18.869 linhas

# GABARITOS

3

## Questão 3

A Contoso está comemorando aniversário de inauguração de 10 anos e pretende fazer uma ação de premiação para os clientes. A empresa quer presentear os primeiros clientes desde a inauguração.

Você foi alocado para levar adiante essa ação. Para isso, você terá que fazer o seguinte:

- A Contoso decidiu presentear os primeiros 100 clientes da história com um vale compras de R\$ 10.000. Utilize um comando em SQL para retornar uma tabela com os primeiros 100 primeiros clientes da tabela dimCustomer (selecione todas as colunas).

```
--Questão 3
--a)
SELECT TOP(100)
*
FROM
DimCustomer
```

	CustomerKey	GeographyKey	CustomerLabel	Title	FirstName	MiddleName	LastName	NameStyle	BirthDate	MaritalStatus	Suffix	Gender	EmailAddress
1	1	680	11000	NULL	Jon	V	Yang	0	1966-04-08	M	NULL	M	jon24@adventure-works.com
2	2	692	11001	NULL	Eugene	L	Huang	0	1965-05-14	S	NULL	M	eugene10@adventure-works.c
3	3	493	11002	NULL	Ruben	NULL	Torres	0	1965-08-12	M	NULL	M	ruben35@adventure-works.co
4	4	519	11003	NULL	Christy	NULL	Zhu	0	1968-02-15	S	NULL	F	christy12@adventure-works.co
5	5	706	11004	NULL	Elizabeth	NULL	Johnson	0	1968-08-08	S	NULL	F	elizabeth5@adventure-works.c
6	6	478	11005	NULL	Julio	NULL	Ruiz	0	1965-08-05	S	NULL	M	julio1@adventure-works.com
7	7	509	11006	NULL	Janet	G	Alvarez	0	1965-12-06	S	NULL	F	janet9@adventure-works.com
8	8	568	11007	NULL	Marco	NULL	Mehta	0	1964-05-09	M	NULL	M	marco14@adventure-works.co
9	9	425	11008	NULL	Rob	NULL	Verhoff	0	1964-07-07	S	NULL	F	rob4@adventure-works.com
10	10	492	11009	NULL	Shannon	C	Carlson	0	1964-04-01	S	NULL	M	shannon38@adventure-works.
11	11	478	11010	NULL	Jacquelyn	C	Suarez	0	1964-02-06	S	NULL	F	jacquelyn20@adventure-works
12	12	478	11011	NULL	Curtis	NULL	Lu	0	1963-11-04	M	NULL	M	curtis9@adventure-works.com
13	13	442	11012	NULL	Leanne	M	Wallace	0	1968-01-10	M	NULL	F	leanne41@adventure-works.com

Consulta executada com êxito. | LAPTOP-HBTCI8PQ (15.0 RTM) | LAPTOP-HBTCI8PQ\Marcus... | ContosoRetailDW | 00:00:00 | 100 linhas

# GABARITOS

3

## Questão 3

A Contoso está comemorando aniversário de inauguração de 10 anos e pretende fazer uma ação de premiação para os clientes. A empresa quer presentear os primeiros clientes desde a inauguração.

Você foi alocado para levar adiante essa ação. Para isso, você terá que fazer o seguinte:

- A Contoso decidiu presentear os primeiros 100 clientes da história com um vale compras de R\$ 10.000. Utilize um comando em SQL para retornar uma tabela com os primeiros 100 primeiros clientes da tabela dimCustomer (selecione todas as colunas).
- A Contoso decidiu presentear os primeiros 20% de clientes da história com um vale compras de R\$ 2.000. Utilize um comando em SQL para retornar 20% das linhas da sua tabela dimCustomer (selecione todas as colunas).

The screenshot shows the SSMS interface with a query window titled "SQLQuery9.sql - LAP...us Cavalcanti (54)\*". The query is:

```
--Questão 3
--b)
SELECT TOP(20) PERCENT
*
FROM
DimCustomer
```

The results window below shows a table titled "Resultados" with 3774 rows. The columns are: CustomerKey, GeographyKey, CustomerLabel, Title, FirstName, MiddleName, LastName, NameStyle, BirthDate, MaritalStatus, Suffix, Gender, EmailAddress. The first few rows are:

	CustomerKey	GeographyKey	CustomerLabel	Title	FirstName	MiddleName	LastName	NameStyle	BirthDate	MaritalStatus	Suffix	Gender	EmailAddress
1	1	680	11000	NULL	Jon	V	Yang	0	1966-04-08	M	NULL	M	jon24@adventure-works.com
2	2	692	11001	NULL	Eugene	L	Huang	0	1965-05-14	S	NULL	M	eugene10@adventure-works.c
3	3	493	11002	NULL	Ruben	NULL	Torres	0	1965-08-12	M	NULL	M	ruben35@adventure-works.co
4	4	519	11003	NULL	Christy	NULL	Zhu	0	1968-02-15	S	NULL	F	christy12@adventure-works.cc

At the bottom of the results window, it says "Consulta executada com êxito." (Query executed successfully.)

# GABARITOS

3

## Questão 3

A Contoso está comemorando aniversário de inauguração de 10 anos e pretende fazer uma ação de premiação para os clientes. A empresa quer presentear os primeiros clientes desde a inauguração.

Você foi alocado para levar adiante essa ação. Para isso, você terá que fazer o seguinte:

- A Contoso decidiu presentear os primeiros 100 clientes da história com um vale compras de R\$ 10.000. Utilize um comando em SQL para retornar uma tabela com os primeiros 100 primeiros clientes da tabela dimCustomer (selecione todas as colunas).
- A Contoso decidiu presentear os primeiros 20% de clientes da história com um vale compras de R\$ 2.000. Utilize um comando em SQL para retornar 20% das linhas da sua tabela dimCustomer (selecione todas as colunas).
- Adapte o código do item a) para retornar apenas as 100 primeiras linhas, mas apenas as colunas FirstName, EmailAddress, BirthDate.

```
-- Questão 3
-- c)
SELECT TOP(100)
    FirstName,
    EmailAddress,
    BirthDate
FROM
    DimCustomer
```

	FirstName	EmailAddress	BirthDate
1	Jon	jon24@adventure-works.com	1966-04-08
2	Eugene	eugene10@adventure-works.com	1965-05-14
3	Ruben	ruben35@adventure-works.com	1965-08-12
4	Christy	christy12@adventure-works.com	1968-02-15
5	Elizabeth	elizabeth5@adventure-works.com	1968-08-08
6	Julio	julio1@adventure-works.com	1965-08-05
7	Janet	janet9@adventure-works.com	1965-12-06
8	Marco	marco14@adventure-works.com	1964-05-09
9	Rob	rob4@adventure-works.com	1964-07-07
10	Shannon	shannon38@adventure-works.com	1964-04-01
11	Jacquelyn	jacquelyn20@adventure-works.com	1964-02-06
12	Curtis	curtis9@adventure-works.com	1963-11-04
13	Lauren	lauren41@adventure-works.com	1968-01-18

Consulta executada com êxito. | LAPTOP-HBTCI8PQ (15.0 RTM) | LAPTOP-HBTCI8PQ\Marcus... | ContosoRetailDW | 00:00:00 | 100 linhas



# GABARITOS

3

## Questão 3

A Contoso está comemorando aniversário de inauguração de 10 anos e pretende fazer uma ação de premiação para os clientes. A empresa quer presentear os primeiros clientes desde a inauguração.

Você foi alocado para levar adiante essa ação. Para isso, você terá que fazer o seguinte:

- A Contoso decidiu presentear os primeiros 100 clientes da história com um vale compras de R\$ 10.000. Utilize um comando em SQL para retornar uma tabela com os primeiros 100 primeiros clientes da tabela dimCustomer (selecione todas as colunas).
- A Contoso decidiu presentear os primeiros 20% de clientes da história com um vale compras de R\$ 2.000. Utilize um comando em SQL para retornar 20% das linhas da sua tabela dimCustomer (selecione todas as colunas).
- Adapte o código do item a) para retornar apenas as 100 primeiras linhas, mas apenas as colunas FirstName, EmailAddress, BirthDate.
- Renomeie as colunas anteriores para nomes em português

	Nome	Email	Data Nascimento
1	Jon	jon24@adventure-works.com	1966-04-08
2	Eugene	eugene10@adventure-works.com	1965-05-14
3	Ruben	ruben35@adventure-works.com	1965-08-12
4	Christy	christy12@adventure-works.com	1968-02-15
5	Elizabeth	elizabeth5@adventure-works.com	1968-08-08
6	Julio	julio1@adventure-works.com	1965-08-05
7	Janet	janet9@adventure-works.com	1965-12-06
8	Marco	marco14@adventure-works.com	1964-05-09
9	Rob	rob4@adventure-works.com	1964-07-07
10	Shannon	shannon38@adventure-works.com	1964-04-01
11	Jacquelyn	jacquelyn20@adventure-works.com	1964-02-06
12	Curtis	curtis9@adventure-works.com	1963-11-04
13	Lauren	lauren41@adventure-works.com	1968-01-18

Consulta executada com êxito.



# GABARITOS

4

## Questão 4

A empresa Contoso precisa fazer contato com os fornecedores de produtos para repor o estoque. Você é da área de compras e precisa descobrir quem são esses fornecedores.

Utilize um comando em SQL para retornar apenas os nomes dos fornecedores na tabela dimProduct e renomeie essa nova coluna da tabela.

The screenshot shows a SQL query window titled "SQLQuery9.sql - LAP...us Cavalcanti (54)\*". The query is:

```
--Questão 4
SELECT
    DISTINCT Manufacturer AS 'Produtor'
FROM
    DimProduct
```

The results pane displays a table with 11 rows, each containing a number and a company name under the column "Produtor".

	Produtor
1	Contoso, Ltd.
2	Adventure Works
3	Southridge Video
4	Prosware, Inc.
5	Fabrikam, Inc.
6	Northwind Traders
7	Tailspin Toys
8	The Phone Company
9	Wide World Importers
10	Litware, Inc.
11	A. Datum Corporation

At the bottom of the results pane, a message says "Consulta executada com êxito." (Query executed successfully).

# GABARITOS

5

## Questão 5

O seu trabalho de investigação não para. Você precisa descobrir se existe algum produto registrado na base de produtos que ainda não tenha sido vendido. Tente chegar nessa informação.

Obs: caso tenha algum produto que ainda não tenha sido vendido, você não precisa descobrir qual é, é suficiente saber se teve ou não algum produto que ainda não foi vendido. **Existe 1 produto que está registrado na tabela DimProduct e que não foi vendido.**

The screenshot shows a SQL query window titled "SQLQuery9.sql - LAP...us Cavalcanti (54)\*". The window contains two queries:

```
--Questão 5
SELECT DISTINCT ProductKey FROM FactSales
2.516 linhas

SELECT DISTINCT ProductKey FROM DimProduct
2.517 linhas
```

The first query, "SELECT DISTINCT ProductKey FROM FactSales", returns 2.516 rows. The second query, "SELECT DISTINCT ProductKey FROM DimProduct", returns 2.517 rows.

# SQL

Ordenando e filtrando  
dados

# ORDER BY: Ordenando uma tabela

O ORDER BY é usado para ordenar os valores de uma tabela em ordem crescente ou decrescente.

Por padrão, o ORDER BY ordena os dados em ordem crescente (ASC). Para ordenar de forma decrescente, usamos o DESC.

-- Exemplo 1: Selecione as 100 primeiras linhas da tabela DimStore e ordene de acordo com a coluna de quantidade de funcionários

```
SQLQuery1.sql - LAP...us Cavalcanti (55)*  X
SELECT TOP(100)
    StoreName,
    EmployeeCount
FROM DimStore
-- ordena de forma crescente de
-- acordo com a coluna EmployeeCount
ORDER BY EmployeeCount
```

	StoreName	EmployeeCount
1	Contoso Cheshire Store	NULL
2	Contoso Europe Online Store	7
3	Contoso Asia Online Store	8
4	Contoso Strasbourg Store	10
5	Contoso Manchester Store	11
6	Contoso Nantes Store	11
7	Contoso Venezia Store	11
8	Contoso Paris Store	12
9	Contoso Europe Reseller	12
10	Contoso Marseille Store	13
11	Contoso Naples Store	13
12	Contoso Berlin Store	13

```
SQLQuery1.sql - LAP...us Cavalcanti (55)*  X
SELECT TOP(100)
    StoreName,
    EmployeeCount
FROM DimStore
-- ordena de forma decrescente
-- de acordo com a coluna EmployeeCount
ORDER BY EmployeeCount DESC
```

	StoreName	EmployeeCount
1	Contoso North America Online Store	325
2	Contoso Guangzhou Store	131
3	Contoso Catalog Store	120
4	Contoso Shanghai No.1 Store	95
5	Contoso Shanghai No.2 Store	95
6	Contoso Ottawa No.1 Store	95
7	Contoso Ottawa No.2 Store	95
8	Contoso Toronto No.1 Store	95
9	Contoso Toronto No.2 Store	95
10	Contoso Toronto No.3 Store	95
11	Contoso Montreal No.1 Store	95
12	Contoso Montreal No.2 Store	95



# ORDER BY: Ordenando uma tabela

Também é possível ordenar uma tabela por uma coluna de textos.

-- Exemplo 2: Selecione as 100 primeiras linhas da tabela DimStore e ordene de acordo com a coluna de StoreName

```
SQLQuery1.sql - LAP...us Cavalcanti (55)* ▾ X
SELECT
    TOP(100) StoreKey, StoreName
    FROM DimStore
    -- ordena de forma crescente (A-Z),
    -- de acordo com a coluna StoreName
    ORDER BY StoreName
```

	StoreKey	StoreName
1	114	Contoso Albany Store
2	184	Contoso Alexandria Store
3	240	Contoso Amsterdam Store
4	183	Contoso Anchorage Store
5	191	Contoso Annapolis Store
6	43	Contoso Appleton Store
7	93	Contoso Arlington Store
8	298	Contoso Ashgabat No.2 Store
9	264	Contoso Ashgabat No.1 Store
10	307	Contoso Asia Online Store
11	310	Contoso Asia Reseller
12	250	Contoso Athens Store

```
SQLQuery1.sql - LAP...us Cavalcanti (55)* ▾ X
SELECT
    TOP(100) StoreKey, StoreName
    FROM DimStore
    -- ordena de forma decrescente (Z-A),
    -- de acordo com a coluna StoreName
    ORDER BY StoreName DESC
```

	StoreKey	StoreName
1	204	Contoso York Store
2	269	Contoso Yokohama Store
3	254	Contoso Yerevan Store
4	6	Contoso Yakima Store
5	148	Contoso Worcester No.2 Store
6	147	Contoso Worcester No.1 Store
7	182	Contoso Winchester Store
8	21	Contoso Wheat Ridge Store
9	22	Contoso Westminster Store
10	208	Contoso West Yorkshire Store
11	54	Contoso Waukesha No.2 Store
12	50	Contoso Waukesha No.1 Store

# ORDER BY: Ordenando uma tabela

Também é possível ordenar uma tabela por mais de uma coluna.

-- Exemplo 3: Selecione as 100 primeiras linhas da tabela dimProduct e ordene de acordo com as coluna de UnitCost (DESC) e Weight (ASC)

```
SQLQuery1.sql - LAP...us Cavalcanti (55)*  X
SELECT TOP(100)
    ProductName,
    Weight,
    UnitCost
FROM DimProduct
-- ordena de forma crescente (A-Z)
ORDER BY UnitCost DESC, Weight ASC
```

	ProductName	Weight	UnitCost
1	Litware Refrigerator L1200 Orange	36	1060,22
2	Fabrikam Refrigerator 24.7CuPt X9800 Green	38,1	1060,22
3	Litware Refrigerator 24.7CuPt X980 Blue	56	1060,22
4	Fabrikam Refrigerator 24.7CuPt X9800 Brown	56,8	1060,22
5	Fabrikam Refrigerator 24.7CuPt X9800 Orange	58	1060,22
6	Litware Refrigerator 24.7CuPt X980 Silver	68	1060,22
7	Litware Refrigerator 24.7CuPt X980 Brown	70	1060,22
8	Fabrikam Refrigerator 24.7CuPt X9800 White	71	1060,22
9	Litware Refrigerator 24.7CuPt X980 Green	71	1060,22
10	Fabrikam Refrigerator 24.7CuPt X9800 Blue	75	1060,22
11	Litware Refrigerator 24.7CuPt X980 Grey	132	1060,22
12	Litware Refrigerator 24.7CuPt X980 White	175	1060,22

# WHERE ...LIKE: Um filtro especial

O **LIKE** é usado em conjunto com o **WHERE** para procurar por um determinado padrão em uma coluna.

Existem 2 caracteres especiais usados em conjunto com o **LIKE**:

O sinal de porcentagem (%) representa zero, um ou múltiplos caracteres.

O underline (\_) representa um único caractere.

Aplicação do LIKE	Descrição
WHERE CustomerName LIKE 'a%'	Encontra qualquer valor que começa com "a"
WHERE CustomerName LIKE '%a'	Encontra qualquer valor que termina com "a"
WHERE CustomerName LIKE '%or%	Encontra qualquer valor que tenha "or" em qualquer posição da palavra
WHERE CustomerName LIKE '_r%	Encontra qualquer valor que tenha "r" a partir do segundo caractere
WHERE CustomerName LIKE 'a_%'	Encontra qualquer valor que comece com "a" e tenha pelo menos 2 caracteres
WHERE CustomerName LIKE 'a__%'	Encontra qualquer valor que comece com "a" e tenha pelo menos 3 caracteres
WHERE CustomerName LIKE 'a%o'	Encontra qualquer valor que comece com "a" e termine com "o"

# WHERE ...LIKE: Um filtro especial

O **LIKE** é usado em conjunto com o **WHERE** para procurar por um determinado padrão em uma coluna.

Existem 2 caracteres especiais usados em conjunto com o **LIKE**:

- O sinal de porcentagem (%) representa zero, um ou múltiplos caracteres.
- O underline (\_) representa um único caractere.

**Exemplos:**

```
SQLQuery1.sql - LAP...us Cavalcanti (55)* ✘ X
-- Seleciona todas as linhas da tabela onde
-- o nome do produto contenha a palavra "MP3"
SELECT
*
FROM
dimProduct
WHERE ProductName LIKE '%MP3%'
```

Resultados | Mensagens

ProductKey	ProductLabel	ProductName	ProductDescription	ProductSubcategoryKey	Manufacturer	BrandName	Class
1	0101001	Contoso 512MB MP3 Player E51 Silver	512MB USB driver plays MP3 and WMA	1	Contoso, Ltd	Contoso	1
2	0101002	Contoso 512MB MP3 Player E51 Blue	512MB USB driver plays MP3 and WMA	1	Contoso, Ltd	Contoso	1
3	0101003	Contoso 1G MP3 Player E100 White	1GB flash memory and USB driver plays MP3 and WMA	1	Contoso, Ltd	Contoso	1
4	0101004	Contoso 2G MP3 Player E200 Silver	2GB flash memory, LCD display, plays MP3 and WMA	1	Contoso, Ltd	Contoso	1
5	0101005	Contoso 2G MP3 Player E200 Red	2GB flash memory, LCD display, plays MP3 and WMA	1	Contoso, Ltd	Contoso	1
6	0101006	Contoso 2G MP3 Player E200 Black	2GB flash memory, LCD display, plays MP3 and WMA	1	Contoso, Ltd	Contoso	1
7	0101007	Contoso 2G MP3 Player E200 Blue	2GB flash memory, LCD display, plays MP3 and WMA	1	Contoso, Ltd	Contoso	1
8	0101008	Contoso 4G MP3 Player E400 Silver	4GB flash memory and FM Radio, LCD Display with 7...	1	Contoso, Ltd	Contoso	1
9	0101009	Contoso 4G MP3 Player E400 Black	4GB flash memory and FM Radio, LCD Display with 7...	1	Contoso, Ltd	Contoso	1
10	0101010	Contoso 4G MP3 Player E400 Green	4GB flash memory and FM Radio, LCD Display with 7...	1	Contoso, Ltd	Contoso	1
11	0101011	Contoso 4G MP3 Player E400 Orange	4GB flash memory and FM Radio, LCD Display with 7...	1	Contoso, Ltd	Contoso	1

Consulta executada com êxito. | LAPTOP-HBTCI8PQ (15.0 RTM) | LAPTOP-HBTCI8PQ\Marcus... | ContosoRetailDW | 00:00:00 | 35 linhas

# WHERE ...AND, OR e NOT: Filtrar por mais de uma condição

O comando WHERE pode ser combinado com os operadores AND, OR e NOT.

Os operadores AND e OR são usados para filtrar linhas da tabela baseado em mais de uma condição.

- O **AND** mostra as linhas da tabela se todas as condições forem atendidas.
- O **OR** mostra as linhas da tabela se pelo menos uma das condições for atendida.
- O **NOT** simplesmente mostra o oposto do que for considerado no filtro. Ex: mostra todas as linhas que NÃO forem da marca “Contoso”.

## Exemplos:

-- Seleciona todas as linhas da tabela em que a marca é Contoso E a cor é Prata.

```
SELECT * FROM dimProduct
```

```
WHERE BrandName = 'Contoso' AND ColorName = 'Silver'
```

-- Seleciona todas as linhas da tabela em que a cor é Azul OU a cor é Prata.

```
SELECT * FROM dimProduct
```

```
WHERE ColorName = 'Blue' OR ColorName = 'Silver'
```

-- Seleciona todas as linhas da tabela em que a cor NÃO é azul.

```
SELECT * FROM dimProduct
```

```
WHERE NOT ColorName = 'Blue'
```

# WHERE ...IN: Especificando mais de 1 critério por vez

O operador IN permite que sejam especificados múltiplos critérios dentro do WHERE.

O IN é uma alternativa reduzida ao OR.

## Exemplos:

```
SQLQuery1.sql - LAP...us Cavalcanti (55)* □ X
-- Seleciona todas as linhas da tabela
-- com produtos das cores Azul, Prata e Preta.

SELECT * FROM dimProduct

WHERE ColorName IN ('Blue', 'Silver', 'Black')
```

	ProductKey	ProductLabel	ProductName	ProductDescription	ProductSubcategoryKey	Manufacturer	BrandName
1	2	0101002	Contoso 512MB MP3 Player E51 Blue	512MB USB driver plays MP3 and WMA	1	Contoso, Ltd	Contoso
2	6	0101006	Contoso 2G MP3 Player E200 Black	2GB flash memory, LCD display, plays MP3 and WMA	1	Contoso, Ltd	Contoso
3	7	0101007	Contoso 2G MP3 Player E200 Blue	2GB flash memory, LCD display, plays MP3 and WMA	1	Contoso, Ltd	Contoso
4	9	0101009	Contoso 4G MP3 Player E400 Black	4GB flash memory and FM Radio, LCD Display with ...	1	Contoso, Ltd	Contoso
5	12	0101012	Contoso 4GB Flash MP3 Player E401 Blue	1.8" color LCD, play MP3, WMA and Video MTV, a...	1	Contoso, Ltd	Contoso
6	13	0101013	Contoso 4GB Flash MP3 Player E401 Black	1.8" color LCD, play MP3, WMA and Video MTV, a...	1	Contoso, Ltd	Contoso
7	20	0101020	Contoso 8GB MP3 Player new model M820 Black	2" LCD with blue-white LED, 320x240-pixel, plays m...	1	Contoso, Ltd	Contoso
8	21	0101021	Contoso 8GB MP3 Player new model M820 Blue	2" LCD with blue-white LED, 320x240-pixel, plays m...	1	Contoso, Ltd	Contoso
9	24	0101024	Contoso 16GB Mp5 Player M1600 Blue	3" 16.9 TFT Touch screen, 16GB flash memory, pla...	1	Contoso, Ltd	Contoso
10	25	0101025	Contoso 16GB Mp5 Player M1600 Black	3" 16.9 TFT Touch screen, 16GB flash memory, pla...	1	Contoso, Ltd	Contoso
11	33	0101033	Contoso 32GB Video MP3 Player M3200 Black	4.3" Touch screen, 32GB flash memory, beyond 30 ...	1	Contoso, Ltd	Contoso

Observe que a quantidade de linhas é bem menor, pois estamos considerando apenas os produtos de 3 marcas específicas.

## Exemplos:

```
SQLQuery1.sql - LAP...us Cavalcanti (55)* □ X
-- Seleciona todas as linhas que NÃO correspondem
-- às cores Azul, Prata e Preta.

SELECT * FROM dimProduct

WHERE ColorName NOT IN ('Blue', 'Silver', 'Black')
```

	ProductKey	ProductLabel	ProductName	ProductDescription	ProductSubcategoryKey	Manufacturer	BrandName
1	1	0101001	Contoso 512MB MP3 Player E51 Silver	512MB USB driver plays MP3 and WMA	1	Contoso, Ltd	Contoso
2	3	0101003	Contoso 1G MP3 Player E100 White	1GB flash memory and USB driver plays MP3 and WMA	1	Contoso, Ltd	Contoso
3	4	0101004	Contoso 2G MP3 Player E200 Silver	2GB flash memory, LCD display, plays MP3 and WMA	1	Contoso, Ltd	Contoso
4	5	0101005	Contoso 2G MP3 Player E200 Red	2GB flash memory, LCD display, plays MP3 and WMA	1	Contoso, Ltd	Contoso
5	8	0101008	Contoso 4G MP3 Player E400 Silver	4GB flash memory and FM Radio, LCD Display with 7...	1	Contoso, Ltd	Contoso
6	10	0101010	Contoso 4G MP3 Player E400 Green	4GB flash memory and FM Radio, LCD Display with 7...	1	Contoso, Ltd	Contoso
7	11	0101011	Contoso 4G MP3 Player E400 Orange	4GB flash memory and FM Radio, LCD Display with 7...	1	Contoso, Ltd	Contoso
8	14	0101014	Contoso 4GB Flash MP3 Player E401 Silver	1.8" color LCD, play MP3, WMA and Video MTV, and...	1	Contoso, Ltd	Contoso
9	15	0101015	Contoso 4GB Flash MP3 Player E401 White	1.8" color LCD, play MP3, WMA and Video MTV, and...	1	Contoso, Ltd	Contoso
10	16	0101016	Contoso 8GB Super-Slim MP3/Video Player M800 White	2" color LCD, Touchpad, Plays music, video, photos ...	1	Contoso, Ltd	Contoso
11	17	0101017	Contoso 8GB Super-Slim MP3/Video Player M800 Red	2" color LCD, Touchpad, Plays music, video, photos ...	1	Contoso, Ltd	Contoso



# WHERE ... (NOT) BETWEEN: Como filtrar entre valores

- O **BETWEEN** seleciona valores em um intervalo. Estes valores podem ser números, textos ou datas.
- O **BETWEEN** é inclusivo: ou seja, os valores dos extremos (valor inicial e valor final) também são incluídos no intervalo.

Exemplos:

```
SQLQuery1.sql - LAP...us Cavalcanti (55)* ✎ X
-- Seleciona todas as vendas da tabela
-- com quantidade vendida entre 10 e 15.

SELECT TOP(1000) * FROM factSales
WHERE SalesQuantity BETWEEN 10 AND 15
```

Resultados Mensagens

SalesKey	DateKey	channelKey	StoreKey	ProductKey	PromotionKey	CurrencyKey	UnitCost	UnitPrice	SalesQuantity	RetumQuantity	RetumAmount	Discou
1	12 2007-04-29 00:00:00.000	1	119	543	1	1	116,75	229,00	10	0	0,00	0
2	13 2007-07-25 00:00:00.000	1	171	739	3	1	78,19	236,00	12	0	0,00	0
3	14 2008-12-16 00:00:00.000	1	16	1269	13	1	25,47	49,96	13	0	0,00	1
4	15 2008-10-17 00:00:00.000	2	199	1788	1	1	21,92	43,00	10	0	0,00	0
5	19 2008-07-16 00:00:00.000	1	292	519	1	1	205,09	619,00	12	0	0,00	0
6	25 2007-06-06 00:00:00.000	1	108	2351	1	1	183,94	399,99	10	0	0,00	0
7	27 2009-04-29 00:00:00.000	1	186	2483	1	1	160,95	350,00	10	0	0,00	0
8	28 2009-12-23 00:00:00.000	1	4	1368	22	1	18,48	40,19	13	0	0,00	0
9	30 2008-12-20 00:00:00.000	1	302	363	14	1	321,44	699,00	13	0	0,00	1
10	34 2007-05-24 00:00:00.000	1	214	1494	1	1	95,65	208,00	10	0	0,00	0
11	35 2007-10-04 00:00:00.000	1	55	1992	1	1	71,37	139,99	10	1	139,99	0

Consulta executada com êxito. | LAPTOP-HBTCI8PQ (15.0 RTM) | LAPTOP-HBTCI8PQ\Marcus... | ContosoRetailDW | 00:00:00 | 1.000 linhas

# WHERE ...IS (NOT) NULL: Filtrando valores vazios

Podemos encontrar valores nulos (em branco) em uma tabela. Para filtrar esses valores, podemos usar os comandos **IS NULL** ou **IS NOT NULL**.

Exemplos:

```
SQLQuery1.sql - LAP...us Cavalcanti (55)* ↗ X
-- Seleciona todas as linhas
-- onde o primeiro nome é nulo
SELECT * FROM dimCustomer
WHERE firstName IS NULL
```

	CustomerKey	GeographyKey	CustomerLabel	Title	FirstName	MiddleName	LastName	NameStyle	BirthDate	MaritalStatus	Suffix	Gender	EmailAddress	YearlyIncome
1	18761	424	CS424	NULL	NULL	NULL	NULL	1	NULL	NULL	NULL	NULL	10000000,00	
2	18762	430	CS430	NULL	NULL	NULL	NULL	1	NULL	NULL	NULL	NULL	10000000,00	
3	18763	431	CS431	NULL	NULL	NULL	NULL	1	NULL	NULL	NULL	NULL	10000000,00	
4	18764	432	CS432	NULL	NULL	NULL	NULL	1	NULL	NULL	NULL	NULL	10000000,00	
5	18765	433	CS433	NULL	NULL	NULL	NULL	1	NULL	NULL	NULL	NULL	10000000,00	
6	18766	434	CS434	NULL	NULL	NULL	NULL	1	NULL	NULL	NULL	NULL	10000000,00	
7	18767	436	CS436	NULL	NULL	NULL	NULL	1	NULL	NULL	NULL	NULL	10000000,00	
8	18768	438	CS438	NULL	NULL	NULL	NULL	1	NULL	NULL	NULL	NULL	10000000,00	
9	18769	440	CS440	NULL	NULL	NULL	NULL	1	NULL	NULL	NULL	NULL	10000000,00	
10	18770	441	CS441	NULL	NULL	NULL	NULL	1	NULL	NULL	NULL	NULL	10000000,00	
11	18771	442	CS442	NULL	NULL	NULL	NULL	1	NULL	NULL	NULL	NULL	10000000,00	

Exemplos:

```
SQLQuery1.sql - LAP...us Cavalcanti (55)* ↗ X
-- Seleciona todas as linhas
-- onde o primeiro nome NÃO é nulo
SELECT * FROM dimCustomer
WHERE firstName IS NOT NULL
```

	CustomerKey	GeographyKey	CustomerLabel	Title	FirstName	MiddleName	LastName	NameStyle	BirthDate	MaritalStatus	Suffix	Gender	EmailAddress	YearlyIncome
1	1	680	11000	NULL	Jon	V	Yang	0	1966-04-08	M	NULL	M	jon24@adventure-works.com	
2	2	692	11001	NULL	Eugene	L	Huang	0	1965-05-14	S	NULL	M	eugene10@adventure-works.co	
3	3	493	11002	NULL	Ruben	NULL	Torres	0	1965-08-12	M	NULL	M	ruben35@adventure-works.co	
4	4	519	11003	NULL	Christy	NULL	Zhu	0	1968-02-15	S	NULL	F	christy12@adventure-works.co	
5	5	706	11004	NULL	Elizabeth	NULL	Johnson	0	1968-08-08	S	NULL	F	elizabeth5@adventure-works.co	
6	6	478	11005	NULL	Julio	NULL	Ruiz	0	1965-08-05	S	NULL	M	julio1@adventure-works.com	
7	7	509	11006	NULL	Janet	G	Alvarez	0	1965-12-06	S	NULL	F	janet9@adventure-works.com	
8	8	568	11007	NULL	Marco	NULL	Mehta	0	1964-05-09	M	NULL	M	marco14@adventure-works.co	
9	9	425	11008	NULL	Rob	NULL	Verhoff	0	1964-07-07	S	NULL	F	rob4@adventure-works.com	
10	10	492	11009	NULL	Shannon	C	Carlson	0	1964-04-01	S	NULL	M	shannon38@adventure-works.co	
11	11	478	11010	NULL	Jacquelyn	C	Suarez	0	1964-02-06	S	NULL	F	jacquelyn20@adventure-works.co	



# EXERCÍCIOS



1

## Questão 1

Você é o gerente da área de compras e precisa criar um relatório com as TOP 100 vendas, de acordo com a quantidade vendida. Você precisa fazer isso em 10min pois o diretor de compras solicitou essa informação para apresentar em uma reunião.

Utilize seu conhecimento em SQL para buscar essas TOP 100 vendas, de acordo com o total vendido (SalesAmount).

2

## Questão 2

Os TOP 10 produtos com maior UnitPrice possuem exatamente o mesmo preço. Porém, a empresa quer diferenciar esses preços de acordo com o peso (Weight) de cada um.

O que você precisará fazer é ordenar esses top 10 produtos, de acordo com a coluna de UnitPrice e, além disso, estabelecer um critério de desempate, para que seja mostrado na ordem, do maior para o menor.

Caso ainda assim haja um empate entre 2 ou mais produtos, pense em uma forma de criar um segundo critério de desempate (além do peso).

# EXERCÍCIOS



3

## Questão 3

Você é responsável pelo setor de logística da empresa Contoso e precisa dimensionar o transporte de todos os produtos em categorias, de acordo com o peso.

Os produtos da categoria A, com peso acima de 100kg, deverão ser transportados na primeira leva.

Faça uma consulta no banco de dados para descobrir quais são estes produtos que estão na categoria A.

- a. Você deverá retornar apenas 2 colunas nessa consulta: Nome do Produto e Peso.
- b. Renomeie essas colunas com nomes mais intuitivos.
- c. Ordene esses produtos do mais pesado para o mais leve.

# EXERCÍCIOS



4

## Questão 4

Você foi alocado para criar um relatório das lojas registradas atualmente na Contoso.

- a. Descubra quantas lojas a empresa tem no total. Na consulta que você deverá fazer à tabela DimStore, retorne as seguintes informações: StoreName, OpenDate, EmployeeCount.
- b. Renomeie as colunas anteriores para deixar a sua consulta mais intuitiva.
- c. Desses lojas, descubra quantas (e quais) lojas ainda estão ativas.

5

## Questão 5

O gerente da área de controle de qualidade notificou à Contoso que todos os produtos Home Theater da marca Litware, disponibilizados para venda no dia 15 de março de 2009, foram identificados com defeitos de fábrica.

O que você deverá fazer é identificar os ID's desses produtos e repassar ao gerente para que ele possa notificar as lojas e consequentemente solicitar a suspensão das vendas desses produtos.

# EXERCÍCIOS



6

## Questão 6

Imagine que você precise extrair um relatório da tabela DimStore, com informações de lojas. Mas você precisa apenas das lojas que não estão mais funcionando atualmente.

- Utilize a coluna de Status para filtrar a tabela e trazer apenas as lojas que não estão mais funcionando.
- Agora imagine que essa coluna de Status não existe na sua tabela. Qual seria a outra forma que você teria de descobrir quais são as lojas que não estão mais funcionando.

7

## Questão 7

De acordo com a quantidade de funcionários, cada loja receberá uma determinada quantidade de máquinas de café. As lojas serão divididas em 3 categorias:

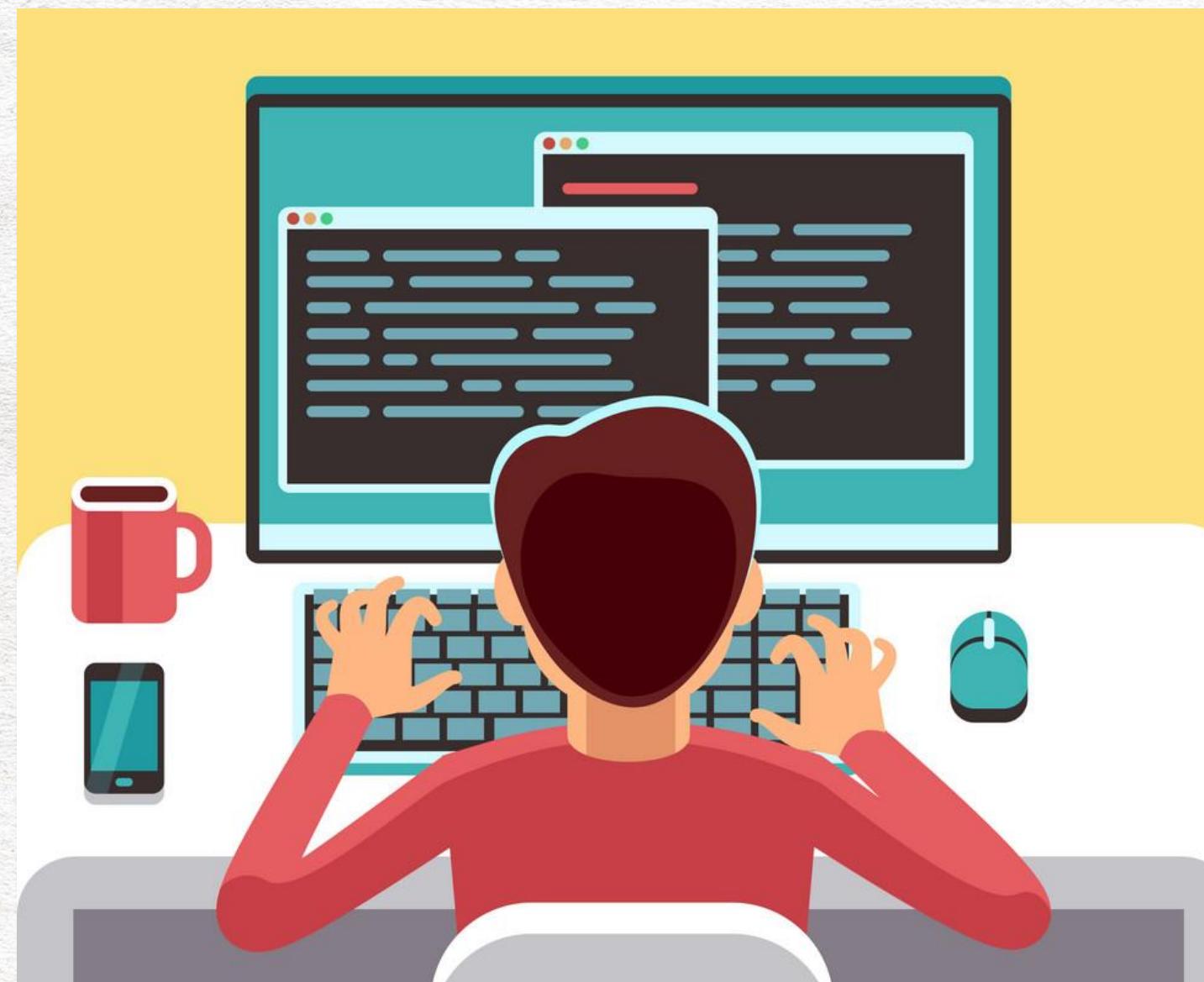
CATEGORIA 1: De 1 a 20 funcionários -> 1 máquina de café

CATEGORIA 2: De 21 a 50 funcionários -> 2 máquinas de café

CATEGORIA 3: Acima de 51 funcionários -> 3 máquinas de café

Identifique, para cada caso, quais são as lojas de cada uma das 3 categorias acima (basta fazer uma verificação).

# EXERCÍCIOS



8

## Questão 8

A empresa decidiu que todos os produtos LCD receberão um super desconto no próximo mês. O seu trabalho é fazer uma consulta à tabela DimProduct e retornar os ID's, Nomes e Preços de todos os produtos LCD existentes.

9

## Questão 9

Faça uma lista com todos os produtos das cores: Green, Orange, Black, Silver e Pink. Estes produtos devem ser exclusivamente das marcas: Contoso, Litware e Fabrikam.

10

## Questão 10

A empresa possui 16 produtos da marca Contoso, da cor Silver e com um UnitPrice entre 10 e 30. Descubra quais são esses produtos e ordene o resultado em ordem decrescente de acordo com o preço (UnitPrice).

# GABARITOS

1

## Questão 1

Você é o gerente da área de compras e precisa criar um relatório com as TOP 100 vendas, de acordo com a quantidade vendida. Você precisa fazer isso em 10min pois o diretor de compras solicitou essa informação para apresentar em uma reunião.

Utilize seu conhecimento em SQL para buscar essas TOP 100 vendas, de acordo com o total vendido (SalesAmount).

The screenshot shows a SQL query in the SQL Query window titled 'SQLQuery9.sql - LAP...us Cavalcanti (54)\*'. The query is:

```
--Questão 1
SELECT
    TOP(100) *
FROM
    FactSales
ORDER BY SalesAmount DESC
```

The results window below shows the top 8 rows of data from the FactSales table:

	SalesKey	DateKey	channelKey	StoreKey	ProductKey	PromotionKey	CurrencyKey	UnitCost	UnitPrice	SalesQuantity	ReturnQuantity	ReturnAmount	Discount
1	145071	2007-11-19 00:00:00.000	3	200	1852	4	1	878,96	2652,90	156	0	0,00	11
2	1873989	2007-11-22 00:00:00.000	3	200	1951	4	1	1060,22	3199,99	104	0	0,00	5
3	729520	2007-11-03 00:00:00.000	3	200	1862	4	1	878,96	2652,90	104	0	0,00	6
4	1453383	2007-11-16 00:00:00.000	3	200	1951	4	1	1060,22	3199,99	78	0	0,00	7
5	2241482	2007-11-26 00:00:00.000	3	200	1969	4	1	1060,22	3199,99	78	1	3199,99	7
6	541145	2007-11-16 00:00:00.000	3	200	587	4	1	760,38	2295,00	104	0	0,00	7
7	1366947	2007-11-12 00:00:00.000	3	200	2085	4	1	488,70	1475,00	156	0	0,00	9
8	383636	2007-11-29 00:00:00.000	3	200	1852	4	1	878,96	2652,90	78	0	0,00	2

Consulta executada com êxito.

# GABARITOS

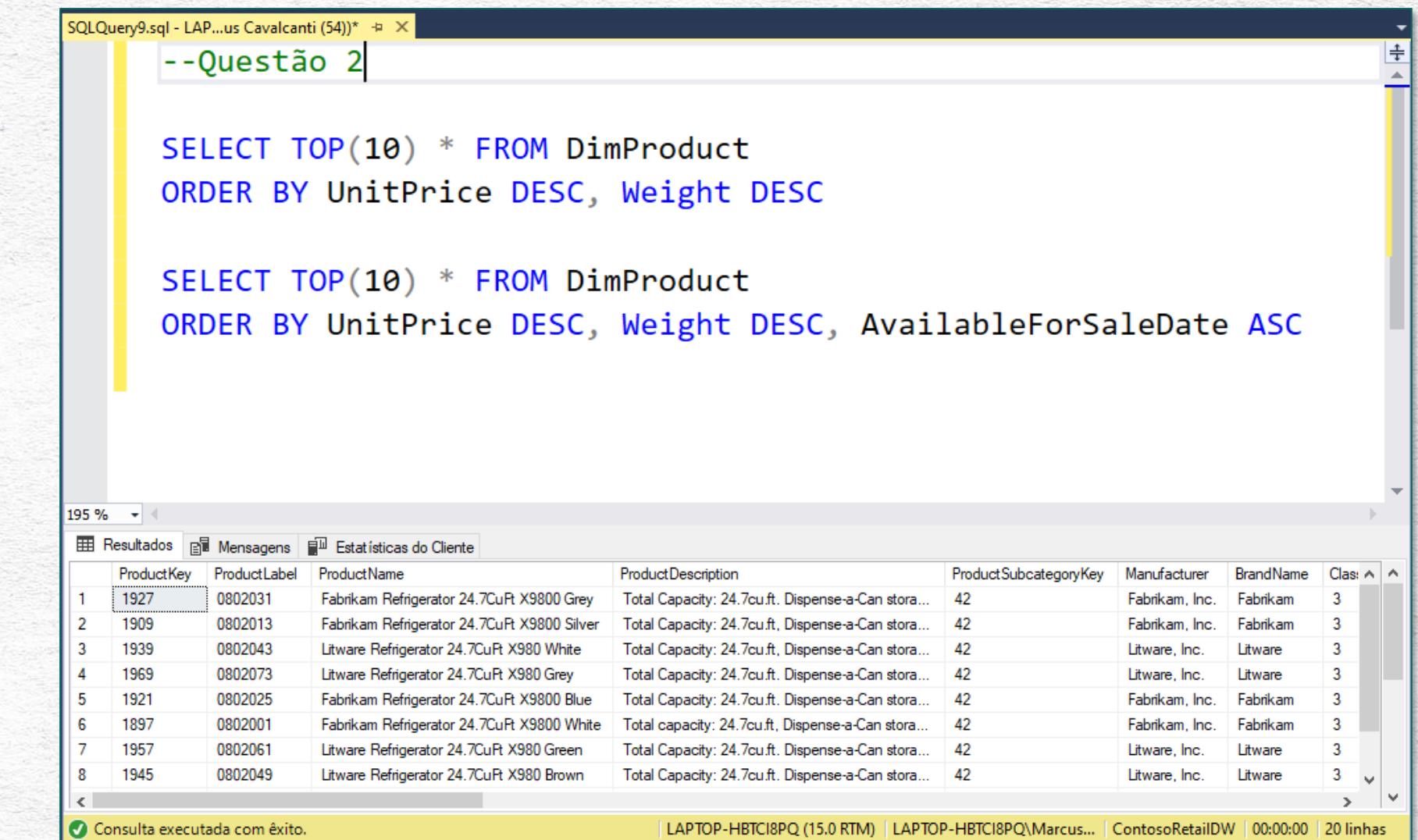
2

## Questão 2

Os TOP 10 produtos com maior UnitPrice possuem exatamente o mesmo preço. Porém, a empresa quer diferenciar esses preços de acordo com o peso (Weight) de cada um.

O que você precisará fazer é ordenar esses top 10 produtos, de acordo com a coluna de UnitPrice e, além disso, estabelecer um critério de desempate, para que seja mostrado na ordem, do maior para o menor.

Caso ainda assim haja um empate entre 2 ou mais produtos, pense em uma forma de criar um segundo critério de desempate (além do peso).



```
--Questão 2

SELECT TOP(10) * FROM DimProduct
ORDER BY UnitPrice DESC, Weight DESC

SELECT TOP(10) * FROM DimProduct
ORDER BY UnitPrice DESC, Weight DESC, AvailableForSaleDate ASC
```

ProductKey	ProductLabel	ProductName	ProductDescription	ProductSubcategoryKey	Manufacturer	BrandName	Class
1	1927	0802031	Fabrikam Refrigerator 24.7CuFt X9800 Grey	Total Capacity: 24.7cu.ft. Dispense-a-Can stor...	42	Fabrikam, Inc.	3
2	1909	0802013	Fabrikam Refrigerator 24.7CuFt X9800 Silver	Total Capacity: 24.7cu.ft. Dispense-a-Can stor...	42	Fabrikam, Inc.	3
3	1939	0802043	Litware Refrigerator 24.7CuFt X980 White	Total Capacity: 24.7cu.ft. Dispense-a-Can stor...	42	Litware, Inc.	3
4	1969	0802073	Litware Refrigerator 24.7CuFt X980 Grey	Total Capacity: 24.7cu.ft. Dispense-a-Can stor...	42	Litware, Inc.	3
5	1921	0802025	Fabrikam Refrigerator 24.7CuFt X9800 Blue	Total Capacity: 24.7cu.ft. Dispense-a-Can stor...	42	Fabrikam, Inc.	3
6	1897	0802001	Fabrikam Refrigerator 24.7CuFt X9800 White	Total capacity: 24.7cu.ft. Dispense-a-Can stor...	42	Fabrikam, Inc.	3
7	1957	0802061	Litware Refrigerator 24.7CuFt X980 Green	Total Capacity: 24.7cu.ft. Dispense-a-Can stor...	42	Litware, Inc.	3
8	1945	0802049	Litware Refrigerator 24.7CuFt X980 Brown	Total Capacity: 24.7cu.ft. Dispense-a-Can stor...	42	Litware, Inc.	3

Consulta executada com êxito.



# GABARITOS

3

## Questão 3

Você é responsável pelo setor de logística da empresa Contoso e precisa dimensionar o transporte de todos os produtos em categorias, de acordo com o peso.

Os produtos da categoria A, com peso acima de 100kg, deverão ser transportados na primeira leva.

Faça uma consulta no banco de dados para descobrir quais são estes produtos que estão na categoria A.

- Você deverá retornar apenas 2 colunas nessa consulta: Nome do Produto e Peso.
- Renomeie essas colunas com nomes mais intuitivos.
- Ordene esses produtos do mais pesado para o mais leve.

```
--Questão 3
SELECT
    ProductName AS 'Nome do Produto',
    Weight AS 'Peso (Libras)'
FROM
    DimProduct
WHERE Weight > 100
ORDER BY Weight DESC
```

	Nome do Produto	Peso (Libras)
1	Litware Washer & Dryer 27in L420 Silver	239
2	NT Washer & Dryer 24in M2400 White	239
3	NT Washer & Dryer 24in M2400 Green	239
4	Fabrikam Refrigerator 9.7CuRt M5600 White	228
5	Fabrikam Refrigerator 1.7CuRt E1200 Silver	228
6	Litware Refrigerator 3.2CuRt E160 Green	228
7	Litware Refrigerator 4.6CuRt E280 White	227
8	Fabrikam Refrigerator 24.7CuRt X9800 Grey	220
9	Litware Refrigerator 1.7CuRt E120 Grey	220

Consulta executada com êxito.

# GABARITOS

4

## Questão 4

Você foi alocado para criar um relatório das lojas registradas atualmente na Contoso.

- Descubra quantas lojas a empresa tem no total. Na consulta que você deverá fazer à tabela DimStore, retorne as seguintes informações: StoreName, OpenDate, EmployeeCount.
- Renomeie as colunas anteriores para deixar a sua consulta mais intuitiva.
- Dessas lojas, descubra quantas (e quais) lojas ainda estão ativas.

```

SQLQuery9.sql - LAP...us Cavalcanti (54)* - X
--Questão 4

SELECT
    StoreName AS 'Nome da Loja',
    OpenDate AS 'Data de abertura',
    EmployeeCount AS 'Qtd. Funcionários'
FROM
    DimStore
WHERE StoreType = 'Store' AND Status = 'On'
  
```

	Nome da Loja	Data de abertura	Qtd. Funcionários
1	Contoso Seattle No.1 Store	2004-04-12 00:00:00.000	17
2	Contoso Seattle No.2 Store	2004-02-14 00:00:00.000	25
3	Contoso Kennewick Store	2004-02-12 00:00:00.000	26
4	Contoso Bellevue Store	2004-03-01 00:00:00.000	19
5	Contoso Redmond Store	2004-04-02 00:00:00.000	33
6	Contoso Yakima Store	2005-05-15 00:00:00.000	47
7	Contoso Granger Store	2004-06-16 00:00:00.000	22
8	Contoso Sunnyside Store	2004-07-02 00:00:00.000	17
9	Contoso Toppenish Store	2004-08-18 00:00:00.000	25

Consulta executada com êxito.



# GABARITOS

5

## Questão 5

O gerente da área de controle de qualidade notificou à Contoso que todos os produtos Home Theater da marca Litware, disponibilizados para venda no dia 15 de março de 2009, foram identificados com defeitos de fábrica.

O que você deverá fazer é identificar os ID's desses produtos e repassar ao gerente para que ele possa notificar as lojas e consequentemente solicitar a suspensão das vendas desses produtos.

The screenshot shows a SQL query window titled "SQLQuery9.sql - LAP...us Cavalcanti (54)\*". The query is:

```
--Questão 5
SELECT *
FROM DimProduct
WHERE BrandName = 'Litware' AND AvailableForSaleDate = '15/03/2009'
```

The results pane displays a table with three rows of data:

	ProductKey	ProductLabel	ProductName	ProductDescription	ProductSubcategoryKey	Manufacturer	BrandName
1	198	0203007	Litware Home Theater System 4.1 Channel M411 Black	4.1 Channel Front Surround. 1080p/720p up conver...	11	Litware, Inc.	Litware
2	214	0203023	Litware Home Theater System 4.1 Channel M411 Silver	4.1 channel front surround, 1080p/720p up convers...	11	Litware, Inc.	Litware
3	230	0203039	Litware Home Theater System 4.1 Channel M411 Brown	4.1 Channel front surround. 1080p/720p up conver...	11	Litware, Inc.	Litware

At the bottom of the results pane, a message says "Consulta executada com êxito." (Query executed successfully).

# GABARITOS

6

## Questão 6

Imagine que você precise extrair um relatório da tabela DimStore, com informações de lojas. Mas você precisa apenas das lojas que não estão mais funcionando atualmente.

- Utilize a coluna de Status para filtrar a tabela e trazer apenas as lojas que não estão mais funcionando.
- Agora imagine que essa coluna de Status não existe na sua tabela. Qual seria a outra forma que você teria de descobrir quais são as lojas que não estão mais funcionando.

--Questão 6

```
SELECT * FROM DimStore
--WHERE Status = 'Off'
WHERE CloseDate IS NOT NULL
```

StoreKey	GeographyKey	StoreManager	StoreType	StoreName	StoreDescription	Status	OpenDate	CloseDate	Ent
1	12	893	45	Contoso North Bend Store	Contoso North Bend Store	Off	2004-11-21 00:00:00.000	2009-04-12 00:00:00.000	640
2	19	812	51	Contoso Cheney Store	Contoso Cheney Store	Off	2004-08-28 00:00:00.000	2009-05-12 00:00:00.000	651
3	29	927	61	Contoso Sedalia Store	Contoso Sedalia Store	Off	2004-07-20 00:00:00.000	2009-07-12 00:00:00.000	661
4	48	878	75	Contoso Milwaukee No.2 Store	Contoso Milwaukee No.2 Store	Off	2004-02-09 00:00:00.000	2009-05-12 00:00:00.000	681
5	62	934	84	Contoso Stoughton Store	Contoso Stoughton Store	Off	2004-04-30 00:00:00.000	2009-07-12 00:00:00.000	691
6	84	851	102	Contoso Humble Store	Contoso Humble Store	Off	2004-06-22 00:00:00.000	2009-02-12 00:00:00.000	711
7	112	857	128	Contoso Key West Store	Contoso Key West Store	Off	2004-01-12 00:00:00.000	2009-07-02 00:00:00.000	741
8	119	805	134	Contoso Buffalo Store	Contoso Buffalo Store	Off	2004-07-02 00:00:00.000	2008-09-22 00:00:00.000	751

Consulta executada com êxito.

LAPTOP-HBTCI8PQ (15.0 RTM) | LAPTOP-HBTCI8PQ\Marcus... | ContosoRetailDW | 00:00:02 | 12 linhas



# GABARITOS

7

## Questão 7

De acordo com a quantidade de funcionários, cada loja receberá uma determinada quantidade de máquinas de café. As lojas serão divididas em 3 categorias:

CATEGORIA 1: De 1 a 20 funcionários -> 1 máquina de café

CATEGORIA 2: De 21 a 50 funcionários -> 2 máquinas de café

CATEGORIA 3: Acima de 51 funcionários -> 3 máquinas de café

Identifique, para cada caso, quais são as lojas de cada uma das 3 categorias acima (basta fazer uma verificação).

--Questão 7

```

SELECT * FROM DimStore
--WHERE EmployeeCount BETWEEN 1 AND 20
--WHERE EmployeeCount BETWEEN 21 AND 50
WHERE EmployeeCount > 50

```

StoreKey	GeographyKey	StoreManager	StoreType	StoreName	StoreDescription	Status	OpenDate	CloseDate	EntityKey	Zi
1	199	800	212	Online	Contoso North America Online Store	On	2004-08-25 00:00:00.000	NULL	59	2
2	200	894	213	Catalog	Contoso Catalog Store	On	2004-10-26 00:00:00.000	NULL	58	2
3	204	576	222	Store	Contoso York Store	On	2004-08-27 00:00:00.000	NULL	836	N
4	208	781	226	Store	Contoso West Yorkshire Store	On	2004-03-21 00:00:00.000	NULL	840	N
5	212	651	230	Store	Contoso Liverpool Store	On	2004-03-02 00:00:00.000	NULL	844	N
6	246	740	263	Store	Contoso Beme Store	On	2004-04-20 00:00:00.000	NULL	884	N
7	249	746	266	Store	Contoso Dublin Store	On	2006-02-20 00:00:00.000	NULL	887	N
8	262	709	276	Store	Contoso Bangkok No.1 Store	On	2008-11-06 00:00:00.000	NULL	899	1

Consulta executada com êxito.

LAPTOP-HBTCI8PQ (15.0 RTM) | LAPTOP-HBTCI8PQ\Marcus... | ContosoRetailDW | 00:00:00 | 43 linhas

# GABARITOS

8

## Questão 8

A empresa decidiu que todos os produtos LCD receberão um super desconto no próximo mês. O seu trabalho é fazer uma consulta à tabela DimProduct e retornar os ID's, Nomes e Preços de todos os produtos LCD existentes.

The screenshot shows a SQL query window titled "SQLQuery9.sql - LAP...us Cavalcanti (54)\*". The query is:

```
--Questão 8
SELECT * FROM DimProduct
WHERE ProductDescription LIKE '%LCD%'
```

The results grid displays 191 rows of data from the DimProduct table, filtered by products containing the string "LCD" in their description. The columns shown are ProductKey, ProductLabel, ProductName, ProductDescription, ProductSubcategoryKey, Manufacturer, BrandName, ClassID, and C. The data includes various Contoso products like "Contoso 2G MP3 Player E200 Silver" and "Contoso 4GB Flash MP3 Player E400 Silver".

	ProductKey	ProductLabel	ProductName	ProductDescription	ProductSubcategoryKey	Manufacturer	BrandName	ClassID	C
1	4	0101004	Contoso 2G MP3 Player E200 Silver	2GB flash memory, LCD display, plays MP3 and WMA	1	Contoso, Ltd	Contoso	1	E
2	5	0101005	Contoso 2G MP3 Player E200 Red	2GB flash memory, LCD display, plays MP3 and WMA	1	Contoso, Ltd	Contoso	1	E
3	6	0101006	Contoso 2G MP3 Player E200 Black	2GB flash memory, LCD display, plays MP3 and WMA	1	Contoso, Ltd	Contoso	1	E
4	7	0101007	Contoso 2G MP3 Player E200 Blue	2GB flash memory, LCD display, plays MP3 and WMA	1	Contoso, Ltd	Contoso	1	E
5	8	0101008	Contoso 4G MP3 Player E400 Silver	4GB flash memory and FM Radio, LCD Display with ...	1	Contoso, Ltd	Contoso	1	E
6	9	0101009	Contoso 4G MP3 Player E400 Black	4GB flash memory and FM Radio, LCD Display with ...	1	Contoso, Ltd	Contoso	1	E
7	10	0101010	Contoso 4G MP3 Player E400 Green	4GB flash memory and FM Radio, LCD Display with ...	1	Contoso, Ltd	Contoso	1	E
8	11	0101011	Contoso 4G MP3 Player E400 Orange	4GB flash memory and FM Radio, LCD Display with ...	1	Contoso, Ltd	Contoso	1	E
9	12	0101012	Contoso 4GB Flash MP3 Player E40...	1.8" color LCD, play MP3, WMA and Video MTV, a...	1	Contoso, Ltd	Contoso	1	E
10	13	0101013	Contoso 4GB Flash MP3 Player E40...	1.8" color LCD, play MP3, WMA and Video MTV, a...	1	Contoso, Ltd	Contoso	1	E
11	14	0101014	Contoso 4GB Flash MP3 Player E40...	1.8" color LCD, play MP3, WMA and Video MTV, a...	1	Contoso, Ltd	Contoso	1	E
12	15	0101015	Contoso 4GB Flash MP3 Player E40...	1.8" color LCD, play MP3, WMA and Video MTV, a...	1	Contoso, Ltd	Contoso	1	E
13	16	0101016	Contoso 8GB Super-Slim MP3/Vide...	2" color LCD, Touchpad, Plays music, video, photo...	1	Contoso, Ltd	Contoso	2	F
14	17	0101017	Contoso 8GB Super-Slim MP3/Vide...	2" color LCD, Touchpad, Plays music, video, photo...	1	Contoso, Ltd	Contoso	2	F
15	18	0101018	Contoso 8GB Super-Slim MP3/Vide...	2" color LCD, Touchpad, Plays music, video, photo...	1	Contoso, Ltd	Contoso	2	F
16	19	0101019	Contoso 8GB Super-Slim MP3/Vide...	2" color LCD, Touchpad, Plays music, video, photo...	1	Contoso, Ltd	Contoso	2	F

Consulta executada com êxito. | LAPTOP-HBTCI8PQ (15.0 RTM) | LAPTOP-HBTCI8PQ\Marcus... | ContosoRetailDW | 00:00:00 | 191 linhas

# GABARITOS

9

## Questão 9

Faça uma lista com todos os produtos das cores: Green, Orange, Black, Silver e Pink. Estes produtos devem ser exclusivamente das marcas: Contoso, Litware e Fabrikam.

--Questão 9

```
SELECT * FROM DimProduct
WHERE ColorName IN ('Green', 'Orange', 'Black', 'Silver', 'Pink') AND BrandName IN ('Contoso', 'Litware', 'Fabrikam')
```

ProductKey	ProductLabel	ProductName	ProductDescription	ProductSubcategoryKey	Manufacturer	BrandName
1	0101001	Contoso 512MB MP3 Player E51 Silver	512MB USB driver plays MP3 and WMA	1	Contoso, Ltd	Contoso
2	0101004	Contoso 2G MP3 Player E200 Silver	2GB flash memory, LCD display, plays MP3 and WMA	1	Contoso, Ltd	Contoso
3	0101006	Contoso 2G MP3 Player E200 Black	2GB flash memory, LCD display, plays MP3 and WMA	1	Contoso, Ltd	Contoso
4	0101008	Contoso 4G MP3 Player E400 Silver	4GB flash memory and FM Radio, LCD Display with ...	1	Contoso, Ltd	Contoso
5	0101009	Contoso 4G MP3 Player E400 Black	4GB flash memory and FM Radio, LCD Display with ...	1	Contoso, Ltd	Contoso
6	0101010	Contoso 4G MP3 Player E400 Green	4GB flash memory and FM Radio, LCD Display with ...	1	Contoso, Ltd	Contoso
7	0101011	Contoso 4G MP3 Player E400 Orange	4GB flash memory and FM Radio, LCD Display with ...	1	Contoso, Ltd	Contoso
8	0101013	Contoso 4GB Flash MP3 Player E401 Black	1.8" color LCD, play MP3, WMA and Video MTV, a...	1	Contoso, Ltd	Contoso
9	0101014	Contoso 4GB Flash MP3 Player E401 Silver	1.8" color LCD, play MP3, WMA and Video MTV, a...	1	Contoso, Ltd	Contoso
10	0101018	Contoso 8GB Super-Slim MP3/Video Player M800 Green	2" color LCD, Touchpad, Plays music, video, photo...	1	Contoso, Ltd	Contoso
11	0101019	Contoso 8GB Super-Slim MP3/Video Player M800 Pink	2" color LCD, Touchpad, Plays music, video, photo...	1	Contoso, Ltd	Contoso
12	0101020	Contoso 8GB MP3 Player new model M820 Black	2" LCD with blue-white LED, 320x240-pixel, plays m...	1	Contoso, Ltd	Contoso

Consulta executada com êxito.



# GABARITOS

10

## Questão 10

A empresa possui 16 produtos da marca Contoso, da cor Silver e com um UnitPrice entre 10 e 30. Descubra quais são esses produtos e ordene o resultado em ordem decrescente de acordo com o preço (UnitPrice).

--Questão 10

```
SELECT * FROM DimProduct
WHERE BrandName = 'Contoso' AND ColorName = 'Silver' AND Weight BETWEEN 10 AND 30
ORDER BY UnitPrice DESC
```

	ProductKey	ProductLabel	ProductName	ProductDescription	ProductSubcategoryKey	Manufacturer	BrandId
1	2122	0805007	Contoso Coffee Maker Super-Auto 12C X1250 Silver	Combination 12 cup super-auto coffee maker, 1250-wa...	45	Contoso, Ltd	Contoso
2	2100	0804016	Contoso Water Heater 7.2GPM X1800 Silver	Delivers 7.2 gallons per minute, Can save up to 50% of...	44	Contoso, Ltd	Contoso
3	2101	0804017	Contoso Water Heater 4.3GPM M1250 Silver	Delivers up to 4.3 gallons per minute. Endless hot water	44	Contoso, Ltd	Contoso
4	2102	0804018	Contoso Water Heater 4.0GPM M1250 Silver	Provides 4 gallons of hot water per minute, Includes on...	44	Contoso, Ltd	Contoso
5	2103	0804019	Contoso Water Heater 2.6GPM E0900 Silver	Provides 2.6 gallons of hot water per minute, average ...	44	Contoso, Ltd	Contoso
6	2124	0805009	Contoso Coffee Maker 12C M1000 Silver	Combination 12 cup coffee maker, 24-hour clock/timer...	45	Contoso, Ltd	Contoso
7	257	0203066	Contoso Home Theater System 5.1 Channel M1500 Si...	1000 watts over 5 channels, 1 center-channel speaker	11	Contoso, Ltd	Contoso
8	265	0203074	Contoso Home Theater System 2.1 Channel M1230 Si...	2.1 channel home theater system, includes 2 front spe...	11	Contoso, Ltd	Contoso
9	2104	0804020	Contoso Water Heater 1.5GPM E0800 Silver	Delivers 1.5 gallons per minute, 99-percent efficient	44	Contoso, Ltd	Contoso
10	607	0305069	Contoso Screen 106in M060 Silver	16:9 aspect ratio-106in, electric/motorized screen for h...	19	Contoso, Ltd	Contoso
11	1623	0602053	Contoso DVD Recorder L210 Silver	DVD and VHS Recorder with two way dubbing, Recor...	35	Contoso, Ltd	Contoso
12	2126	0805011	Contoso Coffee Maker 5C E0900 Silver	5-Cup Capacity with glass carafe, digital soft touch pad...	45	Contoso, Ltd	Contoso

Consulta executada com êxito.

LAPTOP-HBTCI8PQ (15.0 RTM) | LAPTOP-HBTCI8PQ\Marcus... | ContosoRetailDW | 00:00:00 | 16 linhas



# SQ L

## Funções de Agregação

# COUNT: Contando valores em uma tabela

A função **COUNT()** retorna a quantidade de linhas de uma coluna. É possível fazer uma contagem e especificar um determinado critério.

```
SQLQuery1.sql - LAP...us Cavalcanti (55)* ▾ X
-- Retorna a quantidade total de valores da coluna CustomerKey
SELECT
    COUNT(CustomerKey) AS 'Qtd. Clientes'
FROM dimCustomer

162 % ▾
Resultados Mensagens
Qtd. Clientes
1 18869
```

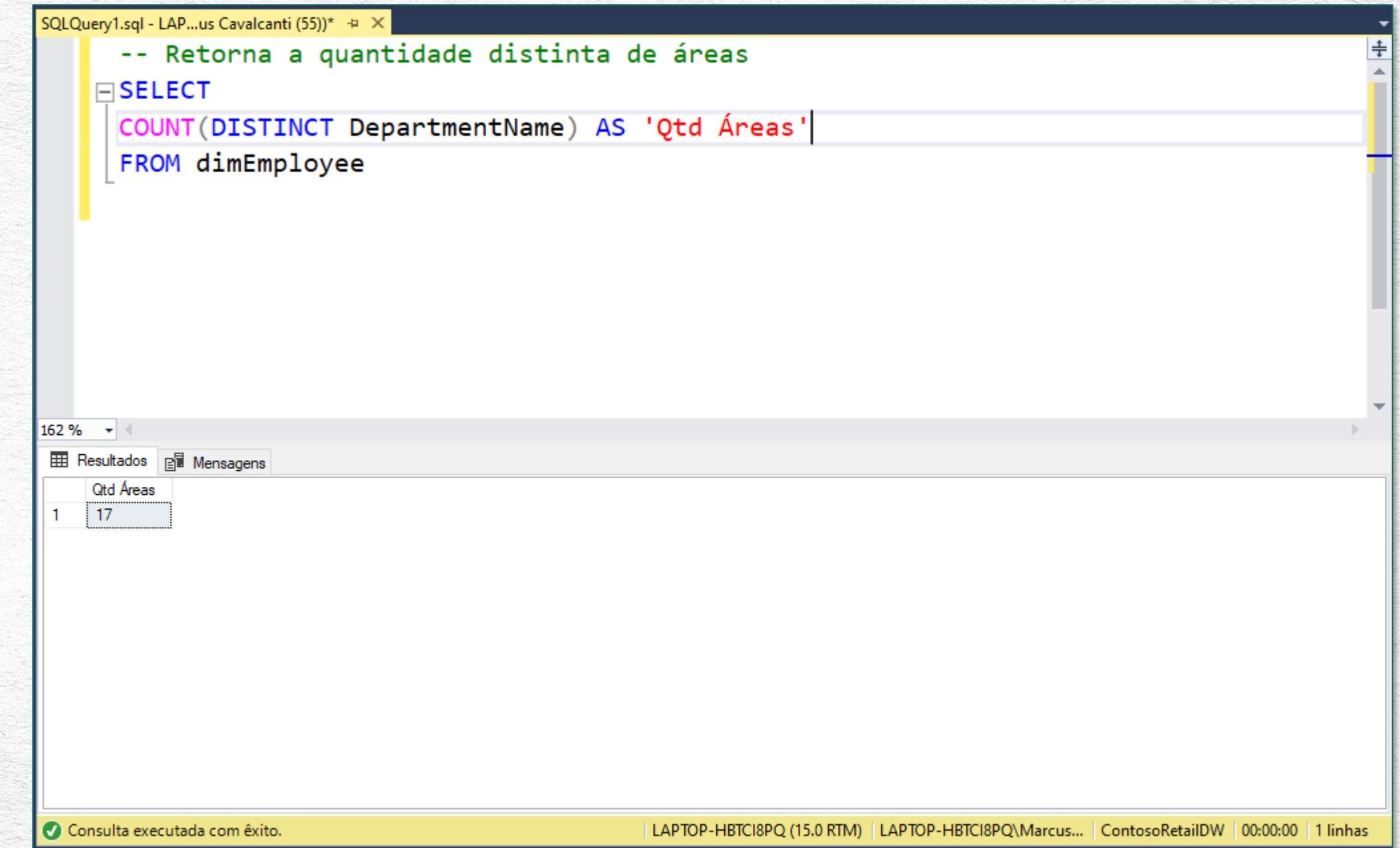
```
SQLQuery1.sql - LAP...us Cavalcanti (55)* ▾ X
-- Retorna a quantidade total de valores da coluna CustomerKey
SELECT
    COUNT(CustomerKey) AS 'Qtd. Mulheres'
FROM dimCustomer

-- Adiciona um filtro. Agora, ele irá retornar a quantidade de clientes do sexo feminino.
WHERE Gender = 'F'

162 % ▾
Resultados Mensagens
Qtd. Clientes
1 18869
```

# COUNT+DISTINCT: Contando valores distintos

Podemos usar o **COUNT()** combinado com o **DISTINCT()** para fazer uma contagem distinta dos valores de uma coluna.



The screenshot shows a SQL Server Management Studio window. The query window contains the following code:

```
-- Retorna a quantidade distinta de áreas
SELECT
    COUNT(DISTINCT DepartmentName) AS 'Qtd Áreas'
FROM dimEmployee
```

The results pane shows a single row with the column 'Qtd Áreas' containing the value '17'. At the bottom of the window, a message indicates the query was executed successfully.

Qtd Áreas
17

Consulta executada com êxito. | LAPTOP-HBTCI8PQ (15.0 RTM) | LAPTOP-HBTCI8PQ\Marcus... | ContosoRetailDW | 00:00:00 | 1 linhas

# SUM: Somando os valores de uma coluna

Para retornar a soma total de uma determinada coluna, usamos o **SUM()**.

Também é possível fazer uma soma e especificar um determinado critério.

The screenshot shows a SQL query being run in SQL Server Management Studio. The query is:

```
-- Retorna a soma da quantidade vendida da tabela factSales
SELECT
    SUM(SalesQuantity) AS 'Total Vendido'
FROM factSales
-- Filtra apenas para os produtos com preço unitário maior que 100
WHERE UnitPrice >= 100
```

The results window displays the output:

Total Vendido
1 27985853

At the bottom of the interface, a message indicates the query was executed successfully.

# AVG: Tirando a média dos valores de uma coluna

Para retornar a média de uma determinada coluna, usamos o **AVG()**.

Também é possível fazer uma média e especificar um determinado critério.

The screenshot shows the SSMS interface with a query window containing the following SQL code:

```
-- Retorna a média da quantidade vendida da tabela factSales
SELECT
    AVG(SalesQuantity) AS 'Média de Vendas'
FROM factSales

-- Filtra apenas para os produtos com preço unitário maior que 100
WHERE UnitPrice >= 100
```

The results pane displays the output of the query:

Média de Vendas
12

At the bottom of the results pane, a message indicates the query was executed successfully: "Consulta executada com êxito."

# MIN e MAX: Mínimo e Máximo valor de uma coluna

O **MIN()** retorna o mínimo valor de uma determinada coluna.

O **MAX()** retorna o máximo valor de uma determinada coluna.

The screenshot shows a SQL query window with two queries. The first query retrieves the minimum sales quantity from the factSales table, labeled 'Min Venda'. The second query retrieves the maximum sales quantity from the same table, labeled 'Max Venda'. Both queries use the MIN and MAX functions respectively, along with AS clauses to name the results.

```
-- Retorna a menor quantidade vendida em SalesQuantity
SELECT
    MIN(SalesQuantity) AS 'Min Venda'
FROM factSales

-- Retorna a maior quantidade vendida em SalesQuantity
SELECT
    MAX(SalesQuantity) AS 'Max Venda'
FROM factSales
```

Resultados

Min Venda
1 2

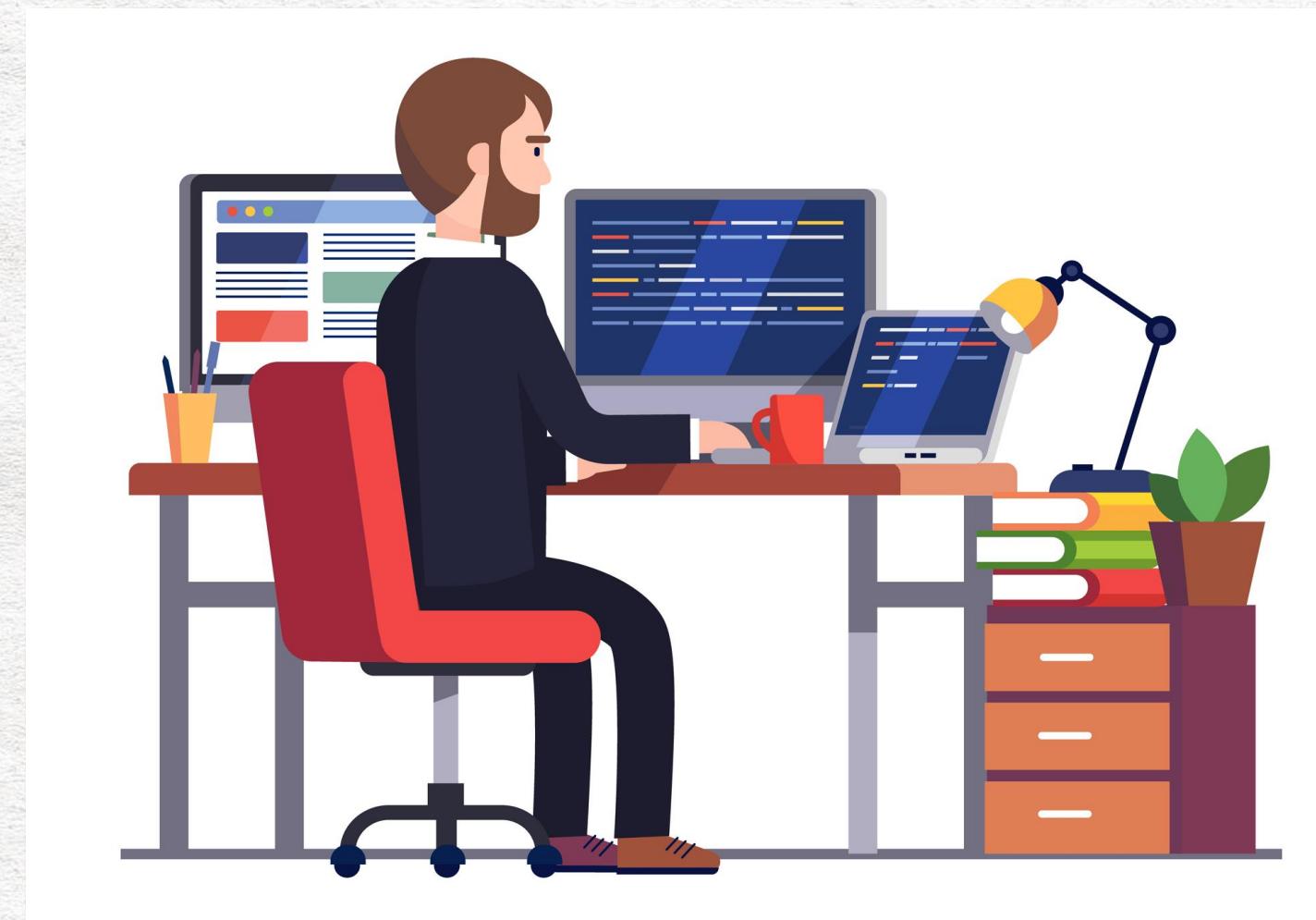
  

Max Venda
1 2880

Consulta executada com êxito.

LAPTOP-HBTCI8PQ (15.0 RTM) | LAPTOP-HBTCI8PQ\Marcus... | ContosoRetailDW | 00:00:18 | 2 linhas

# EXERCÍCIOS



1

## Questão 1

O gerente comercial pediu a você uma análise da Quantidade Vendida e Quantidade Devolvida para o canal de venda mais importante da empresa: Store.

Utilize uma função SQL para fazer essas consultas no seu banco de dados.  
Obs: Faça essa análise considerando a tabela FactSales.

2

## Questão 2

Uma nova ação no setor de Marketing precisará avaliar a média salarial de todos os clientes da empresa, mas apenas de ocupação Professional. Utilize um comando SQL para atingir esse resultado.

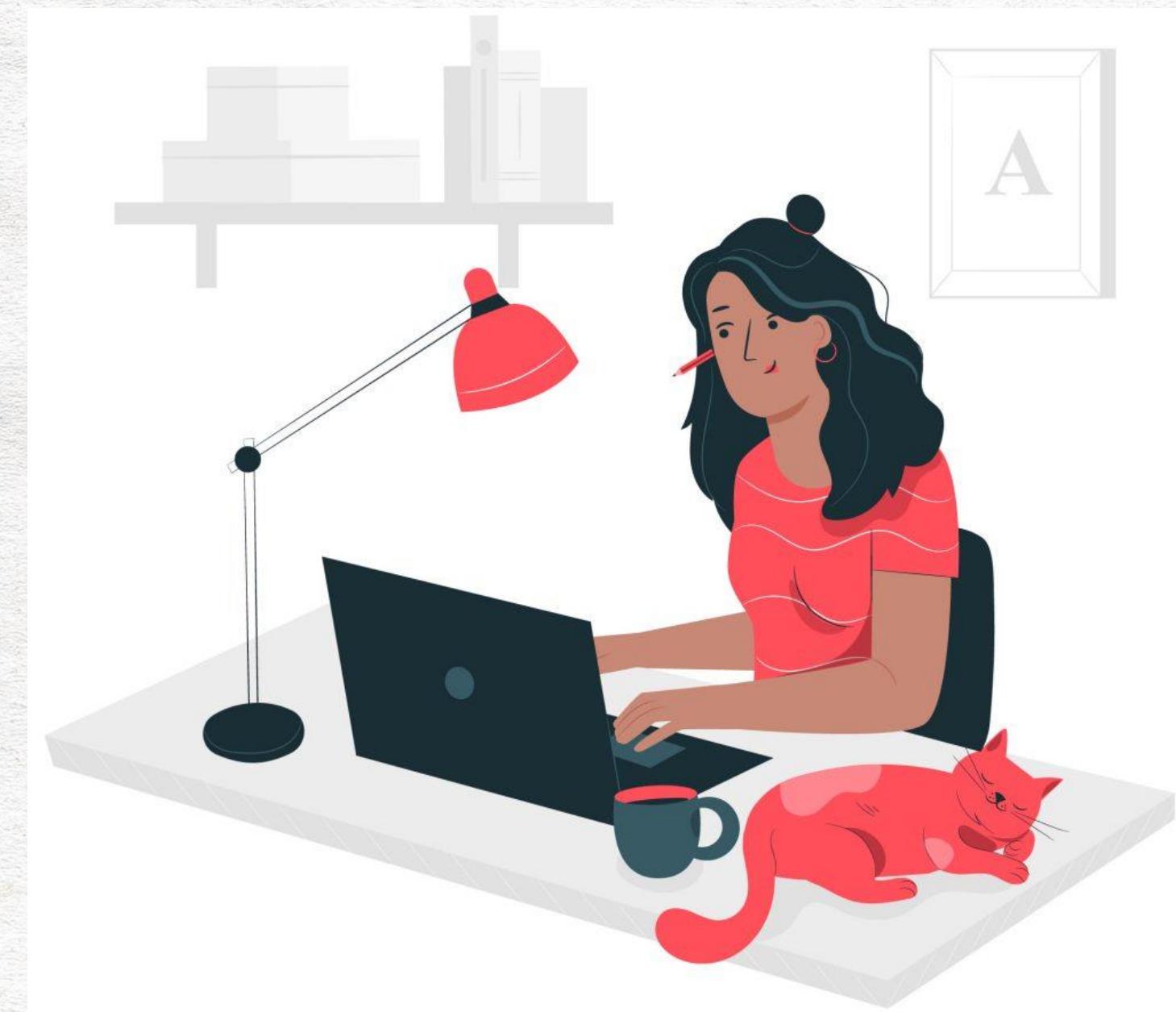
3

## Questão 3

Você precisará fazer uma análise da quantidade de funcionários das lojas registradas na empresa. O seu gerente te pediu os seguintes números e informações:

- Quantos funcionários tem a loja com mais funcionários?
- Qual é o nome dessa loja?
- Quantos funcionários tem a loja com menos funcionários?
- Qual é o nome dessa loja?

# EXERCÍCIOS



4

## Questão 4

A área de RH está com uma nova ação para a empresa, e para isso precisa saber a quantidade total de funcionários do sexo Masculino e do sexo Feminino.

- Descubra essas duas informações utilizando o SQL.
- O funcionário e a funcionária mais antigos receberão uma homenagem. Descubra as seguintes informações de cada um deles: Nome, E-mail, Data de Contratação.

5

## Questão 5

Agora você precisa fazer uma análise dos produtos. Será necessário descobrir as seguintes informações:

- Quantidade distinta de cores de produtos.
- Quantidade distinta de marcas
- Quantidade distinta de classes de produto

Para simplificar, você pode fazer isso em uma mesma consulta.

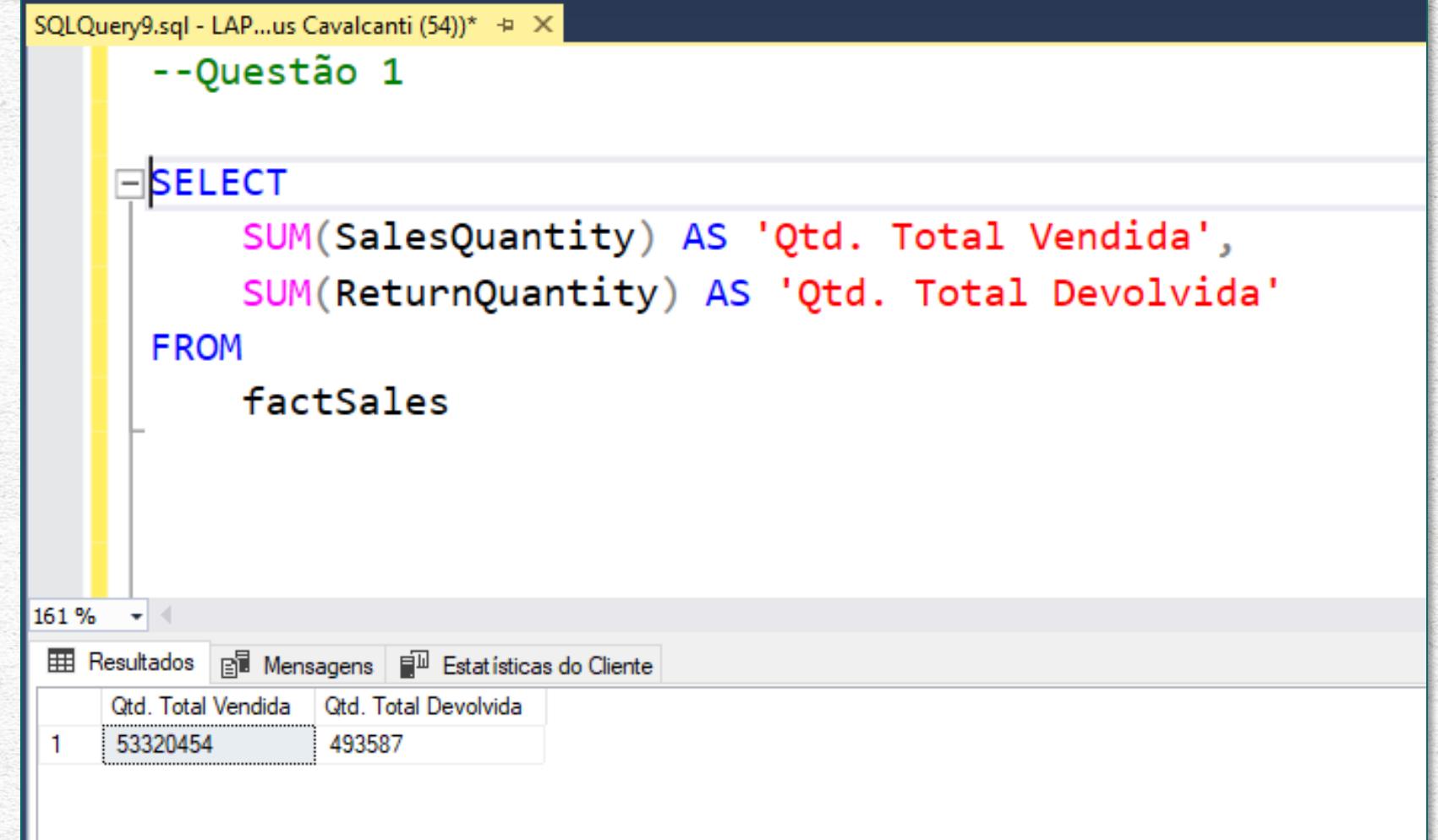
# GABARITOS

1

## Questão 1

O gerente comercial pediu a você uma análise da Quantidade Vendida e Quantidade Devolvida para o canal de venda mais importante da empresa: Store.

Utilize uma função SQL para fazer essas consultas no seu banco de dados.  
Obs: Faça essa análise considerando a tabela FactSales.



```
--Questão 1
SELECT
    SUM(SalesQuantity) AS 'Qtd. Total Vendida',
    SUM(ReturnQuantity) AS 'Qtd. Total Devolvida'
FROM
    factSales
```

	Qtd. Total Vendida	Qtd. Total Devolvida
1	53320454	493587

# GABARITOS

2

## Questão 2

Uma nova ação no setor de Marketing precisará avaliar a média salarial de todos os clientes da empresa, mas apenas de ocupação Professional. Utilize um comando SQL para atingir esse resultado.

The screenshot shows a SQL query window titled "SQLQuery9.sql - LAP...us Cavalcanti (54)\*". The query is:

```
--Questão 2  
SELECT  
    AVG(YearlyIncome) AS 'Média Salarial'  
FROM  
    DimCustomer  
WHERE Occupation = 'Professional'
```

The results pane shows a single row with the title "Média Salarial" and the value "74184,7826".

# GABARITOS

3

## Questão 3

Você precisará fazer uma análise da quantidade de funcionários das lojas registradas na empresa. O seu gerente te pediu os seguintes números e informações:

- Quantos funcionários tem a loja com mais funcionários?
- Qual é o nome dessa loja?
- Quantos funcionários tem a loja com menos funcionários?
- Qual é o nome dessa loja?

```

SQLQuery9.sql - LAP...us Cavalcanti (54)*  X
--Questão 3
--a) e c)

SELECT
    MAX(EmployeeCount) AS 'Maior Qtd. Funcionários',
    MIN(EmployeeCount) AS 'Menor Qtd. Funcionários'
FROM
    DimStore
--b)

SELECT TOP(1)
    StoreName,
    EmployeeCount
FROM
    DimStore
ORDER BY EmployeeCount DESC
--d)

SELECT TOP(1)
    StoreName,
    EmployeeCount
FROM
    DimStore
WHERE EmployeeCount IS NOT NULL
ORDER BY EmployeeCount ASC
  
```

83 %

	Maior Qtd. Funcionários	Menor Qtd. Funcionários
1	325	7

	StoreName	EmployeeCount
1	Contoso North America Online Store	325

	StoreName	EmployeeCount
1	Contoso Europe Online Store	7

Consulta executada com êxito.

LAPTOP-HBTCL8PQ (15.0 RTM) | LAPTOP-HBTCL8PQ\Marcus... | ContosoRetailDW | 00:00:00 | 3 linhas



# GABARITOS

4

## Questão 4

A área de RH está com uma nova ação para a empresa, e para isso precisa saber a quantidade total de funcionários do sexo Masculino e do sexo Feminino.

- Descubra essas duas informações utilizando o SQL.
- O funcionário e a funcionária mais antigos receberão uma homenagem. Descubra as seguintes informações de cada um deles: Nome, E-mail, Data de Contratação.

```
--Questão 4
--a)
SELECT
    COUNT(FirstName)
FROM
    DimEmployee
--WHERE Gender = 'M'
WHERE Gender = 'F'

--b)
SELECT TOP(1)
    FirstName,
    HireDate,
    EmailAddress
FROM
    DimEmployee
WHERE Gender = 'F'
ORDER BY HireDate ASC
```

110 %

	Resultados	Mensagens	Estatísticas do Cliente
1	(Nenhum nome de coluna)		
1	87		

	FirstName	HireDate	EmailAddress
1	Terry	1998-01-26	jolynn0@contoso.com



# GABARITOS

5

## Questão 5

Agora você precisa fazer uma análise dos produtos. Será necessário descobrir as seguintes informações:

- a. Quantidade distinta de cores de produtos.
- b. Quantidade distinta de marcas
- c. Quantidade distinta de classes de produto

Para simplificar, você pode fazer isso em uma mesma consulta.

```
--Questão 5

SELECT
    COUNT(DISTINCT BrandName) AS 'Nome da Marca',
    COUNT(DISTINCT ClassName) AS 'Classe do Produto',
    COUNT(DISTINCT ColorName) AS 'Cor do Produto'
FROM
    dimProduct
```

	Nome da Marca	Classe do Produto	Cor do Produto
1	11	3	16

# S Q L

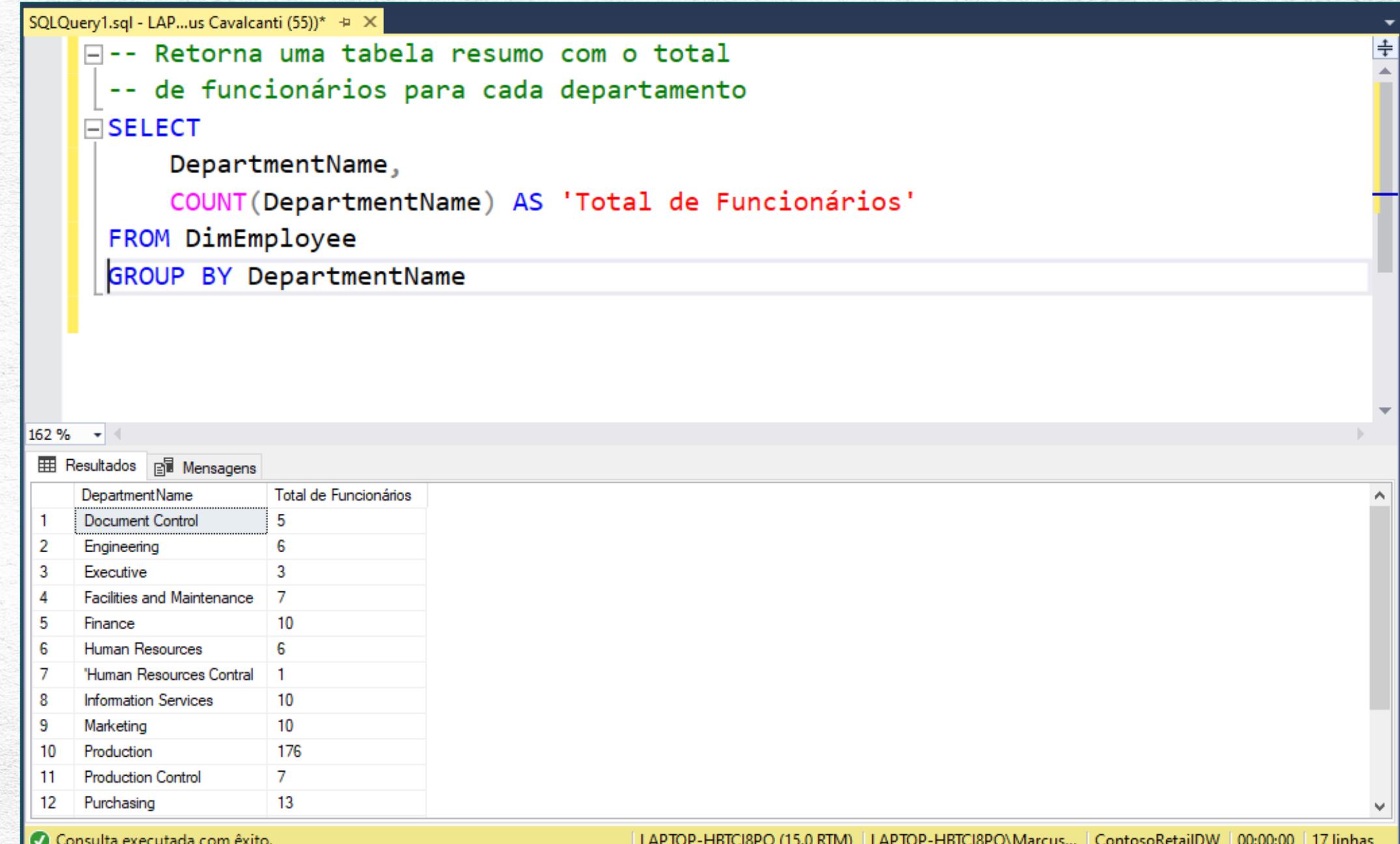
## Criando agrupamentos



# GROUP BY: Criando agrupamentos

O comando **GROUP BY** permite agrupar valores de acordo com uma coluna, por exemplo "total de funcionários de acordo com o departamento".

O **GROUP BY** é usado junto com funções de agregação (COUNT(), MAX(), MIN(), SUM(), AVG()) para agrupar valores de acordo com uma ou mais colunas.



The screenshot shows a SQL Server Management Studio (SSMS) interface. The top pane displays a SQL query in the 'SQLQuery1.sql' editor:

```
-- Retorna uma tabela resumo com o total  
-- de funcionários para cada departamento  
SELECT  
    DepartmentName,  
    COUNT(DepartmentName) AS 'Total de Funcionários'  
FROM DimEmployee  
GROUP BY DepartmentName
```

The bottom pane shows the 'Resultados' tab with the query results:

	DepartmentName	Total de Funcionários
1	Document Control	5
2	Engineering	6
3	Executive	3
4	Facilities and Maintenance	7
5	Finance	10
6	Human Resources	6
7	'Human Resources Contral	1
8	Information Services	10
9	Marketing	10
10	Production	176
11	Production Control	7
12	Purchasing	13

A message at the bottom of the results pane says: 'Consulta executada com êxito.' (Query executed successfully.)

# GROUP BY+ORDER BY: Ordenando agrupamentos

O comando GROUP BY combinado com o ORDER BY permite que a tabela agrupada seja também ordenada.

The screenshot shows a SQL query window titled "SQLQuery1.sql - LAP...us Cavalcanti (55)\*". The query is as follows:

```
--Agrupa a tabela DimStore de acordo com a contagem  
-- de funcionários e ordena de acordo com essa quantidade  
SELECT  
    StoreType,  
    SUM(EmployeeCount) AS 'Qtd. Total Funcionários'  
FROM  
    DimStore  
GROUP BY StoreType  
ORDER BY SUM(EmployeeCount) DESC
```

The results tab displays the following data:

	StoreType	Qtd. Total Funcionários
1	Store	10535
2	Online	340
3	Catalog	120
4	Reseller	44

# GROUP BY+WHERE: Filtrando a tabela principal antes de agrupar

O comando **GROUP BY** combinado com o **WHERE** permite que a tabela principal seja filtrada antes de o agrupamento ser criado.

The screenshot shows a SQL query in the SQL Query Editor and its execution results. The query is:

```
-- Cria um agrupamento por Cor do Produto,  
-- mas antes considera apenas os produtos da marca Contoso  
SELECT  
    ColorName AS 'Cor do Produto',  
    COUNT(ColorName) AS 'Qtd. Total'  
FROM  
    DimProduct  
WHERE BrandName = 'Contoso'  
GROUP BY ColorName
```

The results grid displays the following data:

	Cor do Produto	Qtd. Total
1	Black	174
2	Blue	45
3	Brown	15
4	Gold	10
5	Green	15
6	Grey	95
7	Orange	8
8	Pink	21
9	Purple	1
10	Red	36
11	Silver	109
12	Silver Grey	4

At the bottom of the results grid, a message indicates: "Consulta executada com êxito." (Query executed successfully).

# GROUP BY+HAVING: Filtrando um agrupamento

O comando **GROUP BY** combinado com o **HAVING** permite que a tabela agrupada seja filtrada e que apenas os valores totais que atendam ao critério do HAVING sejam considerados.

The screenshot shows a SQL query in the SQL Query window of SSMS. The query uses GROUP BY BrandName and HAVING COUNT(BrandName) >= 200 to filter brands with at least 200 products. The results grid shows four rows: Litware, Contoso, Proseware, and Fabrikam, each with their respective product counts.

```
-- Cria um agrupamento por Marca.  
-- Em seguida, é utilizado o HAVING para  
-- filtrar a tabela agrupada.  
SELECT  
    BrandName AS 'Marca',  
    COUNT(BrandName) AS 'Total Marca'  
FROM  
    DimProduct  
GROUP BY BrandName  
HAVING COUNT(BrandName) >= 200
```

	Marca	Total Marca
1	Litware	264
2	Contoso	710
3	Proseware	244
4	Fabrikam	267

# WHERE vs. HAVING: Qual é a diferença?

É muito importante que fique claro pra você a diferença entre o comando WHERE e o HAVING.

Para entender a diferença entre os dois, vale lembrar que o GROUP BY permite que a gente crie um agrupamento a partir de uma tabela principal.

O comando WHERE será utilizado para filtrar a tabela principal antes de criar o agrupamento.

Já o HAVING será utilizado para filtrar a tabela agrupada.

Portanto, podemos entender que:

- O WHERE será utilizado antes do agrupamento ser feito.
- O HAVING será um filtro utilizado após o agrupamento ser feito.

The screenshot shows a SQL query in the SQL Query Editor titled 'SQLQuery1.sql'. The code is as follows:

```
-- No código abaixo, antes de realizar o agrupamento,  
-- consideramos apenas os produtos da classe 'Economy'.  
-- E após realizar o agrupamento, utiliza-se o HAVING para filtrar a tabela agrupada  
SELECT  
    BrandName AS 'Marca',  
    COUNT(BrandName) AS 'Total Marca'  
FROM  
    DimProduct  
WHERE ClassName = 'Economy'      -- Filtra a tabela original, antes do agrupamento  
GROUP BY BrandName  
HAVING COUNT(BrandName) >= 200   -- Filtra a tabela depois de agrupada
```

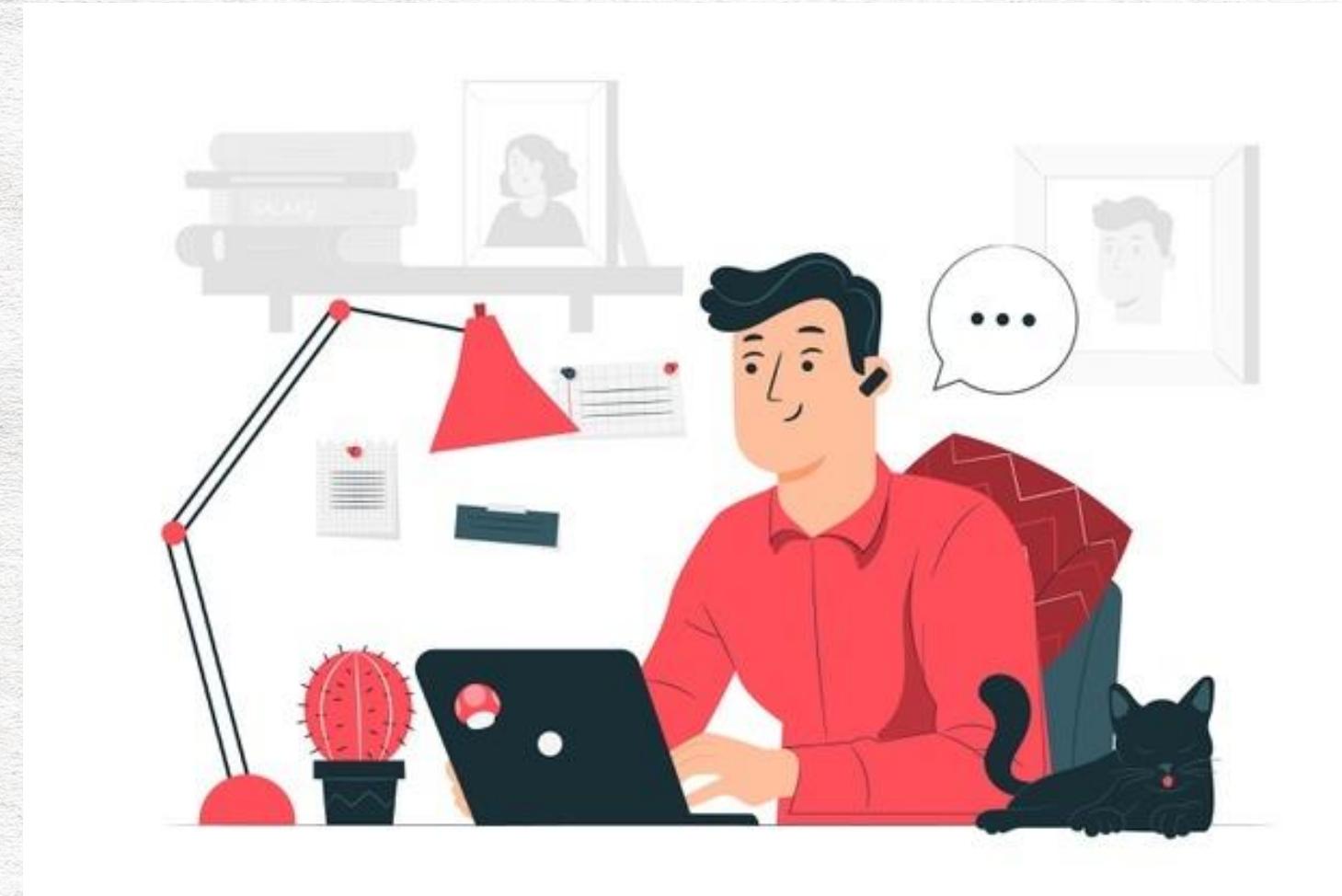
The results pane shows a single row of data:

Marca	Total Marca
Contoso	263

At the bottom of the results pane, a message says 'Consulta executada com êxito.' (Query executed successfully).



# EXERCÍCIOS



1

## Questão 1

- Faça um resumo da quantidade vendida (SalesQuantity) de acordo com o canal de vendas (channelkey).
- Faça um agrupamento mostrando a quantidade total vendida (SalesQuantity) e quantidade total devolvida (Return Quantity) de acordo com o ID das lojas (StoreKey).
- Faça um resumo do valor total vendido (SalesAmount) para cada canal de venda, mas apenas para o ano de 2007.

2

## Questão 2

Você precisa fazer uma análise de vendas por produtos. O objetivo final é descobrir o valor total vendido (SalesAmount) por produto (ProductKey).

- A tabela final deverá estar ordenada de acordo com a quantidade vendida e, além disso, mostrar apenas os produtos que tiveram um resultado final de vendas maior do que \$5.000.000.
- Faça uma adaptação no exercício anterior e mostre os Top 10 produtos com mais vendas. Desconsidere o filtro de \$5.000.000 aplicado.

# EXERCÍCIOS



3

## Questão 3

- Você deve fazer uma consulta à tabela FactOnlineSales e descobrir qual é o ID (CustomerKey) do cliente que mais realizou compras online (de acordo com a coluna SalesQuantity).
- Feito isso, faça um agrupamento de total vendido (SalesQuantity) por ID do produto e descubra quais foram os top 3 produtos mais comprados pelo cliente da letra a).

4

## Questão 4

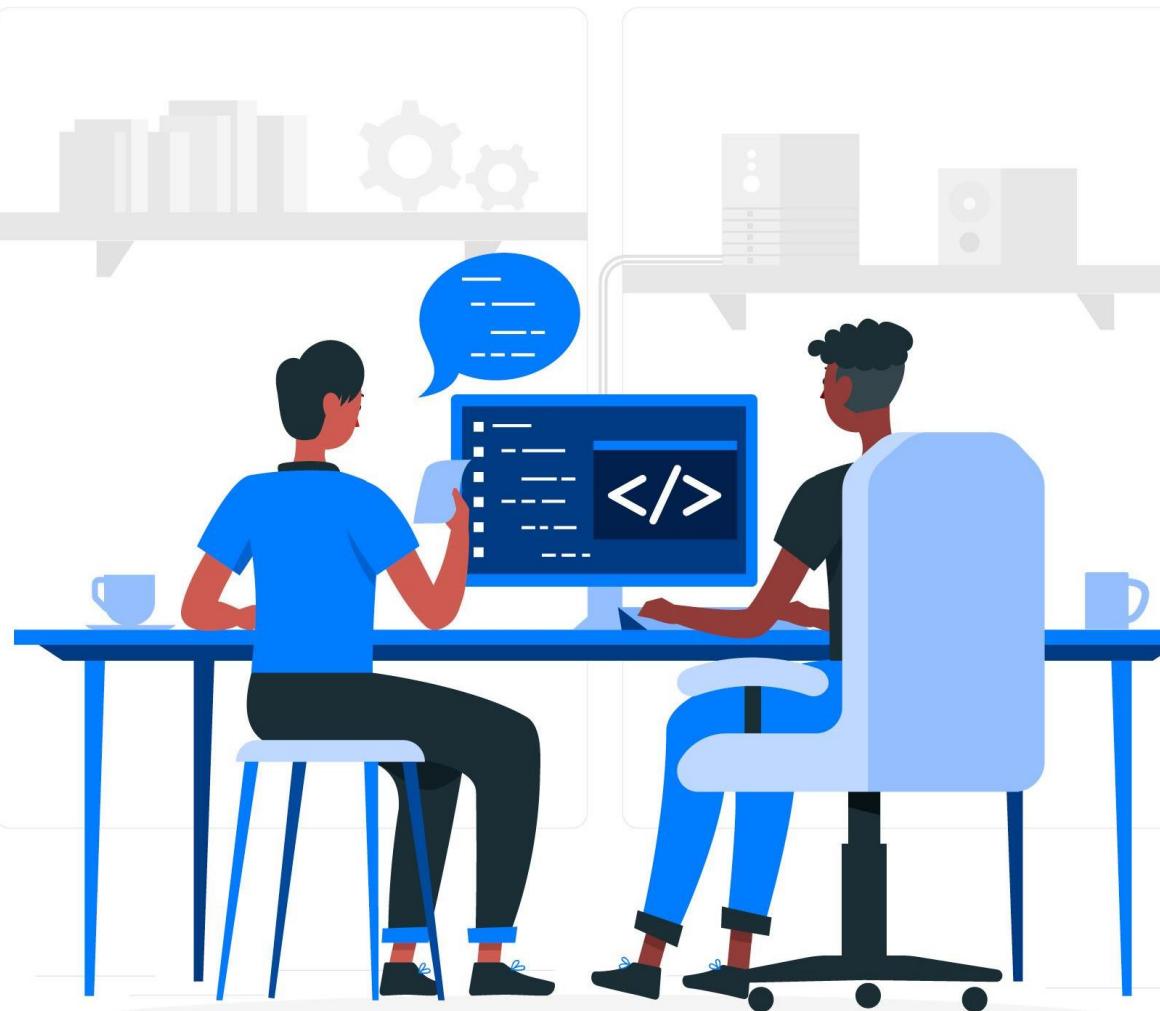
- Faça um agrupamento e descubra a quantidade total de produtos por marca.
- Determine a média do preço unitário (UnitPrice) para cada ClassName.
- Faça um agrupamento de cores e descubra o peso total que cada cor de produto possui.

5

## Questão 5

Você deverá descobrir o peso total para cada tipo de produto (StockTypeName). A tabela final deve considerar apenas a marca 'Contoso' e ter os seus valores classificados em ordem decrescente.

# EXERCÍCIOS



6

## Questão 6

Você seria capaz de confirmar se todas as marcas dos produtos possuem à disposição todas as 16 opções de cores?

7

## Questão 7

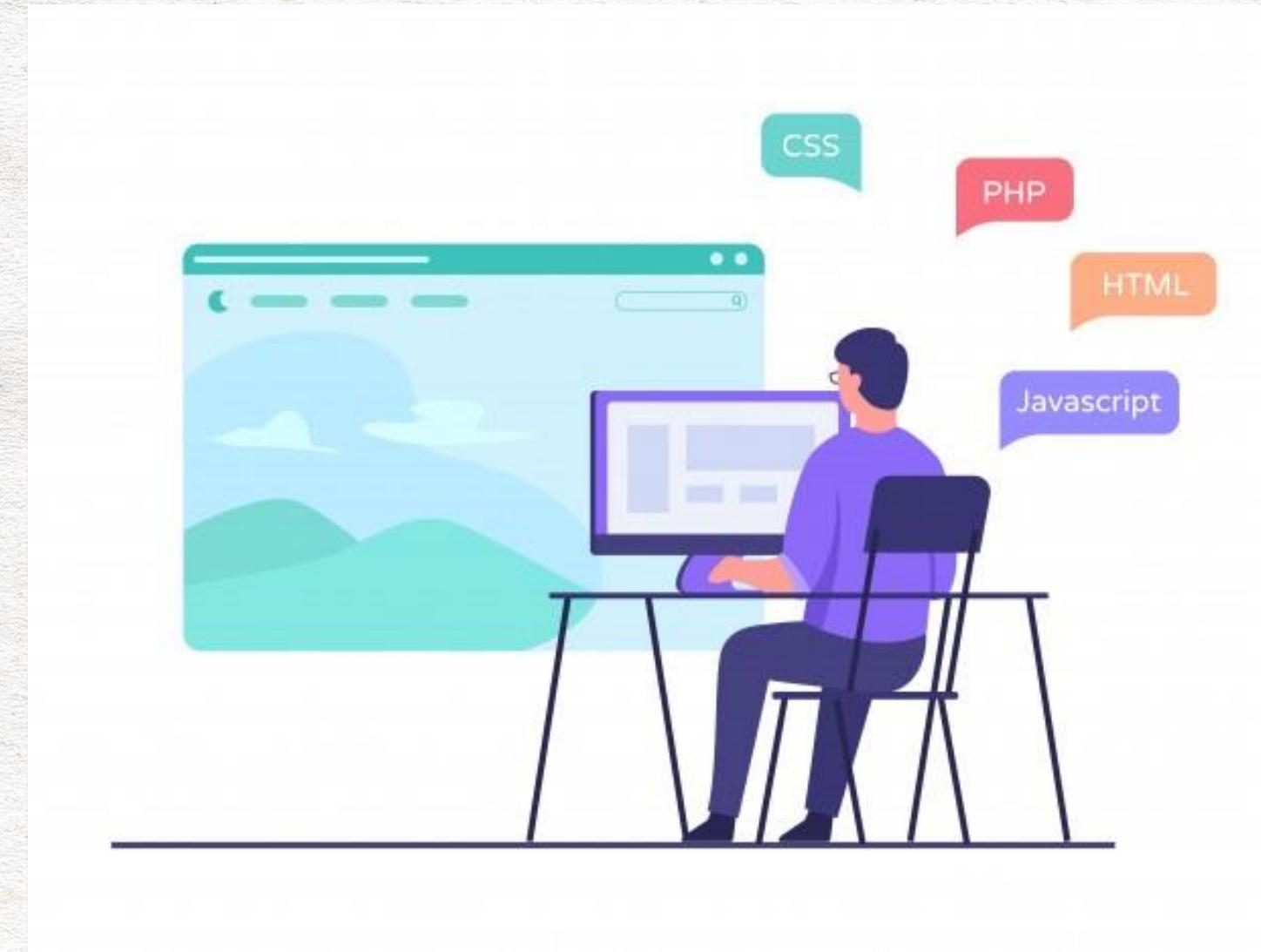
Faça um agrupamento para saber o total de clientes de acordo com o Sexo e também a média salarial de acordo com o Sexo. Corrija qualquer resultado “inesperado” com os seus conhecimentos em SQL.

8

## Questão 8

Faça um agrupamento para descobrir a quantidade total de clientes e a média salarial de acordo com o seu nível escolar. Utilize a coluna Education da tabela DimCustomer para fazer esse agrupamento.

# EXERCÍCIOS



9

## Questão 9

Faça uma tabela resumo mostrando a quantidade total de funcionários de acordo com o Departamento (DepartmentName). Importante: Você deverá considerar apenas os funcionários ativos.

10

## Questão 10

Faça uma tabela resumo mostrando o total de VacationHours para cada cargo (Title). Você deve considerar apenas as mulheres, dos departamentos de Production, Marketing, Engineering e Finance, para os funcionários contratados entre os anos de 1999 e 2000.

# GABARITOS

1

## Questão 1

- a. Faça um resumo da quantidade vendida (SalesQuantity) de acordo com o canal de vendas (channelkey).
- b. Faça um agrupamento mostrando a quantidade total vendida (SalesQuantity) e quantidade total devolvida (Return Quantity) de acordo com o ID das lojas (StoreKey).
- c. Faça um resumo do valor total vendido (SalesAmount) para cada canal de venda, mas apenas para o ano de 2007.

```
SQLQuery9.sql - LAP...us Cavalcanti (54)*  X
--Questão 1
--a)
SELECT
    channelKey,
    SUM(SalesQuantity)
FROM
    FactSales
GROUP BY channelKey

--b)
SELECT
    StoreKey,
    SUM(SalesQuantity),
    SUM(ReturnQuantity)
FROM
    FactSales
GROUP BY StoreKey

--c)
SELECT
    channelKey,
    SUM(SalesQuantity)
FROM
    FactSales
WHERE DateKey BETWEEN '01/01/2007' AND '31/12/2007'
GROUP BY channelKey
```

# GABARITOS

2

## Questão 2

Você precisa fazer uma análise de vendas por produtos. O objetivo final é descobrir o valor total vendido (SalesAmount) por produto (ProductKey).

- A tabela final deverá estar ordenada de acordo com a quantidade vendida e, além disso, mostrar apenas os produtos que tiveram um resultado final de vendas maior do que \$5.000.000.
- Faça uma adaptação no exercício anterior e mostre os Top 10 produtos com mais vendas. Desconsidere o filtro de \$5.000.000 aplicado.

```
SQLQuery9.sql - LAP...us Cavalcanti (54)*  ↗ X
--Questão 2
--a)
SELECT
    ProductKey,
    SUM(SalesAmount)
FROM
    FactSales
GROUP BY ProductKey
HAVING SUM(SalesAmount) >= 5000000
ORDER BY SUM(SalesAmount) DESC

--b)
SELECT TOP(10)
    ProductKey,
    SUM(SalesAmount)
FROM
    FactSales
GROUP BY ProductKey
ORDER BY SUM(SalesAmount) DESC
```

# GABARITOS

3

## Questão 3

- Você deve fazer uma consulta à tabela FactOnlineSales e descobrir qual é o ID (CustomerKey) do cliente que mais realizou compras online (de acordo com a coluna SalesQuantity).
- Feito isso, faça um agrupamento de total vendido (SalesQuantity) por ID do produto e descubra quais foram os top 3 produtos mais comprados pelo cliente da letra a).

```
--Questão 3
--a)
SELECT TOP(1)
    CustomerKey,
    SUM(SalesQuantity)
FROM
    FactOnlineSales
GROUP BY CustomerKey
ORDER BY SUM(SalesQuantity) DESC

--b)
SELECT TOP(3)
    ProductKey,
    SUM(SalesQuantity)
FROM
    FactOnlineSales
GROUP BY ProductKey
ORDER BY SUM(SalesQuantity) DESC
```

# GABARITOS

4

## Questão 4

- Faça um agrupamento e descubra a quantidade total de produtos por marca.
- Determine a média do preço unitário (UnitPrice) para cada ClassName.
- Faça um agrupamento de cores e descubra o peso total que cada cor de produto possui.

```
--Questão 4
--a)
SELECT
    BrandName,
    COUNT(*)
FROM
    DimProduct
GROUP BY BrandName

--b)
SELECT
    ClassName,
    AVG(UnitPrice)
FROM
    DimProduct
GROUP BY ClassName

--c)
SELECT
    ColorName,
    SUM(Weight)
FROM
    DimProduct
GROUP BY ColorName
```

# GABARITOS

5

## Questão 5

Você deverá descobrir o peso total para cada tipo de produto (StockTypeName). A tabela final deve considerar apenas a marca 'Contoso' e ter os seus valores classificados em ordem decrescente.

The screenshot shows a SQL query window titled "SQLQuery9.sql - LAP...us Cavalcanti (54)\*". The query is:

```
--Questão 5
SELECT
    StockTypeName,
    SUM(Weight)
FROM
    DimProduct
WHERE BrandName = 'Contoso'
GROUP BY StockTypeName
```

The results pane shows a table with three rows:

	StockTypeName	(Nenhum nome de coluna)
1	High	14625,89
2	Low	1215,6
3	Mid	1665,15

# GABARITOS

6

## Questão 6

Você seria capaz de confirmar se todas as marcas dos produtos possuem à disposição todas as 16 opções de cores? **Nenhuma marca possui um exemplar de cada cor.**

The screenshot shows a SQL query in the SQL Query window and its results in the Results window.

```
--Questão 6
SELECT
    BrandName,
    COUNT(DISTINCT ColorName)
FROM
    DimProduct
GROUP BY BrandName
```

The results table shows the count of distinct colors for each brand:

	BrandName	(Nenhum nome de coluna)
1	A. Datum	10
2	Adventure Works	7
3	Contoso	15
4	Fabrikam	12
5	Litware	12
6	Northwind Traders	9
7	Proseware	7
8	Southridge Video	10
9	Tailspin Toys	11
10	The Phone Company	6
11	Wide World Importers	12

# GABARITOS

7

## Questão 7

Faça um agrupamento para saber o total de clientes de acordo com o Sexo e também a média salarial de acordo com o Sexo. Corrija qualquer resultado “inesperado” com os seus conhecimentos em SQL.

The screenshot shows a SQL query in the SQL Query window titled 'SQLQuery9.sql - LAP...us Cavalcanti (54)\*'. The query is:

```
-- Questão 7
SELECT
    Gender AS 'Sexo',
    COUNT(Gender) AS 'Total'
FROM
    DimCustomer
WHERE Gender IS NOT NULL
GROUP BY
    Gender
ORDER BY COUNT(Gender) DESC
```

The results pane shows a table with two rows:

	Sexo	Total
1	M	9351
2	F	9133

# GABARITOS

8

## Questão 8

Faça um agrupamento para descobrir a quantidade total de clientes e a média salarial de acordo com o seu nível escolar. Utilize a coluna Education da tabela DimCustomer para fazer esse agrupamento.

The screenshot shows a SQL query in the SQL Query window titled '--Questão 8'. The query uses the DimCustomer table to group by Education level, calculate the count of customers, and find the average yearly income. The results are displayed in a table below the query.

```
--Questão 8
SELECT
    Education AS 'Nível Escolar',
    COUNT(Education) AS 'Qtd. Total por Escolaridade',
    AVG(YearlyIncome) AS 'Média Salarial'
FROM
    DimCustomer
WHERE Education IS NOT NULL
GROUP BY Education
```

	Nível Escolar	Qtd. Total por Escolaridade	Média Salarial
1	High School	3294	49049,7874
2	Partial High School	1581	39468,6907
3	Partial College	5064	55211,2954
4	Graduate Degree	3189	66095,9548
5	Bachelors	5356	64395,0709

# GABARITOS

9

## Questão 9

Faça uma tabela resumo mostrando a quantidade total de funcionários de acordo com o Departamento (DepartmentName). Importante: Você deverá considerar apenas os funcionários ativos.

The screenshot shows a SQL query window titled "SQLQuery9.sql - LAP...us Cavalcanti (54)\*". The query is:

```
--Questão 9
SELECT
    DepartmentName,
    COUNT(*)
FROM
    DimEmployee
WHERE Status = 'Current'
GROUP BY DepartmentName
```

The results grid displays the following data:

	DepartmentName	(Nenhum nome de coluna)
1	Document Control	5
2	Engineering	6
3	Executive	2
4	Facilities and Maintenance	7
5	Finance	10
6	Human Resources	6
7	'Human Resources Contral	1
8	Information Services	10
9	Marketing	9
10	Production	176

At the bottom of the results grid, a message says "Consulta executada com êxito." (Query executed successfully).

# GABARITOS

10

## Questão 10

Faça uma tabela resumo mostrando o total de VacationHours para cada cargo (Title). Você deve considerar apenas as mulheres, dos departamentos de Production, Marketing, Engineering e Finance, para os funcionários contratados entre os anos de 1999 e 2000.

The screenshot shows a SQL query in the SQL Query window titled '--Questão 10'. The query selects the Title and sum of VacationHours from the DimEmployee table, filtered by gender 'M', department names 'Production', 'Marketing', 'Finance', and 'Engineering', and hire date between '01/01/1999' and '31/12/2000', grouped by Title. The results window shows two rows: 'Sales State Manager' with 280 vacation hours and 'Sales Store Manager' with 5698 vacation hours.

```
--Questão 10
SELECT
    Title,
    SUM(VacationHours)
FROM
    DimEmployee
WHERE Gender = 'M' AND DepartmentName IN ('Production', 'Marketing', 'Finance', 'Engineering') AND HireDate BETWEEN '01/01/1999' AND '31/12/2000'
GROUP BY Title
```

Title	(Nenhum nome de coluna)
1 Sales State Manager	280
2 Sales Store Manager	5698

# SQL JOINS

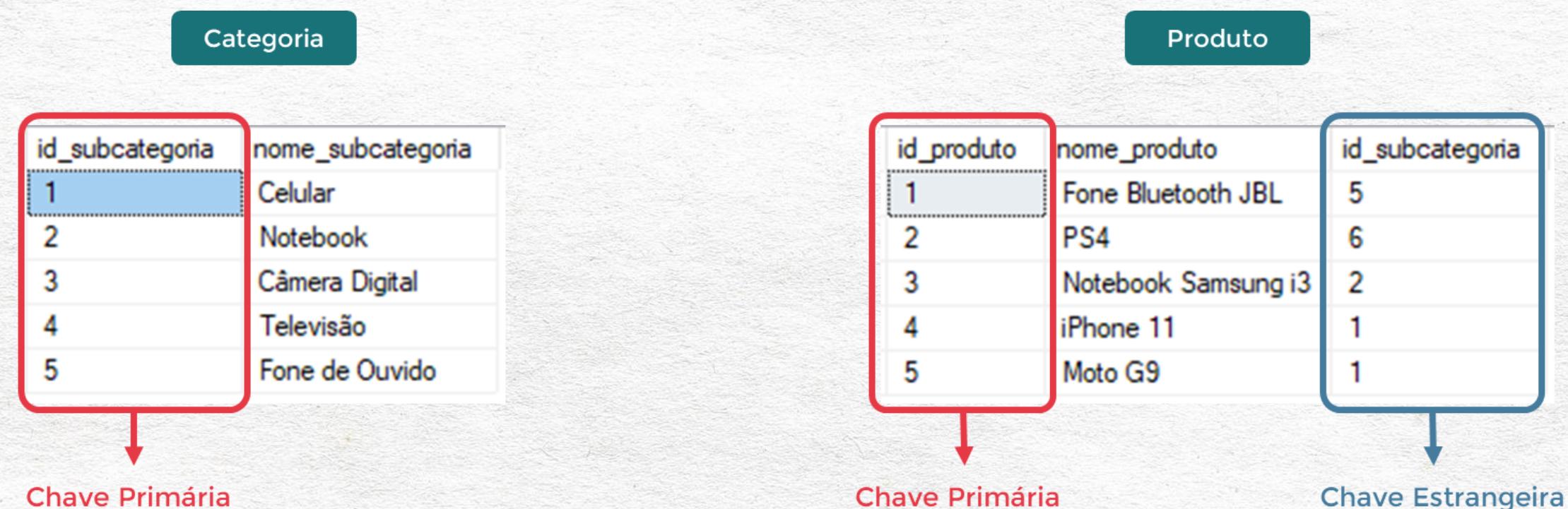


# Chave Primária vs. Chave Estrangeira

Uma **Chave Primária** é uma coluna que identifica as informações distintas em uma tabela. Geralmente é uma coluna de ID. Toda tabela terá uma, e somente uma, chave primária. Essa chave é utilizada como identificador único da tabela, sendo representada por uma coluna que não receberá valores repetidos.

Já uma **Chave Estrangeira** é uma coluna que permite relacionar as linhas de uma segunda tabela com a Chave Primária de uma primeira tabela.

Como pode ser visto abaixo, a tabela **Categoria** possui uma coluna chamada `id_subcategoria`, com valores que não se repetem. Essa será a **Chave Primária**. Já na tabela de Produtos, a coluna de `id_subcategoria` também aparece, mas os valores se repetem. Isso porque podemos ter mais de um produto pertencendo à mesma categoria. Na tabela de **Produtos**, a coluna de `id_subcategoria` vai ser a **Chave Estrangeira** e vai permitir a gente relacionar os valores dessa coluna com a Chave Primária da tabela de Categoria.



**Categoria**

<code>id_subcategoria</code>	<code>nome_subcategoria</code>
1	Celular
2	Notebook
3	Câmera Digital
4	Televisão
5	Fone de Ouvido

**Chave Primária**

**Produto**

<code>id_produto</code>	<code>nome_produto</code>	<code>id_subcategoria</code>
1	Fone Bluetooth JBL	5
2	PS4	6
3	Notebook Samsung i3	2
4	iPhone 11	1
5	Moto G9	1

**Chave Primária**

**Chave Estrangeira**

# Tabela Fato vs. Tabela Dimensão

Você provavelmente deve ter notado em nossas tabelas do banco de dados Contoso que os nomes das tabelas possuem um certo padrão: alguns nomes começam com a inicial **Dim** e outros com a inicial **Fact**. Chegou a hora de entender essa nomenclatura.

Vamos analisar as tabelas **DimChannel** e **FactSales** do nosso banco de dados Contoso, conforme mostrado no print ao lado.

A tabela **DimChannel** resume os 4 canais de venda da empresa Contoso, e a tabela **FactSales** é uma tabela que contém as vendas dos produtos.

Essas duas tabelas possuem características diferentes, que vamos entender agora.

ChannelKey	ChannelName
1	Store
2	Online
3	Catalog
4	Reseller

DimChannel

SalesKey	DateKey	ChannelKey	SalesQuantity
1	2007-01-02 00:00:00.000	1	8
2	2007-02-12 00:00:00.000	4	4
3	2008-01-24 00:00:00.000	1	9
4	2008-01-13 00:00:00.000	2	8
5	2008-01-22 00:00:00.000	2	24
6	2007-07-02 00:00:00.000	3	36
7	2007-11-19 00:00:00.000	4	6
8	2008-04-10 00:00:00.000	2	9
9	2008-07-14 00:00:00.000	2	24
10	2009-04-10 00:00:00.000	4	18

FactSales

[+]	dbo.DimAccount
[+]	dbo.DimChannel
[+]	dbo.DimCurrency
[+]	dbo.DimCustomer
[+]	dbo.DimDate
[+]	dbo.DimEmployee
[+]	dbo.DimEntity
[+]	dbo.DimGeography
[+]	dbo.DimMachine
[+]	dbo.DimOutage
[+]	dbo.DimProduct
[+]	dbo.DimProductCategory
[+]	dbo.DimProductSubcategory
[+]	dbo.DimPromotion
[+]	dbo.DimSalesTerritory
[+]	dbo.DimScenario
[+]	dbo.DimStore
[+]	dbo.FactExchangeRate
[+]	dbo.FactInventory
[+]	dbo.FactITMachine
[+]	dbo.FactITSLA
[+]	dbo.FactOnlineSales
[+]	dbo.FactSales
[+]	dbo.FactSalesQuota
[+]	dbo.FactStrategyPlan

# Tabela Fato vs. Tabela Dimensão

Uma **Tabela Dimensão** é uma tabela que contém características de um determinado elemento: lojas, produtos, funcionários, clientes, etc.

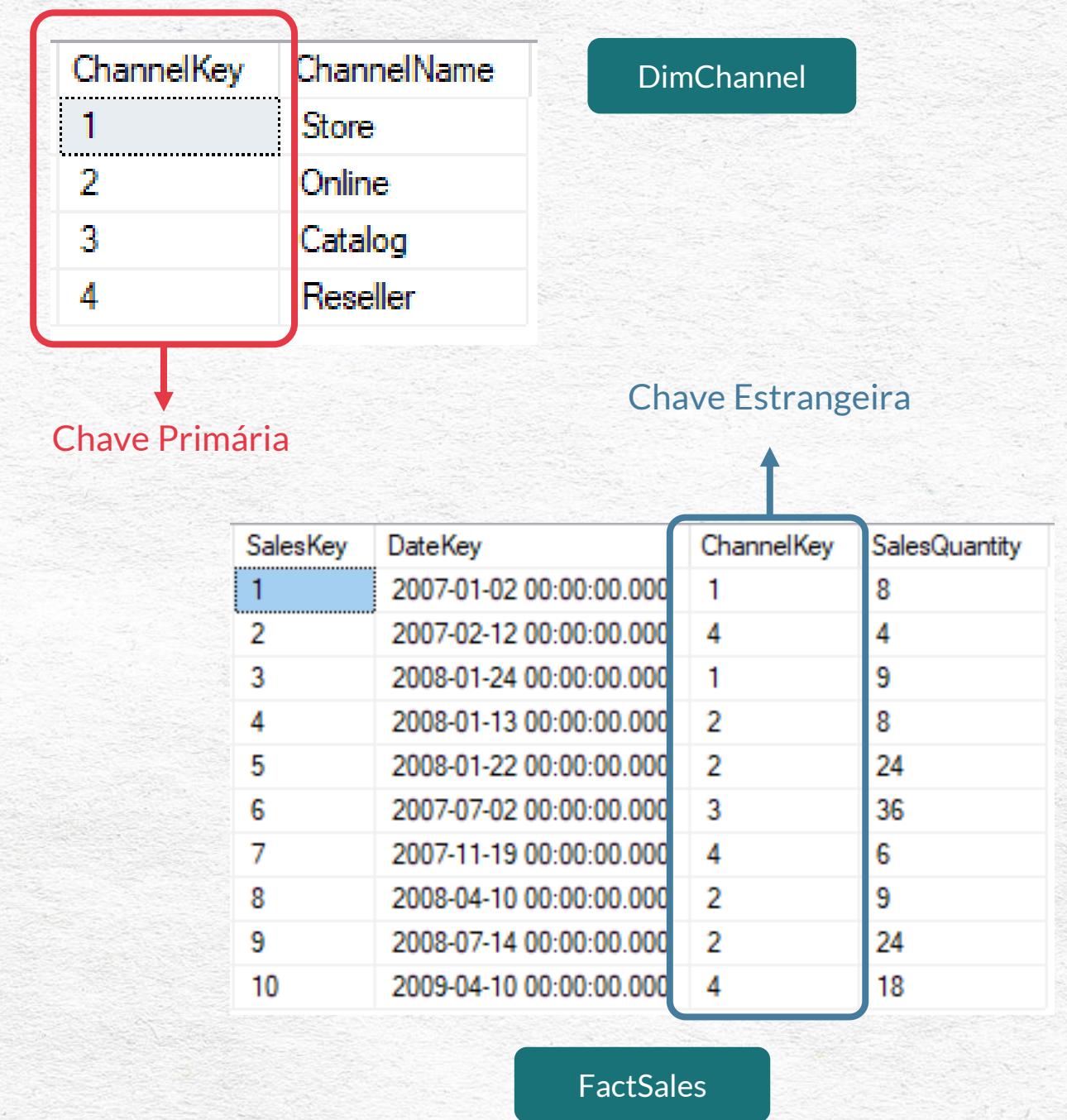
Nesta tabela, nenhum dos elementos principais irá se repetir. É onde vamos encontrar nossas **chaves primárias**.

Uma **Tabela Fato** é uma tabela que vai registrar os fatos ou acontecimentos de uma empresa/negócio em determinados períodos de tempo (vendas, devoluções, aberturas de chamados, receitas, despesas, etc).

Geralmente é uma tabela com milhares de informações e composta essencialmente por colunas de ID usadas para buscar as informações complementares de uma tabela dimensão, conhecidas como **chaves estrangeiras**.

No exemplo ao lado, a FactSales é a nossa tabela Fato e a DimChannel é a nossa tabela Dimensão.

Agora faz sentido o porquê dessas iniciais ‘Fact’ e ‘Dim’ no nome das tabelas?



- + dbo.DimAccount
- + dbo.DimChannel
- + dbo.DimCurrency
- + dbo.DimCustomer
- + dbo.DimDate
- + dbo.DimEmployee
- + dbo.DimEntity
- + dbo.DimGeography
- + dbo.DimMachine
- + dbo.DimOutage
- + dbo.DimProduct
- + dbo.DimProductCategory
- + dbo.DimProductSubcategory
- + dbo.DimPromotion
- + dbo.DimSalesTerritory
- + dbo.DimScenario
- + dbo.DimStore
- + dbo.FactExchangeRate
- + dbo.FactInventory
- + dbo.FactITMachine
- + dbo.FactITSLA
- + dbo.FactOnlineSales
- + dbo.FactSales
- + dbo.FactSalesQuota
- + dbo.FactStrategyPlan

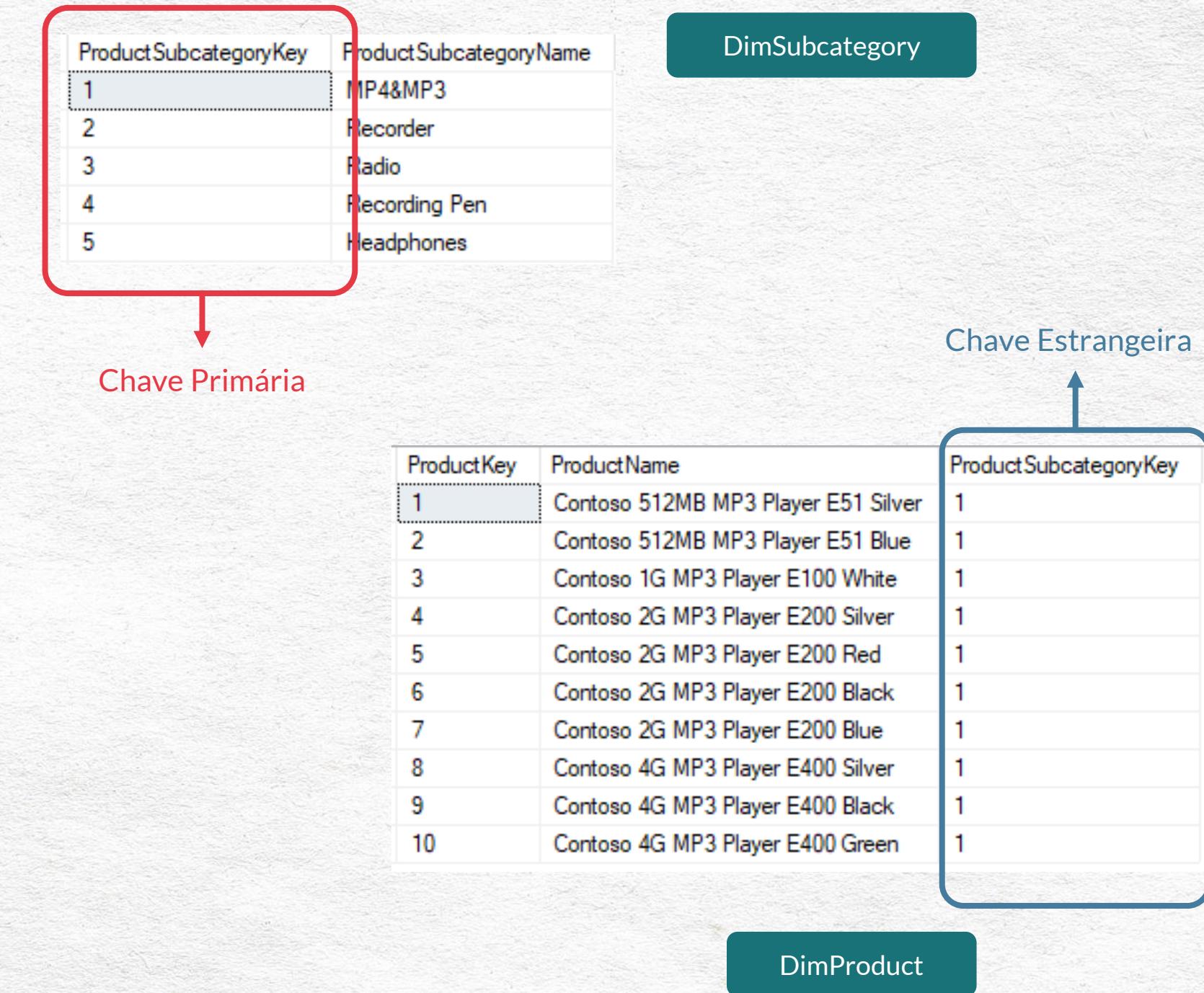
# Tabela Fato vs. Tabela Dimensão



Não necessariamente uma relação acontece entre uma fato e uma dimensão.

Duas dimensões também podem se relacionar, como é o caso do exemplo ao lado.

O que não fazemos é uma relação entre duas fatos.



Nas aulas anteriores vimos que existem dois tipos de tabelas: Dimensão e Fato. Essas tabelas podem ser relacionadas através de uma coluna: na tabela Dimensão, identificamos a chave Primária, que será relacionada com a chave Estrangeira da tabela Fato.

Mas pra que servem essas relações?

Essas relações serão criadas por meio do que chamamos de JOIN's. A tradução literal dessa palavra é “juntar”, “unir”. Os JOINs vão nos permitir fazer exatamente isso: juntar as nossas tabelas Fato e Dimensão, de forma a complementar as informações umas das outras.

Existem diversos tipos de Joins no SQL, os quais estão listados ao lado.

LEFT OUTER JOIN

RIGHT OUTER JOIN

INNER JOIN

FULL OUTER JOIN

LEFT ANTI JOIN

RIGHT ANTI JOIN

FULL ANTI JOIN

Veremos mais a frente que os Joins mais utilizados serão o LEFT JOIN e o INNER JOIN. Porém, vamos aprender na prática como todos eles funcionam.

# LEFT (OUTER) JOIN

LEFT OUTER JOIN



RIGHT OUTER JOIN



INNER JOIN



FULL OUTER JOIN



LEFT ANTI JOIN



RIGHT ANTI JOIN



FULL ANTI JOIN



Tabela A

id_produto	nome_produto	id_subcategoria
1	Fone Bluetooth JBL	5
2	PS4	6
3	Notebook Samsung i3	2
4	iPhone 11	1
5	Moto G9	1

Todos da Tabela A + Interseção com a Tabela B

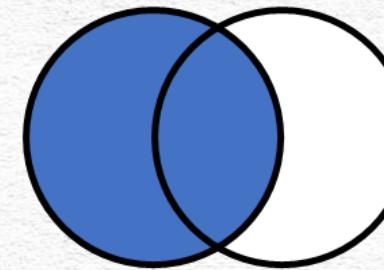


Tabela A

Tabela B

Tabela B

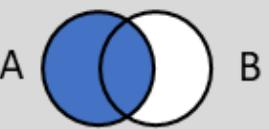
id_subcategoria	nome_subcategoria
1	Celular
2	Notebook
3	Câmera Digital
4	Televisão
5	Fone de Ouvido

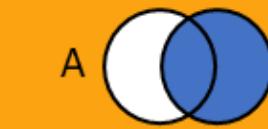
O LEFT JOIN vai nos permitir relacionar duas tabelas (Tabela A e Tabela B) e criar uma nova tabela (Tabela C) que é a junção das duas.

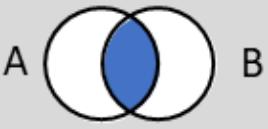
A tabela resultante desse JOIN terá todas as linhas exclusivas apenas da tabela A, mais as linhas que são a interseção entre a Tabela A e a Tabela B.

Nenhuma linha presente apenas na Tabela B será incluída na tabela resultante desse JOIN.

# RIGHT (OUTER) JOIN

LEFT OUTER JOIN  
A  B

RIGHT OUTER JOIN  
A  B

INNER JOIN  
A  B

FULL OUTER JOIN  
A  B

LEFT ANTI JOIN  
A  B

RIGHT ANTI JOIN  
A  B

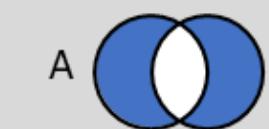
FULL ANTI JOIN  
A  B

Tabela A

id_produto	nome_produto	id_subcategoria
1	Fone Bluetooth JBL	5
2	PS4	6
3	Notebook Samsung i3	2
4	iPhone 11	1
5	Moto G9	1

Todos da Tabela B + Interseção com a Tabela A

Tabela A

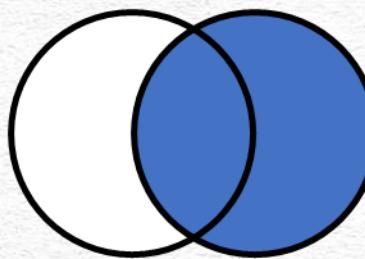


Tabela B

id_subcategoria	nome_subcategoria
1	Celular
2	Notebook
3	Câmera Digital
4	Televisão
5	Fone de Ouvido

Tabela C

id_produto	nome_produto	id_subcategoria	nome_subcategoria
4	iPhone 11	1	Celular
5	Moto G9	1	Celular
3	Notebook Samsung i3	2	Notebook
NULL	NULL	NULL	Câmera Digital
NULL	NULL	NULL	Televisão
1	Fone Bluetooth JBL	5	Fone de Ouvido

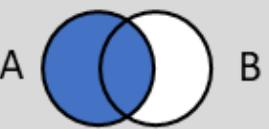
O RIGHT JOIN vai nos permitir relacionar duas tabelas (Tabela A e Tabela B) e criar uma nova tabela (Tabela C) que é a junção das duas.

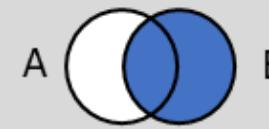
A tabela resultante desse JOIN terá todas as linhas exclusivas apenas da tabela B, mais as linhas que são a interseção entre a Tabela A e a Tabela B.

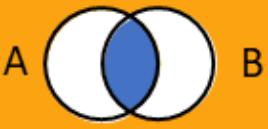
Nenhuma linha presente apenas na Tabela A será incluída na tabela resultante desse JOIN.

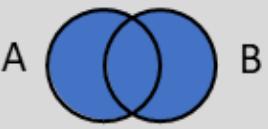


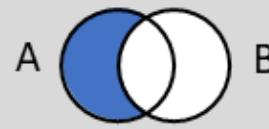
# INNER (OUTER) JOIN

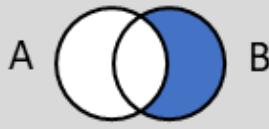
LEFT OUTER JOIN  
A  B

RIGHT OUTER JOIN  
A  B

INNER JOIN  
A  B

FULL OUTER JOIN  
A  B

LEFT ANTI JOIN  
A  B

RIGHT ANTI JOIN  
A  B

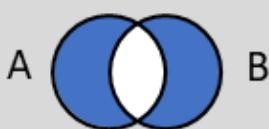
FULL ANTI JOIN  
A  B

Tabela A

id_produto	nome_produto	id_subcategoria
1	Fone Bluetooth JBL	5
2	PS4	6
3	Notebook Samsung i3	2
4	iPhone 11	1
5	Moto G9	1

Apenas a interseção entre Tabelas A e B

Tabela A

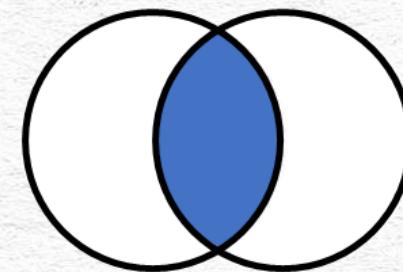


Tabela B

Tabela B

id_subcategoria	nome_subcategoria
1	Celular
2	Notebook
3	Câmera Digital
4	Televisão
5	Fone de Ouvido

Tabela C

id_produto	nome_produto	id_subcategoria	nome_subcategoria
1	Fone Bluetooth JBL	5	Fone de Ouvido
3	Notebook Samsung i3	2	Notebook
4	iPhone 11	1	Celular
5	Moto G9	1	Celular

O INNER JOIN vai nos permitir relacionar duas tabelas (Tabela A e Tabela B) e criar uma nova tabela (Tabela C) que é a junção das duas.

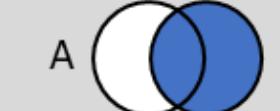
A tabela resultante desse JOIN terá **apenas** as linhas que são a interseção entre a Tabela A e a Tabela B.

# FULL (OUTER) JOIN

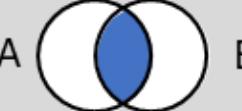
LEFT OUTER JOIN



RIGHT OUTER JOIN



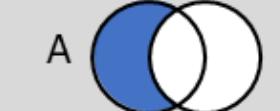
INNER JOIN



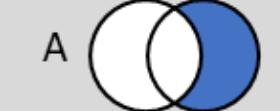
FULL OUTER JOIN



LEFT ANTI JOIN



RIGHT ANTI JOIN



FULL ANTI JOIN



Tabela A

id_produto	nome_produto	id_subcategoria
1	Fone Bluetooth JBL	5
2	PS4	6
3	Notebook Samsung i3	2
4	iPhone 11	1
5	Moto G9	1

Todas as linhas da Tabela A + Todas as linhas da Tabela B

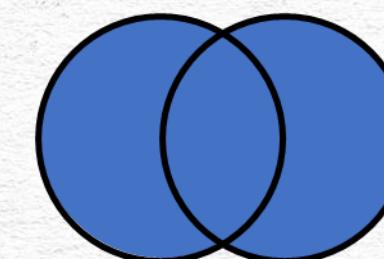


Tabela A

Tabela B

Tabela B

id_subcategoria	nome_subcategoria
1	Celular
2	Notebook
3	Câmera Digital
4	Televisão
5	Fone de Ouvido

Tabela C

id_produto	nome_produto	id_subcategoria	nome_subcategoria
1	Fone Bluetooth JBL	5	Fone de Ouvido
2	PS4	6	NULL
3	Notebook Samsung i3	2	Notebook
4	iPhone 11	1	Celular
5	Moto G9	1	Celular
NULL	NULL	NULL	Câmera Digital
NULL	NULL	NULL	Televisão

O FULL JOIN vai nos permitir relacionar duas tabelas (Tabela A e Tabela B) e criar uma nova tabela (Tabela C) que é a junção das duas.

A tabela resultante desse JOIN terá todas as linhas da Tabela A e todas as linhas da Tabela B.

# LEFT (ANTI) JOIN

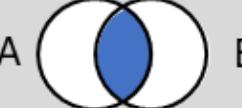
LEFT OUTER JOIN



RIGHT OUTER JOIN



INNER JOIN



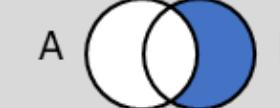
FULL OUTER JOIN



LEFT ANTI JOIN



RIGHT ANTI JOIN



FULL ANTI JOIN



Tabela A

id_produto	nome_produto	id_subcategoria
1	Fone Bluetooth JBL	5
2	PS4	6
3	Notebook Samsung i3	2
4	iPhone 11	1
5	Moto G9	1

Apenas as linhas que aparecem em A

Tabela A      Tabela B

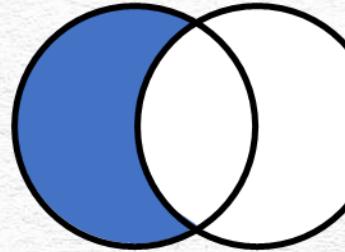


Tabela B

id_subcategoria	nome_subcategoria
1	Celular
2	Notebook
3	Câmera Digital
4	Televisão
5	Fone de Ouvido

Tabela C

id_produto	nome_produto	id_subcategoria	nome_subcategoria
2	PS4	6	NULL

O LEFT (ANTI) JOIN vai nos permitir relacionar duas tabelas (Tabela A e Tabela B) e criar uma nova tabela (Tabela C) que é a junção das duas.

A tabela resultante desse JOIN terá **apenas as linhas que só existem na Tabela A**.

# RIGHT (ANTI) JOIN

LEFT OUTER JOIN



RIGHT OUTER JOIN



INNER JOIN



FULL OUTER JOIN



LEFT ANTI JOIN



RIGHT ANTI JOIN



FULL ANTI JOIN



Tabela A

<code>id_produto</code>	<code>nome_produto</code>	<code>id_subcategoria</code>
1	Fone Bluetooth JBL	5
2	PS4	6
3	Notebook Samsung i3	2
4	iPhone 11	1
5	Moto G9	1

Apenas as linhas que aparecem em B

Tabela A

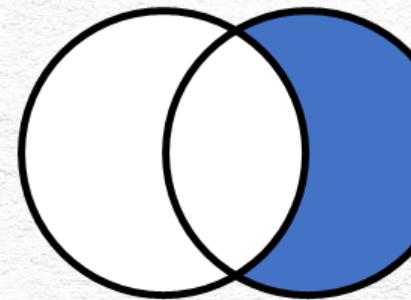


Tabela B

<code>id_subcategoria</code>	<code>nome_subcategoria</code>
1	Celular
2	Notebook
3	Câmera Digital
4	Televisão
5	Fone de Ouvido

Tabela C

<code>id_produto</code>	<code>nome_produto</code>	<code>id_subcategoria</code>	<code>nome_subcategoria</code>
NULL	NULL	NULL	Câmera Digital
NULL	NULL	NULL	Televisão

O RIGHT (ANTI) JOIN vai nos permitir relacionar duas tabelas (Tabela A e Tabela B) e criar uma nova tabela (Tabela C) que é a junção das duas.

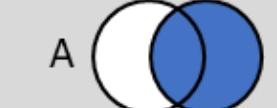
A tabela resultante desse JOIN terá **apenas as linhas que só existem na Tabela B**.

# FULL (ANTI) JOIN

LEFT OUTER JOIN



RIGHT OUTER JOIN



INNER JOIN



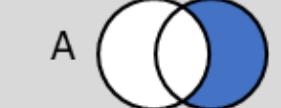
FULL OUTER JOIN



LEFT ANTI JOIN



RIGHT ANTI JOIN



FULL ANTI JOIN



Tabela A

<b>id_produto</b>	<b>nome_produto</b>	<b>id_subcategoria</b>
1	Fone Bluetooth JBL	5
2	PS4	6
3	Notebook Samsung i3	2
4	iPhone 11	1
5	Moto G9	1

Apenas as linhas que aparecem em A + Apenas as linhas que aparecem em B

Tabela A

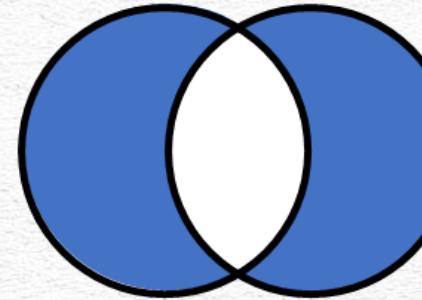


Tabela B

<b>id_subcategoria</b>	<b>nome_subcategoria</b>
1	Celular
2	Notebook
3	Câmera Digital
4	Televisão
5	Fone de Ouvido

Tabela C

<b>id_produto</b>	<b>nome_produto</b>	<b>id_subcategoria</b>	<b>nome_subcategoria</b>
2	PS4	6	NULL
NULL	NULL	NULL	Câmera Digital
NULL	NULL	NULL	Televisão

O FULL (ANTI) JOIN vai nos permitir relacionar duas tabelas (Tabela A e Tabela B) e criar uma nova tabela (Tabela C) que é a junção das duas.

A tabela resultante desse JOIN terá **apenas as linhas que só existem na Tabela B ou na Tabela A**.

# INNER, LEFT e RIGHT JOIN: exemplos práticos

Para fazer os exemplos a seguir, utilize os códigos a seguir para criar as tabelas de exemplo. **Cada código deve ser executado em uma Consulta diferente.**

Crie uma Nova Consulta e utilize o primeiro código abaixo. Nele, vamos apenas criar um novo Banco de Dados para armazenar nossas Tabelas de exemplo (Obs: entraremos mais a fundo na parte de criação de tabelas e bancos de dados no módulo CRUD).

The screenshot shows the Microsoft SQL Server Management Studio interface. A new query window titled "Nova Consulta" is open. The window contains the following SQL code:

```
CREATE DATABASE Joins --Cria um banco de dados novo
GO
USE Joins --Seleciona o banco de dados onde vamos adicionar as tabelas
```

The title bar of the window shows three tabs: "SQLQuery1.sql - LA...5)) - não conectado\*" (selected), "SQLQuery2.sql - LAP...us Cavalcanti (54))\*", and "SQLQuery3.sql - LAP...us Cavalcanti (68))\*".

# INNER, LEFT e RIGHT JOIN: exemplos práticos

Na consulta 2, utilize o código abaixo para criar a tabela de **Produtos**, com 5 produtos de exemplo e as seguintes informações: **id\_produto**, **nome\_produto** e **id\_subcategoria**. (Obs: entraremos mais a fundo na parte de criação de tabelas e bancos de dados no módulo CRUD).

```
USE Joins

CREATE TABLE produtos(
    id_produto int,
    nome_produto varchar(30),
    id_subcategoria int)

INSERT INTO produtos(id_produto, nome_produto, id_subcategoria)
VALUES
    (1, 'Fone Bluetooth JBL', 5),
    (2, 'PS4', 6),
    (3, 'Notebook Samsung i3', 2),
    (4, 'iPhone 11', 1),
    (5, 'Moto G9', 1)
```

# INNER, LEFT e RIGHT JOIN: exemplos práticos

Na consulta 3, utilize o código abaixo para criar a tabela de **Subcategoria**, com 5 subcategorias de exemplo. Essa tabela terá as seguintes informações: **id\_subcategoria** e **nome\_subcategoria**. (Obs: entraremos mais a fundo na parte de criação de tabelas e bancos de dados no módulo CRUD).

Consulta 3

```
SQLQuery1.sql - LAP...us Cavalcanti (55)*          SQLQuery2.sql - LAP...us Cavalcanti (54)*          SQLQuery3.sql - LAP...us Cavalcanti (68))*  X
USE Joins    --Seleciona o banco de dados onde vamos adicionar as tabelas

CREATE TABLE subcategoria(
    id_subcategoria int,
    nome_subcategoria varchar(30))

INSERT INTO subcategoria(id_subcategoria, nome_subcategoria)
VALUES
    (1, 'Celular'),
    (2, 'Notebook'),
    (3, 'Câmera Digital'),
    (4, 'Televisão'),
    (5, 'Fone de Ouvido')
```

# LEFT, RIGHT e INNER JOIN: exemplos práticos

Na consulta ao lado, realizamos um **LEFT JOIN**.

Utilizamos um SELECT comum para selecionar as colunas desejadas. Um detalhe importante é que quando selecionamos uma coluna que é comum às duas tabelas (no caso, a coluna `id_subcategoria`) precisamos especificar qual é a tabela que queremos considerar.

Em relação à estrutura do JOIN, temos sempre o padrão abaixo: uma igualdade, onde do lado esquerdo precisamos informar a chave estrangeira da relação, e do lado direito precisamos informar a chave primária da relação entre as tabelas.

```
FROM
    produtos
LEFT JOIN subcategoria
    ON produtos.id_subcategoria = subcategoria.id_subcategoria
        chave estrangeira      chave primária
```

Por fim, repare que a tabela resultante contém um produto que existe apenas na tabela `produto`, o PS4.

The screenshot shows a SQL Server Management Studio window with three tabs: SQLQuery6.sql, SQLQuery1.sql, and SQLQuery2.sql. The SQLQuery6.sql tab contains a query for a 'LEFT JOIN' example:

```
-- Exemplo: LEFT JOIN
SELECT
    id_produto,
    nome_produto,
    produtos.id_subcategoria,
    nome_subcategoria
FROM
    produtos
LEFT JOIN subcategoria
    ON produtos.id_subcategoria = subcategoria.id_subcategoria
```

The results tab displays the following data:

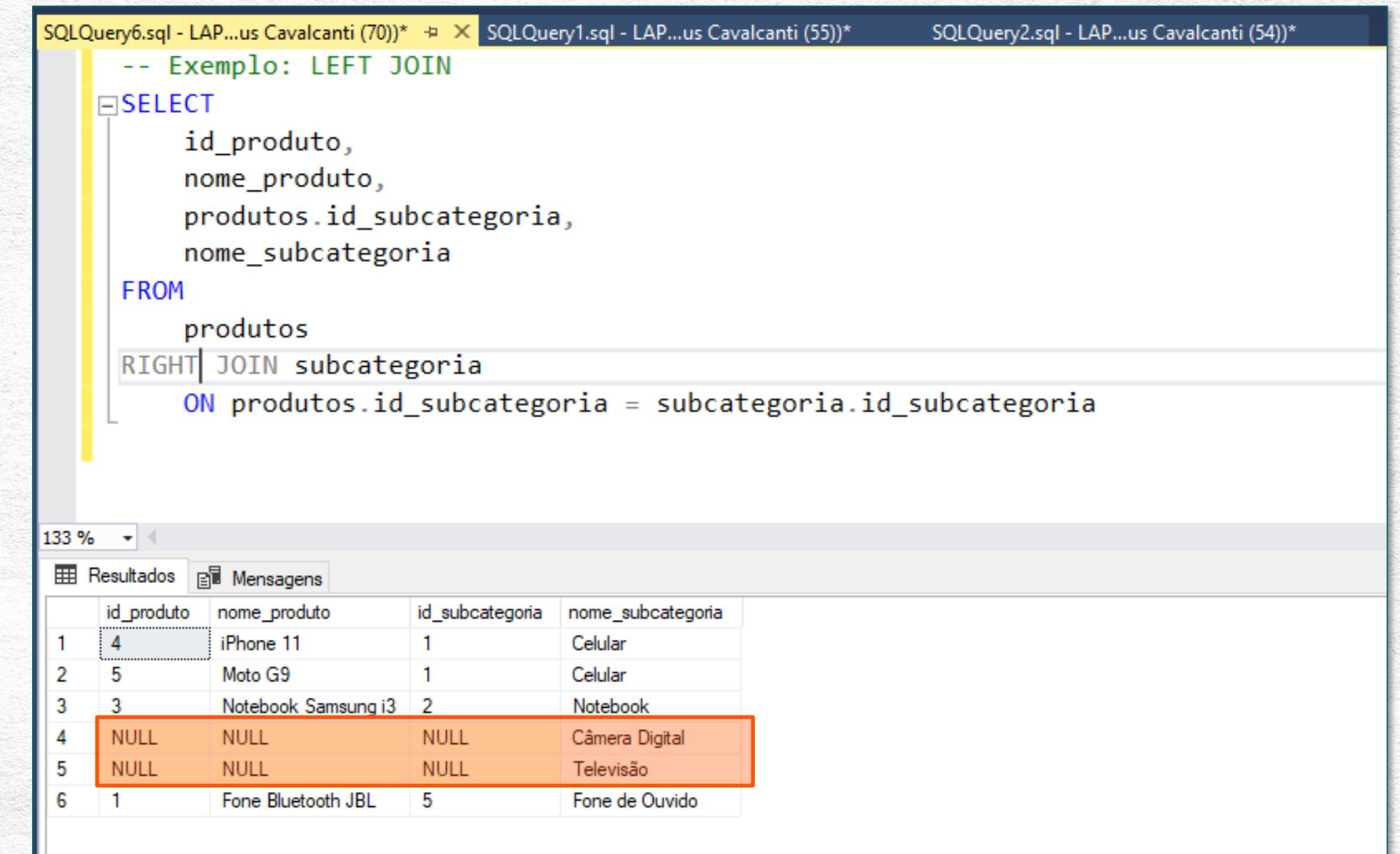
	id_produto	nome_produto	id_subcategoria	nome_subcategoria
1	1	Fone Bluetooth JBL	5	Fone de Ouvido
2	2	PS4	6	NULL
3	3	Notebook Samsung i3	2	Notebook
4	4	iPhone 11	1	Celular
5	5	Moto G9	1	Celular

# LEFT, RIGHT e INNER JOIN: exemplos práticos

Na consulta ao lado, realizamos um **RIGHT JOIN**.

A estrutura é muito semelhante ao LEFT JOIN (apenas trocamos o LEFT por RIGHT).

O resultado final é uma tabela contendo também as subcategorias que existem apenas na tabela da direita (Câmera Digital e Televisão).



The screenshot shows a SQL Server Management Studio window with three tabs at the top: 'SQLQuery6.sql - LAP...us Cavalcanti (70)\*' (selected), 'SQLQuery1.sql - LAP...us Cavalcanti (55)\*', and 'SQLQuery2.sql - LAP...us Cavalcanti (54)\*'. The main area contains a SQL query:

```
-- Exemplo: LEFT JOIN
SELECT
    id_produto,
    nome_produto,
    produtos.id_subcategoria,
    nome_subcategoria
FROM
    produtos
RIGHT JOIN subcategoria
    ON produtos.id_subcategoria = subcategoria.id_subcategoria
```

The results pane shows a table with the following data:

	id_produto	nome_produto	id_subcategoria	nome_subcategoria
1	4	iPhone 11	1	Celular
2	5	Moto G9	1	Celular
3	3	Notebook Samsung i3	2	Notebook
4	NULL	NULL	NULL	Câmera Digital
5	NULL	NULL	NULL	Televisão
6	1	Fone Bluetooth JBL	5	Fone de Ouvido

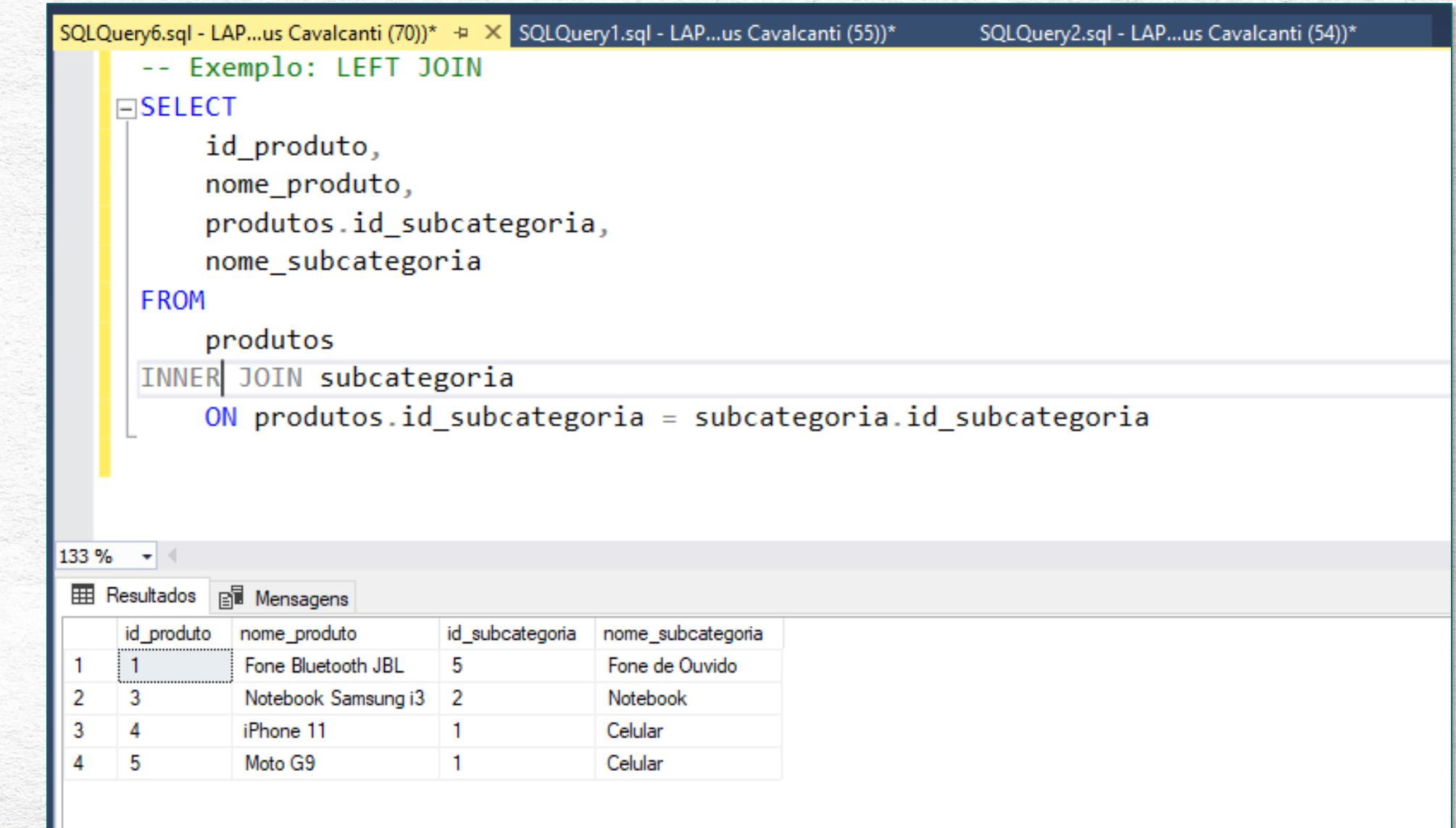
# LEFT, RIGHT e INNER JOIN: exemplos práticos

Por fim, realizamos um **INNER JOIN**.

A estrutura é muito semelhante ao LEFT/RIGHT JOIN (apenas trocamos o LEFT/RIGHT por INNER).

O resultado final é uma tabela contendo apenas as informações presentes nas duas tabelas, ou seja, a interseção.

Nessa tabela resultante não teremos nem o produto PS4 (que existe apenas na tabela **produtos**, nem as subcategorias Câmera Digital e Televisão da tabela **subcategoria**).



```
-- Exemplo: LEFT JOIN
SELECT
    id_produto,
    nome_produto,
    produtos.id_subcategoria,
    nome_subcategoria
FROM
    produtos
INNER JOIN subcategoria
    ON produtos.id_subcategoria = subcategoria.id_subcategoria
```

The screenshot shows a SQL query in the 'SQLQuery6.sql' window of SSMS. The query performs an INNER JOIN between the 'produtos' and 'subcategoria' tables based on their 'id\_subcategoria' columns. The results are displayed in a 'Resultados' tab, showing four rows of data:

	id_produto	nome_produto	id_subcategoria	nome_subcategoria
1	1	Fone Bluetooth JBL	5	Fone de Ouvido
2	3	Notebook Samsung i3	2	Notebook
3	4	iPhone 11	1	Celular
4	5	Moto G9	1	Celular

# FULL JOIN: exemplos práticos

Ao lado, realizamos um **FULL JOIN**.

Esse JOIN retornará **todas as linhas das duas tabelas**: tanto a linha do produto PS4 (que existe apenas na tabela produto) quanto as subcategorias Câmera Digital e Televisão (que existem apenas na tabela subcategoria).

Essa seria uma junção completa das duas tabelas.

The screenshot shows a SQL Server Management Studio window with three tabs at the top: 'SQLQuery6.sql - LAP...us Cavalcanti (70)\*' (selected), 'SQLQuery1.sql - LAP...us Cavalcanti (55)\*', and 'SQLQuery2.sql - LAP...us Cavalcanti (54)\*'. The main area displays a SQL query:

```
-- Exemplo: LEFT JOIN
SELECT
    id_produto,
    nome_produto,
    produtos.id_subcategoria,
    nome_subcategoria
FROM
    produtos
FULL JOIN subcategoria
    ON produtos.id_subcategoria = subcategoria.id_subcategoria
```

Below the query is a results grid titled 'Resultados' (Results). It contains the following data:

	id_produto	nome_produto	id_subcategoria	nome_subcategoria
1	1	Fone Bluetooth JBL	5	Fone de Ouvido
2	2	PS4	6	NULL
3	3	Notebook Samsung i3	2	Notebook
4	4	iPhone 11	1	Celular
5	5	Moto G9	1	Celular
6	NULL	NULL	NULL	Câmera Digital
7	NULL	NULL	NULL	Televisão

# LEFT, RIGHT e FULL ANTI JOIN: exemplos práticos

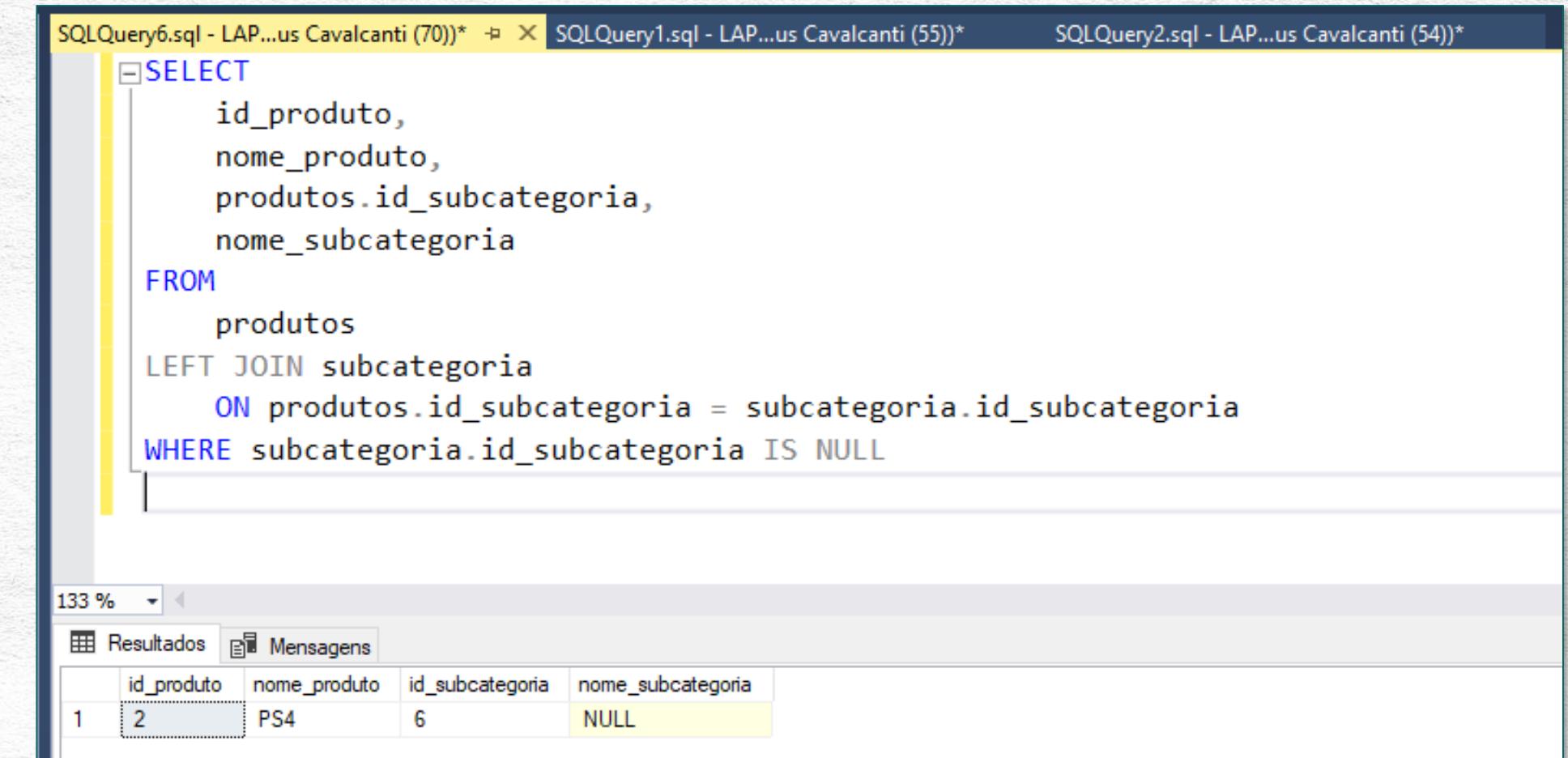
Outro conjunto de JOINs são os chamados ANTI JOIN: LEFT ANTI JOIN, RIGTH ANTI JOIN e FULL ANTI JOIN.

Eles não são tão utilizados na prática, mas é importante que você saiba o que cada um faz.

Ao lado, realizamos um **LEFT ANTI JOIN**.

Este JOIN vai retornar os produtos que existem apenas na tabela da esquerda (**produtos**): no caso, o produto PS4.

Observe que não existe um comando chamado LEFT ANTI JOIN. Para criá-lo, precisamos combinar o LEFT JOIN com um filtro WHERE, onde consideramos todas as subcategorias (tabela da direita) que são nulos.



The screenshot shows a SQL query window in SSMS. The query is:

```

SELECT
    id_produto,
    nome_produto,
    produtos.id_subcategoria,
    nome_subcategoria
FROM
    produtos
LEFT JOIN subcategoria
    ON produtos.id_subcategoria = subcategoria.id_subcategoria
WHERE subcategoria.id_subcategoria IS NULL
  
```

The results pane shows a single row of data:

	id_produto	nome_produto	id_subcategoria	nome_subcategoria
1	2	PS4	6	NULL

# LEFT, RIGHT e FULL ANTI JOIN: exemplos práticos

Ao lado, realizamos um **RIGHT ANTI JOIN**.

Ele terá um comportamento semelhante ao LEFT ANTI JOIN, com a diferença que a tabela resultante terá as linhas que estão presentes apenas na tabela da direita (tabela subcategoria).

The screenshot shows a SQL Server Management Studio window with three tabs at the top: 'SQLQuery6.sql - LAP...us Cavalcanti (70)\*' (selected), 'SQLQuery1.sql - LAP...us Cavalcanti (55)\*', and 'SQLQuery2.sql - LAP...us Cavalcanti (54)\*'. The main area displays the following SQL code:

```
SELECT
    id_produto,
    nome_produto,
    produtos.id_subcategoria,
    nome_subcategoria
FROM
    produtos
RIGHT JOIN subcategoria
    ON produtos.id_subcategoria = subcategoria.id_subcategoria
WHERE produtos.id_subcategoria IS NULL
```

Below the code, the 'Resultados' tab is selected, showing the following table:

	id_produto	nome_produto	id_subcategoria	nome_subcategoria
1	NULL	NULL	NULL	Câmera Digital
2	NULL	NULL	NULL	Televisão

# LEFT, RIGHT e FULL ANTI JOIN: exemplos práticos

Por fim, realizamos um **FULL ANTI JOIN**.

Este JOIN retornará todas as linhas que estão presentes em apenas uma das tabelas, como é mostrado na imagem ao lado.

The screenshot shows a SQL Server Management Studio window with three tabs at the top: 'SQLQuery6.sql - LAP...us Cavalcanti (70)\*' (selected), 'SQLQuery1.sql - LAP...us Cavalcanti (55)\*', and 'SQLQuery2.sql - LAP...us Cavalcanti (54)\*'. The query in the selected tab is:

```
SELECT
    id_produto,
    nome_produto,
    produtos.id_subcategoria,
    nome_subcategoria
FROM
    produtos
FULL JOIN subcategoria
    ON produtos.id_subcategoria = subcategoria.id_subcategoria
WHERE produtos.id_subcategoria IS NULL OR subcategoria.id_subcategoria IS NULL
```

The results are displayed in a table titled 'Resultados' with columns: id\_produto, nome\_produto, id\_subcategoria, and nome\_subcategoria. The data is:

	id_produto	nome_produto	id_subcategoria	nome_subcategoria
1	2	PS4	6	NULL
2	NULL	NULL	NULL	Câmera Digital
3	NULL	NULL	NULL	Televisão

# Como definir quem é a LEFT TABLE e a RIGHT TABLE

Uma dúvida comum é se existe alguma diferença em escolher quem será a LEFT TABLE ou a RIGHT TABLE.

Observe o exercício ao lado. Utilizamos o LEFT JOIN para retornar a junção entre as tabelas **produtos** e **subcategoria**.

Seria possível chegar no mesmo resultado utilizando o RIGHT JOIN, como veremos na página a seguir.

The screenshot shows a SQL Server Management Studio window with three tabs at the top: 'SQLQuery6.sql - LAP...us Cavalcanti (70)\*' (selected), 'SQLQuery1.sql - LAP...us Cavalcanti (55)\*', and 'SQLQuery2.sql - LAP...us Cavalcanti (54)\*'. The main area displays a SQL query:

```
-- 1. LEFT Join para complementar informações da tabela produtos
-- (LEFT) de acordo com a tabela de categoria (RIGHT)
SELECT
    id_produto,
    nome_produto,
    produtos.id_subcategoria,
    nome_subcategoria
FROM
    produtos
LEFT JOIN subcategoria
    ON produtos.id_subcategoria = subcategoria.id_subcategoria
```

The results pane shows a table with five rows:

	id_produto	nome_produto	id_subcategoria	nome_subcategoria
1	1	Fone Bluetooth JBL	5	Fone de Ouvido
2	2	PS4	6	NULL
3	3	Notebook Samsung i3	2	Notebook
4	4	iPhone 11	1	Celular
5	5	Moto G9	1	Celular

# Como definir quem é a LEFT TABLE e a RIGHT TABLE

Observe que conseguimos obter a mesma tabela utilizando também o RIGHT JOIN.

Para isso, precisamos fazer algumas alterações: a tabela de produtos, que antes era a tabela da esquerda, virou a tabela da direita.

Já a tabela de subcategoria, que antes era a tabela da direita, virou a tabela da esquerda.

Com essas alterações, conseguimos chegar no mesmo resultado, utilizando o RIGHT JOIN.

O que vale a gente pensar no momento é que esse join tinha como finalidade complementar as informações de subcategoria em nossa tabela de produtos. É mais intuitivo pensar que queremos complementar a tabela da esquerda com as informações da tabela da direita. E para chegar nesse resultado, usamos o LEFT JOIN.

O RIGHT JOIN nos leva a uma lógica um pouco mais confusa, por isso preferimos utilizar o LEFT JOIN, por ser mais intuitivo.

A lição que fica é: podemos chegar no mesmo resultado usando tanto o LEFT JOIN quanto o RIGHT JOIN, mas o mais comum de ser utilizado é o LEFT JOIN.

```

SQLQuery6.sql - LAP...us Cavalcanti (70)*  SQLQuery1.sql - LAP...us Cavalcanti (55)*      SQLQuery2.sql - LAP...us Cavalcanti (54)*
-- 2. Obtendo o LEFT Join do caso 1 usando um RIGHT Join e invertendo as tabelas de lado

SELECT
    id_produto,
    nome_produto,
    produtos.id_subcategoria,
    nome_subcategoria
FROM
    subcategoria
RIGHT JOIN produtos
    ON subcategoria.id_subcategoria = produtos.id_subcategoria
  
```

	id_produto	nome_produto	id_subcategoria	nome_subcategoria
1	1	Fone Bluetooth JBL	5	Fone de Ouvido
2	2	PS4	6	NULL
3	3	Notebook Samsung i3	2	Notebook
4	4	iPhone 11	1	Celular
5	5	Moto G9	1	Celular

# LEFT JOIN ou INNER JOIN: qual usar?

Em alguns casos, o LEFT JOIN e o INNER JOIN podem trazer o mesmo resultado.

Caso não haja nenhuma informação na tabela A que não esteja presente na tabela B, os resultados serão iguais. Para ficar claro, pense nas seguintes situações:

## Situação 1

Se a Tabela A (esquerda) possui os produtos 1 a 10, e na Tabela B (direita) também temos os produtos de 1 a 10, se fizermos um INNER JOIN ou um LEFT JOIN, teremos o mesmo resultado, pois não há em nenhuma das tabelas um produto que exista apenas em uma dessas tabelas.

## Situação 2

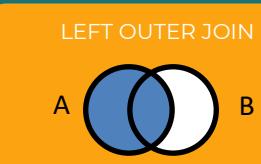
Se a Tabela A (esquerda) possui os produtos de 1 a 11, e na Tabela B (direita) temos apenas os produtos de 1 a 10, repare que há 1 produto na Tabela A que só existe na tabela A. Portanto, se usamos o LEFT JOIN, a tabela resultante terá 1 produto a mais (o que existe apenas na tabela da esquerda) enquanto o INNER JOIN vai retornar apenas os produtos que estão presentes nas duas tabelas (e deixará 1 produto da Tabela A de fora).

Tabela A

id_produto	nome_produto	id_subcategoria
1	Fone Bluetooth JBL	5
2	PS4	6
3	Notebook Samsung i3	2
4	iPhone 11	1
5	Moto G9	1

Tabela B

id_subcategoria	nome_subcategoria
1	Celular
2	Notebook
3	Câmera Digital
4	Televisão
5	Fone de Ouvido



id_produto	nome_produto	id_subcategoria	nome_subcategoria
1	Fone Bluetooth JBL	5	Fone de Ouvido
2	PS4	6	NULL
3	Notebook Samsung i3	2	Notebook
4	iPhone 11	1	Celular
5	Moto G9	1	Celular



id_produto	nome_produto	id_subcategoria	nome_subcategoria
1	Fone Bluetooth JBL	5	Fone de Ouvido
3	Notebook Samsung i3	2	Notebook
4	iPhone 11	1	Celular
5	Moto G9	1	Celular

### LEFT JOIN

No exemplo acima, a tabela resultante terá também o produto PS4, que está presente apenas na tabela A (esquerda).

### INNER JOIN

No exemplo acima, a tabela resultante terá apenas os produtos que estão presentes nas duas tabelas, excluindo o PS4, que está presente apenas na Tabela A.

# CROSS JOIN

Um outro JOIN que pode ser útil é o **CROSS JOIN**.

Com este JOIN, conseguimos criar uma tabela que é a combinação de todos os valores entre duas colunas.

Observe a imagem ao lado. Utilizando o CROSS JOIN com as colunas nome\_produto e nome\_subcategoria, é feita uma combinação de cada produto e cada subcategoria.

Essa combinação é conhecida como produto cartesiano: uma combinação um a um de todos os valores de duas ou mais colunas.

The screenshot shows a SQL query window with the following code:

```
SQLQuery6.sql - LAP...us Cavalcanti (70)*  X  SQLQuery1.sql - LAP...us Cavalc
SELECT
    nome_produto,
    nome_subcategoria
FROM
    produtos CROSS JOIN subcategoria
```

The results pane displays a table with two columns: nome\_produto and nome\_subcategoria. The data consists of 20 rows, showing every combination of a product name from the 'produtos' table and a subcategory name from the 'subcategoria' table.

	nome_produto	nome_subcategoria
1	Fone Bluetooth JBL	Celular
2	PS4	Celular
3	Notebook Samsung i3	Celular
4	iPhone 11	Celular
5	Moto G9	Celular
6	Fone Bluetooth JBL	Notebook
7	PS4	Notebook
8	Notebook Samsung i3	Notebook
9	iPhone 11	Notebook
10	Moto G9	Notebook
11	Fone Bluetooth JBL	Câmera Digital
12	PS4	Câmera Digital
13	Notebook Samsung i3	Câmera Digital
14	iPhone 11	Câmera Digital
15	Moto G9	Câmera Digital
16	Fone Bluetooth JBL	Televisão
17	PS4	Televisão
18	Notebook Samsung i3	Televisão
19	iPhone 11	Televisão
20	Moto G9	Televisão
21	Fone Bluetooth JBL	Fone de Ouvido

# Múltiplos Joins

Em algumas situações, precisamos utilizar múltiplos Joins.

Em nosso banco de dados Contoso, temos uma tabela chamada DimProduct, que contém as informações dos produtos.

Porém, essa tabela não possui informações das categorias desses produtos, apenas as informações de ids das subcategorias.

Portanto, sabemos apenas de forma indireta qual é a categoria do produto: de acordo com o id da subcategoria, conseguimos descobrir qual é a categoria.

Resultados		Mensagens						
	ProductKey	ProductLabel	ProductName	ProductDescription	ProductSubcategoryKey	Manufacturer	BrandName	Class
1	1	0101001	Contoso 512MB MP3 Player E51 Silver	512MB USB driver plays MP3 and WMA	1	Contoso, Ltd	Contoso	1
2	2	0101002	Contoso 512MB MP3 Player E51 Blue	512MB USB driver plays MP3 and WMA	1	Contoso, Ltd	Contoso	1
3	3	0101003	Contoso 1G MP3 Player E100 White	1GB flash memory and USB driver plays MP3 and WMA	1	Contoso, Ltd	Contoso	1
4	4	0101004	Contoso 2G MP3 Player E200 Silver	2GB flash memory, LCD display, plays MP3 and WMA	1	Contoso, Ltd	Contoso	1
5	5	0101005	Contoso 2G MP3 Player E200 Red	2GB flash memory, LCD display, plays MP3 and WMA	1	Contoso, Ltd	Contoso	1

	ProductSubcategoryKey	ProductSubcategoryLabel	ProductSubcategoryName	ProductSubcategoryDescription	ProductCategoryKey	ETLLoadID	LoadDate	UpdateDate
1	1	0101	MP4&MP3	MP4&MP3	1	1	2009-07-07 00:00:00.000	2009-07
2	2	0102	Recorder	Recorder	1	1	2009-07-07 00:00:00.000	2009-07
3	3	0103	Radio	Radio	1	1	2009-07-07 00:00:00.000	2009-07
4	4	0104	Recording Pen	Recording Pen	1	1	2009-07-07 00:00:00.000	2009-07
5	5	0105	Headphones	Headphones	1	1	2009-07-07 00:00:00.000	2009-07
6	6	0106	Bluetooth Headphones	Bluetooth Headphones	1	1	2009-07-07 00:00:00.000	2009-07
7	7	0107	Speakers	Speakers	1	1	2009-07-07 00:00:00.000	2009-07
8	8	0108			1	1	2009-07-07 00:00:00.000	2009-07

	ProductCategoryKey	ProductCategoryLabel	ProductCategoryName	ProductCategoryDescription	ETLLoadID	LoadDate	UpdateDate
1	1	01	Audio	Audio	1	2009-07-07 00:00:00.000	2009-07-07 00:00:00.000
2	2	02	TV and Video	TV and Video	1	2009-07-07 00:00:00.000	2009-07-07 00:00:00.000
3	3	03	Computers	Computers	1	2009-07-07 00:00:00.000	2009-07-07 00:00:00.000
4	4	04	Cameras and camco...	Cameras and camcorders	1	2009-07-07 00:00:00.000	2009-07-07 00:00:00.000
5	5	05	Cell phones	Cell phones	1	2009-07-07 00:00:00.000	2009-07-07 00:00:00.000
6	6	06	Music, Movies and A...	Music, Movies and Audio ...	1	2009-07-07 00:00:00.000	2009-07-07 00:00:00.000
7	7	07	Games and Toys	Games and Toys	1	2009-07-07 00:00:00.000	2009-07-07 00:00:00.000
8	8	08	Home Appliances	Home Appliances	1	2009-07-07 00:00:00.000	2009-07-07 00:00:00.000

# Múltiplos Joins

Para levar uma informação da tabela de categoria até a tabela de Produtos, precisaremos fazer dois joins: um para partir da tabela de produtos e chegar na tabela de subcategoria, e outro para sair da tabela de subcategoria e chegar na de categoria.

O resultado de uma consulta que reúne as informações de nome do produto (DimProduct), nome da subcategoria (DimProductSubcategory) e nome da categoria (DimProductCategory) é mostrado na imagem ao lado.

The screenshot shows a SQL Server Management Studio window with three tabs at the top: 'SQLQuery6.sql - LAP...us Cavalcanti (70)\*' (selected), 'SQLQuery1.sql - LAP...us Cavalcanti (55)\*', and 'SQLQuery2.sql - LAP...us Cavalcanti (54)\*'. The main area displays a SELECT query:

```
SELECT
    ProductKey,
    ProductName,
    DimProductSubcategory.ProductSubcategoryKey,
    ProductSubcategoryName,
    DimProductSubcategory.ProductCategoryKey,
    DimProductCategory.ProductCategoryName
FROM
    DimProduct
INNER JOIN DimProductSubcategory
    ON DimProduct.ProductSubcategoryKey = DimProductSubcategory.ProductSubcategoryKey
INNER JOIN DimProductCategory
    ON DimProductSubcategory.ProductCategoryKey = DimProductCategory.ProductCategoryKey
```

Below the query is a results grid titled 'Resultados' (Results). It contains 11 rows of data:

	ProductKey	ProductName	ProductSubcategoryKey	ProductSubcategoryName	ProductCategoryKey	ProductCategoryName
1	1	Contoso 512MB MP3 Player E51 Silver	1	MP4&MP3	1	Audio
2	2	Contoso 512MB MP3 Player E51 Blue	1	MP4&MP3	1	Audio
3	3	Contoso 1G MP3 Player E100 White	1	MP4&MP3	1	Audio
4	4	Contoso 2G MP3 Player E200 Silver	1	MP4&MP3	1	Audio
5	5	Contoso 2G MP3 Player E200 Red	1	MP4&MP3	1	Audio
6	6	Contoso 2G MP3 Player E200 Black	1	MP4&MP3	1	Audio
7	7	Contoso 2G MP3 Player E200 Blue	1	MP4&MP3	1	Audio
8	8	Contoso 4G MP3 Player E400 Silver	1	MP4&MP3	1	Audio
9	9	Contoso 4G MP3 Player E400 Black	1	MP4&MP3	1	Audio
10	10	Contoso 4G MP3 Player E400 Green	1	MP4&MP3	1	Audio
11	11	Contoso 4G MP3 Player E400 Orange	1	MP4&MP3	1	Audio

# UNION e UNION ALL

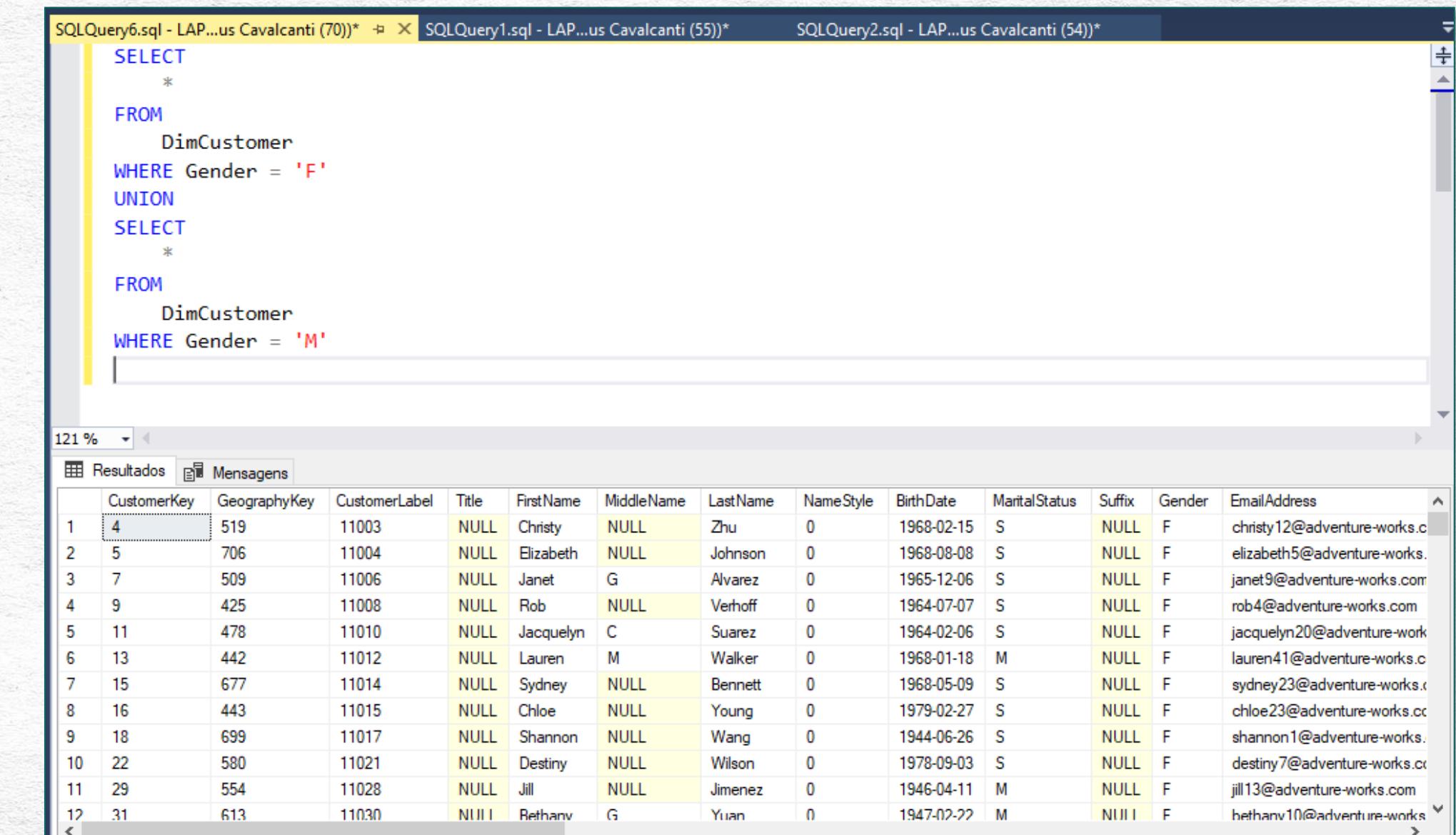
Existem dois outros comandos usamos para junção de tabelas: UNION e UNION ALL.

Estes dois permitem que a gente junte duas ou mais tabelas, como se estivéssemos empilhando as tabelas, ou seja, colocando uma tabela em cima da outra.

Ao lado, temos um exemplo do **UNION**. Imagine que tivéssemos duas tabelas separadas por gênero, com informações de clientes.

Poderíamos juntar as duas tabelas usando o UNION, chegando no resultado de uma tabela única, contendo informações tanto dos clientes do sexo masculino quanto do sexo feminino.

Um detalhe importante sobre o uso do UNION é que as tabelas a serem juntadas precisam ter as mesmas colunas, na mesma ordem.



```

SQLQuery6.sql - LAP...us Cavalcanti (70)*  SQLQuery1.sql - LAP...us Cavalcanti (55)*  SQLQuery2.sql - LAP...us Cavalcanti (54)*
SELECT *
FROM
    DimCustomer
WHERE Gender = 'F'
UNION
SELECT *
FROM
    DimCustomer
WHERE Gender = 'M'

```

The screenshot shows the SQL Server Management Studio interface with three tabs open. The top tab is titled 'SQLQuery6.sql' and contains the provided SQL code. The bottom tabs are titled 'SQLQuery1.sql' and 'SQLQuery2.sql', which are likely results of the UNION operation. Below the tabs is a results grid with 12 rows, each representing a customer record with columns: CustomerKey, GeographyKey, CustomerLabel, Title, FirstName, MiddleName, LastName, NameStyle, BirthDate, MaritalStatus, Suffix, Gender, and EmailAddress. The data includes both male and female customers from the DimCustomer table.

	CustomerKey	GeographyKey	CustomerLabel	Title	FirstName	MiddleName	LastName	NameStyle	BirthDate	MaritalStatus	Suffix	Gender	EmailAddress
1	4	519	11003	NULL	Christy	NULL	Zhu	0	1968-02-15	S	NULL	F	christy12@Adventure-works.com
2	5	706	11004	NULL	Elizabeth	NULL	Johnson	0	1968-08-08	S	NULL	F	elizabeth5@Adventure-works.com
3	7	509	11006	NULL	Janet	G	Alvarez	0	1965-12-06	S	NULL	F	janet9@Adventure-works.com
4	9	425	11008	NULL	Rob	NULL	Verhoff	0	1964-07-07	S	NULL	F	rob4@Adventure-works.com
5	11	478	11010	NULL	Jacquelyn	C	Suarez	0	1964-02-06	S	NULL	F	jacquelyn20@Adventure-work
6	13	442	11012	NULL	Lauren	M	Walker	0	1968-01-18	M	NULL	F	lauren41@Adventure-works.com
7	15	677	11014	NULL	Sydney	NULL	Bennett	0	1968-05-09	S	NULL	F	sydney23@Adventure-works.com
8	16	443	11015	NULL	Chloe	NULL	Young	0	1979-02-27	S	NULL	F	chloe23@Adventure-works.com
9	18	699	11017	NULL	Shannon	NULL	Wang	0	1944-06-26	S	NULL	F	shannon1@Adventure-works.com
10	22	580	11021	NULL	Destiny	NULL	Wilson	0	1978-09-03	S	NULL	F	destiny7@Adventure-works.com
11	29	554	11028	NULL	Jill	NULL	Jimenez	0	1946-04-11	M	NULL	F	jill13@Adventure-works.com
12	31	613	11030	NULL	Rethanv	G	Yuan	0	1947-02-22	M	NULL	F	bethanv10@Adventure-works.com

# UNION e UNION ALL

Além do UNION, temos o **UNION ALL**.

Ele terá um resultado muito semelhante ao UNION, com apenas uma diferença: **todas as linhas duplicadas serão incluídas**.

Observe o exemplo ao lado. Temos uma união de duas tabelas, também contendo informações separadas de clientes por gênero, só que dessa vez apenas as colunas de FirstName e BirthDate.

No UNION (imagem à esquerda), o resultado mostra 18.450 linhas. Já com o UNION ALL, temos um resultado de 18.484.

Qual é a diferença?

O UNION retorna uma tabela removendo todos os duplicados, enquanto o UNION ALL mantém os duplicados.

Isso significa que essa união possui pessoas que nasceram no mesmo dia, e possuem o mesmo primeiro nome. Por isso, no caso do UNION, a quantidade de linhas resultado foi menor.

The screenshot shows two side-by-side SQL query windows in SSMS. Both queries select FirstName and BirthDate from DimCustomer where Gender is either 'F' or 'M'. The left window uses a standard UNION operator, resulting in 18,450 rows. The right window uses UNION ALL, resulting in 18,484 rows. A yellow callout box highlights the difference between the two operators.

```

SQLQuery6.sql - LAP...us Cavalcanti (70)*  SQLQuery1.sql - LAP...us Cav...
SELECT
    FirstName,
    BirthDate
FROM
    DimCustomer
WHERE Gender = 'F'
UNION
SELECT
    FirstName,
    BirthDate
FROM
    DimCustomer
WHERE Gender = 'M'

121 % < >
Resultados Mensagens
1 Aaron 1941-02-05
2 Aaron 1943-11-28
3 Aaron 1944-10-07
4 Aaron 1944-11-14
5 Aaron 1947-02-11
6 Aaron 1949-07-22
7 Aaron 1950-03-06
8 Aaron 1952-11-05
9 Aaron 1953-05-18
10 Aaron 1954-10-24
11 Aaron 1955-10-14
12 Aaron 1956-06-20
13 Aaron 1958-01-11
18.450 linhas

SQLQuery6.sql - LAP...us Cavalcanti (70)*  SQLQuery1.sql - LAP...us Cav...
SELECT
    FirstName,
    BirthDate
FROM
    DimCustomer
WHERE Gender = 'F'
UNION ALL
SELECT
    FirstName,
    BirthDate
FROM
    DimCustomer
WHERE Gender = 'M'

121 % < >
Resultados Mensagens
1 Christy 1968-02-15
2 Elizabeth 1968-08-08
3 Janet 1965-12-06
4 Rob 1964-07-07
5 Jacquelyn 1964-02-06
6 Lauren 1968-01-18
7 Sydney 1968-05-09
8 Chloe 1979-02-27
9 Shannon 1944-06-26
10 Destiny 1978-09-03
11 Jill 1946-04-11
12 Bethany 1947-02-22
13 Theresa 1947-08-22
18.484 linhas

```

# EXERCÍCIOS



1

## Questão 1

Utilize o INNER JOIN para trazer os nomes das subcategorias dos produtos, da tabela DimProductSubcategory para a tabela DimProduct.

2

## Questão 2

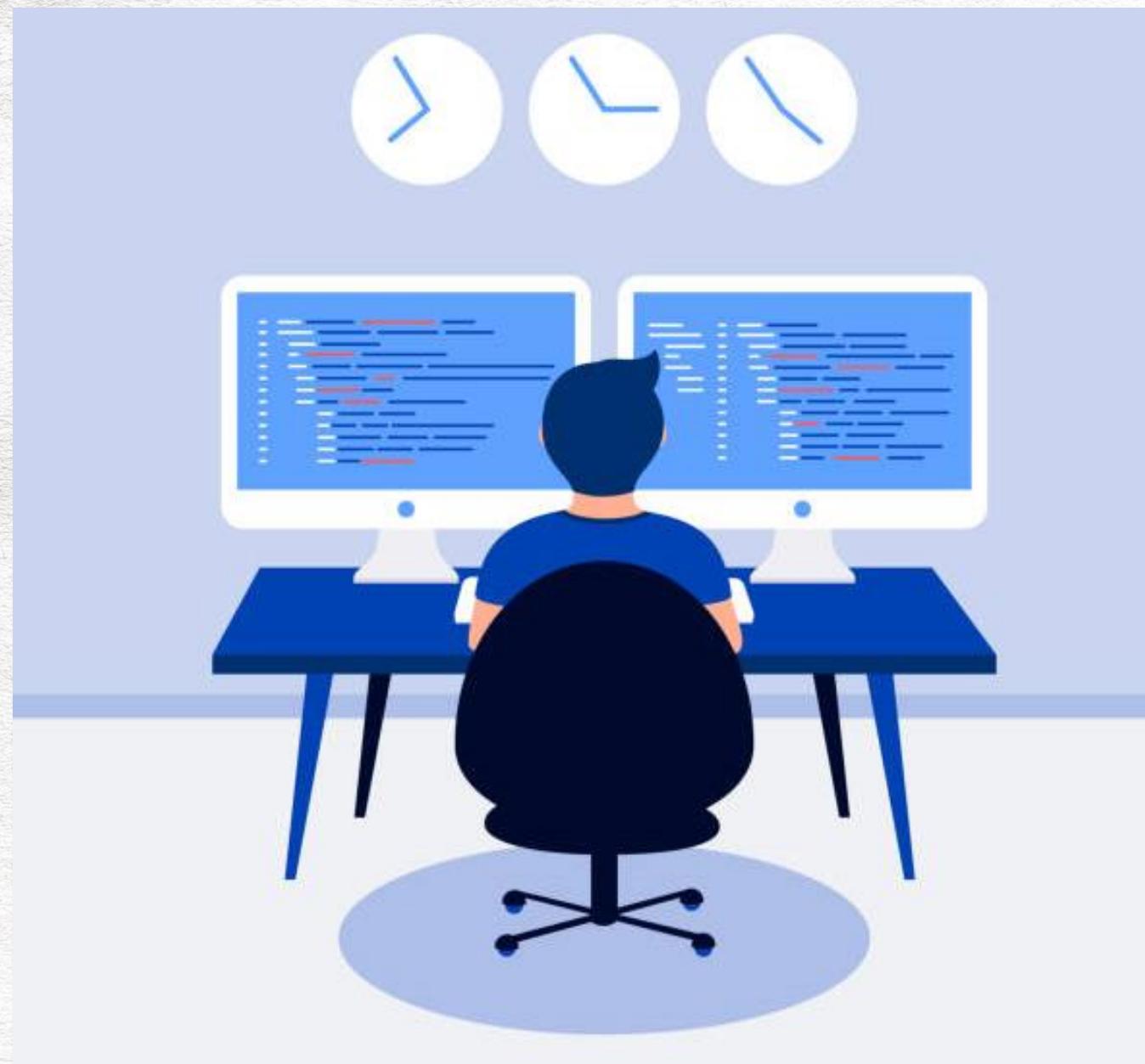
Identifique uma coluna em comum entre as tabelas DimProductSubcategory e DimProductCategory. Utilize essa coluna para complementar informações na tabela DimProductSubcategory a partir da DimProductCategory. Utilize o LEFT JOIN.

3

## Questão 3

Para cada loja da tabela DimStore, descubra qual o Continente e o Nome do País associados (de acordo com DimGeography). Seu SELECT final deve conter apenas as seguintes colunas: StoreKey, StoreName, EmployeeCount, ContinentName e RegionCountryName. Utilize o LEFT JOIN neste exercício.

# EXERCÍCIOS



4

## Questão 4

Completa a tabela DimProduct com a informação de ProductCategoryDescription. Utilize o LEFT JOIN e retorne em seu SELECT apenas as 5 colunas que considerar mais relevantes.

5

## Questão 5

A tabela FactStrategyPlan resume o planejamento estratégico da empresa. Cada linha representa um montante destinado a uma determinada AccountKey.

- a. Faça um SELECT das 100 primeiras linhas de FactStrategyPlan para reconhecer a tabela.
- b. Faça um INNER JOIN para criar uma tabela contendo o AccountName para cada AccountKey da tabela FactStrategyPlan. O seu SELECT final deve conter as colunas:
  - StrategyPlanKey
  - DateKey
  - AccountName
  - Amount

# EXERCÍCIOS



6

## Questão 6

Vamos continuar analisando a tabela FactStrategyPlan. Além da coluna AccountKey que identifica o tipo de conta, há também uma outra coluna chamada ScenarioKey. Essa coluna possui a numeração que identifica o tipo de cenário: Real, Orçado e Previsão.

Faça um INNER JOIN para criar uma tabela contendo o ScenarioName para cada ScenarioKey da tabela FactStrategyPlan. O seu SELECT final deve conter as colunas:

- StrategyPlanKey
- DateKey
- ScenarioName
- Amount

7

## Questão 7

Algumas subcategorias não possuem nenhum exemplar de produto. Identifique que subcategorias são essas.

# EXERCÍCIOS



8

## Questão 8

A tabela abaixo mostra a combinação entre Marca e Canal de Venda, para as marcas Contoso, Fabrikam e Litware. Crie um código SQL para chegar no mesmo resultado.

BrandName	ChannelName
Contoso	Catalog
Contoso	Online
Contoso	Reseller
Contoso	Store
Fabrikam	Catalog
Fabrikam	Online
Fabrikam	Reseller
Fabrikam	Store
Litware	Catalog
Litware	Online
Litware	Reseller
Litware	Store

# EXERCÍCIOS

9

## Questão 9

Neste exercício, você deverá relacionar as tabelas FactOnlineSales com DimPromotion. Identifique a coluna que as duas tabelas têm em comum e utilize-a para criar esse relacionamento.

Retorne uma tabela contendo as seguintes colunas:

- OnlineSalesKey
- DateKey
- PromotionName
- SalesAmount

A sua consulta deve considerar apenas as linhas de vendas referentes a produtos com desconto (`PromotionName <> 'No Discount'`). Além disso, você deverá ordenar essa tabela de acordo com a coluna DateKey, em ordem crescente.



# EXERCÍCIOS



10

## Questão 10

A tabela abaixo é resultado de um Join entre a tabela FactSales e as tabelas: DimChannel, DimStore e DimProduct.

Recrie esta consulta e classifique em ordem crescente de acordo com SalesAmount.

SalesKey	ChannelName	StoreName	ProductName	SalesAmount
145071	Catalog	Contoso Catalog Store	NT Washer & Dryer 27in L2700 Silver	408016,02
1873989	Catalog	Contoso Catalog Store	Litware Refrigerator 24.7CuPt X980 Silver	329598,97
729520	Catalog	Contoso Catalog Store	NT Washer & Dryer 27in L2700 Green	272718,12
1453383	Catalog	Contoso Catalog Store	Litware Refrigerator 24.7CuPt X980 Silver	245119,234
2241482	Catalog	Contoso Catalog Store	Litware Refrigerator 24.7CuPt X980 Grey	245119,234
541145	Catalog	Contoso Catalog Store	Contoso Projector 1080p X980 White	235467,00
1366947	Catalog	Contoso Catalog Store	Contoso Water Heater 7.2GPM X1800 White	227445,00
383636	Catalog	Contoso Catalog Store	NT Washer & Dryer 27in L2700 Silver	205865,04
686002	Catalog	Contoso Catalog Store	NT Washer & Dryer 27in L2700 Blue	204803,88
517342	Catalog	Contoso Catalog Store	NT Washer & Dryer 27in L2700 Blue	202681,56
2883639	Catalog	Contoso Catalog Store	Prosware Projector 1080p DLP86 White	190923,60
1498028	Catalog	Contoso Catalog Store	NT Washer & Dryer 27in L2700 Green	190212,93
3106199	Catalog	Contoso Catalog Store	Litware Refrigerator 24.7CuPt X980 Brown	188799,41
1623258	Catalog	Contoso Catalog Store	Litware Washer & Dryer 24in M260 Blue	186890,40
2850262	Catalog	Contoso Catalog Store	Adventure Works Desktop PC3.0 MS300 Silver	182441,00
2005571	Catalog	Contoso Catalog Store	Galaxy Book S 12.4in 1080p Y2000 MIL	176056,00

# GABARITOS

1

## Questão 1

Utilize o INNER JOIN para trazer os nomes das subcategorias dos produtos, da tabela DimProductSubcategory para a tabela DimProduct.

The screenshot shows a SQL query window titled "SQLQuery9.sql - LAP...us Cavalcanti (54)\*". The query is:

```
-- Questão 1
SELECT
    ProductKey AS 'ID Produto',
    ProductName AS 'Produto',
    ProductSubcategoryName AS 'Subcategoria'
FROM
    DimProduct
INNER JOIN DimProductSubcategory
    ON DimProduct.ProductSubcategoryKey = DimProductSubcategory.ProductSubcategoryKey
```

The results grid displays the following data:

ID Produto	Produto	Subcategoria
1	Contoso 512MB MP3 Player E51 Silver	MP4&MP3
2	Contoso 512MB MP3 Player E51 Blue	MP4&MP3
3	Contoso 1G MP3 Player E100 White	MP4&MP3
4	Contoso 2G MP3 Player E200 Silver	MP4&MP3
5	Contoso 2G MP3 Player E200 Red	MP4&MP3
6	Contoso 2G MP3 Player E200 Black	MP4&MP3
7	Contoso 2G MP3 Player E200 Blue	MP4&MP3
8	Contoso 4G MP3 Player E400 Silver	MP4&MP3
9	Contoso 4G MP3 Player E400 Black	MP4&MP3
10	Contoso 4G MP3 Player E400 Green	MP4&MP3

At the bottom of the results pane, a message says "Consulta executada com êxito." (Query executed successfully).

# GABARITOS

2

## Questão 2

Identifique uma coluna em comum entre as tabelas DimProductSubcategory e DimProductCategory. Utilize essa coluna para complementar informações na tabela DimProductSubcategory a partir da DimProductCategory. Utilize o LEFT JOIN.

The screenshot shows a SQL query window titled "SQLQuery9.sql - LAP...us Cavalcanti (54)\*". The query is:

```
--Questão 2
SELECT
    ProductSubcategoryKey AS 'ID Subcategoria',
    ProductSubcategoryName AS 'Subcategoria',
    ProductCategoryName AS 'Categoria'
FROM
    DimProductSubcategory
LEFT JOIN DimProductCategory
    ON DimProductSubcategory.ProductCategoryKey = DimProductCategory.ProductCategoryKey
```

The results window displays a table with three columns: ID Subcategoria, Subcategoria, and Categoria. The data is as follows:

	ID Subcategoria	Subcategoria	Categoria
1	1	MP4&MP3	Audio
2	2	Recorder	Audio
3	3	Radio	Audio
4	4	Recording Pen	Audio
5	5	Headphones	Audio
6	6	Bluetooth Headphones	Audio
7	7	Speakers	Audio
8	8	Audio Accessories	Audio
9	9	Televisions	TV and Video
10	10	VCD & DVD	TV and Video

At the bottom of the results window, a message says "Consulta executada com êxito." (Query executed successfully).

# GABARITOS

3

## Questão 3

Para cada loja da tabela DimStore, descubra qual o Continente e o Nome do País associados (de acordo com DimGeography). Seu SELECT final deve conter apenas as seguintes colunas: StoreKey, StoreName, EmployeeCount, ContinentName e RegionCountryName. Utilize o LEFT JOIN neste exercício.

The screenshot shows a SQL query window titled "SQLQuery9.sql - LAP...us Cavalcanti (54)\*". The query is labeled "- Questão 3" and contains the following code:

```
SELECT
    StoreKey AS 'ID Loja',
    StoreName AS 'Loja',
    EmployeeCount AS 'Qtd. Funcionários',
    ContinentName AS 'Continente',
    RegionCountryName AS 'País'
FROM
    DimStore
LEFT JOIN DimGeography
    ON DimStore.GeographyKey = DimGeography.GeographyKey
```

The results pane displays a table with the following data:

ID Loja	Loja	Qtd. Funcionários	Continente	País
1	Contoso Bellevue Store	19	North America	United States
2	Contoso Cambridge Store	26	Europe	United Kingdom
3	Contoso Cedar Park Store	19	North America	United States
4	Contoso Leeds Store	25	Europe	United Kingdom
5	Contoso London Store	17	Europe	United Kingdom
6	Contoso Paris Store	12	Europe	France
7	Contoso Europe Reseller	12	Europe	France
8	Contoso Spokane Store	25	North America	United States
9	Contoso Sydney No.1 Store	20	Asia	Australia
10	Contoso Sydney No.2 Store	95	Asia	Australia

At the bottom of the results pane, a message indicates: "Consulta executada com êxito." (Query executed successfully).

# GABARITOS

4

## Questão 4

Complementa a tabela DimProduct com a informação de ProductCategoryDescription. Utilize o LEFT JOIN e retorne em seu SELECT apenas as 5 colunas que considerar mais relevantes.

The screenshot shows a SQL query in the SQL Query window titled '--Questão 4'. The query uses LEFT JOINs to add the 'ProductCategoryDescription' column from the 'DimProductCategory' table to the 'DimProduct' table. The result set contains 10 rows of product names and their corresponding category descriptions, all under the 'Audio' category.

```
SQLQuery9.sql - LAP...us Cavalcanti (54)* --Questão 4
SELECT
    ProductName,
    ProductCategoryDescription
FROM
    DimProduct
LEFT JOIN DimProductSubcategory
    ON DimProduct.ProductSubcategoryKey = DimProductSubcategory.ProductSubcategoryKey
        LEFT JOIN DimProductCategory
            ON DimProductSubcategory.ProductCategoryKey = DimProductCategory.ProductCategoryKey
```

	ProductName	ProductCategoryDescription
1	Contoso 512MB MP3 Player E51 Silver	Audio
2	Contoso 512MB MP3 Player E51 Blue	Audio
3	Contoso 1G MP3 Player E100 White	Audio
4	Contoso 2G MP3 Player E200 Silver	Audio
5	Contoso 2G MP3 Player E200 Red	Audio
6	Contoso 2G MP3 Player E200 Black	Audio
7	Contoso 2G MP3 Player E200 Blue	Audio
8	Contoso 4G MP3 Player E400 Silver	Audio
9	Contoso 4G MP3 Player E400 Black	Audio
10	Contoso 4G MP3 Player E400 Green	Audio

Consulta executada com êxito. | LAPTOP-HBTCI8PQ (15.0 RTM) | LAPTOP-HBTCI8PQ\Marcus... | ContosoRetailDW | 00:00:00 | 2.517 linhas

# GABARITOS

5

## Questão 5

A tabela FactStrategyPlan resume o planejamento estratégico da empresa. Cada linha representa um montante destinado a uma determinada AccountKey.

- Faça um SELECT das 100 primeiras linhas de FactStrategyPlan para reconhecer a tabela.
- Faça um INNER JOIN para criar uma tabela contendo o AccountName para cada AccountKey da tabela FactStrategyPlan. O seu SELECT final deve conter as colunas:
  - StrategyPlanKey
  - DateKey
  - AccountName
  - Amount

```

SQLQuery9.sql - LAP...us Cavalcanti (54)*
--Questão 5
--a)
SELECT TOP(100) * FROM FactStrategyPlan
SELECT * from DimAccount

--b)
SELECT
    StrategyPlanKey,
    DateKey,
    AccountName,
    Amount
FROM
    FactStrategyPlan
INNER JOIN DimAccount
    ON FactStrategyPlan.AccountKey = DimAccount.AccountKey
  
```

	StrategyPlanKey	DateKey	AccountName	Amount
1	98347	2007-03-01 00:00:00.000	Cost of Goods Sold	52481,22
2	98348	2008-01-01 00:00:00.000	Cost of Goods Sold	391,36
3	98349	2009-09-01 00:00:00.000	Cost of Goods Sold	13348,62
4	98350	2008-10-01 00:00:00.000	Cost of Goods Sold	11907,33
5	98351	2008-12-01 00:00:00.000	Cost of Goods Sold	2888,32
6	98352	2007-10-01 00:00:00.000	Cost of Goods Sold	48194,13
7	98353	2007-12-01 00:00:00.000	Cost of Goods Sold	35516,10
8	98354	2009-04-01 00:00:00.000	Cost of Goods Sold	18038,86
9	98355	2009-07-01 00:00:00.000	Cost of Goods Sold	82570,67
10	98356	2009-09-01 00:00:00.000	Cost of Goods Sold	61954,08

Consulta executada com êxito.

# GABARITOS

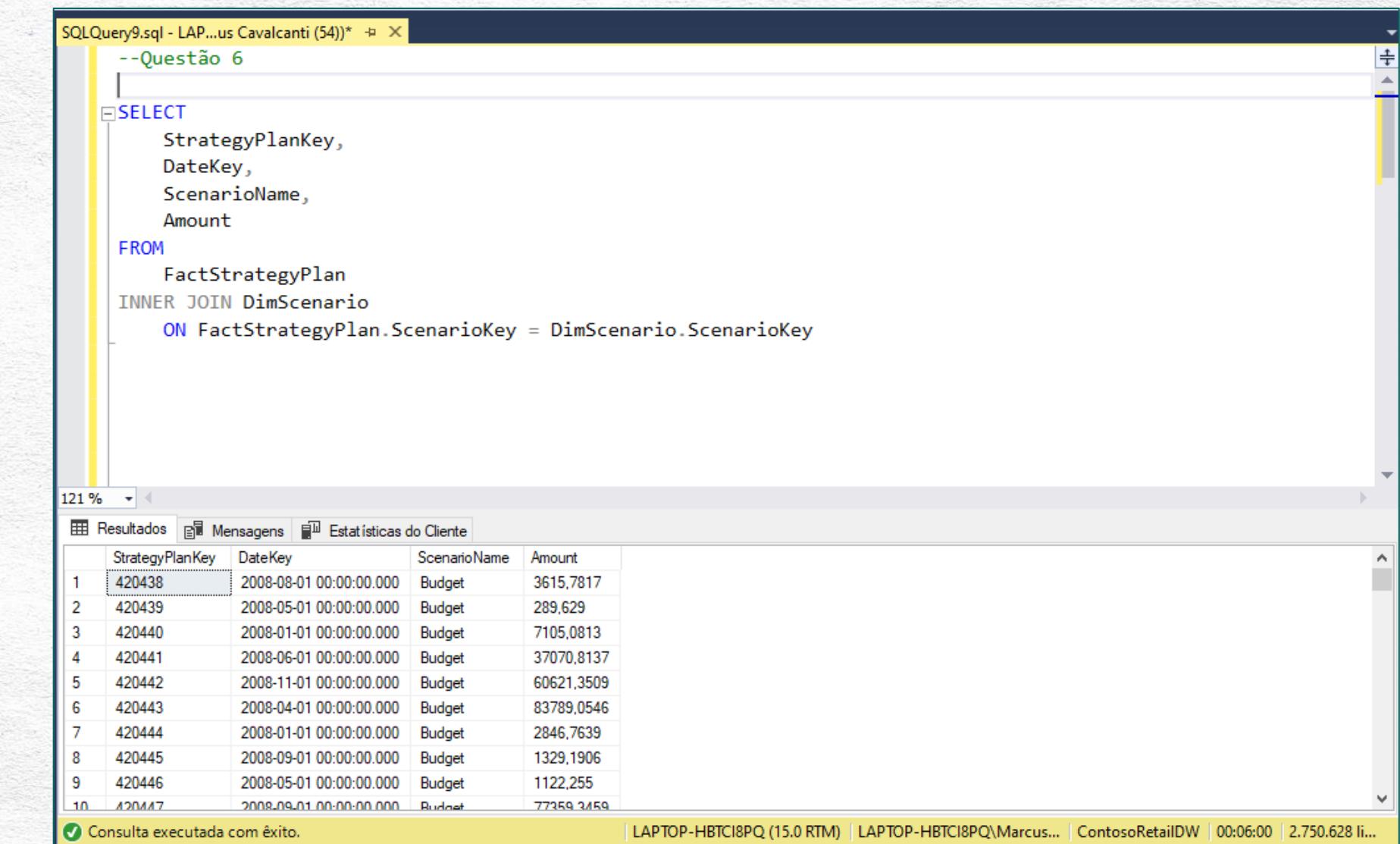
6

## Questão 6

Vamos continuar analisando a tabela FactStrategyPlan. Além da coluna AccountKey que identifica o tipo de conta, há também uma outra coluna chamada ScenarioKey. Essa coluna possui a numeração que identifica o tipo de cenário: Real, Orçado e Previsão.

Faça um INNER JOIN para criar uma tabela contendo o ScenarioName para cada ScenarioKey da tabela FactStrategyPlan. O seu SELECT final deve conter as colunas:

- StrategyPlanKey
- DateKey
- ScenarioName
- Amount



--Questão 6

```
SELECT
    StrategyPlanKey,
    DateKey,
    ScenarioName,
    Amount
FROM
    FactStrategyPlan
INNER JOIN DimScenario
    ON FactStrategyPlan.ScenarioKey = DimScenario.ScenarioKey
```

	StrategyPlanKey	DateKey	ScenarioName	Amount
1	420438	2008-01-01 00:00:00.000	Budget	3615,7817
2	420439	2008-05-01 00:00:00.000	Budget	289,629
3	420440	2008-01-01 00:00:00.000	Budget	7105,0813
4	420441	2008-06-01 00:00:00.000	Budget	37070,8137
5	420442	2008-11-01 00:00:00.000	Budget	60621,3509
6	420443	2008-04-01 00:00:00.000	Budget	83789,0546
7	420444	2008-01-01 00:00:00.000	Budget	2846,7639
8	420445	2008-09-01 00:00:00.000	Budget	1329,1906
9	420446	2008-05-01 00:00:00.000	Budget	1122,255
10	420447	2008-09-01 00:00:00.000	Budget	77359,3159

Consulta executada com êxito.



# GABARITOS

7

## Questão 7

Algumas subcategorias não possuem nenhum exemplar de produto. Identifique que subcategorias são essas.

The screenshot shows a SQL query window titled "SQLQuery9.sql - LAP...us Cavalcanti (54)\*". The query is:

```
--Questão 7
SELECT
    ProductSubcategoryName
FROM
    DimProduct
RIGHT JOIN DimProductSubcategory
    ON DimProduct.ProductSubcategoryKey = DimProductSubcategory.ProductSubcategoryKey
WHERE ProductName IS NULL
```

The results pane shows a table with one row:

ProductSubcategoryName
Radio

At the bottom of the results pane, a message says "Consulta executada com êxito." (Query executed successfully.)

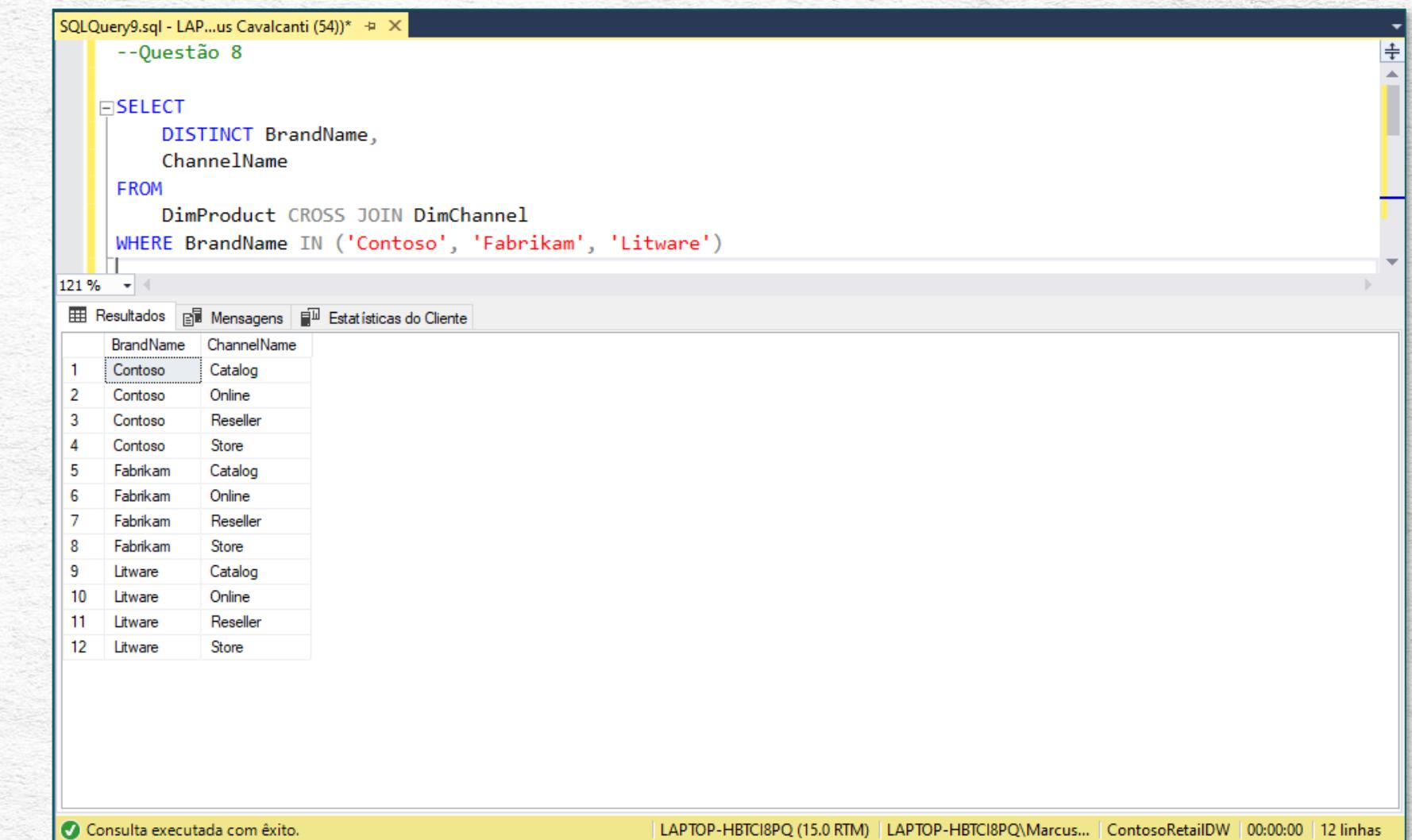
# GABARITOS

8

## Questão 8

A tabela abaixo mostra a combinação entre Marca e Canal de Venda, para as marcas Contoso, Fabrikam e Litware. Crie um código SQL para chegar no mesmo resultado.

BrandName	ChannelName
Contoso	Catalog
Contoso	Online
Contoso	Reseller
Contoso	Store
Fabrikam	Catalog
Fabrikam	Online
Fabrikam	Reseller
Fabrikam	Store
Litware	Catalog
Litware	Online
Litware	Reseller
Litware	Store



The screenshot shows a SQL query window titled "SQLQuery9.sql - LAP...us Cavalcanti (54)\*". The query is:

```
--Questão 8
SELECT
    DISTINCT BrandName,
    ChannelName
FROM
    DimProduct CROSS JOIN DimChannel
WHERE BrandName IN ('Contoso', 'Fabrikam', 'Litware')
```

The results grid displays 12 rows of data:

	BrandName	ChannelName
1	Contoso	Catalog
2	Contoso	Online
3	Contoso	Reseller
4	Contoso	Store
5	Fabrikam	Catalog
6	Fabrikam	Online
7	Fabrikam	Reseller
8	Fabrikam	Store
9	Litware	Catalog
10	Litware	Online
11	Litware	Reseller
12	Litware	Store

At the bottom of the window, a message says "Consulta executada com êxito." (Query executed successfully).

# GABARITOS

9

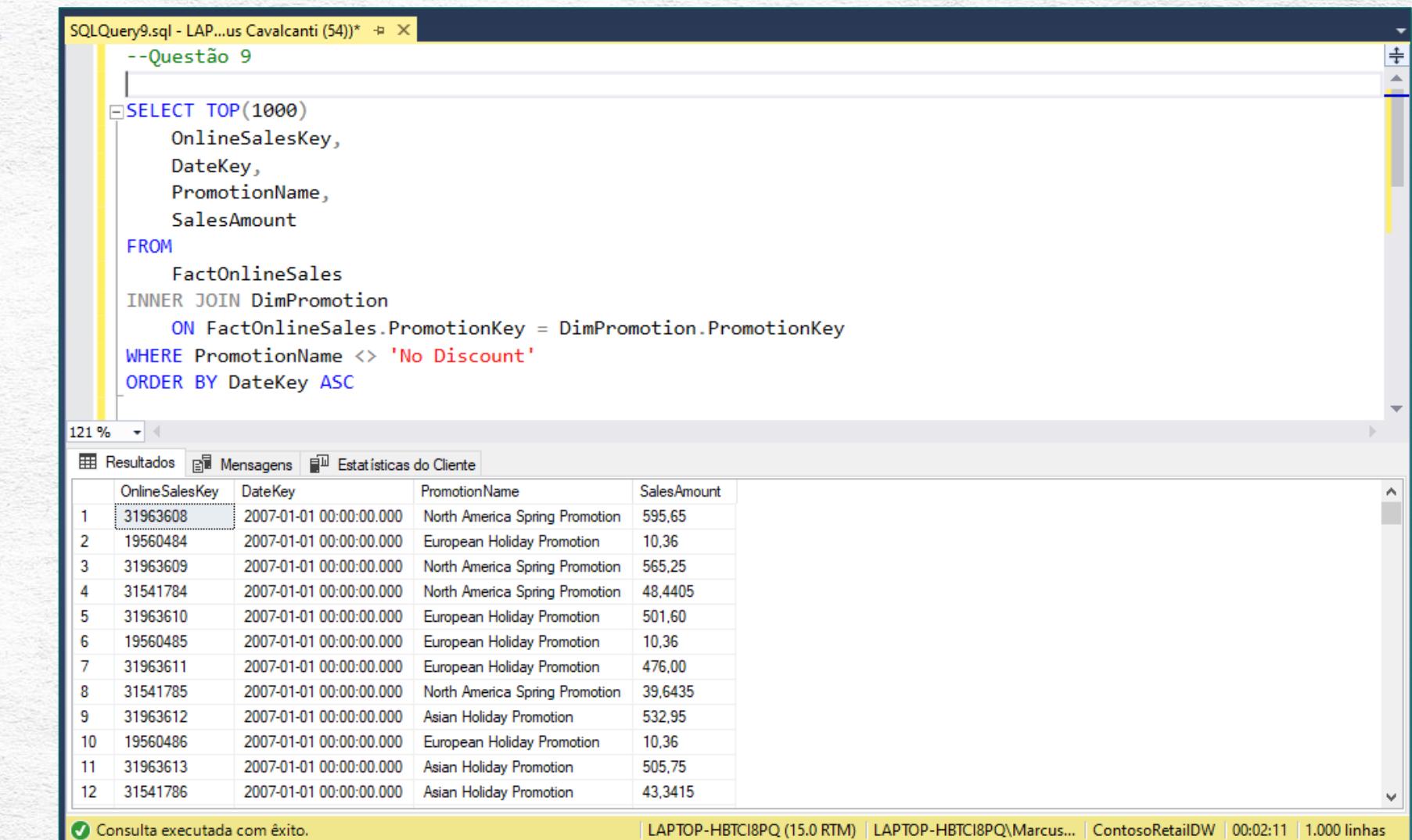
## Questão 9

Neste exercício, você deverá relacionar as tabelas FactOnlineSales com DimPromotion. Identifique a coluna que as duas tabelas têm em comum e utilize-a para criar esse relacionamento.

Retorne uma tabela contendo as seguintes colunas:

- OnlineSalesKey
- DateKey
- PromotionName
- SalesAmount

A sua consulta deve considerar apenas as linhas de vendas referentes a produtos com desconto (PromotionName <> 'No Discount'). Além disso, você deverá ordenar essa tabela de acordo com a coluna DateKey, em ordem crescente.



The screenshot shows a SQL Server Management Studio window with a query editor and a results grid. The query is:

```
--Questão 9
SELECT TOP(1000)
    OnlineSalesKey,
    DateKey,
    PromotionName,
    SalesAmount
FROM
    FactOnlineSales
INNER JOIN DimPromotion
    ON FactOnlineSales.PromotionKey = DimPromotion.PromotionKey
WHERE PromotionName <> 'No Discount'
ORDER BY DateKey ASC
```

The results grid displays 12 rows of data:

	OnlineSalesKey	DateKey	PromotionName	SalesAmount
1	31963608	2007-01-01 00:00:00.000	North America Spring Promotion	595,65
2	19560484	2007-01-01 00:00:00.000	European Holiday Promotion	10,36
3	31963609	2007-01-01 00:00:00.000	North America Spring Promotion	565,25
4	31541784	2007-01-01 00:00:00.000	North America Spring Promotion	48,4405
5	31963610	2007-01-01 00:00:00.000	European Holiday Promotion	501,60
6	19560485	2007-01-01 00:00:00.000	European Holiday Promotion	10,36
7	31963611	2007-01-01 00:00:00.000	European Holiday Promotion	476,00
8	31541785	2007-01-01 00:00:00.000	North America Spring Promotion	39,6435
9	31963612	2007-01-01 00:00:00.000	Asian Holiday Promotion	532,95
10	19560486	2007-01-01 00:00:00.000	European Holiday Promotion	10,36
11	31963613	2007-01-01 00:00:00.000	Asian Holiday Promotion	505,75
12	31541786	2007-01-01 00:00:00.000	Asian Holiday Promotion	43,3415

Consulta executada com êxito.

# GABARITOS

10

## Questão 10

A tabela abaixo é resultado de um Join entre a tabela FactSales e as tabelas: DimChannel, DimStore e DimProduct.

Recrie esta consulta e classifique em ordem crescente de acordo com SalesAmount.

SalesKey	ChannelName	StoreName	ProductName	SalesAmount
145071	Catalog	Contoso Catalog Store	NT Washer & Dryer 27in L2700 Silver	408016,02
1873989	Catalog	Contoso Catalog Store	Litware Refrigerator 24.7CuPt X980 Silver	329598,97
729520	Catalog	Contoso Catalog Store	NT Washer & Dryer 27in L2700 Green	272718,12
1453383	Catalog	Contoso Catalog Store	Litware Refrigerator 24.7CuPt X980 Silver	245119,234
2241482	Catalog	Contoso Catalog Store	Litware Refrigerator 24.7CuPt X980 Grey	245119,234
541145	Catalog	Contoso Catalog Store	Contoso Projector 1080p X980 White	235467,00
1366947	Catalog	Contoso Catalog Store	Contoso Water Heater 7.2GPM X1800 White	227445,00
383636	Catalog	Contoso Catalog Store	NT Washer & Dryer 27in L2700 Silver	205865,04
686002	Catalog	Contoso Catalog Store	NT Washer & Dryer 27in L2700 Blue	204803,88
517342	Catalog	Contoso Catalog Store	NT Washer & Dryer 27in L2700 Blue	202681,56
2883639	Catalog	Contoso Catalog Store	Proseware Projector 1080p DLP86 White	190923,60
1498028	Catalog	Contoso Catalog Store	NT Washer & Dryer 27in L2700 Green	190212,93
3106199	Catalog	Contoso Catalog Store	Litware Refrigerator 24.7CuPt X980 Brown	188799,41
1623258	Catalog	Contoso Catalog Store	Litware Washer & Dryer 24in M260 Blue	186890,40
2850262	Catalog	Contoso Catalog Store	Adventure Works Desktop PC3.0 MS300 Silver	182441,00
3065571	Cat...	Cat...	Cat...	176256,80

--Questão 10

```

SELECT TOP(100)
    SalesKey,
    ChannelName,
    StoreName,
    ProductName,
    SalesAmount
FROM
    factSales
INNER JOIN DimChannel
    ON FactSales.channelKey = DimChannel.channelKey
INNER JOIN DimStore
    ON FactSales.StoreKey = DimStore.StoreKey
INNER JOIN DimProduct
    ON FactSales.ProductKey = DimProduct.ProductKey
ORDER BY SalesAmount DESC

```

Resultados

SalesKey	ChannelName	StoreName	ProductName	SalesAmount
1	145071	Catalog	Contoso Catalog Store	NT Washer & Dryer 27in L2700 Silver
2	1873989	Catalog	Contoso Catalog Store	Litware Refrigerator 24.7CuPt X980 Silver
3	729520	Catalog	Contoso Catalog Store	NT Washer & Dryer 27in L2700 Green
4	2241482	Catalog	Contoso Catalog Store	Litware Refrigerator 24.7CuPt X980 Grey
5	1453383	Catalog	Contoso Catalog Store	Litware Refrigerator 24.7CuPt X980 Silver
6	541145	Catalog	Contoso Catalog Store	Contoso Projector 1080p X980 White
7	1366947	Catalog	Contoso Catalog Store	Contoso Water Heater 7.2GPM X1800 White
8	383636	Catalog	Contoso Catalog Store	NT Washer & Dryer 27in L2700 Silver
9	686002	Catalog	Contoso Catalog Store	NT Washer & Dryer 27in L2700 Blue

Consulta executada com êxito.

# SQL

Group By + Joins

## Aplicações

# Group By + Inner Join

Nos últimos dois módulos, aprendemos a criar agrupamentos com o **GROUP BY** e também aprendemos a fazer relações/junções entre tabelas, por meio dos **JOINS**.

Nesse módulo, vamos praticar com alguns casos que combinam os dois conceitos: criar agrupamentos utilizando o Join.

Ao lado, temos um exemplo dessa combinação (enunciado do exercício está na própria imagem). Não há nenhuma novidade em relação a combinação do GROUP BY e do JOIN.

Na verdade, é até mais fácil do que parece. Para criar agrupamentos, aproveitando as relações criadas com o JOIN, basta adicionar um GROUP BY logo ao final do código da consulta, seguindo exatamente a mesma ordem que já aprendemos.

Aqui também valem os filtros WHERE e HAVING, combinados com o GROUP BY.

Agora, vamos praticar!

```

SQLQuery9.sql - LAP...us Cavalcanti (54)*  X
--GROUP BY + JOIN: Exemplos
--1.a) Crie um agrupamento mostrando o total de vendas (SalesQuantity) por ano (CalendarYear).
--1.b) Considere apenas o mês (CalendarMonthLabel) de 'January'.
--1.c) Na tabela resultante, mostre apenas os anos com um total de vendas maior ou igual a 1.200.000

SELECT
    CalendarYear AS 'Ano',
    SUM(SalesQuantity) AS 'Total Vendido'
FROM
    FactSales
INNER JOIN DimDate
    ON FactSales.DateKey = DimDate.Datekey
WHERE CalendarMonthLabel = 'January'
GROUP BY CalendarYear
HAVING SUM(SalesQuantity) >= 1200000
ORDER BY SUM(SalesQuantity) DESC

```

Ano	Total Vendido
2009	1496731

Consulta executada com êxito.



# EXERCÍCIOS



1

## Questão 1

- Faça um resumo da quantidade vendida (Sales Quantity) de acordo com o nome do canal de vendas (ChannelName). Você deve ordenar a tabela final de acordo com SalesQuantity, em ordem decrescente.
- Faça um agrupamento mostrando a quantidade total vendida (Sales Quantity) e quantidade total devolvida (Return Quantity) de acordo com o nome das lojas (StoreName).
- Faça um resumo do valor total vendido (Sales Amount) para cada mês (CalendarMonthLabel) e ano (CalendarYear).

2

## Questão 2

Você precisa fazer uma análise de vendas por produtos. O objetivo final é descobrir o valor total vendido (SalesAmount) por produto.

- Descubra qual é a cor de produto que mais é vendida (de acordo com SalesQuantity).
- Quantas cores tiveram uma quantidade vendida acima de 3.000.000.

# EXERCÍCIOS



3

## Questão 3

Crie um agrupamento de quantidade vendida (SalesQuantity) por categoria do produto (ProductCategoryName).

Obs: Você precisará fazer mais de 1 INNER JOIN, dado que a relação entre FactSales e DimProductCategory não é direta.

4

## Questão 4

a. Você deve fazer uma consulta à tabela FactOnlineSales e descobrir qual é o nome completo do cliente que mais realizou compras online (de acordo com a coluna SalesQuantity).

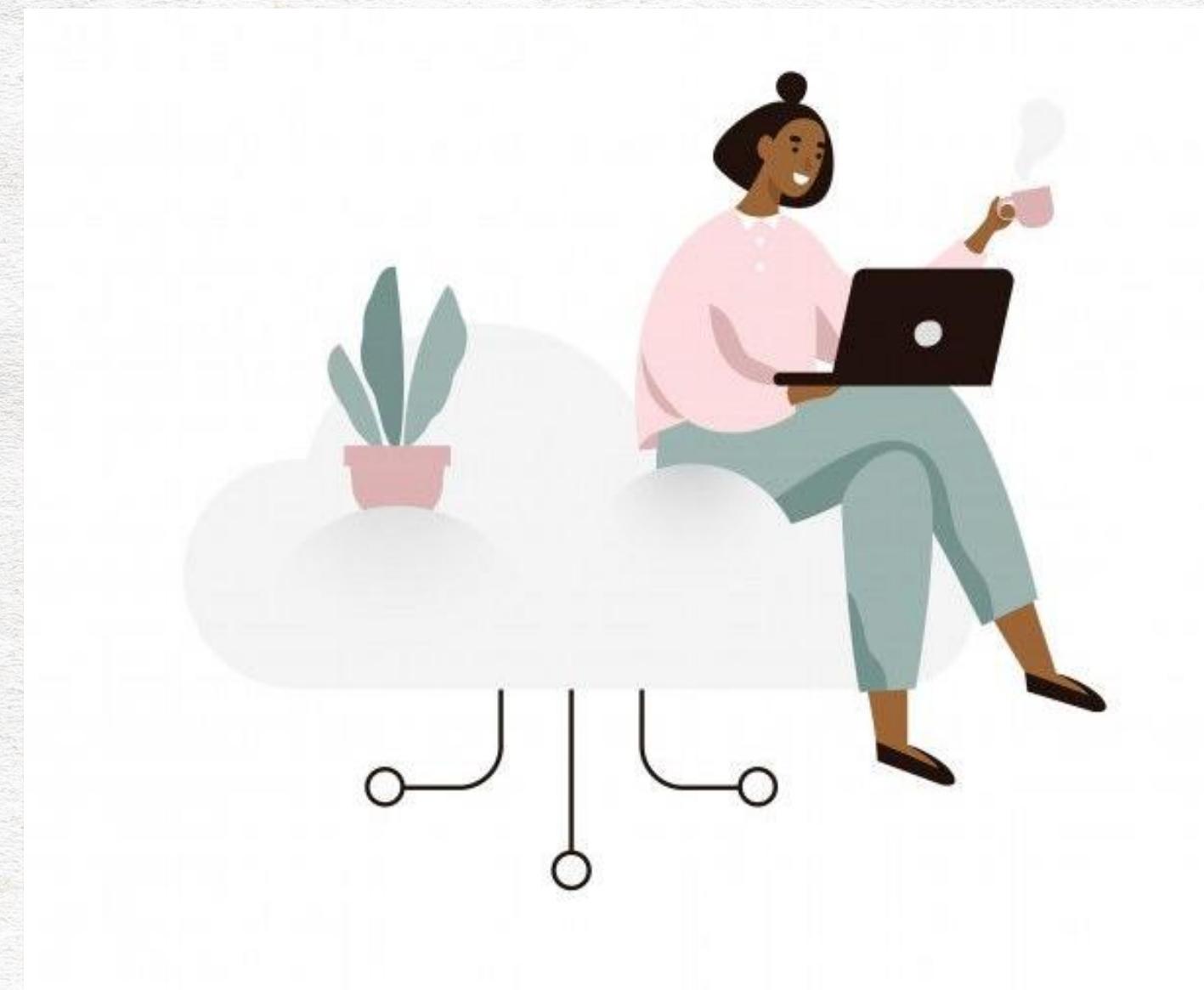
b. Feito isso, faça um agrupamento de produtos e descubra quais foram os top 10 produtos mais comprados pelo cliente da letra a, considerando o nome do produto.

5

## Questão 5

Faça um resumo mostrando o total de produtos comprados (Sales Quantity) de acordo com o sexo dos clientes.

# EXERCÍCIOS



6

## Questão 6

Faça uma tabela resumo mostrando a taxa de câmbio média de acordo com cada CurrencyDescription. A tabela final deve conter apenas taxas entre 10 e 100.

7

## Questão 7

Descubra o valor total na tabela FactStrategyPlan destinado aos cenários: Actual e Budget.

8

## Questão 8

Faça uma tabela resumo mostrando o resultado do planejamento estratégico (FactStrategyPlan) por ano.

# EXERCÍCIOS



9

## Questão 9

Faça um agrupamento de quantidade de produtos por ProductSubcategoryName. Leve em consideração em sua análise apenas a marca Contoso e a cor Silver.

10

## Questão 10

Faça um agrupamento duplo de quantidade de produtos por BrandName e ProductSubcategoryName. A tabela final deverá ser ordenada de acordo com a coluna BrandName.

# GABARITOS

1

## Questão 1

- Faça um resumo da quantidade vendida (Sales Quantity) de acordo com o nome do canal de vendas (ChannelName). Você deve ordenar a tabela final de acordo com SalesQuantity, em ordem decrescente.
- Faça um agrupamento mostrando a quantidade total vendida (Sales Quantity) e quantidade total devolvida (Return Quantity) de acordo com o nome das lojas (StoreName).
- Faça um resumo do valor total vendido (Sales Amount) para cada mês (CalendarMonthLabel) e ano (CalendarYear).

SQLQuery9.sql - LAP...us Cavalcanti (54)\*

```
--Questão 1
--a)

SELECT
    ChannelName AS 'Canal de Venda',
    SUM(SalesQuantity) AS 'Total Vendido'
FROM
    FactSales
INNER JOIN DimChannel
    ON FactSales.channelKey = DimChannel.ChannelKey
GROUP BY ChannelName
ORDER BY SUM(SalesQuantity)|
```

Resultados Mensagens Estatísticas do Cliente

	Canal de Venda	Total Vendido
1	Catalog	4454373
2	Reseller	7270570
3	Online	12506063
4	Store	29089448

# GABARITOS

1

## Questão 1

- Faça um resumo da quantidade vendida (Sales Quantity) de acordo com o nome do canal de vendas (ChannelName). Você deve ordenar a tabela final de acordo com SalesQuantity, em ordem decrescente.
- Faça um agrupamento mostrando a quantidade total vendida (Sales Quantity) e quantidade total devolvida (Return Quantity) de acordo com o nome das lojas (StoreName).
- Faça um resumo do valor total vendido (Sales Amount) para cada mês (CalendarMonthLabel) e ano (CalendarYear).

SQLQuery9.sql - LAP...us Cavalcanti (54)\* ↗ X

```
--Questão 1
--b)
SELECT
    StoreName AS 'Loja',
    SUM(SalesQuantity) AS 'Total Vendido',
    SUM(ReturnQuantity) AS 'Total Devolvido'
FROM
    FactSales
INNER JOIN DimStore
    ON FactSales.StoreKey = DimStore.StoreKey
GROUP BY StoreName
ORDER BY StoreName
```

121 %

	Loja	Total Vendido	Total Devolvido
1	Contoso Albany Store	94833	856
2	Contoso Alexandria Store	94804	865
3	Contoso Amsterdam Store	82194	946
4	Contoso Anchorage Store	96089	1016
5	Contoso Annapolis Store	94081	867
6	Contoso Appleton Store	92402	865
7	Contoso Arlington Store	93675	898
8	Contoso Ashgabat No.2 Store	160581	1384
9	Contoso Ashgabat No.1 Store	163541	1450
10	Contoso Asia Online Store	4232977	34326
11	Contoso Asia Reseller	2452197	20677
12	Contoso Athens Store	86461	945
13	Contoso Atlantic City Store	90935	902
14	Contoso Attleboro Store	92652	805



# GABARITOS

1

## Questão 1

- Faça um resumo da quantidade vendida (Sales Quantity) de acordo com o nome do canal de vendas (ChannelName). Você deve ordenar a tabela final de acordo com SalesQuantity, em ordem decrescente.
- Faça um agrupamento mostrando a quantidade total vendida (Sales Quantity) e quantidade total devolvida (Return Quantity) de acordo com o nome das lojas (StoreName).
- Faça um resumo do valor total vendido (Sales Amount) para cada mês (CalendarMonthLabel) e ano (CalendarYear).

The screenshot shows a SQL query in the SQL Query window and its results in the Results window.

```

SQLQuery9.sql - LAP...us Cavalcanti (54)*  X
--Questão 1
--c)
SELECT
    CalendarYear AS 'Ano',
    CalendarMonthLabel AS 'Mês',
    SUM(SalesAmount) AS 'Faturamento Total'
FROM
    FactSales
INNER JOIN DimDate
    ON FactSales.DateKey = DimDate.DateKey
GROUP BY CalendarYear, CalendarMonthLabel, CalendarMonth
ORDER BY CalendarMonth ASC
  
```

**Resultados**

	Ano	Mês	Faturamento Total
1	2007	January	269835263,2337
2	2007	February	298215968,3498
3	2007	March	300486926,9015
4	2007	April	400160331,5966
5	2007	May	423429127,7894
6	2007	June	409797545,5526
7	2007	July	389617372,2722
8	2007	August	388429827,1124
9	2007	September	379144599,5604
10	2007	October	423213240,8428
11	2007	November	453750209,243
12	2007	December	42500512,5600

# GABARITOS

2

## Questão 2

Você precisa fazer uma análise de vendas por produtos. O objetivo final é descobrir o valor total vendido (SalesAmount) por produto.

- Descubra qual é a cor de produto que mais é vendida (de acordo com SalesQuantity).
- Quantas cores tiveram uma quantidade vendida acima de 3.000.000.

The screenshot shows a SQL query in the SQL Query window and its execution results in the Results window.

```

SQLQuery9.sql - LAP...us Cavalcanti (54)*  X
--Questão 2
--a)
SELECT
    ColorName AS 'Cor',
    SUM(SalesQuantity) AS 'Qtd.Vendida'
FROM
    FactSales
INNER JOIN DimProduct
    ON FactSales.ProductKey = DimProduct.ProductKey
GROUP BY ColorName
  
```

**Resultados**

	Cor	Qtd.Vendida
1	Azure	251937
2	Purple	65736
3	Silver	9726178
4	Brown	1302543
5	Transparent	399520
6	Red	3223538
7	Blue	3179880
8	Orange	981342
9	Silver Grey	286107
10	Yellow	1329259
11	White	10897426
12	Gold	744064

# GABARITOS

2

## Questão 2

Você precisa fazer uma análise de vendas por produtos. O objetivo final é descobrir o valor total vendido (SalesAmount) por produto.

- Descubra qual é a cor de produto que mais é vendida (de acordo com SalesQuantity).
- Quantas cores tiveram uma quantidade vendida acima de 3.000.000.

The screenshot shows a SQL query in the SQL Query window and its results in the Results window.

```

SQLQuery9.sql - LAP...us Cavalcanti (54)*
--Questão 2
--b)
SELECT
    ColorName AS 'Cor',
    SUM(SalesQuantity) AS 'Qtd.Vendida'
FROM
    FactSales
INNER JOIN DimProduct
    ON FactSales.ProductKey = DimProduct.ProductKey
GROUP BY ColorName
HAVING SUM(SalesQuantity) >= 3000000
ORDER BY SUM(SalesQuantity) DESC
  
```

**Results:**

	Cor	Qtd.Vendida
1	Black	13230557
2	White	10897426
3	Silver	9726178
4	Grey	4784400
5	Red	3223538
6	Blue	3179880

# GABARITOS

3

## Questão 3

Crie um agrupamento de quantidade vendida (SalesQuantity) por categoria do produto (ProductCategoryName).

Obs: Você precisará fazer mais de 1 INNER JOIN, dado que a relação entre FactSales e DimProductCategory não é direta.

The screenshot shows a SQL query in the 'SQLQuery9.sql' window and its results in the 'Resultados' tab.

```
--Questão 3
SELECT
    ProductCategoryName AS 'Categoria',
    SUM(SalesQuantity) AS 'Qtd.Vendida'
FROM
    FactSales
INNER JOIN DimProduct
        ON FactSales.ProductKey = DimProduct.ProductKey
    INNER JOIN DimProductSubcategory
        ON DimProduct.ProductSubcategoryKey = DimProductSubcategory.ProductSubcategoryKey
    INNER JOIN DimProductCategory
        ON DimProductSubcategory.ProductCategoryKey = DimProductCategory.ProductCategoryKey
GROUP BY ProductCategoryName
```

The results table displays the following data:

	Categoria	Qtd.Vendida
1	Games and Toys	6040572
2	Audio	1353298
3	Cell phones	13459636
4	Music, Movies and Audio Books	1523415
5	TV and Video	3449196
6	Cameras and camcorders	6551189
7	Computers	10563676
8	Home Appliances	10379472

# GABARITOS

4

## Questão 4

- Você deve fazer uma consulta à tabela FactOnlineSales e descobrir qual é o nome completo do cliente que mais realizou compras online (de acordo com a coluna SalesQuantity).
- Feito isso, faça um agrupamento de produtos e descubra quais foram os top 10 produtos mais comprados pelo cliente da letra a, considerando o nome do produto.

SQLQuery9.sql - LAP...us Cavalcanti (54)\*

```
--Questão 4
--a)
SELECT
    DimCustomer.CustomerKey AS 'Categoria',
    FirstName AS 'Nome',
    LastName AS 'Sobrenome',
    SUM(SalesQuantity) AS 'Qtd.Vendida'
FROM
    FactOnlineSales
INNER JOIN DimCustomer
    ON FactOnlineSales.CustomerKey = DimCustomer.CustomerKey
WHERE CustomerType = 'Person'
GROUP BY DimCustomer.CustomerKey, FirstName, LastName
ORDER BY SUM(SalesQuantity) DESC
```

	Categoria	Nome	Sobrenome	Qtd.Vendida
1	7665	Robert	Long	376
2	17994	Destiny	Perry	359
3	3735	Kevin	Edwards	358
4	12667	Olivia	Anderson	355
5	4250	Abigail	Anderson	353
6	13928	Kaitlyn	Haris	353
7	17989	Harold	Perez	352
8	17783	Cody	Howard	351
9	11348	Abigail	Long	351
10	12700	Brian	Walker	250

# GABARITOS

4

## Questão 4

- Você deve fazer uma consulta à tabela FactOnlineSales e descobrir qual é o nome completo do cliente que mais realizou compras online (de acordo com a coluna SalesQuantity).
- Feito isso, faça um agrupamento de produtos e descubra quais foram os top 10 produtos mais comprados pelo cliente da letra a, considerando o nome do produto.

The screenshot shows a SQL query in the SQL Query window and its results in the Results window.

```

SQLQuery9.sql - LAP...us Cavalcanti (54)* X
--Questão 4
--b)
SELECT
    ProductName AS 'Produto',
    SUM(SalesQuantity) AS 'Qtd.Vendida'
FROM
    FactOnlineSales
INNER JOIN DimProduct
    ON FactOnlineSales.ProductKey = DimProduct.ProductKey
WHERE CustomerKey = 7665
GROUP BY ProductName
ORDER BY SUM(SalesQuantity) DESC
  
```

The Results window displays the following table:

	Produto	Qtd.Vendida
1	Adventure Works 26" 720p LCD HDTV M140 Silver	37
2	SV 16xDVD M360 Black	37
3	Contoso Telephoto Conversion Lens X400 Silver	24
4	A. Datum SLR Camera X137 Grey	24
5	NT Bluetooth Stereo Headphones E52 Blue	17
6	Contoso 4G MP3 Player E400 Silver	16
7	Contoso Optical USB Mouse M45 White	15
8	SV Keyboard E90 White	15
9	SV 40GB USB2.0 Portable Hard Disk E400 Silver	9
10	Contoso USB Cable M250 White	9

# GABARITOS

5

## Questão 5

Faça um resumo mostrando o total de produtos comprados (Sales Quantity) de acordo com o sexo dos clientes.

The screenshot shows a SQL query in the SQL Query window titled "SQLQuery9.sql - LAP...us Cavalcanti (54)\*". The query is:

```
--Questão 5
SELECT
    Gender AS 'Sexo',
    SUM(SalesQuantity) AS 'Qtd.Vendida'
FROM
    FactOnlineSales
INNER JOIN DimCustomer
    ON FactOnlineSales.CustomerKey = DimCustomer.CustomerKey
WHERE Gender IS NOT NULL
GROUP BY Gender
```

The results window shows the following table:

	Sexo	Qtd.Vendida
1	M	1429353
2	F	1386701

# GABARITOS

6

## Questão 6

Faça uma tabela resumo mostrando a taxa de câmbio média de acordo com cada CurrencyDescription. A tabela final deve conter apenas taxas entre 10 e 100.

The screenshot shows a SQL query in the SQL Query window and its results in the Results window.

**SQL Query Window:**

```
--Questão 6
SELECT
    CurrencyDescription,
    AVG(AverageRate) AS 'Taxa Média'
FROM
    FactExchangeRate
INNER JOIN DimCurrency
    ON FactExchangeRate.CurrencyKey = DimCurrency.CurrencyKey
GROUP BY CurrencyDescription
HAVING AVG(AverageRate) BETWEEN 10 AND 100
```

**Results Window:**

	CurrencyDescription	Taxa Média
1	Bhutan Ngultrum	45,3475367741935
2	Indian Rupee	43,3835251612903
3	Russian Ruble	26,4664903225806
4	Syrian Pound	48,6909677419355
5	Taiwan Dollar	31,9811390322581
6	Thai Baht	32,636354516129

# GABARITOS

7

## Questão 7

Descubra o valor total na tabela FactStrategyPlan destinado aos cenários: Actual e Budget.

The screenshot shows a SQL query in the SQL Query window and its execution results in the Results pane.

**SQL Query:**

```
--Questão 7
SELECT
    ScenarioName AS 'Cenario',
    SUM(Amount) AS 'Total'
FROM
    FactStrategyPlan
INNER JOIN DimScenario
    ON FactStrategyPlan.ScenarioKey = DimScenario.ScenarioKey
GROUP BY ScenarioName
HAVING ScenarioName IN ('Actual', 'Budget')
```

**Results:**

	Cenario	Total
1	Actual	22412098587,5267
2	Budget	21041994176,9634

# GABARITOS

8

## Questão 8

Faça uma tabela resumo mostrando o resultado do planejamento estratégico (FactStrategyPlan) por ano.

The screenshot shows a SQL query in the SQL Query window titled "SQLQuery9.sql - LAP...us Cavalcanti (54)\*". The query is:

```
--Questão 8
SELECT
    CalendarYear AS 'Ano',
    SUM(Amount) AS 'Total'
FROM
    FactStrategyPlan
INNER JOIN DimDate
    ON FactStrategyPlan.DateKey = DimDate.DateKey
GROUP BY CalendarYear
```

The results pane shows a table with three rows:

	Ano	Total
1	2007	23909967021,2265
2	2008	21500169509,0507
3	2009	20483426084,2657

# GABARITOS

9

## Questão 9

Faça um agrupamento de quantidade de produtos por ProductSubcategoryName. Leve em consideração em sua análise apenas a marca Contoso e a cor Silver.

The screenshot shows a SQL query in the SQL Query window and its execution results in the Results window.

**SQL Query:**

```
--Questão 9
SELECT
    ProductSubcategoryName AS 'Subcategoria',
    COUNT(*) AS 'Qtd. Produtos'
FROM
    DimProduct
INNER JOIN DimProductSubcategory
    ON DimProduct.ProductSubcategoryKey = DimProductSubcategory.ProductSubcategoryKey
WHERE
    BrandName = 'Contoso' AND ColorName = 'Silver'
GROUP BY ProductSubcategoryName
```

**Results:**

Subcategoria	Qtd. Produtos
Air Conditioners	7
Cameras & Camcorders Accessories	17
Cell phones Accessories	7
Coffee Machines	6
Computers Accessories	8
Digital SLR Cameras	4
Home Theater System	13
Microwaves	6
Movie DVD	13
MP3&MP4	6

# GABARITOS

10

## Questão 10

Faça um agrupamento duplo de quantidade de produtos por BrandName e ProductSubcategoryName. A tabela final deverá ser ordenada de acordo com a coluna BrandName.

The screenshot shows a SQL query in the SQL Query window and its results in the Results window.

**SQL Query:**

```
--Questão 10
SELECT
    BrandName AS 'Marca',
    ProductSubcategoryName AS 'Subcategoria',
    COUNT(*) AS 'Qtd. Produtos'
FROM
    DimProduct
INNER JOIN DimProductSubcategory
    ON DimProduct.ProductSubcategoryKey = DimProductSubcategory.ProductSubcategoryKey
GROUP BY BrandName, ProductSubcategoryName
ORDER BY BrandName ASC
```

**Results:**

	Marca	Subcategoria	Qtd. Produtos
1	A. Datum	Digital Cameras	100
2	A. Datum	Digital SLR Cameras	32
3	Adventure Works	Coffee Machines	24
4	Adventure Works	Desktops	22
5	Adventure Works	Lamps	40
6	Adventure Works	Laptops	28
7	Adventure Works	Monitors	28
8	Adventure Works	Televisions	50
9	Contoso	Air Conditioners	35
10	Contoso	Cameras & Camcorders Accessories	69

# SQL

## Variáveis

# Tipos de dados e operações básicas

Todo dado possui um tipo específico: ele pode ser um número, um texto ou uma data.

Em resumo, temos abaixo os 4 principais tipos de dados:

## Decimal (FLOAT)

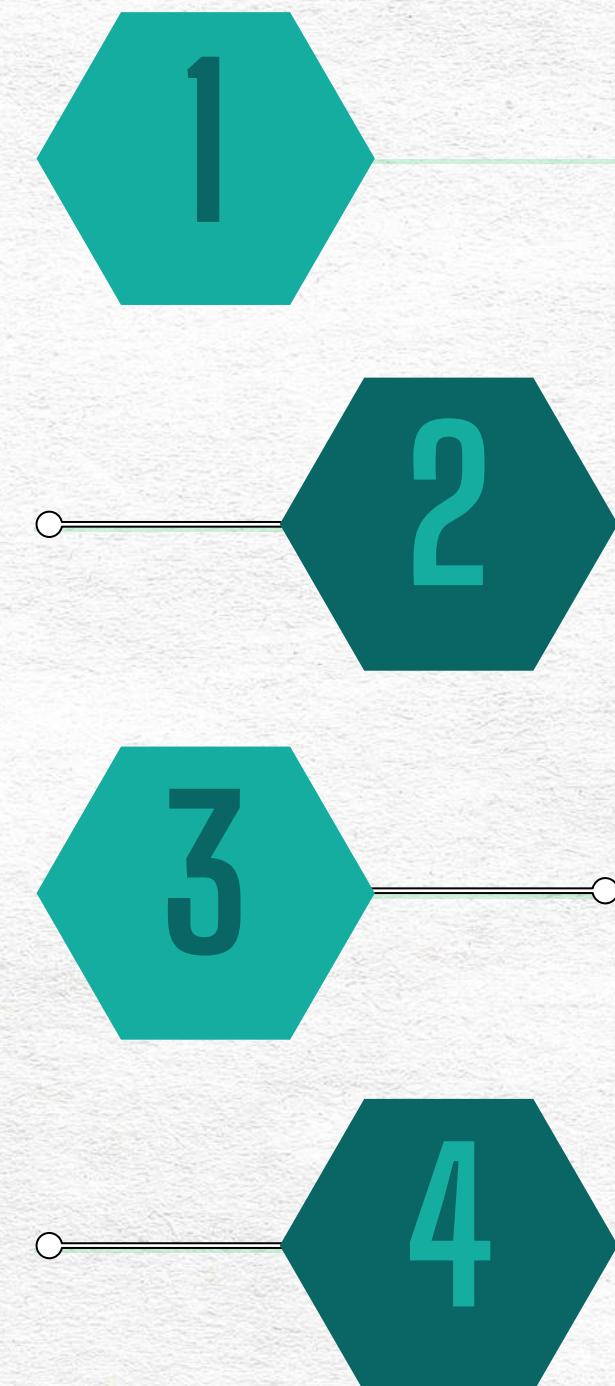
Qualquer número que contenha casas decimais. Exemplos: 10.33, 90.91, 410.87.

A forma como o SQL vai entender que estamos trabalhando com um número decimal será através do identificador **FLOAT**.

## Data (DATETIME)

Qualquer dado no formato de data. Exemplos: '01/01/2021', '23/03/2021'.

A forma como o SQL vai entender que estamos trabalhando com uma data será através do identificador **DATETIME**



## Inteiro (INT)

Qualquer número que seja representado apenas pela sua parte inteira, sem casas decimais. Exemplos: 1, 100, 589.

A forma como o SQL vai entender que estamos trabalhando com um número inteiro será através do identificador **INT**

## Texto (VARCHAR)

Toda cadeia de caracteres que pode ser interpretada como um texto. Exemplos: 'Carla', 'Motorola', 'Pastel', '44'.

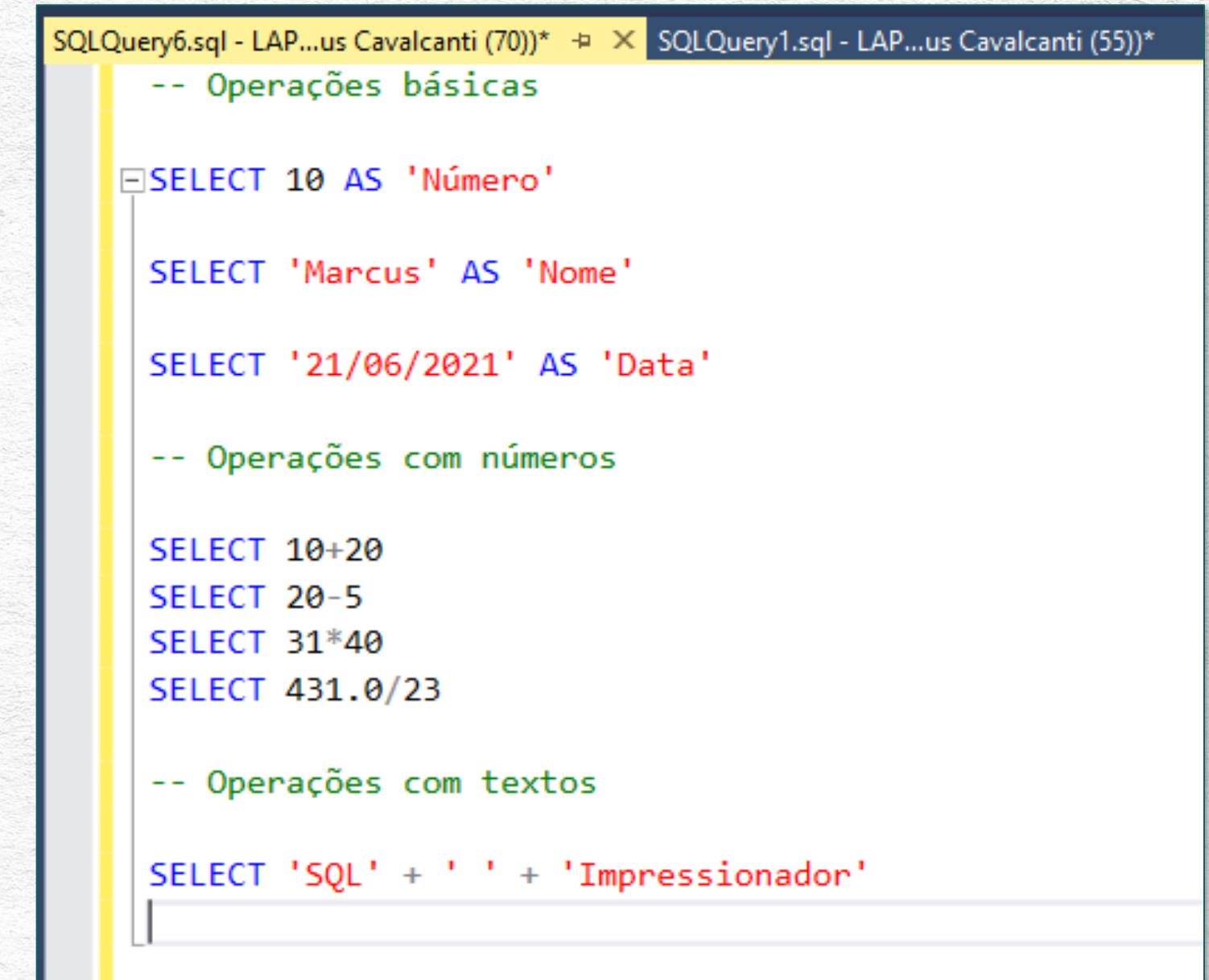
A forma como o SQL vai entender que estamos trabalhando com um texto será através do identificador **VARCHAR**



# Tipos de dados e operações básicas

Os tipos de dados determinam quais operações conseguimos fazer com os dados: operações matemáticas, concatenação, etc.

Ao lado, temos alguns exemplos de operações que conseguimos fazer com os diferentes tipos de dados.



The screenshot shows a SQL query editor window with two tabs: 'SQLQuery6.sql' and 'SQLQuery1.sql'. The code is organized into sections:

- Operações básicas
  - SELECT 10 AS 'Número'
  - SELECT 'Marcus' AS 'Nome'
  - SELECT '21/06/2021' AS 'Data'
- Operações com números
  - SELECT 10+20
  - SELECT 20-5
  - SELECT 31\*40
  - SELECT 431.0/23
- Operações com textos
  - SELECT 'SQL' + ' ' + 'Impressionador'

# CAST e SQL\_VARIANT\_PROPERTY

A função **CAST** nos permite especificar o tipo de um dado. Ao lado, temos alguns exemplos.

Observe na forma como o SQL entende cada tipo de dado:

- **INT**: inteiro
- **FLOAT**: decimal
- **VARCHAR**: texto/string
- **DATETIME**: data

Um comando muito útil para se descobrir o tipo dos dados é o **SQL\_VARIANT\_PROPERTY**.

Esta função nos pede dois argumentos: o **primeiro argumento** é o dado que queremos descobrir o tipo, e o **segundo argumento** é a informação que queremos: **BaseType**, ou seja, o tipo do dado.

O resultado pode ser visto ao lado. Na primeira linha, passar apenas a data '31/05/2014' faz com que o dado seja armazenado como texto (**varchar**). Em contrapartida, quando usamos o **CAST** para forçar o tipo do dado para data, temos como resultado o **datetime**.

```

SELECT CAST(21.45 AS INT)
SELECT CAST(21.45 AS FLOAT)
SELECT CAST(18.7 AS VARCHAR(30))
SELECT CAST('15.6' AS FLOAT)
SELECT CAST('31/05/2014' AS DATETIME)
  
```

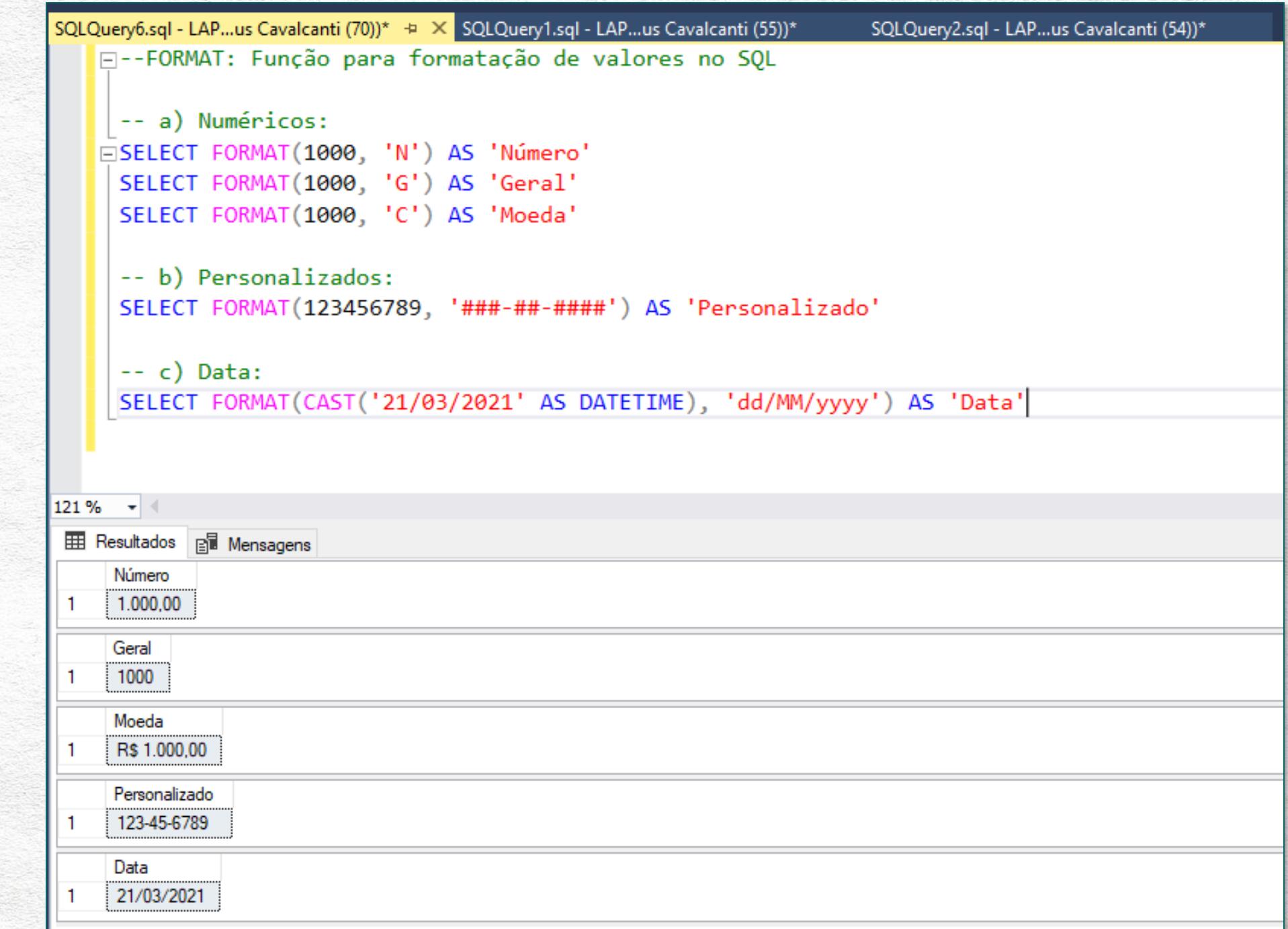
Data 1	varchar
1	

Data 2	datetime
1	

# FORMAT

A função **FORMAT** nos permite formatar um valor a partir de uma determinado código de formatação.

Ao lado, temos os principais formatos que podemos utilizar com valores.



The screenshot shows a SQL Server Management Studio window with three tabs at the top: SQLQuery6.sql, SQLQuery1.sql, and SQLQuery2.sql. The SQLQuery1.sql tab is active, displaying code examples for the FORMAT function:

```

--FORMAT: Função para formatação de valores no SQL

-- a) Numéricos:
SELECT FORMAT(1000, 'N') AS 'Número'
SELECT FORMAT(1000, 'G') AS 'Geral'
SELECT FORMAT(1000, 'C') AS 'Moeda'

-- b) Personalizados:
SELECT FORMAT(123456789, '###-##-####') AS 'Personalizado'

-- c) Data:
SELECT FORMAT(CAST('21/03/2021' AS DATETIME), 'dd/MM/yyyy') AS 'Data'

```

The Results tab displays the output for each query:

	Número
1	1.000,00

	Geral
1	1000

	Moeda
1	R\$ 1.000,00

	Personalizado
1	123-45-6789

	Data
1	21/03/2021

# ROUND, FLOOR e CEILING: Funções de arredondamento

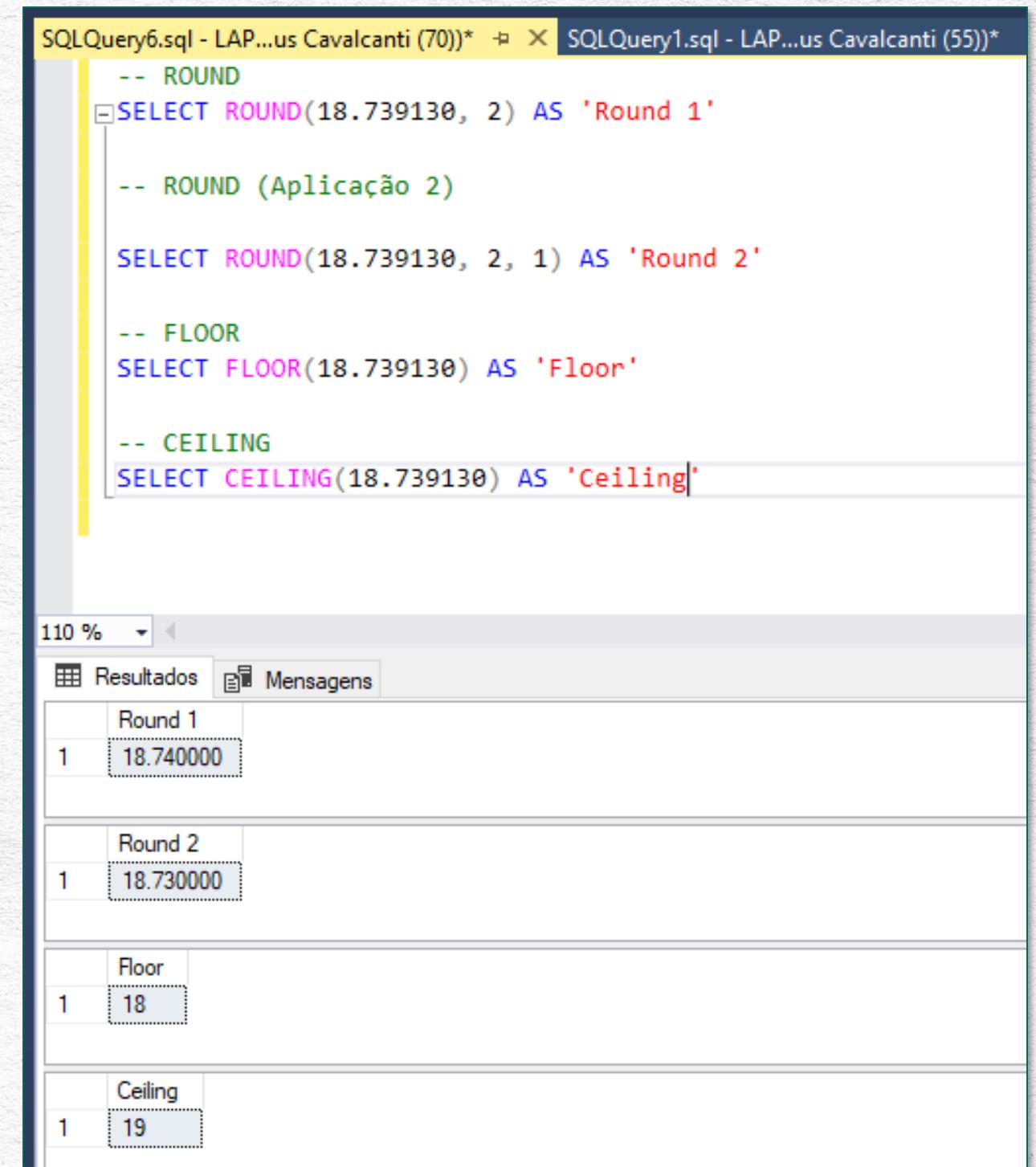
Existem algumas funções muito úteis para arredondamento de valores:  
**ROUND, FLOOR e CEILING.**

Essas funções são bem intuitivas:

- ROUND: arredonda um valor com uma quantidade de casas decimais.
- FLOOR: arredonda um valor para baixo, ou seja, para um número inteiro logo abaixo.
- CEILING: arredonda um valor para cima, ou seja, para um número inteiro logo acima.

Na imagem ao lado, é possível visualizar o resultado dessas funções, considerando o valor 18.739130.

**Obs:** A função ROUND possui uma aplicação além de apenas arredondar um valor de acordo com um número de casas decimais. Na aplicação 2 da imagem ao lado, utilizamos um terceiro argumento (número 1) para informar que o que queremos na verdade não é arredondar, e sim cortar o número na segunda casa decimal. Então em vez do valor ficar em 18.74, ele fica em 18.73.



The screenshot shows a SQL Server Management Studio window with two query panes. The top pane contains the following SQL code:

```
-- ROUND
SELECT ROUND(18.739130, 2) AS 'Round 1'

-- ROUND (Aplicação 2)
SELECT ROUND(18.739130, 2, 1) AS 'Round 2'

-- FLOOR
SELECT FLOOR(18.739130) AS 'Floor'

-- CEILING
SELECT CEILING(18.739130) AS 'Ceiling'
```

The bottom pane shows the results in four tabs: Resultados, Mensagens, Round 1, Round 2, Floor, and Ceiling. The results are:

Resultados	Mensagens	Round 1	Round 2	Floor	Ceiling
1		18.740000	18.730000	18	19

# DECLARE e SET

Chegou a hora de entender o que são variáveis no SQL e como declará-las.

## 1. O que é uma variável?

Uma variável é um objeto que armazena o valor de um dado. Imagine por exemplo a sua carteira. A sua carteira é capaz de armazenar uma determinada quantidade de dinheiro: 10 reais, 20 reais, 100 reais. A carteira é como se fosse a variável, o local onde armazenamos um valor, no caso, o dinheiro.

Variáveis são muito úteis para tornar nossos cálculos mais automáticos. Sempre que precisarmos utilizar um mesmo valor diversas vezes ao longo do código, podemos utilizar variáveis pro caso da gente precisar alterar estes valores de uma vez. Com variáveis, alteramos uma vez só e o código ficará muito mais otimizado.

## 2. Declarando uma variável

Declaramos uma variável dentro do SQL conforme a estrutura abaixo:

```
DECLARE @var TIPO  
SET @var = valor
```

Ou seja, toda declaração começa com um DECLARE, seguido do nome da variável (iniciando com o @), um TIPO (que pode ser INT, FLOAT, VARCHAR ou DATETIME, e na linha seguinte atribuímos um valor a essa variável).

Exemplo:

```
DECLARE @quantidade INT  
SET @quantidade = 100
```



# DECLARE e SET

## 3. Declarando mais de uma variável

Também é possível declarar mais de uma variável por vez:

```
DECLARE @var1 TIPO1, @var2 TIPO2  
SET @var1 = valor1  
SET @var2 = valor2
```

Exemplo:

```
DECLARE @quantidade INT, @preco FLOAT, @faturamento FLOAT  
SET @quantidade = 100  
SET @preco = 8.99  
SET @faturamento = @quantidade * @preco
```

# DECLARE e SET

## 4. Exemplos

Ao lado, temos alguns exemplos de variáveis. Tente exercitar!

```
SQLQuery6.sql - LAP...us Cavalcanti (70)* ⇔ X SQLQuery1.sql - LAP...us Cavalcanti (55)*      SQLQuery2.sql - LAP...
-- Exemplo 1: Declare uma variável chamada 'idade' e armazene o valor 30

DECLARE @idade INT
SET @idade = 30

SELECT @idade

-- Exemplo 2: Declare uma variável chamada 'preco' e armazene o valor 10.89

DECLARE @preco FLOAT
SET @preco = 10.89
SELECT @preco

DECLARE @preco2 FLOAT
SET @preco2 = 10.89
SELECT @preco2

-- Exemplo 3: Declare uma variável chamada 'nome' e armazene o valor 'Mateus'

DECLARE @nome VARCHAR(50)
SET @nome = 'Mateus'
SELECT @nome

-- Exemplo 4: Declare uma variável chamada 'data' e armazene uma data.

DECLARE @data DATETIME
SET @data = '2021-10-28'
SELECT @data
```



# Utilizando uma variável em uma consulta (Exemplo 1)

É possível utilizar variáveis dentro de consultas, como mostrado no exemplo ao lado.

Criamos uma variável **@varDesconto** para armazenar um determinado valor e usamos em nossa consulta para calcular o preço com desconto em cima dos produtos.

```

SQLQuery6.sql - LAP...us Cavalcanti (70)*  X SQLQuery1.sql - LAP...us Cavalcanti (55)*      SQLQ
-- Aplique um desconto de 10% em todos os preços dos produtos.
-- Sua consulta final deve conter as colunas ProductKey, ProductName,
-- UnitPrice e Preço com Desconto.

DECLARE @varDesconto FLOAT
SET @varDesconto = 0.1

SELECT
    ProductKey AS 'ID',
    ProductName AS 'Nome do Produto',
    UnitPrice AS 'Preço',
    UnitPrice * (1 - @varDesconto) AS 'Preço com Desconto'
FROM
    DimProduct
  
```

100 %

	ID	Nome do Produto	Preço	Preço com Desconto
1	1	Contoso 512MB MP3 Player E51 Silver	12,99	11,691
2	2	Contoso 512MB MP3 Player E51 Blue	12,99	11,691
3	3	Contoso 1G MP3 Player E100 White	14,52	13,068
4	4	Contoso 2G MP3 Player E200 Silver	21,57	19,413
5	5	Contoso 2G MP3 Player E200 Red	21,57	19,413
6	6	Contoso 2G MP3 Player E200 Black	21,57	19,413
7	7	Contoso 2G MP3 Player E200 Blue	21,57	19,413
8	8	Contoso 4G MP3 Player E400 Silver	59,99	53,991
9	9	Contoso 4G MP3 Player E400 Black	59,99	53,991
10	10	Contoso 4G MP3 Player E400 Green	59,99	53,991
11	11	Contoso 4G MP3 Player E400 Orange	59,99	53,991
12	12	Contoso 4GB Flash MP3 Player F401 Blue	77,68	69,912

Consulta executada com êxito. | LAPTOP-HBTCI8PQ (15.0)

# Utilizando uma variável em uma consulta (Exemplo 1)

Ao lado, temos mais um exemplo de como usar variáveis dentro de consultas.

Dessa vez, criamos uma variável `@varData` para tornar o filtro das consultas mais automático.

Assim, sempre que a gente quiser analisar uma nova data, alteramos o valor da variável uma única vez, e todos os lugares que utilizarem esse valor dentro do código serão atualizados automaticamente.

```

SQLQuery6.sql - LAP...us Cavalcanti (70)* ⇢ X SQLQuery1.sql - LAP...us Cavalcanti (55)*
-- Crie uma variável de data para otimizar a consulta abaixo.
DECLARE @varData DATETIME
SET @varData = '01/01/1980'

SELECT
    FirstName AS 'Nome',
    LastName AS 'Sobrenome',
    BirthDate AS 'Nascimento',
    'Cliente' AS 'Tipo'
FROM
    DimCustomer
WHERE BirthDate >= @varData

UNION

SELECT
    FirstName AS 'Nome',
    LastName AS 'Sobrenome',
    BirthDate AS 'Nascimento',
    'Funcionário' AS 'Tipo'
FROM
    DimEmployee
WHERE BirthDate >= @varData
ORDER BY Nascimento
  
```

	Nome	Sobrenome	Nascimento	Tipo
1	Angela	Kelly	1980-01-06	Cliente
2	Arthur	Fernandez	1980-01-06	Cliente
3	Caroline	Russell	1980-01-06	Cliente
4	Christine	Luo	1980-01-06	Cliente
5	Terrence	Xu	1980-01-06	Cliente
6	Ian	Price	1980-01-08	Cliente
7	Manuel	Fernandez	1980-01-11	Cliente
8	Toni	Sai	1980-01-11	Cliente

# Armazenando o resultado de uma consulta em uma variável

Podemos também armazenar o resultado de uma consulta dentro de uma variável.

No exemplo ao lado, armazenamos dentro de duas variáveis os resultados de dois SELECTs: um que calcula a quantidade de funcionários em uma empresa, e o outro que calcula a quantidade de clientes, ambos considerando uma data de nascimento.

The screenshot shows the SQL Server Management Studio interface. In the top tab bar, the active query is 'SQLQuery6.sql - LAP...us Cavalcanti (70)\*'. Other tabs include 'SQLQuery1.sql - LAP...us Cavalcanti (55)\*' and 'SQLQuery2.sql - LAP...us Cavalcanti (54)\*'. The main area displays the following T-SQL script:

```
-- Crie uma variável de data para otimizar a consulta abaixo.  
DECLARE @varData DATETIME  
DECLARE @varNumFuncionarios INT  
DECLARE @varNumClientes INT  
SET @varData = '01/01/1980'  
SET @varNumFuncionarios = (SELECT COUNT(*) FROM DimEmployee WHERE BirthDate >= @varData)  
SET @varNumClientes = (SELECT COUNT(*) FROM DimCustomer WHERE BirthDate >= @varData)  
  
SELECT 'Número de Funcionários', @varNumFuncionarios  
UNION  
SELECT 'Número de Clientes', @varNumClientes
```

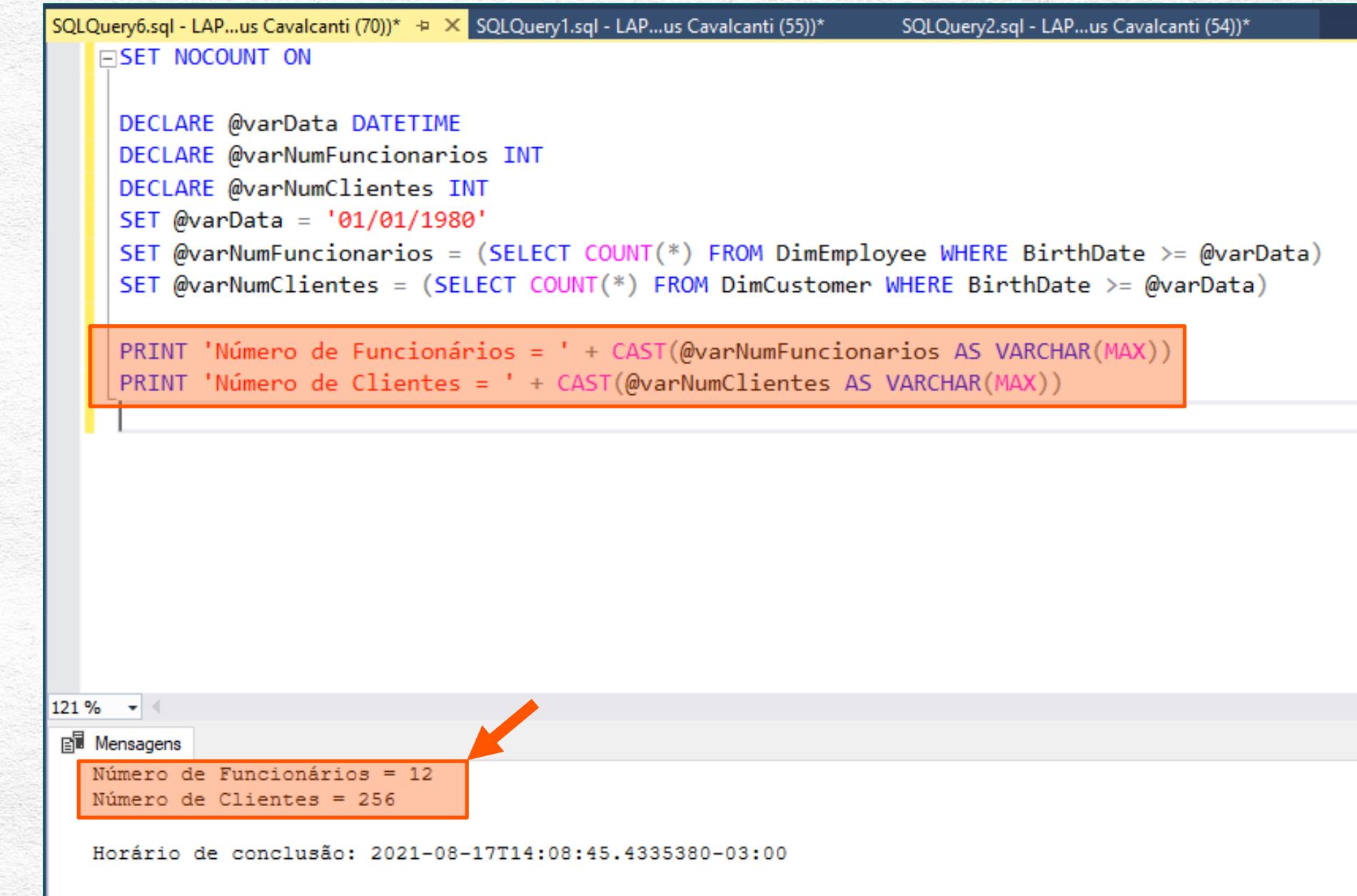
In the bottom results pane, there is a single row of data:

(Nenhum nome de coluna)	(Nenhum nome de coluna)
1	Número de Funcionários
2	Número de Clientes

The first row corresponds to the 'Número de Funcionários' column, showing the value 12. The second row corresponds to the 'Número de Clientes' column, showing the value 256.

# PRINT

Um outro comando útil é o **PRINT**. Podemos criar um código para printar um determinado valor na caixa de mensagens, indicada na imagem abaixo:



The screenshot shows a SQL Server Management Studio interface with three tabs at the top: 'SQLQuery6.sql - LAP...us Cavalcanti (70)\*', 'SQLQuery1.sql - LAP...us Cavalcanti (55)\*', and 'SQLQuery2.sql - LAP...us Cavalcanti (54)\*'. The code in the main window is as follows:

```
SET NOCOUNT ON  
  
DECLARE @varData DATETIME  
DECLARE @varNumFuncionarios INT  
DECLARE @varNumClientes INT  
SET @varData = '01/01/1980'  
SET @varNumFuncionarios = (SELECT COUNT(*) FROM DimEmployee WHERE BirthDate >= @varData)  
SET @varNumClientes = (SELECT COUNT(*) FROM DimCustomer WHERE BirthDate >= @varData)  
  
PRINT 'Número de Funcionários = ' + CAST(@varNumFuncionarios AS VARCHAR(MAX))  
PRINT 'Número de Clientes = ' + CAST(@varNumClientes AS VARCHAR(MAX))
```

The output window shows the results of the PRINT statements:

```
Número de Funcionários = 12  
Número de Clientes = 256
```

An orange arrow points from the bottom right towards the output window, highlighting the results.

# Armazenando um registro de uma consulta em uma variável

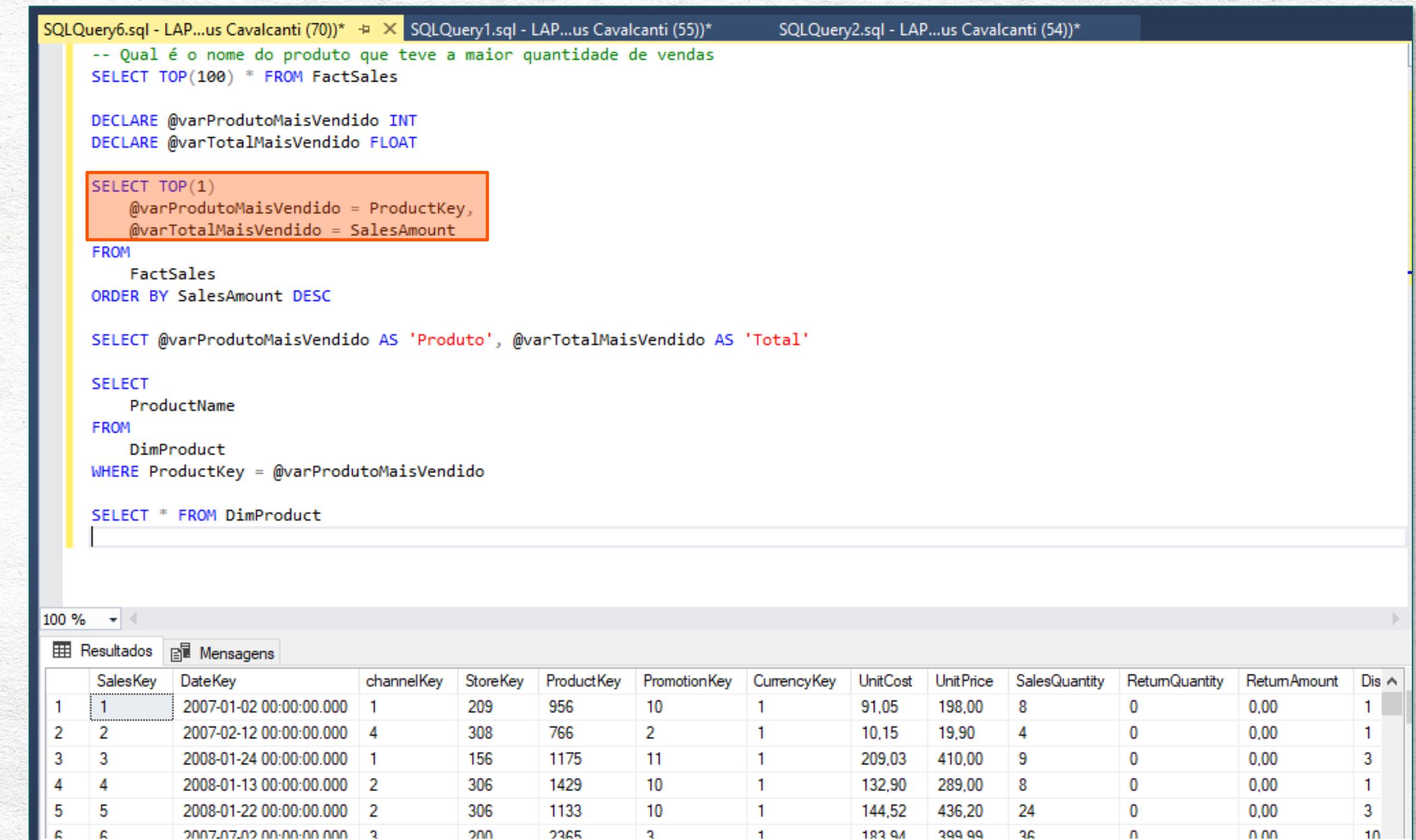
Variáveis também nos permitem armazenar determinados registros dentro de uma tabela.

Como assim registros?

Registros são as informações que temos dentro de uma tabela. Por meio de um filtro ou topn é possível acessar esses registros.

E pra fechar, conseguimos armazenar esses registros dentro de variáveis.

No exemplo ao lado, armazenamos dentro de duas variáveis o nome e o total vendido para o produto mais vendido. Utilizamos para isso o TOPN e o ORDER BY.



The screenshot shows a SQL Server Management Studio interface. The top bar has tabs for 'SQLQuery6.sql - LAP...us Cavalcanti (70)\*' (selected), 'SQLQuery1.sql - LAP...us Cavalcanti (55)\*', and 'SQLQuery2.sql - LAP...us Cavalcanti (54)\*'. The main area contains the following T-SQL code:

```

-- Qual é o nome do produto que teve a maior quantidade de vendas
SELECT TOP(100) * FROM FactSales

DECLARE @varProdutoMaisVendido INT
DECLARE @varTotalMaisVendido FLOAT

SELECT TOP(1)
    @varProdutoMaisVendido = ProductKey,
    @varTotalMaisVendido = SalesAmount
FROM
    FactSales
ORDER BY SalesAmount DESC

SELECT @varProdutoMaisVendido AS 'Produto', @varTotalMaisVendido AS 'Total'

SELECT
    ProductName
FROM
    DimProduct
WHERE ProductKey = @varProdutoMaisVendido

SELECT * FROM DimProduct

```

The bottom part of the screenshot shows a results grid titled 'Resultados' with the following data:

	SalesKey	DateKey	channelKey	StoreKey	ProductKey	PromotionKey	CurrencyKey	UnitCost	UnitPrice	SalesQuantity	RetunQuantity	RetunAmount	Dis
1	1	2007-01-02 00:00:00.000	1	209	956	10	1	91,05	198,00	8	0	0,00	1
2	2	2007-02-12 00:00:00.000	4	308	766	2	1	10,15	19,90	4	0	0,00	1
3	3	2008-01-24 00:00:00.000	1	156	1175	11	1	209,03	410,00	9	0	0,00	3
4	4	2008-01-13 00:00:00.000	2	306	1429	10	1	132,90	289,00	8	0	0,00	1
5	5	2008-01-22 00:00:00.000	2	306	1133	10	1	144,52	436,20	24	0	0,00	3
6	6	2007-07-02 00:00:00.000	2	200	2265	2	1	182,94	399,99	26	0	0,00	10

# Acumulando valores dentro de uma variável

Variáveis também podem acumular mais de um valor de uma vez.

Observe na imagem ao lado. Temos a variável `@ListaNomes` e queremos armazenar dentro dela todos os funcionários do departamento de Marketing, isso de uma só vez.

A estrutura para acumular os valores dentro da variável é mostrado ao lado.

The screenshot shows a SQL Server Management Studio window with three tabs at the top: 'SQLQuery6.sql - LAP...us Cavalcanti (70)\*', 'SQLQuery1.sql - LAP...us Cavalcanti (55)\*', and 'SQLQuery2.sql - LAP...us Cavalcanti (54)\*'. The main pane displays the following T-SQL script:

```
-- Retorne uma lista com os nomes dos funcionários do departamento de Marketing
DECLARE @ListaNomes VARCHAR(MAX)
SET @ListaNomes = ''

SELECT
    @ListaNomes = @ListaNomes + FirstName + ', ' + CHAR(10)
FROM
    DimEmployee
WHERE DepartmentName = 'Marketing'

PRINT LEFT(@ListaNomes, DATALENGTH(@ListaNomes) - 3)
```

Below the script, the 'Mensagens' (Messages) pane shows the output of the query:

```
Sagiv,
Pilar,
Aaron,
Robin,
Daniel,
Tanja,
Cliff,
Janet,
Cesar,
Mary
```

At the bottom of the messages pane, the timestamp is shown: 'Horário de conclusão: 2021-08-17T14:31:50.2817726-03:00'

# Variáveis globais

O SQL possui algumas variáveis padrão, que podem ser acessadas quando utilizamos o @@, como mostrado ao lado.

No código da imagem, conseguimos acessar variáveis que retornam informações como: Nome do Servidor, Versão do Sistema ou até mesmo a contagem de linhas de uma tabela.

The screenshot shows a SQL Server Management Studio window with three tabs at the top: SQLQuery6.sql - LAP...us Cavalcanti (70)\*, SQLQuery1.sql - LAP...us Cavalcanti (55)\*, and SQLQuery2.sql - LAP...us. The SQLQuery6.sql tab is active, displaying the following T-SQL code:

```
-- Variáveis globais
SELECT @@SERVERNAME AS 'Nome do Servidor'

SELECT @@VERSION AS 'Versão'

--SELECT * FROM DimProduct
SELECT @@ROWCOUNT AS 'Contagem de linhas'
```

The results pane below shows three rows of data corresponding to the three queries:

	Nome do Servidor
1	LAPTOP-HBTCI8PQ

	Versão
1	Microsoft SQL Server 2019 (RTM) - 15.0.2000.5 (X...)

	Contagem de linhas
1	1

# EXERCÍCIOS



1

## Questão 1

Declare 4 variáveis inteiros. Atribua os seguintes valores a elas:

valor1 = 10

valor2 = 5

valor3 = 34

valor4 = 7

- Crie uma nova variável para armazenar o resultado da soma entre valor1 e valor2. Chame essa variável de soma.
- Crie uma nova variável para armazenar o resultado da subtração entre valor3 e valor 4. Chame essa variável de subtracao.
- Crie uma nova variável para armazenar o resultado da multiplicação entre o valor 1 e o valor4. Chame essa variável de multiplicacao.
- Crie uma nova variável para armazenar o resultado da divisão do valor3 pelo valor4. Chame essa variável de divisao. Obs: O resultado deverá estar em decimal, e não em inteiro.

Arredonde o resultado da letra d) para 2 casas decimais.

# EXERCÍCIOS



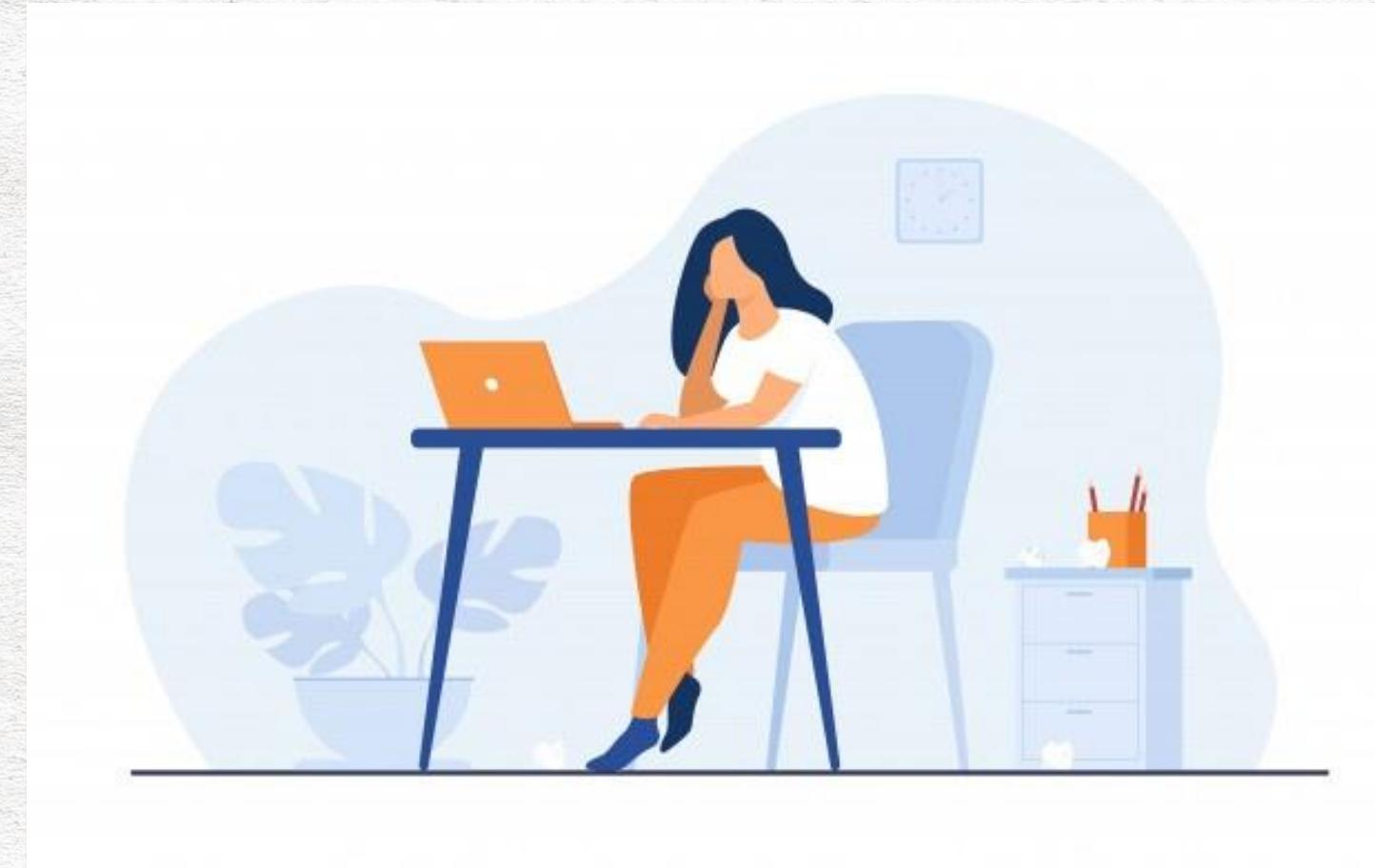
2

## Questão 2

Para cada declaração das variáveis abaixo, atenção em relação ao tipo de dado que deverá ser especificado.

- a. Declare uma variável chamada 'produto' e atribua o valor de 'Celular'.
- b. Declare uma variável chamada 'quantidade' e atribua o valor de 12.
- c. Declare uma variável chamada 'preco' e atribua o valor 9.99.
- d. Declare uma variável chamada 'faturamento' e atribua o resultado da multiplicação entre 'quantidade' e 'preco'.
- e. Visualize o resultado dessas 4 variáveis em uma única consulta, por meio do SELECT.

# EXERCÍCIOS



3

## Questão 3

Você é responsável por gerenciar um banco de dados onde são recebidos dados externos de usuários. Em resumo, esses dados são:

- Nome do usuário
- Data de nascimento
- Quantidade de pets que aquele usuário possui

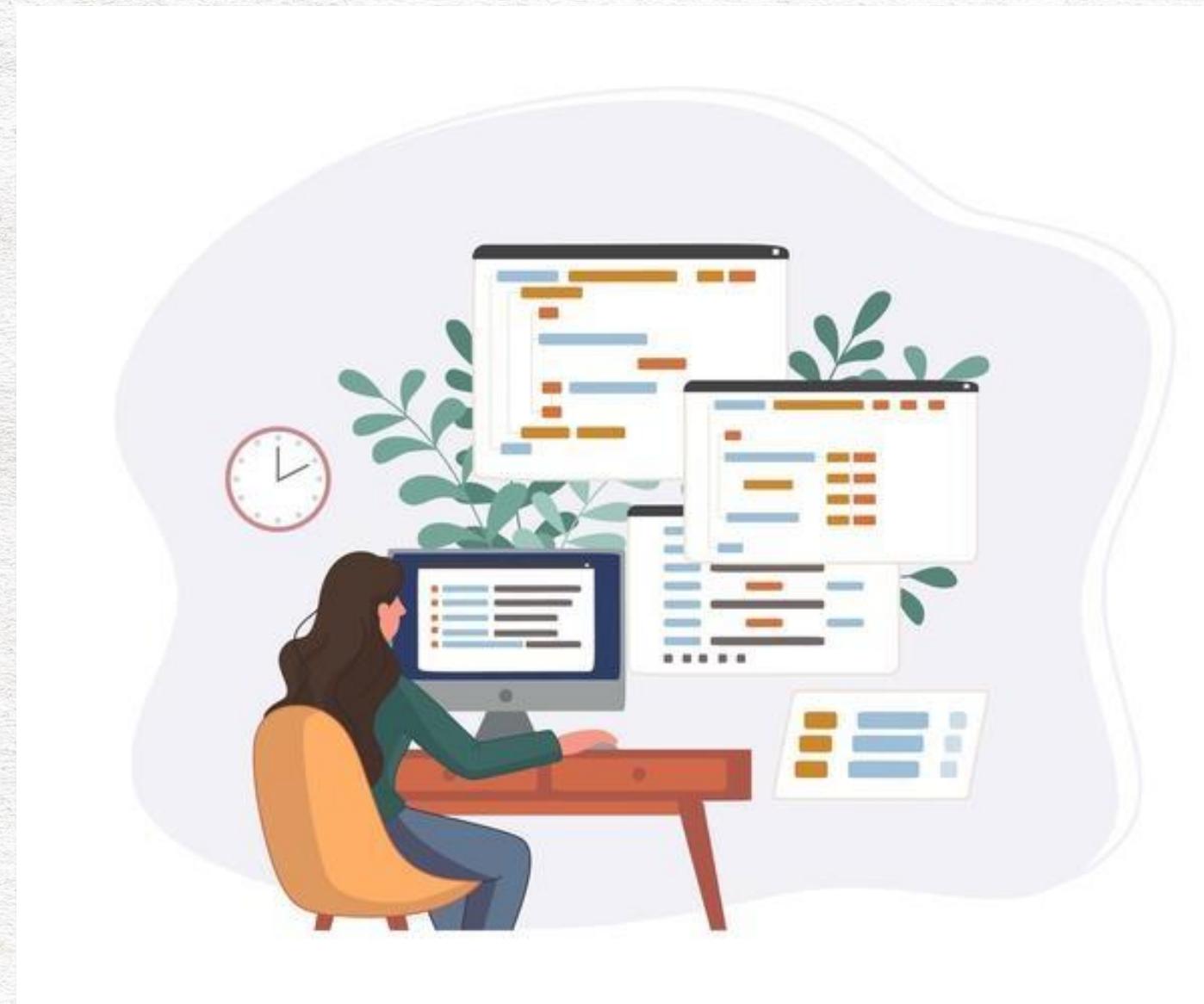
Você precisará criar um código em SQL capaz de juntar as informações fornecidas por este usuário. Para simular estes dados, crie 3 variáveis, chamadas: nome, data\_nascimento e num\_pets. Você deverá armazenar os valores 'André', '10/02/1998' e 2, respectivamente.

O resultado final a ser alcançado é mostrado no print abaixo:

(Nenhum nome de coluna)
Meu nome é André, nasci em 10/02/1998 e tenho 2 pets.

Dica: você precisará utilizar as funções CAST e FORMAT para chegar no resultado.

# EXERCÍCIOS



4

## Questão 4

Você acabou de ser promovido e o seu papel será realizar um controle de qualidade sobre as lojas da empresa.

A primeira informação que é passada a você é que o ano de 2008 foi bem complicado para a empresa, pois foi quando duas das principais lojas fecharam. O seu primeiro desafio é descobrir o nome dessas lojas que fecharam no ano de 2008, para que você possa entender o motivo e mapear planos de ação para evitar que outras lojas importantes tomem o mesmo caminho.

O seu resultado deverá estar estruturado em uma frase, com a seguinte estrutura:

*'As lojas fechadas no ano de 2008 foram: ' + nome\_das\_lojas*

Obs: utilize o comando PRINT (e não o SELECT!) para mostrar o resultado.

5

## Questão 5

Você precisa criar uma consulta para mostrar a lista de produtos da tabela DimProduct para uma subcategoria específica: 'Lamps'.

Utilize o conceito de variáveis para chegar neste resultado.

# GABARITOS

1

## Questão 1

Declare 4 variáveis inteiros. Atribua os seguintes valores a elas:

```
valor1 = 10
valor2 = 5
valor3 = 34
valor4 = 7
```

- Crie uma nova variável para armazenar o resultado da soma entre valor1 e valor2. Chame essa variável de soma.
- Crie uma nova variável para armazenar o resultado da subtração entre valor3 e valor 4. Chame essa variável de subtracao.
- Crie uma nova variável para armazenar o resultado da multiplicação entre o valor 1 e o valor4. Chame essa variável de multiplicacao.
- Crie uma nova variável para armazenar o resultado da divisão do valor3 pelo valor4. Chame essa variável de divisao. Obs: O resultado deverá estar em decimal, e não em inteiro.

Arredonde o resultado da letra d) para 2 casas decimais.

The screenshot shows a SQL query window titled "SQLQuery9.sql - LAP...us Cavalcanti (54)\*". The code is as follows:

```
--Questão 1
DECLARE @valor1 FLOAT
DECLARE @valor2 FLOAT
DECLARE @valor3 FLOAT
DECLARE @valor4 FLOAT
DECLARE @divisao FLOAT

SET @valor1 = 10.5
SET @valor2 = 5
SET @valor3 = 34
SET @valor4 = 7
SET @divisao = @valor3 / @valor4

SELECT ROUND(@divisao, 2)
```

The results pane shows a single row with the value 4,86.

# GABARITOS

2

## Questão 2

Para cada declaração das variáveis abaixo, atenção em relação ao tipo de dado que deverá ser especificado.

- a. Declare uma variável chamada 'produto' e atribua o valor de 'Celular'.
- b. Declare uma variável chamada 'quantidade' e atribua o valor de 12.
- c. Declare uma variável chamada 'preco' e atribua o valor 9.99.
- d. Declare uma variável chamada 'faturamento' e atribua o resultado da multiplicação entre 'quantidade' e 'preco'.
- e. Visualize o resultado dessas 4 variáveis em uma única consulta, por meio do SELECT.

```
--Questão 2
DECLARE @produto VARCHAR(30) = 'Celular'
DECLARE @quantidade INT = 12
DECLARE @preco FLOAT = 9.99
DECLARE @faturamento FLOAT = @quantidade * @preco

SELECT
    @produto AS 'Produto',
    @quantidade AS 'Quantidade',
    @preco AS 'Preço',
    @faturamento AS 'Faturamento'
```

	Produto	Quantidade	Preço	Faturamento
1	Celular	12	9,99	119,88

# GABARITOS

3

## Questão 3

Você é responsável por gerenciar um banco de dados onde são recebidos dados externos de usuários. Em resumo, esses dados são:

- Nome do usuário
- Data de nascimento
- Quantidade de pets que aquele usuário possui

Você precisará criar um código em SQL capaz de juntar as informações fornecidas por este usuário. Para simular estes dados, crie 3 variáveis, chamadas: nome, data\_nascimento e num\_pets. Você deverá armazenar os valores 'André', '10/02/1998' e 2, respectivamente.

O resultado final a ser alcançado é mostrado no print abaixo:

(Nenhum nome de coluna)

Meu nome é André, nasci em 10/02/1998 e tenho 2 pets.

Dica: você precisará utilizar as funções CAST e FORMAT para chegar no resultado.

```

SQLQuery9.sql - LAP...us Cavalcanti (54)* - Questão 3
-- Questão 3

DECLARE @nome VARCHAR(30),
        @nascimento DATETIME,
        @pets INT

SET @nome = 'André'
SET @nascimento = '10/02/1998'
SET @pets = 2

SELECT 'Meu nome é ' + CAST(@nome AS VARCHAR(30)) + ', nasci em ' + FORMAT(@nascimento, 'dd/MM/yyyy') + ' e tenho ' + CAST(@pets AS VARCHAR(30)) + ' pets.'

```

# GABARITOS

4

## Questão 4

Você acabou de ser promovido e o seu papel será realizar um controle de qualidade sobre as lojas da empresa.

A primeira informação que é passada a você é que o ano de 2008 foi bem complicado para a empresa, pois foi quando duas das principais lojas fecharam. O seu primeiro desafio é descobrir o nome dessas lojas que fecharam no ano de 2008, para que você possa entender o motivo e mapear planos de ação para evitar que outras lojas importantes tomem o mesmo caminho.

O seu resultado deverá estar estruturado em uma frase, com a seguinte estrutura:

'As lojas fechadas no ano de 2008 foram: ' + nome\_das\_lojas

Obs: utilize o comando PRINT (e não o SELECT!) para mostrar o resultado.

The screenshot shows a SQL query window titled 'SQLQuery9.sql - LAP...us Cavalcanti (54)\*'. The query is as follows:

```
--Questão 4
DECLARE @nome_loja VARCHAR(100)
SET @nome_loja = ''

SELECT
    @nome_loja = @nome_loja + StoreName + ', '
FROM DimStore
WHERE FORMAT(CloseDate, 'yyyy') = 2008

PRINT 'As lojas fechadas no ano de 2008 foram: ' + @nome_loja
```

The results pane shows the output of the query:

Mensagens E Estatísticas do Cliente  
As lojas fechadas no ano de 2008 foram: Contoso Buffalo Store, Contoso Trenton No.2 Store,  
Horário de conclusão: 2021-08-18T03:09:55.5080488-03:00

# GABARITOS

5

## Questão 5

Você precisa criar uma consulta para mostrar a lista de produtos da tabela DimProduct para uma subcategoria específica: 'Lamps'.

Utilize o conceito de variáveis para chegar neste resultado.

The screenshot shows a SQL query window titled "SQLQuery9.sql - LAP...us Cavalcanti (54)\*". The query is as follows:

```
--Questão 5
SELECT * FROM DimProductSubcategory
DECLARE @id_subcategoria INT
DECLARE @subcategoria VARCHAR(100)

SET @subcategoria = 'Radio'
SET @id_subcategoria = (SELECT ProductSubcategoryKey FROM DimProductSubcategory WHERE ProductSubcategoryName = @subcategoria)

SELECT *
FROM
    DimProduct
WHERE ProductSubcategoryKey = @id_subcategoria
```

The results grid displays the following data:

	ProductSubcategoryKey	ProductSubcategoryLabel	ProductSubcategoryName	ProductSubcategoryDescription	ProductCategoryKey	ETLLoadID	LoadDate	UpdateC
1	1	0101	MP4&MP3	MP4&MP3	1	1	2009-07-07 00:00:00.000	2009-07
2	2	0102	Recorder	Recorder	1	1	2009-07-07 00:00:00.000	2009-07
3	3	0103	Radio	Radio	1	1	2009-07-07 00:00:00.000	2009-07
4	4	0104	Recording Pen	Recording Pen	1	1	2009-07-07 00:00:00.000	2009-07
5	5	0105	Headphones	Headphones	1	1	2009-07-07 00:00:00.000	2009-07
6	6	0106	Bluetooth Headphones	Bluetooth Headphones	1	1	2009-07-07 00:00:00.000	2009-07
7	7	0107	Speakers	Speakers	1	1	2009-07-07 00:00:00.000	2009-07
8	8	0108	Audio Accessories	Audio Accessories	1	1	2009-07-07 00:00:00.000	2009-07

At the bottom of the results grid, a message states: "Consulta executada com êxito." (Query executed successfully.)

# SQL

## Manipulando Strings e Datas no SQL

# LEN e DATALENGTH

As funções **LEN** e **DATALENGTH** possuem resultados semelhantes: ambos retornam a quantidade de caracteres de um determinado texto.

Ao lado, é possível visualizar um exemplo.

A grande diferença entre os dois é sutil: o **LEN** retorna a quantidade de caracteres de um texto, porém ignora quando aparecem espaços a mais.

Já o **DATALENGTH** retorna a quantidade de caracteres, incluindo espaços a mais dentro do texto.

The screenshot shows a SQL Server Management Studio window with three panes. The top pane contains comments explaining the difference between **LEN** and **DATALENGTH**. The middle pane contains a **SELECT** statement comparing the two functions on the string 'SQL Hashtag '. The bottom pane shows the results in a grid:

	Len	DataLength
1	11	14

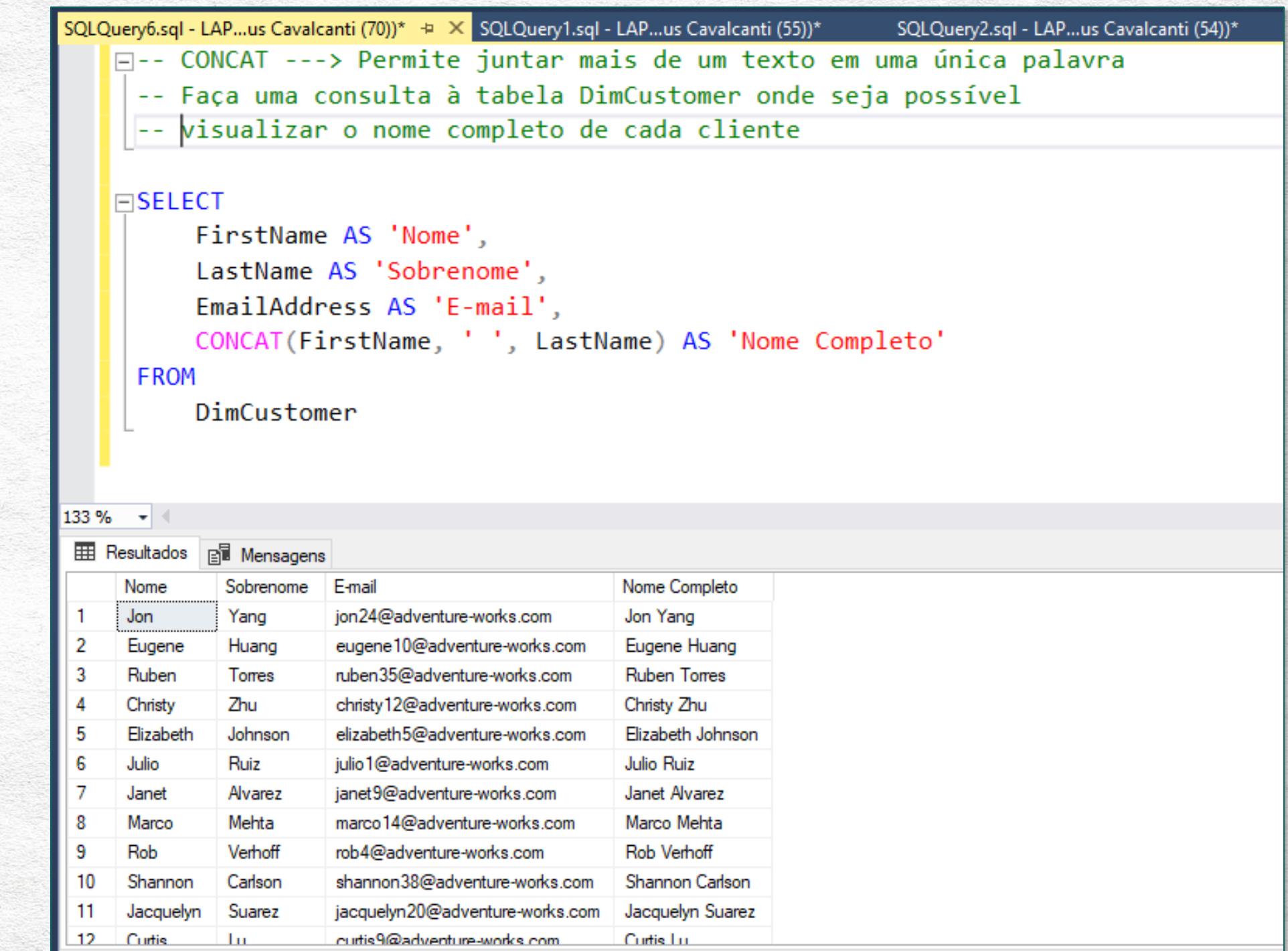
A red arrow points from the cell containing '11' in the 'Len' column to the cell containing '11' in the 'DataLength' column, highlighting the difference in how they handle spaces.

# CONCAT

A função **CONCAT** permite juntar dois ou mais textos em um só.

Observe a imagem ao lado. A partir das colunas de Nome e Sobrenome da nossa tabela DimCustomer, podemos criar uma nova coluna, chamada Nome Completo, que é a junção dos dois textos: FirstName e LastName.

Para isso, usamos a função **CONTAT**.



The screenshot shows a SQL Server Management Studio window with three tabs at the top: 'SQLQuery6.sql - LAP...us Cavalcanti (70)\*' (selected), 'SQLQuery1.sql - LAP...us Cavalcanti (55)\*', and 'SQLQuery2.sql - LAP...us Cavalcanti (54)\*'. The selected query contains the following code:

```
-- CONCAT ---> Permite juntar mais de um texto em uma única palavra
-- Faça uma consulta à tabela DimCustomer onde seja possível
-- visualizar o nome completo de cada cliente

SELECT
    FirstName AS 'Nome',
    LastName AS 'Sobrenome',
    EmailAddress AS 'E-mail',
    CONCAT(FirstName, ' ', LastName) AS 'Nome Completo'
FROM
    DimCustomer
```

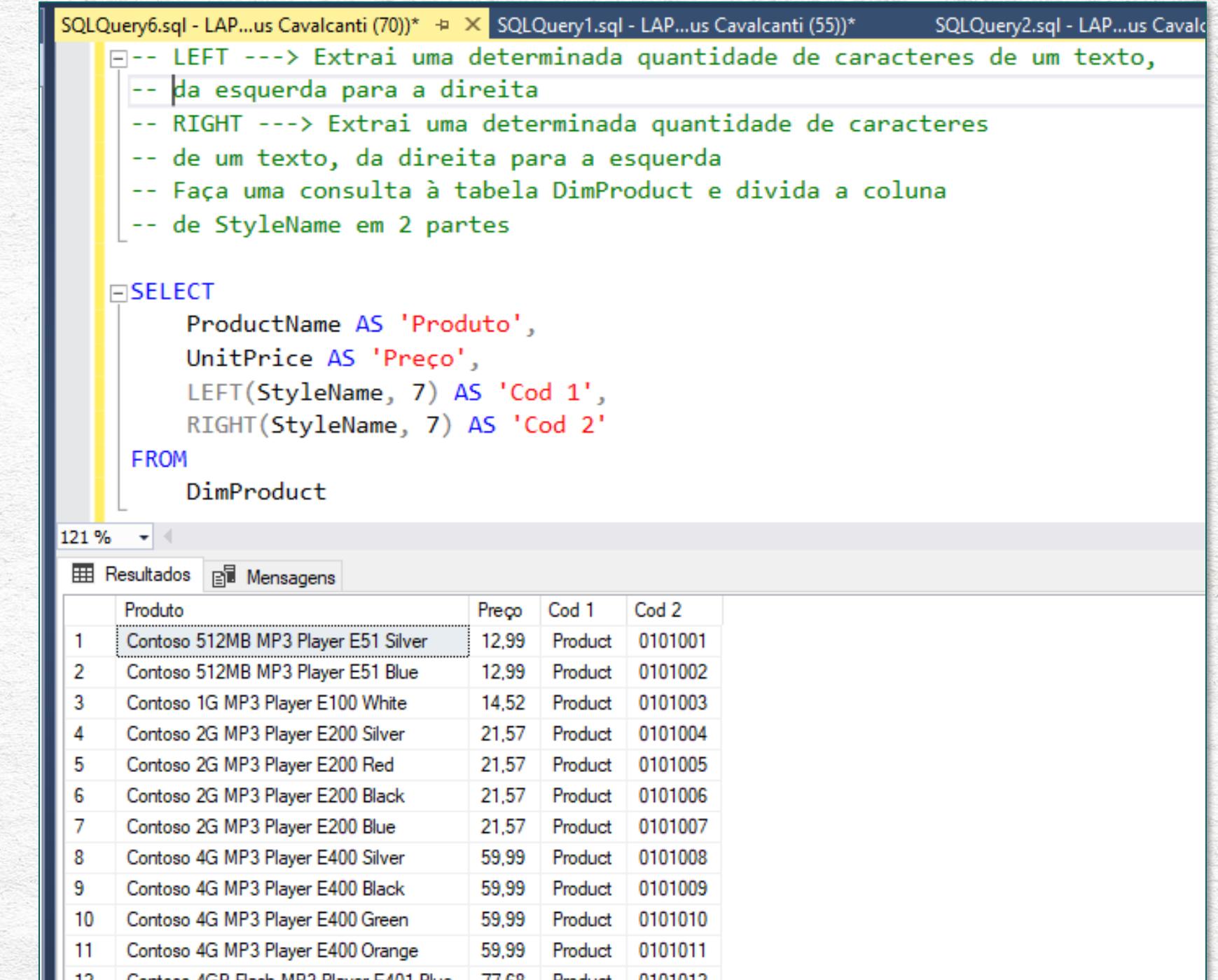
The results tab displays the following table:

	Nome	Sobrenome	E-mail	Nome Completo
1	Jon	Yang	jon24@adventure-works.com	Jon Yang
2	Eugene	Huang	eugene10@adventure-works.com	Eugene Huang
3	Ruben	Torres	ruben35@adventure-works.com	Ruben Torres
4	Christy	Zhu	christy12@adventure-works.com	Christy Zhu
5	Elizabeth	Johnson	elizabeth5@adventure-works.com	Elizabeth Johnson
6	Julio	Ruiz	julio1@adventure-works.com	Julio Ruiz
7	Janet	Alvarez	janet9@adventure-works.com	Janet Alvarez
8	Marco	Mehta	marco14@adventure-works.com	Marco Mehta
9	Rob	Verhoff	rob4@adventure-works.com	Rob Verhoff
10	Shannon	Carlson	shannon38@adventure-works.com	Shannon Carlson
11	Jacquelyn	Suarez	jacquelyn20@adventure-works.com	Jacquelyn Suarez
12	Curtis	Iu	curtis9@adventure-works.com	Curtis Iu

# LEFT e RIGHT

As funções **LEFT** e **RIGHT** permitem extrair uma parte de um texto, sempre começando da esquerda pra direita, ou da direita pra esquerda, respectivamente.

Na imagem ao lado, temos um exemplo prático de como aplicar essas funções.



The screenshot shows a SQL Server Management Studio window with three tabs: SQLQuery6.sql, SQLQuery1.sql, and SQLQuery2.sql. The SQLQuery6.sql tab is active, displaying the following code:

```
-- LEFT ---> Extrai uma determinada quantidade de caracteres de um texto,
-- da esquerda para a direita
-- RIGHT ---> Extrai uma determinada quantidade de caracteres
-- de um texto, da direita para a esquerda
-- Faça uma consulta à tabela DimProduct e divida a coluna
-- de StyleName em 2 partes

SELECT
    ProductName AS 'Produto',
    UnitPrice AS 'Preço',
    LEFT(StyleName, 7) AS 'Cod 1',
    RIGHT(StyleName, 7) AS 'Cod 2'
FROM
    DimProduct
```

The results tab shows the output of the query:

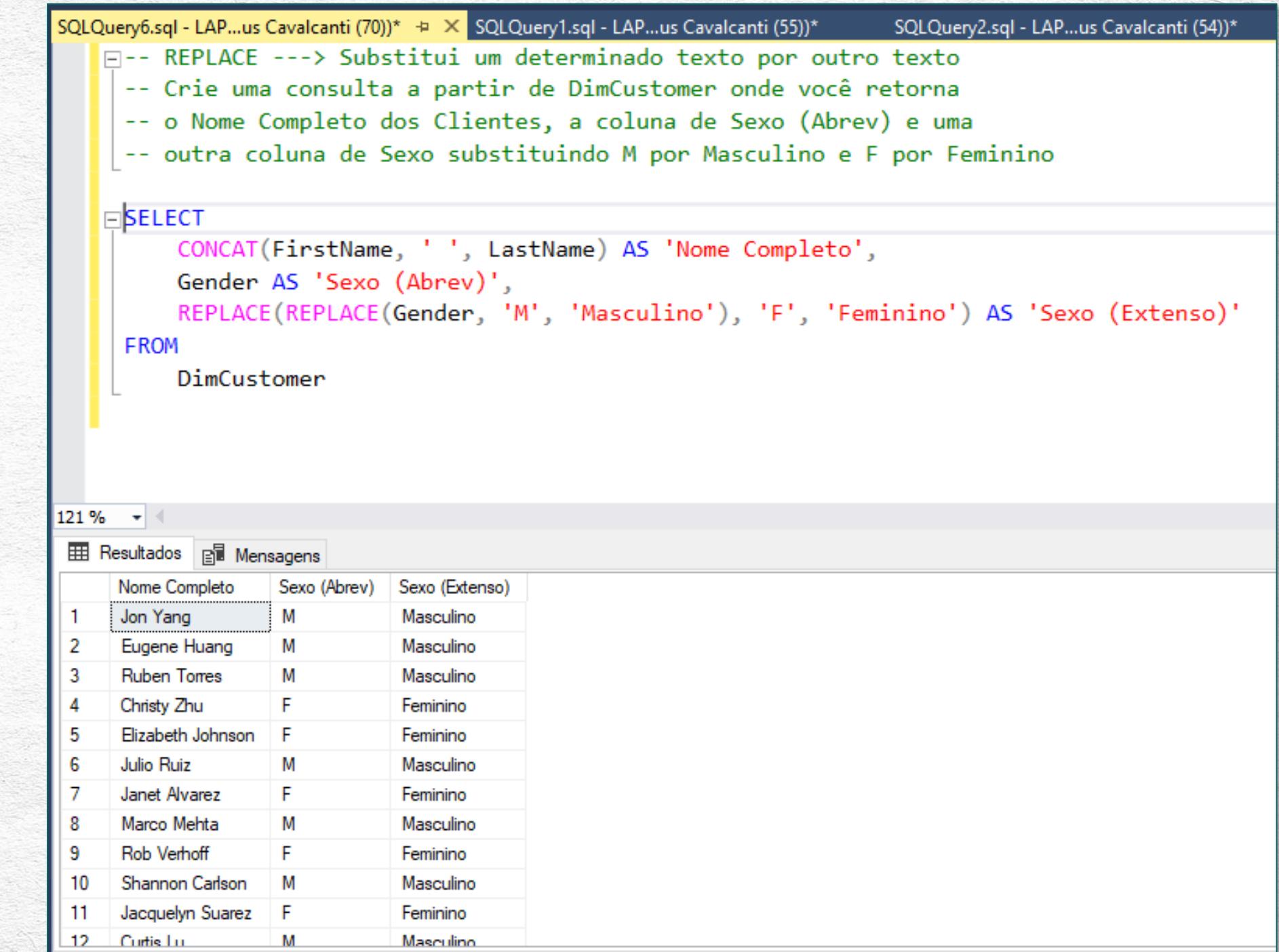
	Produto	Preço	Cod 1	Cod 2
1	Contoso 512MB MP3 Player E51 Silver	12,99	Product	0101001
2	Contoso 512MB MP3 Player E51 Blue	12,99	Product	0101002
3	Contoso 1G MP3 Player E100 White	14,52	Product	0101003
4	Contoso 2G MP3 Player E200 Silver	21,57	Product	0101004
5	Contoso 2G MP3 Player E200 Red	21,57	Product	0101005
6	Contoso 2G MP3 Player E200 Black	21,57	Product	0101006
7	Contoso 2G MP3 Player E200 Blue	21,57	Product	0101007
8	Contoso 4G MP3 Player E400 Silver	59,99	Product	0101008
9	Contoso 4G MP3 Player E400 Black	59,99	Product	0101009
10	Contoso 4G MP3 Player E400 Green	59,99	Product	0101010
11	Contoso 4G MP3 Player E400 Orange	59,99	Product	0101011
12	Contoso 4GB Flash MP3 Player E401 Blue	77,68	Product	0101012

# REPLACE

A função **REPLACE** permite que a gente substitua um determinado texto por outro texto.

Na imagem ao lado, vemos uma aplicação prática do REPLACE.

Nela, substituímos os textos M e F por Masculino e Feminino, respectivamente.



The screenshot shows a SQL Server Management Studio window with three tabs at the top: 'SQLQuery6.sql - LAP...us Cavalcanti (70)\*', 'SQLQuery1.sql - LAP...us Cavalcanti (55)\*', and 'SQLQuery2.sql - LAP...us Cavalcanti (54)\*'. The main pane displays a SQL query:

```

-- REPLACE ---> Substitui um determinado texto por outro texto
-- Crie uma consulta a partir de DimCustomer onde você retorna
-- o Nome Completo dos Clientes, a coluna de Sexo (Abrev) e uma
-- outra coluna de Sexo substituindo M por Masculino e F por Feminino

SELECT
    CONCAT(FirstName, ' ', LastName) AS 'Nome Completo',
    Gender AS 'Sexo (Abrev)',
    REPLACE(REPLACE(Gender, 'M', 'Masculino'), 'F', 'Feminino') AS 'Sexo (Extenso)'
FROM
    DimCustomer
  
```

The results pane shows the output of the query:

	Nome Completo	Sexo (Abrev)	Sexo (Extenso)
1	Jon Yang	M	Masculino
2	Eugene Huang	M	Masculino
3	Ruben Torres	M	Masculino
4	Christy Zhu	F	Feminino
5	Elizabeth Johnson	F	Feminino
6	Julio Ruiz	M	Masculino
7	Janet Alvarez	F	Feminino
8	Marco Mehta	M	Masculino
9	Rob Verhoff	F	Feminino
10	Shannon Carlson	M	Masculino
11	Jacquelyn Suarez	F	Feminino
12	Curtis Lu	M	Masculino

# TRANSLATE e STUFF

Existem outras duas funções usadas para substituir textos:  
**TRANSLATE** e **STUFF**.

A função **TRANSLATE** permite que a gente informa uma sequência de caracteres antigos que desejamos substituir, além da sequência de caracteres novos que serão usados para substituir os caracteres antigos, seguindo exatamente a mesma sequência.

Observe os exemplos ao lado. Essa função pede 3 argumentos:

1. Texto
2. Sequência antiga de caracteres
3. Sequência nova de caracteres.

Na primeira aplicação do **TRANSLATE** (Translate 1), todos os caracteres [ são substituídos por (, todos os caracteres ] são substituídos pelo ), e assim vai. Exatamente nessa ordem.

Já no exemplo Translate 2, sempre que a letra A aparecer, vamos substituir por W. Sempre que aparecer a letra B, vamos substituir por X, e assim vai.

Em resumo, essa substituição é feita caractere por caractere, exatamente na ordem especificada.

```

SQLQuery6.sql - LAP...us Cavalcanti (70)*  SQLQuery1.sql - LAP...us Cavalcanti (55)*  SQLQuery2.sql - LAP...us Cavalcanti (54)*
-- TRANSLATE e STUFF: Outras funções de substituição

SELECT TRANSLATE('3*[2+1]/{8-4}', '[]{}', '()()') AS 'Translate 1'
SELECT TRANSLATE('ABCD-490123', 'ABCD', 'WXYZ') AS 'Translate 2'
SELECT STUFF('VBA Impressionador', 1, 3, 'Excel') AS 'Stuff'

Resultados
Translate 1
1 3*(2+1)/(8-4)

Translate 2
1 WXYZ-490123

Stuff
1 Excel Impressionador
  
```

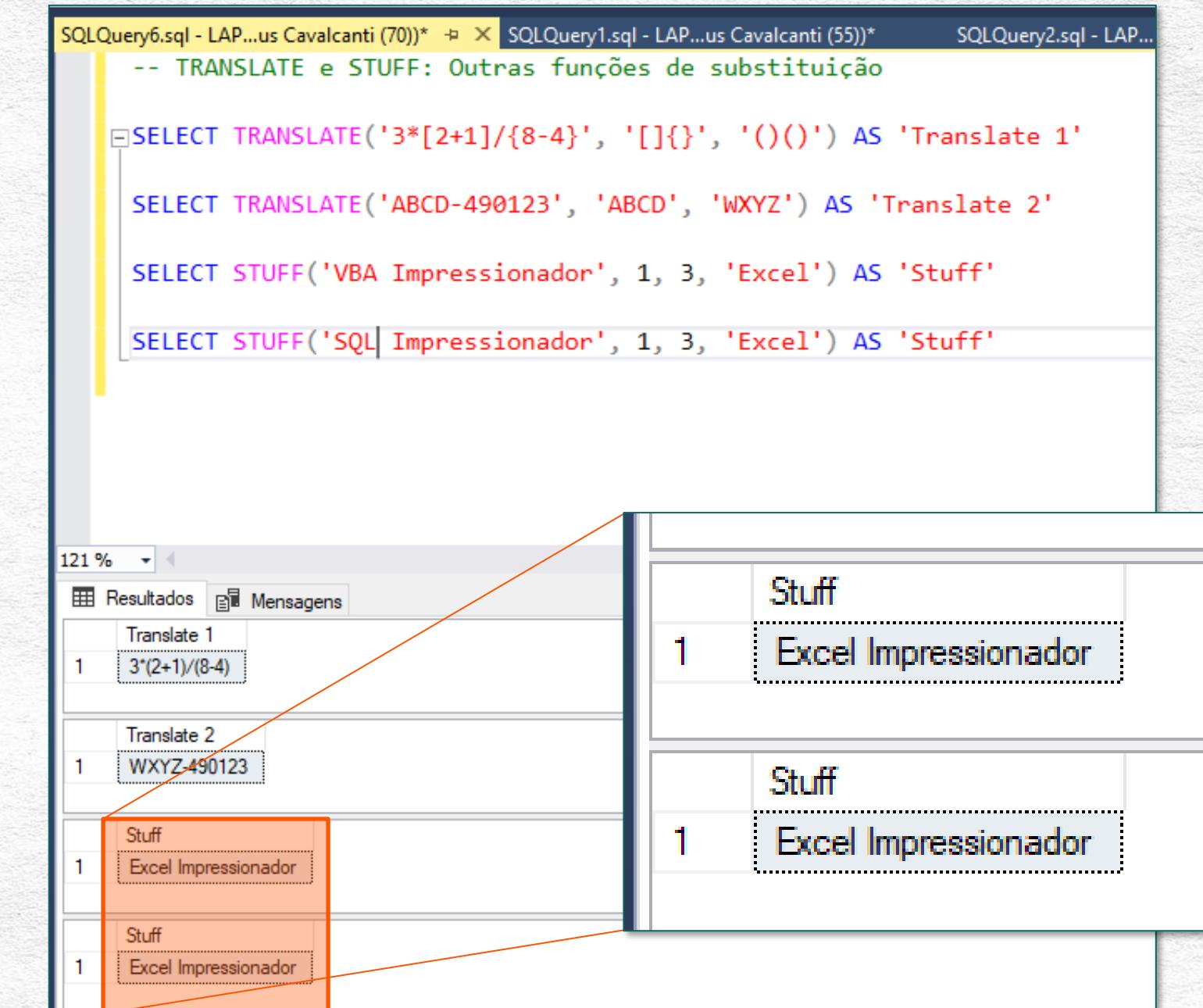
# TRANSLATE e STUFF

A função **STUFF** permite que a gente substitua uma sequência de caracteres com um tamanho específico, por um outro texto.

Essa função pede 4 argumentos:

1. Texto
2. Posição inicial onde começa o texto antigo
3. Quantidade de caracteres do texto antigo
4. Novo texto

No exemplo ao lado, temos duas situações: VBA Impressionador e SQL Impressionador. Ambos tem 3 caracteres iniciais (referentes ao nome do curso) e podemos utilizar esse padrão, junto com a função STUFF, para substituir sempre os 3 primeiros caracteres (independente do que esteja escrito nesses 3 primeiros caracteres) pelo texto 'Excel'.



```

SQLQuery6.sql - LAP...us Cavalcanti (70)*  SQLQuery1.sql - LAP...us Cavalcanti (55)*  SQLQuery2.sql - LAP...
-- TRANSLATE e STUFF: Outras funções de substituição

SELECT TRANSLATE('3*[2+1]/{8-4}', '[]{}', '()()') AS 'Translate 1'

SELECT TRANSLATE('ABCD-490123', 'ABCD', 'WXYZ') AS 'Translate 2'

SELECT STUFF('VBA Impressionador', 1, 3, 'Excel') AS 'Stuff'

SELECT STUFF('SQL| Impressionador', 1, 3, 'Excel') AS 'Stuff'
  
```

The screenshot shows the SQL Server Management Studio interface. At the top, there are three tabs: 'SQLQuery6.sql - LAP...us Cavalcanti (70)\*', 'SQLQuery1.sql - LAP...us Cavalcanti (55)\*', and 'SQLQuery2.sql - LAP...'. Below the tabs, a comment '-- TRANSLATE e STUFF: Outras funções de substituição' is present. Four SQL queries are listed, each with an alias in blue: 'Translate 1', 'Translate 2', 'Stuff', and 'Stuff'. The 'Results' tab is selected, displaying the execution results for each query. The results are as follows:

	Translate 1	Translate 2	Stuff	Stuff
1	<code>3*(2+1)/(8-4)</code>	<code>WXYZ-490123</code>	<code>Excel Impressionador</code>	<code>Excel Impressionador</code>

# UPPER e LOWER

As funções UPPER e LOWER são bastante intuitivas e nos permitem ajustar as palavras para que fiquem em letras maiúsculas e minúsculas.

Ao lado, temos uma aplicação dessas funções.

```

SQLQuery6.sql - LAP...us Cavalcanti (70)* ✎ SQLQuery1.sql - LAP...us Cavalcanti (55)* SQLQuery2.sql - LAP...us Cavalcanti (54)*
-- UPPER ---> Transforma um texto em maiúscula
-- LOWER ---> Transforma um texto em minúscula
-- Faça uma consulta à tabela DimCustomer e utilize as funções
-- UPPER e LOWER na coluna de FirstName para observar o resultado

SELECT
    UPPER(FirstName) AS 'NOME',
    LOWER(FirstName) AS 'nome',
    EmailAddress AS 'E-mail'
FROM
    DimCustomer

```

121 %

	NOME	nome	E-mail
1	JON	jon	jon24@adventure-works.com
2	EUGENE	eugene	eugene10@adventure-works.com
3	RUBEN	ruben	ruben35@adventure-works.com
4	CHRISTY	christy	christy12@adventure-works.com
5	ELIZABETH	elizabeth	elizabeth5@adventure-works.com
6	JULIO	julio	julio1@adventure-works.com
7	JANET	janet	janet9@adventure-works.com
8	MARCO	marco	marco14@adventure-works.com
9	ROB	rob	rob4@adventure-works.com
10	SHANNON	shannon	shannon38@adventure-works.com
11	JACQUELYN	jacquelyn	jacquelyn20@adventure-works.com
12	CURTIS	curtis	curtis9@adventure-works.com

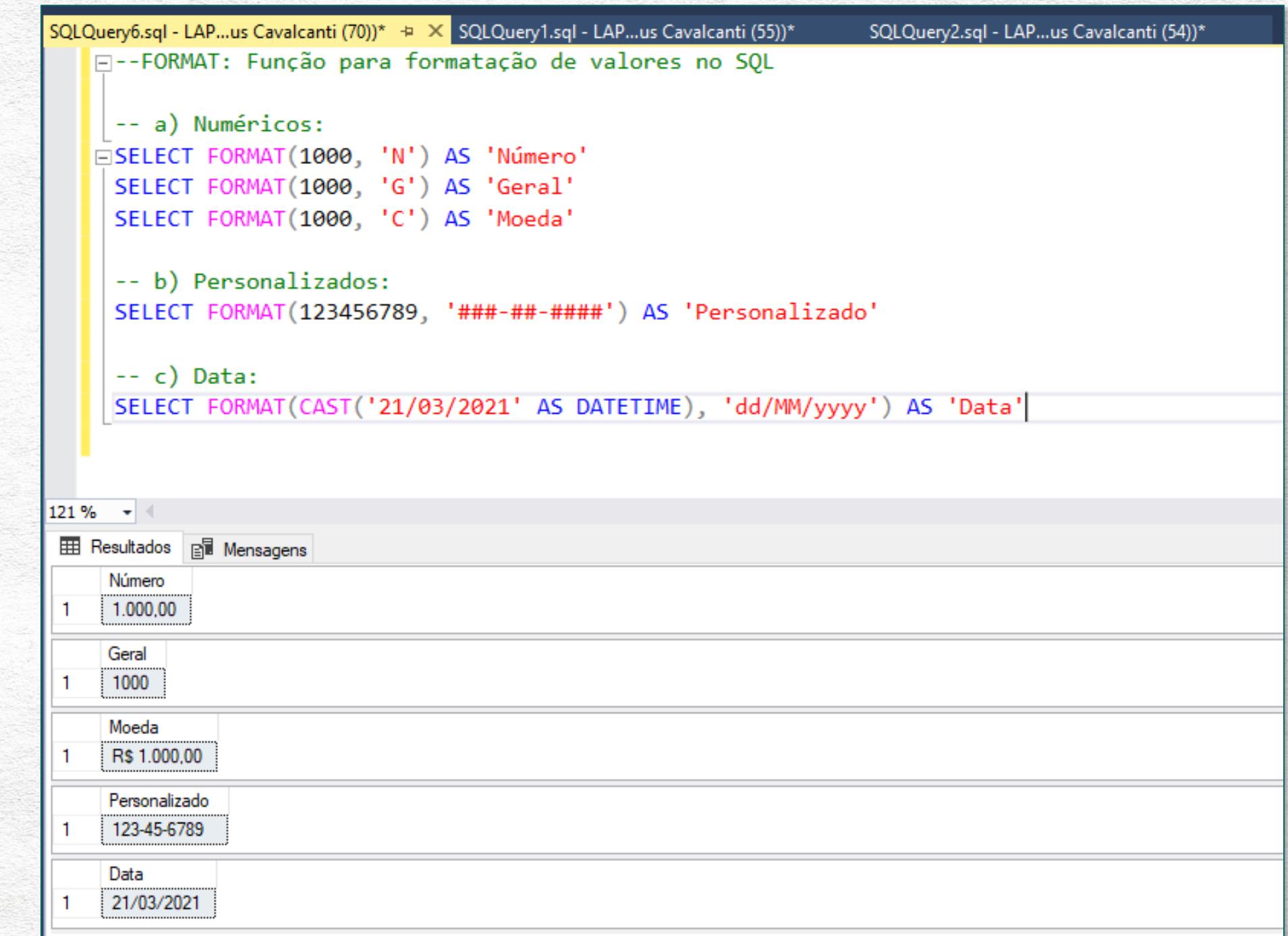
# FORMAT

A função **FORMAT** é uma função muito útil para realizar formatações personalizadas em nossas consultas.

Essa função nos pede 2 argumentos:

1. Valor a ser formatado
2. Código de formatação

As principais formações são mostradas ao lado.



The screenshot shows three tabs in SSMS: SQLQuery6.sql, SQLQuery1.sql, and SQLQuery2.sql. The code in SQLQuery6.sql demonstrates the FORMAT function with three examples: Número (1000), Geral (1000), and Moeda (R\$ 1.000,00). The code in SQLQuery1.sql shows how to format a date (21/03/2021) as dd/MM/yyyy. The code in SQLQuery2.sql shows how to format a number (123456789) as a personalized string ('###-##-####'). The results pane shows the output for each query.

```

SQLQuery6.sql - LAP...us Cavalcanti (70)*  SQLQuery1.sql - LAP...us Cavalcanti (55)*  SQLQuery2.sql - LAP...us Cavalcanti (54)*
--FORMAT: Função para formatação de valores no SQL
-- a) Numéricos:
SELECT FORMAT(1000, 'N') AS 'Número'
SELECT FORMAT(1000, 'G') AS 'Geral'
SELECT FORMAT(1000, 'C') AS 'Moeda'

-- b) Personalizados:
SELECT FORMAT(123456789, '###-##-####') AS 'Personalizado'

-- c) Data:
SELECT FORMAT(CAST('21/03/2021' AS DATETIME), 'dd/MM/yyyy') AS 'Data'

121 %  Resultados  Mensagens
Número
1 1.000,00
Geral
1 1000
Moeda
1 R$ 1.000,00
Personalizado
1 123-45-6789
Data
1 21/03/2021

```

# CHARINDEX e SUBSTRING

As funções **CHARINDEX** e **SUBSTRING** combinadas são muito úteis para manipular textos de forma mais avançada.

A função **CHARINDEX** nos permite descobrir a posição de uma determinada cadeira de caracteres dentro de um texto. Ela pede 2 argumentos:

1. Texto a ser procurado
2. Texto completo

No exemplo ao lado, vemos que o texto ‘Moreno’ encontra-se na posição 8 do nome ‘Raquel Moreno’.

Já a função **SUBSTRING** nos permite retornar uma cadeia de caracteres de dentro de um texto, a partir de uma posição inicial.

Essa função nos pede 3 argumentos:

1. Texto
2. Posição inicial do texto a ser extraído
3. Quantidade de caracteres a ser extraído

Na imagem ao lado, temos exemplos algumas aplicações dessas duas funções.

The screenshot shows three tabs in the SSMS interface: 'SQLQuery6.sql - LAP...us Cavalcanti (70)\*' (active), 'SQLQuery1.sql - LAP...us Cavalcanti (55)\*', and 'SQLQuery2.sql - LAP...us Cavalcanti (54)\*'. The active query window contains the following code:

```

SELECT 'Raquel Moreno' AS 'Nome'
SELECT CHARINDEX('Moreno', 'Raquel Moreno') AS 'Posição Sobrenome'
SELECT SUBSTRING('Raquel Moreno', 8, 6) AS 'Sobrenome'

```

The results pane displays the output of these queries:

	Nome
1	Raquel Moreno

	Posição Sobrenome
1	8

	Sobrenome
1	Moreno

# TRIM, LTRIM e RTRIM

As funções **TRIM** e **LTRIM** e **RTRIM** são funções que nos permitem retirar espaços adicionais de um texto.

A função **TRIM** retira espaços adicionais à esquerda e à direita do texto.

A função **LTRIM** retira espaços adicionais à esquerda do texto.

A função **RTRIM** retira espaços adicionais à direita do texto.

Visualmente, parece que não há diferença na aplicação das 3 funções. Porém, se utilizarmos a função **DATALENGTH** sobre cada um dos resultados, vemos a diferença.

```

SQLQuery6.sql - LAP...us Cavalcanti (70)*  X SQLQuery1.sql - LAP...us Cavalcanti (55)*
SELECT
    DATALENGTH(TRIM(@varCodigo)) AS 'Trim',
    DATALENGTH(LTRIM(@varCodigo)) AS 'Ltrim',
    DATALENGTH(RTRIM(@varCodigo)) AS 'Rtrim'
  
```

	Trim	Ltrim	Rtrim
1	ABC123	ABC123	ABC123

	Trim	Ltrim	Rtrim
1	6	10	9

```

SQLQuery6.sql - LAP...us Cavalcanti (70)*  X SQLQuery1.sql - LAP...us Cavalcanti (55)*      SQLQuery2.sql - LAP...us Cavalcanti (55)*
-- Funções para retirar espaços adicionais dentro de um texto
-- TRIM, LTRIM e RTRIM

DECLARE @varCodigo VARCHAR(50)
SET @varCodigo = ' ABC123 '

SELECT
    TRIM(@varCodigo) AS 'Trim',
    LTRIM(@varCodigo) AS 'Ltrim',
    RTRIM(@varCodigo) AS 'Rtrim'
  
```

	Trim	Ltrim	Rtrim
1	ABC123	ABC123	ABC123

# DAY, MONTH e YEAR

As funções **DAY** e **MONTH** e **YEAR** são funções que nos permitem descobrir o dia, mês e ano de uma data.

Ao lado, temos alguns exemplos do funcionamento dessas funções.

The screenshot shows a SQL Server Management Studio window with three tabs at the top: 'SQLQuery6.sql - LAP...us Cavalcanti (70)\*' (selected), 'SQLQuery1.sql - LAP...us Cavalcanti (55)\*', and 'SQLQuery2.sql - LAP...us Cavalcanti (54)\*'. The main area contains the following SQL code:

```
-- Utilize as funções DAY, MONTH e YEAR  
-- para descobrir o dia, mês e ano da data: 18/05/2020  
  
DECLARE @varData DATETIME  
SET @varData = '18/05/2020'  
  
SELECT  
    DAY(@varData) AS 'Dia',  
    MONTH(@varData) AS 'Mês',  
    YEAR(@varData) AS 'Ano'
```

The results pane shows a single row of data:

	Dia	Mês	Ano
1	18	5	2020

# DATEFROMPARTS

A função **DATEFROMPARTS** é uma função que permite retornar uma data completa a partir das informações separadas de dia, mês e ano.

The screenshot shows a SQL Server Management Studio window with three tabs at the top: SQLQuery6.sql - LAP...us Cavalcanti (70)\*, SQLQuery1.sql - LAP...us Cavalcanti (55)\*, and SQLQuery2.sql - LAP...us Cavalcanti (54)\*. The middle tab is active. The query in the editor is:

```
-- Utilize a função DATEFROMPARTS para retornar uma data  
-- a partir do seu dia, mês e ano  
  
DECLARE @varDia INT, @varMes INT, @varAno INT  
SET @varDia = 29  
SET @varMes = 12  
SET @varAno = 2020  
  
SELECT  
    DATEFROMPARTS(@varAno, @varMes, @varDia) AS 'Data'
```

The results pane below shows a single row with the column 'Data' containing the value '2020-12-29'. The results tab is selected.

# GETDATE e SYSDATETIME

As funções **GETDATE** e **SYSDATETIME** nos permitem retornar a data e hora atual do sistema.

A diferença entre elas é que a **SYSDATETIME** tem uma maior precisão em relação à **GETDATE**.

SQLQuery6.sql - LAP...us Cavalcanti (70)\*

```
SELECT GETDATE()
```

146 %

	(Nenhum nome de coluna)
1	2021-08-17 16:05:51.280

SQLQuery6.sql - LAP...us Cavalcanti (70)\*

```
SELECT SYSDATETIME()
```

146 %

	(Nenhum nome de coluna)
1	2021-08-17 16:06:30.3650970

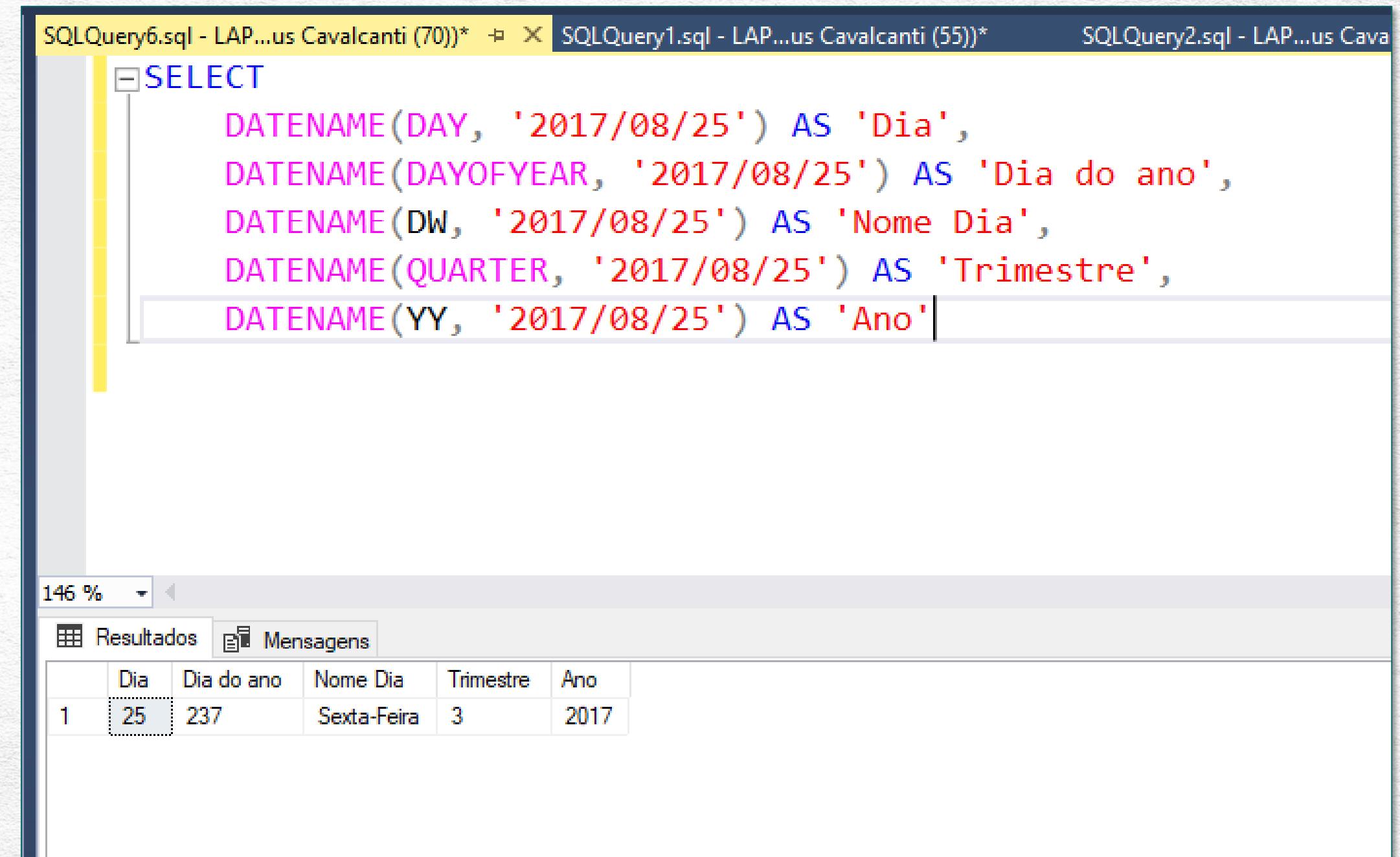
# DATENAME e DATEPART

As funções **DATENAME** e **DATEPART** nos retornam informações de uma data, como o dia, mês, ano, trimestre, e assim vai.

As duas são usadas da mesma forma, com 2 argumentos:

1. Informação que queremos da data
2. Data

A diferença é que o resultado da função DATENAME é um valor do tipo texto (varchar) enquanto o resultado da função DATEPART é do tipo inteiro.



```
SQLQuery6.sql - LAP...us Cavalcanti (70)*  X SQLQuery1.sql - LAP...us Cavalcanti (55)*      SQLQuery2.sql - LAP...us Cava
SELECT
    DATENAME(DAY, '2017/08/25') AS 'Dia',
    DATENAME(DAYOFYEAR, '2017/08/25') AS 'Dia do ano',
    DATENAME(DW, '2017/08/25') AS 'Nome Dia',
    DATENAME(QUARTER, '2017/08/25') AS 'Trimestre',
    DATENAME(YY, '2017/08/25') AS 'Ano'|
```

The screenshot shows a SQL Server Management Studio interface. In the top tab bar, there are three tabs: 'SQLQuery6.sql - LAP...us Cavalcanti (70)\*', 'SQLQuery1.sql - LAP...us Cavalcanti (55)\*' (which is the active tab), and 'SQLQuery2.sql - LAP...us Cava'. The main area displays a T-SQL query using the DATENAME function to extract various components from the date '2017/08/25'. The query is as follows:

```
SELECT
    DATENAME(DAY, '2017/08/25') AS 'Dia',
    DATENAME(DAYOFYEAR, '2017/08/25') AS 'Dia do ano',
    DATENAME(DW, '2017/08/25') AS 'Nome Dia',
    DATENAME(QUARTER, '2017/08/25') AS 'Trimestre',
    DATENAME(YY, '2017/08/25') AS 'Ano'|
```

Below the code, the 'Resultados' tab is selected, showing the execution results in a grid:

	Dia	Dia do ano	Nome Dia	Trimestre	Ano
1	25	237	Sexta-Feira	3	2017

# DATEADD e DATEDIFF

As funções DATEADD e DATEDIFF são funções que nos permitem realizar cálculos com datas.

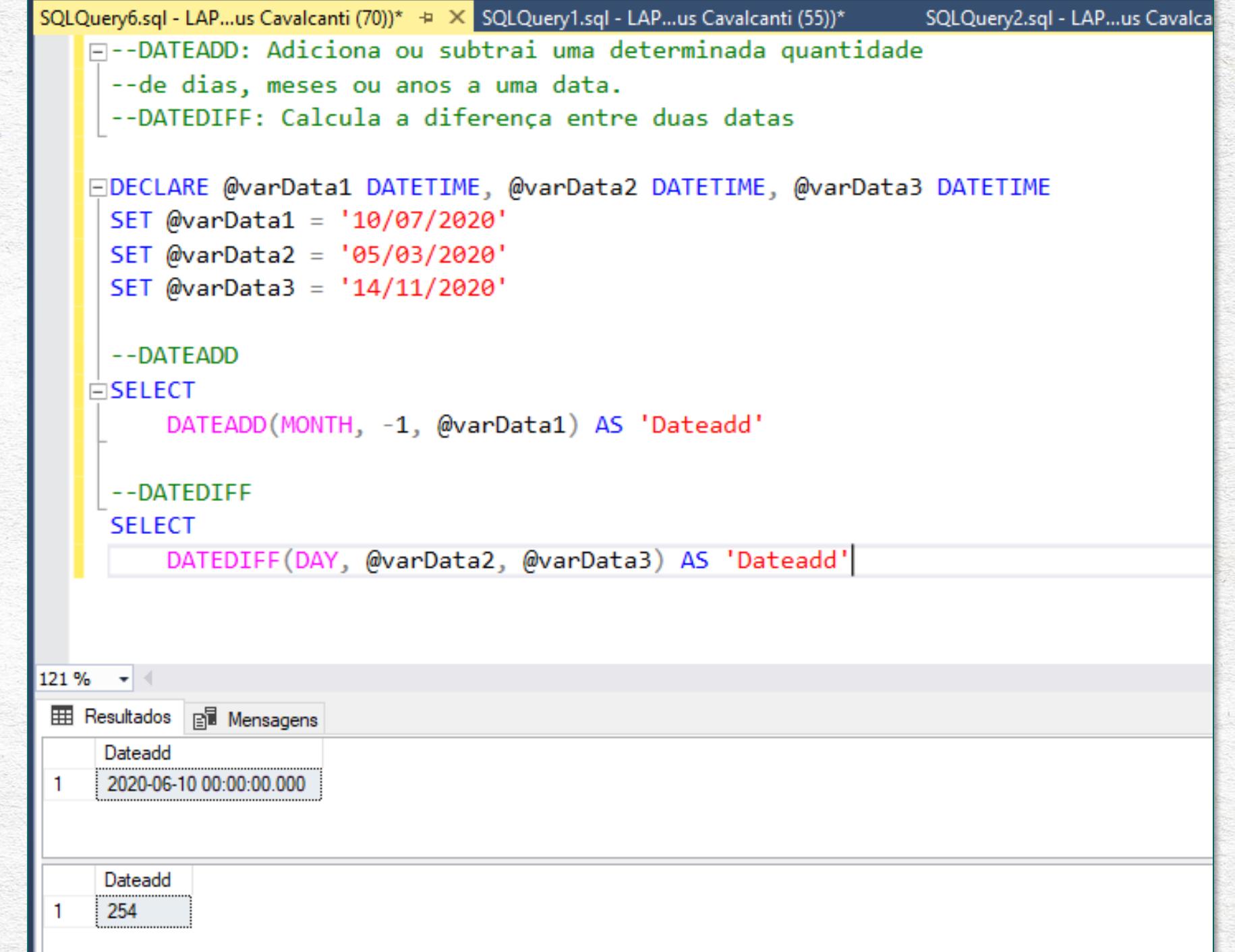
A função **DATEADD** adiciona ou subtrai uma determinada quantidade de dias, meses ou anos a uma data. Essa função pede 3 argumentos:

1. Intervalo de tempo que queremos considerar (dia, mês, ano).
2. Quantidade de intervalos que queremos considerar.
3. Data na qual vamos adicionar ou subtrair os intervalos.

Já a função **DATEDIFF** permite o cálculo da diferença entre duas datas. Essa função pede 3 argumentos:

1. Intervalo de tempo que queremos subtrair (dia, mês, ano).
2. Data 1 a ser considerada no cálculo.
3. Data 2 a ser considerada no cálculo.

Ao lado, podemos ver exemplos práticos dessas funções.



The screenshot shows a SQL Server Management Studio window with three tabs: SQLQuery6.sql, SQLQuery1.sql, and SQLQuery2.sql. The SQLQuery6.sql tab contains the following code:

```

--DATEADD: Adiciona ou subtrai uma determinada quantidade
--de dias, meses ou anos a uma data.
--DATEDIFF: Calcula a diferença entre duas datas

DECLARE @varData1 DATETIME, @varData2 DATETIME, @varData3 DATETIME
SET @varData1 = '10/07/2020'
SET @varData2 = '05/03/2020'
SET @varData3 = '14/11/2020'

--DATEADD
SELECT
    DATEADD(MONTH, -1, @varData1) AS 'Dateadd'

--DATEDIFF
SELECT
    DATEDIFF(DAY, @varData2, @varData3) AS 'Dateadd'

```

The Results tab displays two rows of data:

	Dateadd
1	2020-06-10 00:00:00.000

	Dateadd
1	254

# SQL

## Funções Condicionais

# CASE WHEN.. ELSE

Assim como qualquer linguagem de programação, no SQL também podemos tratar condições.

A estrutura utilizada para tratamento de condições aqui é o **CASE**. A função CASE nos permite tratar condições e retornar um determinado valor, de acordo com o resultado de um ou mais testes lógicos. Essa função tem exatamente o mesmo objetivo das funções SE e IF que você já deve ter visto em algum momento.

Ao lado, temos um exemplo simples de como usar o CASE.

Ele será incluído em nosso código dentro de um SELECT como se fosse uma nova coluna.

No exemplo, utilizamos o CASE para verificar se o Gênero do cliente é igual a 'M'. Se for, queremos retornar o texto 'Masculino'. Caso contrário, retornamos o resultado 'Feminino'.

The screenshot shows a SQL Server Management Studio interface with three tabs at the top: 'SQLQuery6.sql - LAP...us Cavalcanti (70)\*', 'SQLQuery1.sql - LAP...us Cavalcanti (55)\*', and 'SQLQuery2.s'. The main window displays a SELECT statement:

```

SELECT
    CustomerKey,
    FirstName,
    Gender,
    CASE
        WHEN Gender = 'M' THEN 'Masculino'
        ELSE 'Feminino'
    END AS 'Sexo'
FROM
    DimCustomer
  
```

The 'Sexo' column in the result set is highlighted with an orange border. The result set data is as follows:

	CustomerKey	FirstName	Gender	Sexo
1	1	Jon	M	Masculino
2	2	Eugene	M	Masculino
3	3	Ruben	M	Masculino
4	4	Christy	F	Feminino
5	5	Elizabeth	F	Feminino
6	6	Julio	M	Masculino
7	7	Janet	F	Feminino

# CASE WHEN.. ELSE (mais de uma condição)

No exemplo anterior, vimos como utilizar o CASE com apenas um teste lógico. Ou seja, verificamos apenas uma vez o sexo do cliente. Caso o gênero não fosse 'M', já assumimos que seria 'F'.

Porém, alguns clientes na verdade são empresas, e na coluna de Gender o que temos é o valor NULL.

Neste caso, se utilizarmos o CASE criado no exemplo anterior, todas as empresas ficarão com o Sexo 'Feminino'.

O ideal seria que a gente fizesse um novo teste, para conseguir verificar se, caso o Gênero não seja nem 'M' nem 'F', ai sim podemos considerar como 'Empresa'.

Observe na imagem ao lado o resultado final deste CASE com dois testes lógicos.

```

SQLQuery6.sql - LAP...us Cavalcanti (70)* SQLQuery1.sql - LAP...us Cavalcanti (55)* SQLQuery2.
SELECT
    CustomerKey,
    FirstName,
    Gender,
    CASE
        WHEN Gender = 'M' THEN 'Masculino'
        WHEN Gender = 'F' THEN 'Feminino'
        ELSE 'Empresa'
    END AS 'Sexo'
FROM
    DimCustomer
  
```

	CustomerKey	FirstName	Gender	Sexo
1...	18481	Nina	F	Feminino
1...	18482	Ivan	M	Masculino
1...	18483	Clayton	M	Masculino
1...	18484	Jéssus	M	Masculino
1...	18761	NULL	NULL	Empresa
1...	18762	NULL	NULL	Empresa
1...	18763	NULL	NULL	Empresa

# CASE + AND

Podemos combinar o **CASE** com o **AND** para criar testes lógicos ainda mais avançados.

Observe o exemplo ao lado. Queremos criar uma nova coluna em nossa tabela para registrar o desconto de cada produto.

Os produtos que forem da marca Contoso E também forem da cor Red, receberão um desconto de 10%. Caso contrário, não recebem nenhum desconto.

A solução final é mostrada ao lado. Repare que a única diferença em relação aos exemplos anteriores foi que adicionamos um AND para nos permitir passar mais de um teste lógico.

**Como o operador lógico AND, as duas condições (Marca igual à Contoso E Cor igual à Red) precisam ser satisfeitas para que o desconto seja aplicado.**

```

SQLQuery6.sql - LAP...us Cavalcanti (70)* SQLQuery1.sql - LAP...us Cavalcanti (55)* SQLQuery2.sql - LAP...us Cavalcanti (54)*
--CASE/AND: Exemplo
-- Aplique um desconto de 10% aos produtos que foram da marca CONTOSO
-- E TAMBÉM da cor Red
SELECT
    ProductName,
    BrandName,
    ColorName,
    UnitPrice,
    CASE
        WHEN BrandName = 'Contoso' AND ColorName = 'Red' THEN 0.1
        ELSE 0
    END AS 'Desconto'
FROM
    DimProduct

```

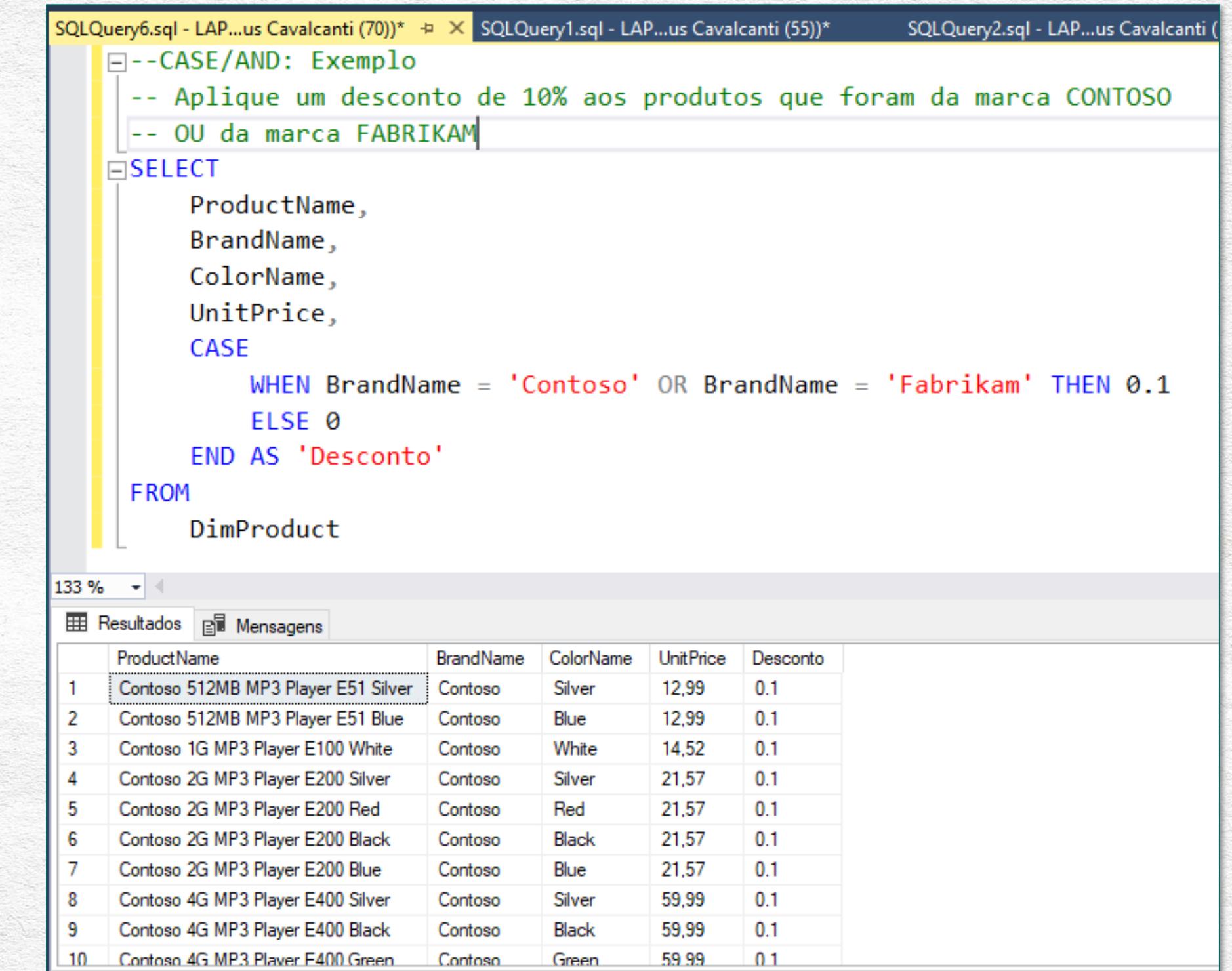
	ProductName	BrandName	ColorName	UnitPrice	Desconto
1	Contoso 512MB MP3 Player E51 Silver	Contoso	Silver	12,99	0,0
2	Contoso 512MB MP3 Player E51 Blue	Contoso	Blue	12,99	0,0
3	Contoso 1G MP3 Player E100 White	Contoso	White	14,52	0,0
4	Contoso 2G MP3 Player E200 Silver	Contoso	Silver	21,57	0,0
5	Contoso 2G MP3 Player E200 Red	Contoso	Red	21,57	0,1
6	Contoso 2G MP3 Player E200 Black	Contoso	Black	21,57	0,0
7	Contoso 2G MP3 Player E200 Blue	Contoso	Blue	21,57	0,0
8	Contoso 4G MP3 Player E400 Silver	Contoso	Silver	59,99	0,0
9	Contoso 4G MP3 Player E400 Black	Contoso	Black	59,99	0,0
10	Contoso 4G MP3 Player E400 Green	Contoso	Green	59,99	0,0

# CASE + OR

Podemos também combinar o **CASE** com o operador lógico **OR**.

No exemplo ao lado, os produtos que forem da marca Contoso OU da marca Fabrikam, receberão um desconto de 10%. Caso contrário, não recebem nenhum desconto.

Como o operador lógico **OR**, qualquer uma das duas condições poderá ser satisfeita (Marca igual à Contoso OU Marca igual à Fabrikam) para que o desconto seja aplicado.



```

SQLQuery6.sql - LAP...us Cavalcanti (70)*  X SQLQuery1.sql - LAP...us Cavalcanti (55)*      SQLQuery2.sql - LAP...us Cavalcanti (55)*
-- CASE/AND: Exemplo
-- Aplique um desconto de 10% aos produtos que foram da marca CONTOSO
-- OU da marca FABRIKAM
SELECT
    ProductName,
    BrandName,
    ColorName,
    UnitPrice,
    CASE
        WHEN BrandName = 'Contoso' OR BrandName = 'Fabrikam' THEN 0.1
        ELSE 0
    END AS 'Desconto'
FROM
    DimProduct
  
```

	ProductName	BrandName	ColorName	UnitPrice	Desconto
1	Contoso 512MB MP3 Player E51 Silver	Contoso	Silver	12,99	0.1
2	Contoso 512MB MP3 Player E51 Blue	Contoso	Blue	12,99	0.1
3	Contoso 1G MP3 Player E100 White	Contoso	White	14,52	0.1
4	Contoso 2G MP3 Player E200 Silver	Contoso	Silver	21,57	0.1
5	Contoso 2G MP3 Player E200 Red	Contoso	Red	21,57	0.1
6	Contoso 2G MP3 Player E200 Black	Contoso	Black	21,57	0.1
7	Contoso 2G MP3 Player E200 Blue	Contoso	Blue	21,57	0.1
8	Contoso 4G MP3 Player E400 Silver	Contoso	Silver	59,99	0.1
9	Contoso 4G MP3 Player E400 Black	Contoso	Black	59,99	0.1
10	Contoso 4G MP3 Player E400 Green	Contoso	Green	59,99	0.1

# CASE Aninhado

O CASE Aninhado nada mais é do que um CASE dentro de outro CASE. Em algumas situações ele pode ser muito útil.

No exemplo ao lado, criamos uma lógica onde, de acordo com a quantidade de funcionários da loja (EmployeeCount) estabelecemos um status para aquela loja:

- Acima de 40 funcionários
- Entre 30 e 39 funcionários
- Entre 20 e 29 funcionários
- Abaixo de 19 funcionários

Porém, especificamente para as lojas que têm mais de 40 funcionários, queremos fazer um novo teste e dividir em 'LOJA ABERTA' ou 'LOJA FECHADA'.

Para isso, utilizamos um novo CASE, como mostrado na imagem.

The screenshot shows a SQL Server Management Studio window with three tabs at the top: 'SQLQuery6.sql - LAP...us Cavalcanti (70)\*', 'SQLQuery1.sql - LAP...us Cavalcanti (55)\*', and 'SQLQuery2.sql - LAP...us Cavalcanti (5\*'. The main pane displays a nested CASE statement:

```

SELECT
    StoreKey,
    StoreName,
    StoreType,
    Status,
    EmployeeCount,
    CASE
        WHEN EmployeeCount >= 40 THEN
            CASE
                WHEN Status = 'Off' THEN 'LOJA FECHADA: Acima de 40 funcionários'
                WHEN Status = 'ON' THEN 'LOJA ABERTA: Acima de 40 funcionários'
            END
        WHEN EmployeeCount >= 30 THEN 'Entre 30 e 39 funcionários'
        WHEN EmployeeCount >= 20 THEN 'Entre 20 e 29 funcionários'
        ELSE 'Abaixo de 19 funcionários'
    END AS 'Categoria'
FROM
    DimStore
  
```

The 'Resultados' tab at the bottom shows the output of the query:

	StoreKey	StoreName	StoreType	Status	EmployeeCount	Categoria
1	1	Contoso Seattle No.1 Store	Store	On	17	Abaixo de 19 funcionários
2	2	Contoso Seattle No.2 Store	Store	On	25	Entre 20 e 29 funcionários
3	3	Contoso Kennewick Store	Store	On	26	Entre 20 e 29 funcionários
4	4	Contoso Bellevue Store	Store	On	19	Abaixo de 19 funcionários
5	5	Contoso Redmond Store	Store	On	33	Entre 30 e 39 funcionários
6	6	Contoso Yakima Store	Store	On	47	LOJA ABERTA: Acima de 40 funcionários
7	7	Contoso Granger Store	Store	On	22	Entre 20 e 29 funcionários
8	8	Contoso Sunnyside Store	Store	On	17	Abaixo de 19 funcionários
9	9	Contoso Toppenish Store	Store	On	25	Entre 20 e 29 funcionários
10	10	Contoso Wanato Store	Store	On	25	Entre 20 e 29 funcionários

# CASE Aditivo

Uma outra aplicação do CASE é em uma situação especial, mostrada ao lado.

Nessa aplicação, conseguimos adicionar uma condição a uma condição anterior.

Por exemplo, no exercício ao lado, queremos aplicar um desconto de 10% aos produtos da categoria 'TV and Video'. E além disso, caso também seja da subcategoria 'Televisão', queremos aplicar mais 5%, fechando em 15% de desconto.

Criamos um primeiro CASE para verificar a categoria. Logo em seguida, somamos ao CASE que verifica se a subcategoria é televisão.

O resultado final é mostrado na imagem ao lado.

The screenshot shows a SQL query in the SQL Query Editor of SSMS. The code uses additive CASE statements to calculate discounts. It first checks if the product category is 'TV and Video' and applies a 10% discount. Then, it checks if the product subcategory is 'Televisions' and adds another 5% discount, resulting in a total discount of 15%. The query joins three dimensions: DimProduct, DimProductSubcategory, and DimProductCategory.

```

SQLQuery6.sql - LAP...us Cavalcanti (70)*  X SQLQuery1.sql - LAP...us Cavalcanti (55)*      SQLQuery2.sql - LAP...us Cavalcanti (54)*
-- CASE STATEMENT Aditivo
-- Os produtos da categoria 'TV and Video' terão um desconto de 10%
-- Se além de ser da categoria 'TV and Video', o produto for da subcategoria 'Televisions', receberá mais 5%. Total, 15%

SELECT
    ProductKey,
    ProductName,
    ProductCategoryName,
    ProductSubcategoryName,
    UnitPrice,
    CASE WHEN ProductCategoryName = 'TV and Video'
        THEN 0.10 ELSE 0.00 END
    + CASE WHEN ProductSubCategoryName = 'Televisions'
        THEN 0.05 ELSE 0.00 END
FROM DimProduct
INNER JOIN DimProductSubcategory
    ON DimProduct.ProductSubcategoryKey = DimProductSubcategory.ProductSubcategoryKey
INNER JOIN DimProductCategory
    ON DimProductSubcategory.ProductCategoryKey = DimProductCategory.ProductCategoryKey

```

**Resultados**

	ProductKey	ProductName	ProductCategoryName	ProductSubcategoryName	UnitPrice	(Nenhum nome de coluna)
1	1	Contoso 512MB MP3 Player E51 Silver	Audio	MP4&MP3	12,99	0,00
2	2	Contoso 512MB MP3 Player E51 Blue	Audio	MP4&MP3	12,99	0,00
3	3	Contoso 1G MP3 Player E100 White	Audio	MP4&MP3	14,52	0,00
4	4	Contoso 2G MP3 Player E200 Silver	Audio	MP4&MP3	21,57	0,00
5	5	Contoso 2G MP3 Player E200 Red	Audio	MP4&MP3	21,57	0,00
6	6	Contoso 2G MP3 Player E200 Black	Audio	MP4&MP3	21,57	0,00
7	7	Contoso 2G MP3 Player E200 Blue	Audio	MP4&MP3	21,57	0,00
8	8	Contoso 4G MP3 Player E400 Silver	Audio	MP4&MP3	59,99	0,00
9	9	Contoso 4G MP3 Player E400 Black	Audio	MP4&MP3	59,99	0,00
10	10	Contoso 4G MP3 Player E400 Green	Audio	MP4&MP3	59,99	0,00

Consulta executada com êxito.

# IIF: alternativa ao CASE

Uma alternativa ao CASE seria a função IIF. Ela é muito semelhante com a fórmula SE do Excel, pois pede exatamente 3 argumentos:

1. Teste Lógico
2. Valor se verdadeiro
3. Valor se falso

No exercício ao lado, queremos classificar um projeto de acordo com o seu nível de risco. Utilizamos para isso uma variável.

O resultado final é mostrado na imagem.

The screenshot shows a SQL query window with three tabs: SQLQuery6.sql, SQLQuery1.sql, and SQLQuery2.sql. The SQLQuery6.sql tab is active and contains the following code:

```
-- Função IIF
-- Exemplo: Qual é a categoria de risco do projeto abaixo, de acordo com a sua nota:
-- Risco Alto: Classificacao >= 5
-- Risco Baixo: Classificacao < 5

DECLARE @varClassificacao INT
SET @varClassificacao = 2

SELECT
    IIF(
        @varClassificacao >= 5,
        'Risco Alto',
        'Risco Baixo') AS 'Status'
```

The results pane shows a single row with the value 'Risco Baixo' under the 'Status' column.

# IIF Composto

É possível utilizar um IIF dentro de outro IIF, criando o que chamamos de **IIF composto**.

Com essa aplicação, tratamos vários resultados possíveis, de acordo com mais de um teste lógico.

No exemplo ao lado, queremos fazer uma consulta à tabela DimProduct e retornar o responsável pelos estoques dos produtos, de acordo com o StockTypeName.

Como temos 3 tipos de estoque e consequentemente 3 responsáveis diferentes, teremos que utilizar mais de 1 IIF para conseguir cobrir todas as situações possíveis.

O resultado final da consulta é mostrado ao lado.

```

SQLQuery9.sql - LAP...us Cavalcanti (54)* ✎ X
--IIF Composto
--Faça um SELECT na tabela DimProduct e
--crie uma coluna que identifique os responsáveis
--por cada produto, de acordo com o seu estoque (StockTypeName)
--João é responsável pelos produtos High
--Maria é responsável pelos produtos Mid
--Luis é responsável pelos produtos Low

SELECT
    ProductKey,
    ProductName,
    StockTypeName,
    IIF(
        StockTypeName = 'High',
        'João',
        IIF(
            StockTypeName = 'Mid',
            'Maria',
            'Luis')
    ) AS 'Responsável'
FROM
    DimProduct
  
```

# ISNULL: Tratando valores nulos

A função **ISNULL** nos permite retornar um resultado alternativo para o caso de um valor ser nulo (NULL).

Observe na imagem ao lado a coluna **CityName**. Diversas dessas cidades estão com o valor NULL.

Podemos utilizar a função **ISNULL** para verificar essa coluna (no 1º argumento) e caso o valor seja nulo, ele retornará um resultado alternativo.

No exemplo ao lado, o resultado alternativo para as cidades com o nome NULL será o texto ‘Local desconhecido’.

```
--ISNULL: Tratando valores nulos
SELECT
    GeographyKey,
    ContinentName,
    CityName,
    ISNULL(CityName, 'Local desconhecido')
FROM
    DimGeography
```

The screenshot shows a SQL query in the SSMS interface. The query uses the ISNULL function to replace NULL values in the CityName column with the string 'Local desconhecido'. The results grid displays 164 rows from the DimGeography table, showing various geographical keys, continent names, and city names, with the last column showing the result of the ISNULL function.

	GeographyKey	ContinentName	CityName	(Nenhum nome de coluna)
148	413	Europe	NULL	Local desconhecido
149	414	Europe	NULL	Local desconhecido
150	415	North America	NULL	Local desconhecido
151	416	Europe	NULL	Local desconhecido
152	417	North America	NULL	Local desconhecido
153	418	Europe	NULL	Local desconhecido
154	419	Europe	NULL	Local desconhecido
155	420	Asia	NULL	Local desconhecido
156	421	North America	NULL	Local desconhecido
157	422	North America	NULL	Local desconhecido
158	423	Europe	Basings...	Basingstoke Hants
159	424	North America	Bellevue	Bellevue
160	425	Asia	Bendigo	Bendigo
161	426	Europe	Berks	Berks
162	430	North America	Beverly...	Beverly Hills
163	431	North America	Billings	Billings
164	432	North America	Rilovi	Rilovi

# S O U L

## Views



# Introdução

Até aqui vimos como criar diferentes consultas aos bancos de dados. Utilizamos comandos como o SELECT, GROUP BY, JOINs, etc para criar tabelas como a mostrada ao lado.

Mas pra onde foram todas essas tabelas que a gente criou? Elas estão em algum lugar?

A resposta é: elas não estão em nenhum lugar!

Tudo o que fizemos até agora foi apenas visualizar alguns dados das nossas tabelas do Banco de Dados, nada além disso. Quando executamos um SELECT e logo em seguida executamos outro SELECT, o resultado do primeiro é perdido.

Nenhuma das consultas que fizemos ficou salvo em algum lugar. Inclusive, diversas vezes precisamos criar as mesmas consultas, pois elas se perdem a cada novo SELECT, ou quando fechamos uma consulta e abrimos uma nova.

Existe uma solução pra gente conseguir salvar essas tabelas em algum lugar, e essa solução é a **View**.

The screenshot shows a SQL Server Management Studio (SSMS) interface. At the top, there's a title bar for 'SQLQuery1.sql - LAP...us Cavalcanti (54)\*'. Below it is a code editor window containing the following SQL query:

```

SELECT
    FirstName AS 'Nome',
    EmailAddress AS 'E-mail',
    BirthDate AS 'Data Nascimento'
FROM
    dimCustomer
  
```

Below the code editor is a results grid titled 'Resultados' (Results). The grid displays 12 rows of data from the 'dimCustomer' table, with columns labeled 'Nome', 'E-mail', and 'Data Nascimento'. The data includes names like Jon, Eugene, Ruben, Christy, Elizabeth, Julio, Janet, Marco, Rob, Shannon, Jacquelyn, and Curtis, along with their corresponding email addresses and birth dates. A yellow border highlights the results grid. At the bottom of the results grid, a message says 'Consulta executada com êxito.' (Query executed successfully). The status bar at the bottom of the screen shows 'LAPTOP-HBTCI8PQ (15.0 RTM) | LAPTOP-HBTCI8PQ\Marcus... | ContosoRetailDW | 00:00:00 | 18.869 linhas'.

# O que é uma View?

- Uma View (ou traduzindo, uma exibição), é uma tabela virtual criada a partir de uma consulta a uma ou mais tabelas (ou até mesmo de outras views) no banco de dados.
- Ela contém linhas e colunas assim como uma tabela real. Nela podemos utilizar comandos como o JOIN, WHERE e outras funções.
- Sempre mostra resultados atualizados dos dados, ou seja, uma vez criada uma View, caso haja alterações no Banco de Dados, as Views são atualizadas automaticamente.
- Caso o servidor seja desligado (ou o SSMS fechado), a view continua armazenada no sistema.

Através de uma View, conseguimos armazenar o resultado de um SELECT (como por exemplo, a tabela à direita) e acessar sempre que precisar, como se fosse uma tabela, com a vantagem de não precisar criar esse SELECT do zero.

The screenshot shows the SSMS interface with a query window titled "SQLQuery1.sql - LAP...us Cavalcanti (54)\*". The query is:

```

SELECT
    FirstName AS 'Nome',
    EmailAddress AS 'E-mail',
    BirthDate AS 'Data Nascimento'
FROM |
    dimCustomer
  
```

The results grid below shows 12 rows of data from the "dimCustomer" table:

	Nome	E-mail	Data Nascimento
1	Jon	jon24@adventure-works.com	1966-04-08
2	Eugene	eugene10@adventure-works.com	1965-05-14
3	Ruben	ruben35@adventure-works.com	1965-08-12
4	Christy	christy12@adventure-works.com	1968-02-15
5	Elizabeth	elizabeth5@adventure-works.com	1968-08-08
6	Julio	julio1@adventure-works.com	1965-08-05
7	Janet	janet9@adventure-works.com	1965-12-06
8	Marco	marco14@adventure-works.com	1964-05-09
9	Rob	rob4@adventure-works.com	1964-07-07
10	Shannon	shannon38@adventure-works.com	1964-04-01
11	Jacquelyn	jacquelyn20@adventure-works.com	1964-02-06
12	Curtis	curtis9@adventure-works.com	1963-11-04

At the bottom, a message says "Consulta executada com êxito." (Query executed successfully.)

# Por que criar uma View?

São muitas as vantagens de uma View. Abaixo temos algumas das principais:

## Reutilização



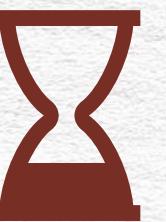
Sempre que necessário, podemos consultar aquela View, pois ela fica armazenada no sistema.

## Segurança



Ao criar uma View, estamos ocultando linhas ou colunas da tabela original do banco de dados. Desta forma, apenas algumas informações relevantes serão visualizadas na View.

## Ganho de tempo



Quando criamos Views, estamos poupando o tempo de recriar vários SELECTs, o que aumenta a produtividade.

# CREATE VIEW: Criando a primeira View

Para criar uma View, utilizamos o comando **CREATE VIEW**. Na imagem abaixo temos a estrutura padrão:

```
CREATE VIEW nome_da_view AS
```

```
SELECT
```

```
Coluna1,  
Coluna2,  
Coluna3
```

```
FROM
```

```
Tabela
```

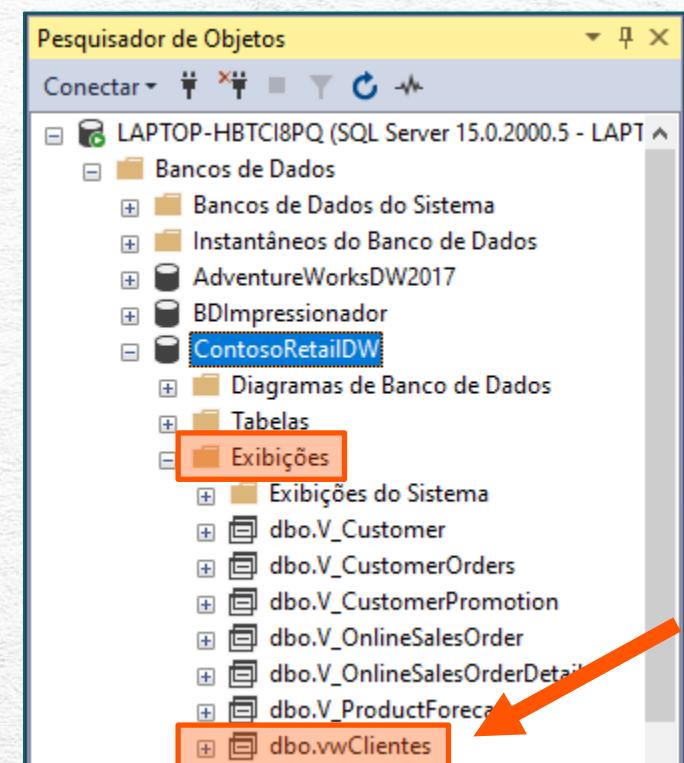
The screenshot shows the SQL Server Management Studio interface. In the center pane, there is a code editor window with the following SQL script:

```
CREATE VIEW vwClientes AS
SELECT
    FirstName AS 'Nome',
    EmailAddress AS 'E-mail',
    BirthDate AS 'Data Nascimento'
FROM
    DimCustomer
```

The word "vwClientes" is highlighted with a red wavy underline, indicating it is a new object being created.

Uma vez criada, a View ficará armazenada no banco de dados, na pasta de **Exibições**.

E para selecionar essa View, utilizamos o comando **SELECT**.



The screenshot shows the results pane of SQL Server Management Studio. A query window contains the following SQL command:

```
SELECT * FROM vwClientes
```

Below the query window, the results grid displays the data returned by the view. The columns are labeled "Nome", "E-mail", and "Data Nascimento". The data rows are:

Nome	E-mail	Data Nascimento
Jon	jon24@adventure-works.com	1966-04-08
Eugene	eugene10@adventure-works.com	1965-05-14
Ruben	ruben35@adventure-works.com	1965-08-12
Christy	christy12@adventure-works.com	1968-02-15
Elizabeth	elizabeth5@adventure-works.com	1968-08-08
Julio	julio1@adventure-works.com	1965-08-05
Janet	janet9@adventure-works.com	1965-12-06
Marco	marco14@adventure-works.com	1964-05-09
Rob	rob4@adventure-works.com	1964-07-07
Shannon	shannon38@adventure-works.com	1964-04-01
Jacque...	jacquelyn20@adventure-works.c...	1964-02-06
Curtis	curtis9@adventure-works.com	1963-11-04
Lauren	lauren41@adventure-works.com	1968-01-18
Ian	ian47@adventure-works.com	1968-08-06

# ALTER VIEW: Alterando uma View criada

Para alterar uma View criada, usamos o comando **ALTER VIEW**.

Na imagem ao lado, temos um exemplo. A **vwClientes**, criada anteriormente, foi alterada para considerar apenas os clientes do sexo feminino.

The screenshot shows the SQL Server Management Studio interface. In the top tab bar, the active window is titled "SQLQuery6.sql - LAP...us Cavalcanti (70)\*". Below it, another tab is visible: "SQLQuery1.sql - LAP...us Cavalcanti (55)\*". The main pane displays the following T-SQL code:

```
ALTER VIEW vwClientes AS
SELECT
    FirstName AS 'Nome',
    EmailAddress AS 'E-mail',
    BirthDate AS 'Data Nascimento'
FROM
    dimCustomer
WHERE Gender = 'F'

SELECT * FROM vwClientes
```

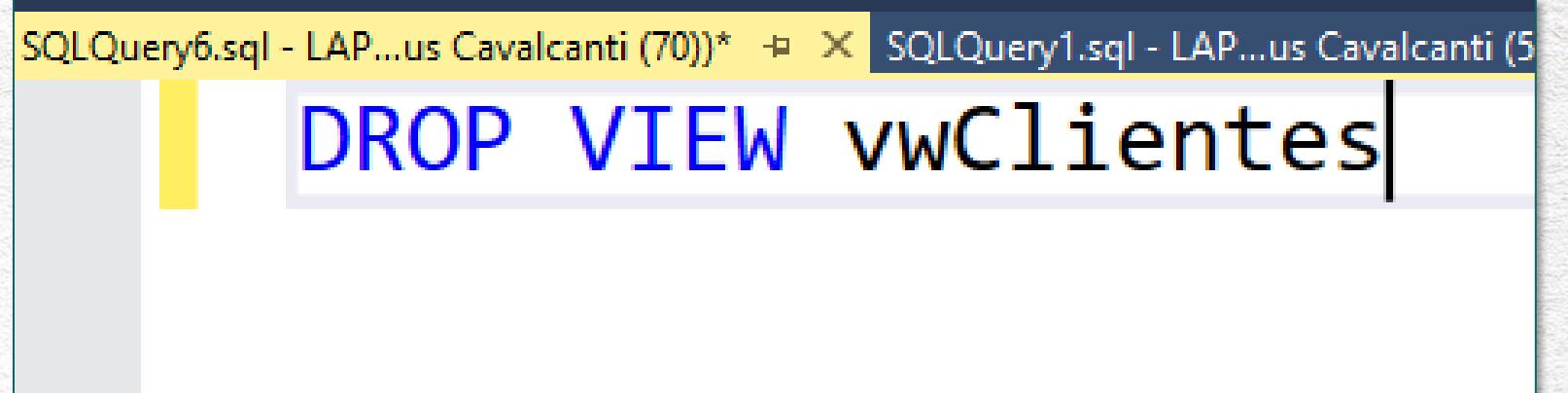
Below the code, the results pane is open, showing the title "Resultados". The results table contains the following data:

	Nome	E-mail	Data Nascimento
1	Christy	christy12@adventure-works.com	1968-02-15
2	Elizabeth	elizabeth5@adventure-works.com	1968-08-08
3	Janet	janet9@adventure-works.com	1965-12-06
4	Rob	rob4@adventure-works.com	1964-07-07
5	Jacquelyn	jacquelyn20@adventure-works.com	1964-02-06
6	Lauren	lauren41@adventure-works.com	1968-01-18
7	Sydney	sydney23@adventure-works.com	1968-05-09

# DROP VIEW: Excluindo uma View

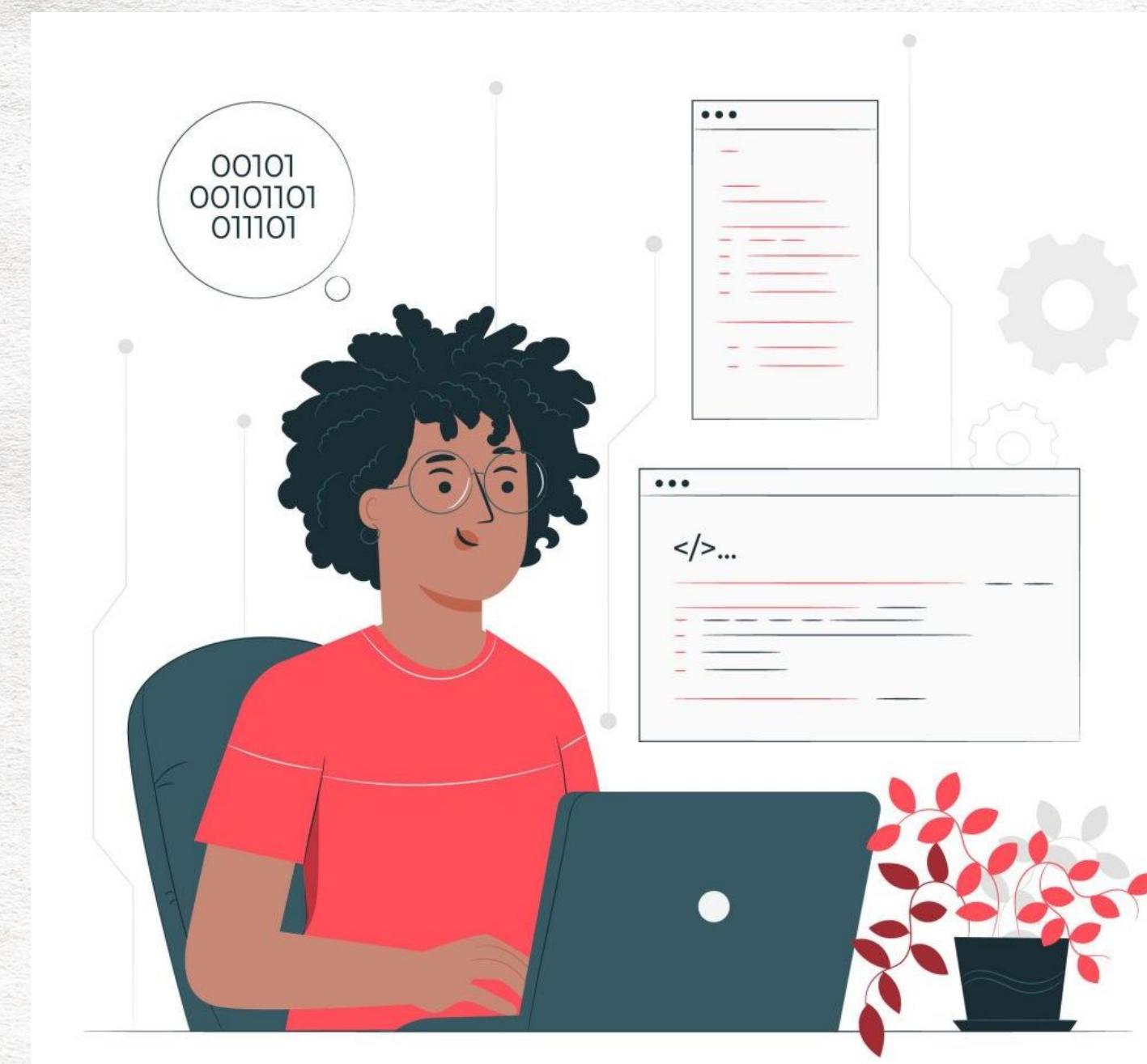
Para excluir uma View criada, usamos o comando **DROP VIEW**.

O comando é bem simples e é ilustrado na imagem ao lado.



The screenshot shows a SQL Server Management Studio window with two tabs: 'SQLQuery6.sql - LAP...us Cavalcanti (70)\*' and 'SQLQuery1.sql - LAP...us Cavalcanti (5)'. The query 'DROP VIEW vwClientes' is typed into the second tab's editor area. The text 'DROP VIEW vwClientes' is highlighted in blue, indicating it is a SQL keyword.

# EXERCÍCIOS



1

## Questão 1

- A partir da tabela DimProduct, crie uma View contendo as informações de ProductName, ColorName, UnitPrice e UnitCost, da tabela DimProduct. Chame essa View de vwProdutos.
- A partir da tabela DimEmployee, crie uma View mostrando FirstName, BirthDate, DepartmentName. Chame essa View de vwFuncionarios.
- A partir da tabela DimStore, crie uma View mostrando StoreKey, StoreName e OpenDate. Chame essa View de vwLojas.

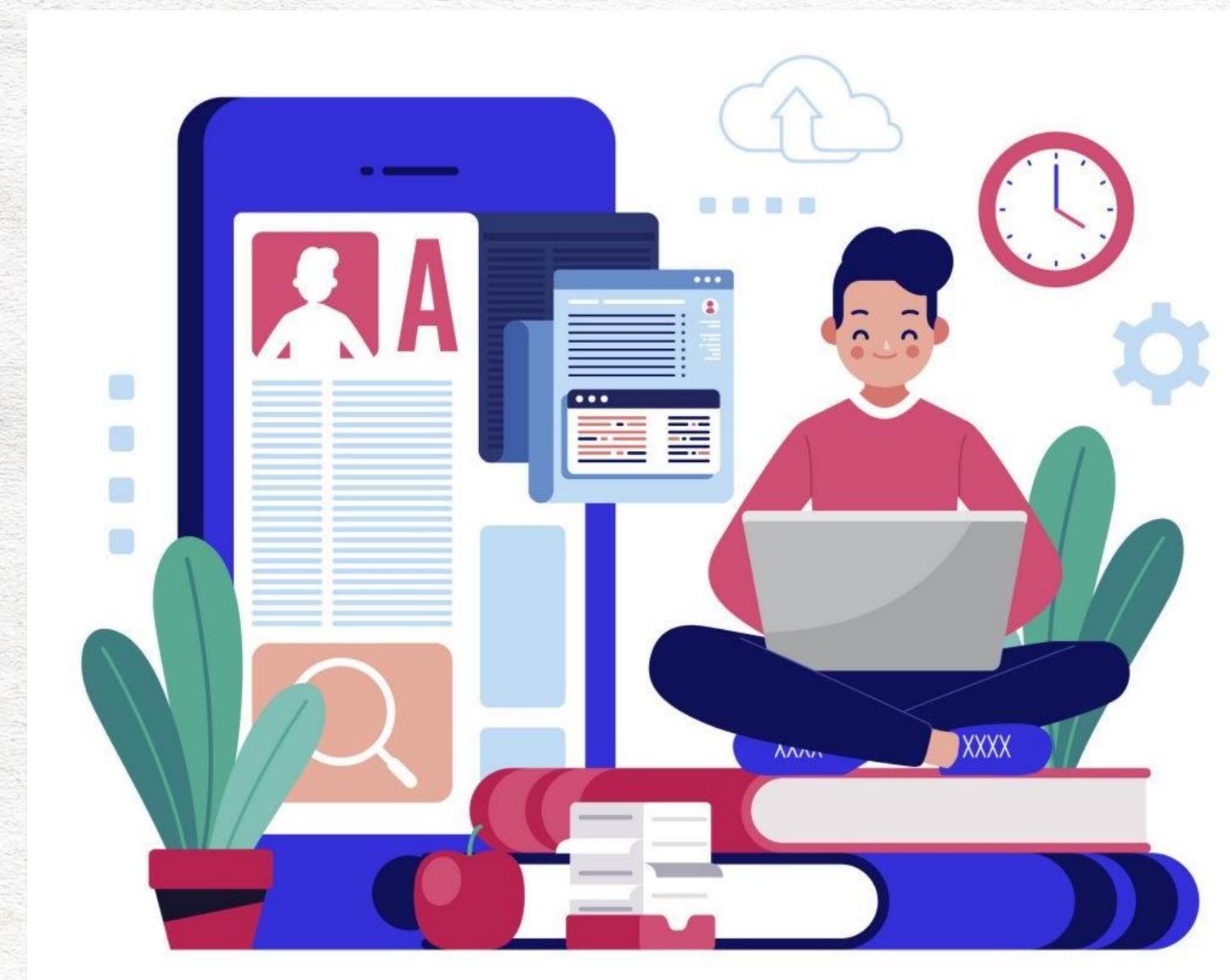
2

## Questão 2

Crie uma View contendo as informações de Nome Completo (FirstName + LastName), Gênero (por extenso), E-mail e Renda Anual (formatada com R\$).

Utilize a tabela DimCustomer. Chame essa View de vwClientes.

# EXERCÍCIOS



3

## Questão 3

- A partir da tabela DimStore, crie uma View que considera apenas as lojas ativas. Faça um SELECT de todas as colunas. Chame essa View de vwLojasAtivas.
- A partir da tabela DimEmployee, crie uma View de uma tabela que considera apenas os funcionários da área de Marketing. Faça um SELECT das colunas: FirstName, EmailAddress e DepartmentName. Chame essa de vwFuncionariosMkt.
- Crie uma View de uma tabela que considera apenas os produtos das marcas Contoso e Litware. Além disso, a sua View deve considerar apenas os produtos de cor Silver. Faça um SELECT de todas as colunas da tabela DimProduct. Chame essa View de vwContosoLitwareSilver.

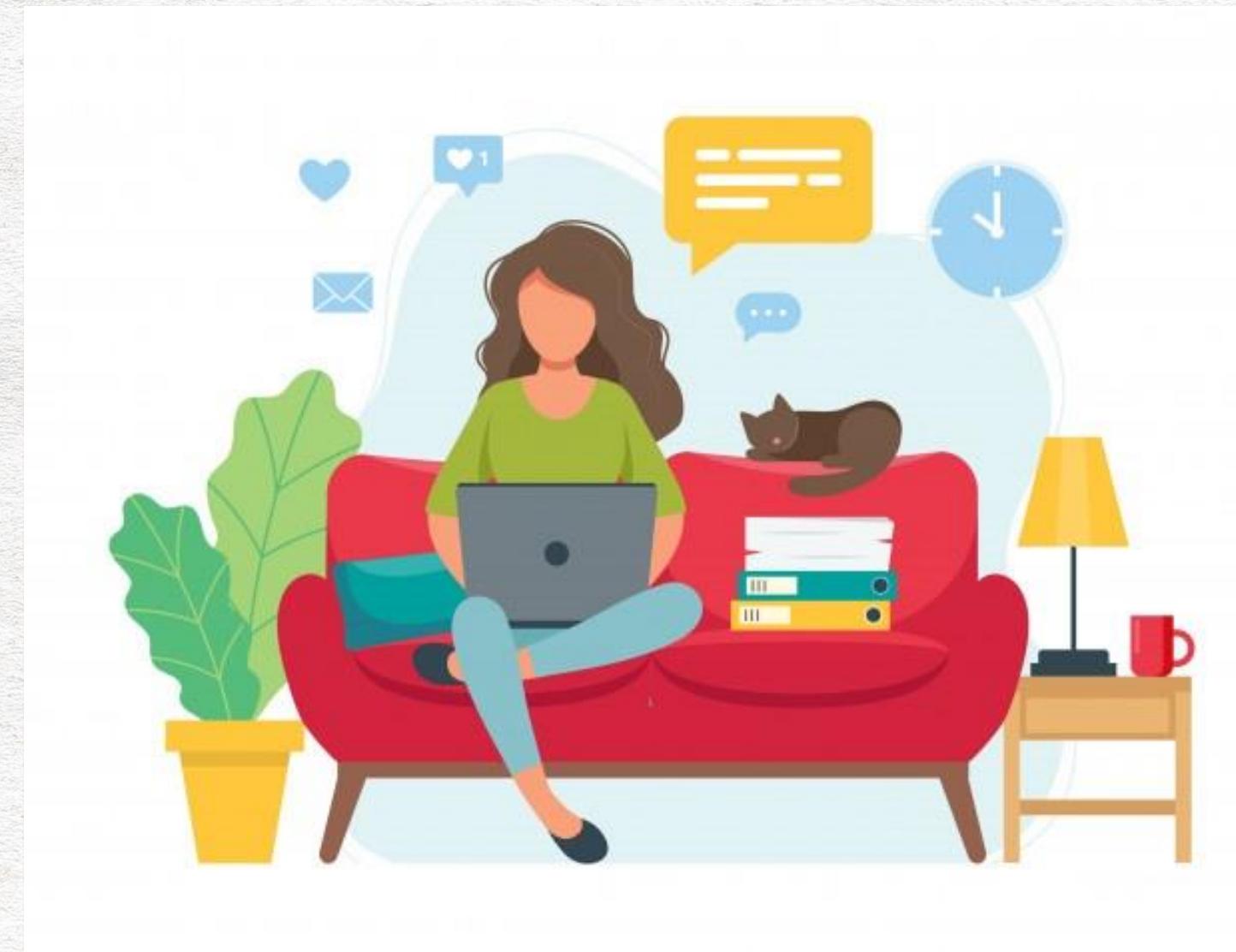
4

## Questão 4

Crie uma View que seja o resultado de um agrupamento da tabela FactSales. Este agrupamento deve considerar o SalesQuantity (Quantidade Total Vendida) por Nome do Produto. Chame esta View de vwTotalVendidoProdutos.

OBS: Para isso, você terá que utilizar um JOIN para relacionar as tabelas FactSales e DimProduct.

# EXERCÍCIOS



5

## Questão 5

Faça as seguintes alterações nas tabelas da questão 1.

- Na View criada na letra a da questão 1, adicione a coluna de BrandName.
- Na View criada na letra b da questão 1, faça um filtro e considere apenas os funcionários do sexo feminino.
- Na View criada na letra c da questão 1, faça uma alteração e filtre apenas as lojas ativas.

6

## Questão 6

- Crie uma View que seja o resultado de um agrupamento da tabela DimProduct. O resultado esperado da consulta deverá ser o total de produtos por marca. Chame essa View de vw\_6a.
- Altere a View criada no exercício anterior, adicionando o peso total por marca. Atenção: sua View final deverá ter então 3 colunas: Nome da Marca, Total de Produtos e Peso Total.
- Exclua a View vw\_6a.

# GABARITOS

1

## Questão 1

- a. A partir da tabela DimProduct, crie uma View contendo as informações de ProductName, ColorName, UnitPrice e UnitCost, da tabela DimProduct. Chame essa View de vwProdutos.
- b. A partir da tabela DimEmployee, crie uma View mostrando FirstName, BirthDate, DepartmentName. Chame essa View de vwFuncionarios.
- c. A partir da tabela DimStore, crie uma View mostrando StoreKey, StoreName e OpenDate. Chame essa View de vwLojas.

```
-- Questão 1
-- a)
CREATE VIEW vw_1a AS
SELECT
    ProductName AS 'Nome do Produto',
    ColorName AS 'Cor',
    UnitPrice AS 'Preço Unitário',
    UnitCost AS 'Custo Unitário'
FROM
    DimProduct

-- b)
CREATE VIEW vw_1b AS
SELECT
    FirstName AS 'Nome',
    BirthDate AS 'Data de Nascimento',
    DepartmentName AS 'Departamento'
FROM
    DimEmployee

-- c)
CREATE VIEW vw_1c AS
SELECT
    StoreKey AS 'ID Loja',
    StoreName AS 'Nome da Loja',
    OpenDate AS 'Data de Abertura'
FROM
    DimStore
```

# GABARITOS

2

## Questão 2

Crie uma View contendo as informações de Nome Completo (FirstName + LastName), Gênero (por extenso), E-mail e Renda Anual (formatada com R\$).

Utilize a tabela DimCustomer. Chame essa View de vwClientes.

```
SQLQuery9.sql - LAP...us Cavalcanti (54)* ↵ X
CREATE VIEW vw_2 AS
SELECT
    CONCAT(FirstName, ' ', LastName) AS 'Nome Completo',
    REPLACE(REPLACE(Gender, 'M', 'Masculino'), 'F', 'Feminino') AS 'Gênero',
    EmailAddress AS 'E-mail',
    FORMAT(YearlyIncome, 'C') AS 'Renda Anual'
FROM
    DimCustomer
```

# GABARITOS

3

## Questão 3

- A partir da tabela DimStore, crie uma View que considera apenas as lojas ativas. Faça um SELECT de todas as colunas. Chame essa View de vwLojasAtivas.
- A partir da tabela DimEmployee, crie uma View de uma tabela que considera apenas os funcionários da área de Marketing. Faça um SELECT das colunas: FirstName, EmailAddress e DepartmentName. Chame essa de vwFuncionariosMkt.
- Crie uma View de uma tabela que considera apenas os produtos das marcas Contoso e Litware. Além disso, a sua View deve considerar apenas os produtos de cor Silver. Faça um SELECT de todas as colunas da tabela DimProduct. Chame essa View de vwContosoLitwareSilver.

```

SQLQuery9.sql - LAP...us Cavalcanti (54)* ↗ X
--Questão 3
--a)
CREATE VIEW vw_3a AS
SELECT
*
FROM
DimStore
WHERE Status = 'On'

--b)
CREATE VIEW vw_3b AS
SELECT
FirstName,
EmailAddress,
DepartmentName
FROM
DimEmployee
WHERE DepartmentName = 'Marketing'

--c)
CREATE VIEW vw_3c AS
SELECT
*
FROM
DimProduct
WHERE BrandName IN ('Contoso', 'Litware') AND ColorName = 'Silver'

```

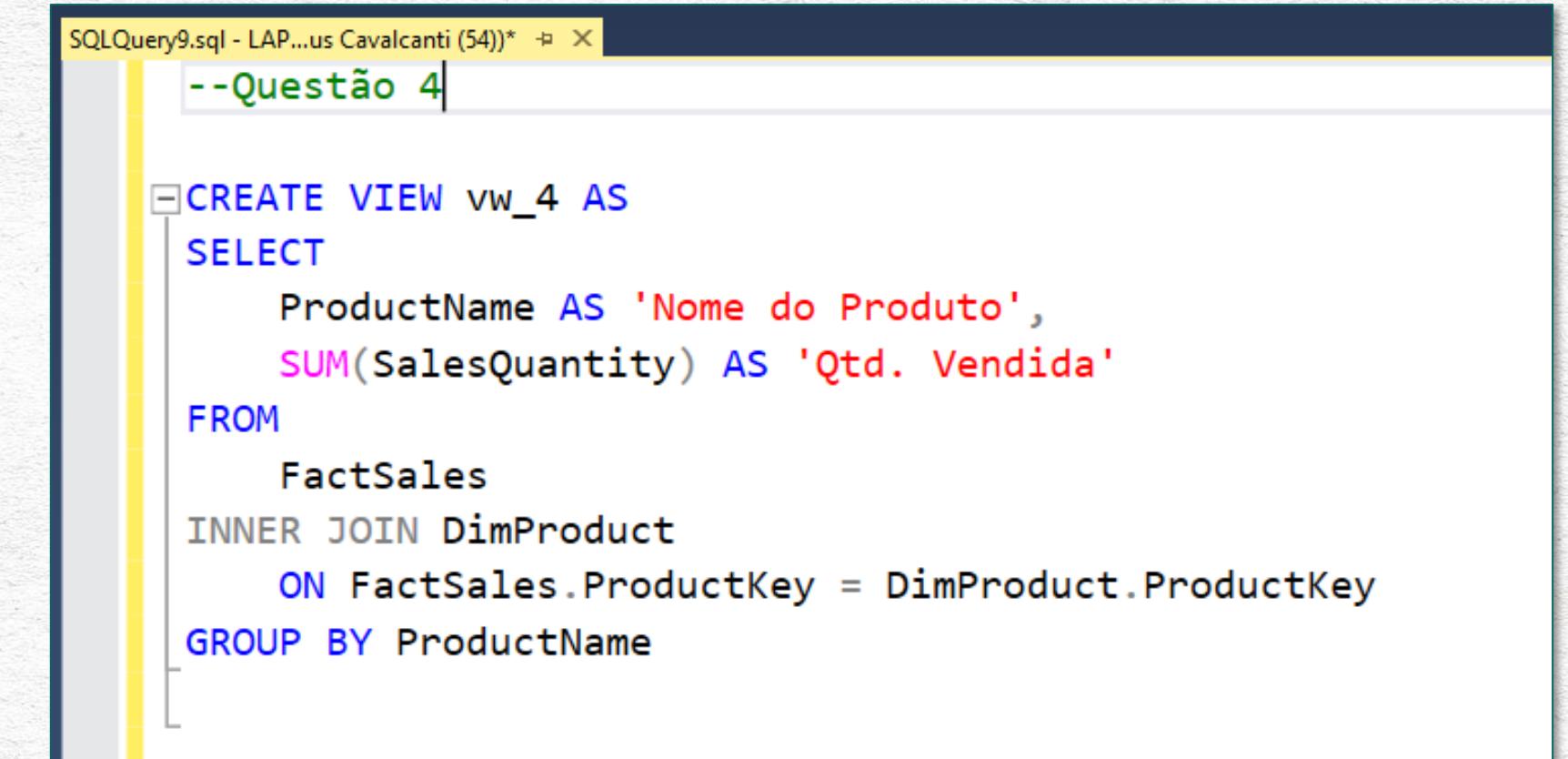
# GABARITOS

4

## Questão 4

Crie uma View que seja o resultado de um agrupamento da tabela FactSales. Este agrupamento deve considerar o SalesQuantity (Quantidade Total Vendida) por Nome do Produto. Chame esta View de vwTotalVendidoProdutos.

OBS: Para isso, você terá que utilizar um JOIN para relacionar as tabelas FactSales e DimProduct.



The screenshot shows a SQL query window titled "SQLQuery9.sql - LAP...us Cavalcanti (54)\*". The query is labeled "--Questão 4" and contains the following SQL code:

```
CREATE VIEW vw_4 AS
SELECT
    ProductName AS 'Nome do Produto',
    SUM(SalesQuantity) AS 'Qtd. Vendida'
FROM
    FactSales
INNER JOIN DimProduct
    ON FactSales.ProductKey = DimProduct.ProductKey
GROUP BY ProductName
```

# GABARITOS

5

## Questão 5

Faça as seguintes alterações nas tabelas da questão 1.

- a. Na View criada na letra a da questão 1, adicione a coluna de BrandName.
- b. Na View criada na letra b da questão 1, faça um filtro e considere apenas os funcionários do sexo feminino.
- c. Na View criada na letra c da questão 1, faça uma alteração e filtre apenas as lojas ativas.

```
--Questão 5
--a)
ALTER VIEW vw_1a AS
SELECT
    ProductName AS 'Nome do Produto',
    BrandName AS 'Marca',
    ColorName AS 'Cor',
    UnitPrice AS 'Preço Unitário',
    UnitCost AS 'Custo Unitário'
FROM
    DimProduct

--b)
ALTER VIEW vw_1b AS
SELECT
    FirstName AS 'Nome',
    BirthDate AS 'Data de Nascimento',
    DepartmentName AS 'Departamento'
FROM
    DimEmployee
WHERE Gender = 'F'

--c)
ALTER VIEW vw_1c AS
SELECT
    StoreKey AS 'ID Loja',
    StoreName AS 'Nome da Loja',
    OpenDate AS 'Data de Abertura'
FROM
    DimStore
WHERE Status = 'On'
```

# GABARITOS

6

## Questão 6

- a. Crie uma View que seja o resultado de um agrupamento da tabela DimProduct. O resultado esperado da consulta deverá ser o total de produtos por marca. Chame essa View de vw\_6a.
- b. Altere a View criada no exercício anterior, adicionando o peso total por marca. Atenção: sua View final deverá ter então 3 colunas: Nome da Marca, Total de Produtos e Peso Total.
- c. Exclua a View vw\_6a.

```
--Questão 6
--a)
CREATE VIEW vw_6a AS
SELECT
    BrandName AS 'Marca',
    COUNT(*) AS 'Total'
FROM
    DimProduct
GROUP BY BrandName

--b)
ALTER VIEW vw_6a AS
SELECT
    BrandName AS 'Marca',
    COUNT(*) AS 'Total',
    SUM(Weight) AS 'Peso'
FROM
    DimProduct
GROUP BY BrandName

--c)
DROP VIEW vw_6a
```

# SQL

## CRUD



# O que é CRUD?

Operações **CRUD** são operações que conseguimos fazer em um Banco de Dados. Essa sigla significa o seguinte:

## CREATE

- Permite criar Bancos de Dados, Tabelas ou Exibições (Views)

## READ

- Permite ler os dados do banco de dados. Basicamente foi o que mais fizemos no Curso, através do SELECT.

## UPDATE

- Permite atualizar os dados do banco de dados, tabelas ou views.

## DELETE

- Permite deletar dados de um banco de dados, tabelas ou views.

No módulo anterior, vimos como criar Views (exibições). Essas exibições não são entendidas exatamente como tabelas do banco de dados.

Agora o que vamos fazer é aprender como criar tabelas propriamente ditas,

# Lembrando o que é um Banco de Dados...

Um banco de dados é composto por um conjunto de tabelas (vide exemplo Contoso).

As tabelas armazenam dados em linhas e colunas.

ProductKey	ProductName	BrandName	UnitPrice
1	Contoso 512MB MP3 Player E51 Silver	Contoso	12,99

Uma linha abrange várias colunas que juntas descrevem alguma característica de um objeto. Desta forma, uma coluna é como se fosse uma categoria.

Portanto, cada objeto (registro/record) possui características que pertencem a essas categorias.

Agora o que vamos fazer é aprender como criar tabelas propriamente ditas,

# Por que um Banco de Dados é importante

Como dito anteriormente, um Banco de Dados será usado para guardar todas as tabelas criadas. É importante entender que as tabelas de um banco de dados precisam ter algum tipo de relação.

Tomando como exemplo o Banco de Dados Contoso, todas as tabelas ali dentro têm algum tipo de relação. Não faria sentido, por exemplo, a gente criar uma tabela que contivesse informações de roupas da Riachuelo, sendo que é uma informação que não tem relação nenhuma com a Contoso.

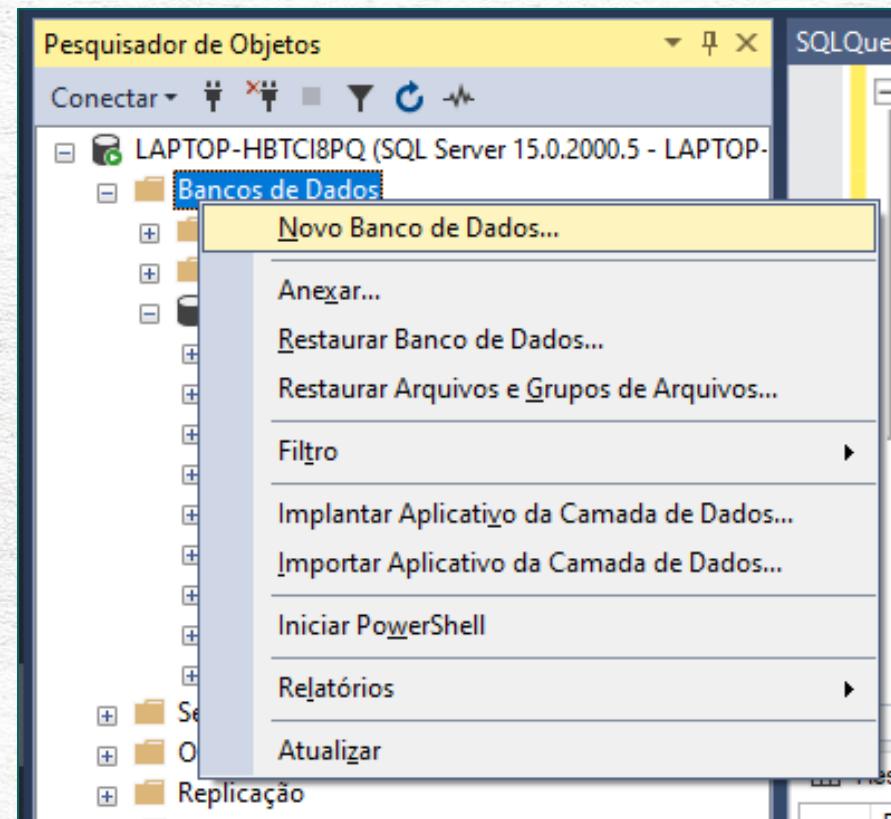
# Como criar um Banco de Dados

## CREATE DATABASE

Vimos no módulo anterior como criar VIEWS, usando o comando CREATE VIEW. Para criar um Banco de Dados, a lógica é semelhante. Isso vale para tabelas.

Existem 2 formas de criar um Banco de Dados. Na **primeira opção**, podemos criar de forma manual clicando com o botão direito em cima da pasta Banco de Dados, e depois em **Novo Banco de Dados...**. A **segunda opção** seria através de um código simples em SQL.

Manualmente

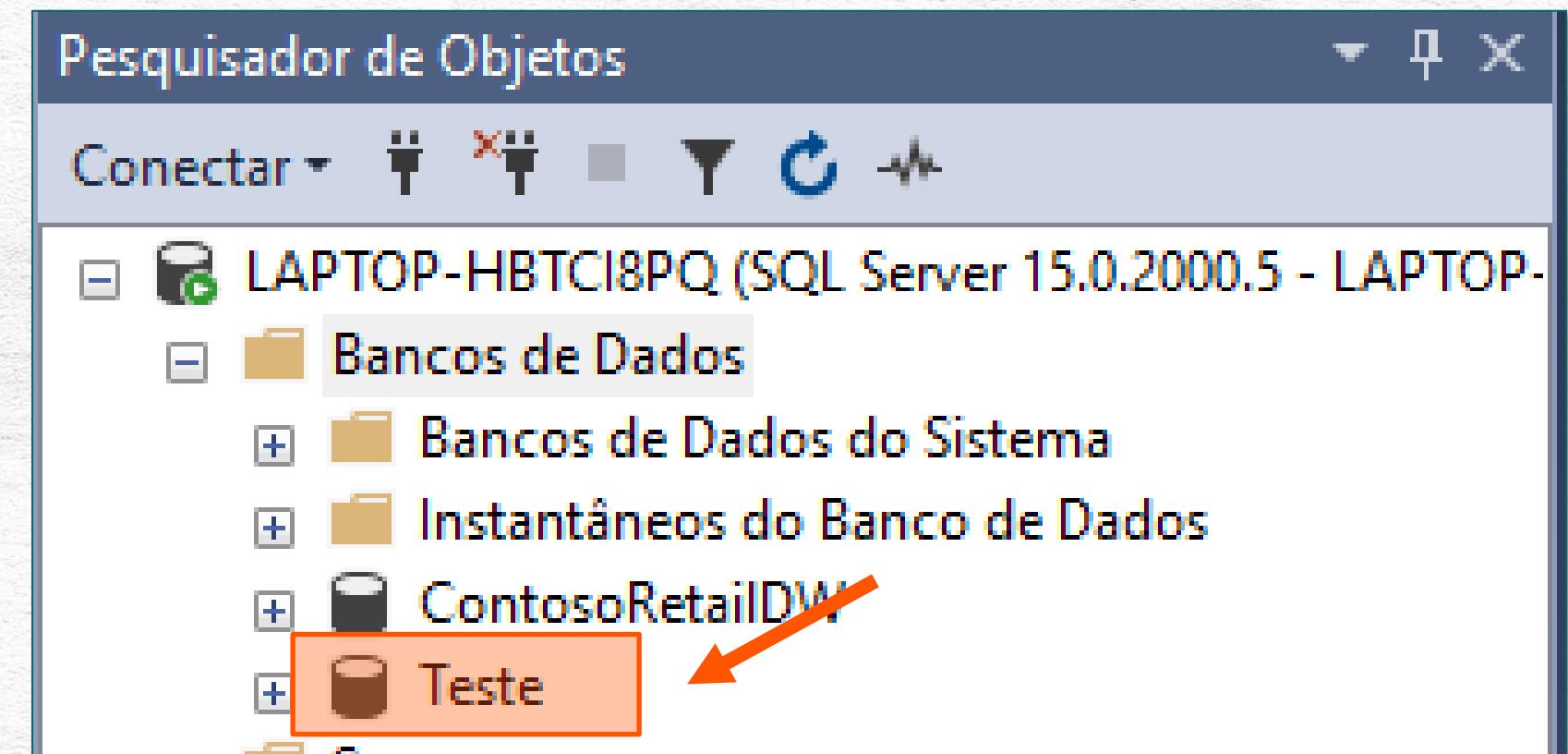


Usando SQL

A screenshot of the SSMS SQL Query window. The title bar says 'SQLQuery1.sql - LAP...us Cavalcanti (54)\*'. The main area contains the SQL command 'CREATE DATABASE Teste'.

# Como criar um Banco de Dados

Uma vez criado, o Banco de Dados aparecerá da seguinte forma:



# Como criar uma tabela

Agora que temos o nosso Banco de Dados, vamos começar a adicionar tabelas dentro dele. Mas antes, alguns pontos importantes que devemos ter em mente ao criar uma nova tabela:

1

Cada coluna da tabela vai ter um mesmo tipo de informação. Ou seja, uma coluna de Nome não pode conter outras informações além de nomes. O mesmo vale para qualquer outra coluna: é necessário seguir um padrão no tipo de informação que temos naquela coluna, nunca misturar!

CustomerKey	FirstName	BirthDate	YearlyIncome	UpdateDate
1	Jon	1966-04-08	90000,00	2009-10-01 00:00:00.000
2	Eugene	1965-05-14	60000,00	2009-10-01 00:00:00.000
3	Ruben	1965-08-12	60000,00	2009-10-01 00:00:00.000
4	Christy	1968-02-15	70000,00	2009-10-01 00:00:00.000
5	Elizabeth	1968-08-08	80000,00	2009-10-01 00:00:00.000
6	Julio	1965-08-05	70000,00	2009-10-01 00:00:00.000
7	Janet	1965-12-06	70000,00	2009-10-01 00:00:00.000
8	Marco	1964-05-09	60000,00	2009-10-01 00:00:00.000
9	Rob	1964-07-07	60000,00	2009-10-01 00:00:00.000
10	Shannon	1964-04-01	70000,00	2009-10-01 00:00:00.000



# Como criar uma tabela

Agora que temos o nosso Banco de Dados, vamos começar a adicionar tabelas dentro dele. Mas antes, alguns pontos importantes que devemos ter em mente ao criar uma nova tabela:



As tabelas criadas devem ter uma estrutura particular: para adicionar uma nova informação nessa tabela (um novo cliente) adicionamos sempre uma nova linha! Se na sua tabela você adiciona uma nova coluna sempre que precisa adicionar um novo dado, reveja a sua estrutura.



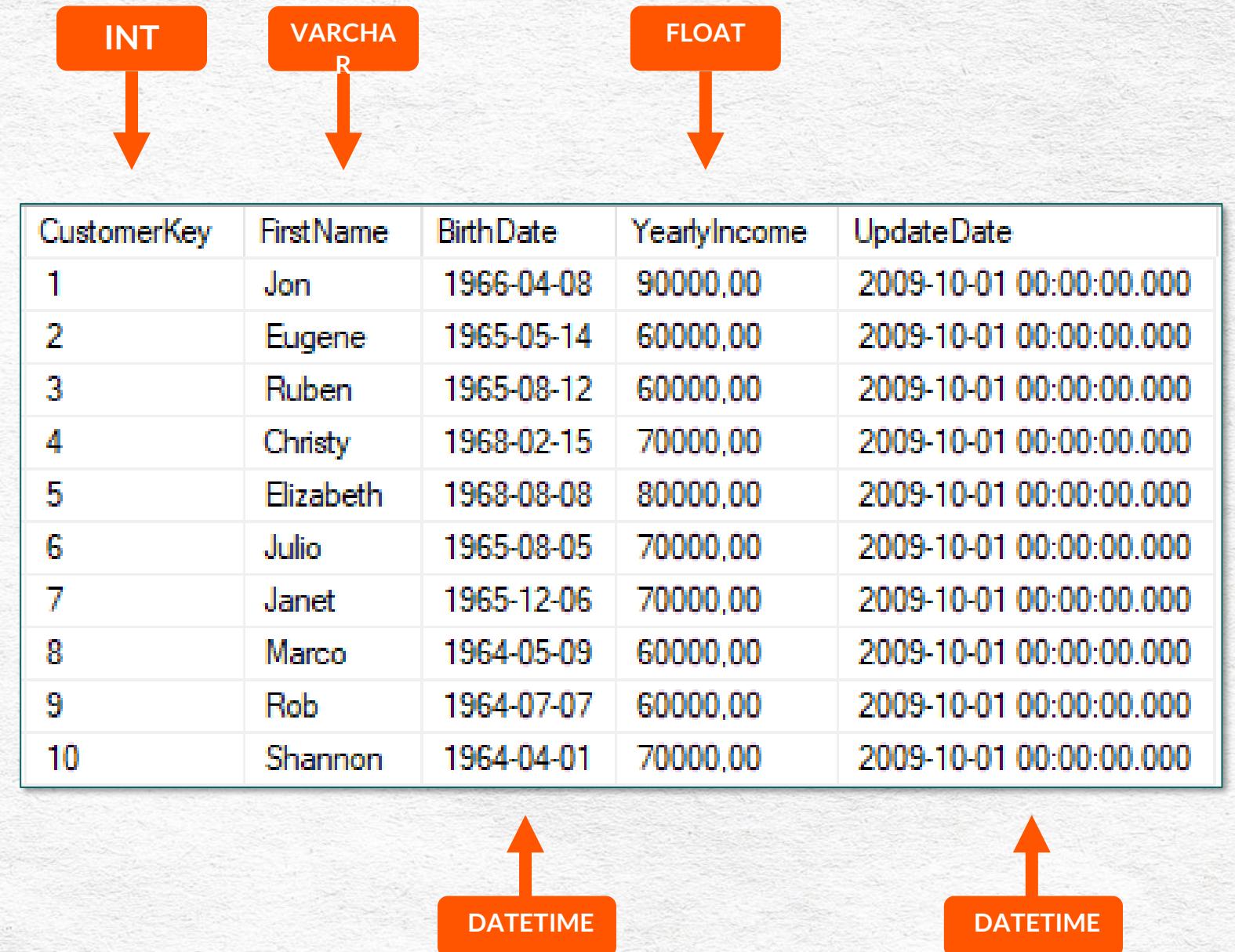
CustomerKey	FirstName	BirthDate	YearlyIncome	UpdateDate
1	Jon	1966-04-08	90000,00	2009-10-01 00:00:00.000
2	Eugene	1965-05-14	60000,00	2009-10-01 00:00:00.000
3	Ruben	1965-08-12	60000,00	2009-10-01 00:00:00.000
4	Christy	1968-02-15	70000,00	2009-10-01 00:00:00.000
5	Elizabeth	1968-08-08	80000,00	2009-10-01 00:00:00.000
6	Julio	1965-08-05	70000,00	2009-10-01 00:00:00.000
7	Jane			
8	Marc			
9	Rob			
10	Shan			
CustomerKey	FirstName	BirthDate	YearlyIncome	UpdateDate
1	Jon	1966-04-08	90000,00	2009-10-01 00:00:00.000
2	Eugene	1965-05-14	60000,00	2009-10-01 00:00:00.000
3	Ruben	1965-08-12	60000,00	2009-10-01 00:00:00.000
4	Christy	1968-02-15	70000,00	2009-10-01 00:00:00.000
5	Elizabeth	1968-08-08	80000,00	2009-10-01 00:00:00.000
6	Julio	1965-08-05	70000,00	2009-10-01 00:00:00.000
7	Janet	1965-12-06	70000,00	2009-10-01 00:00:00.000
8	Marco	1964-05-09	60000,00	2009-10-01 00:00:00.000
9	Rob	1964-07-07	60000,00	2009-10-01 00:00:00.000
10	Shannon	1964-04-01	70000,00	2009-10-01 00:00:00.000
11	Jacquelyn	1964-02-06	70000,00	2009-10-01 00:00:00.000

# Como criar uma tabela

Agora que temos o nosso Banco de Dados, vamos começar a adicionar tabelas dentro dele. Mas antes, alguns pontos importantes que devemos ter em mente ao criar uma nova tabela:

# 3

Cada coluna vai ter um mesmo tipo de dado. Ou seja, na coluna de ID, sempre teremos um número; na coluna de Nome, sempre teremos textos, na coluna de data, sempre datas, e assim vai.



# Como criar uma tabela

Para criar as tabelas nos Bancos de Dados, vamos utilizar as operações CRUD da seguinte maneira:

1

## CREATE TABLE

O comando CREATE TABLE vai nos permitir criar uma nova tabela. Ao lado, temos um exemplo da estrutura para utilização do comando. É neste momento que declaramos o tipo de cada dado.

```
CREATE TABLE Produtos(
    id_produto INT,
    nome_produto VARCHAR(50),
    data_validade DATETIME,
    preco_produto FLOAT)
```

# Como criar uma tabela

Para criar as tabelas nos Bancos de Dados, vamos utilizar as operações CRUD da seguinte maneira:



## INSERT INTO

Será através do comando **INSERT INTO** que vamos adicionar novos valores em nossa tabela. Além deste comando, precisaremos também do **VALUES** para especificar os valores adicionados.

```
INSERT INTO Produtos(id_produto, nome_produto, data_validade, preco_produto)
VALUES
    (1, 'Arroz', '2021-12-31', 22.50),
    (2, 'Feijão', '2021-12-31', 8.99)
```

# Como criar uma tabela

Para criar as tabelas nos Bancos de Dados, vamos utilizar as operações CRUD da seguinte maneira:

# 3

## UPDATE

Através do comando UPDATE conseguiremos atualizar um valor dentro de uma tabela. Geralmente esse comando é utilizado em conjunto com o WHERE, que vai nos possibilitar identificar a linha onde vamos fazer a alteração.

```
UPDATE Produtos  
SET nome_produto = 'Macarrão'  
WHERE id_produto = 3
```

# Como criar uma tabela

Para criar as tabelas nos Bancos de Dados, vamos utilizar as operações CRUD da seguinte maneira:

# 4

## DELETE

O comando DELETE é o que vai permitir a exclusão de dados em uma tabela. Observe que ainda não estamos falando de exclusão de tabelas, e sim exclusão de dados dentro de uma tabela.

```
DELETE  
FROM Produtos  
WHERE id_produto = 3
```



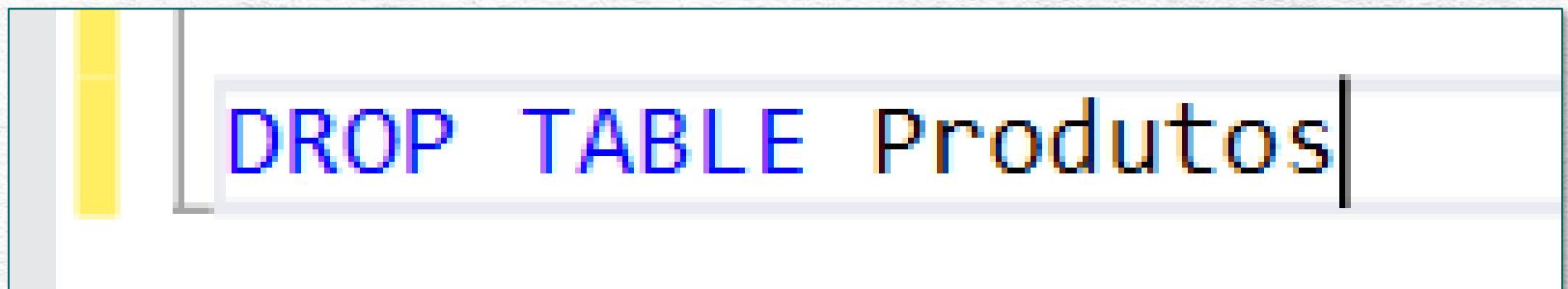
# Como criar uma tabela

Para criar as tabelas nos Bancos de Dados, vamos utilizar as operações CRUD da seguinte maneira:

5

## DROP TABLE

O comando DROP TABLE permite a exclusão definitiva de uma tabela dentro de um Banco de Dados.



# CREATE e DROP DATABASE

Vamos criar/excluir nosso primeiro Banco de Dados.

Utilize o comando **CREATE DATABASE** ao lado para criar o banco de dados 'Teste'.

Em seguida, utilize o comando **DROP DATABASE** para excluir o banco de dados criado.

Por fim, crie um novo banco de dados chamado **db\_Impressionador**. Este será o banco de dados onde criaremos nossas tabelas nas próximas aulas.

The screenshot shows the SQL Server Management Studio interface with three tabs open:

- SQLQuery6.sql - LAP...us Cavalcanti (70)\***: Contains the command to create the database 'Teste'. It is highlighted with a yellow selection bar.
- SQLQuery1.sql - LAP...us Cavalcanti (55)\***: Contains the command to drop the database 'Teste'. It is also highlighted with a yellow selection bar.
- SQLQuery2.sql -**: Contains the command to create the database 'db\_Impressionador', which is currently not highlighted.

```
-- Criando o Banco de Dados 'Teste'  
CREATE DATABASE Teste  
  
-- Excluindo o Banco de Dados 'Teste'  
DROP DATABASE Teste  
  
-- Criando o Banco de Dados 'db_Impressionador'  
CREATE DATABASE db_Impressionador
```

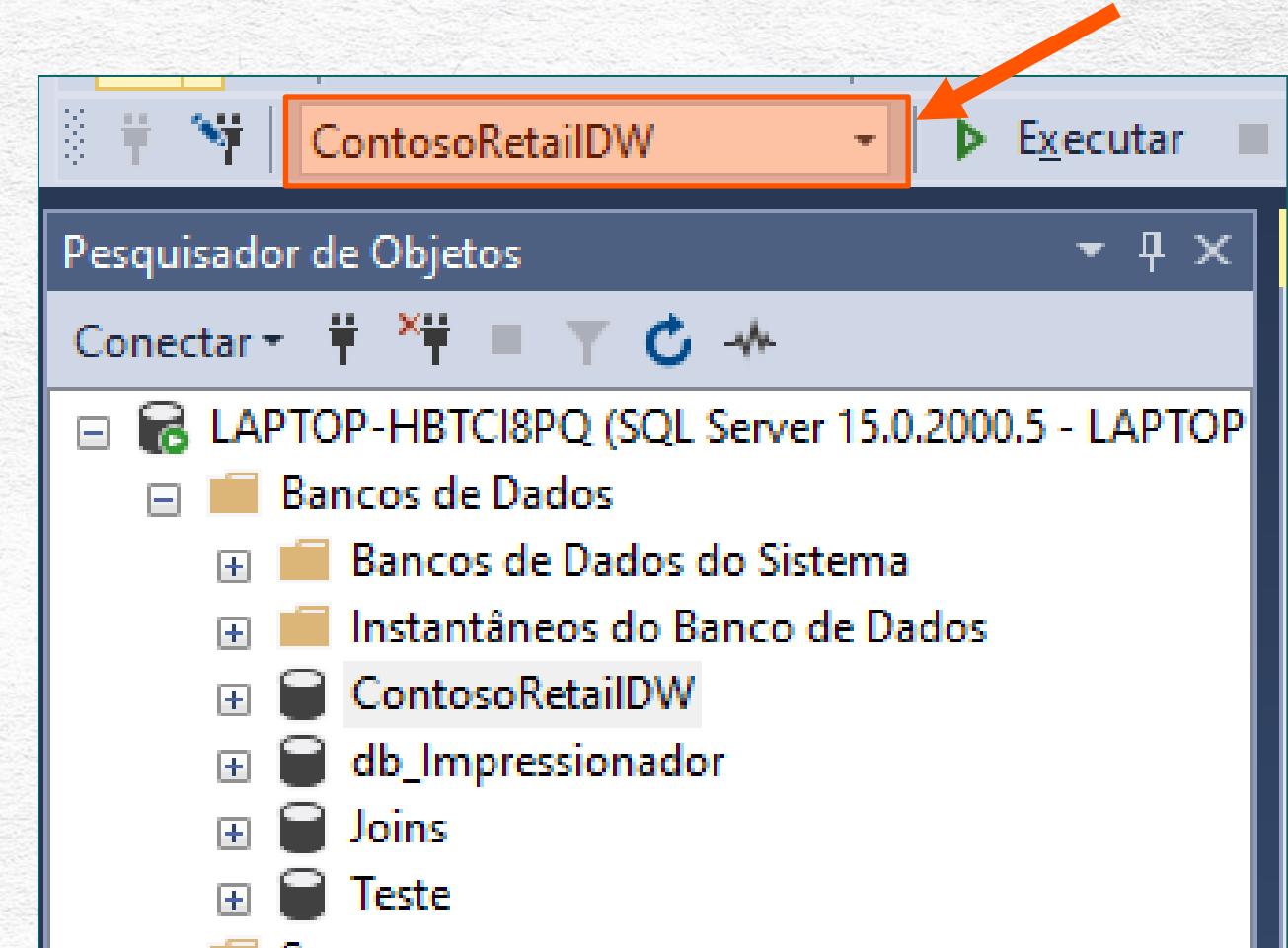
# USE DATABASE

Agora que temos mais de um banco de dados, precisamos tomar cuidado antes de criar uma nova tabela.

O que podemos fazer para garantir que a tabela será criada no banco de dados certo seria alterar o nome do banco de dados no local indicado na imagem à direita.

Porém, essa seria uma solução manual.

O ideal será utilizar o comando USE, que nos permite identificar, logo no início do nosso código, qual o Banco de Dados queremos selecionar. Isso deixará essa seleção muito mais automática e evitará qualquer problema de criação de tabelas no banco de dados errado.



```
USE db_Impressionador
```

# CREATE TABLE

Utilize o comando **CREATE TABLE** ao lado para criar a tabela 'Produtos'.

A nova tabela deverá conter 4 colunas:

- id\_produto (INT)
- nome\_produto (VARCHAR)
- data\_validade (DATETIME)
- preco\_produto (FLOAT)

```
- USE db_Impressionador  
- CREATE TABLE Produtos(  
    id_produto INT,  
    nome_produto VARCHAR(200),  
    data_validade DATETIME,  
    preco_produto FLOAT  
)
```



# INSERT INTO

Utilize o comando **INSERT INTO** ao lado para adicionar novos valores à tabela.

Um detalhe importante: não faz diferença a ordem em que os valores são adicionados dentro do comando **VALUES**, desde que essa ordem respeite a ordem das colunas dentro dos parênteses, logo após o nome da tabela.

	id_produto	nome_produto	data_validade	preco_produto
1	1	Arroz	2021-12-31 00:00:00.000	22,5
2	2	Feijão	2021-12-31 00:00:00.000	8,99

```
INSERT INTO Produtos(id_produto, nome_produto,
                     data_validade, preco_produto)
VALUES
      (1, 'Arroz', '31/12/2021', 22.50),
      (2, 'Feijão', '31/12/2021', 8.99)
```

```
INSERT INTO Produtos(id_produto, nome_produto,
                     data_validade, preco_produto)
VALUES
      (1, 'Arroz', '31/12/2021', 22.50),
      (2, 'Feijão', '31/12/2021', 8.99)
```

# Existe ordem correta para adicionar dados no INSERT?

Não existe ordem correta para adicionar os valores dentro do INSERT INTO. Observe na imagem ao lado que podemos adicionar os valores de forma diferente da ordem das colunas da tabela.

A única condição é que os valores dentro do VALUES sigam a mesma ordem dos argumentos informados dentro dos parênteses da tabela.

Inclusive, observe que podemos até mesmo não passar uma das informações (nome do produto) e não teremos nenhum erro.

```
INSERT INTO Produtos(id_produto, preco_produto,  
                     data_validade)  
VALUES  
      (3, 33.99, '31/12/2999')
```

Resultados				
	id_produto	nome_produto	data_validade	preco_produto
1	1	Amox	2021-12-31 00:00:00.000	22.5
2	2	Feijão	2021-12-31 00:00:00.000	8,99
3	3	NULL	2999-12-31 00:00:00.000	33,99

# UPDATE

Utilize o comando **UPDATE** para atualizar um dado dentro de uma tabela.

O WHERE será fundamental para que a gente possa especificar exatamente qual é o id que queremos alterar.

Observe que, após o SET, deve ser informado o nome da coluna que queremos alterar o valor (no exemplo, **nome\_produto**).

The screenshot shows a SQL query window with the following content:

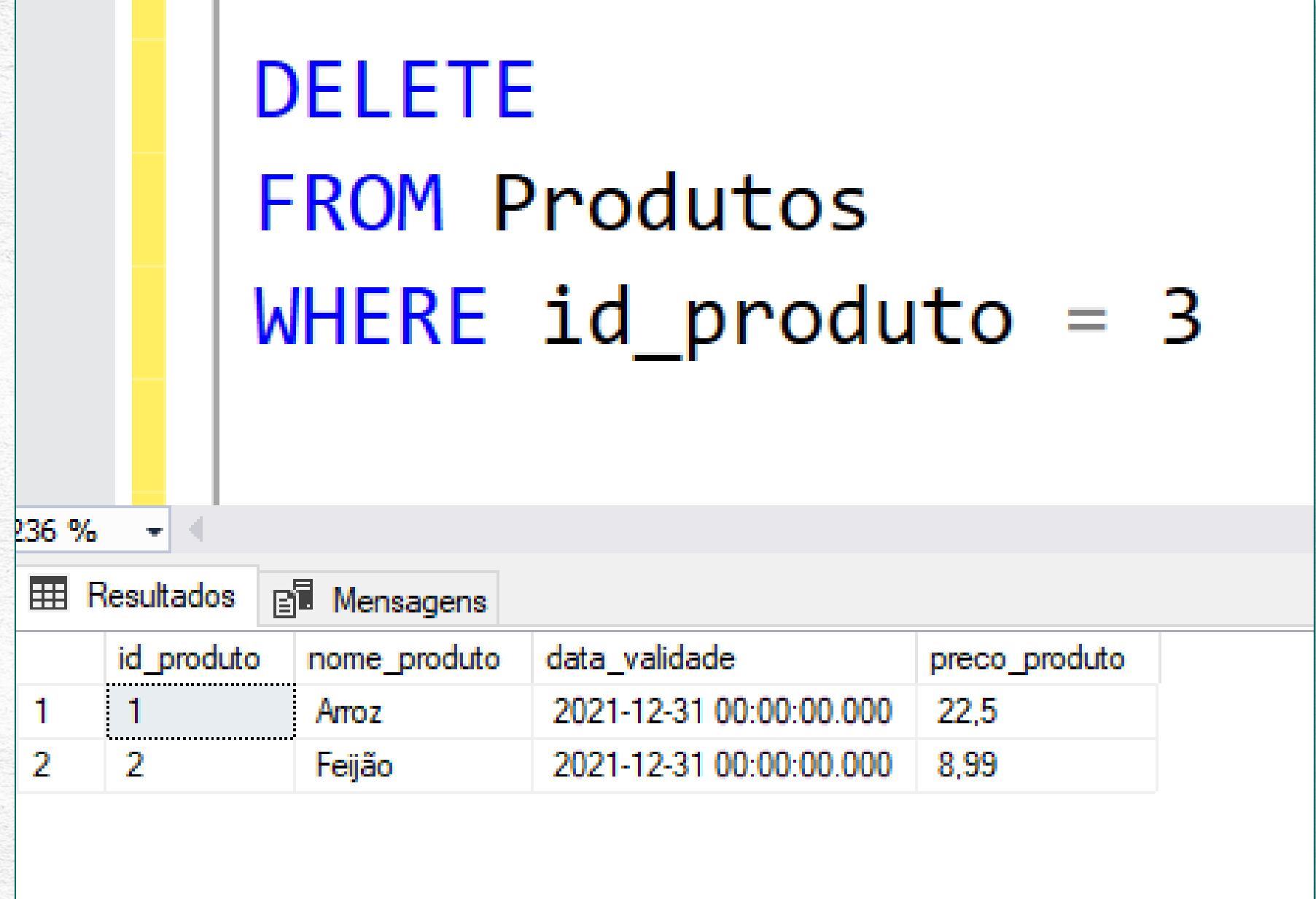
```
UPDATE Produtos
SET nome_produto = 'Macarrão'
WHERE id_produto = 3
```

The results tab displays a table with the following data:

	id_produto	nome_produto	data_validade	preco_produto
1	1	Arroz	2021-12-31 00:00:00.000	22,5
2	2	Feijão	2021-12-31 00:00:00.000	8,99
3	3	Macarrão	2999-12-31 00:00:00.000	33,99

Utilize o comando **DELETE** para deletar um dado dentro de uma tabela.

O WHERE será fundamental para que a gente possa especificar exatamente qual é o id que queremos deletar.



The screenshot shows a MySQL Workbench interface. On the left, a large blue box contains the SQL code:

```
DELETE  
FROM Produtos  
WHERE id_produto = 3
```

On the right, the results of the query are displayed in a table titled "Resultados". The table has columns: id\_produto, nome\_produto, data\_validade, and preco\_produto. It contains two rows of data:

	id_produto	nome_produto	data_validade	preco_produto
1	1	Arroz	2021-12-31 00:00:00.000	22,5
2	2	Feijão	2021-12-31 00:00:00.000	8,99

# Código para criação de uma tabela exemplo

Utilize o código ao lado para criar uma tabela exemplo de Funcionários. Essa tabela será usada para as próximas aplicações, onde veremos como alterar colunas de uma tabela.

The screenshot shows a SQL query window in SSMS with three tabs at the top: SQLQuery6.sql, SQLQuery1.sql, and SQLQuery2.sql. The SQLQuery6.sql tab is active and contains the following code:

```
USE db_Impressionador
CREATE TABLE Funcionarios(
    id_funcionario INT,
    nome_funcionario VARCHAR(100),
    salario FLOAT,
    data_nascimento DATETIME
)
INSERT INTO Funcionarios(id_funcionario, nome_funcionario, salario, data_nascimento)
VALUES
    (1, 'Lucas' , 1500, '20/03/1990'),
    (2, 'Andressa' , 2300, '07/12/1988'),
    (3, 'Felipe' , 4000, '13/02/1993'),
    (4, 'Marcelo' , 7100, '10/04/1993'),
    (5, 'Carla' , 3200, '02/09/1986'),
    (6, 'Juliana' , 5500, '21/01/1989'),
    (7, 'Mateus' , 1900, '02/11/1993'),
    (8, 'Sandra' , 3900, '09/05/1990'),
    (9, 'André' , 1000, '13/03/1994'),
    (10, 'Julio' , 4700, '05/07/1992')
```

# ALTER TABLE: Adicionar coluna

Para adicionar uma coluna, utilizamos o comando **ALTER TABLE** em conjunto com o comando ADD, assim como mostrado no print ao lado.

Em seguida, utilizamos o UPDATE em conjunto com o WHERE para atualizar os valores dessa coluna.

```
ALTER TABLE Funcionarios
ADD cargo VARCHAR(100), bonus FLOAT
```

```
UPDATE Funcionarios
SET cargo = 'Analista'
WHERE id_funcionario = 1
```

	id_funcionario	nome_funcionario	salario	data_nascimento	cargo	bonus
1	1	Lucas	1500	1990-03-20 00:00:00.000	Analista	NULL
2	2	Andressa	2300	1988-12-07 00:00:00.000	NULL	NULL
3	3	Felipe	4000	1993-02-13 00:00:00.000	NULL	NULL
4	4	Marcelo	7100	1993-04-10 00:00:00.000	NULL	NULL
5	5	Carla	3200	1986-09-02 00:00:00.000	NULL	NULL
6	6	Juliana	5500	1989-01-21 00:00:00.000	NULL	NULL
7	7	Mateus	1900	1993-11-02 00:00:00.000	NULL	NULL
8	8	Sandra	3900	1990-05-09 00:00:00.000	NULL	NULL
9	9	André	1000	1994-03-13 00:00:00.000	NULL	NULL
10	10	Julio	4700	1992-07-05 00:00:00.000	NULL	NULL

# ALTER TABLE: Alterando tipo de dados

Para alterar o tipo de dados de uma coluna, basta seguir o exemplo da imagem ao lado.

```
ALTER TABLE Funcionarios  
ALTER COLUMN salario INT
```



# ALTER TABLE: Excluindo uma coluna de uma tabela

Para excluir uma ou mais colunas de uma tabela, utilizamos o comando **DROP COLUMN**, como mostrado no exemplo ao lado.

```
ALTER TABLE Funcionarios  
DROP COLUMN cargo, bonus
```

# EXERCÍCIOS



1

2

## Questão 1

- a) Crie um banco de dados chamado BD\_Teste.
- b) Exclua o banco de dados criado no item anterior.
- c) Crie um banco de dados chamado Exercicios.

## Questão 2

No banco de dados criado no exercício anterior, crie 3 tabelas, cada uma contendo as seguintes colunas.

Tabela 1 (dCliente): ID\_Cliente, Nome\_Cliente, Data\_de\_Nascimento

Tabela 2 (dGerente): ID\_Gerente, Nome\_Gerente, Data\_de\_Contratacao, Salario

Tabela 3 (fContratos): ID\_Contrato, Data\_de\_Assinatura, ID\_Cliente, ID\_Gerente, Valor\_do\_Contrato

Lembre-se dos seguintes pontos:

- a) Garantir que o Banco de Dados Exercicios está selecionado.
- b) Definir qual será o tipo de dados mais adequado para cada coluna das tabelas. Lembrando que os tipos de dados mais comuns são: INT, FLOAT, VARCHAR e DATETIME.

Por fim, faça um SELECT para visualizar cada tabela.

# EXERCÍCIOS



3

## Questão 3

Em cada uma das 3 tabelas, adicione os seguintes valores:

**TABELA dCliente**

ID_Cliente	Nome_Cliente	Data_de_Nascimento
1	André Martins	1989-02-12 00:00:00.000
2	Bárbara Campos	1992-05-07 00:00:00.000
3	Carol Freitas	1985-04-23 00:00:00.000
4	Diego Cardoso	1994-10-11 00:00:00.000
5	Eduardo Pereira	1988-11-09 00:00:00.000
6	Fabiana Silva	1989-09-02 00:00:00.000
7	Gustavo Barb...	1993-06-27 00:00:00.000
8	Helen Viana	1990-02-11 00:00:00.000

**TABELA dGerente**

ID_Gerente	Nome_Gerente	Data_de_Contratacao	Salario
1	Lucas Sampaio	2015-03-21 00:00:00.000	6700
2	Mariana Padilha	2011-01-10 00:00:00.000	9900
3	Nathália Santos	2018-10-03 00:00:00.000	7200
4	Otávio Costa	2017-04-18 00:00:00.000	11000

**TABELA fContratos**

ID_Contrato	Data_de_Assinatura	ID_Cliente	ID_Gerente	Valor_do_Contrato
1	2019-01-12 00:00:00.000	8	1	23000
2	2019-02-10 00:00:00.000	3	2	15500
3	2019-03-07 00:00:00.000	7	2	6500
4	2019-03-15 00:00:00.000	1	3	33000
5	2019-03-21 00:00:00.000	5	4	11100
6	2019-03-23 00:00:00.000	4	2	5500
7	2019-03-28 00:00:00.000	9	3	55000
8	2019-04-04 00:00:00.000	2	1	31000
9	2019-04-05 00:00:00.000	10	4	3400
10	2019-04-05 00:00:00.000	6	2	9200

# EXERCÍCIOS



4

## Questão 4

Novos dados deverão ser adicionados nas tabelas dCliente, dGerente e fContratos. Fique livre para adicionar uma nova linha em cada tabela contendo, respectivamente,

- (1) um novo cliente (id cliente, nome e data de nascimento)
- (2) um novo gerente (id gerente, nome, data de contratação e salário)
- (3) um novo contrato (id, data assinatura, id cliente, id gerente, valor do contrato)

5

## Questão 5

O contrato de ID igual a 4 foi registrado com alguns erros na tabela fContratos. Faça uma alteração na tabela atualizando os seguintes valores:

Data\_de\_Assinatura: 17/03/2019  
ID\_Gerente: 2  
Valor\_do\_Contrato: 33500

6

## Questão 6

Delete a linha da tabela fContratos que você criou na questão 4.

# GABARITOS

1

## Questão 1

- a) Crie um banco de dados chamado BD\_Teste.
- b) Exclua o banco de dados criado no item anterior.
- c) Crie um banco de dados chamado Exercicios.

The screenshot shows a SQL Server Management Studio window with two tabs: 'Coluna.sql - LAPTOP...us Cavalcanti (59)\*' and 'SQLQuery1.sql - LAP...'. The 'SQLQuery1.sql' tab contains the following SQL code:

```
-- Questão 1
-- a)
CREATE DATABASE BD_Teste

-- b)
DROP DATABASE BD_Teste

-- c)
CREATE DATABASE Exercicios
```

# GABARITOS

2

## Questão 2

No banco de dados criado no exercício anterior, crie 3 tabelas, cada uma contendo as seguintes colunas.

Tabela 1 (dCliente): ID\_Cliente, Nome\_Cliente, Data\_de\_Nascimento

Tabela 2 (dGerente): ID\_Gerente, Nome\_Gerente, Data\_de\_Contratacao, Salario

Tabela 3 (fContratos): ID\_Contrato, Data\_de\_Assinatura, ID\_Cliente, ID\_Gerente, Valor\_do\_Contrato

Lembre-se dos seguintes pontos:

- Garantir que o Banco de Dados Exercicios está selecionado.
- Definir qual será o tipo de dados mais adequado para cada coluna das tabelas. Lembrando que os tipos de dados mais comuns são: INT, FLOAT, VARCHAR e DATETIME.

Por fim, faça um SELECT para visualizar cada tabela.

```

Coluna.sql - LAPTOP...us Cavalcanti (59)*  X  SQLQuery1.sql - LA
-- Questão 2

USE Exercicios

CREATE TABLE dCliente(
    ID_Cliente INT,
    Nome_Cliente VARCHAR(100),
    Data_de_Nascimento DATETIME
)

CREATE TABLE dGerente(
    ID_Gerente INT,
    Nome_Gerente VARCHAR(100),
    Data_de_Contratacao DATETIME,
    Salario FLOAT
)

CREATE TABLE fContratos(
    ID_Contrato INT,
    Data_de_Assinatura DATETIME,
    ID_Cliente INT,
    ID_Gerente INT,
    Valor_do_Contrato FLOAT
)

SELECT * FROM dCliente
SELECT * FROM dGerente
SELECT * FROM fContratos

```

# GABARITOS

3

## Questão 3

Em cada uma das 3 tabelas, adicione os seguintes valores:

TABELA dCliente

ID_Cliente	Nome_Cliente	Data_de_Nascimento
1	André Martins	1989-02-12 00:00:00.000
2	Bárbara Campos	1992-05-07 00:00:00.000
3	Carol Freitas	1985-04-23 00:00:00.000
4	Diego Cardoso	1994-10-11 00:00:00.000
5	Eduardo Pereira	1988-11-09 00:00:00.000
6	Fabiana Silva	1989-09-02 00:00:00.000
7	Gustavo Barb...	1993-06-27 00:00:00.000
8	Helen Viana	1990-02-11 00:00:00.000

TABELA fContratos

ID_Contrato	Data_de_Assinatura	ID_Cliente	ID_Gerente	Valor_do_Contrato
1	2019-01-12 00:00:00.000	8	1	23000
2	2019-02-10 00:00:00.000	3	2	15500
3	2019-03-07 00:00:00.000	7	2	6500
4	2019-03-15 00:00:00.000	1	3	33000
5	2019-03-21 00:00:00.000	5	4	11100
6	2019-03-23 00:00:00.000	4	2	5500
7	2019-03-28 00:00:00.000	9	3	55000
8	2019-04-04 00:00:00.000	2	1	31000
9	2019-04-05 00:00:00.000	10	4	3400
10	2019-04-05 00:00:00.000	6	2	9200

TABELA dGerente

ID_Gerente	Nome_Gerente	Data_de_Contratacao	Salario
1	Lucas Sampaio	2015-03-21 00:00:00.000	6700
2	Mariana Padilha	2011-01-10 00:00:00.000	9900
3	Nathália Santos	2018-10-03 00:00:00.000	7200
4	Otávio Costa	2017-04-18 00:00:00.000	11000

```

Coluna.sql - LAPTOP...us Cavalcanti (59)* ⇢ X SQLQuery1.sql - LAP...us Cavalcanti (51))
└-- Questão 3
    -- Inserindo valores na tabela dCliente
    INSERT INTO dCliente(ID_Cliente, Nome_Cliente, Data_de_Nascimento)
    VALUES
        (1, 'André Martins', '12/02/1989'),
        (2, 'Bárbara Campos', '07/05/1992'),
        (3, 'Carol Freitas', '23/04/1985'),
        (4, 'Diego Cardoso', '11/10/1994'),
        (5, 'Eduardo Pereira', '09/11/1988'),
        (6, 'Fabiana Silva', '02/09/1989'),
        (7, 'Gustavo Barbosa', '27/06/1993'),
        (8, 'Helen Viana', '11/02/1990'),
        (9, 'Igor Castro', '21/08/1989'),
        (10, 'Juliana Pires', '13/01/1991')

    -- Inserindo valores na tabela dGerente
    INSERT INTO dGerente(ID_Gerente, Nome_Gerente, Data_de_Contratacao, Salario)
    VALUES
        (1, 'Lucas Sampaio', '21/03/2015', 6700),
        (2, 'Mariana Padilha', '10/01/2011', 9900),
        (3, 'Nathália Santos', '03/10/2018', 7200),
        (4, 'Otávio Costa', '18/04/2017', 11000)

    -- Inserindo valores na tabela fContratos
    INSERT INTO fContratos(ID_Contrato, Data_de_Assinatura, ID_Cliente, ID_Gerente, Valor_do_Contrato)
    VALUES
        (1, '12/01/2019', 8, 1, 23000),
        (2, '10/02/2019', 3, 2, 15500),
        (3, '07/03/2019', 7, 2, 6500),
        (4, '15/03/2019', 1, 3, 33000),
        (5, '21/03/2019', 5, 4, 11100),
        (6, '23/03/2019', 4, 2, 5500),
        (7, '28/03/2019', 9, 3, 55000),
        (8, '04/04/2019', 2, 1, 31000),
        (9, '05/04/2019', 10, 4, 3400),
        (10, '05/04/2019', 6, 2, 9200)

```

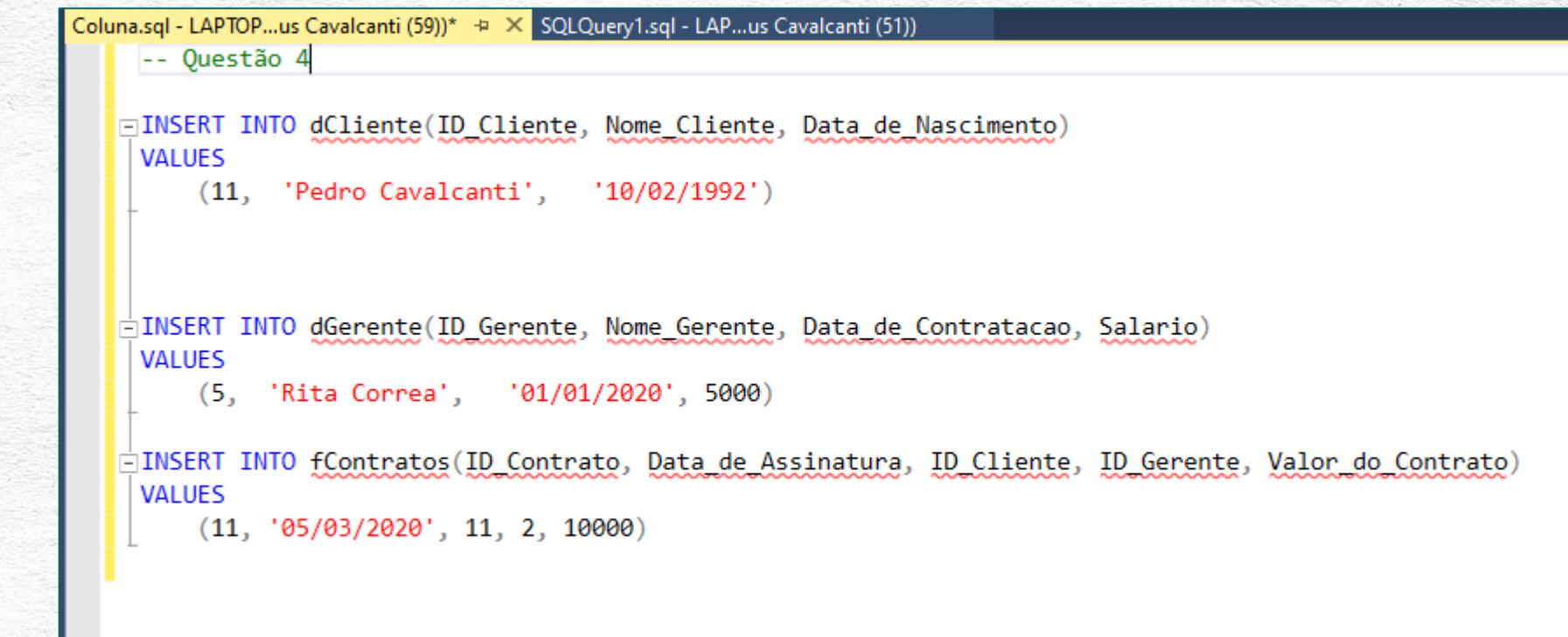
# GABARITOS

4

## Questão 4

Novos dados deverão ser adicionados nas tabelas dCliente, dGerente e fContratos. Fique livre para adicionar uma nova linha em cada tabela contendo, respectivamente,

- (1) um novo cliente (id cliente, nome e data de nascimento)
- (2) um novo gerente (id gerente, nome, data de contratação e salário)
- (3) um novo contrato (id, data assinatura, id cliente, id gerente, valor do contrato)



```
-- Questão 4

INSERT INTO dCliente(ID_Cliente, Nome_Cliente, Data_de_Nascimento)
VALUES
    (11, 'Pedro Cavalcanti', '10/02/1992')

INSERT INTO dGerente(ID_Gerente, Nome_Gerente, Data_de_Contratacao, Salario)
VALUES
    (5, 'Rita Correa', '01/01/2020', 5000)

INSERT INTO fContratos(ID_Contrato, Data_de_Assinatura, ID_Cliente, ID_Gerente, Valor_do_Contrato)
VALUES
    (11, '05/03/2020', 11, 2, 10000)
```

# GABARITOS

5

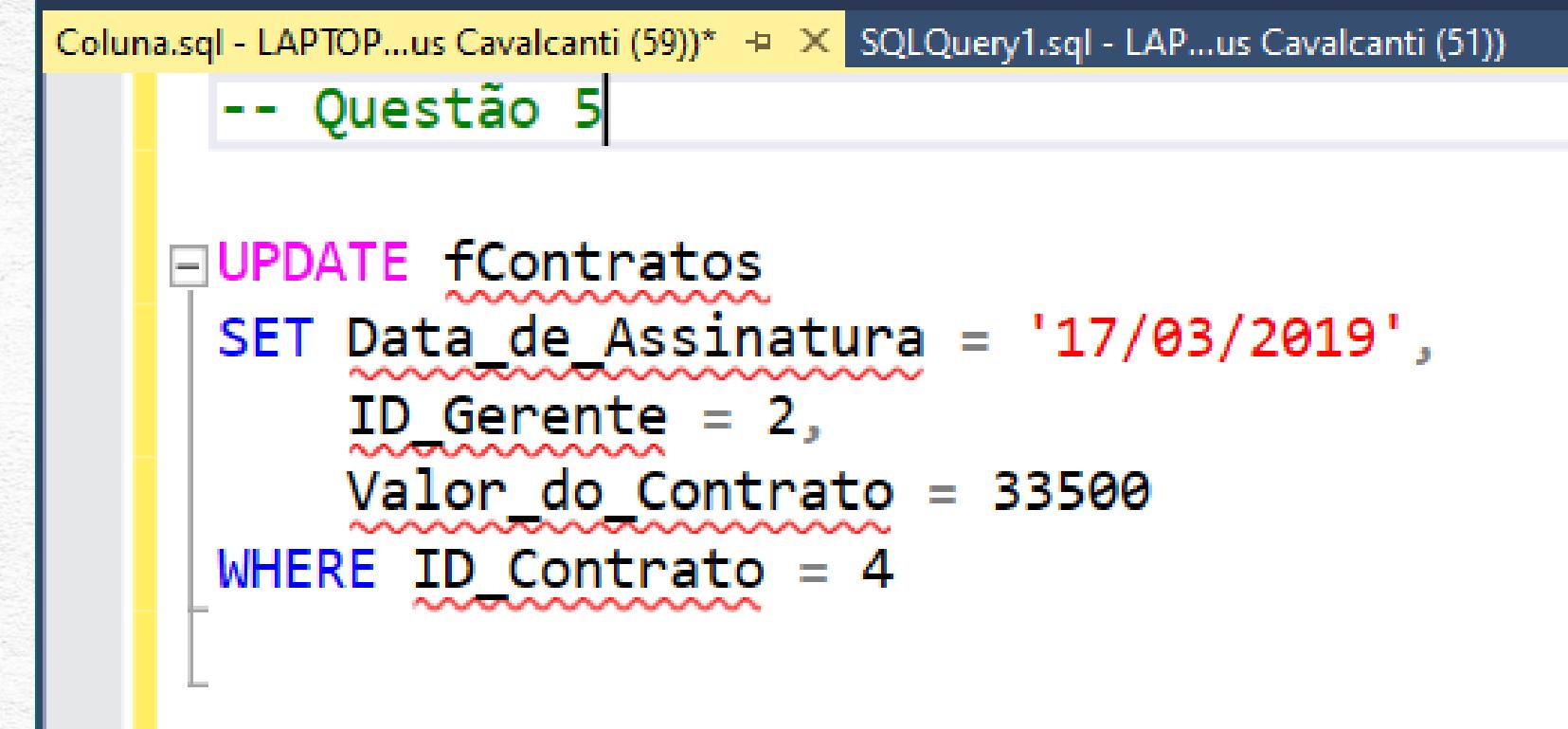
## Questão 5

O contrato de ID igual a 4 foi registrado com alguns erros na tabela fContratos. Faça uma alteração na tabela atualizando os seguintes valores:

Data\_de\_Assinatura: 17/03/2019

ID\_Gerente: 2

Valor\_do\_Contrato: 33500



The screenshot shows a SQL Server Management Studio window with two tabs: 'Coluna.sql - LAPTOP...us Cavalcanti (59)\*' and 'SQLQuery1.sql - LAP...us Cavalcanti (51)'. The 'SQLQuery1.sql' tab contains the following SQL code:

```
-- Questão 5
UPDATE fContratos
SET Data_de_Assinatura = '17/03/2019',
ID_Gerente = 2,
Valor_do_Contrato = 33500
WHERE ID_Contrato = 4
```

# GABARITOS

6

## Questão 6

Delete a linha da tabela fContratos que você criou na questão 4.

The screenshot shows a SQL Server Management Studio interface with two windows. The title bar indicates 'Coluna.sql - LAPTOP...us Cavalcanti (59)\*' and 'SQLQuery1.sql - LAP...us Caval'. The main window displays the following SQL code:

```
-- Questão 6
DELETE
FROM fContratos
WHERE ID_Contrato = 11
```

# SQL

## SUBQUERIES



# Introdução

## O QUE É UMA SUBQUERY?

Uma Subconsulta (ou Subquery ou SubSELECT) nada mais é do que uma consulta dentro de outra consulta. Ou seja, com uma subquery conseguimos utilizar o resultado de uma query (consulta) dentro de outra consulta.

O exemplo abaixo ilustra bem a ideia principal de uma subquery. Para descobrir os produtos que têm o UnitPrice maior do que a média, podemos adicionar um SELECT dentro de outro SELECT para criar uma consulta mais otimizada.

```
-- Descobrir os produtos que têm um UnitPrice
-- maior do que a média.
-- (Sem subquery)

-- 1) Primeiro descobrimos a média de UnitPrice
SELECT AVG(UnitPrice) FROM dimProduct

-- 2) Depois utilizamos o resultado dentro do filtro WHERE
SELECT
  *
FROM dimProduct
WHERE UnitPrice > 356.8301
```



```
-- Descobrir os produtos que têm um UnitPrice
-- maior do que a média.
-- (Com subquery)

-- 1) Podemos pular o passo 1 direto para a consulta principal
SELECT
  *
FROM dimProduct
WHERE UnitPrice > (SELECT AVG(UnitPrice) FROM dimProduct)
```

# Onde utilizamos subqueries

Utilizamos subqueries nas seguintes situações:

1

## Subqueries com WHERE (escalar)

Nesta situação, utilizamos o resultado de um SELECT para filtrar um outro SELECT.

Observe no print ao lado. Temos um SELECT mais interno sendo utilizado com o filtro WHERE de uma consulta principal.

Para este caso, utilizamos o termo “escalar” para dizer que o resultado do SELECT mais interno é um valor único, e não uma lista de valores.

```
SELECT  
    Coluna1,  
    Coluna2  
FROM  
    Tabela  
WHERE Coluna1 = (SELECT)
```



# Onde utilizamos subqueries

Utilizamos subqueries nas seguintes situações:



## Subqueries com WHERE (lista)

Aqui temos uma situação semelhante à anterior, com a diferença de que o resultado da subquery seria uma lista de vários valores que poderiam ser passados no WHERE.

Neste caso, como o filtro seria de mais de 1 valor, utilizamos o IN.

```
SELECT  
    Coluna1,  
    Coluna2  
FROM  
    Tabela  
WHERE Coluna1 IN (SELECT)
```

# Onde utilizamos subqueries

Utilizamos subqueries nas seguintes situações:

# 3

## Subqueries dentro de um SELECT

Outra opção seria utilizar o resultado de um SELECT como uma nova coluna de uma query principal.

```
SELECT  
    Coluna1,  
    Coluna2,  
    (SELECT)  
FROM  
    Tabela
```

# Onde utilizamos subqueries

Utilizamos subqueries nas seguintes situações:

4

## Subqueries junto com um FROM

Podemos utilizar o resultado de um SELECT como uma tabela de um outro SELECT.

Neste caso, precisamos utilizar o comando AS para dar um nome para essa tabela.

```
SELECT  
Coluna1,  
Coluna2  
FROM  
(SELECT) AS T
```

# Subquery na prática: Aplicação com WHERE (Exemplo 1)

Para entender a ideia por trás das subqueries, vamos começar fazendo 3 exemplos.

No **Exemplo 1**, imagine que você queira fazer uma consulta à tabela DimProduct e considerar apenas os produtos com preços acima da média. Como poderíamos fazer isso?

O **primeiro passo** seria descobrir essa média de preço. Fazemos isso através da consulta simples abaixo:

```
SELECT  
    AVG(UnitPrice)  
FROM DimProduct
```



	(Nenhum nome de coluna)
1	356,8301

O **segundo passo** seria utilizar esse resultado em um filtro WHERE na tabela DimProduct.

```
SELECT  
    *  
FROM DimProduct  
WHERE UnitPrice > 356.8301
```

# Subquery na prática: Aplicação com WHERE (Exemplo 1)

Fazer dessa forma não é muito automático, pois escrevemos manualmente a média de preço no filtro. Se os preços alterarem ou se tivermos adição ou exclusão de produtos, essa filtragem não será automática. O valor de 356.8301 continuará fixo mesmo que a média se altere.

O ideal então seria utilizar uma subquery, substituindo o valor fixo de 356.8301 pelo SELECT que criamos para descobrir essa média:

```
SELECT
*
FROM DimProduct
WHERE UnitPrice > (SELECT AVG(UnitPrice) FROM DimProduct)
```

Na solução acima, utilizamos uma subquery, ou seja, um SELECT dentro de outro SELECT. Dessa forma, o cálculo da média de preço ficará 100% automático, sem necessidade de atualizar manualmente o valor.

Vamos agora para o exemplo 2.

# Subquery na prática: Aplicação com WHERE (Exemplo 2)

No Exemplo 2 queremos filtrar nossa tabela DimProduct e retornar os produtos da categoria 'Televisions'. Porém, não temos a informação de Nome da Subcategoria na tabela DimProduct.

Dessa forma, precisaremos criar um SELECT que descubra o ID da categoria 'Televisions', e passar esse resultado como o valor que queremos filtrar dentro do WHERE.

O resultado final ficaria da seguinte forma:

```
SELECT * FROM dimProduct
WHERE ProductSubcategoryKey =
  (SELECT ProductSubcategoryKey FROM DimProductSubcategory
  WHERE ProductSubcategoryName = 'Televisions')
```

Finalmente, vamos para o nosso exemplo 3.

# Subquery na prática: Aplicação com WHERE (Exemplo 3)

No Exemplo 3 queremos filtrar nossa tabela **FactSales\_JUL2007** (código mostrado mais a frente) e mostrar apenas as vendas referentes às lojas com 100 ou mais funcionários.

O primeiro passo é descobrir os IDs dessas lojas, através da consulta abaixo:

Agora, vamos utilizar o resultado do SELECT anterior dentro do filtro da tabela **factSales\_JUL2007**:

```
SELECT StoreKey
FROM DimStore
WHERE EmployeeCount >= 100
```

The screenshot shows a SQL query in a code editor. The query selects the 'StoreKey' column from the 'DimStore' table where the 'EmployeeCount' is greater than or equal to 100. Below the code is a results grid titled 'Resultados' showing three rows of data:

	StoreKey
1	199
2	200
3	280

```
SELECT *
FROM factSales_JUL2007
WHERE StoreKey IN (199, 200, 280)
```

The screenshot shows a SQL query in a code editor. The query selects all columns (\*) from the 'factSales\_JUL2007' table where the 'StoreKey' is in the list (199, 200, 280). The 'factSales\_JUL2007' table name and the 'IN' clause are highlighted in red, indicating they are part of the current query being demonstrated.

# Subquery na prática: Aplicação com WHERE (Exemplo 3)

Obviamente, a consulta anterior não é a mais otimizada, pois os Ids das lojas com 100 ou mais funcionários estão fixados. Se uma nova loja passar a ter mais do que 100 funcionários, a forma como a gente fez não vai servir.

Portanto, utilizamos como uma subquery o SELECT onde descobrimos os Ids das lojas, tornando nossa consulta mais automática:

```
SELECT * FROM factSales_JUL2007
WHERE StoreKey IN (
    SELECT
        StoreKey
    FROM DimStore
    WHERE EmployeeCount >= 100
)
ORDER BY StoreKey DESC
```

(Código para criar a factSales\_JUL2007 na próxima página)

# Subquery na prática: Aplicação com WHERE (Exemplo 3)

```
CREATE TABLE factSales_JUL2007(
    SalesKey int,
    DateKey datetime,
    channelKey int,
    StoreKey int,
    ProductKey int,
    UnitCost decimal(9,2),
    UnitPrice decimal(9,2),
    SalesQuantity int,
    TotalCost decimal(9,2),
    SalesAmount decimal(9,2)
)

INSERT INTO factSales_JUL2007(SalesKey, DateKey, channelKey, StoreKey, ProductKey, UnitCost, UnitPrice, SalesQuantity, TotalCost, SalesAmount)
SELECT SalesKey, DateKey, channelKey, StoreKey, ProductKey, UnitCost, UnitPrice, SalesQuantity, TotalCost, SalesAmount FROM FactSales
WHERE DateKey >= '20070701' AND DateKey <= '20070731'

select * from factSales_JUL2007
order by DateKey ASC

SELECT * FROM factSales_JUL2007
WHERE StoreKey IN (
    SELECT
        StoreKey
    FROM DimStore
    WHERE EmployeeCount >= 100
)
ORDER BY StoreKey DESC
```



# ANY e SOME

Os operadores ANY e SOME são equivalentes e permitem realizar uma comparação entre um único valor de uma coluna e um intervalo de outros valores. O =ANY e =SOME são equivalentes. Quando usados com o sinal lógico =, eles possuem a mesma finalidade do operador IN, permitindo retornar todas as linhas que sejam iguais à lista de valores especificados.

**Exemplo:**

**=ANY(valor1, valor2, valor3)**

Retorna todas as linhas da tabela que sejam iguais ao valor1 OU valor2 OU valor3.

O >ANY e >SOME retornam todas as linhas da tabela que tenham valores que sejam maiores a qualquer um dos valores da lista de valores especificados.

**Exemplo:**

**>ANY(valor1, valor2, valor3)**

Retorna todas as linhas da tabela com valores maiores que o valor1 OU valor2 OU valor3. Ou seja, maior que o menor de todos esses valores.

# ANY, SOME e ALL

O <ANY e <SOME retornam todas as linhas da tabela que tenham valores que sejam menores que qualquer um dos valores da lista de valores especificados.

**Exemplo:**

<ANY(valor1, valor2, valor3)

Retorna todas as linhas da tabela com valores menores que o valor1 OU valor2 OU valor3. Ou seja, menor que o maior de todos esses valores.

O operador ALL também permite uma comparação entre valores especificados, mas para que a consulta retorne algum resultado, é necessário que o valor comparado seja maior ou menor do que TODOS os valores da lista. (Obs: dificilmente utilizamos =ALL pois estaríamos comparando um valor e esperando que ele fosse igual a todos os valores da lista. A menos que todos os valores da lista fossem iguais, o =ALL não faria sentido)

#### Exemplo:

>ALL(valor1, valor2, valor3)

Retorna todas as linhas da tabela que sejam maiores que os valores valor1 E valor2 E valor3. Ou seja, as linhas que possuem valores maiores que o máximo.

#### Exemplo:

<ALL(valor1, valor2, valor3)

Retorna todas as linhas da tabela com valores menores que o valor1 E valor2 E valor3. Ou seja, menor que o menor de todos esses valores.

# EXISTS

O operador EXISTS é usado para testar a existência de qualquer registro (linha) em uma subconsulta.

Pensando em um exemplo prático, imagina que a gente queira descobrir quais são os produtos que possuem vendas em um determinado dia, por exemplo, 01/01/2007.

Neste caso, poderíamos utilizar o EXISTS como mostrado ao lado.

The screenshot shows a SQL Server Management Studio window with two tabs: 'Coluna.sql - LAPTOP...us Cavalcanti (59)\*' and 'SQLQuery1.sql - LAP...us Cavalcanti (51)'. The 'SQLQuery1.sql' tab contains the following query:

```
SELECT ProductKey, ProductName
FROM DimProduct
WHERE EXISTS(
    SELECT ProductKey
    FROM factSales
    WHERE DateKey = '01/01/2007'
    AND factSales.ProductKey = DimProduct.ProductKey)
```

The results pane displays a table with columns 'ProductKey' and 'ProductName', containing 8 rows of data:

	ProductKey	ProductName
1	67	NT Bluetooth Stereo Headphones E52 Black
2	69	NT Bluetooth Stereo Headphones E52 Pink
3	244	Contoso Home Theater System 5.1 Channe...
4	246	Contoso Home Theater System 4.1 Channe...
5	346	Fabrikam Laptop15.4 M5400 White
6	405	Prosware Laptop15 M510 Black
7	603	Contoso Projector 480p M480 Silver
8	737	Prosware Color Ink Jet Fax, Copier, Phone...

At the bottom of the results pane, a message says 'Consulta executada com êxito.' (Query executed successfully).

# Subquery na prática: Aplicação com SELECT

Podemos também utilizar subqueries para criar novas colunas dentro de uma consulta.

Repare o exemplo ao lado: queremos descobrir qual foi o total de vendas de cada produto da nossa tabela DimProduct.

Para isso, adicionamos uma subquery (um novo SELECT) como sendo uma coluna extra da nossa consulta principal.

Assim, além de mostrar as colunas ProductKey e ProductName da tabela dimProduct, adicionamos também uma coluna que faz a contagem de vendas na tabela FactSales.

The screenshot shows the SQL Server Management Studio interface. The top bar has two tabs: 'Coluna.sql - LAPTOP-HB...us Cavalcanti (59)\*' and 'SQLQuery1.sql - LAP...us Cavalcanti (51)'. The main area contains the following SQL code:

```
SELECT
    ProductKey,
    ProductName,
    (SELECT
        COUNT(ProductKey)
    FROM FactSales
    WHERE FactSales.ProductKey = DimProduct.ProductKey)
FROM
    DimProduct
```

Below the code is a results grid titled 'Resultados' (Results). It displays eight rows of data:

	ProductKey	ProductName	(Nenhum nome de coluna)
1	139	Adventure Works 32" LCD HDTV M130 White	805
2	140	Adventure Works 32" LCD HDTV M130 Brown	803
3	657	Proseware Duplex Scanner M200 Black	720
4	658	Proseware High Speed Laser M2000 Black	1742
5	840	Contoso Laptop Cooling Hub notebook fan with 4 p...	667
6	843	Contoso Bright Light battery E20 blue	1795
7	1058	A. Datum SLR Camera M138 Silver Grey	988
8	1243	Fabrikam Social Videographer 1" 25mm E400 White	1378

At the bottom of the results grid, a message says 'Consulta executada com êxito.' (Query executed successfully.). The status bar at the bottom right shows 'LAPTOP-HBTCI8PQ (15.0 RTM) | LAPTOP-HBTCI8PQ\Marcus... | ContosoRetailDW | 00:00:08 | 2.517 linhas'.

# Subquery na prática: Aplicação com FROM

No exemplo ao lado, vemos duas formas de descobrir o total de produtos da marca Contoso.

Na opção 1, fizemos uma contagem na tabela DimProduct e por fim um filtro na coluna BrandName.

Já na opção 2, utilizamos a ideia de subqueries: a contagem foi feita não na tabela DimProduct, e sim em uma tabela que foi filtrada antes de ser aplicada no FROM.

The screenshot shows two SQL queries in a dual-pane interface of SQL Server Management Studio. The top pane is titled 'Coluna.sql - LAPTOP...us Cavalcanti (59)\*' and the bottom pane is titled 'SQLQuery1.sql - LAP...us Cavalcanti (51)'. Both panes contain identical code:

```
-- Opção 1
SELECT
    COUNT(*)
FROM DimProduct
WHERE BrandName = 'Contoso'

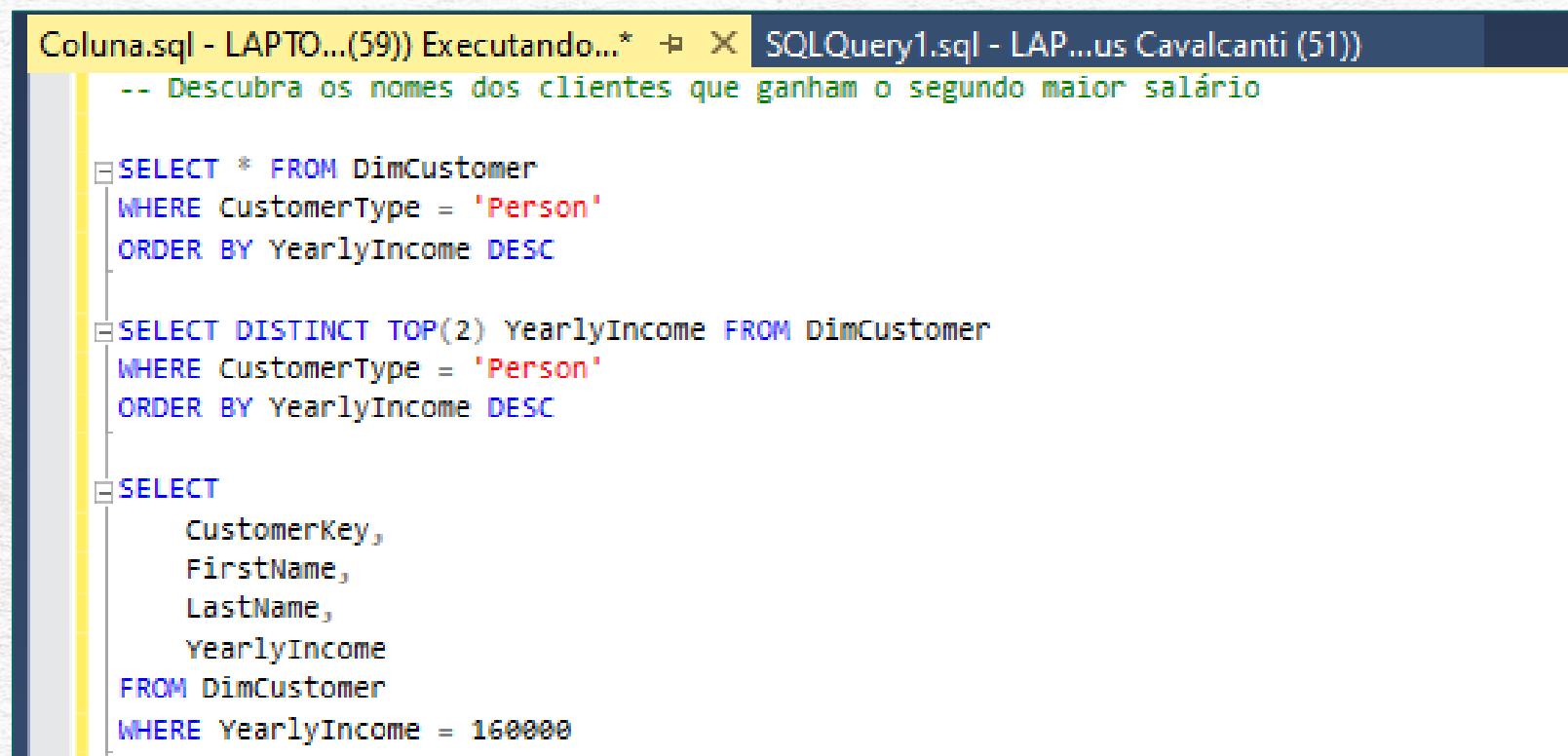
-- Opção 2
SELECT
    COUNT(*)
FROM (SELECT
        *
    FROM DimProduct
    WHERE BrandName = 'Contoso') AS T
```

Both panes show the results of the query in a 'Resultados' grid. The first row contains '(Nenhum nome de coluna)' and the second row contains '1' followed by '710'.

# Subqueries aninhadas

Também é possível criar subqueries aninhadas, ou seja, subqueries dentro de subqueries. Para entender como vai funcionar, vamos fazer o seguinte exemplo: descobrir os nomes dos clientes que ganham o segundo maior salário.

Poderíamos descobrir o segundo maior salário aplicando um TOP 2 + ORDER BY. Feito isso, descobriríamos que o segundo maior salário é \$160.000 e poderíamos utilizar esse resultado para filtrar a nossa tabela DimCustomer e descobrir os clientes que recebem o 2º maior salário. **Mas como tornar isso mais automático?**



The screenshot shows a SQL Server Management Studio window with three queries:

```
-- Descubra os nomes dos clientes que ganham o segundo maior salário

SELECT * FROM DimCustomer
WHERE CustomerType = 'Person'
ORDER BY YearlyIncome DESC

SELECT DISTINCT TOP(2) YearlyIncome FROM DimCustomer
WHERE CustomerType = 'Person'
ORDER BY YearlyIncome DESC

SELECT
    CustomerKey,
    FirstName,
    LastName,
    YearlyIncome
FROM DimCustomer
WHERE YearlyIncome = 160000
```

# Subqueries aninhadas

Para resolver esse problema utilizando subqueries, teríamos que dividi-lo em etapas:

- 1. Descobrir o maior salário
- 2. Descobrir o segundo maior salário
- 3. Descobrir os nomes dos clientes que ganham o segundo maior salário

# Subqueries aninhadas

A solução é mostrada ao lado. Repare que temos 3 SELECTs ao todo.

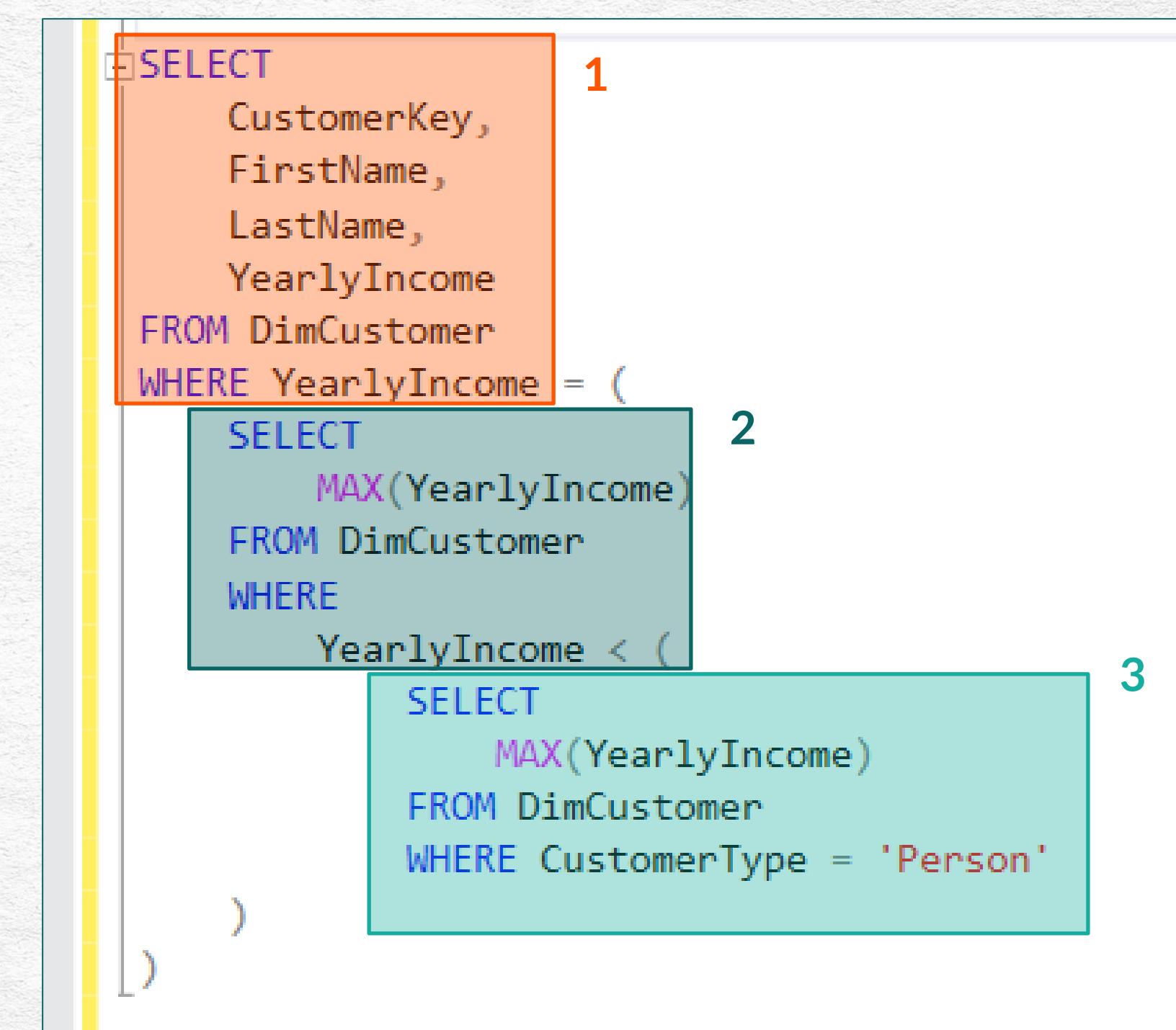
```
SELECT
    CustomerKey,
    FirstName,
    LastName,
    YearlyIncome
FROM DimCustomer
WHERE YearlyIncome = (
    SELECT
        MAX(YearlyIncome)
    FROM DimCustomer
    WHERE
        YearlyIncome < (
            SELECT
                MAX(YearlyIncome)
            FROM DimCustomer
            WHERE CustomerType = 'Person'
        )
    )
```

# Subqueries aninhadas

O SELECT mais interno (3) é responsável por retornar pra gente o maior de todos os salários.

Com este resultado, criamos um SELECT intermediário (2) para descobrir qual é o salário máximo, mas que seja menor que o máximo descoberto em (3). Este seria exatamente o segundo maior salário.

Por fim, utilizamos o resultado para filtrar a coluna de YearlyIncome do SELECT (1), e finalmente chegamos no resultado desejado: a lista de clientes que recebem o segundo maior salário.

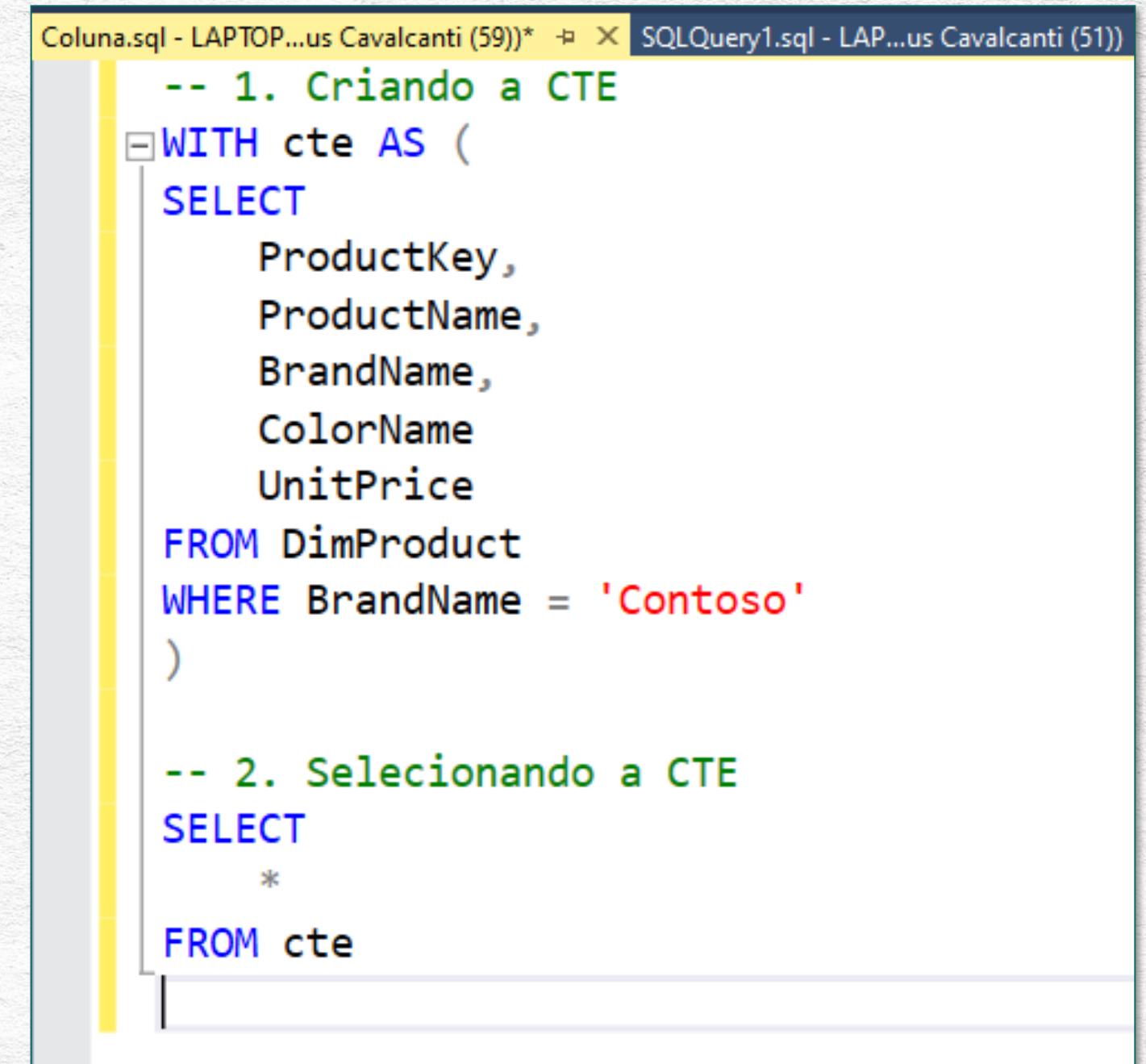


# CTEs (Common Table Expression)

Uma CTE se trata de uma tabela temporária que pode ser criada e reaproveitada em nosso código.

Ao lado, temos um exemplo de uma CTE. A estrutura para cria-la é bem simples. Utilizamos o comando `WITH`, seguido do nome da CTE, e em seguida o `AS` para especificar o que vai compor essa CTE. No exemplo ao lado, a CTE vai armazenar determinadas colunas da tabela `DimProduct`, mas considerando apenas a marca Contoso.

Em seguida, podemos selecionar essa CTE utilizando o comando `SELECT`.



```
-- 1. Criando a CTE
WITH cte AS (
    SELECT
        ProductKey,
        ProductName,
        BrandName,
        ColorName,
        UnitPrice
    FROM DimProduct
    WHERE BrandName = 'Contoso'
)

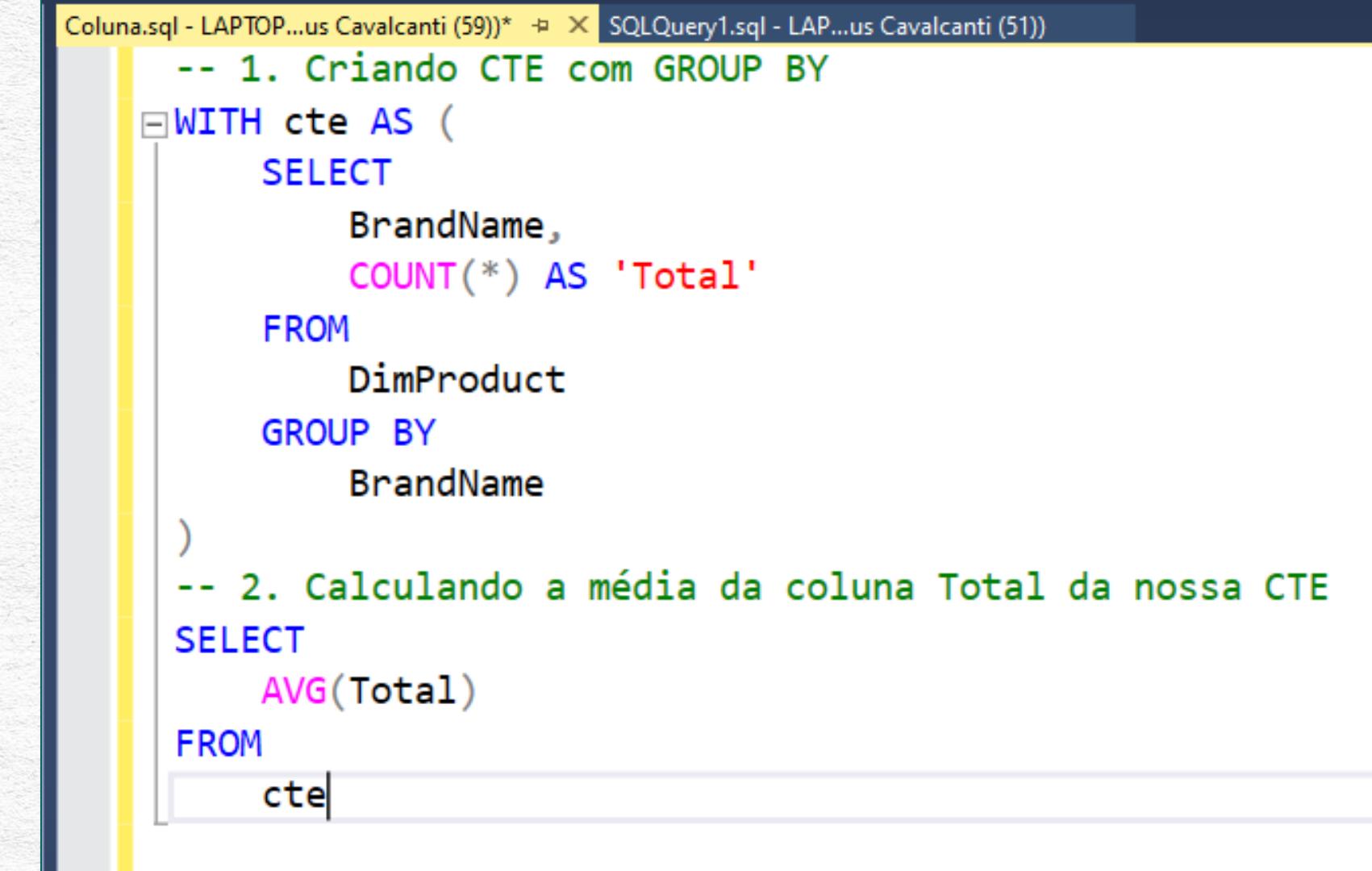
-- 2. Selecionando a CTE
SELECT *
FROM cte
```

# Calculando agregações com CTEs

Podemos armazenar consultas mais complexas dentro de uma CTE.

No exemplo ao lado, criamos uma CTE que é o agrupamento da tabela DimProduct, contendo o total de produtos por marca.

Por fim, podemos descobrir a média de produtos por marca, aplicando o AVG na coluna Total da nossa CTE.



```
Coluna.sql - LAPTOP...us Cavalcanti (59)* ⇡ X SQLQuery1.sql - LAP...us Cavalcanti (51))
-- 1. Criando CTE com GROUP BY
WITH cte AS (
    SELECT
        BrandName,
        COUNT(*) AS 'Total'
    FROM
        DimProduct
    GROUP BY
        BrandName
)
-- 2. Calculando a média da coluna Total da nossa CTE
SELECT
    AVG(Total)
FROM
    cte
```

# Nomeando colunas de uma CTE

Também seria possível criar uma CTE especificando os nomes das suas colunas.

No exemplo ao lado, temos a CTE com colunas chamadas ‘Marca’ e ‘Total’. Posteriormente, podemos fazer um SELECT referenciando o nome delas.

```
Coluna.sql - LAPTOP...us Cavalcanti (59)* ⇢ X SQLQuery1.sql - LAP...us Cavalcanti (51))
-- 1. Criando CTE e nomeando as colunas
WITH cte(Marca, Total) AS (
    SELECT
        BrandName,
        COUNT(*)
    FROM
        DimProduct
    GROUP BY
        BrandName
)
-- 2. Selecionando a nossa CTE
SELECT
    Marca,
    Total
FROM
    cte|
```

# Criando múltiplas CTE's

Podemos criar múltiplas CTEs e em seguida criar uma relação entre elas.

Observe na consulta ao lado: criamos duas CTEs (produtos\_contoso e vendas\_top100) e por fim, realizamos um JOIN entre essas CTEs.

```
Coluna.sql - LAPTOP...us Cavalcanti (59)* ⇔ X SQLQuery1.sql - LAP...us Cavalcanti (51))
-- 1. Criando a CTE produtos_contoso
WITH produtos_contoso AS (
    SELECT
        ProductKey,
        ProductName,
        BrandName
    FROM
        DimProduct
    WHERE
        BrandName = 'Contoso'
),
-- 2. Criando a CTE vendas_top100
vendas_top100 AS (
    SELECT TOP(100)
        SalesKey,
        ProductKey,
        DateKey,
        SalesQuantity
    FROM
        FactSales
    ORDER BY
        DateKey DESC
)
-- 3. Realizando um JOIN entre as duas CTEs
SELECT * FROM vendas_top100
INNER JOIN produtos_contoso
    ON vendas_top100.ProductKey = produtos_contoso.ProductKey
```



# EXERCÍCIOS



1

## Questão 1

Para fins fiscais, a contabilidade da empresa precisa de uma tabela contendo todas as vendas referentes à loja 'Contoso Orlando Store'. Isso porque essa loja encontra-se em uma região onde a tributação foi modificada recente.

2

## Questão 2

O setor de controle de produtos quer fazer uma análise para descobrir quais são os produtos que possuem um UnitPrice maior que o UnitPrice do produto de ID igual a 1893.

- A sua consulta resultante deve conter as colunas ProductKey, ProductName e UnitPrice da tabela DimProduct.
- Nessa query você também deve retornar uma coluna extra, que informe o UnitPrice do produto 1893.

# EXERCÍCIOS



3

## Questão 3

A empresa Contoso criou um programa de bonificação chamado “**Todos por 1**”. Este programa consistia no seguinte: 1 funcionário seria escolhido ao final do ano como o funcionário destaque, só que a bonificação seria recebida por todos da área daquele funcionário em particular. O objetivo desse programa seria o de incentivar a colaboração coletiva entre os funcionários de uma mesma área. Desta forma, o funcionário destaque beneficiaria não só a si, mas também a todos os colegas de sua área.

Ao final do ano, o funcionário escolhido como destaque foi o **Miguel Severino**. Isso significa que todos os funcionários da área do Miguel seriam beneficiados com o programa.

O seu objetivo é realizar uma consulta à tabela **DimEmployee** e retornar todos os funcionários da área “vencedora” para que o setor Financeiro possa realizar os pagamentos das bonificações.

# EXERCÍCIOS



4

## Questão 4

Faça uma query que retorne os clientes que recebem um salário anual acima da média. A sua query deve retornar as colunas CustomerKey, FirstName, LastName, EmailAddress e YearlyIncome.

Obs: considere apenas os clientes que são 'Pessoas Físicas'.

5

## Questão 5

A ação de desconto da Asian Holiday Promotion foi uma das mais bem sucedidas da empresa. Agora, a Contoso quer entender um pouco melhor sobre o perfil dos clientes que compraram produtos com essa promoção.

Seu trabalho é criar uma query que retorne a lista de clientes que compraram nessa promoção.

# EXERCÍCIOS



6

## Questão 6

A empresa implementou um programa de fidelização de clientes empresariais. Todos aqueles que comprarem mais de 3000 unidades de um mesmo produto receberá descontos em outras compras.

Você deverá descobrir as informações de CustomerKey e CompanyName destes clientes.

7

## Questão 7

Você deverá criar uma consulta para o setor de vendas que mostre as seguintes colunas da tabela DimProduct:

ProductKey, ProductName, BrandName, UnitPrice, Média de UnitPrice.

8

## Questão 8

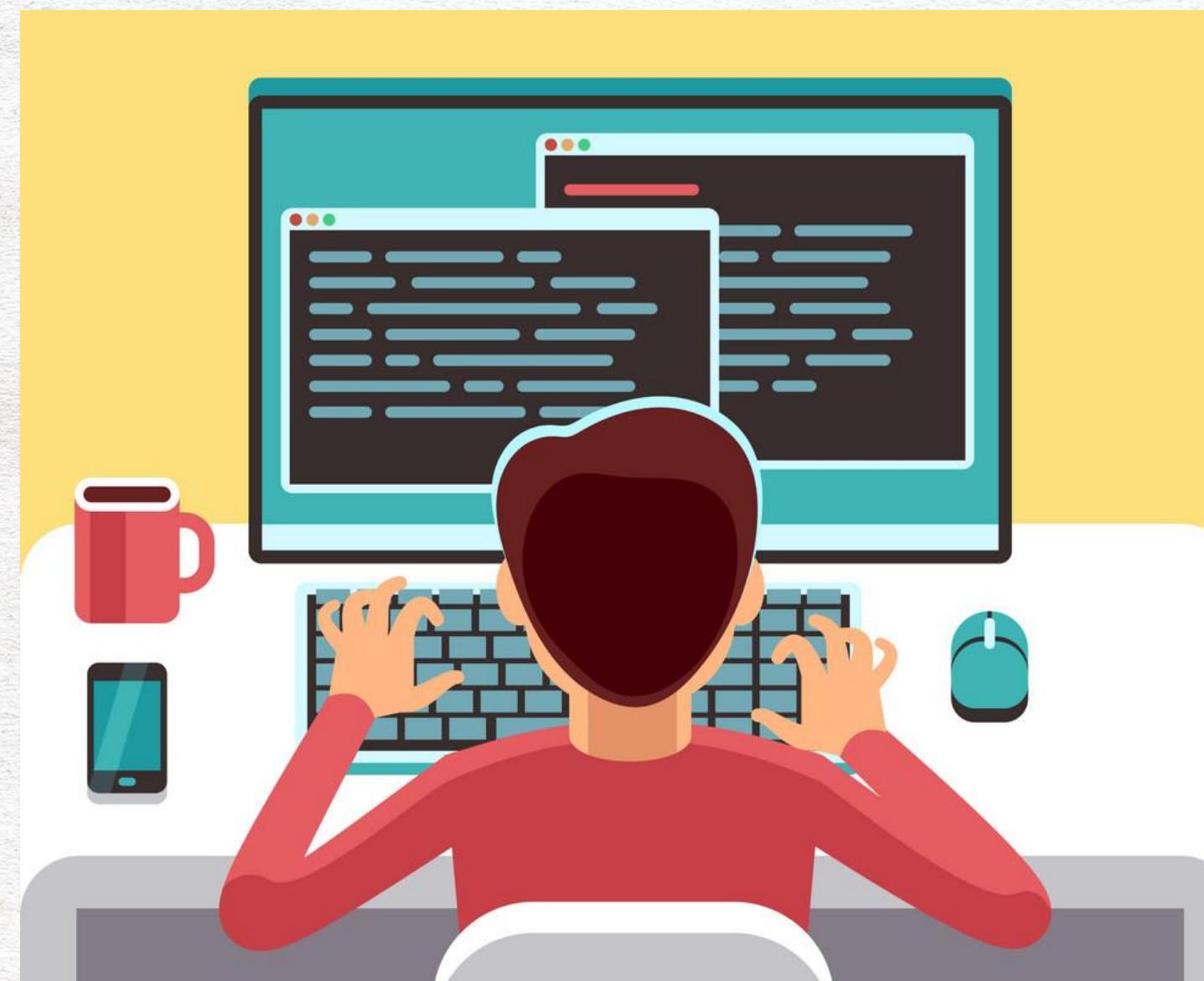
Faça uma consulta para descobrir os seguintes indicadores dos seus produtos:

Maior quantidade de produtos por marca

Menor quantidade de produtos por marca

Média de produtos por marca

# EXERCÍCIOS



9

10

## Questão 9

Crie uma CTE que seja o agrupamento da tabela DimProduct, armazenando o total de produtos por marca. Em seguida, faça um SELECT nesta CTE, descobrindo qual é a quantidade máxima de produtos para uma marca. Chame esta CTE de **CTE\_QtdProdutosPorMarca**.

## Questão 10

Crie duas CTEs:

(i) a primeira deve conter as colunas ProductKey, ProductName, ProductSubcategoryKey, BrandName e UnitPrice, da tabela DimProduct, mas apenas os produtos da marca Adventure Works. Chame essa CTE de **CTE\_ProdutosAdventureWorks**.

(ii) a segunda deve conter as colunas ProductSubcategoryKey, ProductSubcategoryName, da tabela DimProductSubcategory mas apenas para as subcategorias 'Televisions' e 'Monitors'. Chame essa CTE de **CTE\_CategoriaTelevisionsERadio**.

Faça um Join entre essas duas CTEs, e o resultado deve ser uma query contendo todas as colunas das duas tabelas. Observe nesse exemplo a diferença entre o LEFT JOIN e o INNER JOIN.

# GABARITOS

1

## Questão 1

Para fins fiscais, a contabilidade da empresa precisa de uma tabela contendo todas as vendas referentes à loja 'Contoso Orlando Store'. Isso porque essa loja encontra-se em uma região onde a tributação foi modificada recente.

Portanto, crie uma consulta ao Banco de Dados para obter uma tabela FactSales contendo todas as vendas desta loja.

The screenshot shows a SQL query window titled "SQLQuery1.sql - M...MARCUS\caval (53)\*". The query uses a Common Table Expression (CTE) to select all sales from the "FactSales" table where the store key matches a specific store key found in the "DimStore" table. The result set contains 11 rows of sales data, including columns like SalesKey, DateKey, channelKey, StoreKey, ProductKey, PromotionKey, CurrencyKey, UnitCost, UnitPrice, SalesQuantity, ReturnQuantity, and ReturnAmount.

```

SELECT *
FROM FactSales
WHERE StoreKey = (
    SELECT
        StoreKey
    FROM DimStore
    WHERE StoreName = 'Contoso Orlando Store'
)

```

SalesKey	DateKey	channelKey	StoreKey	ProductKey	PromotionKey	CurrencyKey	UnitCost	UnitPrice	SalesQuantity	ReturnQuantity	ReturnAmount	
1	276687	2009-06-13 00:00:00.000	1	110	1495	1	105,77	230,00	10	0	0,00	
2	278188	2008-02-14 00:00:00.000	1	110	791	11	13,75	29,90	9	0	0,00	
3	278247	2009-09-09 00:00:00.000	1	110	2494	21	1,50	2,94	480	0	0,00	
4	278296	2007-10-05 00:00:00.000	1	110	1986	1	71,37	139,99	10	1	139,99	
5	278389	2008-07-07 00:00:00.000	1	110	1161	12	1	404,68	880,00	12	0	0,00
6	278926	2008-01-05 00:00:00.000	1	110	320	11	1	321,44	699,00	9	0	0,00
7	279028	2007-09-02 00:00:00.000	1	110	903	3	1	38,74	75,99	6	0	0,00
8	279788	2007-03-30 00:00:00.000	1	110	2467	2	1	15,80	30,99	4	1	30,99
9	280286	2009-04-05 00:00:00.000	1	110	724	1	1	74,96	163,00	10	1	163,00
10	280406	2009-02-13 00:00:00.000	1	110	341	20	1	444,69	967,00	9	0	0,00
11	281313	2007-12-23 00:00:00.000	1	110	906	4	1	38,74	75,99	6	0	0,00

Consulta executada com êxito.

# GABARITOS

2

## Questão 2

O setor de controle de produtos quer fazer uma análise para descobrir quais são os produtos que possuem um UnitPrice maior que o UnitPrice do produto de ID igual a 1893.

- a) A sua consulta resultante deve conter as colunas ProductKey, ProductName e UnitPrice da tabela DimProduct.
- b) Nessa query você também deve retornar uma coluna extra, que informe o UnitPrice do produto 1893.

The screenshot shows a SQL query window titled "SQLQuery1.sql - M...MARCUS\caval (53)\*". The query uses a Common Table Expression (CTE) to find products with a unit price higher than the one for product ID 1893. The results grid below shows 12 rows of data from the DimProduct table, including columns for ProductKey, ProductName, UnitPrice, and a fourth column labeled "(Nenhum nome de coluna)".

```

SELECT
    ProductKey,
    ProductName,
    UnitPrice,
    (
        SELECT
            UnitPrice
        FROM DimProduct
        WHERE ProductKey = 1893
    ) AS ReferenceUnitPrice
FROM DimProduct
WHERE UnitPrice > (
    SELECT
        UnitPrice
    FROM DimProduct
    WHERE ProductKey = 1893
)

```

	ProductKey	ProductName	UnitPrice	(Nenhum nome de coluna)
1	145	Adventure Works 52" LCD HDTV X590 Silver	2899,99	1989,00
2	146	Adventure Works 52" LCD HDTV X590 Black	2899,99	1989,00
3	147	Adventure Works 52" LCD HDTV X590 White	2899,99	1989,00
4	148	Adventure Works 52" LCD HDTV X590 Brown	2899,99	1989,00
5	539	Proseware Projector 1080p LCD86 Black	2295,00	1989,00
6	540	Proseware Projector 1080p DLP86 Black	2499,00	1989,00
7	551	Proseware Projector 1080p LCD86 White	2295,00	1989,00
8	552	Proseware Projector 1080p DLP86 White	2499,00	1989,00
9	563	Proseware Projector 1080p LCD86 Silver	2295,00	1989,00
10	564	Proseware Projector 1080p DLP86 Silver	2499,00	1989,00
11	575	Contoso Projector 1080p X980 Black	2295,00	1989,00
12	576	Contoso Projector 1080p X981 Black	2499,00	1989,00

Consulta executada com êxito. | MARCUS (15.0 RTM) | MARCUS\caval (53) | ContosoRetailDW | 00:00:02 | 54 linhas

# GABARITOS

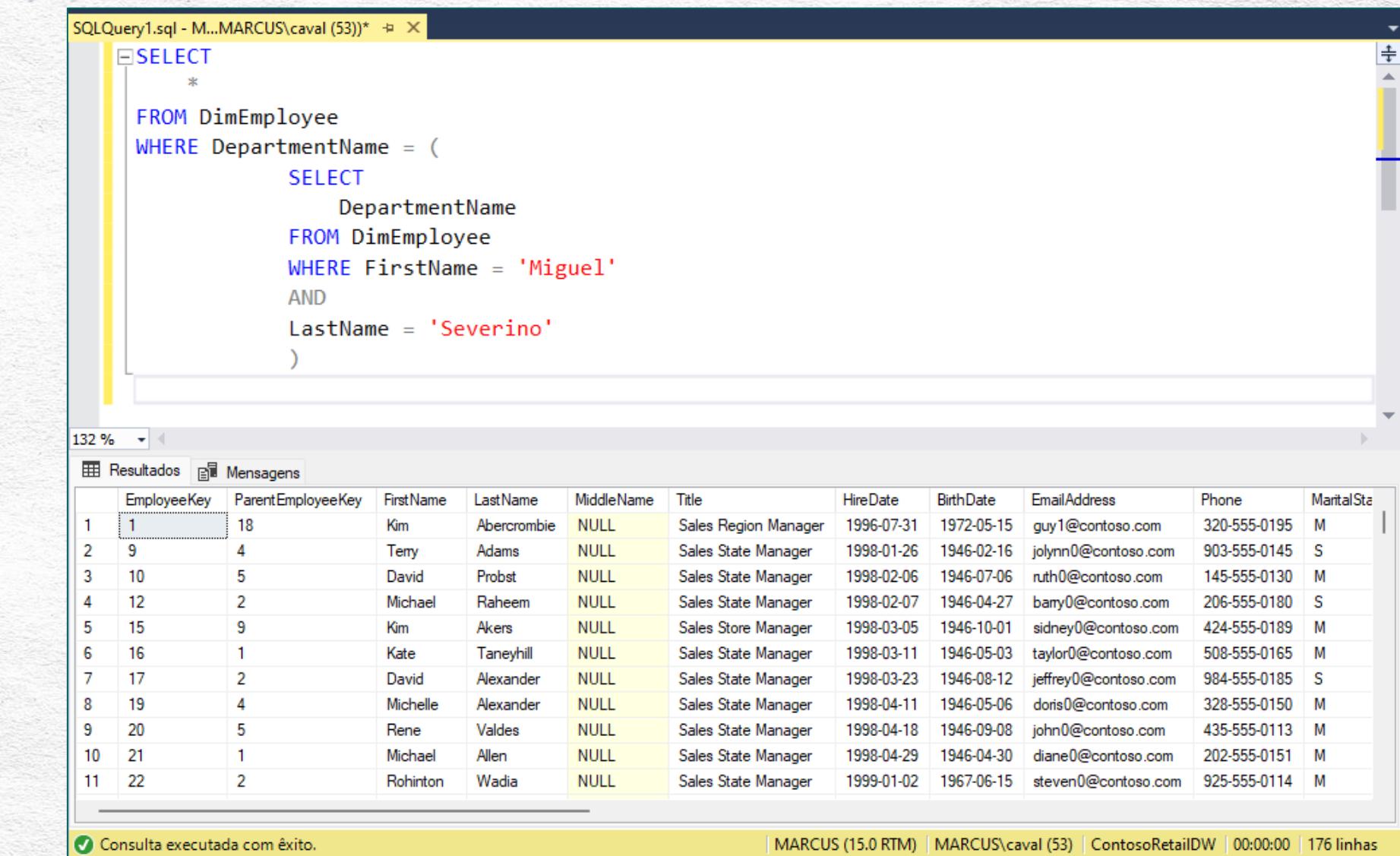
3

## Questão 3

A empresa Contoso criou um programa de bonificação chamado “**Todos por 1**”. Este programa consistia no seguinte: 1 funcionário seria escolhido ao final do ano como o funcionário destaque, só que a bonificação seria recebida por todos da área daquele funcionário em particular. O objetivo desse programa seria o de incentivar a colaboração coletiva entre os funcionários de uma mesma área. Desta forma, o funcionário destaque beneficiaria não só a si, mas também a todos os colegas de sua área.

Ao final do ano, o funcionário escolhido como destaque foi o **Miguel Severino**. Isso significa que todos os funcionários da área do Miguel seriam beneficiados com o programa.

O seu objetivo é realizar uma consulta à tabela **DimEmployee** e retornar todos os funcionários da área “vencedora” para que o setor Financeiro possa realizar os pagamentos das bonificações.



```

SQLQuery1.sql - M...MARCUS\caval (53)*  X
SELECT *
FROM DimEmployee
WHERE DepartmentName = (
    SELECT
        DepartmentName
    FROM DimEmployee
    WHERE FirstName = 'Miguel'
    AND
    LastName = 'Severino'
)

```

The screenshot shows a SQL query in the SQL Server Management Studio (SSMS) interface. The query uses a Common Table Expression (CTE) to find all employees in the same department as Miguel Severino. The results are displayed in a grid below the query editor, showing 11 rows of employee data from the DimEmployee table.

	EmployeeKey	ParentEmployeeKey	FirstName	LastName	MiddleName	Title	HireDate	BirthDate	EmailAddress	Phone	MaritalStatus
1	1	18	Kim	Abercrombie	NULL	Sales Region Manager	1996-07-31	1972-05-15	guy1@contoso.com	320-555-0195	M
2	9	4	Terry	Adams	NULL	Sales State Manager	1998-01-26	1946-02-16	jolynn0@contoso.com	903-555-0145	S
3	10	5	David	Probst	NULL	Sales State Manager	1998-02-06	1946-07-06	ruth0@contoso.com	145-555-0130	M
4	12	2	Michael	Rahem	NULL	Sales State Manager	1998-02-07	1946-04-27	barry0@contoso.com	206-555-0180	S
5	15	9	Kim	Akers	NULL	Sales Store Manager	1998-03-05	1946-10-01	sidney0@contoso.com	424-555-0189	M
6	16	1	Kate	Taneyhill	NULL	Sales State Manager	1998-03-11	1946-05-03	taylor0@contoso.com	508-555-0165	M
7	17	2	David	Alexander	NULL	Sales State Manager	1998-03-23	1946-08-12	jeffrey0@contoso.com	984-555-0185	S
8	19	4	Michelle	Alexander	NULL	Sales State Manager	1998-04-11	1946-05-06	doris0@contoso.com	328-555-0150	M
9	20	5	Rene	Valdes	NULL	Sales State Manager	1998-04-18	1946-09-08	john0@contoso.com	435-555-0113	M
10	21	1	Michael	Allen	NULL	Sales State Manager	1998-04-29	1946-04-30	diane0@contoso.com	202-555-0151	M
11	22	2	Rohinton	Wadia	NULL	Sales State Manager	1999-01-02	1967-06-15	steven0@contoso.com	925-555-0114	M

Consulta executada com êxito. | MARCUS (15.0 RTM) | MARCUS\caval (53) | ContosoRetailDW | 00:00:00 | 176 linhas

# GABARITOS

4

## Questão 4

Faça uma query que retorne os clientes que recebem um salário anual acima da média. A sua query deve retornar as colunas CustomerKey, FirstName, LastName, EmailAddress e YearlyIncome.

Obs: considere apenas os clientes que são 'Pessoas Físicas'.

```

SQLQuery1.sql - M...MARCUS\caval (53)* - X
SELECT
    CustomerKey,
    FirstName,
    LastName,
    EmailAddress,
    YearlyIncome
FROM DimCustomer
WHERE YearlyIncome > (
    SELECT
        AVG(YearlyIncome)
    FROM DimCustomer
    WHERE CustomerType = 'Person'
)
AND
CustomerType = 'Person'
ORDER BY YearlyIncome DESC
  
```

The screenshot shows a SQL query window in SSMS. The query uses a Common Table Expression (CTE) to calculate the average yearly income for customers of type 'Person'. It then selects the top 9 customers from the DimCustomer table whose yearly income is greater than this average, ordered by yearly income in descending order. The results are displayed in a grid:

	CustomerKey	FirstName	LastName	EmailAddress	YearlyIncome
1	81	Damien	Chander	damien32@adventure-works.com	170000,00
2	182	Devin	Martin	devin13@adventure-works.com	170000,00
3	245	Alexis	Coleman	alexis28@adventure-works.com	170000,00
4	251	Shannon	Liu	shannon4@adventure-works.com	170000,00
5	287	Hunter	Griffin	hunter16@adventure-works.com	170000,00
6	423	Dustin	Deng	dustin1@adventure-works.com	170000,00
7	435	Andre	Lopez	andre16@adventure-works.com	170000,00
8	468	Arturo	Zheng	arturo21@adventure-works.com	170000,00
9	1124	Wesley	Liang	wesley16@adventure-works.com	170000,00

Consulta executada com êxito. | MARCUS (15.0 RTM) | MARCUS\caval (53) | ContosoRetailDW | 00:00:09 | 9.858 linhas



# GABARITOS

5

## Questão 5

A ação de desconto da Asian Holiday Promotion foi uma das mais bem sucedidas da empresa. Agora, a Contoso quer entender um pouco melhor sobre o perfil dos clientes que compraram produtos com essa promoção.

Seu trabalho é criar uma query que retorne a lista de clientes que compraram nessa promoção.

The screenshot shows a SQL query being run in SSMS. The query uses Common Table Expressions (CTEs) to find customers who purchased items under the 'Asian Holiday Promotion'. The results are displayed in a table with columns: CustomerKey, FirstName, LastName, and EmailAddress. The results show 9 rows of customer information.

```
SQLQuery1.sql - M...MARCUS\caval (53)* ✓ ×
SELECT
    CustomerKey,
    FirstName,
    LastName,
    EmailAddress
FROM DimCustomer
WHERE CustomerKey IN(
    SELECT
        CustomerKey
    FROM FactOnlineSales
    WHERE PromotionKey IN (
        SELECT
            PromotionKey
        FROM DimPromotion
        WHERE PromotionName = 'Asian Holiday Promotion'
    )
)
```

	CustomerKey	FirstName	LastName	EmailAddress
1	4789	Barbara	Li	barbara13@adventure-works.com
2	4800	Victoria	Anderson	victoria11@adventure-works.com
3	4807	Devin	Allen	devin22@adventure-works.com
4	4832	Omar	He	omar17@adventure-works.com
5	5032	Ian	Allen	ian23@adventure-works.com
6	5157	Javier	Hernandez	javier0@adventure-works.com
7	5168	Devin	Moore	devin6@adventure-works.com
8	5175	Melinda	Hernandez	melinda0@adventure-works.com
9	5182	Daisy	Rubio	daisy16@adventure-works.com

Consulta executada com êxito. | MARCUS (15.0 RTM) | MARCUS\caval (53) | ContosoRetailDW | 00:00:05 | 3.660 linhas

# GABARITOS

6

## Questão 6

A empresa implementou um programa de fidelização de clientes empresariais. Todos aqueles que comprarem mais de 3000 unidades de um mesmo produto receberá descontos em outras compras.

Você deverá descobrir as informações de CustomerKey e CompanyName destes clientes.

The screenshot shows a SQL query being run in SQL Server Management Studio. The query uses a Common Table Expression (CTE) to find customer keys that have purchased at least 3000 units of a single product, and then selects the customer key and company name for those customers.

```
SQLQuery1.sql - M...MARCUS\caval (53)*
SELECT
    CustomerKey,
    CompanyName
FROM DimCustomer
WHERE CustomerKey IN (
    SELECT
        CustomerKey
    FROM FactOnlineSales
    GROUP BY CustomerKey, ProductKey
    HAVING COUNT(*) >= 3000
)
```

The results window displays the following data:

	CustomerKey	CompanyName
1	19066	CologneCompany
2	19114	SeafordCompany
3	19140	TehranCompany
4	19142	TokyoCompany
5	19145	Yokohama Company

At the bottom of the results window, a message indicates: "Consulta executada com êxito." (Query executed successfully).

# GABARITOS

7

## Questão 7

Você deverá criar uma consulta para o setor de vendas que mostre as seguintes colunas da tabela DimProduct:

ProductKey, ProductName, BrandName, UnitPrice, Média de UnitPrice.

The screenshot shows a SQL query window titled "SQLQuery1.sql - M...MARCUS\caval (53)\*". The query is:

```
SELECT
    ProductKey,
    ProductName,
    BrandName,
    UnitPrice,
    (SELECT AVG(UnitPrice) FROM DimProduct) AS 'Média de
        UnitPrice'
FROM DimProduct
```

The results pane displays a table with 9 rows of data from the DimProduct table, including the calculated average unit price.

	ProductKey	ProductName	BrandName	UnitPrice	Média de UnitPrice
1	1	Contoso 512MB MP3 Player E51 Silver	Contoso	12,99	356,8301
2	2	Contoso 512MB MP3 Player E51 Blue	Contoso	12,99	356,8301
3	3	Contoso 1G MP3 Player E100 White	Contoso	14,52	356,8301
4	4	Contoso 2G MP3 Player E200 Silver	Contoso	21,57	356,8301
5	5	Contoso 2G MP3 Player E200 Red	Contoso	21,57	356,8301
6	6	Contoso 2G MP3 Player E200 Black	Contoso	21,57	356,8301
7	7	Contoso 2G MP3 Player E200 Blue	Contoso	21,57	356,8301
8	8	Contoso 4G MP3 Player E400 Silver	Contoso	59,99	356,8301
9	9	Contoso 4G MP3 Player E400 Black	Contoso	59,99	356,8301

Consulta executada com êxito.

# GABARITOS

8

## Questão 8

Faça uma consulta para descobrir os seguintes indicadores dos seus produtos:

- Maior quantidade de produtos por marca
- Menor quantidade de produtos por marca
- Média de produtos por marca

The screenshot shows a SQL query in the SQL Query window and its results in the Results window.

**SQL Query Window:**

```
SQLQuery1.sql - M...MARCUS\caval (53)*
SELECT
    MAX(Quantidade) AS 'Máximo',
    MIN(Quantidade) AS 'Mínimo',
    AVG(Quantidade) AS 'Média'
FROM(
    SELECT
        BrandName,
        COUNT(*) AS 'Quantidade'
    FROM DimProduct
    GROUP BY BrandName
) AS T
```

**Results Window:**

	Máximo	Mínimo	Média
1	710	47	228

Consulta executada com êxito. | MARCUS (15.0 RTM) | MARCUS\caval (53) | ContosoRetailDW | 00:00:00 | 1 linhas

# GABARITOS

9

## Questão 9

Crie uma CTE que seja o agrupamento da tabela **DimProduct**, armazenando o total de produtos por marca. Em seguida, faça um SELECT nesta CTE, descobrindo qual é a quantidade máxima de produtos para uma marca. Chame esta CTE de **CTE\_QtdProdutosPorMarca**.

The screenshot shows a SQL query window titled "SQLQuery1.sql - M...MARCUS\caval (53)\*". The query is:

```
WITH CTE_QtdProdutosPorMarca AS(
    SELECT
        BrandName,
        COUNT(*) AS 'Quantidade'
    FROM DimProduct
    GROUP BY BrandName
)
SELECT MAX(Quantidade) FROM CTE_QtdProdutosPorMarca
```

The results pane shows a single row with the value 710.

(Nenhum nome de coluna)
1 710

At the bottom, a green status bar indicates: "Consulta executada com êxito." and "MARCUS (15.0 RTM) | MARCUS\caval (53) | ContosoRetailDW | 00:00:00 | 1 linhas".

# GABARITOS

10

## Questão 10

Crie duas CTEs:

- (i) a primeira deve conter as colunas ProductKey, ProductName, ProductSubcategoryKey, BrandName e UnitPrice, da tabela DimProduct, mas apenas os produtos da marca Adventure Works. Chame essa CTE de CTE\_ProdutosAdventureWorks.
- (ii) a segunda deve conter as colunas ProductSubcategoryKey, ProductSubcategoryName, da tabela DimProductSubcategory mas apenas para as subcategorias 'Televisions' e 'Monitors'. Chame essa CTE de CTE\_CategoriaTelevisionsERadio.

Faça um Join entre essas duas CTEs, e o resultado deve ser uma query contendo todas as colunas das duas tabelas. Observe nesse exemplo a diferença entre o LEFT JOIN e o INNER JOIN.

```

SQLQuery1.sql - M...MARCUS\caval (53)* - X
WITH CTE_ProdutosAdventureWorks AS(
SELECT
    ProductKey,
    ProductName,
    ProductSubcategoryKey,
    BrandName,
    UnitPrice
FROM DimProduct
WHERE BrandName = 'Adventure Works'
),
CTE_CategoriaTelevisionsERadio AS(
SELECT
    ProductSubcategoryKey,
    ProductSubcategoryName
FROM DimProductSubcategory
WHERE ProductSubcategoryName IN ('Televisions', 'Monitors')
)

SELECT
    CTE_ProdutosAdventureWorks.*,
    CTE_CategoriaTelevisionsERadio.ProductSubcategoryName
FROM CTE_ProdutosAdventureWorks
LEFT JOIN CTE_CategoriaTelevisionsERadio
    ON CTE_ProdutosAdventureWorks.ProductSubcategoryKey = CTE_CategoriaTelevisionsERadio.ProductSubcategoryKey
  
```

**Resultados**

	ProductKey	ProductName	ProductSubcategoryKey	BrandName	Unit Price	ProductSubcategoryName
1	116	Adventure Works 20" CRT TV E15 Silver	9	Adventure Works	169,99	Televisions
2	117	Adventure Works 20" CRT TV E15 Black	9	Adventure Works	169,99	Televisions
3	118	Adventure Works 20" CRT TV E15 White	9	Adventure Works	169,99	Televisions
4	119	Adventure Works 13" Color TV E25 Silver	9	Adventure Works	119,99	Televisions
5	120	Adventure Works 13" Color TV E25 Black	9	Adventure Works	119,99	Televisions

Consulta executada com êxito.

MARCUS (15.0 RTM) | MARCUS\caval (53) | ContosoRetailDW | 00:00:00 | 192 linhas

# SQL LOOPS



# WHILE

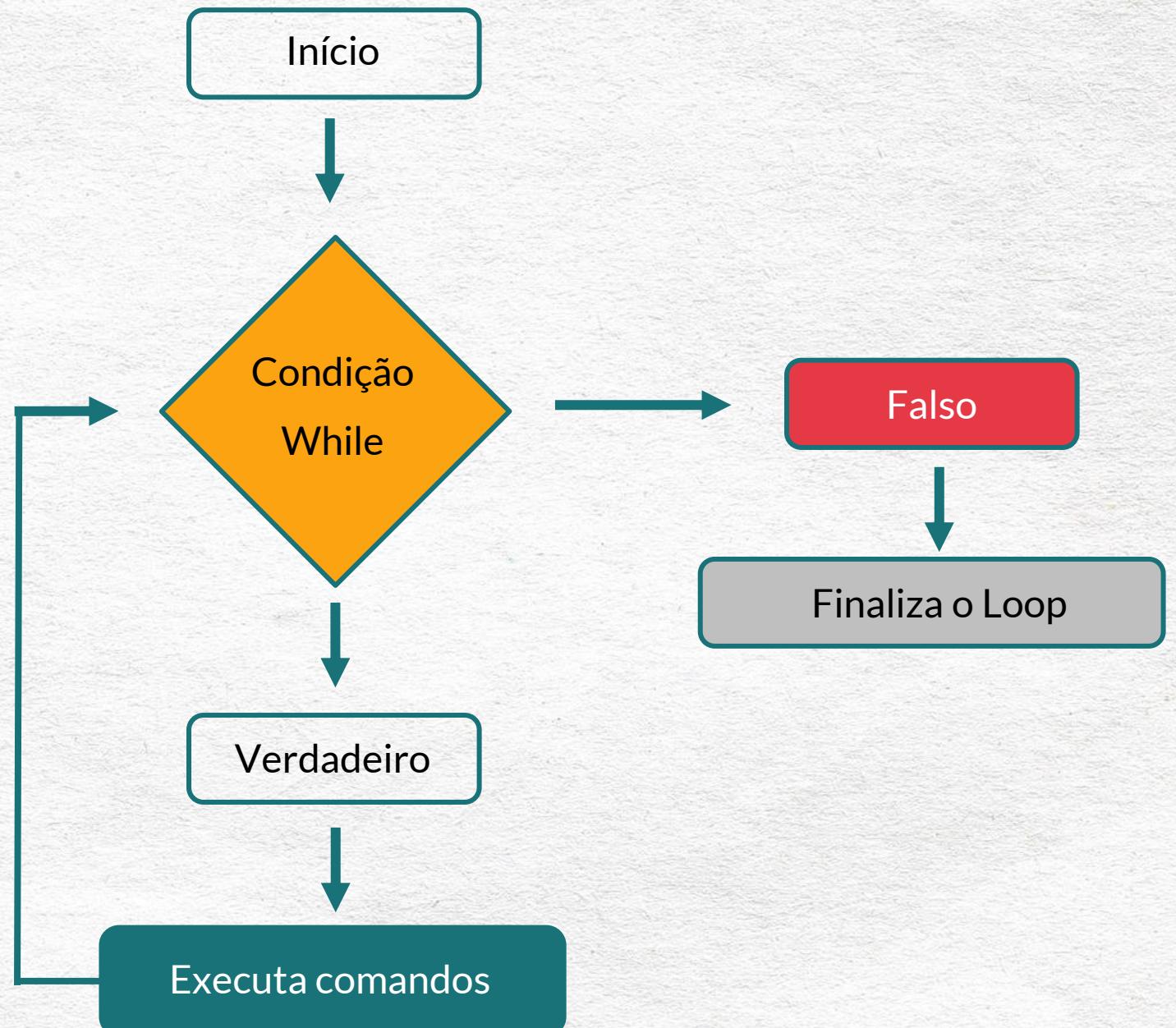
## O QUE É UM LOOP

Para que servem?

Um loop em uma linguagem de programação tem como objetivo executar repetidas vezes uma série de comandos.

O While é uma estrutura de repetição que tem exatamente esse objetivo. Com ele, conseguimos executar diversas vezes um ou mais comandos, enquanto uma determinada condição é satisfeita. E a partir do momento que essa condição deixa de ser satisfeita, o loop finaliza.

Ao lado, podemos ver um esquema visual da lógica While.

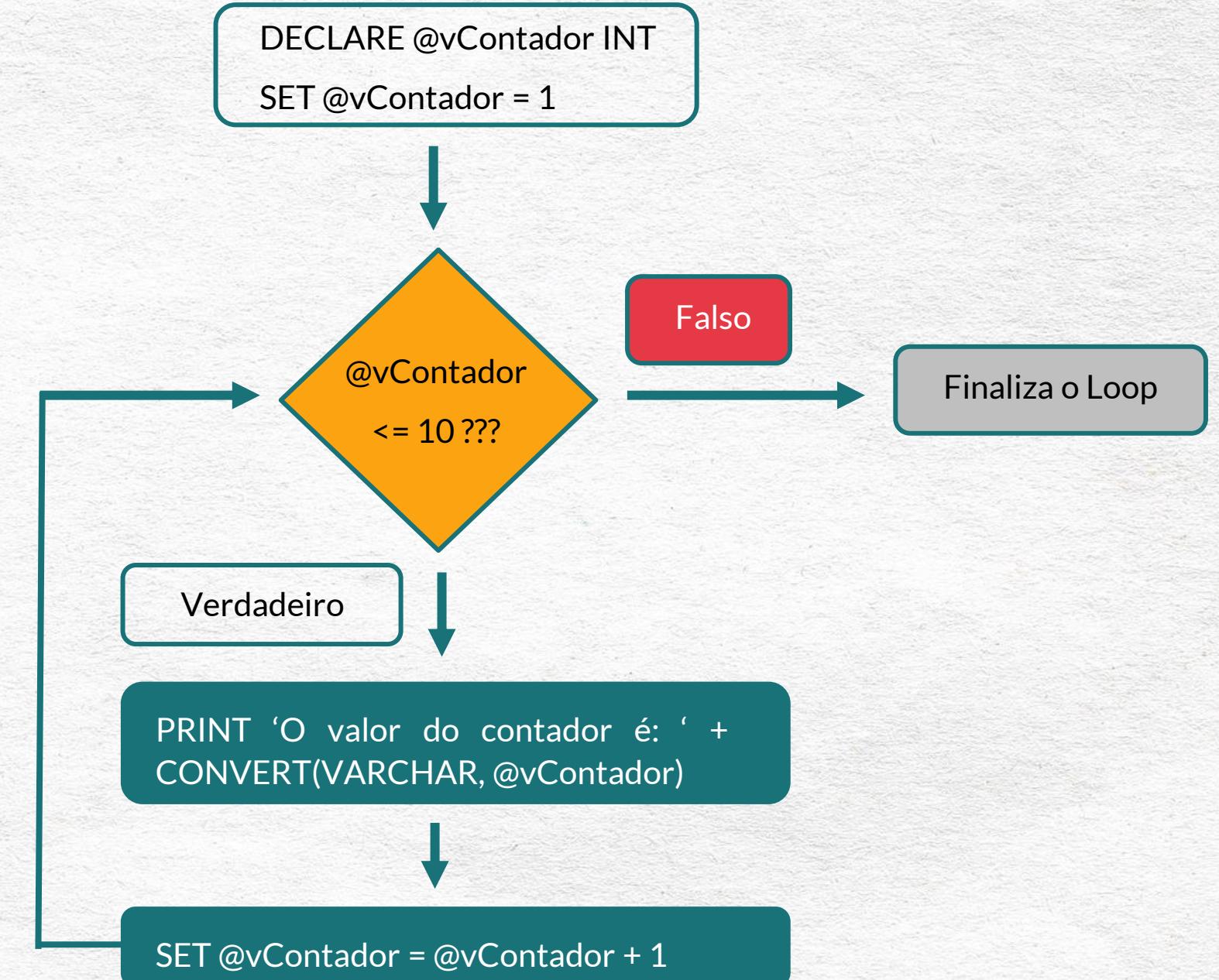


# WHILE

## EXEMPLO

Imagine que a gente queira criar um contador, que vai de 1 a 10, utilizando a lógica While.

Na imagem ao lado é possível visualizar a lógica.



# WHILE

## ESTRUTURA BÁSICA

A estrutura básica do WHILE é a seguinte:

**WHILE** condição

**BEGIN**

comandos

**END**

Ou seja, enquanto uma determinada condição é satisfeita, todos os comandos dentro do bloco BEGIN.. END serão executados.

# WHILE - Exemplo

Imagine que a gente queira criar um contador, de 1 até 10, que printasse na tela:

*“O valor do contador é: \_\_”*

Poderíamos criar uma estrutura em Loop para executar o comando repetidas vezes, da forma como é mostrada no exemplo ao lado.

Iniciamos uma variável chamada @vContador = 1, e a nossa condição WHILE é executar o PRINT enquanto o valor de @vContador for menor ou igual a 10.

Observe que criamos uma linha para incrementar esta variável cada vez que finalizamos um loop (SET @vContador = @vContador + 1).

Desta forma, em algum momento, a condição @vContador não será mais satisfeita e consequentemente o WHILE será finalizado.

```

Coluna.sql - LAPTOP...us Cavalcanti (59)*  SQLQuery1.sql - LAP...us Cavalcanti (51)
DECLARE @vContador INT
SET @vContador = 1

WHILE @vContador <= 10
BEGIN
    PRINT 'O valor do contador é: ' + CONVERT(VARCHAR, @vContador)
    SET @vContador = @vContador + 1
END
  
```

Mensagens

```

O valor do contador é: 1
O valor do contador é: 2
O valor do contador é: 3
O valor do contador é: 4
O valor do contador é: 5
O valor do contador é: 6
O valor do contador é: 7
O valor do contador é: 8
O valor do contador é: 9
O valor do contador é: 10
  
```

# BREAK - Exemplo

Podemos utilizar o comando BREAK para finalizar um loop assim que uma condição intermediária for satisfeita.

Observe no exemplo ao lado que desejamos que o loop WHILE seja executado enquanto o momento em que @vContador for menor ou igual a 100.

Porém, criamos um teste IF para, a cada loop, verificar se o valor de @vContador é igual a 15. Caso seja, o loop é automaticamente interrompido e finalizado.

No exemplo ao lado, observe que o contador vai apenas até 15.

The screenshot shows a dual-pane interface of SQL Server Management Studio. The left pane displays a script named 'Coluna.sql' containing a WHILE loop. The right pane shows the results of the execution in the 'Mensagens' (Messages) tab. The script is as follows:

```
DECLARE @vContador INT  
SET @vContador = 1  
  
WHILE @vContador <= 100  
BEGIN  
    PRINT 'O valor do contador é: ' + CONVERT(VARCHAR, @vContador)  
    IF @vContador = 15  
        BREAK  
    SET @vContador = @vContador + 1  
END
```

The output in the 'Mensagens' tab shows the values from 1 to 15, each preceded by the message 'O valor do contador é:':

```
O valor do contador é: 1  
O valor do contador é: 2  
O valor do contador é: 3  
O valor do contador é: 4  
O valor do contador é: 5  
O valor do contador é: 6  
O valor do contador é: 7  
O valor do contador é: 8  
O valor do contador é: 9  
O valor do contador é: 10  
O valor do contador é: 11  
O valor do contador é: 12  
O valor do contador é: 13  
O valor do contador é: 14  
O valor do contador é: 15
```

# CONTINUE - Exemplo

O comando CONTINUE pode ser usado para pular repetições em um loop. Ou seja, cada vez que uma condição mais interna ao loop é atendida, o loop não executado, mas continua acontecendo até que a condição do WHILE seja atendida.

No exemplo ao lado, temos um contador que deve contar até 10 (WHILE @varContador <= 10). Porém, no meio do código, testamos dois números em particular: 3 e 6. Ou seja, sempre que @varContador for igual a 3 ou @varContador for igual a 6, o comando PRINT é pulado, porém a contagem se mantém funcionando até que @varContador seja igual a 10.

Observe que na contagem ao lado, na janela de mensagens, os números 3 e 6 foram pulados.

The screenshot shows a SQL query window in SSMS. The code is as follows:

```
DECLARE @varContador INT  
SET @varContador = 0  
  
WHILE @varContador <= 10  
BEGIN  
    SET @varContador += 1  
    IF @varContador = 3 OR @varContador = 6  
        CONTINUE  
    PRINT @varContador  
END
```

The status bar indicates the execution is at 22%. Below the code, a 'Mensagens' (Messages) window is open, showing the output of the script:

```
1  
2  
4  
5  
7  
8  
9  
10  
11
```

At the bottom of the window, the timestamp is shown as "Horário de conclusão: 2022-01-06T16:43:53.2218654-03:00".

# EXERCÍCIOS



1

2

## Questão 1

Utilize o Loop While para criar um contador que comece em um valor inicial @ValorInicial e termine em um valor final @ValorFinal. Você deverá printar na tela a seguinte frase:

"O valor do contador é: " + \_\_

## Questão 2

Você deverá criar uma estrutura de repetição que printe na tela a quantidade de contratações para cada ano, desde 1996 até 2003. A informação de data de contratação encontra-se na coluna HireDate da tabela DimEmployee. Utilize o formato:

X contratações em 1996

Y contratações em 1997

Z contratações em 1998

...

...

N contratações em 2003

Obs: a coluna HireDate contém a data completa (dd/mm/aaaa). Lembrando que você deverá printar a quantidade de contratações por ano.

# EXERCÍCIOS

3

## Questão 3

Utilize um Loop While para criar uma tabela chamada Calendario, contendo uma coluna que comece com a data 01/01/2021 e vá até 31/12/2021.



# GABARITOS

1

## Questão 1

Utilize o Loop While para criar um contador que comece em um valor inicial @ValorInicial e termine em um valor final @ValorFinal. Você deverá printar na tela a seguinte frase:

“O valor do contador é: “ + \_\_

The screenshot shows a SQL query window titled "SQLQuery1.sql - M...MARCUS\caval (53)\*". The query is as follows:

```
DECLARE @ValorInicial INT  
DECLARE @ValorFinal INT  
  
SET @ValorInicial=1  
SET @ValorFinal=10  
  
WHILE (@ValorInicial <= @ValorFinal)  
BEGIN  
    PRINT 'O valor do contador é: ' + CONVERT(VARCHAR, @ValorInicial)  
    SET @ValorInicial = @ValorInicial + 1  
END
```

The "Mensagens" (Messages) pane below the query window displays the output of the query, which is a sequence of numbers from 1 to 10, each preceded by the message "O valor do contador é: ". The output ends with the timestamp "Horário de conclusão: 2022-04-18T10:41:16.1120478-03:00". At the bottom of the window, a status bar indicates "Consulta executada com êxito." (Query executed successfully).



# GABARITOS

2

## Questão 2

Você deverá criar uma estrutura de repetição que printe na tela a quantidade de contratações para cada ano, desde 1996 até 2003. A informação de data de contratação encontra-se na coluna **HireDate** da tabela **DimEmployee**. Utilize o formato:

*X contratações em 1996*

*Y contratações em 1997*

*Z contratações em 1998*

...

...

*N contratações em 2003*

Obs: a coluna **HireDate** contém a data completa (dd/mm/aaaa).

Lembrando que você deverá printar a quantidade de contratações **por ano**.

The screenshot shows a SQL query in the SQL Query window and its execution results in the Messages window.

```

SQLQuery1.sql - M...MARCUS\caval (53)*  +
DECLARE @AnoInicial INT = 1996
DECLARE @AnoFinal INT = 2003

WHILE @AnoInicial <= @AnoFinal
BEGIN
    DECLARE @QtdFuncionarios INT =(SELECT COUNT(*) FROM DimEmployee WHERE YEAR(HireDate) = @AnoInicial)
    PRINT CAST(@QtdFuncionarios as varchar(4)) + ' contratações em ' + cast(@AnoInicial as varchar(4))
    SET @AnoInicial += 1
END
  
```

Mensagens

Quantidade de contratações	Ano
1	1996
2	1997
18	1998
198	1999
44	2000
23	2001
4	2002
3	2003

Horário de conclusão: 2022-04-18T10:42:05.8585776-03:00

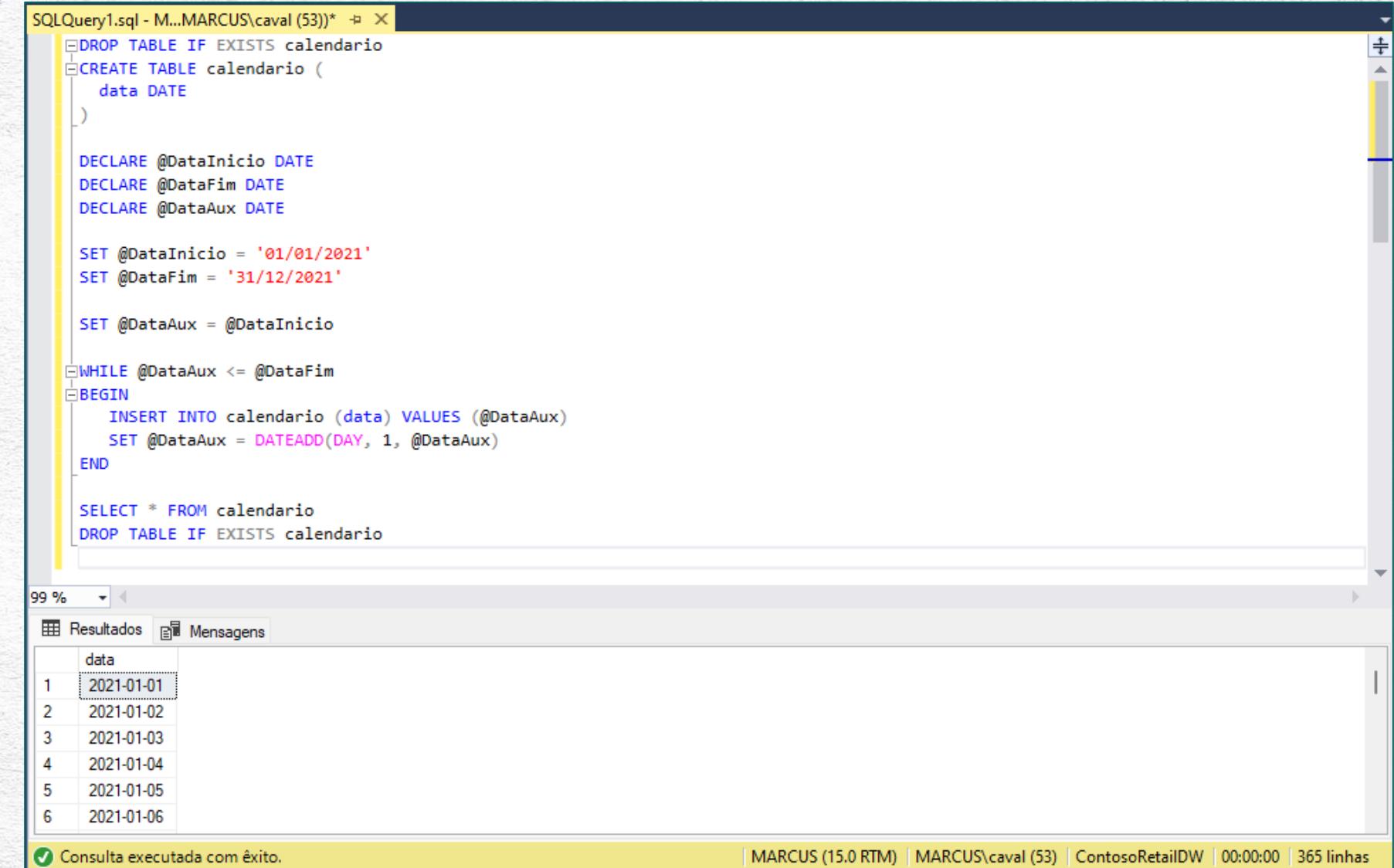
Consulta executada com êxito.

# GABARITOS

3

## Questão 3

Utilize um Loop While para criar uma tabela chamada Calendario, contendo uma coluna que comece com a data 01/01/2021 e vá até 31/12/2021.



The screenshot shows a SQL query window titled "SQLQuery1.sql - M...MARCUS\caval (53)\*". The query uses a WHILE loop to generate a calendar table named "calendario". The loop starts at "01/01/2021" and ends at "31/12/2021", incrementing by one day each iteration. The results are displayed in a table titled "Resultados" with columns "data" and "id". The data shows the first six days of January 2021. A status bar at the bottom indicates the query was executed successfully.

```
SQLQuery1.sql - M...MARCUS\caval (53)*
DROP TABLE IF EXISTS calendario
CREATE TABLE calendario (
    data DATE
)

DECLARE @DataInicio DATE
DECLARE @DataFim DATE
DECLARE @DataAux DATE

SET @DataInicio = '01/01/2021'
SET @DataFim = '31/12/2021'

SET @DataAux = @DataInicio

WHILE @DataAux <= @DataFim
BEGIN
    INSERT INTO calendario (data) VALUES (@DataAux)
    SET @DataAux = DATEADD(DAY, 1, @DataAux)
END

SELECT * FROM calendario
DROP TABLE IF EXISTS calendario
```

	data
1	2021-01-01
2	2021-01-02
3	2021-01-03
4	2021-01-04
5	2021-01-05
6	2021-01-06

Consulta executada com êxito. | MARCUS (15.0 RTM) | MARCUS\caval (53) | ContosoRetailDW | 00:00:00 | 365 linhas

# SQL WINDOW FUNCTIONS



## PARA QUE SERVEM?

- São utilizadas para cálculos mais avançados de análises de dados;
- Possuem um uso semelhante ao GROUP BY, só que mais avançado e personalizável;
- A instrução OVER permite definir qual será a “janela” (conjunto de linhas) a ser considerado no cálculo;
- A instrução PARTITION BY divide o conjunto em “partições” nas quais as funções de janela são aplicadas.

# Window Functions - Introdução

## QUAIS AS FINALIDADES?

As funções de janela possuem as seguintes finalidades:

1. Cálculos de agregação: COUNT, SUM, AVG, MIN, MAX
2. Cálculos de deslocamento: FIRST\_VALUE, LAST\_VALUE, LEAD, LAG
3. Cálculos estatísticos: RANK, DENSE\_RANK, NTILE

# Window Functions - Introdução

Imagine a tabela abaixo, com a rede de 10 lojas de uma determinada empresa.

ID_Loja	Nome_Loja	Regiao	Qtd_Vendida
1	Botafogo Praia&Mar	Sudeste	1500
2	Lojas Vitoria	Sudeste	800
3	Emporio Mineirinho	Sudeste	2300
4	Central Paulista	Sudeste	1800
5	Casa Flor & Anópolis	Sul	2100
6	Pampas & Co	Sul	990
7	Paraná Papéis	Sul	2800
8	Amazonas Prime	Norte	4200
9	Pará Bens	Norte	3200
10	Tintas Rio Branco	Norte	2400

```
SELECT SUM(Qtd_Vendida) AS 'Total Vendido' FROM Lojas
```

Total Vendido  
22090

Como poderíamos fazer para calcular o % vendido por cada loja?

# Window Functions - Introdução

Imagine a tabela abaixo, com a rede de 10 lojas de uma determinada empresa. A imagem 1 mostra as 10 lojas e suas respectivas vendas. Já a figura 2 mostra o comando em SQL para calcular a soma total da quantidade vendida de todas essas lojas.

1

ID_Loja	Nome_Loja	Regiao	Qtd_Vendida
1	Botafogo Praia&Mar	Sudeste	1500
2	Lojas Vitoria	Sudeste	800
3	Emporio Mineirinho	Sudeste	2300
4	Central Paulista	Sudeste	1800
5	Casa Flor & Anópolis	Sul	2100
6	Pampas & Co	Sul	990
7	Paraná Papéis	Sul	2800
8	Amazonas Prime	Norte	4200
9	Pará Bens	Norte	3200
10	Tintas Rio Branco	Norte	2400

2

```
SELECT SUM(Qtd_Vendida) AS 'Total Vendido' FROM Lojas
```

Total Vendido  
22090

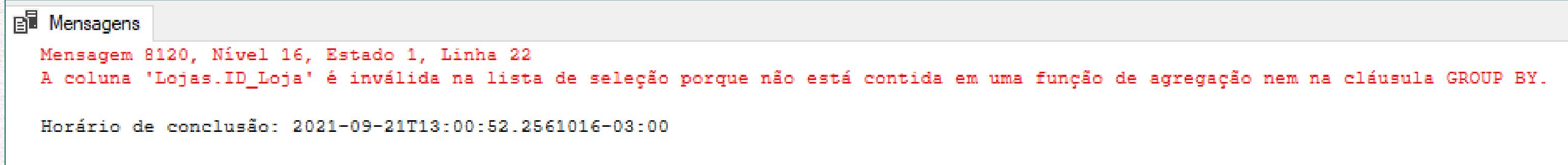
Como poderíamos fazer para calcular o % vendido por cada loja?

# Window Functions - Introdução

Primeiro, teríamos que calcular o total vendido e incluir na nossa tabela.

Porém, ao executar o código ao lado, incluindo uma nova coluna a ser calculada pela função SUM, obtemos o erro abaixo:

```
SELECT
    ID_Loja,
    Nome_Loja,
    Regiao,
    Qtd_Vendida,
    SUM(Qtd_Vendida) AS 'Total Vendido'
FROM Lojas
```



O erro ocorreu porque quando incluímos o SUM dentro do SELECT, ele espera que a gente faça algum agrupamento com o GROUP BY.

# Window Functions - Introdução

Uma solução que poderíamos pensar seria utilizar o GROUP BY, agrupando por todas as colunas da tabela, dado que queremos visualizar todas essas colunas.

Porém, ao fazer isso, o Total Vendido é calculado de forma diferente para cada linha, um resultado que não queremos. O que queremos na verdade é poder ter um valor total para todas as linhas, no caso 22090, para poder dividir os valores de Qtd\_Vendida por este total.

Portanto, para resolver este problema, temos que usar funções de janela e a instrução OVER.

```

SELECT
    ID_Loja,
    Nome_Loja,
    Regiao,
    Qtd_Vendida,
    SUM(Qtd_Vendida) AS 'Total Vendido'
FROM Lojas
GROUP BY ID_Loja, Nome_Loja, Regiao, Qtd_Vendida
  
```

	ID_Loja	Nome_Loja	Regiao	Qtd_Vendida	Total Vendido
1	1	Botafogo Praia&Mar	Sudeste	1500	1500
2	2	Lojas Vitoria	Sudeste	800	800
3	3	Emporio Mineirinho	Sudeste	2300	2300
4	4	Central Paulista	Sudeste	1800	1800
5	5	Casa Flor & Anópolis	Sul	2100	2100
6	6	Pampas & Co	Sul	990	990
7	7	Paraná Papéis	Sul	2800	2800
8	8	Amazonas Prime	Norte	4200	4200
9	9	Pará Bens	Norte	3200	3200
10	10	Tintas Rio Branco	Norte	2400	2400

# Window Functions - Introdução

Simplesmente adicionamos o OVER() logo depois do SUM e o Total Vendido será calculado igual para todas as linhas, permitindo que a gente faça o cálculo do %.

Lembrando que a instrução OVER permite definir qual será a “janela” (conjunto de linhas) a ser considerado no cálculo.

```

SELECT
    ID_Loja,
    Nome_Loja,
    Regiao,
    Qtd_Vendida,
    SUM(Qtd_Vendida) OVER() AS 'Total Vendido'
FROM Lojas

```

ID_Loja	Nome_Loja	Regiao	Qtd_Vendida	Total Vendido
1	Botafogo Praia&Mar	Sudeste	1500	22090
2	Lojas Vitoria	Sudeste	800	22090
3	Emporio Mineirinho	Sudeste	2300	22090
4	Central Paulista	Sudeste	1800	22090
5	Casa Flor & Anópolis	Sul	2100	22090
6	Pampas & Co	Sul	990	22090
7	Paraná Papéis	Sul	2800	22090
8	Amazonas Prime	Norte	4200	22090
9	Pará Bens	Norte	3200	22090
10	Tintas Rio Branco	Norte	2400	22090

# Window Functions - Introdução

E se a gente usasse o PARTITION BY junto com o OVER?

Lembrando que a instrução PARTITION BY divide o conjunto em “partições” nas quais as funções de janela são aplicadas.

Se a gente informar o ID\_Loja junto com o PARTITION BY, a janela vai considerar os agrupamentos de ID\_Loja. Como cada ID é um ID diferente, o resultado do Total Vendido não faz nenhum agrupamento, retornando o resultado diferente para cada loja.

The screenshot shows a SQL query in the top pane and its results in the bottom pane. The query is:

```
SELECT
    ID_Loja,
    Nome_Loja,
    Regiao,
    Qtd_Vendida,
    SUM(Qtd_Vendida) OVER(PARTITION BY ID_Loja) AS 'Total Vendido'
FROM Lojas
```

The results table has columns: ID\_Loja, Nome\_Loja, Regiao, Qtd\_Vendida, and Total Vendido. The data is as follows:

ID_Loja	Nome_Loja	Regiao	Qtd_Vendida	Total Vendido
1	Botafogo Praia&Mar	Sudeste	1500	1500
2	Lojas Vitoria	Sudeste	800	800
3	Emporio Mineirinho	Sudeste	2300	2300
4	Central Paulista	Sudeste	1800	1800
5	Casa Flor & Anópolis	Sul	2100	2100
6	Pampas & Co	Sul	990	990
7	Paraná Papéis	Sul	2800	2800
8	Amazonas Prime	Norte	4200	4200
9	Pará Bens	Norte	3200	3200
10	Tintas Rio Branco	Norte	2400	2400

# Window Functions - Introdução

Agora se a gente informar a Regiao junto com o PARTITION BY, a janela vai considerar os agrupamentos de Regiao.

Como as regiões podem se repetir para diferentes lojas, o cálculo da soma vai levar isso em consideração.

Ou seja, na linha de uma loja do Sudeste, o Total Vendido será calculado de acordo com aquela região.

```

SELECT
    ID_Loja,
    Nome_Loja,
    Regiao,
    Qtd_Vendida,
    SUM(Qtd_Vendida) OVER(PARTITION BY Regiao) AS 'Total Vendido'
FROM Lojas
ORDER BY ID_Loja

```

ID_Loja	Nome_Loja	Regiao	Qtd_Vendida	Total Vendido
1	Botafogo Praia&Mar	Sudeste	1500	6400
2	Lojas Vitoria	Sudeste	800	6400
3	Emporio Mineirinho	Sudeste	2300	6400
4	Central Paulista	Sudeste	1800	6400
5	Casa Flor & Anópolis	Sul	2100	5890
6	Pampas & Co	Sul	990	5890
7	Paraná Papéis	Sul	2800	5890
8	Amazonas Prime	Norte	4200	9800
9	Pará Bens	Norte	3200	9800
10	Tintas Rio Branco	Norte	2400	9800

# Código para criar a tabela lojas

Para os exemplos e explicações das funções de janela, será necessário que você crie a tabela ao lado.

Portanto, copie o código e replique no seu SQL Server para que assim você possa acompanhar as próximas aulas.

```
CREATE TABLE Lojas(
    ID_Loja INT,
    Nome_Loja VARCHAR(100),
    Regiao VARCHAR(100),
    Qtd_Vendida FLOAT)

INSERT INTO Lojas(ID_Loja, Nome_Loja, Regiao, Qtd_Vendida)
VALUES
    (1, 'Botafogo Praia&Mar', 'Sudeste', 1800),
    (2, 'Lojas Vitoria', 'Sudeste', 800),
    (3, 'Emporio Mineirinho', 'Sudeste', 2300),
    (4, 'Central Paulista', 'Sudeste', 1800),
    (5, 'Rio 90 graus', 'Sudeste', 700),
    (6, 'Casa Flor & Anópolis', 'Sul', 2100),
    (7, 'Pampas & Co', 'Sul', 990),
    (8, 'Paraná Papéis', 'Sul', 2800),
    (9, 'Amazonas Prime', 'Norte', 4200),
    (10, 'Pará Bens', 'Norte', 3200),
    (11, 'Tintas Rio Branco', 'Norte', 1500),
    (12, 'Nordestemido Hall', 'Nordeste', 1910),
    (13, 'Cachoeirinha Loft', 'Nordeste', 2380)
```

# Funções de Agregação - SUM, COUNT, AVG, MIN, MAX

## OVER

Os primeiros exemplos que vamos ver é com as funções de agregação. Conseguimos usar essas funções para cálculos de totais de contagens, somas, médias, mínimos e máximos.

Na imagem ao lado é possível visualizar o resultado de cada uma dessas funções de agregação aplicadas às funções de janela.

Um detalhe importante é que neste exemplo, utilizamos a instrução **OVER** apenas. Dessa forma, o cálculo do total será considerando todas as lojas. Se quiséssemos calcular totais levando em consideração cada região (por exemplo, o pra uma loja do Sudeste, calcular o total do sudeste) teríamos que utilizar a instrução **OVER** em conjunto com a **PARTITION BY**.

```

SELECT
    ID_Loja,
    Nome_Loja,
    Regiao,
    Qtd_Vendida,
    COUNT(*) OVER() AS 'Qtd. Lojas',
    SUM(Qtd_Vendida) OVER() AS 'Total Vendas',
    AVG(Qtd_Vendida) OVER() AS 'Média Vendas',
    MIN(Qtd_Vendida) OVER() AS 'Mínimo Vendas',
    MAX(Qtd_Vendida) OVER() AS 'Máximo Vendas'
FROM Lojas
ORDER BY ID_Loja

```

	ID_Loja	Nome_Loja	Regiao	Qtd_Vendida	Qtd. Lojas	Total Vendas	Média Vendas	Mínimo Vendas	Máximo Vendas
1	1	Botafogo Praia&Mar	Sudeste	1800	13	26480	2036,92307692308	700	4200
2	2	Lojas Vitoria	Sudeste	800	13	26480	2036,92307692308	700	4200
3	3	Emporio Mineirinho	Sudeste	2300	13	26480	2036,92307692308	700	4200
4	4	Central Paulista	Sudeste	1800	13	26480	2036,92307692308	700	4200
5	5	Rio 90 graus	Sudeste	700	13	26480	2036,92307692308	700	4200
6	6	Casa Flor & Anópolis	Sul	2100	13	26480	2036,92307692308	700	4200
7	7	Pampas & Co	Sul	990	13	26480	2036,92307692308	700	4200
8	8	Paraná Papéis	Sul	2800	13	26480	2036,92307692308	700	4200
9	9	Amazonas Prime	Norte	4200	13	26480	2036,92307692308	700	4200
10	10	Pará Bens	Norte	3200	13	26480	2036,92307692308	700	4200
11	11	Tintas Rio Branco	Norte	1500	13	26480	2036,92307692308	700	4200
12	12	Nordestemido Hall	Nordeste...	1910	13	26480	2036,92307692308	700	4200
13	13	Cachoeirinha Loft	Nordeste...	2380	13	26480	2036,92307692308	700	4200

# Funções de Agregação - SUM, COUNT, AVG, MIN, MAX

## OVER + PARTITION BY

Se quisermos calcular estas agregações para fazer análises considerando agrupamentos específicos (por região, por exemplo) apenas adicionamos dentro do OVER a instrução PARTITION BY e informamos por qual coluna queremos fazer um agrupamento. Teria uma lógica semelhante ao GROUP BY, sendo que não agrupamos uma tabela, e sim o cálculo.

Ou seja, os cálculos de contagem, soma, média, mínimo e máximo são feitos considerando um determinado grupo especificado. No caso do exemplo, o grupo é a coluna Regiao.

```

SELECT
    ID_Loja,
    Nome_Loja,
    Regiao,
    Qtd_Vendida,
    COUNT(*) OVER(PARTITION BY Regiao) AS 'Qtd Lojas P/ Região',
    SUM(Qtd_Vendida) OVER(PARTITION BY Regiao) AS 'Total Vendas P/ Região',
    AVG(Qtd_Vendida) OVER(PARTITION BY Regiao) AS 'Média Vendas P/ Região',
    MIN(Qtd_Vendida) OVER(PARTITION BY Regiao) AS 'Mínimo Vendas P/ Região',
    MAX(Qtd_Vendida) OVER(PARTITION BY Regiao) AS 'Máximo Vendas P/ Região'
FROM Lojas
ORDER BY ID_Loja
  
```

Resultados

ID_Loja	Nome_Loja	Regiao	Qtd_Vendida	Qtd Lojas P/ Região	Total Vendas P/ Região	Média Vendas P/ Região	Mínimo Vendas P/ Região	Máximo Vendas P/ Região
1	Botafogo Praia&Mar	Sudeste	1800	5	7400	1480	700	2300
2	Lojas Vitoria	Sudeste	800	5	7400	1480	700	2300
3	Emporio Mineirinho	Sudeste	2300	5	7400	1480	700	2300
4	Central Paulista	Sudeste	1800	5	7400	1480	700	2300
5	Rio 90 graus	Sudeste	700	5	7400	1480	700	2300
6	Casa Flor & Anópolis	Sul	2100	3	5890	1963,333333333333	990	2800
7	Pampas & Co	Sul	990	3	5890	1963,333333333333	990	2800
8	Paraná Papéis	Sul	2800	3	5890	1963,333333333333	990	2800
9	Amazonas Prime	Norte	4200	3	8900	2966,666666666667	1500	4200
10	Pará Bens	Norte	3200	3	8900	2966,666666666667	1500	4200
11	Tintas Rio Branco	Norte	1500	3	8900	2966,666666666667	1500	4200
12	Nordestemido Hall	Nordeste	1910	2	4290	2145	1910	2380
13	Cachoerinha Loft	Nordeste	2380	2	4290	2145	1910	2380

# Funções de Agregação - SUM, COUNT, AVG, MIN, MAX

## OVER + PARTITION BY

Olhando mais atentamente para o exemplo anterior, vemos que a coluna 'Qtd Lojas P/ Região' nada mais é do que o total por região.

Para a linha 1, da loja Botafogo Praia&Mar, por exemplo, o valor 5 significa que há 5 lojas da região sudeste.

ID_Loja	Nome_Loja	Regiao	Qtd_Vendida	Qtd Lojas P/ Região
1	Botafogo Praia&Mar	Sudeste	1800	5
2	Lojas Vitoria	Sudeste	800	5
3	Emporio Mineirinho	Sudeste	2300	5
4	Central Paulista	Sudeste	1800	5
5	Rio 90 graus	Sudeste	700	5
6	Casa Flor & Anópolis	Sul	2100	3
7	Pampas & Co	Sul	990	3
8	Paraná Papéis	Sul	2800	3
9	Amazonas Prime	Norte	4200	3
10	Pará Bens	Norte	3200	3
11	Tintas Rio Branco	Norte	1500	3
12	Nordestemido Hall	Nordeste	1910	2
13	Cachoeirinha Loft	Nordeste	2380	2

# Cálculo de participação % (Parte 1)

Sabendo como fazer agregamentos totais (OVER apenas) e agregamentos particionados (OVER + PARTITION BY), poderemos facilmente agora fazer cálculos de participação %.

Observe no exemplo ao lado. Se quisermos descobrir qual é a participação de cada loja em termos de total vendido, basta dividirmos a coluna Qtd\_Vendida pelo bloco SUM(Qtd\_Vendida) OVER().

Obs: utilizamos a função FORMAT apenas para formatar os resultados como percentual. Não haveria necessidade de incluí-lo.

```

SELECT
    ID_Loja,
    Nome_Loja,
    Regiao,
    Qtd_Vendida,
    SUM(Qtd_Vendida) OVER() AS 'Total Vendido',
    FORMAT(Qtd_Vendida/SUM(Qtd_Vendida) OVER(), '0.00') AS '% Por Região'
FROM Lojas
ORDER BY ID_Loja
  
```

The screenshot shows a SQL query editor with the following interface:

- Top Panel:** Displays the SQL query code.
- Bottom Panel:** Shows the execution results in a grid format.

**Results Grid Headers:**

ID_Loja	Nome_Loja	Regiao	Qtd_Vendida	Total Vendido	% Por Região
---------	-----------	--------	-------------	---------------	--------------

**Results Grid Data:**

1	Botafogo Praia&Mar	Sudeste	1800	26480	6,80%
2	Lojas Vitoria	Sudeste	800	26480	3,02%
3	Emporio Mineirinho	Sudeste	2300	26480	8,69%
4	Central Paulista	Sudeste	1800	26480	6,80%
5	Rio 90 graus	Sudeste	700	26480	2,64%
6	Casa Flor & Anápolis	Sul	2100	26480	7,93%
7	Pampas & Co	Sul	990	26480	3,74%
8	Paraná Papéis	Sul	2800	26480	10,57%
9	Amazonas Prime	Norte	4200	26480	15,86%
10	Pará Bens	Norte	3200	26480	12,08%
11	Tintas Rio Branco	Norte	1500	26480	5,66%
12	Nordestemido Hall	Nordeste	1910	26480	7,21%
13	Cachoeirinha Loft	Nordeste	2380	26480	8,99%

# Cálculo de participação % (Parte 2)

Fazendo uma alteração simples e incluindo o PARTITION BY, podemos calcular o % que cada loja tem de participação, considerando suas respectivas regiões.

Tome cuidado porque no exemplo ao lado, a soma de % não é 100% porque o cálculo está sendo considerando cada região.

Ou seja, a loja Botafogo Praia&Mar tem uma quantidade vendida que representa 6,8% do valor total, representa 24,32% das vendas da região Sudeste.

```

SELECT
    ID_Loja,
    Nome_Loja,
    Regiao,
    Qtd_Vendida,
    SUM(Qtd_Vendida) OVER(PARTITION BY Regiao) AS 'Total Vendido',
    FORMAT(Qtd_Vendida/SUM(Qtd_Vendida) OVER(PARTITION BY Regiao), '0.00%') AS '% Por Região'
FROM Lojas
ORDER BY ID_Loja

```

ID_Loja	Nome_Loja	Regiao	Qtd_Vendida	Total Vendido	% Por Região
1	Botafogo Praia&Mar	Sudeste	1800	7400	24,32%
2	Lojas Vitoria	Sudeste	800	7400	10,81%
3	Emporio Mineirinho	Sudeste	2300	7400	31,08%
4	Central Paulista	Sudeste	1800	7400	24,32%
5	Rio 90 graus	Sudeste	700	7400	9,46%
6	Casa Flor & Anópolis	Sul	2100	5890	35,65%
7	Pampas & Co	Sul	990	5890	16,81%
8	Paraná Papéis	Sul	2800	5890	47,54%
9	Amazonas Prime	Norte	4200	8900	47,19%
10	Pará Bens	Norte	3200	8900	35,96%
11	Tintas Rio Branco	Norte	1500	8900	16,85%
12	Nordestemido Hall	Nordeste	1910	4290	44,52%
13	Cachoeirinha Loft	Nordeste	2380	4290	55,48%

# Funções de Classificação



## ROW\_NUMBER

Cria uma coluna com a numeração das linhas da tabela. É como se fosse uma coluna de ID.



## RANK

Cria uma coluna de ranking. Em caso de empates, os rankings dos números se repetem, mas o próximo ranking pula a sequência. Ex: dois valores na posição 8, o valor seguinte pula para a posição 10.



## DENSE\_RANK

Cria uma coluna de ranking. Em caso de empates, os rankings dos números se repetem, e o próximo ranking continua a sequência. Ex: dois valores na posição 8, o valor seguinte continua na posição 9.



## NTILE

Distribui os valores de uma coluna em grupos.

# Funções de Classificação – ROW\_NUMBER, RANK, DENSE\_RANK e NTILE

Para fins de entendimento e comparação, todas as funções de classificação são mostradas ao lado.

Observe que dentro da instrução OVER é utilizado o ORDER BY. O que faz sentido, pois de certa forma as funções de classificação têm como objetivo ordenar os valores.

Porém, com as funções de classificação, podemos potencializar a aplicação do ORDER BY (que aqui também pode ser usado com as opções ASC e DESC, fazendo a ordenação crescente e decrescente, respectivamente).

```

SELECT
    *,
    ROW_NUMBER() OVER(ORDER BY Qtd_Vendida DESC) AS 'rownumber',
    RANK() OVER(ORDER BY Qtd_Vendida DESC) AS 'rank',
    DENSE_RANK() OVER(ORDER BY Qtd_Vendida DESC) AS 'dense',
    NTILE(3) OVER(ORDER BY Qtd_Vendida DESC) AS 'ntile'
FROM
    Lojas
  
```

ID_Loja	Nome_Loja	Regiao	Qtd_Vendida	rownumber	rank	dense	ntile
9	Amazonas Prime	Norte	4200	1	1	1	1
10	Pará Bens	Norte	3200	2	2	2	1
8	Paraná Papéis	Sul	2800	3	3	3	1
13	Cachoeirinha Loft	Nordeste	2380	4	4	4	1
3	Emporio Mineirinho	Sudeste	2300	5	5	5	1
6	Casa Flor & Anópolis	Sul	2100	6	6	6	2
12	Nordestemido Hall	Nordeste	1910	7	7	7	2
4	Central Paulista	Sudeste	1800	8	8	8	2
1	Botafogo Praia&Mar	Sudeste	1800	9	8	8	2
11	Tintas Rio Branco	Norte	1500	10	10	9	3
7	Pampas & Co	Sul	990	11	11	10	3
2	Lojas Vitoria	Sudeste	800	12	12	11	3
5	Rio 90 graus	Sudeste	700	13	13	12	3

# Funções de Classificação com PARTITION BY

As funções de classificação também podem ser utilizadas dentro de repartições, ou seja, grupos internos a nossa tabela, por meio do PARTITION BY.

No exemplo ao lado, as funções de classificação são aplicadas particionando por regiões. Ou seja, é como se o PARTITION BY deixasse claro para o SQL que a tabela deve ser entendida como diversas tabelas separadas, fazendo essa divisão por região.

The screenshot shows a SQL query editor with a code pane and a results pane. The code pane contains the following SQL:

```
SELECT
    *,
    ROW_NUMBER() OVER(PARTITION BY Regiao ORDER BY Qtd_Vendida DESC) AS 'rownumber',
    RANK() OVER(PARTITION BY Regiao ORDER BY Qtd_Vendida DESC) AS 'rank',
    DENSE_RANK() OVER(PARTITION BY Regiao ORDER BY Qtd_Vendida DESC) AS 'dense',
    NTILE(3) OVER(PARTITION BY Regiao ORDER BY Qtd_Vendida DESC) AS 'ntile'
FROM
    Lojas
```

The results pane displays a table titled "Resultados" with the following data:

ID_Loja	Nome_Loja	Regiao	Qtd_Vendida	rownumber	rank	dense	ntile
13	Cachoerinha Loft	Nordeste	2380	1	1	1	1
12	Nordestemido Hall	Nordeste	1910	2	2	2	2
9	Amazonas Prime	Norte	4200	1	1	1	1
10	Pará Bens	Norte	3200	2	2	2	2
11	Tintas Rio Branco	Norte	1500	3	3	3	3
3	Emporio Mineirinho	Sudeste	2300	1	1	1	1
4	Central Paulista	Sudeste	1800	2	2	2	1
1	Botafogo Praia&Mar	Sudeste	1800	3	2	2	2
2	Lojas Vitoria	Sudeste	800	4	4	3	2
5	Rio 90 graus	Sudeste	700	5	5	4	3
8	Paraná Papéis	Sul	2800	1	1	1	1
6	Casa Flor & Anópolis	Sul	2100	2	2	2	2
7	Pampas & Co	Sul	990	3	3	3	3

# RANK com GROUP BY

Podemos também utilizar a função RANK para criar rankings em dados agrupados.

Ou seja, criamos agrupamentos com o GROUP BY e fazemos um ranking considerando estes dados agrupados.

No exemplo ao lado, criamos um agrupamento por Região, e adicionamos a essa tabela agrupada uma coluna de ranking.

```
SELECT
    Regiao AS 'Região',
    SUM(Qtd_Vendida) AS 'Total Vendido',
    RANK() OVER(ORDER BY SUM(Qtd_Vendida) DESC) AS 'Rank'
FROM
    Lojas
GROUP BY Regiao
ORDER BY Rank
```

Região	Total Vendido	Rank
Norte	8900	1
Sudeste	7400	2
Sul	5890	3
Nordeste	4290	4

# Código para criar a tabela Resultado

Para os próximos exercícios, será importante ter esta tabela. Basta copiar o código e reproduzir no seu SSMS.

```
CREATE TABLE Resultado(
```

```
    Data_Fechamento DATETIME,
```

```
    Faturamento_MM FLOAT)
```

```
INSERT INTO Resultado(Data_Fechamento, Faturamento_MM)
```

```
VALUES
```

```
    ('01/01/2020', 8),
```

```
    ('01/02/2020', 10),
```

```
    ('01/03/2020', 6),
```

```
    ('01/04/2020', 9),
```

```
    ('01/05/2020', 5),
```

```
    ('01/06/2020', 4),
```

```
    ('01/07/2020', 7),
```

```
    ('01/08/2020', 11),
```

```
    ('01/09/2020', 9),
```

```
    ('01/10/2020', 12),
```

```
    ('01/11/2020', 11),
```

```
    ('01/12/2020', 10)
```



# Cálculo de soma móvel

Uma outra análise que podemos fazer é a de soma móvel. A forma mais fácil de entender o funcionamento é observar o exemplo ao lado.

Na tabela, temos uma coluna com o faturamento de cada mês e outra coluna com a soma móvel.

129 %

Resultados Mensagens

	Data do Fechamento	Faturamento Total (em milhões)	Soma móvel (em milhões)
1	2020-01-01 00:00:00.000	8	8
2	2020-02-01 00:00:00.000	10	18
3	2020-03-01 00:00:00.000	6	16
4	2020-04-01 00:00:00.000	9	15
5	2020-05-01 00:00:00.000	5	14
6	2020-06-01 00:00:00.000	4	9
7	2020-07-01 00:00:00.000	7	11
8	2020-08-01 00:00:00.000	11	18
9	2020-09-01 00:00:00.000	9	20
10	2020-10-01 00:00:00.000	12	21
11	2020-11-01 00:00:00.000	11	23
12	2020-12-01 00:00:00.000	10	21

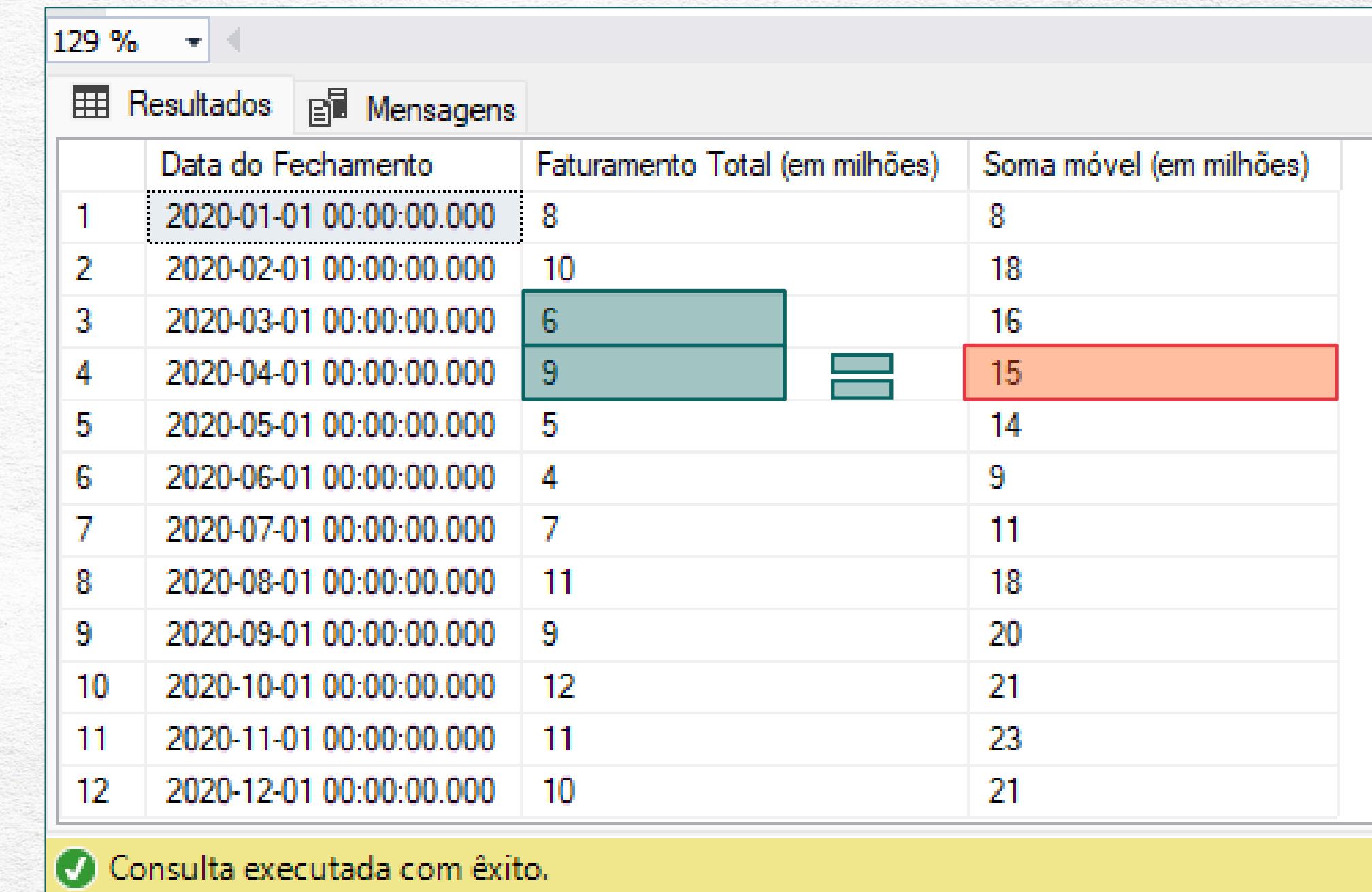
Consulta executada com êxito.

# Cálculo de soma móvel

Se você observar bem, o **valor 15** marcado na imagem ao lado se trata do **valor 9** (Faturamento Total da linha 4) **somado** com o **valor 6** (Faturamento Total da linha 3).

Na verdade, qualquer valor da coluna Soma móvel é a soma do valor do Faturamento Total na sua linha mais o Faturamento Total na linha anterior. Isso é o que chamamos de soma móvel.

Vamos ver como é feito este cálculo usando comandos em SQL.



The screenshot shows a database query results window with the following details:

- Zoom Level:** 129 %
- Results Tab:** Selected (Resultados)
- Columns:**
  - Data do Fechamento
  - Faturamento Total (em milhões)
  - Soma móvel (em milhões)
- Rows:**

	Data do Fechamento	Faturamento Total (em milhões)	Soma móvel (em milhões)
1	2020-01-01 00:00:00.000	8	8
2	2020-02-01 00:00:00.000	10	18
3	2020-03-01 00:00:00.000	6	16
4	2020-04-01 00:00:00.000	9	15
5	2020-05-01 00:00:00.000	5	14
6	2020-06-01 00:00:00.000	4	9
7	2020-07-01 00:00:00.000	7	11
8	2020-08-01 00:00:00.000	11	18
9	2020-09-01 00:00:00.000	9	20
10	2020-10-01 00:00:00.000	12	21
11	2020-11-01 00:00:00.000	11	23
12	2020-12-01 00:00:00.000	10	21
- Status Bar:** Consulta executada com êxito.

# Cálculo de soma móvel

Para fazer essa soma móvel, precisamos usar uma função de janela. Usamos a função SUM junto com a cláusula OVER para somar os valores de um determinado intervalo, como mostrado na figura ao lado.

The screenshot shows a SQL query window and a results grid. The query is:

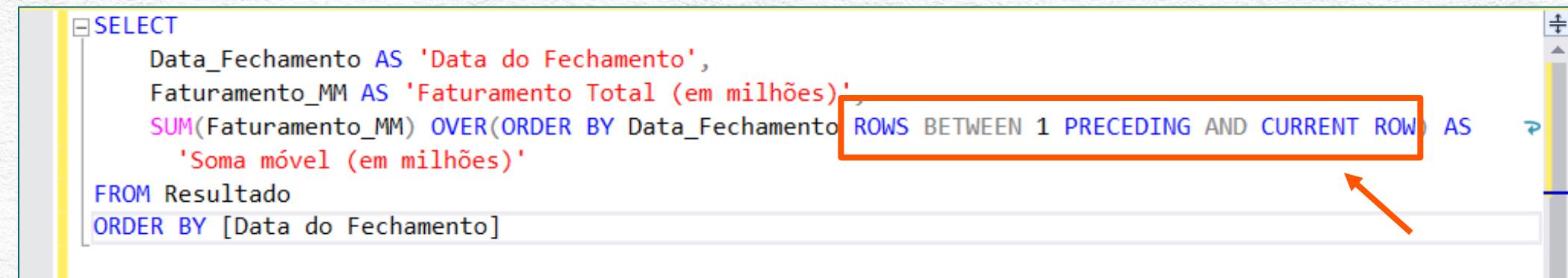
```
SELECT
    Data_Fechamento AS 'Data do Fechamento',
    Faturamento_MM AS 'Faturamento Total (em milhões)',
    SUM(Faturamento_MM) OVER(ORDER BY Data_Fechamento ROWS BETWEEN 1 PRECEDING AND CURRENT ROW) AS
        'Soma móvel (em milhões)'
FROM Resultado
ORDER BY [Data do Fechamento]
```

The results grid displays 12 rows of data, ordered by Data do Fechamento. The columns are:

	Data do Fechamento	Faturamento Total (em milhões)	Soma móvel (em milhões)
1	2020-01-01 00:00:00.000	8	8
2	2020-02-01 00:00:00.000	10	18
3	2020-03-01 00:00:00.000	6	16
4	2020-04-01 00:00:00.000	9	15
5	2020-05-01 00:00:00.000	5	14
6	2020-06-01 00:00:00.000	4	9
7	2020-07-01 00:00:00.000	7	11
8	2020-08-01 00:00:00.000	11	18
9	2020-09-01 00:00:00.000	9	20
10	2020-10-01 00:00:00.000	12	21
11	2020-11-01 00:00:00.000	11	23
12	2020-12-01 00:00:00.000	10	21

A message at the bottom of the results grid says "Consulta executada com êxito." (Query executed successfully).

# Cálculo de soma móvel



```
SELECT
    Data_Fechamento AS 'Data do Fechamento',
    Faturamento_MM AS 'Faturamento Total (em milhões)',
    SUM(Faturamento_MM) OVER(ORDER BY Data_Fechamento
        ROWS BETWEEN 1 PRECEDING AND CURRENT ROW) AS
        'Soma móvel (em milhões)'
    FROM Resultado
    ORDER BY [Data do Fechamento]
```

Este intervalo de soma pode ser especificado pelo trecho marcado na imagem acima. Observe a parte:

ROWS BETWEEN 1 PRECEDING AND CURRENT ROW

Podemos traduzir esta parte literalmente por:

LINHAS ENTRE 1 ANTES E A LINHA ATUAL

Como queremos fazer uma soma, isto significa que: a nossa janela de soma será entre as linhas anterior e atual. Ou seja, essa soma será feita considerando sempre os valores da linha atual + a linha anterior.

# Cálculo de soma móvel

Podemos testar outros resultados personalizando a janela de soma. Dessa vez, se colocarmos

**ROWS BETWEEN 3 PRECEDING AND CURRENT ROW**

Ele vai fazer a soma da linha atual com as 3 linhas anteriores, conforme imagem ao lado.

```

SELECT
    Data_Fechamento AS 'Data do Fechamento',
    Faturamento_MM AS 'Faturamento Total (em milhões)',
    SUM(Faturamento_MM) OVER(ORDER BY Data_Fechamento ROWS BETWEEN 3 PRECEDING AND CURRENT ROW) AS
        'Soma móvel (em milhões)'
FROM Resultado
ORDER BY [Data do Fechamento]

```

	Data do Fechamento	Faturamento Total (em milhões)	Soma móvel (em milhões)
1	2020-01-01 00:00:00.000	8	8
2	2020-02-01 00:00:00.000	10	18
3	2020-03-01 00:00:00.000	6	24
4	2020-04-01 00:00:00.000	9	33
5	2020-05-01 00:00:00.000	5	30
6	2020-06-01 00:00:00.000	4	24
7	2020-07-01 00:00:00.000	7	25
8	2020-08-01 00:00:00.000	11	27
9	2020-09-01 00:00:00.000	9	31
10	2020-10-01 00:00:00.000	12	39
11	2020-11-01 00:00:00.000	11	43
12	2020-12-01 00:00:00.000	10	42

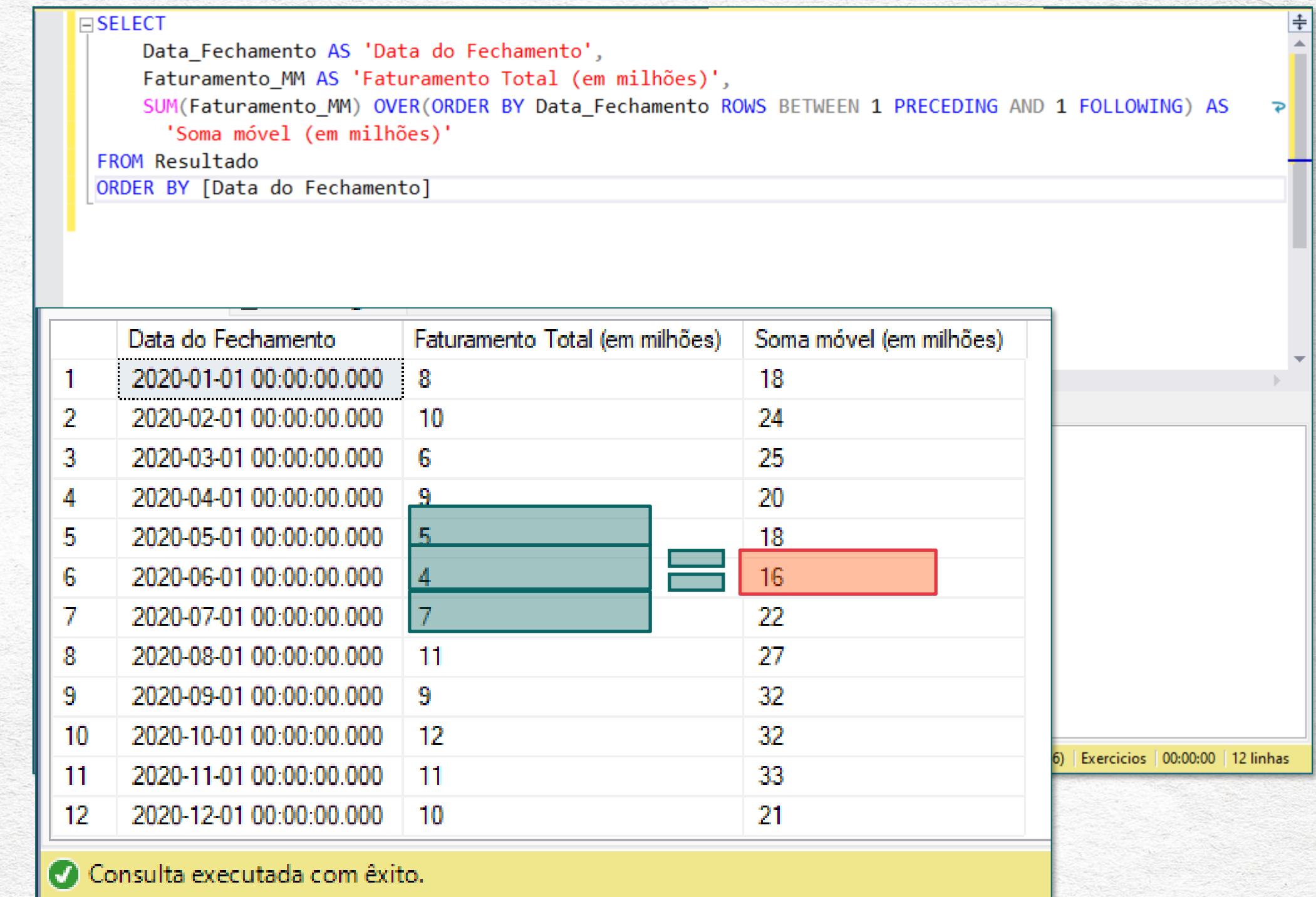
Consulta executada com êxito.

# Cálculo de soma móvel

Outra possibilidade é personalizar a janela para que ela considere valores antes da linha e depois da linha em questão. Por exemplo, na imagem ao lado, cada valor da coluna Soma móvel é a soma do Faturamento Total da linha anterior + Faturamento Total da linha atual + Faturamento Total da linha seguinte. Para conseguir estender a nossa janela para a linha seguinte, usamos o comando FOLLOWING.

**ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING**

Em resumo, podemos usar o PRECEDING ou o FOLLOWING para personalizar o “tamanho” da nossa janela de cálculo.



The screenshot shows a SQL query and its execution results. The query uses the OVER clause with ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING to calculate a moving sum of the 'Faturamento\_MM' column. The results are ordered by 'Data\_Fechamento'. A yellow bar at the bottom indicates the query was executed successfully.

```

SELECT
    Data_Fechamento AS 'Data do Fechamento',
    Faturamento_MM AS 'Faturamento Total (em milhões)',
    SUM(Faturamento_MM) OVER(ORDER BY Data_Fechamento ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS 'Soma móvel (em milhões)'
FROM Resultado
ORDER BY [Data do Fechamento]
  
```

	Data do Fechamento	Faturamento Total (em milhões)	Soma móvel (em milhões)
1	2020-01-01 00:00:00.000	8	18
2	2020-02-01 00:00:00.000	10	24
3	2020-03-01 00:00:00.000	6	25
4	2020-04-01 00:00:00.000	9	20
5	2020-05-01 00:00:00.000	5	18
6	2020-06-01 00:00:00.000	4	16
7	2020-07-01 00:00:00.000	7	22
8	2020-08-01 00:00:00.000	11	27
9	2020-09-01 00:00:00.000	9	32
10	2020-10-01 00:00:00.000	12	32
11	2020-11-01 00:00:00.000	11	33
12	2020-12-01 00:00:00.000	10	21

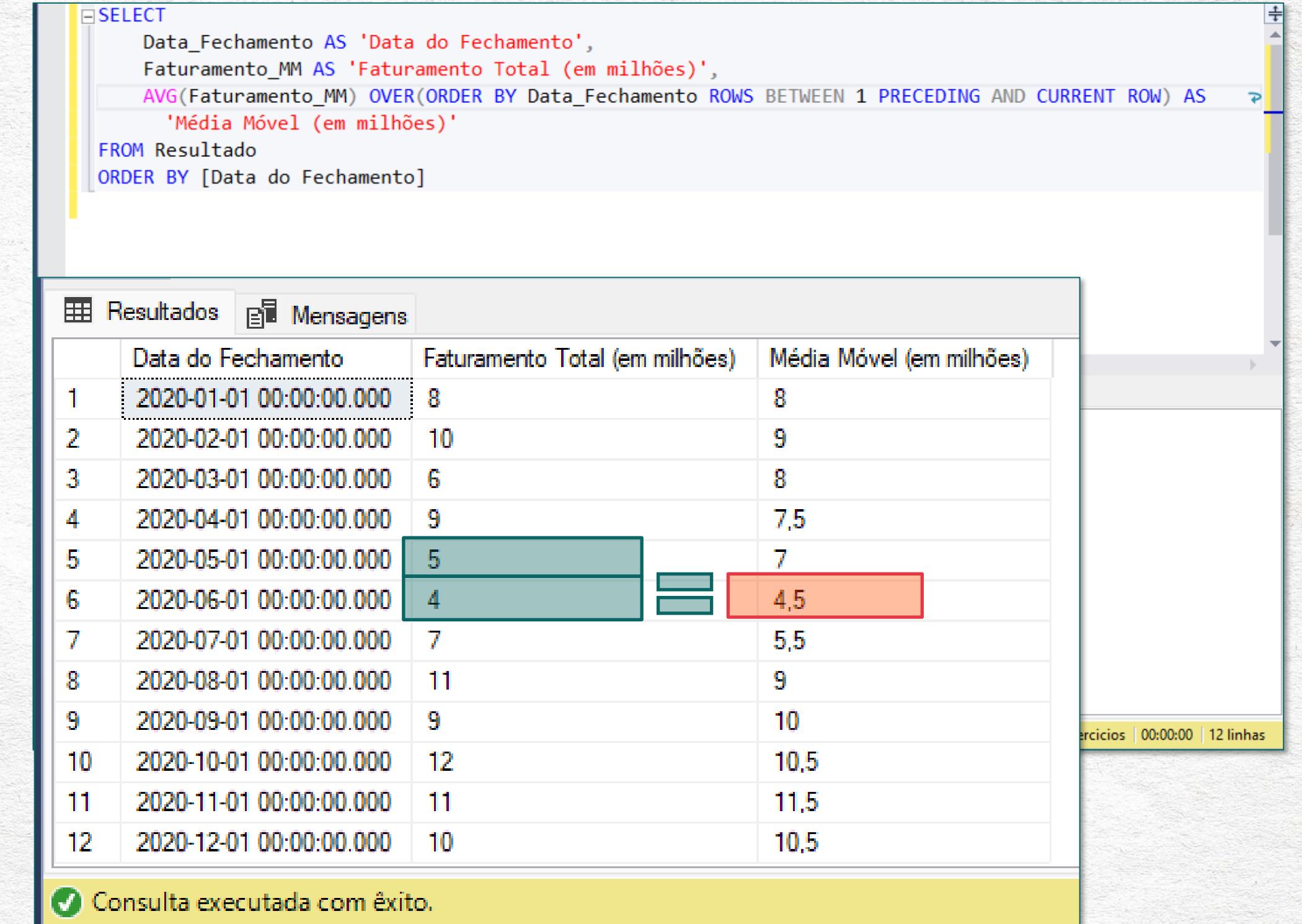
6) | Exercícios | 00:00:00 | 12 linhas

Consulta executada com êxito.

# Cálculo de média móvel

Para fazer uma média móvel, seguimos exatamente a mesma lógica explicada anteriormente. A única diferença é que agora usamos o AVG no lugar do SUM.

Na verdade, podemos fazer qualquer análise móvel apenas trocando a função (COUNT, MIN, MAX, SUM, AVG).



The screenshot shows a SQL query window with the following code:

```

SELECT
    Data_Fechamento AS 'Data do Fechamento',
    Faturamento_MM AS 'Faturamento Total (em milhões)',
    AVG(Faturamento_MM) OVER(ORDER BY Data_Fechamento ROWS BETWEEN 1 PRECEDING AND CURRENT ROW) AS
        'Média Móvel (em milhões)'
FROM Resultado
ORDER BY [Data do Fechamento]
  
```

The results table has three columns: Data do Fechamento, Faturamento Total (em milhões), and Média Móvel (em milhões). The data is as follows:

	Data do Fechamento	Faturamento Total (em milhões)	Média Móvel (em milhões)
1	2020-01-01 00:00:00.000	8	8
2	2020-02-01 00:00:00.000	10	9
3	2020-03-01 00:00:00.000	6	8
4	2020-04-01 00:00:00.000	9	7,5
5	2020-05-01 00:00:00.000	5	7
6	2020-06-01 00:00:00.000	4	4,5
7	2020-07-01 00:00:00.000	7	5,5
8	2020-08-01 00:00:00.000	11	9
9	2020-09-01 00:00:00.000	9	10
10	2020-10-01 00:00:00.000	12	10,5
11	2020-11-01 00:00:00.000	11	11,5
12	2020-12-01 00:00:00.000	10	10,5

A message at the bottom says "Consulta executada com êxito." (Query executed successfully.)

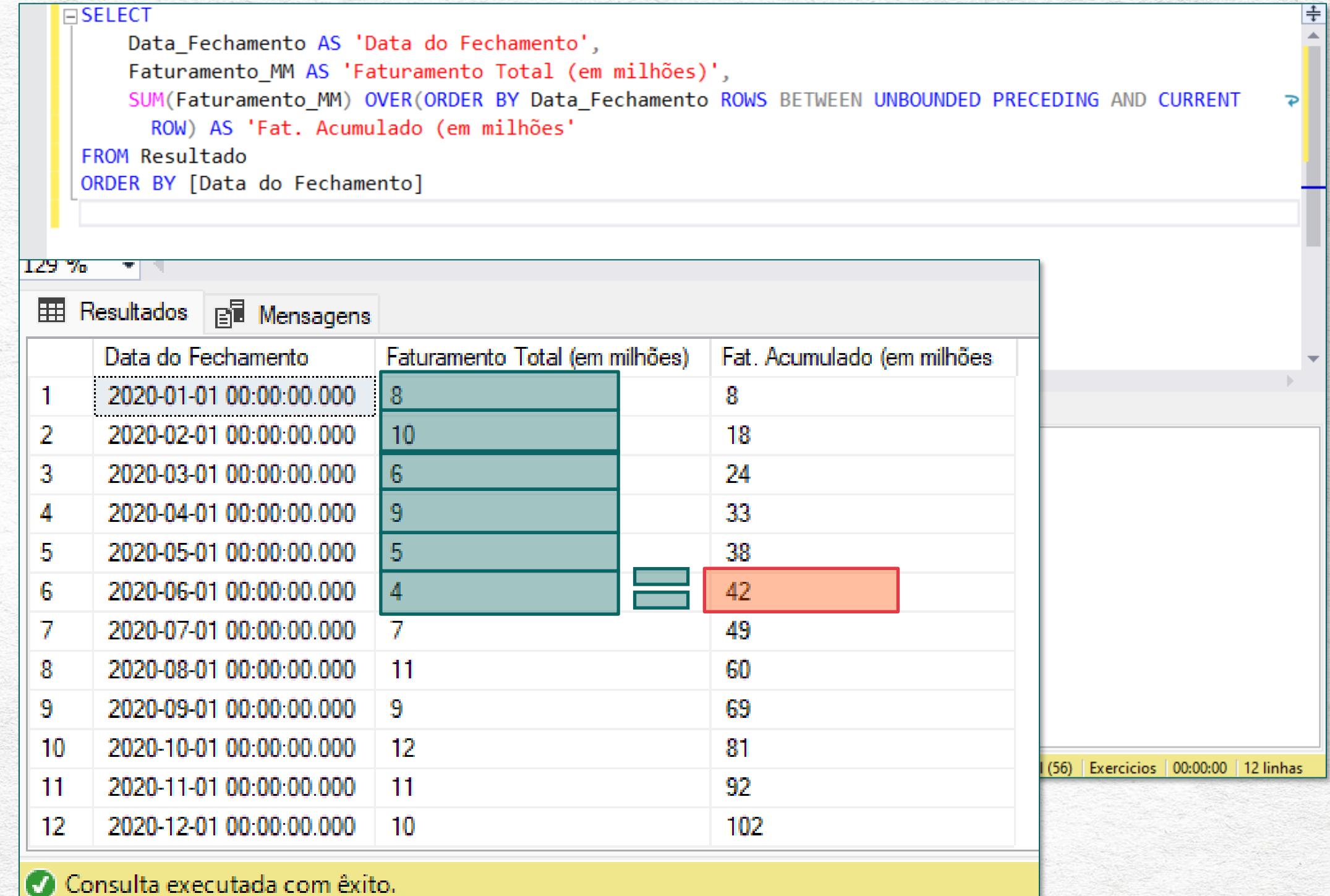
# Cálculo de acumulado

Para fazer um cálculo de acumulado, precisamos fazer com que a nossa janela de análise comece sempre na data mais antiga e vá até a linha atual.

Observe a imagem ao lado. O valor de Fat. Acumulado na linha onde o valor é 42 é a soma de Faturamento Total desde a primeira linha da tabela até aquela linha ( $8+10+6+9+5+4=42$ ).

Isso significa que a janela de soma deve ser entre a linha mais anterior até a linha atual. Informar a linha atual nós sabemos: será por meio da CURRENT ROW.

Já para informar a linha mais anterior, usamos o UNBOUNDED PRECEDING.



```

SELECT
    Data_Fechamento AS 'Data do Fechamento',
    Faturamento_MM AS 'Faturamento Total (em milhões)',
    SUM(Faturamento_MM) OVER(ORDER BY Data_Fechamento ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS 'Fat. Acumulado (em milhões)'
FROM Resultado
ORDER BY [Data do Fechamento]
  
```

	Data do Fechamento	Faturamento Total (em milhões)	Fat. Acumulado (em milhões)
1	2020-01-01 00:00:00.000	8	8
2	2020-02-01 00:00:00.000	10	18
3	2020-03-01 00:00:00.000	6	24
4	2020-04-01 00:00:00.000	9	33
5	2020-05-01 00:00:00.000	5	38
6	2020-06-01 00:00:00.000	4	42
7	2020-07-01 00:00:00.000	7	49
8	2020-08-01 00:00:00.000	11	60
9	2020-09-01 00:00:00.000	9	69
10	2020-10-01 00:00:00.000	12	81
11	2020-11-01 00:00:00.000	11	92
12	2020-12-01 00:00:00.000	10	102

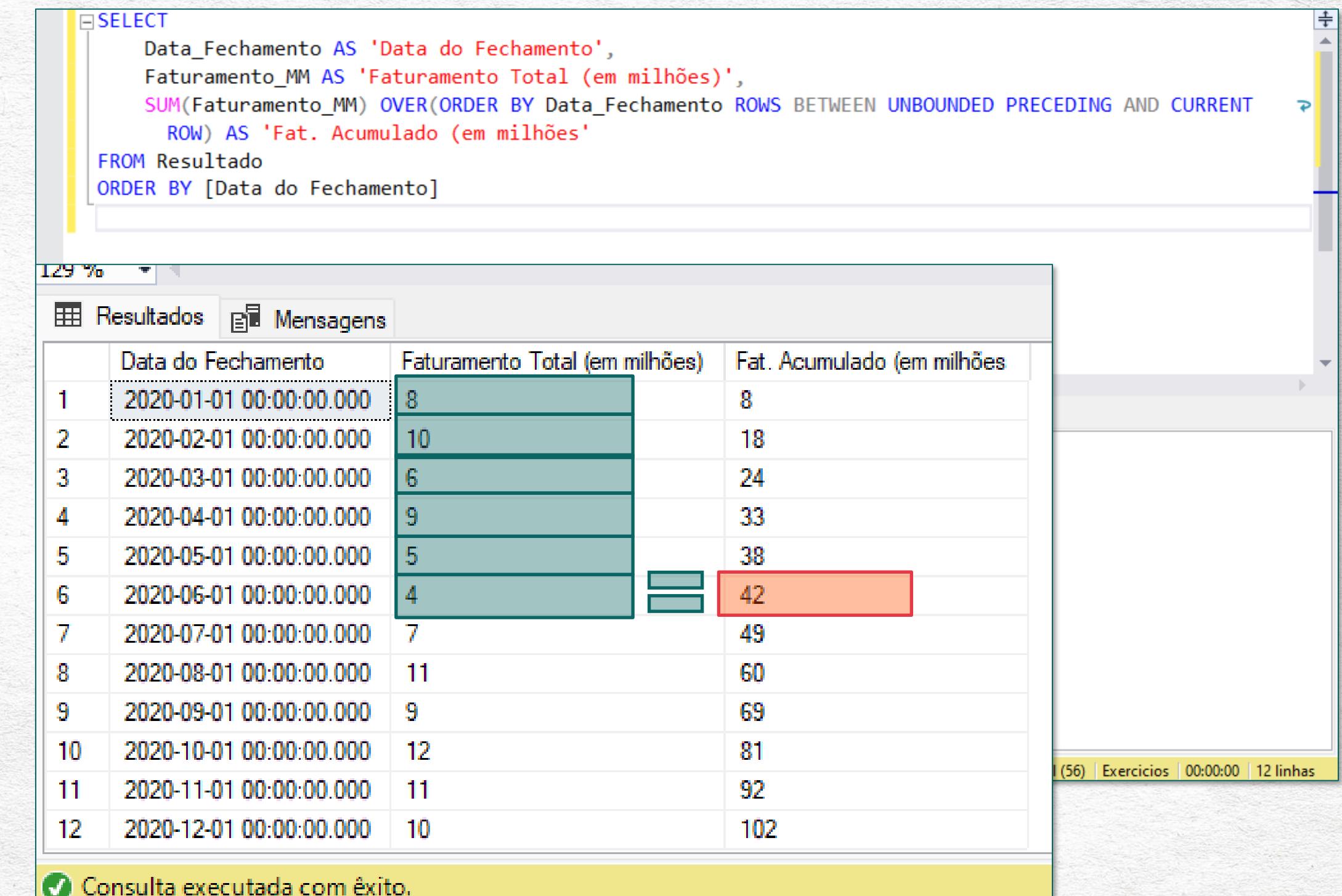
Consulta executada com êxito.

# Cálculo de acumulado

Ou seja, sempre que precisarmos fazer um cálculo com uma janela que começa na primeira linha da tabela, o nosso intervalo começará com:

## UNBOUNDED PRECEDING

De forma muito semelhante, se quisermos uma janela que vá até a última linha, usamos o UNBOUNDED FOLLOWING.



```

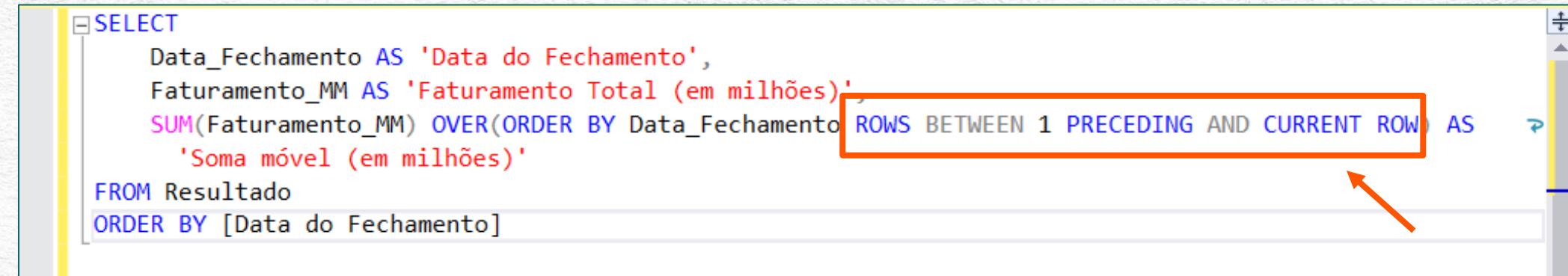
SELECT
    Data_Fechamento AS 'Data do Fechamento',
    Faturamento_MM AS 'Faturamento Total (em milhões)',
    SUM(Faturamento_MM) OVER(ORDER BY Data_Fechamento ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS 'Fat. Acumulado (em milhões)'
FROM Resultado
ORDER BY [Data do Fechamento]
  
```

The screenshot shows a SQL query in the top pane and its results in the bottom pane. The query calculates cumulative sales from January to the current month. The results are as follows:

	Data do Fechamento	Faturamento Total (em milhões)	Fat. Acumulado (em milhões)
1	2020-01-01 00:00:00.000	8	8
2	2020-02-01 00:00:00.000	10	18
3	2020-03-01 00:00:00.000	6	24
4	2020-04-01 00:00:00.000	9	33
5	2020-05-01 00:00:00.000	5	38
6	2020-06-01 00:00:00.000	4	42
7	2020-07-01 00:00:00.000	7	49
8	2020-08-01 00:00:00.000	11	60
9	2020-09-01 00:00:00.000	9	69
10	2020-10-01 00:00:00.000	12	81
11	2020-11-01 00:00:00.000	11	92
12	2020-12-01 00:00:00.000	10	102

A message at the bottom says "Consulta executada com êxito."

# Resumo das janelas



```
SELECT
    Data_Fechamento AS 'Data do Fechamento',
    Faturamento_MM AS 'Faturamento Total (em milhões)',
    SUM(Faturamento_MM) OVER(ORDER BY Data_Fechamento ROWS BETWEEN 1 PRECEDING AND CURRENT ROW) AS 'Soma móvel (em milhões)'
FROM Resultado
ORDER BY [Data do Fechamento]
```

Você pode entender também a parte destacada acima como uma especificação do intervalo da nossa janela de cálculo. Ela terá a seguinte estrutura:

ROWS BETWEEN *inicio\_do\_intervalo* AND *fim\_do\_intervalo*

Exemplos desses intervalos podem ser:

- **ROWS BETWEEN 1 PRECEDING AND CURRENT ROW**: 1 linha antes e a linha atual.
- **ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW**: todas as linhas antes e a linha atual.
- **ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING**: 1 linha antes e 1 linha depois da atual.
- **ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING**: todas as linhas antes e todas as linhas depois da linha atual.



## LAG

Faz um deslocamento nos valores, retornando um valor que está em uma posição anterior (1 linha anterior, 2 linhas anteriores, etc).



## LEAD

Faz um deslocamento nos valores, retornando o valor que está em uma posição posterior (1 linha seguinte, 2 linhas seguintes, etc).



## FIRST\_VALUE

Cria uma coluna de ranking. Em caso de empates, os rankings dos números se repetem, e o próximo ranking continua a sequência. Ex: dois valores na posição 8, o valor seguinte continua na posição 9.



## LAST\_VALUE

Distribui os valores de uma coluna em grupos.

# Função LAG

O funcionamento da função LAG é bem intuitivo. Ele permite acessar valores anteriores.

No exemplo ao lado, cada valor da coluna Valor anterior se trata do valor da linha anterior, na coluna Faturamento Total.

`LAG(Faturamento_MM, 1, 0) OVER(ORDER BY Data_Fechamento) AS 'Valor anterior'`

A função LAG pede 3 argumentos:

- 1) Qual é a coluna com os valores que queremos deslocar.
- 2) Quantas linhas pra trás queremos deslocar.
- 3) Quando não for encontrado um valor anterior, qual é o valor que deve aparecer.

```

SELECT
    Data_Fechamento AS 'Data do Fechamento',
    Faturamento_MM AS 'Faturamento Total (em milhões)',
    LAG(Faturamento_MM, 1, 0) OVER(ORDER BY Data_Fechamento) AS 'Valor anterior'
FROM Resultado
ORDER BY [Data do Fechamento]

```

	Data do Fechamento	Faturamento Total (em milhões)	Valor anterior
1	2020-01-01 00:00:00.000	8	0
2	2020-02-01 00:00:00.000	10	8
3	2020-03-01 00:00:00.000	6	10
4	2020-04-01 00:00:00.000	9	6
5	2020-05-01 00:00:00.000	5	9
6	2020-06-01 00:00:00.000	4	5
7	2020-07-01 00:00:00.000	7	4
8	2020-08-01 00:00:00.000	11	7
9	2020-09-01 00:00:00.000	9	11
10	2020-10-01 00:00:00.000	12	9
11	2020-11-01 00:00:00.000	11	12
12	2020-12-01 00:00:00.000	10	11

RTM) | MARCUS\caval (57) | Exercícios | 00:00:00 | 12 linhas

Consulta executada com êxito.

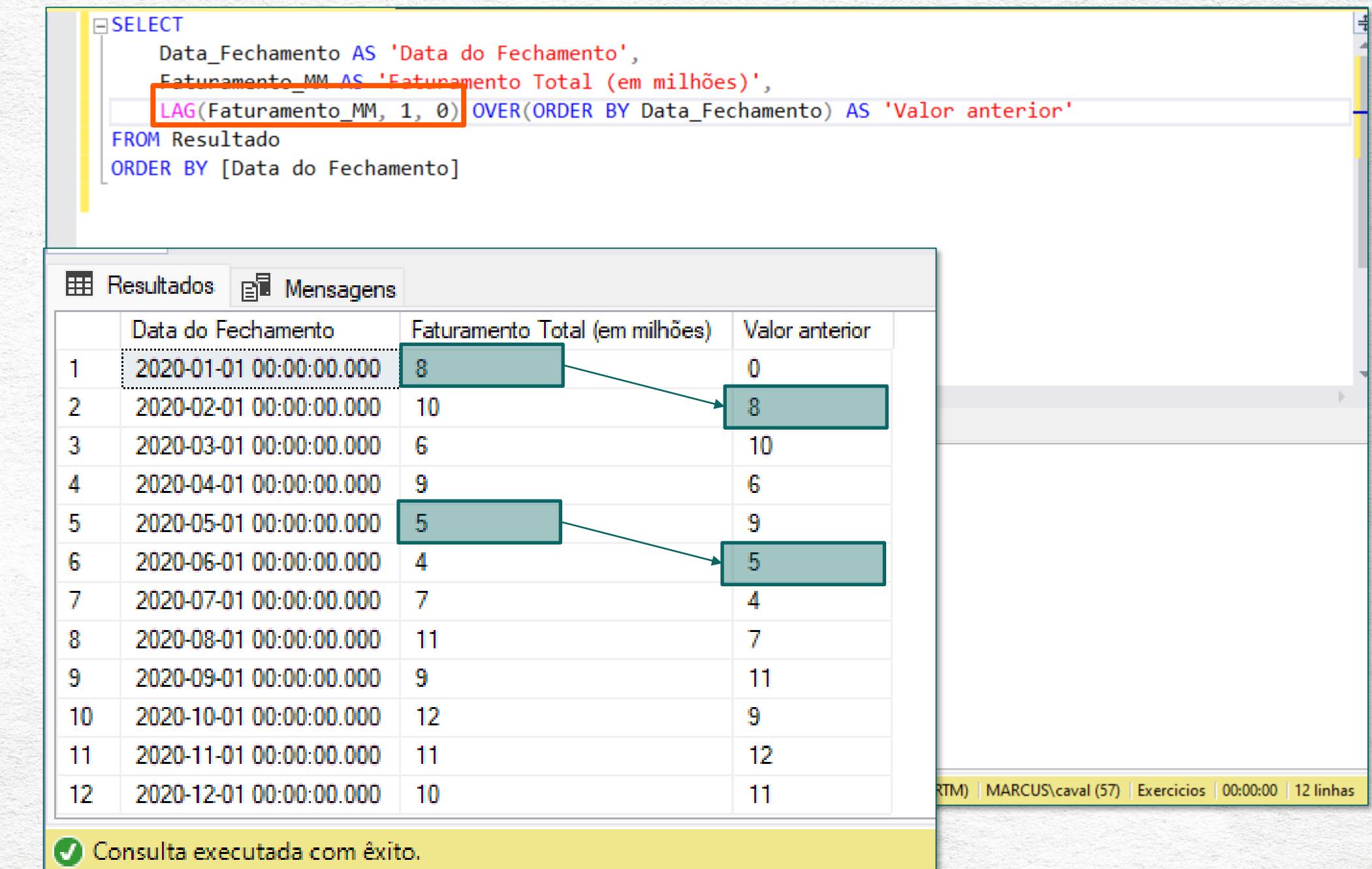
# Função LAG

No exemplo ao lado, temos os seguintes argumentos para a função:

**1º argumento)** Expressão: Faturamento\_MM - coluna que queremos acessar valores anteriores.

**2º argumento)** Offset: 1 - quantas linhas queremos deslocar para trás.

**3º argumento)** Default: 0 - valor padrão que queremos que retorne caso ele não encontre nenhum valor (na linha 1 da tabela ao lado, retornou zero porque não tem nenhum outro valor de Faturamento antes da linha 1).



The screenshot shows a SQL query in the top pane and its results in the bottom pane. The query is:

```

SELECT
    Data_Fechamento AS 'Data do Fechamento',
    Faturamento_MM AS 'Faturamento Total (em milhões)',
    LAG(Faturamento_MM, 1, 0) OVER(ORDER BY Data_Fechamento) AS 'Valor anterior'
FROM Resultado
ORDER BY [Data do Fechamento]

```

The results table has three columns: Data do Fechamento, Faturamento Total (em milhões), and Valor anterior. The Valor anterior column shows the value from the previous row for each row, starting with 0 for the first row (2020-01-01).

	Data do Fechamento	Faturamento Total (em milhões)	Valor anterior
1	2020-01-01 00:00:00.000	8	0
2	2020-02-01 00:00:00.000	10	8
3	2020-03-01 00:00:00.000	6	10
4	2020-04-01 00:00:00.000	9	6
5	2020-05-01 00:00:00.000	5	9
6	2020-06-01 00:00:00.000	4	5
7	2020-07-01 00:00:00.000	7	4
8	2020-08-01 00:00:00.000	11	7
9	2020-09-01 00:00:00.000	9	11
10	2020-10-01 00:00:00.000	12	9
11	2020-11-01 00:00:00.000	11	12
12	2020-12-01 00:00:00.000	10	11

A yellow banner at the bottom of the results pane says "Consulta executada com êxito."

# Função LAG

Vejamos um outro exemplo:

**1º argumento)** Expressão: Faturamento\_MM - coluna que queremos acessar valores anteriores.

**2º argumento)** Offset: 2 - quantas linhas queremos deslocar para trás.

**3º argumento)** Default: não informado.

Repare agora que a coluna Valor anterior está “pegando” o valor 2 casas para trás na coluna Faturamento Total. E como não informamos o default, os dois primeiros valores ficaram NULL (porque deslocando o Faturamento Total duas casas para trás, não tem nenhum valor disponível).

```

SELECT
    Data_Fechamento AS 'Data do Fechamento',
    Faturamento_MM AS 'Faturamento Total (em milhões)',
    LAG(Faturamento_MM, 2) OVER(ORDER BY Data_Fechamento) AS 'Valor anterior'
FROM Resultado
ORDER BY [Data do Fechamento]
  
```

	Data do Fechamento	Faturamento Total (em milhões)	Valor anterior
1	2020-01-01 00:00:00.000	8	NULL
2	2020-02-01 00:00:00.000	10	NULL
3	2020-03-01 00:00:00.000	6	8
4	2020-04-01 00:00:00.000	9	10
5	2020-05-01 00:00:00.000	5	6
6	2020-06-01 00:00:00.000	4	9
7	2020-07-01 00:00:00.000	7	5
8	2020-08-01 00:00:00.000	11	4
9	2020-09-01 00:00:00.000	9	7
10	2020-10-01 00:00:00.000	12	11
11	2020-11-01 00:00:00.000	11	9
12	2020-12-01 00:00:00.000	10	12

S\caval (57) | Exercícios | 00:00:00 | 12 linhas

Consulta executada com êxito.

# Função LEAD

A função LEAD fará o oposto. Agora, o deslocamento será feito para frente. Então observe na imagem ao lado que ao usar o LEAD com o 2º argumento igual a 1, acessamos o valor de Faturamento Total da linha seguinte.

Ou seja, na linha 1, o valor 10 refere-se ao Faturamento Total da linha 2.

```

SELECT
    Data_Fechamento AS 'Data do Fechamento',
    Faturamento_MM AS 'Faturamento Total (em milhões)',
    LEAD(Faturamento_MM, 1, 0) OVER(ORDER BY Data_Fechamento) AS 'Valor seguinte'
FROM Resultado
ORDER BY [Data do Fechamento]
  
```

	Data do Fechamento	Faturamento Total (em milhões)	Valor seguinte
1	2020-01-01 00:00:00.000	8	10
2	2020-02-01 00:00:00.000	10	6
3	2020-03-01 00:00:00.000	6	9
4	2020-04-01 00:00:00.000	9	5
5	2020-05-01 00:00:00.000	5	4
6	2020-06-01 00:00:00.000	4	7
7	2020-07-01 00:00:00.000	7	11
8	2020-08-01 00:00:00.000	11	9
9	2020-09-01 00:00:00.000	9	12
10	2020-10-01 00:00:00.000	12	11
11	2020-11-01 00:00:00.000	11	10
12	2020-12-01 00:00:00.000	10	0

Acaval (57) | Exercícios | 00:00:00 | 12 linhas

Consulta executada com êxito.

# Função NULLIF

Vamos agora entender o funcionamento de uma função bem simples, mas muito útil: a função NULLIF.

Essa função recebe dois argumentos. Quando o valor dos dois argumentos é igual, ela retorna NULL. Quando o valor dos dois argumentos é diferente, ele retorna o valor do primeiro argumento.

Observe no exemplo ao lado. NULLIF(3, 3) retorna como resultado o valor NULL. Já o NULLIF(3, 5) retorna como resultado o valor 3, que é o valor do primeiro argumento.

Essa função tem uma funcionalidade muito simples, que será muito útil na nossa aplicação de cálculo do MoM a seguir.

The screenshot shows a SQL query window with the following code:

```
SELECT
    NULLIF(3, 3) AS 'Igual',
    NULLIF(3, 5) AS 'Diferente'
```

The results pane displays the output:

	Igual	Diferente
1	NULL	3

The 'Resultados' tab is selected, and the status bar at the bottom left indicates '29 %'.

# Cálculo MoM

O cálculo MoM (Month Over Month) nada mais é do que um cálculo de variação percentual de crescimento/decrescimento de um valor em relação ao mês anterior.

Por exemplo: qual foi a variação percentual do Faturamento de fevereiro em relação ao mês anterior? A fórmula é a seguinte:

Variação % MoM = (Faturamento Mês Atual/Faturamento Mês Anterior) – 1

O resultado para fevereiro portanto seria:

$$\text{Variação \%} = (10/8) - 1 = 24,25\%$$

```
SELECT
    Data_Fechamento AS 'Data do Fechamento',
    Faturamento_MM AS 'Faturamento Total (em milhões)',
    LAG(Faturamento_MM, 1, 0) OVER(ORDER BY Data_Fechamento) AS 'Faturamento Mês Anterior'
FROM Resultado
ORDER BY [Data do Fechamento]
```

	Data do Fechamento	Faturamento Total (em milhões)	Faturamento Mês Anterior
1	2020-01-01 00:00:00.000	8	0
2	2020-02-01 00:00:00.000	10	8
3	2020-03-01 00:00:00.000	6	10
4	2020-04-01 00:00:00.000	9	6
5	2020-05-01 00:00:00.000	5	9
6	2020-06-01 00:00:00.000	4	5
7	2020-07-01 00:00:00.000	7	4
8	2020-08-01 00:00:00.000	11	7
9	2020-09-01 00:00:00.000	9	11
10	2020-10-01 00:00:00.000	12	9
11	2020-11-01 00:00:00.000	11	12
12	2020-12-01 00:00:00.000	10	11

Consulta executada com êxito.

# Cálculo MoM

Para fazer esse cálculo de variação % precisaremos trabalhar com funções de janela, mais especificamente a função de offset **LAG** pois precisamos fazer o Faturamento Total dividido pelo Faturamento do Mês Anterior.

Na imagem ao lado é possível ver o Faturamento Total e o Faturamento Mês Anterior lado a lado, graças à aplicação da função de janela LAG. Agora, fica fácil fazer um dividido pelo outro.

```
SELECT
    Data_Fechamento AS 'Data do Fechamento',
    Faturamento_MM AS 'Faturamento Total (em milhões)',
    LAG(Faturamento_MM, 1, 0) OVER(ORDER BY Data_Fechamento) AS 'Faturamento Mês Anterior'
FROM Resultado
ORDER BY [Data do Fechamento]
```

	Data do Fechamento	Faturamento Total (em milhões)	Faturamento Mês Anterior
1	2020-01-01 00:00:00.000	8	0
2	2020-02-01 00:00:00.000	10	8
3	2020-03-01 00:00:00.000	6	10
4	2020-04-01 00:00:00.000	9	6
5	2020-05-01 00:00:00.000	5	9
6	2020-06-01 00:00:00.000	4	5
7	2020-07-01 00:00:00.000	7	4
8	2020-08-01 00:00:00.000	11	7
9	2020-09-01 00:00:00.000	9	11
10	2020-10-01 00:00:00.000	12	9
11	2020-11-01 00:00:00.000	11	12
12	2020-12-01 00:00:00.000	10	11

Consulta executada com êxito.

# Cálculo MoM

Ao fazer o cálculo de Faturamento\_MM/Faturamento Mês Anterior, no entanto, temos um erro no resultado: Erro de divisão por zero.

Isso aconteceu porque especificamente porque para a primeira linha, o Faturamento Mês Anterior é zero, pois não temos nenhum valor antes de 2020-01-01. Por isso, ao fazer  $(8/0) - 1$  temos uma divisão por zero neste caso.

	Data do Fechamento	Faturamento Total (em milhões)	Faturamento Mês Anterior
1	2020-01-01 00:00:00.000	8	0

Precisaremos portanto tratar essa divisão por zero. É ai que entra a função NULLIF.

```

SELECT
    Data_Fechamento AS 'Data do Fechamento',
    Faturamento_MM AS 'Faturamento Total (em milhões)',
    LAG(Faturamento_MM, 1, 0) OVER(ORDER BY Data_Fechamento) AS 'Faturamento Mês Anterior',
    Faturamento_MM/LAG(Faturamento_MM, 1, 0) OVER(ORDER BY Data_Fechamento) - 1 AS 'Crescimento MoM'
FROM Resultado
ORDER BY [Data do Fechamento]

```

Mensagem 8134, Nível 16, Estado 1, Linha 1  
Erro de divisão por zero.

Horário de conclusão: 2022-05-25T00:08:47.8174571-03:00

# Cálculo MoM

Usaremos a função NULLIF junto à função LAG, dado que ela retorna o valor zero na primeira linha da tabela, uma vez que não tem nenhum valor de Faturamento antes da primeira linha.

Lembrando que a função NULLIF retorna NULL se o resultado dos dois argumentos é igual, e retorna o resultado do primeiro argumento quando os dois valores são diferentes, ela será exatamente o que a gente precisa.

No primeiro mês, quando o Faturamento Mês Anterior é igual a zero, quanto temos NULLIF(0, 0) o resultado é NULL. Assim, o cálculo de variação fica  $8/\text{NULL} - 1$ . Qualquer valor dividido por NULL retorna NULL, então não teremos erro.

Agora quando o Faturamento Mês Anterior for diferente de zero, teremos NULLIF(Faturamento Mês Anterior, 0) e o resultado será Faturamento Mês Anterior.

```

SELECT
    Data_Fechamento AS 'Data do Fechamento',
    Faturamento_MM AS 'Faturamento Total (em milhões)',
    LAG(Faturamento_MM, 1, 0) OVER(ORDER BY Data_Fechamento) AS 'Faturamento Mês Anterior',
    Faturamento_MM/NULLIF(LAG(Faturamento_MM, 1, 0) OVER(ORDER BY Data_Fechamento), 0) - 1 AS 'Crescimento MoM'
FROM Resultado
ORDER BY [Data do Fechamento]
  
```

	Data do Fechamento	Faturamento Total (em milhões)	Faturamento Mês Anterior	Crescimento MoM
1	2020-01-01 00:00:00.000	8	0	NULL
2	2020-02-01 00:00:00.000	10	8	0,25
3	2020-03-01 00:00:00.000	6	10	-0,4
4	2020-04-01 00:00:00.000	9	6	0,5
5	2020-05-01 00:00:00.000	5	9	-0,4444444444444444
6	2020-06-01 00:00:00.000	4	5	-0,2
7	2020-07-01 00:00:00.000	7	4	0,75
8	2020-08-01 00:00:00.000	11	7	0,571428571428571
9	2020-09-01 00:00:00.000	9	11	-0,181818181818182
10	2020-10-01 00:00:00.000	12	9	0,3333333333333333
11	2020-11-01 00:00:00.000	11	12	-0,0833333333333334
12	2020-12-01 00:00:00.000	10	11	-0,0909090909090909

Consulta executada com êxito.

# Cálculo MoM

Para completar, podemos usar a função **FORMAT** para deixar o valor do Crescimento MoM no formato percentual.

Um detalhe importante é que alguns dos valores do Crescimento MoM é negativo. Isso porque em alguns meses, o Faturamento reduziu em comparação com o mês anterior, por isso a variação é negativa.

```
SELECT
    Data_Fechamento AS 'Data do Fechamento',
    Faturamento_MM AS 'Faturamento Total (em milhões)',
    LAG(Faturamento_MM, 1, 0) OVER(ORDER BY Data_Fechamento) AS 'Faturamento Mês Anterior',
    FORMAT(Faturamento_MM/NULLIF(LAG(Faturamento_MM, 1, 0) OVER(ORDER BY Data_Fechamento), 0) - 1, '0.00%') AS 'Crescimento MoM'
FROM Resultado
ORDER BY [Data do Fechamento]
```

	Data do Fechamento	Faturamento Total (em milhões)	Faturamento Mês Anterior	Crescimento MoM
1	2020-01-01 00:00:00.000	8	0	NULL
2	2020-02-01 00:00:00.000	10	8	25,00%
3	2020-03-01 00:00:00.000	6	10	-40,00%
4	2020-04-01 00:00:00.000	9	6	50,00%
5	2020-05-01 00:00:00.000	5	9	-44,44%
6	2020-06-01 00:00:00.000	4	5	-20,00%
7	2020-07-01 00:00:00.000	7	4	75,00%
8	2020-08-01 00:00:00.000	11	7	57,14%
9	2020-09-01 00:00:00.000	9	11	-18,18%
10	2020-10-01 00:00:00.000	12	9	33,33%
11	2020-11-01 00:00:00.000	11	12	-8,33%
12	2020-12-01 00:00:00.000	10	11	-9,09%

12 linhas

Consulta executada com êxito.

# FIRST\_VALUE e LAST\_VALUE

Fechamos com as funções FIRST\_VALUE e LAST\_VALUE. Estas funções permitem retornar o primeiro valor e o último valor de uma determinada coluna.

Ao lado, temos a aplicação dessas duas funções de janela. Para ambos, consideramos uma janela que vai do valor UNBOUNDED PRECEDING até o UNBOUNDED FOLLOWING, ou seja, todos os valores.

No exemplo ao lado, o FIRST\_VALUE (primeiro valor) de todo o Faturamento\_MM é o 8. Já o LAST\_VALUE (último valor) de todo o Faturamento\_MM é o 10.

```

SELECT
    Data_Fechamento AS 'Data do Fechamento',
    Faturamento_MM AS 'Faturamento Total (em milhões)',
    FIRST_VALUE(Faturamento_MM) OVER(ORDER BY Data_Fechamento ROWS BETWEEN UNBOUNDED PRECEDING AND
        UNBOUNDED FOLLOWING) AS 'Primeiro valor',
    LAST_VALUE(Faturamento_MM) OVER(ORDER BY Data_Fechamento ROWS BETWEEN UNBOUNDED PRECEDING AND
        UNBOUNDED FOLLOWING) AS 'Último valor'
FROM Resultado
ORDER BY [Data do Fechamento]

```

	Data do Fechamento	Faturamento Total (em milhões)	Primeiro valor	Último valor
1	2020-01-01 00:00:00.000	8	8	10
2	2020-02-01 00:00:00.000	10	8	10
3	2020-03-01 00:00:00.000	6	8	10
4	2020-04-01 00:00:00.000	9	8	10
5	2020-05-01 00:00:00.000	5	8	10
6	2020-06-01 00:00:00.000	4	8	10
7	2020-07-01 00:00:00.000	7	8	10
8	2020-08-01 00:00:00.000	11	8	10
9	2020-09-01 00:00:00.000	9	8	10
10	2020-10-01 00:00:00.000	12	8	10
11	2020-11-01 00:00:00.000	11	8	10
12	2020-12-01 00:00:00.000	10	8	10

00:00:00 | 12 linhas

Consulta executada com êxito.

# DESAFIO 1

Para resolver os exercícios 1 a 4, crie uma View chamada **vwProdutos**, que contenha o agrupamento das colunas **BrandName**, **ColorName** e os totais de quantidade vendida por marca/cor e também o total de receita por marca/cor.



# EXERCÍCIOS



1

## Questão 1

Utilize a View vwProdutos para criar uma coluna extra calculando a quantidade total vendida dos produtos.

2

## Questão 2

Crie mais uma coluna na consulta anterior, incluindo o total de produtos vendidos para cada marca.

3

## Questão 3

Calcule o % de participação do total de vendas de produtos por marca.

Ex: A marca A. Datum teve uma quantidade total de vendas de 199.041 de um total de 3.406.089 de vendas. Isso significa que a da marca A. Datum é  $199.041 / 3.406.089 = 5,84\%$ .

# EXERCÍCIOS

4

## Questão 4

Crie uma consulta à View `vwProdutos`, selecionando as colunas Marca, Cor, Quantidade\_Vendida e também criando uma coluna extra de Rank para descobrir a posição de cada Marca/Cor. Você deve obter o resultado abaixo. Obs: Sua consulta deve ser filtrada para que seja mostrada apenas a marca Contoso.



Marca	Cor	Quantidade_Por_Produto	Rank
Contoso	Black	239810	1
Contoso	White	230199	2
Contoso	Silver	149081	3
Contoso	Grey	133244	4
Contoso	Blue	52819	5
Contoso	Red	50168	6
Contoso	Pink	21709	7
Contoso	Brown	21014	8
Contoso	Green	19617	9
Contoso	Orange	10105	10
Contoso	Yellow	9295	11
Contoso	Gold	7502	12
Contoso	Silver Grey	6419	13
Contoso	Purple	1752	14
Contoso	Transparent	1655	15

# DESAFIO 2

Para responder os próximos 2 exercícios, você precisará criar uma View auxiliar.

A sua view deve se chamar vwHistoricoLojas e deve conter um histórico com a quantidade de lojas abertas a cada Ano/Mês (como mostrado na figura à direita). Os desafios são:

- (1) Criar uma coluna de ID para essa View
- (2) Relacionar as tabelas DimDate e DimStore

Dicas:

1- A coluna de ID será criada a partir de uma função de janela. Você deverá se atentar a forma como essa coluna deverá ser ordenada, pensando que queremos visualizar uma ordem de Ano/Mês que seja: 2005/january, 2005/February... e não 2005/April, 2005/August...

2- As colunas Ano, Mês e Qtd\_Lojas correspondem, respectivamente, às seguintes colunas: CalendarYear e CalendarMonthLabel da tabela DimDate e uma contagem da coluna OpenDate da tabela DimStore.

ID	Ano	Mês	Qtd_Lojas
1	2005	January	0
2	2005	February	0
3	2005	March	0
4	2005	April	0
5	2005	May	2
6	2005	June	4
7	2005	July	1
8	2005	August	0
9	2005	September	0
10	2005	October	0
11	2005	November	1
12	2005	December	0
13	2006	January	2
14	2006	February	3
15	2006	March	0
16	2006	April	1
17	2006	May	0
18	2006	June	0
19	2006	July	1
20	2006	August	1
21	2006	September	0
22	2006	October	0

# EXERCÍCIOS



5

6

## Questão 5

A partir da view criada no DESAFIO 2, você deverá fazer uma soma móvel considerando sempre o mês atual + 2 meses para trás.

## Questão 6

Utilize a **vwHistoricoLojas** para calcular o acumulado de lojas abertas a cada ano/mês.

# DESAFIO 3

Neste desafio, você terá que criar suas próprias tabelas e views para conseguir resolver os exercícios 7 e 8. Os próximos exercícios envolverão análises de novos clientes. Para isso, será necessário criar uma nova tabela e uma nova view.

Abaixo, temos um passo a passo para resolver o problema por partes.

**PASSO 1:** Crie um novo banco de dados chamado Desafio e selecione esse banco de dados criado.

**PASSO 2:** Crie uma tabela de datas entre o dia 1 de janeiro do ano com a compra (DateFirstPurchase) mais antiga e o dia 31 de dezembro do ano com a compra mais recente.

**Obs1:** Chame essa tabela de Calendario.

**Obs2:** A princípio, essa tabela deve conter apenas 1 coluna, chamada data e do tipo DATE.

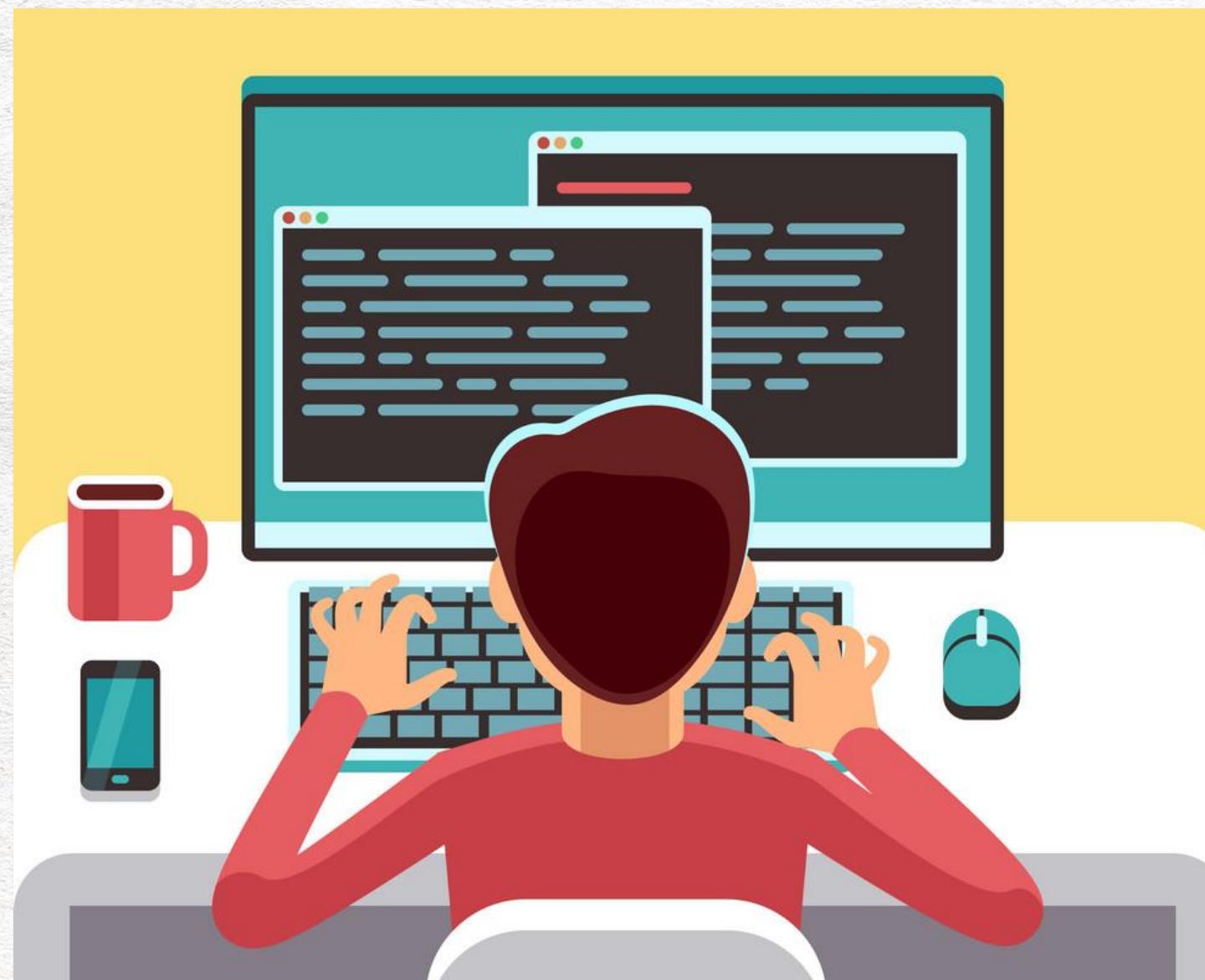
**PASSO 3:** Crie colunas auxiliares na tabela Calendario chamadas: Ano, Mes, Dia, AnoMes e NomeMes. Todas do tipo INT.

**PASSO 4:** Adicione na tabela os valores de Ano, Mês, Dia, AnoMes e NomeMes (nome do mês em português). Dica: utilize a instrução CASE para verificar o mês e retornar o nome certo.

**PASSO 5:** Crie a View vwNovosClientes, que deve ter as colunas mostradas abaixo.

ID	Ano	NomeMes	Novos_Clientes
1	2001	Janeiro	0
2	2001	Fevereiro	0
3	2001	Março	0
4	2001	Abril	0
5	2001	Maio	0
6	2001	Junho	0
7	2001	Julho	146
8	2001	Agosto	156
9	2001	Setembro	146
10	2001	Outubro	161
11	2001	Novembro	169
12	2001	Dezembro	235
13	2002	Janeiro	188
14	2002	Fevereiro	171
15	2002	Março	199
16	2002	Abril	207
17	2002	Maio	214
18	2002	Junho	214
19	2002	Julho	253
20	2002	Agosto	281
21	2002	Setembro	198
22	2002	Outubro	229
23	2002	Novembro	193
24	2002	Dezembro	330
25	2003	Janeiro	244
26	2003	Fevereiro	272
27	2003	Março	272
28	2003	Abril	294
29	2003	Maio	335
30	2003	Junho	321
31	2003	Julho	202
32	2003	Agosto	1210
33	2003	Setembro	1112

# EXERCÍCIOS



7

## Questão 7

Faça um cálculo de soma móvel de novos clientes nos últimos 2 meses.

b) Faça um cálculo de média móvel de novos clientes nos últimos 2 meses.

c) Faça um cálculo de acumulado dos novos clientes ao longo do tempo.

d) Faça um cálculo de acumulado intra-ano, ou seja, um acumulado que vai de janeiro a dezembro de cada ano, e volta a fazer o cálculo de acumulado no ano seguinte.

8

## Questão 8

Faça os cálculos de MoM e YoY para avaliar o percentual de crescimento de novos clientes, entre o mês atual e o mês anterior, e entre um mês atual e o mesmo mês do ano anterior.

# DESAFIO 1

Para resolver os exercícios 1 a 4, crie uma View chamada **vwProdutos**, que contenha o agrupamento das colunas **BrandName**, **ColorName** e os totais de quantidade vendida por marca/cor e também o total de receita por marca/cor.

```
SQLQuery1.sql - M...MARCUS\caval (53)* ✎ X
CREATE VIEW vwProdutos AS
SELECT
    BrandName AS 'Marca',
    ColorName AS 'Cor',
    COUNT(*) AS 'Quantidade_Vendida',
    ROUND(SUM(SalesAmount), 2) AS 'Receita_Total'
FROM DimProduct
INNER JOIN FactSales
    ON DimProduct.ProductKey = FactSales.ProductKey
GROUP BY BrandName, ColorName
```

# GABARITOS

1

## Questão 1

Utilize a View vwProdutos para criar uma coluna extra calculando a quantidade total vendida dos produtos

The screenshot shows a SQL query window titled "SQLQuery1.sql - M...MARCUS\caval (53)\*". The query is:

```
SELECT
    *,
    SUM(Quantidade_Vendida) OVER() AS 'Qtd Total Produtos'
FROM vwProdutos
ORDER BY Marca
```

The results grid has four columns: Marca, Cor, Quantidade\_Vendida, and Receita\_Total. A fifth column, "Qtd Total Produtos", is added using the window function SUM(). The results show 11 rows of data.

	Marca	Cor	Quantidade_Vendida	Receita_Total	Qtd Total Produtos
1	A. Datum	Blue	5601	28597778,95	3406089
2	A. Datum	Gold	4221	24740930,28	3406089
3	A. Datum	Azure	20837	51328936,53	3406089
4	A. Datum	Silver	25036	72671174,86	3406089
5	A. Datum	Green	21350	52020952,44	3406089
6	A. Datum	Grey	24279	79225719,70	3406089
7	A. Datum	Black	28253	90942577,13	3406089
8	A. Datum	Silv...	9990	47717252,60	3406089
9	A. Datum	Ora...	30637	93022209,61	3406089
10	A. Datum	Pink	28837	79536221,46	3406089
11	Advent	Brown	16812	130429712	3406089

Consulta executada com êxito.

MARCUS (15.0 RTM) | MARCUS\caval (53) | ContosoRetailDW | 00:00:00 | 111 linhas

# GABARITOS

2

## Questão 2

Crie mais uma coluna na consulta anterior, incluindo o total de produtos vendidos para cada marca.

The screenshot shows a SQL Server Management Studio window. The query pane contains the following T-SQL code:

```
SQLQuery1.sql - M...MARCUS\caval (53)*
SELECT
    *,
    SUM(Quantidade_Vendida) OVER() AS 'Qtd Total Produtos',
    SUM(Quantidade_Vendida) OVER(PARTITION BY Marca) AS 'Qtd Total Por Marca'
FROM vwProdutos
ORDER BY Marca
```

The results pane displays a table with the following data:

	Marca	Cor	Quantidade_Vendida	Receita_Total	Qtd Total Produtos	Qtd Total Por Marca
1	A. Datum	Blue	5601	28597778,95	3406089	199041
2	A. Datum	Gold	4221	24740930,28	3406089	199041
3	A. Datum	Azure	20837	51328936,53	3406089	199041
4	A. Datum	Silver	25036	72671174,86	3406089	199041
5	A. Datum	Green	21350	52020952,44	3406089	199041
6	A. Datum	Grey	24279	79225719,70	3406089	199041
7	A. Datum	Black	28253	90942577,13	3406089	199041
8	A. Datum	Silver Grey	9990	47717252,60	3406089	199041
9	A. Datum	Orange	30637	93022209,61	3406089	199041
10	A. Datum	Pink	28837	79536221,46	3406089	199041
11	Adventure Works	Brown	16812	130429712,63	3406089	262788

At the bottom of the results grid, there is a message: "Consulta executada com êxito." (Query executed successfully.)

# GABARITOS

3

## Questão 3

Calcule o % de participação do total de vendas de produtos por marca.

Ex: A marca A. Datum teve uma quantidade total de vendas de 199.041 de um total de 3.406.089 de vendas. Isso significa que a da marca A. Datum é  $199.041/3.406.089 = 5,84\%$ .

The screenshot shows a SQL Server Management Studio (SSMS) interface. The top window is titled "SQLQuery1.sql - M...MARCUS\caval (53)\*". It contains the following T-SQL code:

```

SELECT
    *,
    SUM(Quantidade_Vendida) OVER() AS 'Qtd1',
    SUM(Quantidade_Vendida) OVER(PARTITION BY Marca) AS 'Qtd2',
    FORMAT(1.0*(SUM(Quantidade_Vendida) OVER(PARTITION BY Marca))/SUM
        (Quantidade_Vendida) OVER(), '0.00%')
FROM vwProdutos
ORDER BY Marca

```

The bottom window is titled "Resultados" and displays a table with the following data:

	Marca	Cor	Quantidade_Vendida	Receita_Total	Qtd1	Qtd2	(Nenhum nome de coluna)
1	A. Datum	Blue	5601	28597778,95	3406089	199041	5,84%
2	A. Datum	Gold	4221	24740930,28	3406089	199041	5,84%
3	A. Datum	Azure	20837	51328936,53	3406089	199041	5,84%
4	A. Datum	Silver	25036	72671174,86	3406089	199041	5,84%
5	A. Datum	Green	21350	52020952,44	3406089	199041	5,84%
6	A. Datum	Grey	24279	79225719,70	3406089	199041	5,84%
7	A. Datum	Black	28253	90942577,13	3406089	199041	5,84%
8	A. Datum	Silver Grey	9990	47717252,60	3406089	199041	5,84%
9	A. Datum	Orange	30637	93022209,61	3406089	199041	5,84%
10	A. Datum	Pink	28837	79536221,46	3406089	199041	5,84%
11	Adventure Works	Brown	16812	130429712,63	3406089	262788	7,72%

The status bar at the bottom of the SSMS window indicates: "Consulta executada com êxito." and "MARCUS (15.0 RTM) | MARCUS\caval (53) | ContosoRetailDW | 00:00:01 | 111 linhas".

# GABARITOS

4

## Questão 4

Crie uma consulta à View **vwProdutos**, selecionando as colunas Marca, Cor, Quantidade\_Vendida e também criando uma coluna extra de Rank para descobrir a posição de cada Marca/Cor. Você deve obter o resultado abaixo. Obs: Sua consulta deve ser filtrada para que seja mostrada apenas a marca Contoso.

The screenshot shows a SQL query in the SQL Query window and its execution results in the Results window.

**SQL Query Window:**

```
SQLQuery1.sql - M...MARCUS\caval (53)*  X
SELECT
    *,
    RANK() OVER(ORDER BY Quantidade_Vendida DESC) AS 'Rank'
FROM vwProdutos
WHERE Marca = 'Contoso'
```

**Results Window:**

	Marca	Cor	Quantidade_Vendida	Receita_Total	Rank
1	Contoso	Black	239810	483707723,01	1
2	Contoso	White	230199	565737357,75	2
3	Contoso	Silver	149081	556673579,19	3
4	Contoso	Grey	133244	231982195,59	4
5	Contoso	Blue	52819	222594418,98	5
6	Contoso	Red	50168	215977283,56	6
7	Contoso	Pink	21709	105939917,97	7
8	Contoso	Brown	21014	98945737,80	8
9	Contoso	Green	19617	141962803,52	9
10	Contoso	Orange	10105	39367509,98	10
11	Contoso	Yellow	9295	11896070,35	11

Consulta executada com êxito. | MARCUS (15.0 RTM) | MARCUS\caval (53) | ContosoRetailDW | 00:00:00 | 15 linhas

# DESAFIO 2

Para responder os próximos 2 exercícios, você precisará criar uma View auxiliar.

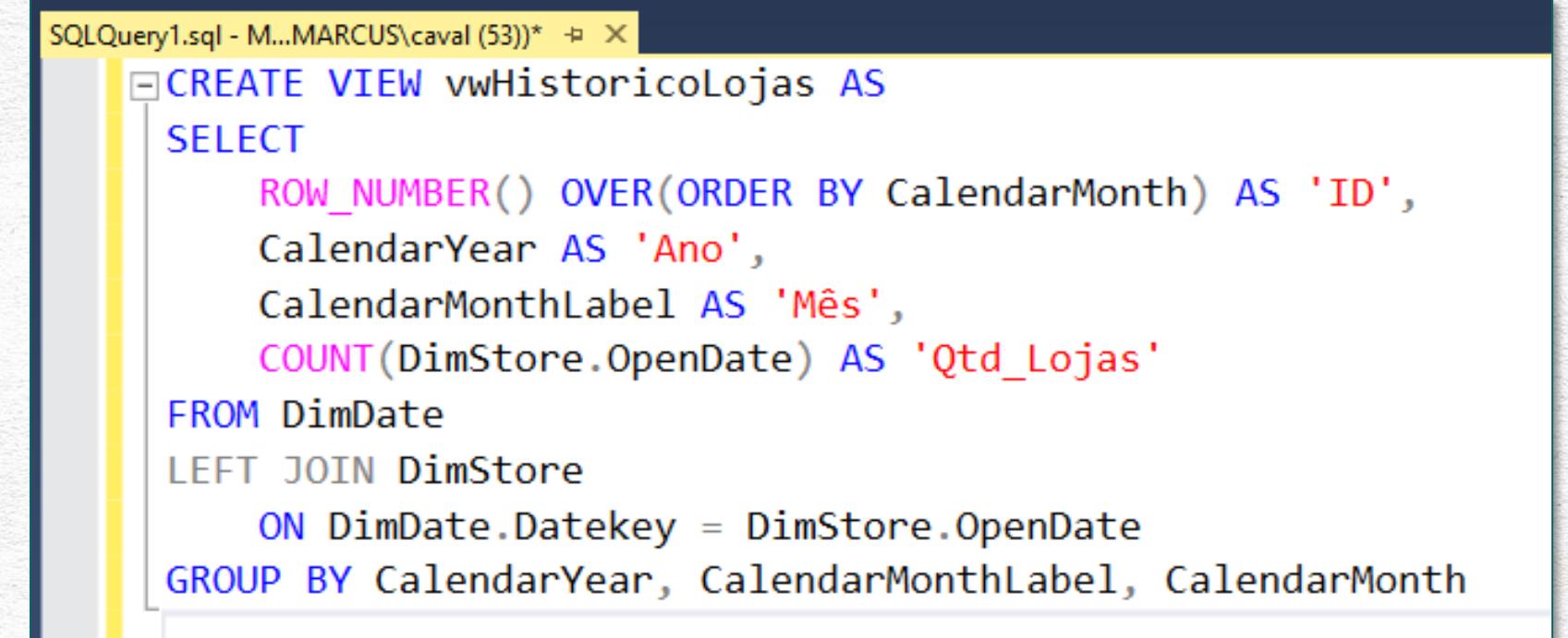
A sua view deve se chamar vwHistoricoLojas e deve conter um histórico com a quantidade de lojas abertas a cada Ano/Mês (como mostrado na figura à direita). Os desafios são:

- (1) Criar uma coluna de ID para essa View
- (2) Relacionar as tabelas DimDate e DimStore

Dicas:

1- A coluna de ID será criada a partir de uma função de janela. Você deverá se atentar a forma como essa coluna deverá ser ordenada, pensando que queremos visualizar uma ordem de Ano/Mês que seja: 2005/january, 2005/February... e não 2005/April, 2005/August...

2- As colunas Ano, Mês e Qtd\_Lojas correspondem, respectivamente, às seguintes colunas: CalendarYear e CalendarMonthLabel da tabela DimDate e uma contagem da coluna OpenDate da tabela DimStore.



```
SQLQuery1.sql - M...MARCUS\caval (53)* ↗ X
CREATE VIEW vwHistoricoLojas AS
SELECT
    ROW_NUMBER() OVER(ORDER BY CalendarMonth) AS 'ID',
    CalendarYear AS 'Ano',
    CalendarMonthLabel AS 'Mês',
    COUNT(DimStore.OpenDate) AS 'Qtd_Lojas'
FROM DimDate
LEFT JOIN DimStore
    ON DimDate.Datekey = DimStore.OpenDate
GROUP BY CalendarYear, CalendarMonthLabel, CalendarMonth
```

# GABARITOS

5

## Questão 5

A partir da view criada no DESAFIO 2, você deverá fazer uma soma móvel considerando sempre o mês atual + 2 meses para trás.

The screenshot shows a SQL Server Management Studio (SSMS) window titled "SQLQuery1.sql - M...MARCUS\caval (53)\*". The query is:

```
SELECT
    *,
    SUM(Qtd_Lojas) OVER(ORDER BY ID ROWS BETWEEN 2 PRECEDING AND CURRENT ROW)
FROM vwHistoricoLojas
```

The results grid displays data from the "vwHistoricoLojas" view. The columns are: ID, Ano, Mês, Qtd\_Lojas, and (Nenhum nome de coluna). The data is as follows:

ID	Ano	Mês	Qtd_Lojas	(Nenhum nome de coluna)
1	2005	January	0	0
2	2005	February	0	0
3	2005	March	0	0
4	2005	April	0	0
5	2005	May	2	2
6	2005	June	4	6
7	2005	July	1	7
8	2005	August	0	5
9	2005	September	0	1
10	2005	October	0	0
11	2005	November	1	1

At the bottom of the results grid, a message says "Consulta executada com êxito." (Query executed successfully).

# GABARITOS

6

## Questão 6

Utilize a **vwHistoricoLojas** para calcular o acumulado de lojas abertas a cada ano/mês.

```
SQLQuery1.sql - M...MARCUS\caval (53)*  X
SELECT
    *,
    SUM(Qtd_Lojas) OVER(ORDER BY ID ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
FROM vwHistoricoLojas
```

The screenshot shows a SQL query window in SSMS. The query uses a window function to calculate the cumulative sum of stores opened ('Qtd\_Lojas') over time ('ID'). The result set shows the ID, year ('Ano'), month ('Mês'), and the total number of stores opened up to that point ('Qtd\_Lojas' and '(Nenhum nome de coluna)').

ID	Ano	Mês	Qtd_Lojas	(Nenhum nome de coluna)
1	2005	January	0	0
2	2005	February	0	0
3	2005	March	0	0
4	2005	April	0	0
5	2005	May	2	2
6	2005	June	4	6
7	2005	July	1	7
8	2005	August	0	7
9	2005	September	0	7
10	2005	October	0	7
11	2005	November	1	8

# DESAFIO 3

Neste desafio, você terá que criar suas próprias tabelas e views para conseguir resolver os exercícios 7 e 8. Os próximos exercícios envolverão análises de novos clientes. Para isso, será necessário criar uma nova tabela e uma nova view.

Abaixo, temos um passo a passo para resolver o problema por partes.

**PASSO 1:** Crie um novo banco de dados chamado Desafio e selecione esse banco de dados criado.

**PASSO 2:** Crie uma tabela de datas entre o dia 1 de janeiro do ano com a compra (DateFirstPurchase) mais antiga e o dia 31 de dezembro do ano com a compra mais recente.

**Obs1:** Chame essa tabela de Calendario.

**Obs2:** A princípio, essa tabela deve conter apenas 1 coluna, chamada data e do tipo DATE.

**CONTINUA NA PRÓXIMA PÁGINA**

```

SQLQuery1.sql - M...MARCUS\caval (53)* 120 %
--1) Passo 1: Criar o Banco de Dados Desafio
CREATE DATABASE Desafio
USE Desafio

--2) Passo 2: Criar uma tabela de datas entre o dia 1 de janeiro do ano com a primeira compra de cliente
mais antiga até o dia 31 de dezembro do ano com a última compra

CREATE TABLE Calendario (
    data DATE
)

DECLARE @varAnoInicial INT = YEAR((SELECT MIN(DateFirstPurchase) FROM ContosoRetailDW.dbo.DimCustomer))
DECLARE @varAnoFinal INT = YEAR((SELECT MAX(DateFirstPurchase) FROM ContosoRetailDW.dbo.DimCustomer))

DECLARE @varDataInicial DATE = DATEFROMPARTS(@varAnoInicial, 1, 1)
DECLARE @varDataFinal DATE = DATEFROMPARTS(@varAnoFinal, 12, 31)

WHILE @varDataInicial <= @varDataFinal
BEGIN
    INSERT INTO Calendario(data) VALUES(@varDataInicial)
    SET @varDataInicial = DATEADD(DAY, 1, @varDataInicial)
END

SELECT * FROM Calendario

```

# DESAFIO 3

Neste desafio, você terá que criar suas próprias tabelas e views para conseguir resolver os exercícios 7 e 8. Os próximos exercícios envolverão análises de novos clientes. Para isso, será necessário criar uma nova tabela e uma nova view.

Abaixo, temos um passo a passo para resolver o problema por partes.

**PASSO 3:** Crie colunas auxiliares na tabela Calendario chamadas: Ano, Mes, Dia, AnoMes e NomeMes. Todas do tipo INT.

**PASSO 4:** Adicione na tabela os valores de Ano, Mês, Dia, AnoMes e NomeMes (nome do mês em português). Dica: utilize a instrução CASE para verificar o mês e retornar o nome certo.

CONTINUA NA PRÓXIMA PÁGINA

```
--3) Passo 3: Criar colunas auxiliares na tabela Calendario
ALTER TABLE Calendario
ADD Ano INT,
    Mes INT,
    Dia INT,
    AnoMes INT,
    NomeMes VARCHAR(50)

--4) Passo 4: Adicionar os valores às colunas
UPDATE Calendario SET Ano = YEAR(data)
UPDATE Calendario SET Mes = MONTH(data)
UPDATE Calendario SET Dia = DAY(data)
UPDATE Calendario SET AnoMes = CONCAT(YEAR(data), FORMAT(MONTH(data), '00'))
UPDATE Calendario SET NomeMes =
CASE
    WHEN MONTH(data) = 1 THEN 'Janeiro'
    WHEN MONTH(data) = 2 THEN 'Fevereiro'
    WHEN MONTH(data) = 3 THEN 'Março'
    WHEN MONTH(data) = 4 THEN 'Abril'
    WHEN MONTH(data) = 5 THEN 'Maio'
    WHEN MONTH(data) = 6 THEN 'Junho'
    WHEN MONTH(data) = 7 THEN 'Julho'
    WHEN MONTH(data) = 8 THEN 'Agosto'
    WHEN MONTH(data) = 9 THEN 'Setembro'
    WHEN MONTH(data) = 10 THEN 'Outubro'
    WHEN MONTH(data) = 11 THEN 'Novembro'
    WHEN MONTH(data) = 12 THEN 'Dezembro'
END
```

# DESAFIO 3

Neste desafio, você terá que criar suas próprias tabelas e views para conseguir resolver os exercícios 7 e 8. Os próximos exercícios envolverão análises de novos clientes. Para isso, será necessário criar uma nova tabela e uma nova view.

Abaixo, temos um passo a passo para resolver o problema por partes.

**PASSO 5:** Crie a View vwNovosClientes, que deve ter as colunas mostradas abaixo.

The screenshot shows a SQL query window titled "SQLQuery1.sql - M...MARCUS\caval (53)\*". The query is as follows:

```
--5) Passo 5: Crie uma View
CREATE VIEW vwNovosClientes AS
SELECT
    ROW_NUMBER() OVER(ORDER BY AnoMes) AS 'ID',
    Ano,
    NomeMes,
    COUNT(DimCustomer.DateFirstPurchase) AS 'Novos_Clientes'
FROM Calendario
LEFT JOIN ContosoRetailDW.dbo.DimCustomer
    ON Calendario.data = DimCustomer.DateFirstPurchase
GROUP BY Ano, NomeMes, AnoMes

SELECT * FROM vwNovosClientes
```

The results pane shows a table with the following data:

ID	Ano	NomeMes	Novos_Clientes
1	2001	Janeiro	0
2	2001	Fevereiro	0
3	2001	Março	0
4	2001	Abril	0
5	2001	Maio	0
6	2001	Junho	0
7	2001	Julho	146
8	2001	Agosto	156

At the bottom of the results pane, a message says "Consulta executada com êxito." (Query executed successfully).

# GABARITOS

7

## Questão 7

- Faça um cálculo de soma móvel de novos clientes nos últimos 2 meses.
- Faça um cálculo de média móvel de novos clientes nos últimos 2 meses.
  - Faça um cálculo de acumulado dos novos clientes ao longo do tempo.
  - Faça um cálculo de acumulado intra-ano, ou seja, um acumulado que vai de janeiro a dezembro de cada ano, e volta a fazer o cálculo de acumulado no ano seguinte.

SQLQuery1.sql - M...MARCUS\caval (53)\*

```

SELECT
    *,
    SUM(Novos_Clientes) OVER(ORDER BY ID ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS 'Soma
        Móvel (2 meses)',
    AVG(Novos_Clientes) OVER(ORDER BY ID ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS 'Média
        Móvel (2 meses)',
    SUM(Novos_Clientes) OVER(ORDER BY ID ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS
        'Acumulado Total',
    SUM(Novos_Clientes) OVER(PARTITION BY Ano ORDER BY ID ROWS BETWEEN UNBOUNDED PRECEDING AND
        CURRENT ROW) AS 'Acumulado (YTD)'
FROM vwNovosClientes

```

Resultados Mensagens

ID	Ano	NomeMes	Novos_Clientes	Soma Móvel (2 meses)	Média Móvel (2 meses)	Acumulado Total	Acumulado (YTD)
1	1	Janeiro	0	0	0	0	0
2	2	Fevereiro	0	0	0	0	0
3	3	Março	0	0	0	0	0
4	4	Abril	0	0	0	0	0
5	5	Maio	0	0	0	0	0
6	6	Junho	0	0	0	0	0
7	7	Julho	146	146	48	146	146
8	8	Agosto	156	302	100	302	302
9	9	Setembro	146	448	149	448	448
10	10	Outubro	161	463	154	609	609
11	11	Novembro	169	476	158	778	778
12	12	Dezembro	235	565	188	1013	1013
13	13	Janeiro	188	592	197	1201	188
14	14	Fevereiro	171	594	198	1372	359

Consulta executada com êxito.

MARCUS (15.0 RTM) | MARCUS\caval (53) | Desafio | 00:00:00 | 48 linhas

# GABARITOS

8

## Questão 8

Faça os cálculos de MoM e YoY para avaliar o percentual de crescimento de novos clientes, entre o mês atual e o mês anterior, e entre um mês atual e o mesmo mês do ano anterior.

SQLQuery1.sql - M...MARCUS\caval (53)\*

```

SELECT
    *,
    FORMAT(CONVERT(FLOAT, Novos_Clientes)/NULLIF(LAG(Novos_Clientes, 1) OVER(ORDER BY ID), 0) - 1, '0.00%') AS '% MoM',
    FORMAT(CONVERT(FLOAT, Novos_Clientes)/NULLIF(LAG(Novos_Clientes, 12) OVER(ORDER BY ID), 0) - 1, '0.00%') AS '% YoY'
FROM vwNovosClientes

```

Resultados

ID	Ano	NomeMes	Novos_Clientes	% MoM	% YoY
1	2001	Janeiro	0	NULL	NULL
2	2001	Fevereiro	0	NULL	NULL
3	2001	Março	0	NULL	NULL
4	2001	Abril	0	NULL	NULL
5	2001	Maio	0	NULL	NULL
6	2001	Junho	0	NULL	NULL
7	2001	Julho	146	NULL	NULL
8	2001	Agosto	156	6,85%	NULL
9	2001	Setembro	146	-6,41%	NULL
10	2001	Outubro	161	10,27%	NULL
11	2001	Novembro	169	4,97%	NULL
12	2001	Dezembro	235	39,05%	NULL
13	2002	Janeiro	188	-20,00%	NULL
14	2002	Fevereiro	171	-9,04%	NULL
15	2002	Março	199	16,37%	NULL
16	2002	Abril	207	4,02%	NULL
17	2002	Maio	214	3,38%	NULL

Consulta executada com êxito.

MARCUS (15.0 RTM) | MARCUS\caval (53) | Desafio | 00:00:00 | 48 linhas

# SQL REGULAR EXPRESSIONS



# Regular Expressions - Introdução

## O QUE É E PARA QUE SERVEM?

- A expressão regular (conhecida como regex ou regexp, do inglês *regular expression*) permite uma forma de identificar cadeias de caracteres de interesse, como caracteres específicos, palavras ou padrões de caracteres.
- Em resumo, a expressão regular é uma forma de permitir realizar, de forma simples, operações bastante complexas com strings, que possivelmente exigiriam várias condições para tratar cada caso.
- O SQL Server já possui uma opção para tratar casos especiais de textos, por meio do comando LIKE, que já vimos anteriormente. Porém, neste módulo vamos ver diversas aplicações mais avançadas.

SQLQuery11.sql - LA...us Cavalcanti (52)\*

```

SELECT
    *
FROM DimProduct
WHERE ProductDescription LIKE '%Touchpad%'

```

**Relembrando o LIKE...**

o LIKE é um comando especial que nos permite filtrar textos de forma mais personalizada. No exemplo acima, estamos filtrando todas as descrições de produto que contenham o texto Touchpad.

ProductKey	ProductLabel	ProductName	ProductDescription	ProductSubcategoryKey	Manufacturer	BrandName
16	0101016	Contoso 8GB Super-Slim MP3/Video Player M800 White	2" color LCD, Touchpad, Plays music, video, phot...	1	Contoso, Ltd	Contoso
17	0101017	Contoso 8GB Super-Slim MP3/Video Player M800 Red	2" color LCD, Touchpad, Plays music, video, phot...	1	Contoso, Ltd	Contoso
18	0101018	Contoso 8GB Super-Slim MP3/Video Player M800 Green	2" color LCD, Touchpad, Plays music, video, phot...	1	Contoso, Ltd	Contoso
19	0101019	Contoso 8GB Super-Slim MP3/Video Player M800 Pink	2" color LCD, Touchpad, Plays music, video, phot...	1	Contoso, Ltd	Contoso

# COLLATE - Introdução

COLLATION é um conjunto de regras que informam ao mecanismo de banco de dados como comparar e classificar os dados de caracteres no SQL Server.

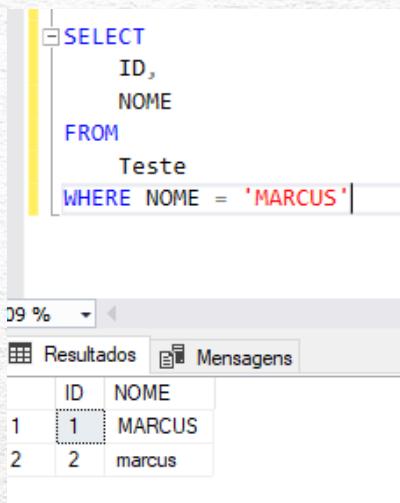
Em resumo, ele serve para indicar se um determinado campo será CASE INSENSITIVE e também a forma como interpreta a acentuação das palavras. Observe os exemplos abaixo.

1

ID	NOME
1	MARCUS
2	marcus
3	André
4	andre

Temos uma tabela com dois nomes: Marcus e André. Só que cada nome é escrito de uma forma diferente: MARCUS e marcus, André e andre. Como será que filtros de texto se comportam com essas diferenças?

2

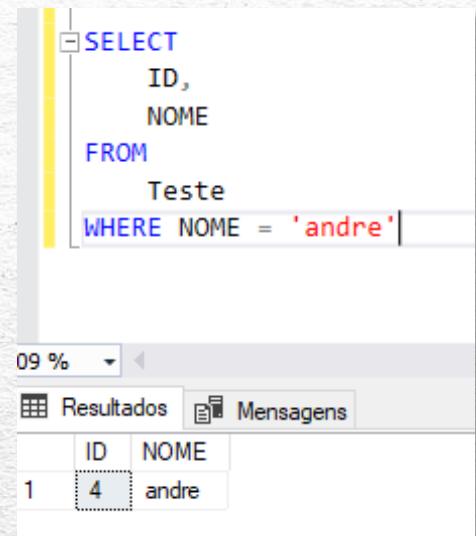


```
SELECT
    ID,
    NOME
FROM
    Teste
WHERE NOME = 'MARCUS'
```

ID	NOME
1	MARCUS
2	marcus

Filtrando o nome 'MARCUS', temos as duas linhas com esse nome como resultado, mesmo que uma delas esteja escrita em letras minúsculas.

3



```
SELECT
    ID,
    NOME
FROM
    Teste
WHERE NOME = 'andre'
```

ID	NOME
4	andre

Filtrando 'andre', apenas 1 linha é retornada, pois o outro nome está escrito com acento.



# COLLATE - Introdução

Quando usamos o COLLATE, conseguimos configurar duas coisas na hora de criar os filtros de textos:

- (i) Se haverá ou não diferenciação entre letras maiúsculas e minúsculas.
- (ii) Se haverá ou não diferenciação entre letras acentuadas e não acentuadas.

Identifica se a configuração será “accent insensitive” ou “accent sensitive”:

- AI: Accent Insensitive (não diferencia palavras acentuadas de não acentuadas)
- AS: Accent Sensitive (diferencia palavras acentuadas de não acentuadas)

COLLATE Latin1\_General\_CI\_AI

Identifica se a configuração será “case insensitive” ou “case sensitive”:

- CI: Case Insensitive (não diferencia maiúsculas de minúsculas)
- CS: Case Sensitive (diferencia maiúsculas de minúsculas)

# COLLATE - Introdução

414

Observe agora uma nova situação, dessa vez usando o COLLATE.

1

ID	NOME
1	MARCUS
2	marcus
3	André
4	andre

Temos a mesma tabela com dois nomes: Marcus e André.

2

```
SELECT
    ID,
    NOME
FROM
    Teste
WHERE NOME COLLATE Latin1_General_CS_AI = 'Marcus'
```

Filtrando o nome 'Marcus', ele não retorna nenhuma linha, pois nenhuma está escrita exatamente dessa forma (considerando maiúsculas e minúsculas).

3

```
SELECT
    ID,
    NOME
FROM
    Teste
WHERE NOME COLLATE Latin1_General_CI_AI = 'Andre'
```

Filtrando 'Andre', as duas linhas são retornadas, mesmo que em uma delas tenhamos o nome com acento.

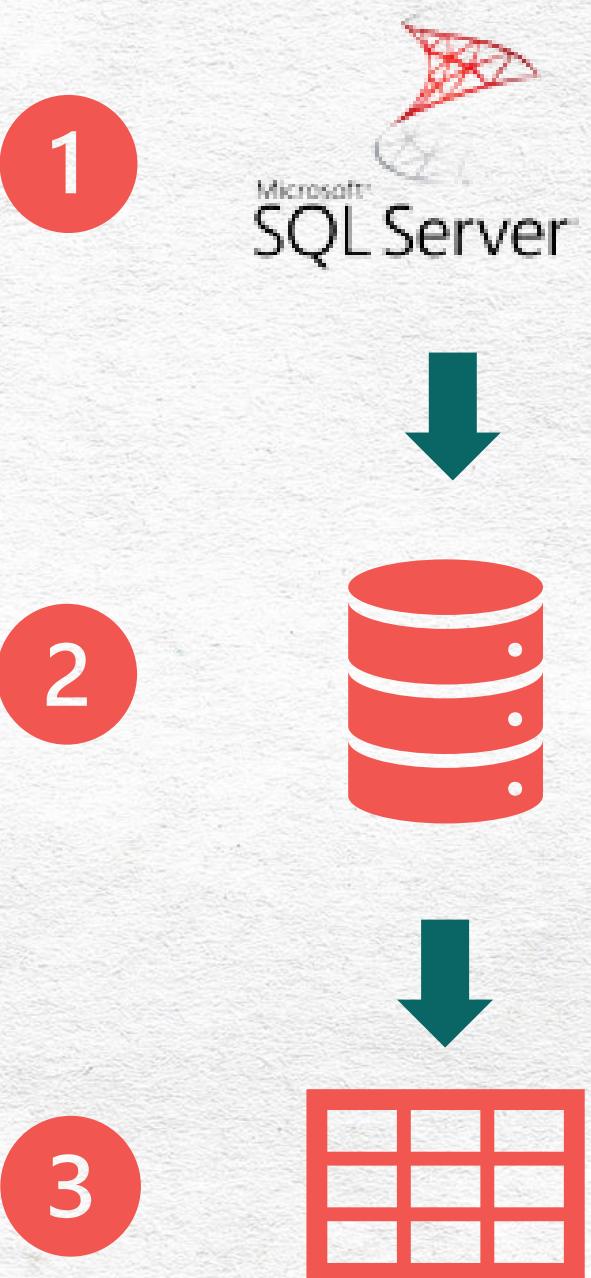


# Um pouco mais sobre COLLATE

O COLLATION nos permite configurar se teremos diferenciação entre maiúsculas e minúsculas, ou entre palavras acentuadas.

Ele pode ser definido em níveis diferentes no SQL Server. Abaixo estão os três níveis:

1. A nível de programa (SQL Server)
2. A nível de Bancos de Dados
3. A nível de tabelas/colunas



# Um pouco mais sobre COLLATE

## 1. A NÍVEL DE SQL SERVER

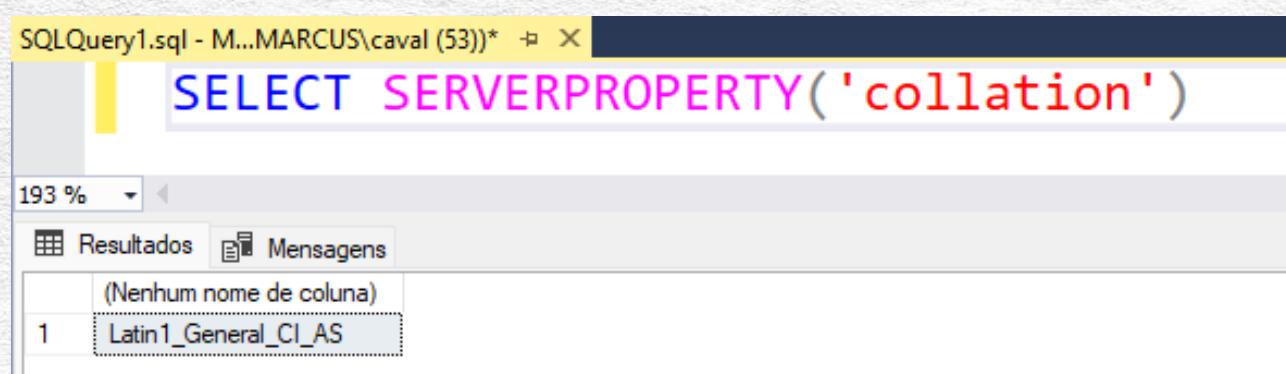
O **COLLATION** a princípio é definido durante a instalação do programa.

Por padrão, o **COLLATION** padrão é o seguinte:

Latin1\_General\_CI\_AS

Onde CI significa Case Insensitive (não diferencia maiúsculas de minúsculas) e AS significa Accent Sensitive (sensível ao sotaque).

Para descobrir o **COLLATION** configurado, podemos utilizar o comando abaixo:



```
SQLQuery1.sql - M...MARCUS\caval (53)*  SELECT SERVERPROPERTY('collation')
193 %  Resultados  Mensagens
(Nenhum nome de coluna)
1 Latin1_General_CI_AS
```

The screenshot shows a SQL Server Management Studio (SSMS) window with a query titled "SQLQuery1.sql". The query is: "SELECT SERVERPROPERTY('collation')". The results pane shows one row with the value "Latin1\_General\_CI\_AS". The "Resultados" tab is selected.

# Um pouco mais sobre COLLATE

## 2. A NÍVEL DE BANCO DE DADOS

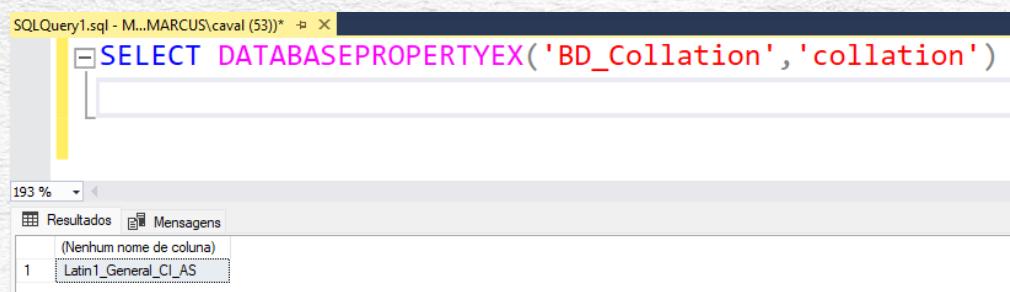
Por padrão, todos os bancos de dados vão herdar a configuração do COLLATION feita durante a instalação do SQL Server. Nós podemos também especificar o COLLATION do Banco de Dados no momento da sua criação.

```
CREATE DATABASE BD_Collation  
COLLATE Latin1_General_CS_AS
```

Para alterar o COLLATE após criar um banco de dados, usamos o comando abaixo:\*/

```
ALTER DATABASE BD_Collation  
COLLATE Latin1_General_CI_AS
```

Para saber o COLLATION de um Banco de Dados específico, podemos usar o comando abaixo:



```
SQLQuery1.sql - M...MARCUS\caval (53)* 193 %  
SELECT DATABASEPROPERTYEX('BD_Collation', 'collation')  
  
Resultados Mensagens  
(Nenhum nome de coluna)  
1 Latin1_General_CI_AS
```



# Um pouco mais sobre COLLATE

## 3. A NÍVEL DE TABELA/COLUNA

Por padrão, uma nova coluna de tipo VARCHAR herda o COLLATION do banco de dados, a menos que você especifique o COLLATION explicitamente ao criar a tabela.

Para criar uma coluna com um COLLATION diferente, você pode especificar usando o comando Collate SQL.

```
CREATE TABLE Nomes(
    ID INT,
    Nome1 VARCHAR(100),
    Nome2 VARCHAR(100) COLLATE Latin1_General_CS_AS)
```

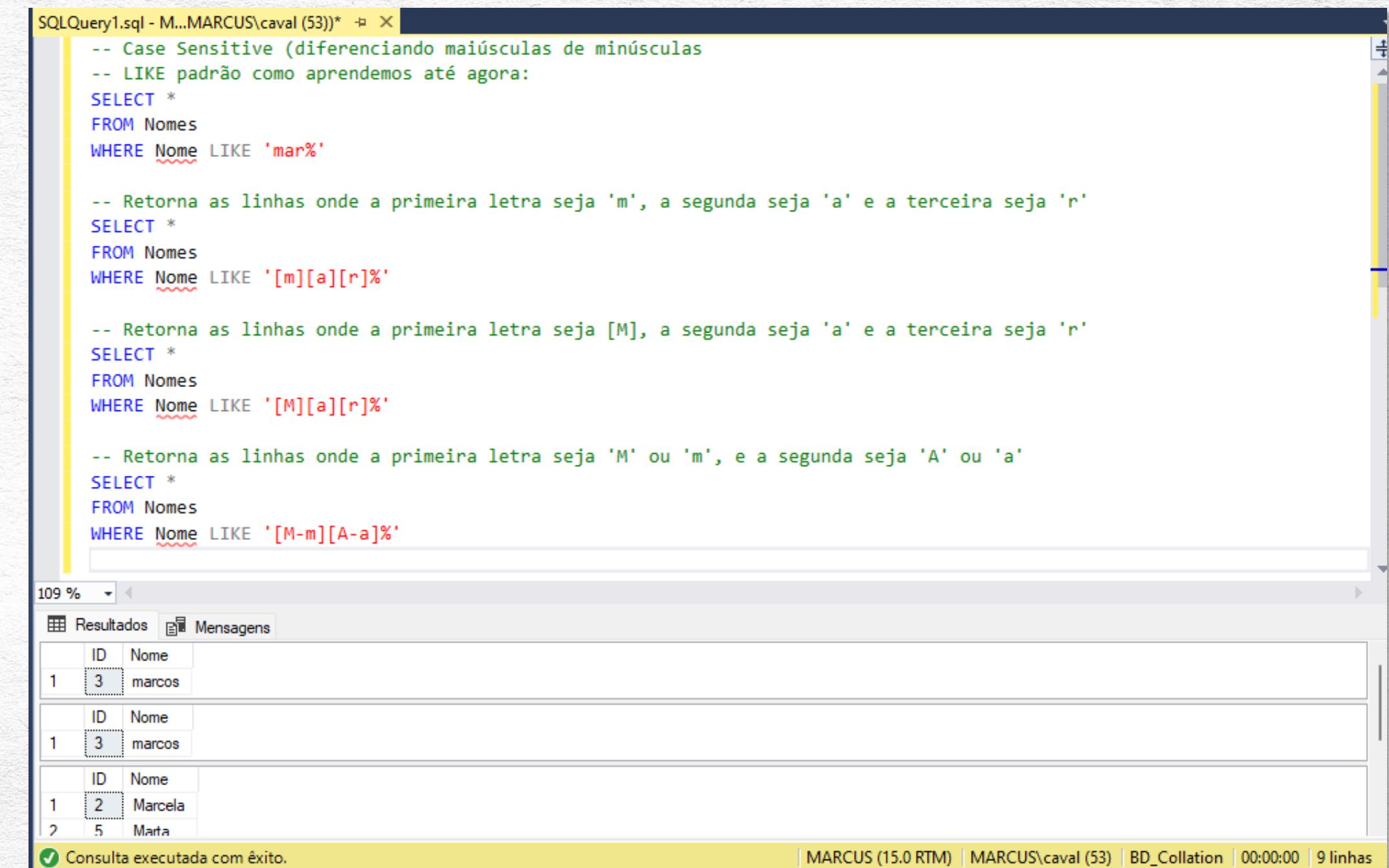
Podemos ver o COLLATION de cada coluna da tabela usando o comando abaixo:

```
sp_help Nomes
```

# LIKE – Case Sensitive

No exemplo ao lado, temos vários exemplos de aplicação do LIKE e suas respectivas descrições.

Como podemos ver na imagem ao lado, podemos usar colchetes como forma de criar filtros ainda mais personalizados.



```

SQLQuery1.sql - M...MARCUS\caval (53)* ✎ X
-- Case Sensitive (diferenciando maiúsculas de minúsculas
-- LIKE padrão como aprendemos até agora:
SELECT *
FROM Nomes
WHERE Nome LIKE 'mar%'

-- Retorna as linhas onde a primeira letra seja 'm', a segunda seja 'a' e a terceira seja 'r'
SELECT *
FROM Nomes
WHERE Nome LIKE '[m][a][r]%' 

-- Retorna as linhas onde a primeira letra seja [M], a segunda seja 'a' e a terceira seja 'r'
SELECT *
FROM Nomes
WHERE Nome LIKE '[M][a][r]%' 

-- Retorna as linhas onde a primeira letra seja 'M' ou 'm', e a segunda seja 'A' ou 'a'
SELECT *
FROM Nomes
WHERE Nome LIKE '[M-m][A-a]%' 

```

Resultados

ID	Nome
1	marcos

ID	Nome
1	marcos

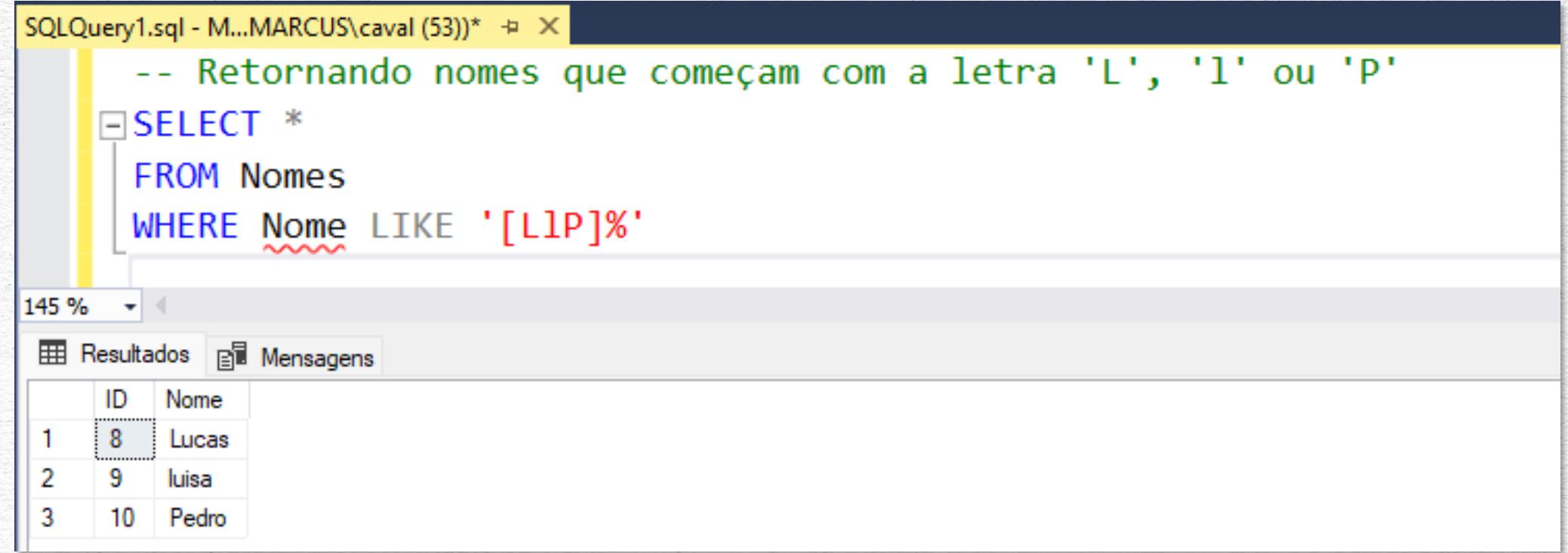
ID	Nome
1	Marcela
2	Marta

Consulta executada com êxito.

MARCUS (15.0 RTM) | MARCUS\caval (53) | BD\_Collation | 00:00:00 | 9 linhas

# LIKE – Filtrando os primeiros caracteres

No próximo exemplo vemos como é possível filtrar uma coluna a partir do primeiro caractere da palavra.



The screenshot shows a SQL query window titled "SQLQuery1.sql - M...MARCUS\caval (53)\*". The query is:

```
-- Retornando nomes que começam com a letra 'L', 'I' ou 'P'  
SELECT *  
FROM Nomes  
WHERE Nome LIKE '[LIP]%'
```

The results pane shows a table with columns "ID" and "Nome". The data is:

ID	Nome
1	8 Lucas
2	9 luisa
3	10 Pedro

# LIKE – Filtrando os primeiros caracteres + case sensitive

Agora vamos trabalhar com uma outra tabela, que vamos chamar de **Textos**.

Neste exemplo vamos ver mais algumas aplicações de filtros especiais, como pode ser visto na imagem ao lado.

```
SQLQuery1.sql - M...MARCUS\caval (53)* X
USE BD_Collation
CREATE TABLE Textos(
ID INT,
Texto VARCHAR(100) COLLATE SQL_Latin1_General_CI_AS)

INSERT INTO Textos(ID, Texto)
VALUES
(1, 'Marcos'), (2, 'Excel'), (3, 'leandro'), (4, 'K'), (5, 'X7'), (6, '19'), (7, '#M'), (8, '@9'), (9, 'M'), (10, 'RT')

-- Retornando nomes que possuem apenas 1 caracter
SELECT *
FROM Textos
WHERE Texto LIKE '[A-z]'

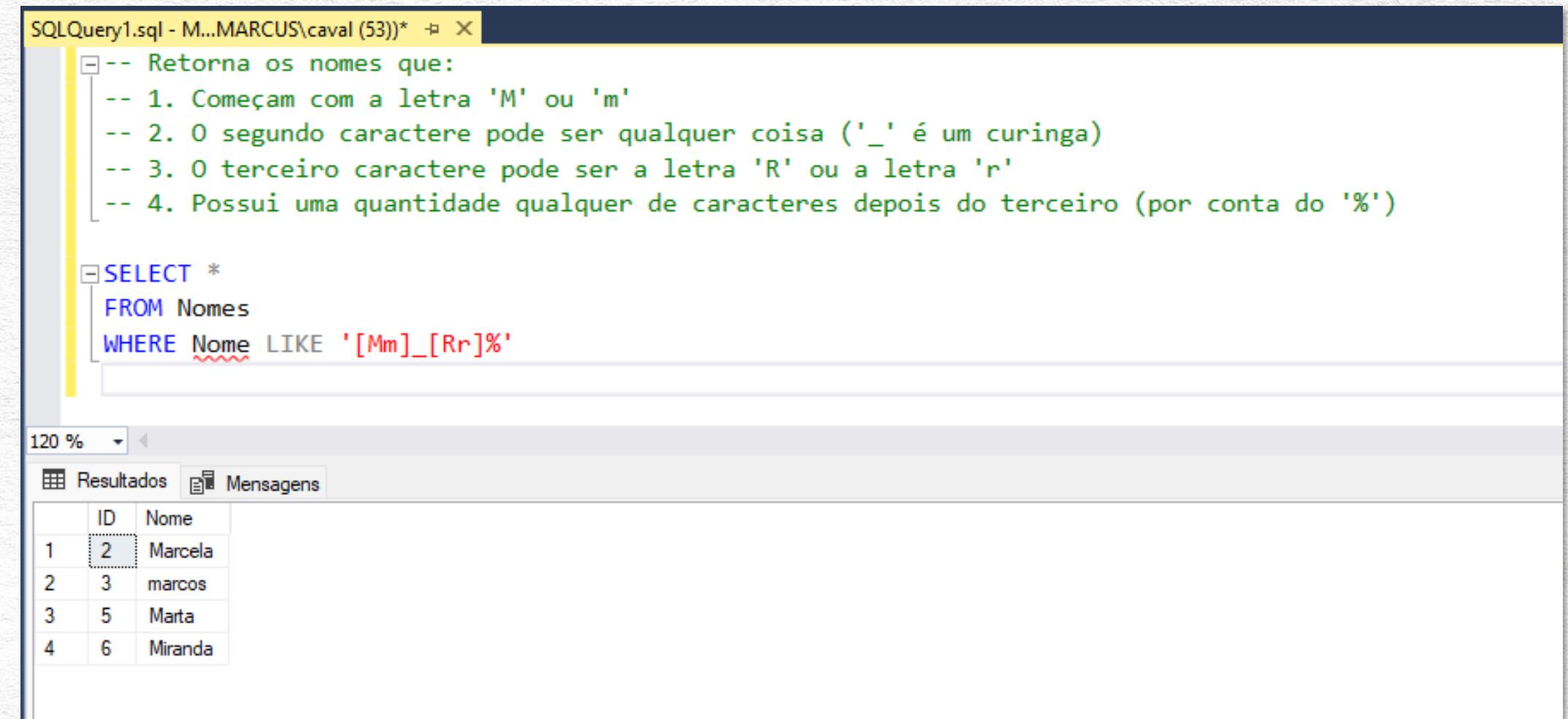
-- Retornando nomes que possuem apenas 2 caracteres
SELECT *
FROM Textos
WHERE Texto LIKE '[A-z][A-z]'

-- Retornando nomes que possuem apenas 2 caracteres: o primeiro uma letra, o segundo um número
SELECT *
FROM Textos
WHERE Texto LIKE '[A-z][0-9]'
```

# LIKE – Aplicando um filtro ainda mais personalizado

Conseguimos personalizar ainda mais o filtro combinando o que vimos até agora.

No exemplo ao lado, usamos o colchetes, o underline (para especificar ‘qualquer caractere’ naquela posição) e também o sinal de percentual, que significa que temos uma quantidade de texto qualquer no final da palavra.



The screenshot shows a SQL query window titled 'SQLQuery1.sql - M...MARCUS\caval (53)\*'. The query is:

```
-- Retorna os nomes que:  
-- 1. Começam com a letra 'M' ou 'm'  
-- 2. O segundo caractere pode ser qualquer coisa ('_ é um curinga)  
-- 3. O terceiro caractere pode ser a letra 'R' ou a letra 'r'  
-- 4. Possui uma quantidade qualquer de caracteres depois do terceiro (por conta do '%')  
  
SELECT *  
FROM Nomes  
WHERE Nome LIKE '[Mm]_[Rr]%'
```

The results pane shows a table with columns 'ID' and 'Nome', containing the following data:

ID	Nome	
1	2	Marcela
2	3	marcos
3	5	Marta
4	6	Miranda

# LIKE – Utilizando o operador de negação

Neste exemplo vemos a aplicação do operador ^, que faz uma negação no nosso filtro de textos.

Ao lado, é possível visualizar a aplicação deste operador.

The screenshot shows a SQL query window titled "SQLQuery1.sql - M...MARCUS\caval (53)\*" with two queries. The first query uses the pattern "[^Ll]%" to select names that do not start with 'L' or 'l'. The second query uses the pattern "[^Ee]%" to select names that do not start with 'E' or 'e'. Both queries return the same set of names from a "Nomes" table.

```
-- Retorna nomes que não começa com as letras 'L' ou 'l'  
SELECT *  
FROM Nomes  
WHERE Nome LIKE '[^Ll]%'  
  
-- Retorna nomes que não começa com as letras 'L' ou 'l'  
SELECT *  
FROM Nomes  
WHERE Nome LIKE '[^Ee]%'
```

Resultados:

ID	Nome
1	Matheus
2	Marcela
3	marcos
4	MAuricio
5	Marta
6	Miranda
7	Melissa
8	Pedro

Mensagens:

ID	Nome
1	Matheus
2	Marcela
3	marcos
4	MAuricio
5	Marta
6	Miranda

# LIKE – Identificando caracteres especiais

É possível fazer um filtro para visualizar todos os textos que possuem caracteres especiais. A aplicação é mostrada ao lado.

The screenshot shows a SQL query window titled "SQLQuery1.sql - M...MARCUS\caval (53)\*". The query is:

```
-- Retornando nomes que possuem caracteres especiais
SELECT *
FROM Textos
WHERE Texto LIKE '%[^A-z0-9]%'
```

The results pane shows a table with two rows:

	ID	Texto
1	7	#M
2	8	@9

# LIKE – Aplicações com números

Por fim, temos uma aplicação especial para números, como mostrado nos exemplos ao lado.

The screenshot shows a SQL Server Management Studio (SSMS) window titled "SQLQuery1.sql - M...MARCUS\caval (53)\*". The code in the query pane demonstrates how to use the LIKE operator with numbers:

```
USE BD_Collation
CREATE TABLE Numeros(
    Numero DECIMAL(20, 2))

INSERT INTO Numeros(Numero)
VALUES
    (50), (30.23), (9), (100.54), (15.9), (6.5), (10), (501.76), (1000.56), (31)

-- Retornando os números que possuem 2 dígitos na parte inteira
SELECT *
FROM Numeros
WHERE Numero LIKE '[0-9][0-9].[0][0]'

-- Retornando linhas que:
-- 1. Possuem 3 dígitos na parte inteira, sendo o primeiro dígito igual a 5
-- 2. O primeiro número da parte decimal seja 7.
-- 3. O segundo número da parte decimal seja um número entre 0 e 9.

SELECT *
FROM Numeros
WHERE Numero LIKE '[5]__[7][0-9]'
```

The results pane shows two sets of data:

Numero
50.00
10.00
31.00

Numero
501.76

# SQL

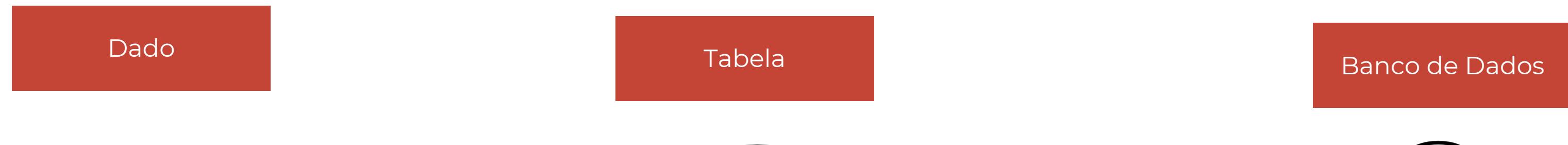
## Módulo MySQL



# O que é um Banco de Dados?

Um banco de dados é um conjunto de dados organizados dentro de uma ou mais tabelas, que terão alguma relação entre si.

Partimos de uma informação isolada, na sua forma mais simples, e a partir de um conjunto de diversas informações isoladas, conseguimos organizar essas informações para criar tabelas e bancos de dados.



- 1 Compra online do João

- Diversas compras online (João, Amanda, Pedro, Carol, ...)

- Diversas compras online (João, Amanda, Pedro, Carol, ...)
- Diversas informações sobre os clientes
- Diversas informações dos produtos comprados

# O que é um Banco de Dados?

000

“

**Bancos de dados** são conjuntos de tabelas, com alguma relação entre si, com dados sobre pessoas, lugares ou coisas.

Estes dados organizados permitem a compreensão de um determinado fenômeno na empresa, seja a preferência dos usuários em uma rede social, seja o perfil de consumo em um aplicativo de transações financeiras.

# Do que é composto um banco de dados?

Um banco de dados é composto pelos seguintes elementos:

- ✓ Tabelas
- ✓ Campos (Atributos)
- ✓ Registros (Tuplas)

# Do que é composto um banco de dados?

Um banco de dados é composto pelos seguintes elementos:

- Tabelas
- Campos (Atributos)
- Registros (Tuplas)

**Tabelas** armazenam dados dentro de um banco de dados.

Uma tabela é um conjunto de dados relacionados, e é composta por linhas e colunas.

# Do que é composto um banco de dados?

Um banco de dados é composto pelos seguintes elementos:

- Tabelas
- Campos (Atributos)
- Registros (Tuplas)

**Campos** (também chamados de atributos ou simplesmente colunas) representam os atributos dos dados, como ID do Produto, Nome do Cliente, Quantidade Vendida, Preço, etc.

Um campo é uma coluna de uma tabela, contendo informações específicas sobre cada registro (linha) da tabela.

# Do que é composto um banco de dados?

Um banco de dados é composto pelos seguintes elementos:

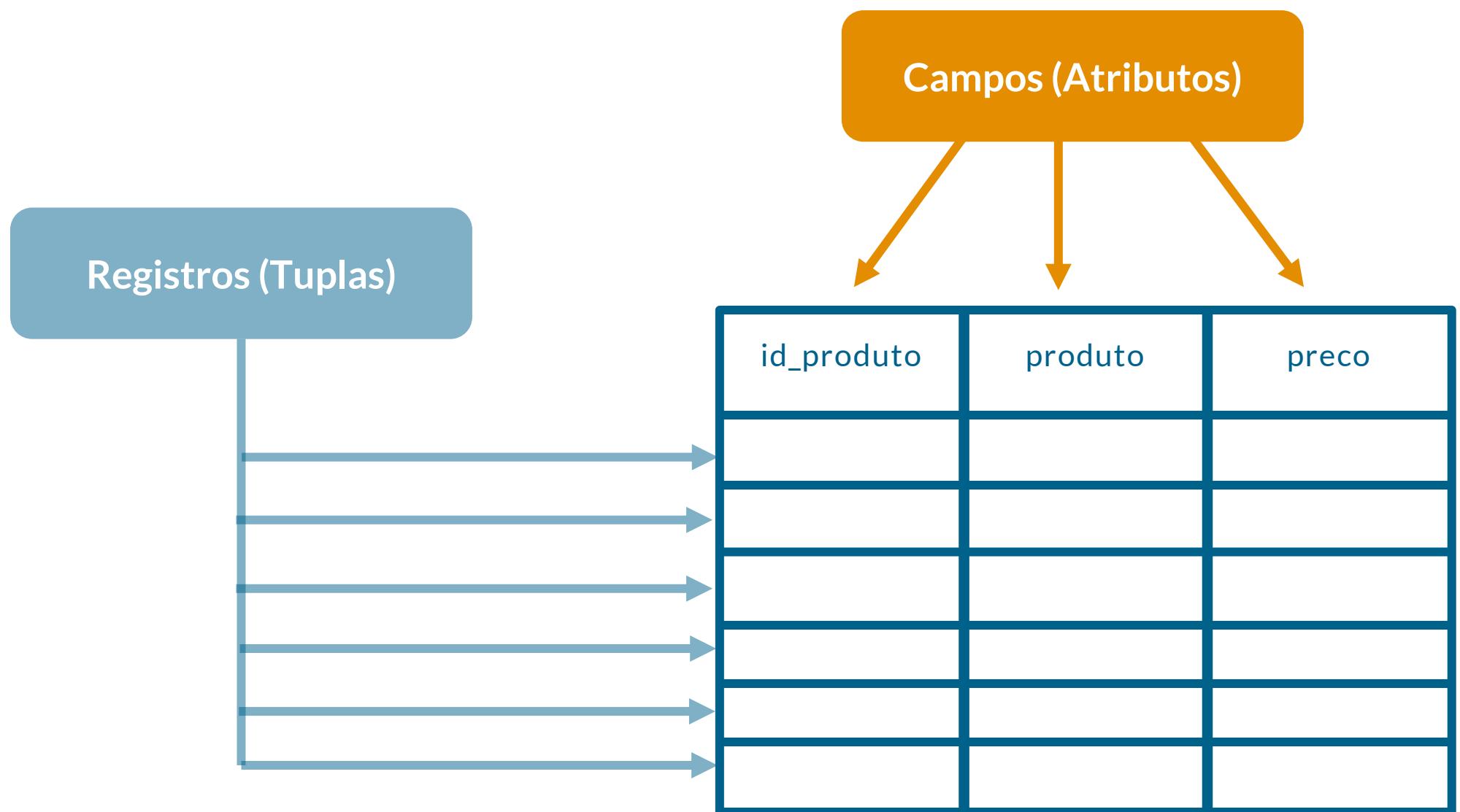
- Tabelas
- Campos (Atributos)
- Registros (Tuplas)

**Registros** (também chamados de tuplas ou simplesmente linhas) representam cada entrada de dados dentro de uma tabela.

# Do que é composto um banco de dados?

Um banco de dados é composto pelos seguintes elementos:

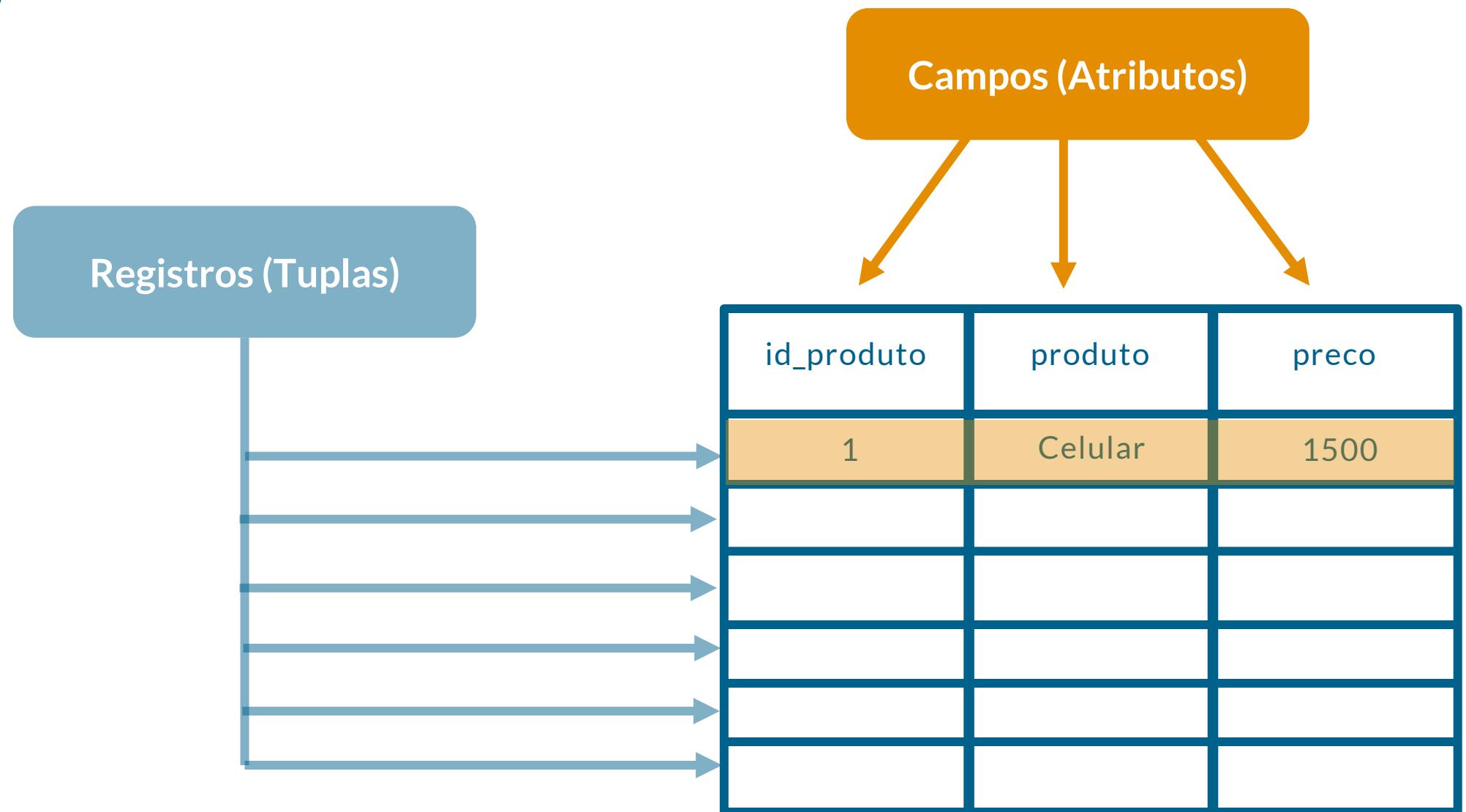
- Tabelas
- Campos (Atributos)
- Registros (Tuplas)



# Do que é composto um banco de dados?

Um banco de dados é composto pelos seguintes elementos:

- Tabelas
- Campos (Atributos)
- Registros (Tuplas)

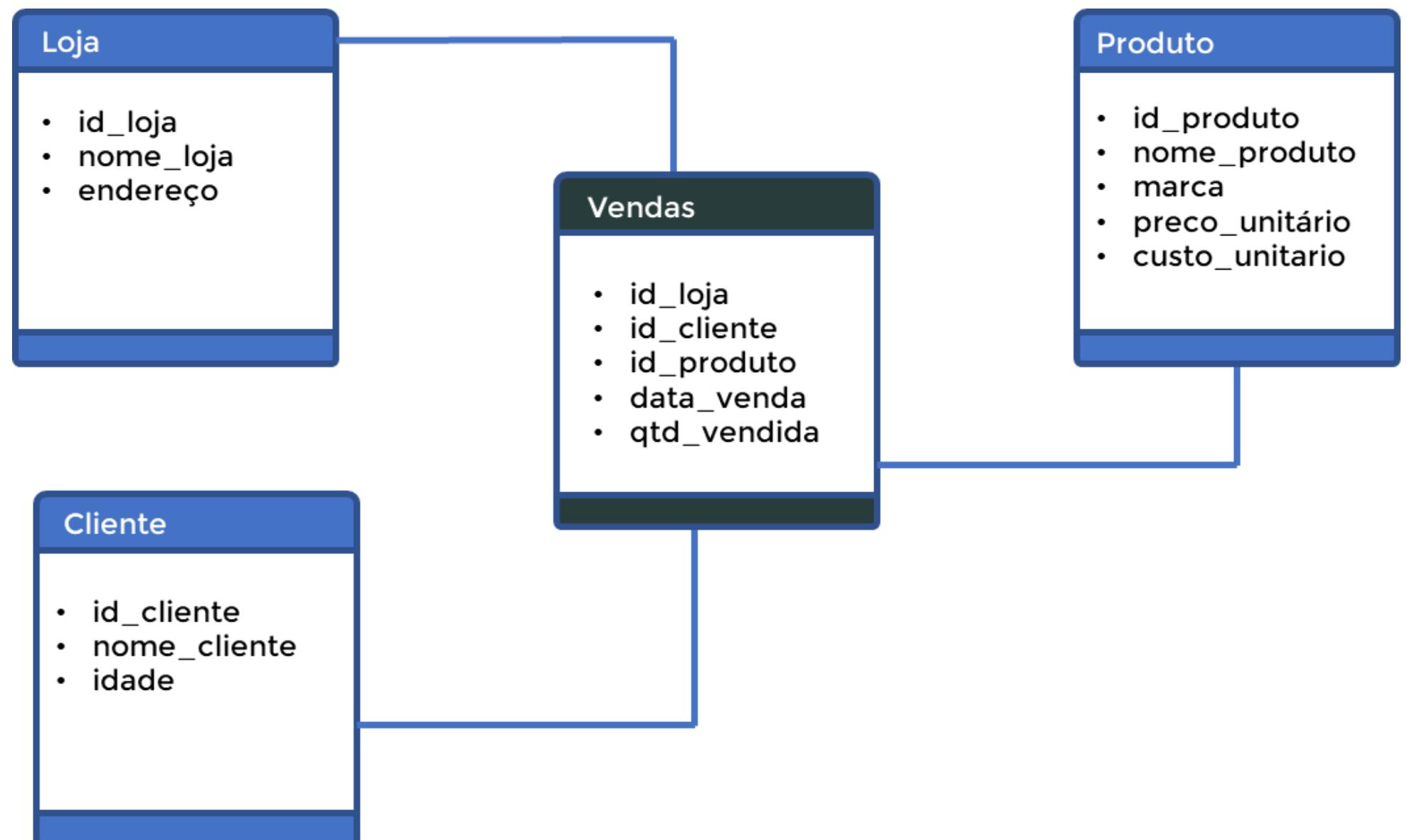


# O que é um Banco de Dados?

O desenho esquemático de um banco de dados é mostrado ao lado. Diversas tabelas, com diferentes informações sobre um negócio, e que possuem algum tipo de relação entre si.

A esse banco de dados damos o nome de **RELACIONAL**.

Bancos de dados relacionais serão o foco do nosso curso, até por serem o tipo de bancos de dados mais comumente encontrados no mercado.

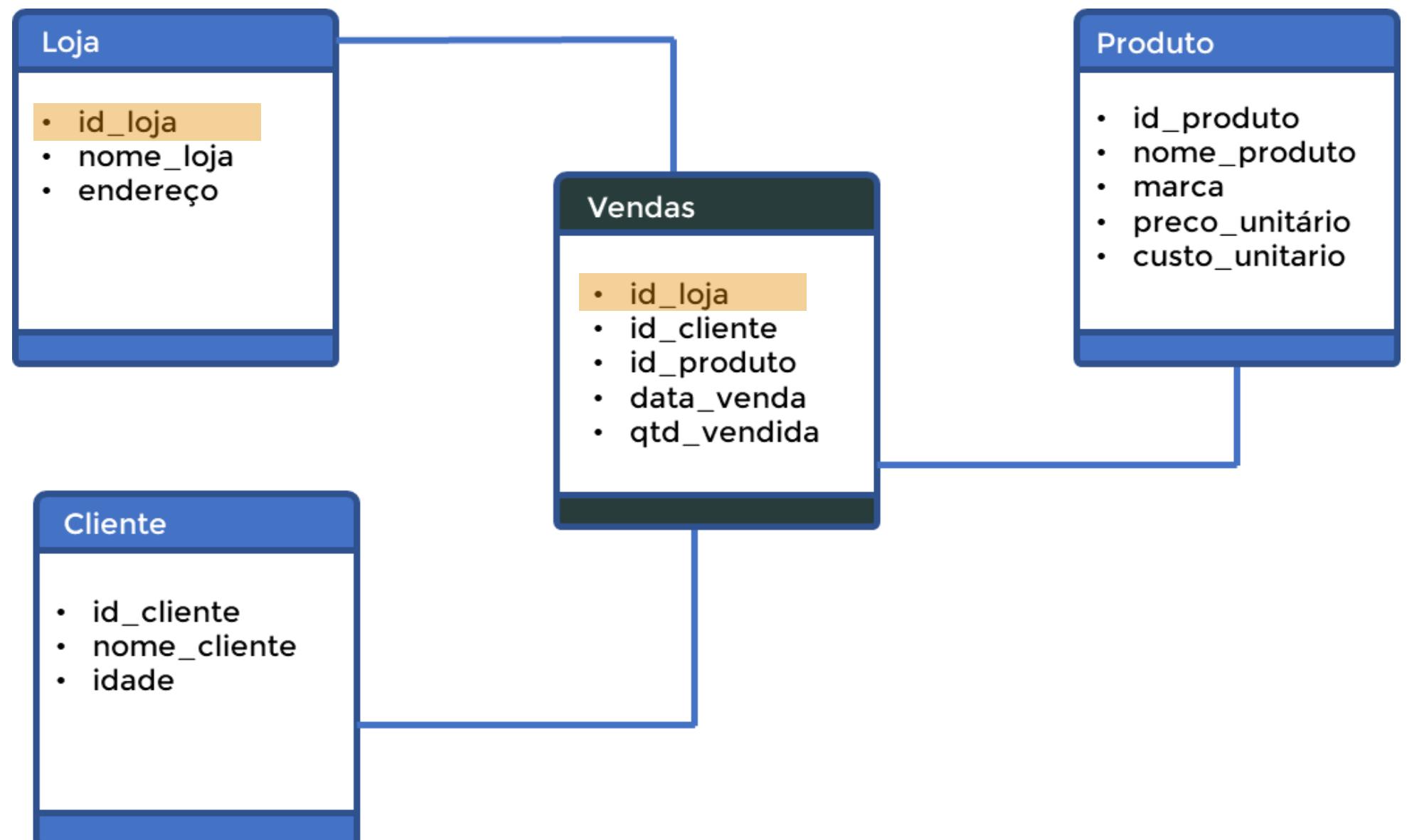


# O que é um Banco de Dados?

O desenho esquemático de um banco de dados é mostrado ao lado. Diversas tabelas, com diferentes informações sobre um negócio, e que possuem algum tipo de relação entre si.

A esse banco de dados damos o nome de **RELACIONAL**.

Bancos de dados relacionais serão o foco do nosso curso, até por serem o tipo de bancos de dados mais comumente encontrados no mercado.

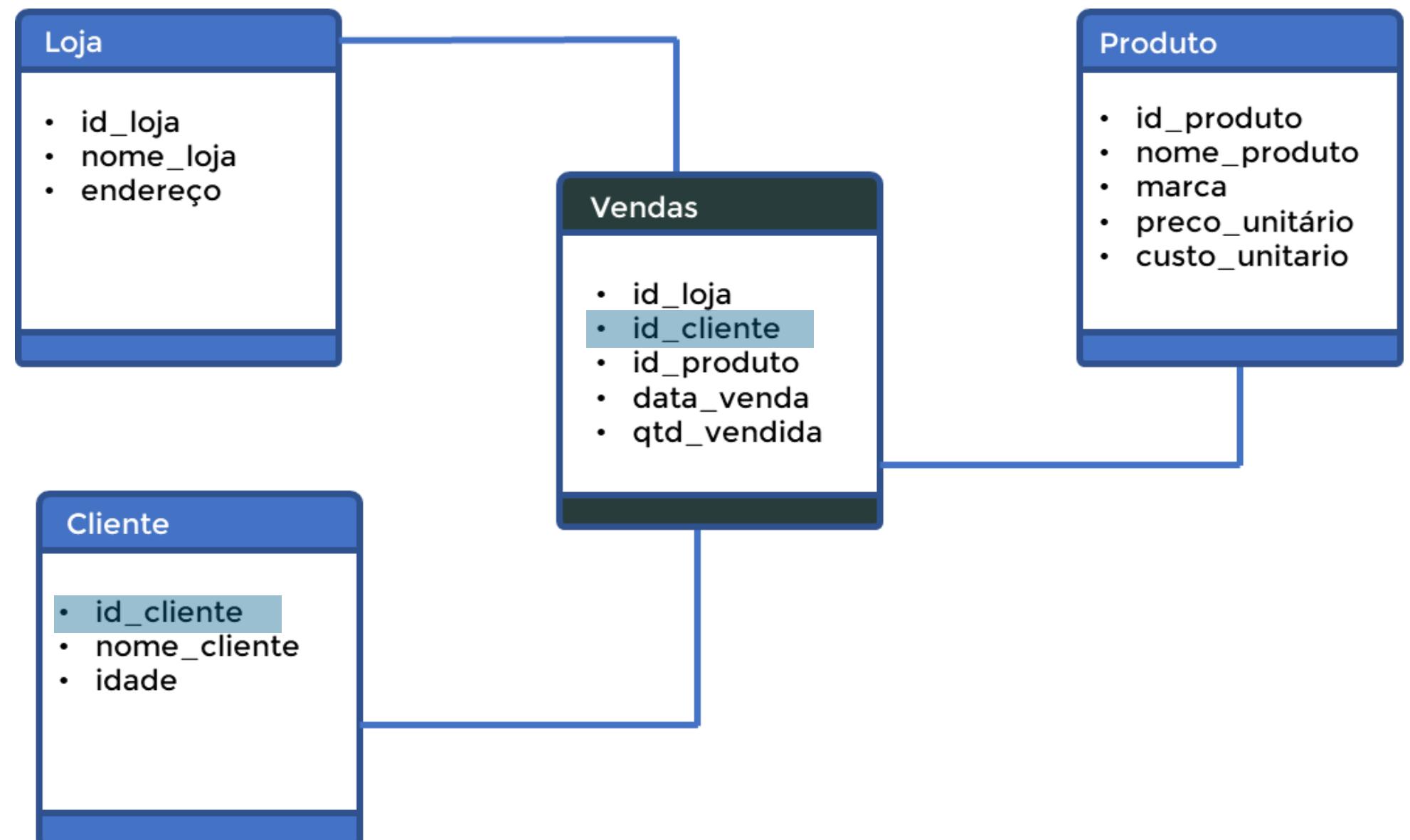


# O que é um Banco de Dados?

O desenho esquemático de um banco de dados é mostrado ao lado. Diversas tabelas, com diferentes informações sobre um negócio, e que possuem algum tipo de relação entre si.

A esse banco de dados damos o nome de **RELACIONAL**.

Bancos de dados relacionais serão o foco do nosso curso, até por serem o tipo de bancos de dados mais comumente encontrados no mercado.

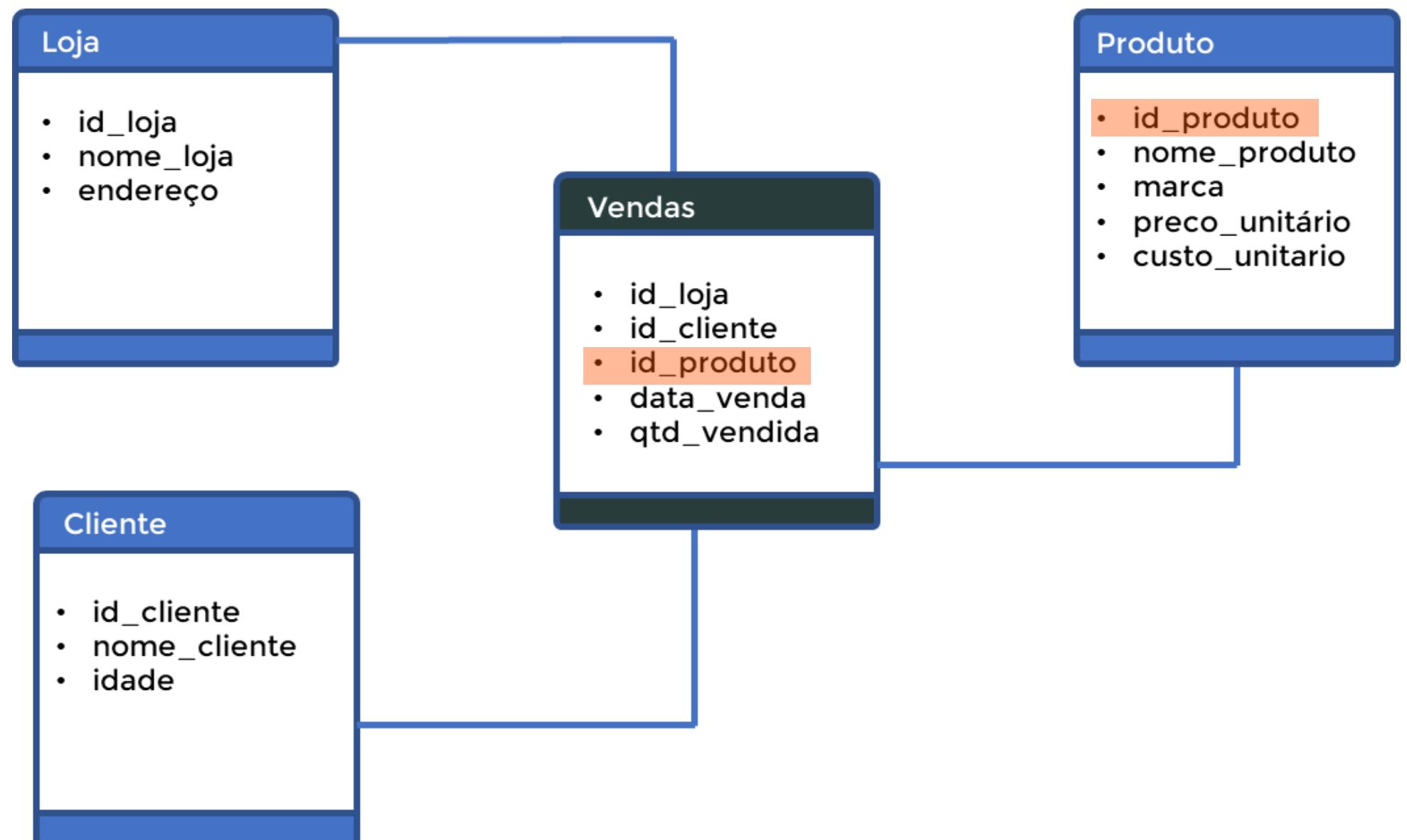


# O que é um Banco de Dados?

O desenho esquemático de um banco de dados é mostrado ao lado. Diversas tabelas, com diferentes informações sobre um negócio, e que possuem algum tipo de relação entre si.

A esse banco de dados damos o nome de **RELACIONAL**.

Bancos de dados relacionais serão o foco do nosso curso, até por serem o tipo de bancos de dados mais comumente encontrados no mercado.

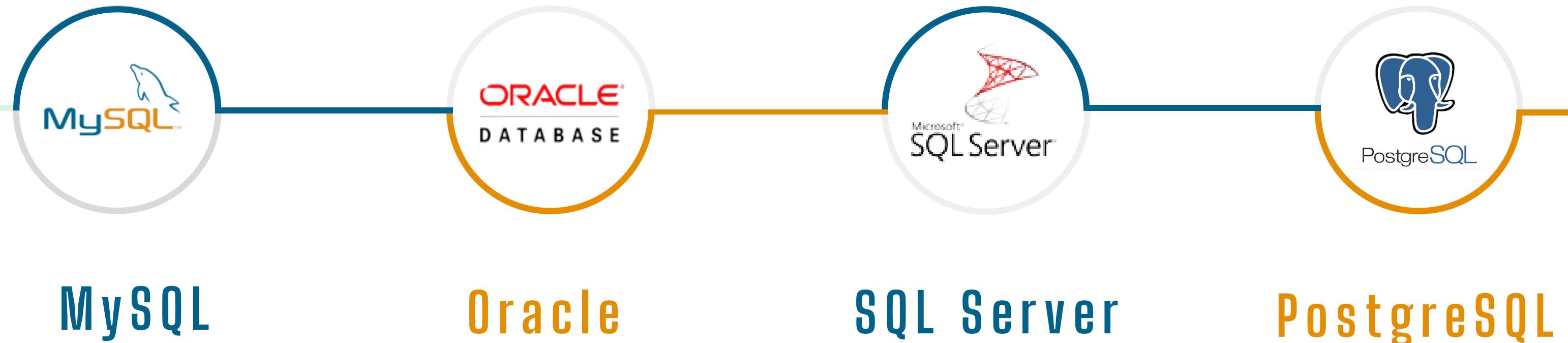


# Sistemas de Bancos de Dados

000

Existem alguns SGBDs para Bancos de Dados Relacionais que são muito utilizados por grandes empresas. Abaixo, temos os 4 principais programas para SGBDs.

É importante que fique claro que **todos esses SGBDs** utilizam o SQL como linguagem de programação.



# SQL: Structured Query Language

Para acessar e consultar os dados em um banco de dados, é necessário o uso de uma série de comandos.

Esses comandos, na verdade, se tratam de uma linguagem de programação, chamada SQL: Structured Query Language.

Traduzindo para o português, a sigla SQL significa Linguagem de Consulta Estruturada.

Essa é uma linguagem de bancos de dados universal e é por dela que será possível a consulta aos dados dentro dos bancos de dados.

# SQL: Structured Query Language

Para acessar e consultar os dados em um banco de dados, é necessário o uso de uma série de comandos.

Esses comandos, na verdade, se tratam de uma linguagem de programação, chamada SQL: Structured Query Language.

Traduzindo para o português, a sigla SQL significa Linguagem de Consulta Estruturada.

Essa é uma linguagem de bancos de dados universal e é por dela que será possível a consulta aos dados dentro dos bancos de dados.

“

SQL significa “**Structured Query Language**”. Se trata de uma linguagem de programação utilizada para armazenar, consultar, adicionar e excluir informações em um banco de dados.

# SQL: Structured Query Language

Para acessar e consultar os dados em um banco de dados, é necessário o uso de uma série de comandos.

Esses comandos, na verdade, se tratam de uma linguagem de programação, chamada SQL: Structured Query Language.

Traduzindo para o português, a sigla SQL significa Linguagem de Consulta Estruturada.

Essa é uma linguagem de bancos de dados universal e é por dela que será possível a consulta aos dados dentro dos bancos de dados.



# SQL: Structured Query Language

Para acessar e consultar os dados em um banco de dados, é necessário o uso de uma série de comandos.

Esses comandos, na verdade, se tratam de uma linguagem de programação, chamada SQL: Structured Query Language.

Traduzindo para o português, a sigla SQL significa Linguagem de Consulta Estruturada.

Essa é uma linguagem de bancos de dados universal e é por dela que será possível a consulta aos dados dentro dos bancos de dados.



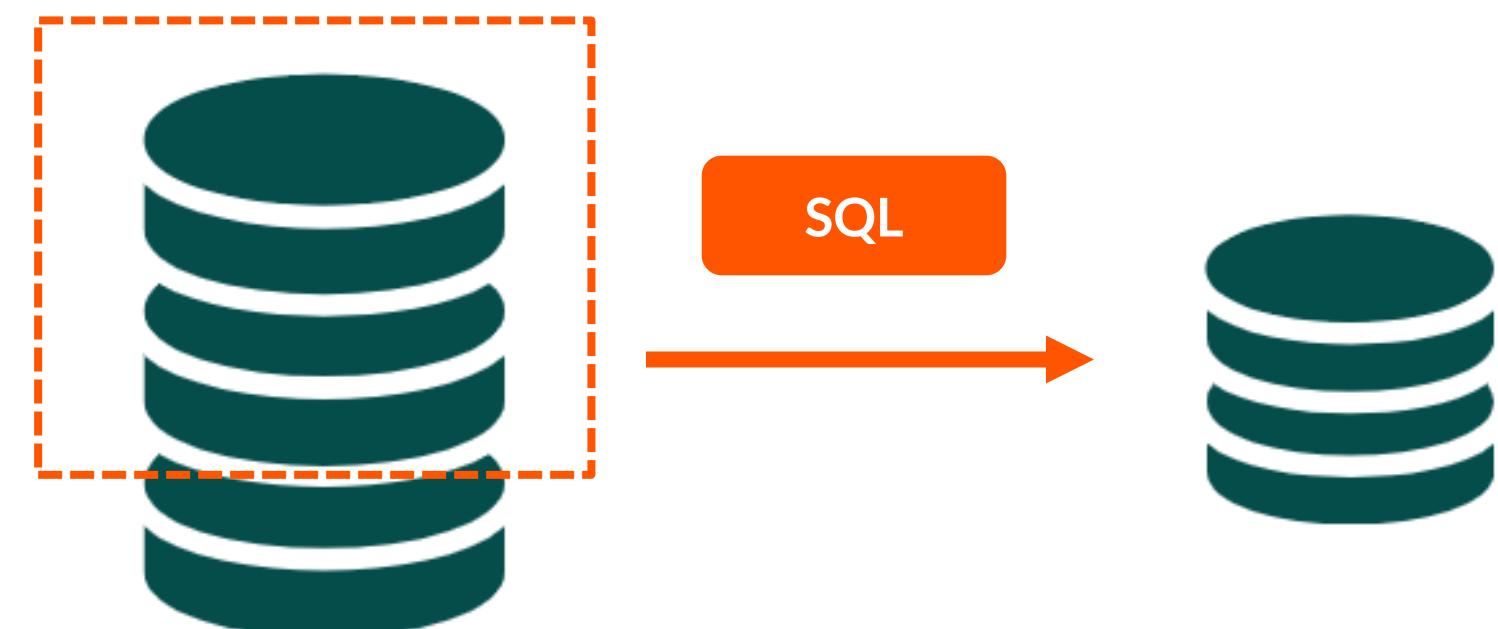
# SQL: Structured Query Language

Para acessar e consultar os dados em um banco de dados, é necessário o uso de uma série de comandos.

Esses comandos, na verdade, se tratam de uma linguagem de programação, chamada SQL: Structured Query Language.

Traduzindo para o português, a sigla SQL significa Linguagem de Consulta Estruturada.

Essa é uma linguagem de bancos de dados universal e é por dela que será possível a consulta aos dados dentro dos bancos de dados.



# O que é uma query (consulta)?

Uma query é um pedido de uma informação ou de um dado. Esse pedido também pode ser entendido como uma consulta, uma solicitação ou, ainda, uma requisição.

Em resumo, uma query (ou consulta) é uma leitura dos dados de uma tabela dentro de um banco de dados. Ou seja, quando queremos visualizar determinados dados de uma tabela, na prática o que queremos é fazer uma consulta aos dados do banco de dados.

Porém, a leitura desses dados não é aleatória. Ela é baseada em uma série de comandos, feitos a partir da linguagem SQL.

# O que é uma query (consulta)?

Uma query é um pedido de uma informação ou de um dado. Esse pedido também pode ser entendido como uma consulta, uma solicitação ou, ainda, uma requisição.

Em resumo, uma query (ou consulta) é uma leitura dos dados de uma tabela dentro de um banco de dados. Ou seja, quando queremos visualizar determinados dados de uma tabela, na prática o que queremos é fazer uma consulta aos dados do banco de dados.

Porém, a leitura desses dados não é aleatória. Ela é baseada em uma série de comandos, feitos a partir da linguagem SQL.



# O que é uma query (consulta)?

Ao lado, temos um exemplo bem simplificado de como seria essa query (consulta) aos dados em uma tabela de um banco de dados, utilizando um código em SQL.

Tabela

id   produto	data_venda	valor
1   televisão	2021-03-20	1500
2   computador	2021-03-22	2300
3   celular	2021-03-25	800
4   ps4	2021-03-28	3100
5   tablet	2021-03-28	650

SQL

```
1• SELECT *
2 FROM tabela_vendas
3 WHERE valor > 1000;
```

id   produto	data_venda	valor
1   televisão	2021-03-20	1500
2   computador	2021-03-22	2300
4   ps4	2021-03-28	3100

Consulta

# Grupos de comando no SQL

000

Os comandos do SQL podem ser divididos em **quatro grupos**.

**DDL**

**Data Definition Language**

**CREATE**

Cria uma nova tabela, view ou outro objeto dentro do banco de dados.

**ALTER**

Modifica um objeto dentro do banco de dados (tabela, view, etc).

**DROP**

Exclui um objeto dentro do banco de dados (tabela, view, etc).

**DML**

**Data Manipulation Language**

**INSERT**

Adiciona uma nova linha em uma tabela.

**UPDATE**

Atualiza os valores das linhas de uma tabela.

**DELETE**

Exclui linhas de uma tabela.

**DCL**

**Data Control Language**

**GRANT**

Dá privilégios a um usuário.

**REVOKE**

Retira privilégios de um usuário.

**DQL**

**Data Query Language**

**SELECT**

Comando de seleção de linhas de uma tabela.

# SQL, MySQL, SQL Server, Oracle e PostgreSQL: Não confunda os SQL's!!

O SQL é uma linguagem de consulta a bancos de dados, enquanto o MySQL, SQL Server, Oracle Database e PostgreSQL são programas utilizados para gerenciamento dos bancos de dados.

# SQL, MySQL, SQL Server, Oracle e PostgreSQL: Não confunda os SQL's!!

O SQL é uma linguagem de consulta a bancos de dados, enquanto o MySQL, SQL Server, Oracle Database e PostgreSQL são programas utilizados para gerenciamento dos bancos de dados.

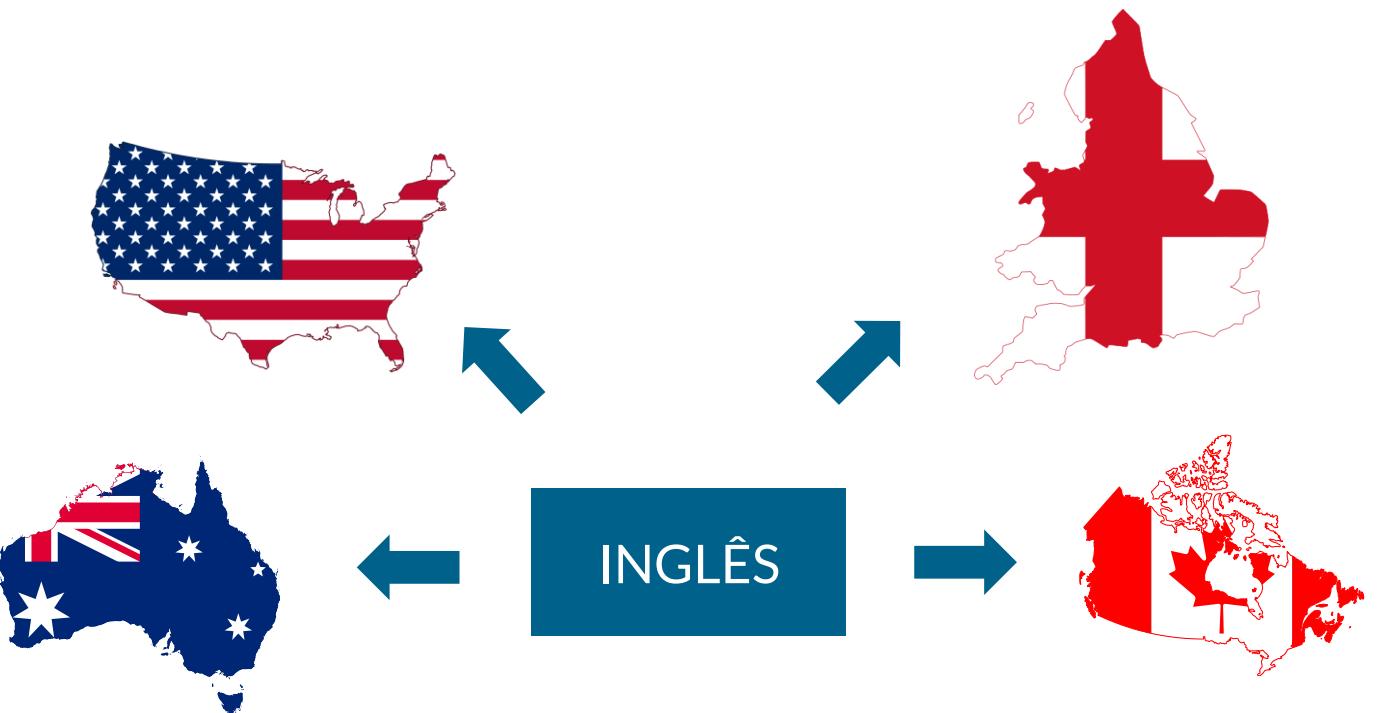
Para ficar clara essa diferença, faça um paralelo com o inglês.

# SQL, MySQL, SQL Server, Oracle e PostgreSQL: Não confunda os SQL's!!

O SQL é uma linguagem de consulta a bancos de dados, enquanto o MySQL, SQL Server, Oracle Database e PostgreSQL são programas utilizados para gerenciamento dos bancos de dados.

Para ficar clara essa diferença, faça um paralelo com o inglês.

O inglês pode ser falado em diferentes países, como o EUA, Inglaterra, Austrália, dentre outros. Porém, não aprendemos inglês aplicado aos EUA, ou inglês aplicado à Inglaterra. Inglês é inglês, e pode ser falado em diferentes países.



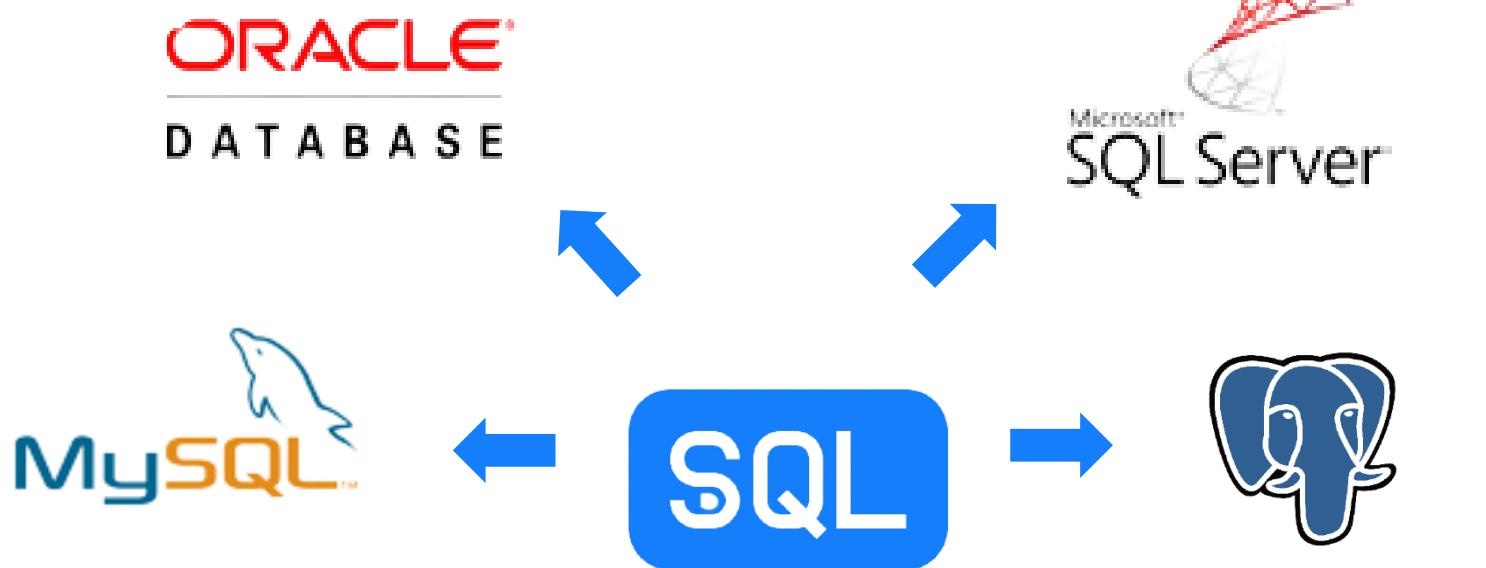
# SQL, MySQL, SQL Server, Oracle e PostgreSQL: Não confunda os SQL's!!

O SQL é uma linguagem de consulta a bancos de dados, enquanto o MySQL, SQL Server, Oracle Database e PostgreSQL são programas utilizados para gerenciamento dos bancos de dados.

Para ficar clara essa diferença, faça um paralelo com o inglês.

O inglês pode ser falado em diferentes países, como o EUA, Inglaterra, Austrália, dentre outros. Porém, não aprendemos inglês aplicado aos EUA, ou inglês aplicado à Inglaterra. Inglês é inglês, e pode ser falado em diferentes países.

O mesmo vale para o SQL. O SQL é uma linguagem de consulta (idioma) que pode ser usado em diferentes programas (países): MySQL, SQL Server, Oracle e PostgreSQL.



# SQL, MySQL, SQL Server, Oracle e PostgreSQL: Não confunda os SQL's!!

O SQL é uma linguagem de consulta a bancos de dados, enquanto o MySQL, SQL Server, Oracle Database e PostgreSQL são programas utilizados para gerenciamento dos bancos de dados.

Para ficar clara essa diferença, faça um paralelo com o inglês.

O inglês pode ser falado em diferentes países, como o EUA, Inglaterra, Austrália, dentre outros. Porém, não aprendemos inglês aplicado aos EUA, ou inglês aplicado à Inglaterra. Inglês é inglês, e pode ser falado em diferentes países.

O mesmo vale para o SQL. O SQL é uma linguagem de consulta (idioma) que pode ser usado em diferentes programas (países): MySQL, SQL Server, Oracle e PostgreSQL.

Portanto, não se assuste com tantos SQLs. Na verdade, só existe um SQL. Todos os demais são programas que utilizam o SQL para realizar as consultas aos bancos de dados.



# SQL

## Instalação do MySQL

# SQL



# Instalação do MySQL

O SGBD escolhido será o **MySQL**.

Lembrando que o SGBD será composto essencialmente por 2 partes: um **Servidor** e uma **Interface**.

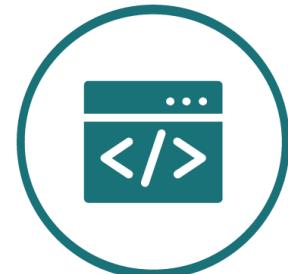
Para o caso do MySQL, teremos que instalar o **MySQL Server** e o **MySQL Workbench**, respectivamente.

MySQL Server



um **servidor**, onde vamos conseguir armazenar os nossos bancos de dados.

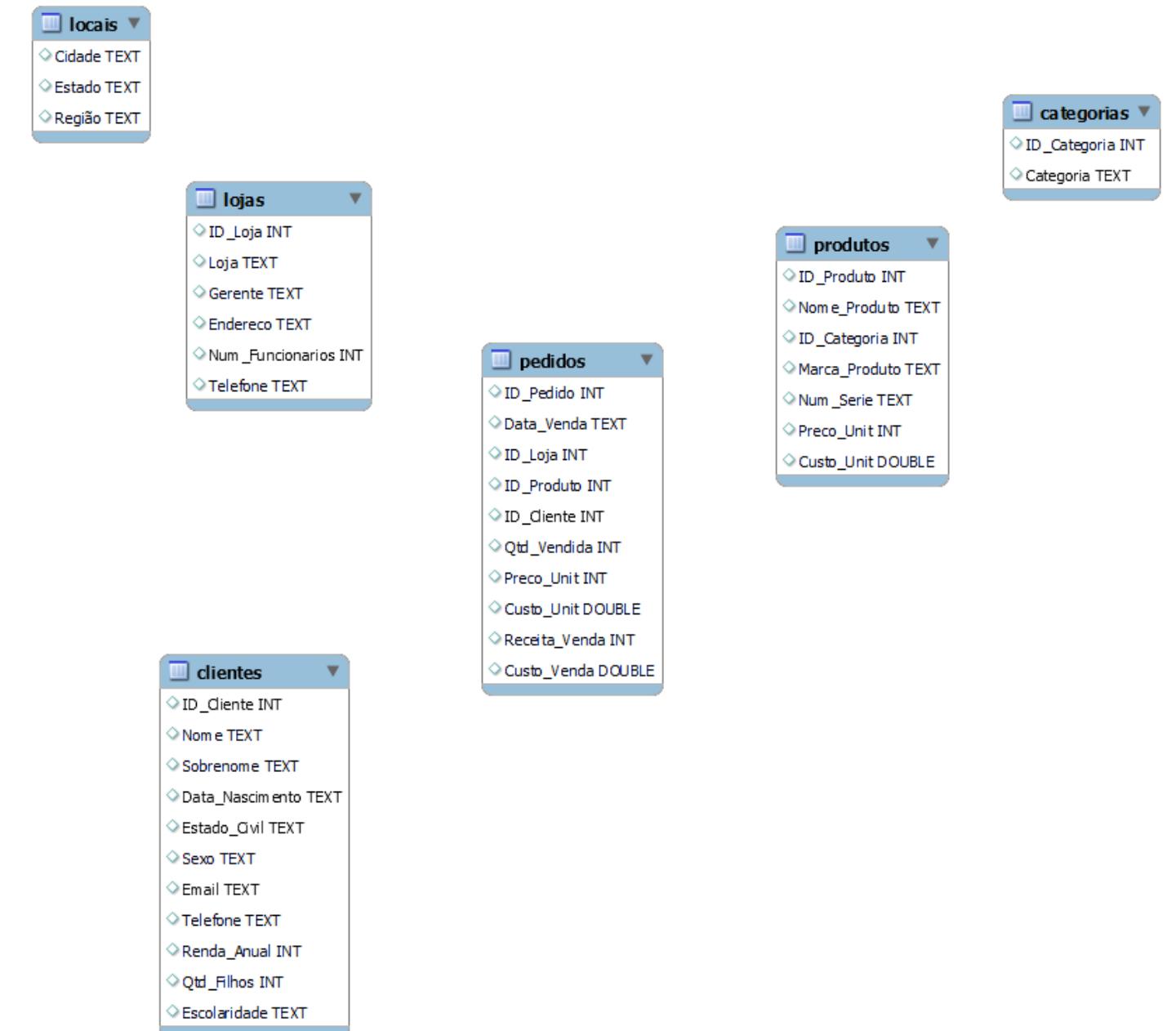
MySQL Workbench



uma **interface** amigável que nos permite escrever os códigos em SQL para acessar os bancos de dados.

# Banco de Dados utilizado

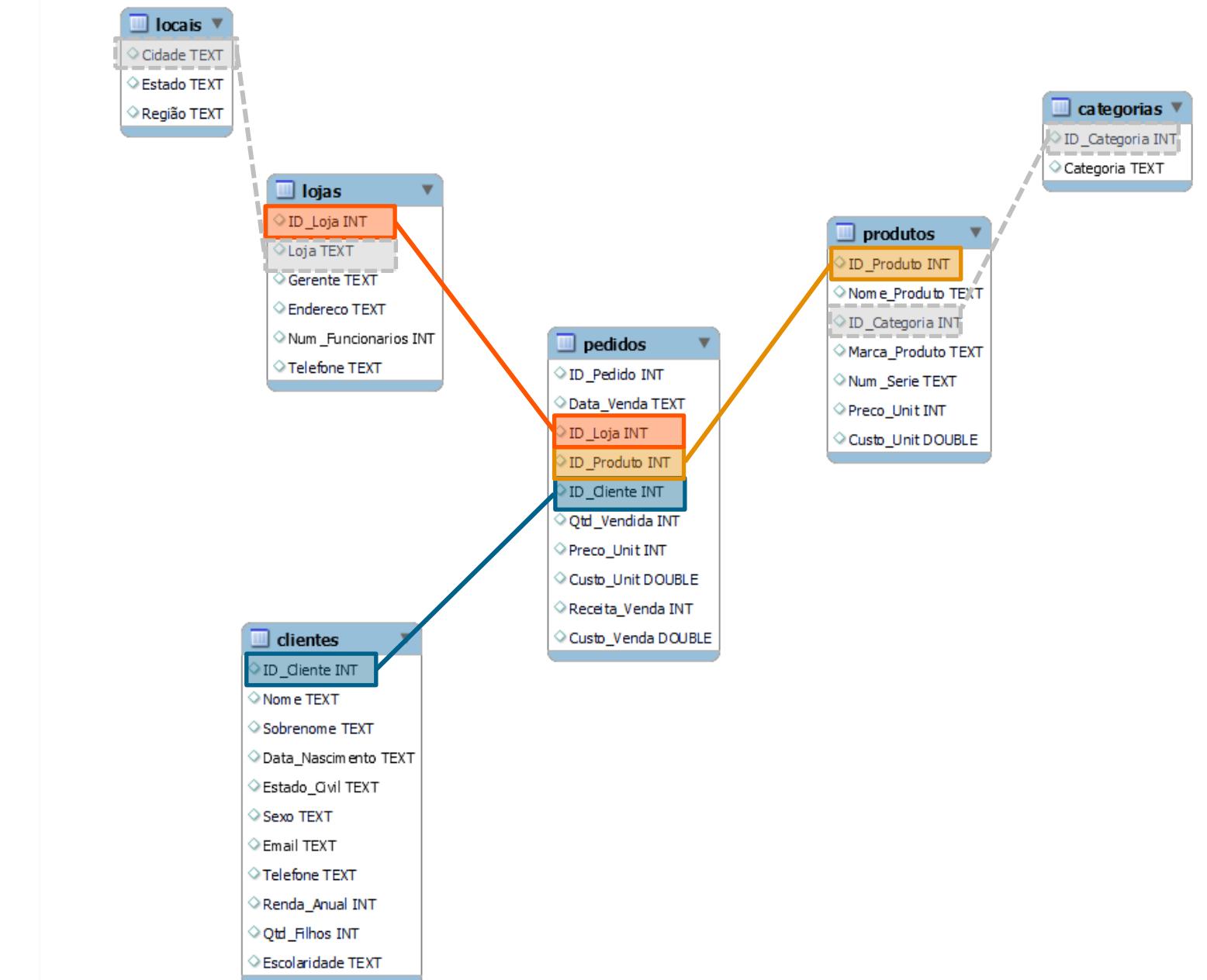
As tabelas do banco de dados estão mostradas na imagem ao lado.



# Banco de Dados utilizado

As tabelas do banco de dados estão mostradas na imagem ao lado.

Existem colunas em comum entre as tabelas, para que seja possível criar um banco de dados relacional.



# SQL

Criando as primeiras consultas



# SELECT \*, SELECT FROM, SELECT AS e SELECT LIMIT

Vamos começar aprendendo sobre os comandos de seleção de dados no MySQL. São eles:

## Comandos de Seleção

SELECT FROM

SELECT LIMIT

SELECT AS

SELECT DISTINCT

# SELECT \*

## OBJETIVO

Selecionar **todas as colunas** e todas as linhas de uma tabela.

## CÓDIGO

```
1 • SELECT *
2 FROM Tabela;
```

## RESULTADO

Col1	Col2	Col3
1	Nome	10.5
2	Texto	5.45
3	Carro	51.3
4	Celular	100.6
5	SQL	500

# SELECT (colunas)

## OBJETIVO

Selecionar **apenas colunas específicas** de uma tabela.

## CÓDIGO

```
1 • SELECT Col1, Col2  
2   FROM Tabela;
```

## RESULTADO

Col1	Col2
1	Nome
2	Texto
3	Carro
4	Celular
5	SQL

# SELECT AS

## OBJETIVO

Selecionar colunas específicas e dar um nome para essas colunas.

## CÓDIGO

```
1 • SELECT  
2     Col1 AS "Coluna 1",  
3     Col2 AS "Coluna 2"  
4 FROM Tabela;
```

## RESULTADO

Coluna 1	Coluna 2
1	Nome
2	Texto
3	Carro
4	Celular
5	SQL

# SELECT LIMIT

## OBJETIVO

Selecionar apenas as N primeiras linhas de uma determinada tabela.

## CÓDIGO

```
1 • SELECT  
2      *  
3 FROM Tabela  
4 LIMIT 2;
```

## RESULTADO

Col1	Col2	Col3
1	Nome	10.5
2	Texto	5.45

# SELECT DISTINCT

## OBJETIVO

Selecionar apenas os valores distintos de uma coluna.

## CÓDIGO

```
SELECT  
    DISTINCT genero  
FROM filmes;
```

## RESULTADO

genero
Comédia
Drama
Ficção e Fantasia
Mistério e Suspense
Arte
Animação
Ação e Aventura

# ORDER BY DESC e ORDER BY ASC

Os **comandos de ordenação** permitem a ordenação dos dados da nossa tabela, a partir de uma coluna. Com ele, podemos ordenar por ordem crescente, ordem alfabética, e assim vai.

## Comandos de Ordenação

ORDER BY ASC

ORDER BY DESC

# ORDER BY (DESC)

## OBJETIVO

Permite ordenar (classificar) uma tabela a partir de uma determinada coluna em **ordem decrescente**.

## CÓDIGO

```
1 • SELECT  
2      *  
3 FROM Tabela  
4 ORDER BY Col3 DESC;
```

## RESULTADO

Col1	Col2	Col3
5	SQL	500
4	Celular	100.6
3	Carro	51.3
1	Nome	10.5
2	Texto	5.45

# ORDER BY (ASC)

## OBJETIVO

Permite ordenar (classificar) uma tabela a partir de uma determinada coluna em **ordem crescente**.

## CÓDIGO

```
1 • SELECT  
2      *  
3 FROM Tabela  
4 ORDER BY Col3;
```

## RESULTADO

Col1	Col2	Col3
2	Texto	5.45
1	Nome	10.5
3	Carro	51.3
4	Celular	100.6
5	SQL	500

# Comentários

Existem 3 formas de comentar códigos, assim como mostrado ao lado.

Obs: As opções de traço duplo e barra-asterisco funcionam em qualquer SGBD que usarmos.

```
1  -- Este é um comentário
2 ❶ um código qualquer
3
4  # Este é outro comentário
5  outro código qualquer
6
7  /* Este é um bloco de código
8  este é outro bloco de código */
9  outro código
10 mais um código
```

# SQL

## Criando filtros



# WHERE: Comando para filtrar dados

Os comandos de filtragem nos permitem criar filtros nas nossas tabelas dos bancos de dados.

## Comandos de Filtragem

WHERE

WHERE AND/OR

WHERE IN

WHERE BETWEEN

# COMANDOS DE FILTRAGEM

Essencialmente, podemos fazer filtros com colunas que contenham 3 tipos de informação.

Veremos também que é possível criar filtros em mais de 1 coluna ao mesmo tempo.

NÚMEROS

TEXTOS

DATAS

# FILTROS DE NÚMEROS

000



## NÚMEROS

Podemos aplicar filtros em colunas numéricas.

Para isso, basta utilizar os sinais lógicos como =, <, >, <=, >=, <>.

```
1
2      -- Mostra apenas os produtos com preços iguais ou maiores que R$1.800
3 • SELECT *
4 FROM produtos
5 WHERE Preco_Unit >= 1800;
```

ID_Produto	Nome_Produto	ID_Categoria	Marca_Produto	Num_Serie	Preco_Unit	Custo_Unit
1	Monitor LED 19,5" Full HD HDMI	1	DELL	MNT-DL-831923	2300	966
2	Monitor Curvo 24" 144Hz HDMI	1	SAMSUNG	MNT-SS-001939	2800	980
6	Cadeira Gamer reclinável Azul/Laranja	3	ALTURA	CGM-AL-9N914J	1800	540
7	Cadeira Gamer PC Racer Vermelha	3	ALTURA	CGM-AL-0147FI	3100	1395
13	Notebook LC2100 Intel Core i5 8GB	6	SAMSUNG	NOT-SS-918457	3400	850
14	Notebook Inspiron 15 5000 4GB	6	DELL	NOT-DL-000012	3100	1209
15	Notebook IdeaPad RF32000	6	DELL	NOT-DL-77164I	4200	1176
16	Notebook Motion Ultra 2	6	SAMSUNG	NOT-SS-13139U	2900	1160

# FILTROS DE TEXTOS

000



## TEXTOS

Podemos aplicar filtros em colunas de texto.

Para isso, basta utilizar o sinal de = e especificar o texto que deseja usar como critério do filtro (ou o <> caso queira apenas o que for diferente de).

```
1  
2 -- Mostra apenas os produtos da marca DELL  
3 • SELECT *  
4 FROM produtos  
5 WHERE Marca_Produto = 'DELL';
```

ID_Produto	Nome_Produto	ID_Categoria	Marca_Produto	Num_Serie	Preco_Unit	Custo_Unit
1	Monitor LED 19,5" Full HD HDMI	1	DELL	MNT-DL-831923	2300	966
4	Kit Tedado + Mouse sem fio Wireless	2	DELL	KTM-DL-041039	350	129.5
5	Kit Tedado + Mouse Slim Bluetooth	2	DELL	KTM-DL-111924	280	109.2
14	Notebook Inspiron 15 5000 4GB	6	DELL	NOT-DL-000012	3100	1209
15	Notebook IdeaPad RF32000	6	DELL	NOT-DL-77164I	4200	1176

# FILTROS DE DATA

000



## DATAS

Podemos aplicar filtros em colunas de data.

Para isso, basta utilizar os sinais lógicos como =, <, >, <=, >=, <>.

```
1  
2 -- Mostra apenas os pedidos feitos no dia 03/01/2019  
3 • SELECT *  
4 FROM pedidos  
5 WHERE Data_Venda = '2019-01-03';
```

ID_Pedido	Data_Venda	ID_Loja	ID_Produto	ID_Cliente	Qtd_Vendida	Receita_Venda	Custo_Venda	Custo_Unit	Preco_Unit
16	2019-01-03	8	4	33	1	350	129.5	129.5	350
17	2019-01-03	8	4	36	1	350	129.5	129.5	350
18	2019-01-03	1	4	37	1	350	129.5	129.5	350
19	2019-01-03	6	4	42	1	350	129.5	129.5	350
20	2019-01-03	8	4	43	1	350	129.5	129.5	350
21	2019-01-03	6	4	44	1	350	129.5	129.5	350
88	2019-01-03	6	8	29	1	600	258	258	600
89	2019-01-03	5	8	46	1	600	258	258	600
90	2019-01-03	2	8	86	1	600	258	258	600
91	2019-01-03	4	8	12	1	600	258	258	600
92	2019-01-03	4	8	82	1	600	258	258	600
93	2019-01-03	3	8	41	1	600	258	258	600

# MÚLTIPLOS FILTROS COM AND E OR

000



## OPERADOR AND

Podemos aplicar mais de um filtro usando o **AND**.

Com ele, todas as condições devem ser satisfeitas para que o resultado seja mostrado.

```
1  
2 -- Mostra apenas os clientes SOLTEIROS do sexo MASCULINO  
3 • SELECT *  
4 FROM clientes  
5 WHERE Estado_Civil = 'S' AND Sexo = 'M';
```

ID_Cliente	Nome	Sobrenome	Data_Nascimento	Estado_Civil	Sexo	Email	Telefone	Renda_Anual	Qtd_Filhos	Escolaridade
4	Julio	Ruiz	1985-07-31	S	M	julio1@hotmail.com	(62) 93391-5891	70000	0	Pós-graduado
8	Shannon	Carlson	1984-03-27	S	M	shannon38@gmail.com	(85) 96795-9950	70000	0	Pós-graduado
17	Clarence	Rai	1964-10-04	S	M	clarence32@outlook.com	(21) 98923-7805	30000	2	Parcial
18	Jordan	King	1998-09-15	S	M	jordan73@gmail.com	(31) 99592-6279	40000	0	Ensino médio
24	Harold	Sai	1966-03-29	S	M	harold3@hotmail.com	(31) 95052-6286	30000	2	Parcial
43	Leonard	Nara	1970-05-14	S	M	leonard18@outlook.com	(31) 93245-4616	30000	3	Ensino médio
49	Daniel	Johnson	1971-07-30	S	M	daniel18@gmail.com	(31) 98422-3549	30000	3	Ensino médio
65	Caleb	Carter	1996-09-20	S	M	caleb40@gmail.com	(31) 97809-1800	60000	0	Parcial
72	Levi	Arun	1976-08-23	S	M	levi6@gmail.com	(71) 92754-9983	70000	2	Ensino médio
74	Blake	Anderson	1977-07-08	S	M	blake9@gmail.com	(11) 98232-2736	80000	2	Ensino médio
77	Donald	Gonzalez	1979-03-06	S	M	donald20@gmail.com	(31) 96897-9735	160000	0	Graduação
82	Lucas	Phillips	1977-09-07	S	M	lucas7@outlook.com	(62) 94668-3507	80000	2	Parcial
88	Trevor	Bryant	1977-12-12	S	M	trevor18@gmail.com	NULL	90000	2	Parcial
92	Cedric	Ma	1982-03-27	S	M	cedric15@outlook.com	(31) 95423-4764	70000	1	Parcial
93	Chad	Kumar	1982-08-27	S	M	chad9@gmail.com	(21) 98828-7409	70000	1	Parcial

# MÚLTIPLOS FILTROS COM AND E OR

000



## OPERADOR OR

Podemos aplicar mais de um filtro usando o **OR**.

Com ele, apenas uma condição precisa ser satisfeita para que o resultado seja mostrado.

```
1
2 -- Mostra apenas os produtos das marcas DELL OU SAMSUNG
3 • SELECT *
4 FROM produtos
5 WHERE Marca_Produto = 'DELL' OR Marca_Produto = 'SAMSUNG';
```

ID_Produto	Nome_Produto	ID_Categoria	Marca_Produto	Num_Serie	Preco_Unit	Custo_Unit
1	Monitor LED 19,5" Full HD HDMI	1	DELL	MNT-DL-831923	2300	966
2	Monitor Curvo 24" 144Hz HDMI	1	SAMSUNG	MNT-SS-001939	2800	980
4	Kit Teclado + Mouse sem fio Wireless	2	DELL	KTM-DL-041039	350	129.5
5	Kit Teclado + Mouse Slim Bluetooth	2	DELL	KTM-DL-111924	280	109.2
13	Notebook LC2100 Intel Core i5 8GB	6	SAMSUNG	NOT-SS-918457	3400	850
14	Notebook Inspiron 15 5000 4GB	6	DELL	NOT-DL-000012	3100	1209
15	Notebook IdeaPad RF32000	6	DELL	NOT-DL-77164I	4200	1176
16	Notebook Motion Ultra 2	6	SAMSUNG	NOT-SS-13139U	2900	1160



## OPERADOR IN

Com o operador IN, podemos passar uma lista de valores que serão utilizados como filtro na nossa coluna.

```
86  -- Seleccione apenas os clientes das seguintes nacionalidades:  
87  -- Canadá, Porto Rico e Irlanda.  
88  
89 • SELECT *  
90   FROM atores  
91   WHERE nacionalidade IN ('Canadá', 'Porto Rico', 'Irlanda');  
92
```

	id_ator	nome_ator	ano_nascimento	nacionalidade	sexo
	11	Barry Pepper	1970	Canadá	masculino
	12	Benicio Del Toro	1967	Porto Rico	masculino
	29	Colin Farrell	1976	Irlanda	masculino
	35	Donald Sutherland	1935	Canadá	masculino
	81	Keanu Reeves	1964	Canadá	masculino
	112	Rachel McAdams	1978	Canadá	feminino
	134	Stephen Rea	1946	Irlanda	masculino

# FILTRO BETWEEN

000



## BETWEEN

Com o operador IN, podemos passar uma lista de valores que serão utilizados como filtro na nossa coluna.

```
86    -- Seleione apenas os clientes que nasceram entre os anos de:  
87    -- 1980 e 1983.  
88  
89 •  SELECT *  
90   FROM atores  
91   WHERE ano_nascimento BETWEEN 1980 AND 1983;  
92
```

	id_ator	nome_ator	ano_nascimento	nacionalidade	sexo
▶	1	Abbie Cornish	1982	Austrália	feminino
	5	Andrea Riseborough	1981	Reino Unido	feminino
	8	Anne Hathaway	1982	EUA	feminino
	18	Bryce Dallas Howard	1981	EUA	feminino
	22	Channing Tatum	1980	EUA	masculino
	26	Christina Ricci	1980	EUA	feminino
	37	Eddie Redmayne	1982	Reino Unido	masculino
	40	Elijah Wood	1981	EUA	masculino
	70	Jesse Eisenberg	1983	EUA	masculino
	84	Kristen Bell	1980	EUA	feminino
	94	Maryam Karimi	1980	Afeganistão	feminino
	102	Natalie Portman	1981	Israel	feminino
	124	Sam Riley	1980	Reino Unido	masculino
	145	Zooey Deschanel	1980	EUA	feminino

# FILTRO LIKE/NOT LIKE

000



## LIKE e NOT LIKE

Com o operador LIKE, podemos filtrar textos que contenham determinada cadeia de caracteres.

```
1 -- WHERE LIKE: filtro de 'contém'  
2 • SELECT *  
3 FROM clientes  
4 WHERE Email LIKE '%gmail%';
```

ID_Cliente	Nome	Sobrenome	Data_Nascimento	Estado_Civil	Sexo	Email	Telefone	Renda_Anual	Qtd_Filhos	Escolaridade
3	Elizabeth	Johnson	1988-08-03	S	F	elizabeth5@gmail.com	(31) 92039-5832	80000	5	Pós-graduado
5	Janet	Alvarez	1985-12-01	S	F	janet9@gmail.com	(21) 93379-3743	70000	0	Pós-graduado
8	Shannon	Carlson	1984-03-27	S	M	shannon38@gmail.com	(85) 96795-9950	70000	0	Pós-graduado
12	Ian	Jenkins	1988-08-01	C	M	ian47@gmail.com	(21) 95385-3393	100000	2	Pós-graduado
13	Sydney	Bennett	1988-05-04	S	F	sydney23@gmail.com	(11) 92024-9186	100000	3	Pós-graduado
15	Wyatt	Hill	1999-04-23	C	M	wyatt32@gmail.com	(71) 98644-2695	70000	0	Parcial
18	Jordan	King	1998-09-15	S	M	jordan73@gmail.com	(31) 99591-0000	100000	0	Ensino médio
21	Seth	Edwards	1998-10-06	C	M	seth46@gmail.com	(41) 97012-3526	40000	0	Parcial
23	Alejandro	Beck	1965-12-18	C	M	alejandro45@gmail.com	(85) 94492-9582	10000	2	Parcial
26	Jill	Jimenez	1966-04-06	C	F	jill13@gmail.com	(41) 97120-5603	30000	2	Parcial
32	Ebony	Gonzalez	1967-06-14	C	F	ebony19@gmail.com	(71) 99664-2156	20000	4	Ensino médio
33	Wendy	Dominguez	1968-02-19	C	F	wendy12@gmail.com	(31) 97278-7135	10000	2	Parcial
35	Chloe	Garcia	1997-11-22	S	F	chloe27@gmail.com	(21) 94134-4552	40000	0	Parcial
36	Diana	Hernandez	1968-03-18	C	F	diana2@gmail.com	(85) 93094-3166	10000	2	Parcial

# SQL

## Operações Matemáticas e Funções de Agregação

# Operações matemáticas

É muito simples fazer operações matemáticas no SQL. Ao lado, temos uma sequência de exemplos que vão desde a soma até o cálculo do resto de uma divisão, usando o operador %.

```

1 -- Operações matemáticas
2 • SELECT
3      10 + 20      AS 'Soma',
4      150 - 50      AS 'Subtração',
5      300 * 3      AS 'Multiplicação',
6      450 / 5      AS 'Divisão',
7      (2 + 3) * 5  AS 'Combinação',
8      22 % 5       AS 'Resto da divisão'

```

	Soma	Subtração	Multiplicação	Divisão	Combinação	Resto da divisão
▶	30	100	900	90.0000	25	2

# Funções matemáticas

As funções matemáticas são extremamente importantes para criação de **cálculos no SQL**. Eles serão a base para as análises que faremos com agrupamentos e GROUP BY!

## Funções Matemáticos

COUNT

SUM

MIN/MAX

AVG

# Funções matemáticas

Essas funções têm como objetivo realizar cálculos no SQL, tais como: contagem, soma, média, mínimo e máximo.

[COUNT](#)[COUNT\(\\*\)](#)[COUNT\(DISTINCT\)](#)[SUM](#)[AVG](#)[MIN/MAX](#)

# COUNT

## OBJETIVO

Retorna a **quantidade total de valores** de uma coluna.

## CÓDIGO

```
1   -- COUNT  
2 • SELECT  
3       COUNT(Nome)  
4 FROM clientes;
```

## RESULTADO

	COUNT(Nome)
▶	100

# COUNT

## OBJETIVO

Retorna a **quantidade total de valores** de uma coluna.

## CÓDIGO

```
1 -- COUNT  
2 • SELECT  
3     COUNT(Telefone)  
4 FROM clientes;
```

## RESULTADO

	COUNT(Telefone)
▶	94



O COUNT ignora os valores nulos de uma coluna. Por isso o resultado pode mudar dependendo da coluna escolhida.

# COUNT(\*)

## OBJETIVO

Retorna a **quantidade total de linhas** de uma tabela.

**Obs:** não ignora valores nulos

## CÓDIGO

```
1   -- COUNT  
2 • SELECT  
3       COUNT(*)  
4   FROM clientes;
```

## RESULTADO

	COUNT(*)
▶	100

# COUNT(DISTINCT)

## OBJETIVO

Retorna a **contagem distinta** de valores de uma tabela.

## CÓDIGO

```
1 -- COUNT(DISTINCT)
2 • SELECT
3     COUNT(DISTINCT Escolaridade)
4 FROM clientes;
```

## RESULTADO

COUNT(DISTINCT Escolaridade)
4

# SUM

## OBJETIVO

Retorna a **soma total** dos valores de uma coluna.

## CÓDIGO

```
1 -- SUM  
2 • SELECT  
3     SUM(Receita_Venda)  
4 FROM pedidos;
```

## RESULTADO

SUM(Receita_Venda)
228900

# AVG

## OBJETIVO

Retorna a **média** dos **valores** de uma coluna.

## CÓDIGO

```
1 -- AVG  
2 • SELECT  
3     AVG(Receita_Venda)  
4 FROM pedidos;
```

## RESULTADO

	AVG(Receita_Venda)
▶	612.0321

# MAX e MIN

## OBJETIVO

Retorna o **valor máximo e mínimo** de uma coluna.

## CÓDIGO

```
1 -- MAX
2 • SELECT
3     MAX(Receita_Venda)
4 FROM pedidos;
```

## RESULTADO

	MAX(Receita_Venda)
▶	1800

# SQL

## Criando agrupamentos



# GROUP BY

000

O comando GROUP BY permite agrupar os dados de uma tabela, criando uma espécie de resumo dos dados.

Data Venda	Produto	Quantidade
2019-01-01	PRODUTO A	10
2019-01-01	PRODUTO B	20
2019-01-01	PRODUTO A	5
2019-01-01	PRODUTO A	15
2019-01-01	PRODUTO B	10
2019-01-01	PRODUTO A	30

GROUP BY



Produto	Quantidade Total
PRODUTO A	60
PRODUTO B	30

# GROUP BY

000

O comando GROUP BY permite agrupar os dados de uma tabela, criando uma espécie de resumo dos dados.

Data Venda	Produto	Quantidade
2019-01-01	PRODUTO A	10
2019-01-01	PRODUTO B	20
2019-01-01	PRODUTO A	5
2019-01-01	PRODUTO A	15
2019-01-01	PRODUTO B	10
2019-01-01	PRODUTO A	30

GROUP BY



Produto	Quantidade Total
PRODUTO A	60
PRODUTO B	30

# GROUP BY

000

O comando GROUP BY permite agrupar os dados de uma tabela, criando uma espécie de resumo dos dados.

Data Venda	Produto	Quantidade
2019-01-01	PRODUTO A	10
2019-01-01	PRODUTO B	20
2019-01-01	PRODUTO A	5
2019-01-01	PRODUTO A	15
2019-01-01	PRODUTO B	10
2019-01-01	PRODUTO A	30

GROUP BY



Produto	Quantidade Total
PRODUTO A	60
PRODUTO B	30

# GROUP BY

000

O comando GROUP BY permite agrupar os dados de uma tabela, criando uma espécie de resumo dos dados.

Data Venda	Produto	Quantidade
2019-01-01	PRODUTO A	10
2019-01-01	PRODUTO B	20
2019-01-01	PRODUTO A	5
2019-01-01	PRODUTO A	15
2019-01-01	PRODUTO B	10
2019-01-01	PRODUTO A	30

GROUP BY

SELECT Produto,  
SUM(Quantidade) AS  
'Quantidade Total'  
FROM Tabela  
GROUP BY Produto;

Produto	Quantidade Total
PRODUTO A	60
PRODUTO B	30



## GROUP BY

O **Group By** é o comando do SQL que vai nos permitir **criar agrupamentos**, ou seja, tabelas resumos das nossas tabelas principais.

```
1 # GROUP BY  
2  
3 -- Exemplo: Utilize o GROUP BY para criar uma  
4 -- consulta e descobrir o total de filmes por gênero.  
5 • SELECT  
6     genero,  
7     COUNT(*)  
8 FROM filmes  
9 GROUP BY genero;
```

genero	COUNT(*)
Comédia	10
Drama	36
Ficção e Fantasia	11
Mistério e Suspense	6
Arte	1
Animação	2
Ação e Aventura	5

# FILTRANDO AGRUPAMENTOS (WHERE)



## GROUP BY + WHERE

Sempre que quisermos realizar um filtro **ANTES** de criar o agrupamento, usamos o WHERE.

```
1 # GROUP BY + WHERE
2
3 -- Exemplo: Utilize o GROUP BY + WHERE para criar uma
4 -- consulta e descobrir o total de filmes por gênero, mas
5 -- considerando apenas os filmes lançados em 2003.
6 • SELECT
7     genero,
8     COUNT(*)
9 FROM filmes
10 WHERE ano_lancamento = 2003
11 GROUP BY genero;
```

genero	COUNT(*)
Mistério e Suspense	2
Comédia	2
Drama	3
Ficção e Fantasia	1

# FILTRANDO AGRUPAMENTOS (HAVING)



## GROUP BY + HAVING

Sempre que quisermos realizar um filtro **DEPOIS** que o agrupamento foi criado, usamos o HAVING.

```

86 # GROUP BY + HAVING
87
88 -- Exemplo: Utilize o GROUP BY + HAVING para filtrar a tabela resultante e mostrar
89 -- apenas os filmes dos gêneros com quantidade MAIOR OU IGUAL A 10 filmes.
90
91 • SELECT
92     genero,
93     COUNT(*) AS 'qtd_filmes'
94 FROM filmes
95 GROUP BY genero
96 HAVING COUNT(*) >= 10;
97
98

```

	genero	qtd_filmes
▶	Comédia	10
	Drama	36
	Ficção e Fantasia	11

# WHERE vs HAVING

Como o **WHERE** funciona?

1

Partimos da tabela **original**

	id_filme	titulo	genero	duracao	ano_lancamento	preco_aluguel
▶	1	Que mulher é essa?	Comédia	93	2001	2.09
2	2	A Senha	Drama	99	2001	2.19
3	3	Do que as mulheres gostam	Comédia	127	2001	2.59
4	4	Dia de Treinamento	Drama	122	2001	1.79
5	5	O Senhor dos Anéis: A sociedade do anel	Ficção e Fantasia	178	2001	2.59
6	6	Harry Potter e a Pedra Filosofal	Ficção e Fantasia	152	2001	2.69
7	7	Os Excêntricos Tenenbaums	Comédia	110	2002	1.89
8	8	Seu marido e minha mulher	Comédia	91	2002	2.59
9	9	11 de setembro	Drama	134	2002	2.99
10	10	Simone	Drama	117	2002	2.69
11	11	É hora do show	Comédia	95	2002	1.79
12	12	O Senhor dos Anéis: As duas torres	Ficção e Fantasia	179	2002	2.39
13	13	Harry Potter e a Câmara Secreta	Ficção e Fantasia	161	2002	1.79
14	14	O Novato	Mistério e Suspense	115	2003	1.69
15	15	Alguém tem que ceder	Comédia	128	2003	1.69
16	16	A última noite	Drama	135	2003	1.59
17	17	Revelações	Mistério e Suspense	106	2003	1.99
18	18	Lições Para Toda a Vida	Drama	111	2003	1.69
19	19	21 gramas	Drama	124	2003	2.09
20	20	Simplesmente amor	Comédia	135	2003	2.29
21	21	O Senhor dos Anéis: O retorno do rei	Ficção e Fantasia	200	2003	1.99
22	22	Visões	Arte	107	2004	2.59
23	23	Dança comigo	Drama	106	2004	1.69
24	24	Uma Eleição Muito Atrapalhada	Comédia	110	2004	2.89
25	25	Bridget Jones: No Limite da Razão	Drama	108	2004	2.89
26	26	Ray	Drama	152	2004	2.59
27	27	Monster - Desejo Assassino	Drama	109	2004	2.09
28	28	Harry Potter e o Prisioneiro de Azkaban	Ficção e Fantasia	142	2004	1.69
29	29	Tudo por Dinheiro	Drama	122	2005	2.79
30	30	Capote	Drama	114	2005	2.39
31	31	Harry Potter e o Cálice de Fogo	Ficção e Fantasia	157	2005	2.69
32	32	Falsária	Drama	93	2006	2.89
33	33	V de Vingança	Drama	132	2006	1.59
34	34	Armações do Amor	Drama	97	2006	1.99
35	35	Happy Feet	Animação	108	2006	1.79
36	36	As Torres Gêmeas	Drama	129	2006	1.59

```
SELECT
    genero,
    COUNT(*)
FROM filmes
WHERE ano_lancamento = 2003
GROUP BY genero;
```

2

Usamos o GROUP BY + WHERE

	id_filme	titulo	genero	duracao	ano_lancamento	preco_aluguel
▶	1	Que mulher é essa?	Comédia	93	2001	2.09
2	2	A Senha	Drama	99	2001	2.19
3	3	Do que as mulheres gostam	Comédia	127	2001	2.59
4	4	Dia de Treinamento	Drama	122	2001	1.79
5	5	O Senhor dos Anéis: A sociedade do anel	Ficção e Fantasia	178	2001	2.59
6	6	Harry Potter e a Pedra Filosofal	Ficção e Fantasia	152	2001	2.69
7	7	Os Excêntricos Tenenbaums	Comédia	110	2002	1.89
8	8	Seu marido e minha mulher	Comédia	91	2002	2.59
9	9	11 de setembro	Drama	134	2002	2.99
10	10	Simone	Drama	117	2002	2.69
11	11	É hora do show	Comédia	95	2002	1.79
12	12	O Senhor dos Anéis: As duas torres	Ficção e Fantasia	179	2002	2.39
13	13	Harry Potter e a Câmara Secreta	Ficção e Fantasia	161	2002	1.79
14	14	O Novato	Mistério e Suspense	115	2003	1.69
15	15	Alguém tem que ceder	Comédia	128	2003	1.69
16	16	A última noite	Drama	135	2003	1.59
17	17	Revelações	Mistério e Suspense	106	2003	1.99
18	18	Lições Para Toda a Vida	Drama	111	2003	1.69
19	19	21 gramas	Drama	124	2003	2.09
20	20	Simplesmente amor	Comédia	135	2003	2.29
21	21	O Senhor dos Anéis: O retorno do rei	Ficção e Fantasia	200	2003	1.99
22	22	Visões	Arte	107	2004	2.59
23	23	Dança comigo	Drama	106	2004	1.69
24	24	Uma Eleição Muito Atrapalhada	Comédia	110	2004	2.89
25	25	Bridget Jones: No Limite da Razão	Drama	108	2004	2.89
26	26	Ray	Drama	152	2004	2.59
27	27	Monster - Desejo Assassino	Drama	109	2004	2.09
28	28	Harry Potter e o Prisioneiro de Azkaban	Ficção e Fantasia	142	2004	1.69
29	29	Tudo por Dinheiro	Drama	122	2005	2.79
30	30	Capote	Drama	114	2005	2.39
31	31	Harry Potter e o Cálice de Fogo	Ficção e Fantasia	157	2005	2.69
32	32	Falsária	Drama	93	2006	2.89
33	33	V de Vingança	Drama	132	2006	1.59
34	34	Armações do Amor	Drama	97	2006	1.99
35	35	Happy Feet	Animação	108	2006	1.79
36	36	As Torres Gêmeas	Drama	129	2006	1.59

3

Apenas os filmes do ano de 2003 são considerados

E o agrupamento final de gêneros considera apenas os filmes de 2003

	genero	COUNT(*)
▶	Mistério e Suspense	2
2	Comédia	2
3	Drama	3
4	Ficção e Fantasia	1

# WHERE vs HAVING

Como o **HAVING** funciona?

1

Partimos da tabela **agrupada**

	genero	COUNT(*)
▶	Comédia	10
	Drama	36
	Ficção e Fantasia	11
	Mistério e Suspense	6
	Arte	1
	Animação	2
	Ação e Aventura	5

2

Usamos o GROUP BY + HAVING

```
SELECT
    genero,
    COUNT(*) AS 'qtd_filmes'
FROM filmes
GROUP BY genero
HAVING COUNT(*) >= 10;
```

3

Apenas os gêneros com  
quantidade MAIOR OU IGUAL  
são considerados

	genero	COUNT(*)
▶	Comédia	10
	Drama	36
	Ficção e Fantasia	11
	Mistério e Suspense	6
	Arte	1
	Animação	2
	Ação e Aventura	5

4

E o agrupamento final de  
gêneros considera apenas os  
gêneros com mais de 10 filmes

	genero	qtd_filmes
▶	Comédia	10
	Drama	36
	Ficção e Fantasia	11

# SQL

## Variáveis



# O que é uma variável

Uma variável é um local onde armazenamos um determinado valor, que pode ser usado ao longo do nosso código.

# O que é uma variável

Uma variável é um local onde armazenamos um determinado valor, que pode ser usado ao longo do nosso código.

Por exemplo, observe a query abaixo:

- `SELECT *`  
`FROM produtos`  
`WHERE Marca_Produto = 'SAMSUNG';`

# O que é uma variável

Uma variável é um local onde armazenamos um determinado valor, que pode ser usado ao longo do nosso código.

Poderíamos utilizar variáveis para deixar facilitar.

```
· SET @varMarca = 'SAMSUNG';  
  
· SELECT *  
  FROM produtos  
  WHERE Marca_Produto = @varMarca;
```

# SQL JOINS



# Chave Primária vs. Chave Estrangeira

Uma **Chave Primária** é uma coluna que identifica as informações distintas em uma tabela. Geralmente é uma coluna de ID. Toda tabela terá uma, e somente uma, chave primária. Essa chave é utilizada como identificador único da tabela, sendo representada por uma coluna que não receberá valores repetidos.

# Chave Primária vs. Chave Estrangeira

Uma **Chave Primária** é uma coluna que identifica as informações distintas em uma tabela. Geralmente é uma coluna de ID. Toda tabela terá uma, e somente uma, chave primária. Essa chave é utilizada como identificador único da tabela, sendo representada por uma coluna que não receberá valores repetidos.

Como pode ser visto abaixo, a tabela **Produtos** possui uma coluna chamada **ID\_Produto**, com valores que não se repetem. Essa será a **Chave Primária**.

Já na tabela de **Pedidos**, temos uma coluna chamada **ID\_Pedido**, e essa também não se repete. Entendemos essa coluna também como uma chave primária.

produtos

ID_Produto	Nome_Produto	ID_Categoria	Marca_Produto	Num_Serie	Preco_Unit	Custo_Unit
1	Monitor LED 19,5" Full HD HDMI	1	DELL	MNT-DL-831923	2300	966
2	Monitor Curvo 24" 144Hz HDMI	1	SAMSUNG	MNT-SS-001939	2800	980
3	Webcam Full HD 1080p	1	LOGITECH	WBC-LT-934GG4	450	90
4	Kit Teclado + Mouse sem fio Wireless	2	DELL	KTM-DL-041039	350	129.5
5	Kit Teclado + Mouse Slim Bluetooth	2	DELL	KTM-DL-111924	280	109.2
6	Cadeira Gamer reclinável Azul/Laranja	3	ALTURA	CGM-AL-9N914J	1800	540
7	Cadeira Gamer PC Racer Vermelha	3	ALTURA	CGM-AL-0147FI	3100	1395
8	Headphone Bluetooth 2000	4	SONY	HDP-SN-194821	600	258
9	Fone de Ouvido Tune T5000	4	JBL	HDP-JB-091934	780	327.6
10	Microfone Condensador MC1000	5	AKG	MIC-AK-237591	1100	275
11	Microfone Condensador com Tripé	5	BLUE	MIC-BL-819455	800	344
12	Microfone de mesa com fio condensa...	5	BLUE	MIC-BL-761411	650	214.5
13	Notebook LC2100 Intel Core i5 8GB	6	SAMSUNG	NOT-SS-918457	3400	850
14	Notebook Inspiron 15 5000 4GB	6	DELL	NOT-DL-000012	3100	1209
15	Notebook IdeaPad RF32000	6	DELL	NOT-DL-77164I	4200	1176
16	Notebook Motion Ultra 2	6	SAMSUNG	NOT-SS-13139U	2900	1160

pedidos

ID_Pedido	Data_Venda	ID_Loja	ID_Produto	ID_Cliente	Qtd_Vendida	Preco_Unit	Custo_Unit	Receita_Venda	Custo_Venda
1	2019/01/01	6	11	45	1	800	344	800	344
2	2019/01/01	2	11	84	1	800	344	800	344
3	2019/01/01	1	14	12	1	3100	1209	3100	1209
4	2019/01/01	8	9	98	1	780	327.6	780	327.6
5	2019/01/01	5	8	21	1	600	258	600	258
6	2019/01/01	6	8	26	1	600	258	600	258
7	2019/01/01	6	13	90	1	3400	850	3400	850
8	2019/01/01	3	3	73	1	450	90	450	90
9	2019/01/01	5	15	17	1	4200	1176	4200	1176
10	2019/01/01	1	4	27	1	350	129.5	350	129.5
11	2019/01/01	5	9	25	1	780	327.6	780	327.6
12	2019/01/01	5	4	75	1	350	129.5	350	129.5
13	2019/01/01	5	11	48	1	800	344	800	344
14	2019/01/01	5	16	8	1	2900	1160	2900	1160
15	2019/01/01	5	9	41	1	780	327.6	780	327.6
16	2019/01/01	8	1	6	1	2300	966	2300	966
17	2019/01/01	6	14	77	1	3100	1209	3100	1209
18	2019/01/01	5	15	55	1	4200	1176	4200	1176
19	2019/01/01	5	3	65	1	450	90	450	90
20	2019/01/01	7	16	16	1	2900	1160	2900	1160
21	2019/01/01	5	11	55	1	800	344	800	344
22	2019/01/01	6	9	36	1	780	327.6	780	327.6
23	2019/01/01	3	4	42	1	350	129.5	350	129.5
24	2019/01/01	5	12	81	1	650	214.5	650	214.5
25	2019/01/01	7	1	77	1	2300	966	2300	966

# Chave Primária vs. Chave Estrangeira

Uma **Chave Primária** é uma coluna que identifica as informações distintas em uma tabela. Geralmente é uma coluna de ID. Toda tabela terá uma, e somente uma, chave primária. Essa chave é utilizada como identificador único da tabela, sendo representada por uma coluna que não receberá valores repetidos.

Como pode ser visto abaixo, a tabela **Produtos** possui uma coluna chamada **ID\_Produto**, com valores que não se repetem. Essa será a **Chave Primária**.

Já na tabela de **Pedidos**, temos uma coluna chamada **ID\_Pedido**, e essa também não se repete. Entendemos essa coluna também como uma chave primária.

produtos

ID_Produto	Nome_Produto	ID_Categoria	Marca_Produto	Num_Serie	Preco_Unit	Custo_Unit
1	Monitor LED 19,5" Full HD HDMI	1	DELL	MNT-DL-831923	2300	966
2	Monitor Curvo 24" 144Hz HDMI	1	SAMSUNG	MNT-SS-001939	2800	980
3	Webcam Full HD 1080p	1	LOGITECH	WBC-LT-934GG4	450	90
4	Kit Teclado + Mouse sem fio Wireless	2	DELL	KTM-DL-041039	350	129.5
5	Kit Teclado + Mouse Slim Bluetooth	2	DELL	KTM-DL-111924	280	109.2
6	Cadeira Gamer reclinável Azul/Laranja	3	ALTURA	CGM-AL-9N914J	1800	540
7	Cadeira Gamer PC Racer Vermelha	3	ALTURA	CGM-AL-0147FI	3100	1395
8	Headphone Bluetooth 2000	4	SONY	HDP-SN-194821	600	258
9	Fone de Ouvido Tune T5000	4	JBL	HDP-JB-091934	780	327.6
10	Microfone Condensador MC1000	5	AKG	MIC-AK-237591	1100	275
11	Microfone Condensador com Tripé	5	BLUE	MIC-BL-819455	800	344
12	Microfone de mesa com fio condensa...	5	BLUE	MIC-BL-761411	650	214.5
13	Notebook LC2100 Intel Core i5 8GB	6	SAMSUNG	NOT-SS-918457	3400	850
14	Notebook Inspiron 15 5000 4GB	6	DELL	NOT-DL-000012	3100	1209
15	Notebook IdeaPad RF32000	6	DELL	NOT-DL-77164I	4200	1176
16	Notebook Motion Ultra 2	6	SAMSUNG	NOT-SS-13139U	2900	1160

Chave Primária



pedidos

ID_Pedido	Data_Venda	ID_Loja	ID_Produto	ID_Cliente	Qtd_Vendida	Preco_Unit	Custo_Unit	Receita_Venda	Custo_Venda
1	2019/01/01	6	11	45	1	800	344	800	344
2	2019/01/01	2	11	84	1	800	344	800	344
3	2019/01/01	1	14	12	1	3100	1209	3100	1209
4	2019/01/01	8	9	98	1	780	327.6	780	327.6
5	2019/01/01	5	8	21	1	600	258	600	258
6	2019/01/01	6	8	26	1	600	258	600	258
7	2019/01/01	6	13	90	1	3400	850	3400	850
8	2019/01/01	3	3	73	1	450	90	450	90
9	2019/01/01	5	15	17	1	4200	1176	4200	1176
10	2019/01/01	1	4	27	1	350	129.5	350	129.5
11	2019/01/01	5	9	25	1	780	327.6	780	327.6
12	2019/01/01	5	4	75	1	350	129.5	350	129.5
13	2019/01/01	5	11	48	1	800	344	800	344
14	2019/01/01	5	16	8	1	2900	1160	2900	1160
15	2019/01/01	5	9	41	1	780	327.6	780	327.6
16	2019/01/01	8	1	6	1	2300	966	2300	966
17	2019/01/01	6	14	77	1	3100	1209	3100	1209
18	2019/01/01	5	15	55	1	4200	1176	4200	1176
19	2019/01/01	5	3	65	1	450	90	450	90
20	2019/01/01	7	16	16	1	2900	1160	2900	1160
21	2019/01/01	5	11	55	1	800	344	800	344
22	2019/01/01	6	9	36	1	780	327.6	780	327.6
23	2019/01/01	3	4	42	1	350	129.5	350	129.5
24	2019/01/01	5	12	81	1	650	214.5	650	214.5
25	2019/01/01	7	1	77	1	2300	966	2300	966

Chave Primária



# Chave Primária vs. Chave Estrangeira

000

Já uma **Chave Estrangeira** é uma coluna que permite relacionar as linhas de uma segunda tabela com a **Chave Primária** de uma primeira tabela.

produtos

ID_Produto	Nome_Produto	ID_Categoria	Marca_Produto	Num_Serie	Preco_Unit	Custo_Unit
1	Monitor LED 19,5" Full HD HDMI	1	DELL	MNT-DL-831923	2300	966
2	Monitor Curvo 24" 144Hz HDMI	1	SAMSUNG	MNT-SS-001939	2800	980
3	Webcam Full HD 1080p	1	LOGITECH	WBC-LT-934GG4	450	90
4	Kit Teclado + Mouse sem fio Wireless	2	DELL	KTM-DL-041039	350	129.5
5	Kit Teclado + Mouse Slim Bluetooth	2	DELL	KTM-DL-111924	280	109.2
6	Cadeira Gamer reclinável Azul/Laranja	3	ALTURA	CGM-AL-9N914J	1800	540
7	Cadeira Gamer PC Racer Vermelha	3	ALTURA	CGM-AL-0147FI	3100	1395
8	Headphone Bluetooth 2000	4	SONY	HDP-SN-194821	600	258
9	Fone de Ouvido Tune T5000	4	JBL	HDP-JB-091934	780	327.6
10	Microfone Condensador MC1000	5	AKG	MIC-AK-237591	1100	275
11	Microfone Condensador com Tripé	5	BLUE	MIC-BL-819455	800	344
12	Microfone de mesa com fio condensa...	5	BLUE	MIC-BL-761411	650	214.5
13	Notebook LC2100 Intel Core i5 8GB	6	SAMSUNG	NOT-SS-918457	3400	850
14	Notebook Inspiron 15 5000 4GB	6	DELL	NOT-DL-000012	3100	1209
15	Notebook IdeaPad RF32000	6	DELL	NOT-DL-77164I	4200	1176
16	Notebook Motion Ultra 2	6	SAMSUNG	NOT-SS-13139U	2900	1160

pedidos

ID_Pedido	Data_Venda	ID_Loja	ID_Produto	ID_Cliente	Qtd_Vendida	Preco_Unit	Custo_Unit	Receita_Venda	Custo_Venda
1	2019/01/01	6	11	45	1	800	344	800	344
2	2019/01/01	2	11	84	1	800	344	800	344
3	2019/01/01	1	14	12	1	3100	1209	3100	1209
4	2019/01/01	8	9	98	1	780	327.6	780	327.6
5	2019/01/01	5	8	21	1	600	258	600	258
6	2019/01/01	6	8	26	1	600	258	600	258
7	2019/01/01	6	13	90	1	3400	850	3400	850
8	2019/01/01	3	3	73	1	450	90	450	90
9	2019/01/01	5	15	17	1	4200	1176	4200	1176
10	2019/01/01	1	4	27	1	350	129.5	350	129.5
11	2019/01/01	5	9	25	1	780	327.6	780	327.6
12	2019/01/01	5	4	75	1	350	129.5	350	129.5
13	2019/01/01	5	11	48	1	800	344	800	344
14	2019/01/01	5	16	8	1	2900	1160	2900	1160
15	2019/01/01	5	9	41	1	780	327.6	780	327.6
16	2019/01/01	8	1	6	1	2300	966	2300	966
17	2019/01/01	6	14	77	1	3100	1209	3100	1209
18	2019/01/01	5	15	55	1	4200	1176	4200	1176
19	2019/01/01	5	3	65	1	450	90	450	90
20	2019/01/01	7	16	16	1	2900	1160	2900	1160
21	2019/01/01	5	11	55	1	800	344	800	344
22	2019/01/01	6	9	36	1	780	327.6	780	327.6
23	2019/01/01	3	4	42	1	350	129.5	350	129.5
24	2019/01/01	5	12	81	1	650	214.5	650	214.5
25	2019/01/01	7	1	77	1	2300	966	2300	966

# Chave Primária vs. Chave Estrangeira

Já uma **Chave Estrangeira** é uma coluna que permite relacionar as linhas de uma segunda tabela com a **Chave Primária** de uma primeira tabela.

Como pode ser visto abaixo, a tabela **Produtos** possui uma coluna chamada **ID\_Produto**, com valores que não se repetem. Essa será a **Chave Primária**. Já na tabela de **Pedidos**, a coluna de **ID\_Produto** também aparece, mas os valores se repetem. Isso porque podemos ter mais de um pedido de um mesmo produto. Na tabela de **Pedidos**, a coluna de **ID\_Produto** vai ser a **Chave Estrangeira** e vai permitir a gente relacionar os valores dessa coluna com a **Chave Primária** da tabela de **Produtos**.

produtos

ID_Produto	Nome_Produto	ID_Categoria	Marca_Produto	Num_Serie	Preco_Unit	Custo_Unit
1	Monitor LED 19,5" Full HD HDMI	1	DELL	MNT-DL-831923	2300	966
2	Monitor Curvo 24" 144Hz HDMI	1	SAMSUNG	MNT-SS-001939	2800	980
3	Webcam Full HD 1080p	1	LOGITECH	WBC-LT-934GG4	450	90
4	Kit Teclado + Mouse sem fio Wireless	2	DELL	KTM-DL-041039	350	129.5
5	Kit Teclado + Mouse Slim Bluetooth	2	DELL	KTM-DL-111924	280	109.2
6	Cadeira Gamer reclinável Azul/Laranja	3	ALTURA	CGM-AL-9N914J	1800	540
7	Cadeira Gamer PC Racer Vermelha	3	ALTURA	CGM-AL-0147FI	3100	1395
8	Headphone Bluetooth 2000	4	SONY	HDP-SN-194821	600	258
9	Fone de Ouvido Tune T5000	4	JBL	HDP-JB-091934	780	327.6
10	Microfone Condensador MC1000	5	AKG	MIC-AK-237591	1100	275
11	Microfone Condensador com Tripé	5	BLUE	MIC-BL-819455	800	344
12	Microfone de mesa com fio condensa...	5	BLUE	MIC-BL-761411	650	214.5
13	Notebook LC2100 Intel Core i5 8GB	6	SAMSUNG	NOT-SS-918457	3400	850
14	Notebook Inspiron 15 5000 4GB	6	DELL	NOT-DL-000012	3100	1209
15	Notebook IdeaPad RF32000	6	DELL	NOT-DL-77164I	4200	1176
16	Notebook Motion Ultra 2	6	SAMSUNG	NOT-SS-13139U	2900	1160

pedidos

ID_Pedido	Data_Venda	ID_Loja	ID_Produto	ID_Cliente	Qtd_Vendida	Preco_Unit	Custo_Unit	Receita_Venda	Custo_Venda
1	2019/01/01	6	11	45	1	800	344	800	344
2	2019/01/01	2	11	84	1	800	344	800	344
3	2019/01/01	1	14	12	1	3100	1209	3100	1209
4	2019/01/01	8	9	98	1	780	327.6	780	327.6
5	2019/01/01	5	8	21	1	600	258	600	258
6	2019/01/01	6	8	26	1	600	258	600	258
7	2019/01/01	6	13	90	1	3400	850	3400	850
8	2019/01/01	3	3	73	1	450	90	450	90
9	2019/01/01	5	15	17	1	4200	1176	4200	1176
10	2019/01/01	1	4	27	1	350	129.5	350	129.5
11	2019/01/01	5	9	25	1	780	327.6	780	327.6
12	2019/01/01	5	4	75	1	350	129.5	350	129.5
13	2019/01/01	5	11	48	1	800	344	800	344
14	2019/01/01	5	16	8	1	2900	1160	2900	1160
15	2019/01/01	5	9	41	1	780	327.6	780	327.6
16	2019/01/01	8	1	6	1	2300	966	2300	966
17	2019/01/01	6	14	77	1	3100	1209	3100	1209
18	2019/01/01	5	15	55	1	4200	1176	4200	1176
19	2019/01/01	5	3	65	1	450	90	450	90
20	2019/01/01	7	16	16	1	2900	1160	2900	1160
21	2019/01/01	5	11	55	1	800	344	800	344
22	2019/01/01	6	9	36	1	780	327.6	780	327.6
23	2019/01/01	3	4	42	1	350	129.5	350	129.5
24	2019/01/01	5	12	81	1	650	214.5	650	214.5
25	2019/01/01	7	1	77	1	2300	966	2300	966

# Chave Primária vs. Chave Estrangeira

Já uma **Chave Estrangeira** é uma coluna que permite relacionar as linhas de uma segunda tabela com a **Chave Primária** de uma primeira tabela.

Como pode ser visto abaixo, a tabela **Produtos** possui uma coluna chamada **ID\_Produto**, com valores que não se repetem. Essa será a **Chave Primária**. Já na tabela de **Pedidos**, a coluna de **ID\_Produto** também aparece, mas os valores se repetem. Isso porque podemos ter mais de um pedido de um mesmo produto. Na tabela de **Pedidos**, a coluna de **ID\_Produto** vai ser a **Chave Estrangeira** e vai permitir a gente relacionar os valores dessa coluna com a Chave Primária da tabela de **Produtos**.

produtos						
ID_Produto	Nome_Produto	ID_Categoria	Marca_Produto	Num_Serie	Preco_Unit	Custo_Unit
1	Monitor LED 19,5" Full HD HDMI	1	DELL	MNT-DL-831923	2300	966
2	Monitor Curvo 24" 144Hz HDMI	1	SAMSUNG	MNT-SS-001939	2800	980
3	Webcam Full HD 1080p	1	LOGITECH	WBC-LT-934GG4	450	90
4	Kit Teclado + Mouse sem fio Wireless	2	DELL	KTM-DL-041039	350	129.5
5	Kit Teclado + Mouse Slim Bluetooth	2	DELL	KTM-DL-111924	280	109.2
6	Cadeira Gamer reclinável Azul/Laranja	3	ALTURA	CGM-AL-9N914J	1800	540
7	Cadeira Gamer PC Racer Vermelha	3	ALTURA	CGM-AL-0147FI	3100	1395
8	Headphone Bluetooth 2000	4	SONY	HDP-SN-194821	600	258
9	Fone de Ouvido Tune T5000	4	JBL	HDP-JB-091934	780	327.6
10	Microfone Condensador MC1000	5	AKG	MIC-AK-237591	1100	275
11	Microfone Condensador com Tripé	5	BLUE	MIC-BL-819455	800	344
12	Microfone de mesa com fio condensa...	5	BLUE	MIC-BL-761411	650	214.5
13	Notebook LC2100 Intel Core i5 8GB	6	SAMSUNG	NOT-SS-918457	3400	850
14	Notebook Inspiron 15 5000 4GB	6	DELL	NOT-DL-000012	3100	1209
15	Notebook IdeaPad RF32000	6	DELL	NOT-DL-77164I	4200	1176
16	Notebook Motion Ultra 2	6	SAMSUNG	NOT-SS-13139U	2900	1160

**Chave Primária**

pedidos						
ID_Pedido	Data_Venda	ID_Loja	ID_Produto	ID_Cliente	Qtd_Vendida	Preco_Unit
1	2019/01/01	6	11	45	1	800
2	2019/01/01	2	11	84	1	800
3	2019/01/01	1	14	12	1	3100
4	2019/01/01	8	9	98	1	780
5	2019/01/01	5	8	21	1	600
6	2019/01/01	6	8	26	1	600
7	2019/01/01	6	13	90	1	3400
8	2019/01/01	3	3	73	1	450
9	2019/01/01	5	15	17	1	4200
10	2019/01/01	1	4	27	1	350
11	2019/01/01	5	9	25	1	780
12	2019/01/01	5	4	75	1	350
13	2019/01/01	5	11	48	1	800
14	2019/01/01	5	16	8	1	2900
15	2019/01/01	5	9	41	1	780
16	2019/01/01	8	1	6	1	2300
17	2019/01/01	6	14	77	1	3100
18	2019/01/01	5	15	55	1	4200
19	2019/01/01	5	3	65	1	450
20	2019/01/01	7	16	16	1	2900
21	2019/01/01	5	11	55	1	800
22	2019/01/01	6	9	36	1	780
23	2019/01/01	3	4	42	1	350
24	2019/01/01	5	12	81	1	650
25	2019/01/01	7	1	77	1	2300

**Chave Estrangeira**

# Tabela Fato vs. Tabela Dimensão

Uma **Tabela Dimensão** é uma tabela que contém características de um determinado elemento: lojas, produtos, funcionários, clientes, etc.

Nesta tabela, nenhum dos elementos principais irá se repetir. É onde vamos encontrar nossas **chaves primárias**.

Já uma **Tabela Fato** é uma tabela que vai registrar os fatos ou acontecimentos de uma empresa/negócio em determinados períodos de tempo (vendas, devoluções, aberturas de chamados, receitas, despesas, etc).

Geralmente é uma tabela com milhares de informações e composta essencialmente por colunas de ID usadas para buscar as informações complementares de uma tabela dimensão, conhecidas como **chaves estrangeiras**.

No exemplo ao lado, a FactSales é a nossa tabela Fato e a DimChannel é a nossa tabela Dimensão.

ID_Produto	Nome_Produto	ID_Categoria	Marca_Produto	Num_Serie	Preco_Unit	Custo_Unit
1	Monitor LED 19,5" Full HD HDMI	1	DELL	MNT-DL-831923	2300	966
2	Monitor Curvo 24" 144Hz HDMI	1	SAMSUNG	MNT-SS-001939	2800	980
3	Webcam Full HD 1080p	1	LOGITECH	WBC-LT-934GG4	450	90
4	Kit Teclado + Mouse sem fio Wireless	2	DELL	KTM-DL-041039	350	129.5
5	Kit Teclado + Mouse Slim Bluetooth	2	DELL	KTM-DL-111924	280	109.2
6	Cadeira Gamer reclinável Azul/Laranja	3	ALTURA	CGM-AL-9N914J	1800	540
7	Cadeira Gamer PC Racer Vermelha	3	ALTURA	CGM-AL-0147FI	3100	1395
8	Headphone Bluetooth 2000	4	SONY	HDP-SN-194821	600	258
9	Fone de Ouvido Tune T5000	4	JBL	HDP-JB-091934	780	327.6
10	Microfone Condensador MC1000	5	AKG	MIC-AK-237591	1100	275
11	Microfone Condensador com Tripé	5	BLUE	MIC-BL-819455	800	344
12	Microfone de mesa com fio condensa...	5	BLUE	MIC-BL-761411	650	214.5
13	notebook LC2100 Intel Core i5 8GB	6	SAMSUNG	NOT-SS-918457	3400	850
14	notebook Inspiron 15 5000 4GB	6	DELL	NOT-DL-000012	3100	1209
15	notebook IdeaPad RF32000	6	DELL	NOT-DL-77164I	4200	1176
16	notebook Motion Ultra 2	6	SAMSUNG	NOT-SS-13139U	2900	1160

produtos

Chave Primária

ID_Pedido	Data_Venda	ID_Loja	ID_Produto	ID_Cliente	Qtd_Vendida	Preco_Unit	Custo_Unit	Receita_Venda	Custo_Venda
1	2019050801	8	33	45	1	800	344	800	344
2	2019050801	2	33	84	1	800	344	800	344
3	2019050801	1	19	52	1	3000	1200	3000	1200
4	2019050801	8	9	98	1	780	327.6	780	327.6
5	2019050801	9	8	21	1	600	258	600	258
6	2019050801	8	8	26	1	600	258	600	258
7	2019050801	6	17	90	1	3400	950	3400	950
8	2019050801	3	3	79	1	400	90	400	90
9	2019050801	9	15	17	1	4000	1176	4000	1176
10	2019050801	1	4	27	1	300	120.5	300	120.5
11	2019050801	9	9	25	1	780	327.6	780	327.6
12	2019050801	9	4	75	1	300	120.5	300	120.5
13	2019050801	9	11	48	1	600	258	600	258
14	2019050801	9	26	8	1	2000	1160	2000	1160
15	2019050801	9	9	41	1	780	327.6	780	327.6
16	2019050801	8	1	6	1	2000	966	2000	966
17	2019050801	6	14	77	1	3000	1200	3000	1200
18	2019050801	9	19	55	1	4000	1176	4000	1176
19	2019050801	9	3	45	1	400	90	400	90
20	2019050801	7	38	36	1	2000	1160	2000	1160
21	2019050801	9	11	55	1	600	258	600	258
22	2019050801	6	9	36	1	780	327.6	780	327.6
23	2019050801	3	4	42	1	300	120.5	300	120.5
24	2019050801	9	12	81	1	600	214.5	600	214.5

pedidos

# Tabela Fato vs. Tabela Dimensão

Uma **Tabela Dimensão** é uma tabela que contém características de um determinado elemento: lojas, produtos, funcionários, clientes, etc.

Nesta tabela, nenhum dos elementos principais irá se repetir. É onde vamos encontrar nossas **chaves primárias**.

Já uma **Tabela Fato** é uma tabela que vai registrar os fatos ou acontecimentos de uma empresa/negócio em determinados períodos de tempo (vendas, devoluções, aberturas de chamados, receitas, despesas, etc).

Geralmente é uma tabela com milhares de informações e composta essencialmente por colunas de ID usadas para buscar as informações complementares de uma tabela dimensão, conhecidas como **chaves estrangeiras**.

No exemplo ao lado, a tabela Pedidos é a nossa tabela Fato e a Produtos é a nossa tabela Dimensão.

ID_Produto	Nome_Produto	ID_Categoria	Marca_Produto	Nome_Serie	Preco_Unit	Custo_Unit
1	Monitor LED 19,5" Full HD HDMI	1	DELL	MONITOR-001923	2800	960
2	Monitor Curvo 24" 144Hz HDMI	1	SAMSUNG	MONITOR-001939	2800	960
3	Webcam Full HD 1080p	1	LOGITECH	WEBCAM-0019404	450	90
4	Kit Teclado + Mouse sem Fio Wireless	2	DELL	KIT-001939	280	129,5
5	Kit Teclado + Mouse Slim Bluetooth	2	DELL	KIT-001934	280	109,2
6	Cadeira Gamer reclinável Azul/Laranja	3	ALTURA	CADEIRA-0019342	1800	540
7	Cadeira Gamer PC Racer Vermelha	3	ALTURA	CADEIRA-01491	2100	1395
8	Headphone Bluetooth 2000	4	SONY	HEADPHONE-0019421	600	250
9	Fone de Ouvido Tune T5000	4	BL	HEADPHONE-001934	780	327,6
10	Microfone Condensador MC0000	5	AKG	MICROFONE-0019391	1200	275
11	Microfone Condensador com Tripé	5	BLUE	MICROFONE-0019403	800	344
12	Microfone de mesa com Fio condensador	5	BLUE	MICROFONE-0019411	600	214,5
13	Notebook LC2100 Intel Core i5 8GB	6	SAMSUNG	NOTEBOOK-0019457	3400	850
14	Notebook Inspiron 15 5000 4GB	6	DELL	NOTEBOOK-0000012	3100	1200
15	Notebook IdeaPad 320000	6	DELL	NOTEBOOK-0019384	4200	1170
16	Notebook Monitor Ultra 2	6	SAMSUNG	NOTEBOOK-0019386	2900	1180

produtos

ID_Pedido	Data_Venda	ID_Loja	ID_Produto	ID_Cliente	Qtd_Vendida	Preco_Unit	Custo_Unit	Receita_Venda	Custo_Venda
1	2019/01/01	6	11	45	1	800	344	800	344
2	2019/01/01	2	11	84	1	800	344	800	344
3	2019/01/01	1	14	12	1	3100	1209	3100	1209
4	2019/01/01	8	9	98	1	780	327,6	780	327,6
5	2019/01/01	5	8	21	1	600	258	600	258
6	2019/01/01	6	8	26	1	600	258	600	258
7	2019/01/01	6	13	90	1	3400	850	3400	850
8	2019/01/01	3	3	73	1	450	90	450	90
9	2019/01/01	5	15	17	1	4200	1176	4200	1176
10	2019/01/01	1	4	27	1	350	129,5	350	129,5
11	2019/01/01	5	9	25	1	780	327,6	780	327,6
12	2019/01/01	5	4	75	1	350	129,5	350	129,5
13	2019/01/01	5	11	48	1	800	344	800	344
14	2019/01/01	5	16	8	1	2900	1160	2900	1160
15	2019/01/01	5	9	41	1	780	327,6	780	327,6
16	2019/01/01	8	1	6	1	2300	966	2300	966
17	2019/01/01	6	14	77	1	3100	1209	3100	1209
18	2019/01/01	5	15	55	1	4200	1176	4200	1176
19	2019/01/01	5	3	65	1	450	90	450	90
20	2019/01/01	7	16	16	1	2900	1160	2900	1160
21	2019/01/01	5	11	55	1	800	344	800	344
22	2019/01/01	6	9	36	1	780	327,6	780	327,6
23	2019/01/01	3	4	42	1	350	129,5	350	129,5
24	2019/01/01	5	12	81	1	650	214,5	650	214,5
25	2019/01/01	7	1	77	1	2200	966	2200	966

pedidos

Chave Estrangeira

Nas aulas anteriores vimos que existem dois tipos de tabelas: Dimensão e Fato. Essas tabelas podem ser relacionadas através de uma coluna: na tabela Dimensão, identificamos a chave Primária, que será relacionada com a chave Estrangeira da tabela Fato.

Nas aulas anteriores vimos que existem dois tipos de tabelas: Dimensão e Fato. Essas tabelas podem ser relacionadas através de uma coluna: na tabela Dimensão, identificamos a chave Primária, que será relacionada com a chave Estrangeira da tabela Fato.

produtos

ID_Produto	Nome_Produto	ID_Categoria	Marca_Produto	Num_Serie	Preco_Unit	Custo_Unit
1	Monitor LED 19,5" Full HD HDMI	1	DELL	MNT-DL-831923	2300	966
2	Monitor Curvo 24" 144Hz HDMI	1	SAMSUNG	MNT-SS-001939	2800	980
3	Webcam Full HD 1080p	1	LOGITECH	WBC-LT-934GG4	450	90
4	Kit Teclado + Mouse sem fio Wireless	2	DELL	KTM-DL-041039	350	129.5
5	Kit Teclado + Mouse Slim Bluetooth	2	DELL	KTM-DL-111924	280	109.2
6	Cadeira Gamer reclinável Azul/Laranja	3	ALTURA	CGM-AL-9N914J	1800	540
7	Cadeira Gamer PC Racer Vermelha	3	ALTURA	CGM-AL-0147FI	3100	1395
8	Headphone Bluetooth 2000	4	SONY	HDP-SN-194821	600	258
9	Fone de Ouvido Tune T5000	4	JBL	HDP-JB-091934	780	327.6
10	Microfone Condensador MC1000	5	AKG	MIC-AK-237591	1100	275
11	Microfone Condensador com Tripé	5	BLUE	MIC-BL-819455	800	344
12	Microfone de mesa com fio condensa...	5	BLUE	MIC-BL-761411	650	214.5
13	Notebook LC2100 Intel Core i5 8GB	6	SAMSUNG	NOT-SS-918457	3400	850
14	Notebook Inspiron 15 5000 4GB	6	DELL	NOT-DL-000012	3100	1209
15	Notebook IdeaPad RF32000	6	DELL	NOT-DL-77164I	4200	1176
16	Notebook Motion Ultra 2	6	SAMSUNG	NOT-SS-13139U	2900	1160

Chave Primária

Tabela Dimensão

Chave Estrangeira

pedidos

ID_Pedido	Data_Venda	ID_Loja	ID_Produto	ID_Cliente	Qtd_Vendida	Preco_Unit	Custo_Unit	Receita_Venda	Custo_Venda
1	2019/01/01	6	11	45	1	800	344	800	344
2	2019/01/01	2	11	84	1	800	344	800	344
3	2019/01/01	1	14	12	1	3100	1209	3100	1209
4	2019/01/01	8	9	98	1	780	327.6	780	327.6
5	2019/01/01	5	8	21	1	600	258	600	258
6	2019/01/01	6	8	26	1	600	258	600	258
7	2019/01/01	6	13	90	1	3400	850	3400	850
8	2019/01/01	3	3	73	1	450	90	450	90
9	2019/01/01	5	15	17	1	4200	1176	4200	1176
10	2019/01/01	1	4	27	1	350	129.5	350	129.5
11	2019/01/01	5	9	25	1	780	327.6	780	327.6
12	2019/01/01	5	4	75	1	350	129.5	350	129.5
13	2019/01/01	5	11	48	1	800	344	800	344
14	2019/01/01	5	16	8	1	2900	1160	2900	1160
15	2019/01/01	5	9	41	1	780	327.6	780	327.6
16	2019/01/01	8	1	6	1	2300	966	2300	966
17	2019/01/01	6	14	77	1	3100	1209	3100	1209
18	2019/01/01	5	15	55	1	4200	1176	4200	1176
19	2019/01/01	5	3	65	1	450	90	450	90
20	2019/01/01	7	16	16	1	2900	1160	2900	1160
21	2019/01/01	5	11	55	1	800	344	800	344
22	2019/01/01	6	9	36	1	780	327.6	780	327.6
23	2019/01/01	3	4	42	1	350	129.5	350	129.5
24	2019/01/01	5	12	81	1	650	214.5	650	214.5
25	2019/01/01	7	1	77	1	2300	966	2300	966

Tabela Fato

Nas aulas anteriores vimos que existem dois tipos de tabelas: Dimensão e Fato. Essas tabelas podem ser relacionadas através de uma coluna: na tabela Dimensão, identificamos a chave Primária, que será relacionada com a chave Estrangeira da tabela Fato.

Mas pra que servem essas relações?

Com essas relações, conseguimos utilizar informações de uma tabela em outra tabela. Será muito útil, por exemplo, pra gente descobrir o nome do produto vendido (na tabela de Pedidos) fazendo essa busca lá na tabela de Produtos.

Nas aulas anteriores vimos que existem dois tipos de tabelas: Dimensão e Fato. Essas tabelas podem ser relacionadas através de uma coluna: na tabela Dimensão, identificamos a chave Primária, que será relacionada com a chave Estrangeira da tabela Fato.

Mas pra que servem essas relações?

Com essas relações, conseguimos utilizar informações de uma tabela em outra tabela. Será muito útil, por exemplo, pra gente descobrir o nome do produto vendido (na tabela de Pedidos) fazendo essa busca lá na tabela de Produtos.

pedidos

ID_Pedido	Data_Venda	ID_Loja	ID_Produto	ID_Cliente	Qtd_Vendida	Preco_Unit	Custo_Unit	Receita_Venda	Custo_Venda
1	2019/01/01	6	11	45	1	800	344	800	344
2	2019/01/01	3	14	12	1	3000	1200	3000	1200
4	2019/01/01	8	9	98	1	700	327.6	700	327.6
5	2019/01/01	9	8	21	1	600	256	600	256
6	2019/01/01	6	8	26	1	600	256	600	256
7	2019/01/01	6	13	90	1	3400	890	3400	890
8	2019/01/01	3	3	79	1	400	96	400	96
9	2019/01/01	5	15	17	1	4200	1176	4200	1176
10	2019/01/01	3	4	27	1	300	128.8	300	128.8
11	2019/01/01	9	9	25	1	700	327.6	700	327.6
12	2019/01/01	6	4	79	1	300	128.8	300	128.8
13	2019/01/01	9	11	40	1	800	344	800	344
14	2019/01/01	6	18	8	1	2800	1160	2800	1160
15	2019/01/01	5	9	41	1	700	327.6	700	327.6
16	2019/01/01	9	1	6	1	2300	960	2300	960
17	2019/01/01	6	14	77	1	3000	1200	3000	1200
18	2019/01/01	5	15	55	1	4200	1176	4200	1176
19	2019/01/01	5	3	63	1	400	96	400	96
20	2019/01/01	7	16	38	1	2800	1160	2800	1160
21	2019/01/01	5	11	55	1	800	344	800	344
22	2019/01/01	6	9	36	1	700	327.6	700	327.6
23	2019/01/01	3	4	42	1	300	128.8	300	128.8
24	2019/01/01	5	12	81	1	600	214.5	600	214.5
25	2019/01/01	9	1	11	1	1500	600	1500	600



Nas aulas anteriores vimos que existem dois tipos de tabelas: Dimensão e Fato. Essas tabelas podem ser relacionadas através de uma coluna: na tabela Dimensão, identificamos a chave Primária, que será relacionada com a chave Estrangeira da tabela Fato.

Mas pra que servem essas relações?

Com essas relações, conseguimos utilizar informações de uma tabela em outra tabela. Será muito útil, por exemplo, pra gente descobrir o nome do produto vendido (na tabela de Pedidos) fazendo essa busca lá na tabela de Produtos.

produtos

ID_Produto	Nome_Produto	ID_Categoria	Marca_Produto	Num_Serie	Preco_Unit	Custo_Unit
1	Monitor LED 19,5" Full HD HDMI	1	DELL	HMT-GL-411923	2300	966
2	Monitor Curvo 24" 144Hz HDMI	3	SAMSUNG	HMT-05-001939	2800	980
3	Webcam Full HD 1080p	1	LOGITECH	WBC-LT-014604	450	90
4	Kit Teclado + Mouse sem Fio Wireless	2	DELL	KTM-GL-040039	280	129,5
5	Kit Teclado + Mouse Slim Bluetooth	2	DELL	KTM-GL-111924	280	129,2
6	Cadeira Gamer reclinável Ajustável Laranja	3	ALTURA	CGH-AL-000143	1800	940
7	Cadeira Gamer PC Racer Vermelha	3	ALTURA	CGH-AL-014071	2100	1395
8	Headphone Bluetooth 2000	4	SONY	HDP-SB-194821	600	298
9	Fone de Ouvido TWS T5000	4	BL	HDP-BB-092834	780	327,6
10	Microfone Condensador MC2000	5	AKG	HSC-AC-237991	1100	275
11	Microfone Condensador com Tripé	5	BLUE	MIC-BL-819455	800	344
12	Microfone de mesa com fio condensador	5	BLUE	MIC-BL-783913	650	214,5
13	Notebook LC2100 Intel Core i5 8GB	6	SAMSUNG	NOTE-05-W18457	3400	890
14	Notebook Inspiron 15 5000 4GB	6	DELL	NOTE-GL-000012	3100	1209
15	Notebook IdeaPad 320000	6	DELL	NOTE-GL-771646	4200	1176
16	Notebook Moltar Ultra 2	6	SAMSUNG	NOTE-05-12139U	2800	1160

pedidos

ID_Pedido	Data_Venda	ID_Loja	ID_Produto	ID_Cliente	Qtd_Vendida	Preco_Unit	Custo_Unit	Receita_Venda	Custo_Venda
1	2019/01/01	6	11	45	1	800	344	800	344
2	2019/01/01	3	14	12	1	3000	1200	3000	1200
3	2019/01/01	8	9	98	1	780	327,6	780	327,6
4	2019/01/01	9	8	21	1	600	258	600	258
5	2019/01/01	6	8	26	1	600	258	600	258
6	2019/01/01	6	8	26	1	600	258	600	258
7	2019/01/01	6	13	90	1	3400	890	3400	890
8	2019/01/01	3	3	79	1	400	90	400	90
9	2019/01/01	5	15	17	1	4200	1176	4200	1176
10	2019/01/01	3	4	27	1	350	129,5	350	129,5
11	2019/01/01	9	9	25	1	780	327,6	780	327,6
12	2019/01/01	6	4	79	1	350	129,5	350	129,5
13	2019/01/01	9	11	40	1	800	344	800	344
14	2019/01/01	6	16	8	1	2800	1160	2800	1160
15	2019/01/01	6	9	41	1	780	327,6	780	327,6
16	2019/01/01	9	1	6	1	2300	940	2300	940
17	2019/01/01	6	14	77	1	3000	1200	3000	1200
18	2019/01/01	5	15	55	1	4000	1176	4000	1176
19	2019/01/01	5	3	63	1	400	90	400	90
20	2019/01/01	7	16	38	1	2800	1160	2800	1160
21	2019/01/01	9	11	55	1	800	344	800	344
22	2019/01/01	6	9	38	1	780	327,6	780	327,6
23	2019/01/01	3	4	42	1	350	129,5	350	129,5
24	2019/01/01	5	12	81	1	650	214,5	650	214,5
25	2019/01/01	9	1	111	1	11000	4400	11000	4400

Nas aulas anteriores vimos que existem dois tipos de tabelas: Dimensão e Fato. Essas tabelas podem ser relacionadas através de uma coluna: na tabela Dimensão, identificamos a chave Primária, que será relacionada com a chave Estrangeira da tabela Fato.

Mas pra que servem essas relações?

Com essas relações, conseguimos utilizar informações de uma tabela em outra tabela. Será muito útil, por exemplo, pra gente descobrir o nome do produto vendido (na tabela de Pedidos) fazendo essa busca lá na tabela de Produtos.

Essas relações serão criadas por meio do que chamamos de JOIN's. A tradução literal dessa palavra é “juntar”, “unir”. Os JOINs vão nos permitir fazer exatamente isso: juntar as nossas tabelas Fato e Dimensão, de forma a complementar as informações umas das outras.

Nas aulas anteriores vimos que existem dois tipos de tabelas: Dimensão e Fato. Essas tabelas podem ser relacionadas através de uma coluna: na tabela Dimensão, identificamos a chave Primária, que será relacionada com a chave Estrangeira da tabela Fato.

Mas pra que servem essas relações?

Com essas relações, conseguimos utilizar informações de uma tabela em outra tabela. Será muito útil, por exemplo, pra gente descobrir o nome do produto vendido (na tabela de Pedidos) fazendo essa busca lá na tabela de Produtos.

Essas relações serão criadas por meio do que chamamos de JOIN's. A tradução literal dessa palavra é “juntar”, “unir”. Os JOINs vão nos permitir fazer exatamente isso: juntar as nossas tabelas Fato e Dimensão, de forma a complementar as informações umas das outras.

Existem diversos tipos de Joins no SQL, os quais estão listados ao lado.

LEFT JOIN

RIGHT JOIN

INNER JOIN

FULL JOIN

Nas aulas anteriores vimos que existem dois tipos de tabelas: Dimensão e Fato. Essas tabelas podem ser relacionadas através de uma coluna: na tabela Dimensão, identificamos a chave Primária, que será relacionada com a chave Estrangeira da tabela Fato.

Mas pra que servem essas relações?

Com essas relações, conseguimos utilizar informações de uma tabela em outra tabela. Será muito útil, por exemplo, pra gente descobrir o nome do produto vendido (na tabela de Pedidos) fazendo essa busca lá na tabela de Produtos.

Essas relações serão criadas por meio do que chamamos de JOIN's. A tradução literal dessa palavra é “juntar”, “unir”. Os JOINs vão nos permitir fazer exatamente isso: juntar as nossas tabelas Fato e Dimensão, de forma a complementar as informações umas das outras.

Existem diversos tipos de Joins no SQL, os quais estão listados ao lado.

LEFT JOIN

RIGHT JOIN

INNER JOIN

FULL JOIN

# INNER JOIN

000

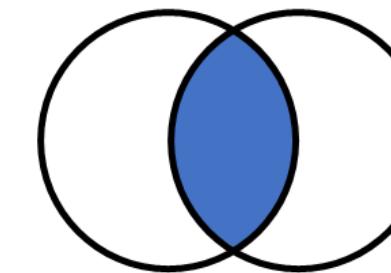
pedidos

ID_Produto	Data_Venda	Qtd_Vendida	Receita_Venda
3	2019/01/01	1	450
4	2019/01/01	1	350
4	2019/01/01	1	350
1	2019/01/01	1	2300
3	2019/01/01	1	450
4	2019/01/01	1	350
1	2019/01/01	1	2300
2	2019/01/01	1	2800
4	2019/01/01	1	350
1	2019/01/01	1	2300

Apenas a interseção entre Tabelas A e B

Tabela A

Tabela B



Resultado

produtos

ID_Produto	Nome_Produto	Marca_Produto	Preco_Unit
1	Monitor LED 19,5" Full HD HDMI	DELL	2300
2	Monitor Curvo 24" 144Hz HDMI	SAMSUNG	2800
3	Webcam Full HD 1080p	LOGITECH	450
4	Kit Teclado + Mouse sem fio Wireless	DELL	350

ID_Produto	Data_Venda	Qtd_Vendida	Receita_Venda	Nome_Produto	Marca_Produto	Preco_Unit
3	2019/01/01	1	450	Webcam Full HD 1080p	LOGITECH	450
4	2019/01/01	1	350	Kit Teclado + Mouse sem fio Wireless	DELL	350
4	2019/01/01	1	350	Kit Teclado + Mouse sem fio Wireless	DELL	350
1	2019/01/01	1	2300	Monitor LED 19,5" Full HD HDMI	DELL	2300
3	2019/01/01	1	450	Webcam Full HD 1080p	LOGITECH	450
4	2019/01/01	1	350	Kit Teclado + Mouse sem fio Wireless	DELL	350
1	2019/01/01	1	2300	Monitor LED 19,5" Full HD HDMI	DELL	2300
2	2019/01/01	1	2800	Monitor Curvo 24" 144Hz HDMI	SAMSUNG	2800
4	2019/01/01	1	350	Kit Teclado + Mouse sem fio Wireless	DELL	350
1	2019/01/01	1	2300	Monitor LED 19,5" Full HD HDMI	DELL	2300

O INNER JOIN vai nos permitir relacionar duas tabelas (Tabela A e Tabela B) e criar uma nova tabela (Tabela C) que é a junção das duas.

A tabela resultante desse JOIN terá **apenas** as linhas que são a interseção entre a Tabela A e a Tabela B.

# INNER JOIN: Estrutura

000



## SELECT

```
Tabela_A.coluna1,  
Tabela_A.coluna2,  
Tabela_A.coluna3,  
Tabela_B.coluna4
```

## FROM

```
Tabela_A
```

```
INNER JOIN Tabela_B
```

```
ON Tabela_A.id_chave_estrangeira = Tabela_B.id_chave_primaria
```

A Tabela\_A será a tabela que vamos querer complementar com as informações da Tabela\_B.



# SQ Views

 SQL IMPRESSIONADOR | HASHTAG TREINAMENTOS

# VIEWS: Introdução

Até aqui vimos como criar diferentes consultas aos bancos de dados. Utilizamos comandos como o SELECT, GROUP BY, JOINs, etc para criar tabelas como a mostrada ao lado.

Mas pra onde foram todas essas “tabelas” que a gente criou? Elas estão em algum lugar?

**A resposta é: elas não estão em nenhum lugar!**

Tudo o que fizemos até agora foi apenas visualizar alguns dados das nossas tabelas do Banco de Dados, nada além disso. Quando executamos um SELECT e logo em seguida executamos outro SELECT, o resultado do primeiro é perdido.

Nenhuma das consultas que fizemos ficou salvo em algum lugar. Inclusive, diversas vezes precisamos criar as mesmas consultas, pois elas se perdem a cada novo SELECT, ou quando fechamos uma consulta e abrimos uma nova.

Existe uma solução pra gente conseguir salvar essas tabelas em algum lugar, e essa solução é a **View**.

## 1 • SELECT

```
2     Nome AS 'Nome do Cliente',
3     Data_Nascimento AS 'Data de Nascimento',
4     Email AS 'E-mail'
5 FROM clientes;
```

Result Grid			
	Nome do Cliente	Data de Nascimento	E-mail
▶	Ruben	1985-08-07	ruben35@hotmail.com
	Christy	1988-02-10	christy12@hotmail.com
	Elizabeth	1988-08-03	elizabeth5@gmail.com
	Julio	1985-07-31	julio1@hotmail.com
	Janet	1985-12-01	janet9@gmail.com
	Marco	1984-05-04	marco14@yahoo.com.br
	Rob	1984-07-02	rob4@hotmail.com
	Shannon	1984-03-27	shannon38@gmail.com
	Jacquelyn	1984-02-01	jacquelyn20@hotmail.com
	Curtis	1983-10-30	curtis9@yahoo.com.br
	Lauren	1988-01-13	lauren41@hotmail.com
	Ian	1988-08-01	ian47@gmail.com
	Sydney	1988-05-04	sydney23@gmail.com
	Chloe	1999-02-22	chloe23@hotmail.com
	Wyatt	1999-04-23	wyatt32@gmail.com
	Shannon	1964-06-21	shannon1@hotmail.com
	Clarence	1964-10-04	clarence32@yahoo.com.br
	Jordan	1998-09-15	jordan73@gmail.com

# O que é uma View?

- Uma View (ou traduzindo, uma exibição), é uma tabela virtual criada a partir de uma consulta a uma ou mais tabelas (ou até mesmo de outras views) no banco de dados.
- Ela contém linhas e colunas assim como uma tabela real. Nela podemos utilizar comandos como o JOIN, WHERE e outras funções.
- Sempre mostra resultados atualizados dos dados, ou seja, uma vez criada uma View, caso haja alterações no Banco de Dados, as Views são atualizadas automaticamente.
- Caso o servidor seja desligado (ou o SSMS fechado), a view continua armazenada no sistema.

Através de uma View, conseguimos armazenar o resultado de um SELECT (como por exemplo, a tabela à direita) e acessar sempre que precisar, como se fosse uma tabela, com a vantagem de não precisar criar esse SELECT do zero.

## 1 • SELECT

```

2     Nome AS 'Nome do Cliente',
3     Data_Nascimento AS 'Data de Nascimento',
4     Email AS 'E-mail'
5 FROM clientes;

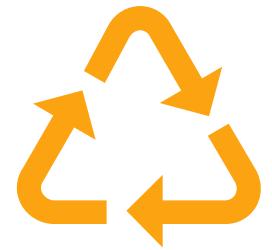
```

Nome do Cliente	Data de Nascimento	E-mail
Ruben	1985-08-07	ruben35@hotmail.com
Christy	1988-02-10	christy12@hotmail.com
Elizabeth	1988-08-03	elizabeth5@gmail.com
Julio	1985-07-31	julio1@hotmail.com
Janet	1985-12-01	janet9@gmail.com
Marco	1984-05-04	marco14@yahoo.com.br
Rob	1984-07-02	rob4@hotmail.com
Shannon	1984-03-27	shannon38@gmail.com
Jacquelyn	1984-02-01	jacquelyn20@hotmail.com
Curtis	1983-10-30	curtis9@yahoo.com.br
Lauren	1988-01-13	lauren41@hotmail.com
Ian	1988-08-01	ian47@gmail.com
Sydney	1988-05-04	sydney23@gmail.com
Chloe	1999-02-22	chloe23@hotmail.com
Wyatt	1999-04-23	wyatt32@gmail.com
Shannon	1964-06-21	shannon1@hotmail.com
Clarence	1964-10-04	clarence32@yahoo.com.br
Jordan	1998-09-15	jordan73@gmail.com

# Por que criar uma View?

São muitas as vantagens de uma View. Abaixo temos algumas das principais:

## Reutilização



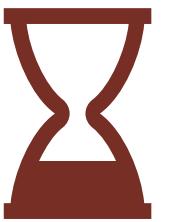
Sempre que necessário, podemos consultar aquela View, pois ela fica armazenada no sistema.

## Segurança



Ao criar uma View, estamos ocultando linhas ou colunas da tabela original do banco de dados. Desta forma, apenas algumas informações relevantes serão visualizadas na View.

## Ganho de tempo



Quando criamos Views, estamos poupando o tempo de recriar vários SELECTs, o que aumenta a produtividade.

# CREATE VIEW: Criando a primeira view

Para criar uma View, utilizamos o comando **CREATE VIEW**. Na imagem abaixo temos a estrutura padrão:

```
CREATE VIEW nome_da_view AS
SELECT
```

```
    Coluna1,
    Coluna2,
    Coluna3
```

```
FROM
    Tabela
```

```
1 • CREATE VIEW vwClientes AS
2   SELECT
3     Nome AS 'Nome do Cliente',
4     Data_Nascimento AS 'Data de Nascimento',
5     Email AS 'E-mail'
6   FROM clientes;
```

Uma vez criada, a View ficará armazenada no banco de dados, na pasta de **Views**.

E para selecionar essa View, utilizamos o comando **SELECT**.

The screenshot shows the MySQL Workbench interface. On the left, the Navigator pane displays the database schema under the 'base' schema. It shows tables like 'categorias', 'clientes', 'locais', 'lojas', 'pedidos', 'produtos', and views like 'vwclientes'. An orange arrow points from the 'vwclientes' view in the Navigator to the 'Result Grid' on the right. The 'Result Grid' displays a list of client records with columns: Nome do Cliente, Data de Nascimento, and E-mail. The records are:

Nome do Cliente	Data de Nascimento	E-mail
Ruben	1985-08-07	ruben35@hotmail.com
Christy	1988-02-10	christy12@hotmail.com
Elizabeth	1988-08-03	elizabeth5@gmail.com
Julio	1985-07-31	julio1@hotmail.com
Janet	1985-12-01	janet9@gmail.com
Marco	1984-05-04	marco14@yahoo.com.br
Rob	1984-07-02	rob4@hotmail.com
Shannon	1984-03-27	shannon38@gmail.com
Jacquelyn	1984-02-01	jacquelyn20@hotmail.com
Curtis	1983-10-30	curtis9@yahoo.com.br
Lauren	1988-01-13	lauren41@hotmail.com

At the top right of the Navigator pane, there is a command line with the text: **1 • SELECT \* FROM vwClientes;**

# ALTER VIEW: Alterando a view criada

Para alterar uma View criada, usamos o comando **ALTER VIEW**.

Na imagem ao lado, temos um exemplo. A vwClientes, criada anteriormente, foi alterada para considerar apenas os clientes do sexo feminino.

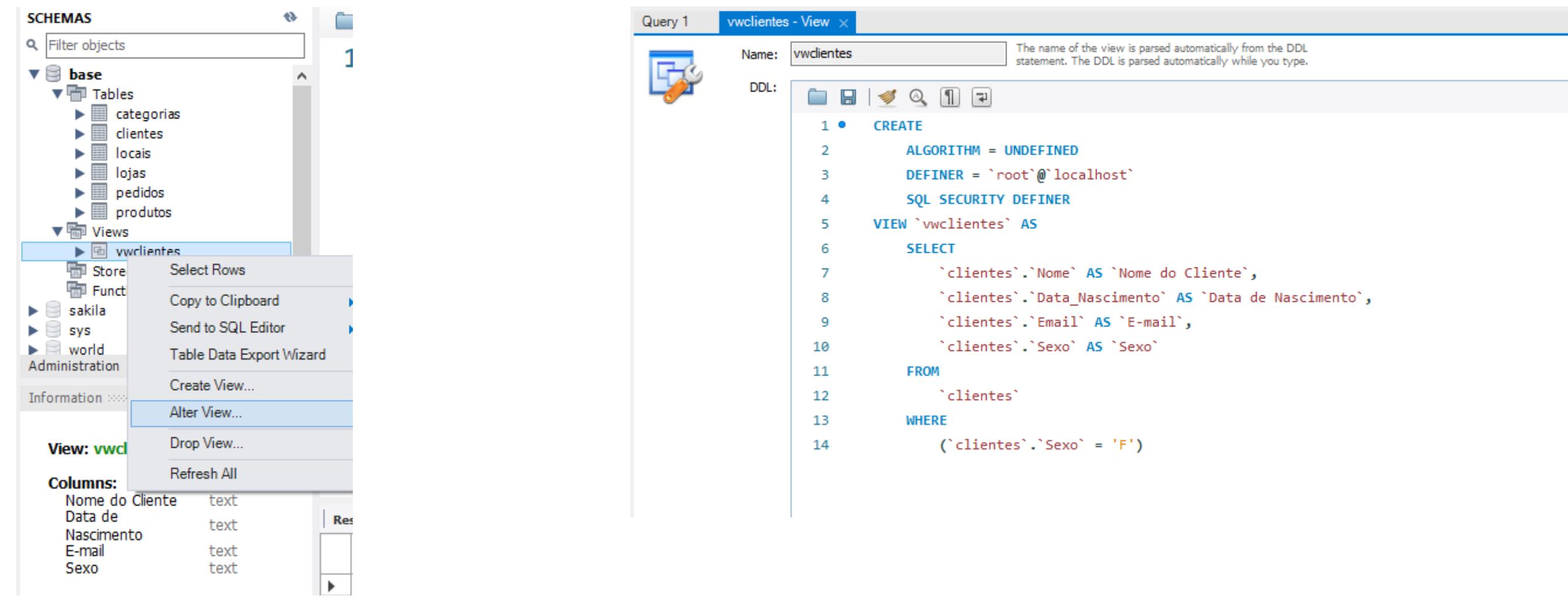
```

1 • ALTER VIEW vwClientes AS
2   SELECT
3     Nome AS 'Nome do Cliente',
4     Data_Nascimento AS 'Data de Nascimento',
5     Email AS 'E-mail',
6     Sexo AS 'Sexo'
7   FROM clientes
8   WHERE Sexo = 'F';
9
10 • SELECT * FROM vwclientes;
  
```

Result Grid   Filter Rows: [ ]   Export: [ ]   Wrap Cell Content: [ ]				
	Nome do Cliente	Data de Nascimento	E-mail	Sexo
▶	Christy	1988-02-10	christy12@hotmail.com	F
	Elizabeth	1988-08-03	elizabeth5@gmail.com	F
	Janet	1985-12-01	janet9@gmail.com	F
	Rob	1984-07-02	rob4@hotmail.com	F
	Jacquelyn	1984-02-01	jacquelyn20@hotmail.com	F
	Lauren	1988-01-13	lauren41@hotmail.com	F
	Sydney	1988-05-04	sydney23@gmail.com	F
	Chloe	1999-02-22	chloe23@hotmail.com	F
	Shannon	1964-06-21	shannon1@hotmail.com	F
	Destiny	1998-08-29	destiny7@hotmail.com	F
	Jill	1966-04-06	jill13@gmail.com	F
	Bethany	1967-02-17	bethany10@hotmail.com	F
	Theresa	1967-02-17	theresa12@hotmail.com	F

# ALTER VIEW: Alterando a view criada

Para alterar a view também podemos clicar nela com o botão direito e ir na opção Alter View. A janela que se abre mostra inclusive o código que deu origem à view criada.



# DROP VIEW: Excluindo uma view

Para excluir uma View criada, usamos o comando **DROP VIEW**.

1

```
DROP VIEW vwclientes;
```

O comando é bem simples e é ilustrado na imagem ao lado.

# SQL

## UDFs e Stored Procedures

# SQL



# Funções e Procedimentos

Se tratam de dois tipos de rotinas. Uma rotina é um conjunto de instruções que você pode salvar no seu banco de dados e executar quando quiser, sem a necessidade de criar o código do zero toda vez que você precisa dele.

Funções (UDFs) e Procedimentos (Stored Procedures) são um pouco similares, mas com algumas diferenças em termos de aplicações.

# Funções (User-Defined Functions): Exemplo 1

Uma função é usada para gerar um valor que pode ser usado em uma expressão.

O valor (resultado) é gerado baseado em um ou mais parâmetros de entrada fornecidos à função.

```
1 • DROP FUNCTION IF EXISTS fn_Faturamento;  
2  
3 • CREATE FUNCTION fn_Faturamento(preco DECIMAL(10, 2), quantidade INT)  
4   RETURNS DECIMAL(10, 2) DETERMINISTIC  
5   RETURN preco * quantidade;  
6  
7  
8 • SELECT fn_Faturamento(2.5, 10) AS 'Faturamento';
```

Result Grid	
Faturamento	
	25.00

# Funções (User-Defined Functions): Exemplo 2

Uma função é usada para gerar um valor que pode ser usado em uma expressão.

O valor (resultado) é gerado baseado em um ou mais parâmetros de entrada fornecidos à função.

```
1 • CREATE FUNCTION fn_CalculaDesconto(preco DECIMAL(10, 2), desconto DECIMAL(10, 2))
2   RETURNS DECIMAL(10, 2) DETERMINISTIC
3   RETURN preco * (1 - desconto);
4
5 • SELECT Nome_Produto, Preco_Unit, fn_CalculaDesconto(Preco_Unit, 0.5) FROM produtos;
```

Nome_Produto	Preco_Unit	fn_CalculaDesconto(Preco_Unit, 0.5)
Monitor LED 19,5" Full HD HDMI	2300	1150.00
Monitor Curvo 24" 144Hz HDMI	2800	1400.00
Webcam Full HD 1080p	450	225.00
Kit Teclado + Mouse sem fio Wireless	350	175.00
Kit Teclado + Mouse Slim Bluetooth	280	140.00
Cadeira Gamer reclinável Azul/Laranja	1800	900.00

# Stored Procedures: (Procedimentos Armazenados)

Se tratam de dois tipos de rotinas armazenadas. Uma rotina é um conjunto de instruções que você pode armazenar e executar posteriormente, sem a necessidade de criar o código do zero toda vez que você precisa dele.

Funções (UDFs) e Procedimentos (Stored Procedures) são um pouco similares, mas com algumas diferenças em termos de aplicações.

# Stored Procedures: Exemplo 1

Se tratam de dois tipos de rotinas armazenadas. Uma rotina é um conjunto de instruções que você pode armazenar e executar posteriormente, sem a necessidade de criar o código do zero toda vez que você precisa dele.

Funções (UDFs) e Procedimentos (Stored Procedures) são um pouco similares, mas com algumas diferenças em termos de aplicações.

```
1  DELIMITER $$  
2 • CREATE PROCEDURE sp_AtualizaPreco(NovoPreco DECIMAL(10, 2), ID INT)  
3  BEGIN  
4      UPDATE dCursos  
5      SET Preco_Curso = NovoPreco  
6      WHERE ID_Curso = ID;  
7      SELECT 'Preço atualizado com sucesso!';  
8  END $$  
9  DELIMITER ;  
10  
11 • CALL sp_AtualizaPreco(600, 1);  
12  
13 • SELECT * FROM dCursos;
```

The screenshot shows the MySQL Workbench interface. At the top, there is a code editor window containing the SQL code for creating a stored procedure and calling it. Below the code editor is a result grid window titled 'Result Grid'. The result grid displays a table with three rows of data from the 'dCursos' table. The columns are labeled 'ID\_Curso', 'Nome\_Curso', and 'Preco\_Curso'. The data is as follows:

ID_Curso	Nome_Curso	Preco_Curso
1	Excel	600.00
2	VBA	200.00
3	Power BI	150.00
*	NULL	NULL

# Stored Procedures: Exemplo 2

Se tratam de dois tipos de rotinas armazenadas. Uma rotina é um conjunto de instruções que você pode armazenar e executar posteriormente, sem a necessidade de criar o código do zero toda vez que você precisa dele.

Funções (UDFs) e Procedimentos (Stored Procedures) são um pouco similares, mas com algumas diferenças em termos de aplicações.

```

16 ######
17 DELIMITER $$ 
18 • CREATE PROCEDURE sp_AplicaDesconto(ID INT, Desconto DECIMAL(10, 2))
19 • BEGIN
20     DECLARE varPrecoComDesconto DECIMAL(10, 2);
21     DECLARE varNomeCurso VARCHAR(100);
22
23     SET varPrecoComDesconto = (SELECT Preco_Curso FROM dCursos WHERE ID_Curso = ID) * (1 - Desconto);
24     SET varNomeCurso = (SELECT Nome_Curso FROM dCursos WHERE ID_Curso = ID);
25
26     UPDATE dCursos
27     SET Preco_Curso = varPrecoComDesconto
28     WHERE ID_Curso = ID;
29     SELECT 'Desconto aplicado com sucesso!';
30 END $$ 
31 DELIMITER ;
32
33 • SELECT * FROM dCursos;
34
35 • CALL sp_AplicaDesconto(1, 0.5);

```

The screenshot shows a MySQL Workbench interface with a result grid. The grid has columns for ID\_Curso, Nome\_Curso, and Preco\_Curso. The data is as follows:

ID_Curso	Nome_Curso	Preco_Curso
1	Excel	300.00
2	VBA	200.00
3	Power BI	150.00
NULL	NULL	NULL

# Stored Procedures: Excluir

Se tratam de dois tipos de rotinas armazenadas. Uma rotina é um conjunto de instruções que você pode armazenar e executar posteriormente, sem a necessidade de criar o código do zero toda vez que você precisa dele.

Funções (UDFs) e Procedimentos (Stored Procedures) são um pouco similares, mas com algumas diferenças em termos de aplicações.

```
DROP PROCEDURE ConsultaID;
```

# BEGIN... END;

Os comandos BEGIN e END têm como objetivo delimitar um bloco de comandos a serem executados por uma Function ou Stored Procedure.

Cada bloco BEGIN / END possui um delimitador (;).

Porém, o delimitador ; pode ser problemático pois, ao ser encontrado em um procedimento ou função, ele a finaliza imediatamente. Para não termos problemas, devemos então mudar esse delimitador e para isso usamos o comando DELIMITER para criar rotinas com declarações compostas.

# BEGIN... END: Function

Os comandos BEGIN e END têm como objetivo delimitar um bloco de comandos a serem executados por uma Function ou Stored Procedure.

Cada bloco BEGIN / END possui um delimitador (;).

Porém, o delimitador ; pode ser problemático pois, ao ser encontrado em um procedimento ou função, ele a finaliza imediatamente. Para não termos problemas, devemos então mudar esse delimitador e para isso usamos o comando DELIMITER para criar rotinas com declarações compostas.

```
1 -- Criar função para cálculo de receita de produto
2
3 DELIMITER //
4 • CREATE FUNCTION fn_CalculaReceita(Quantidade INT, Preco DECIMAL(10, 2))
5 RETURNS DECIMAL(10, 2) DETERMINISTIC
6 BEGIN
7   RETURN quantidade * preco;
8 END //
9 DELIMITER ;
10
11 • SELECT fn_CalculaReceita(10, 99.90);
12
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	fn_CalculaReceita(10, 99.90)			
▶	999.00			

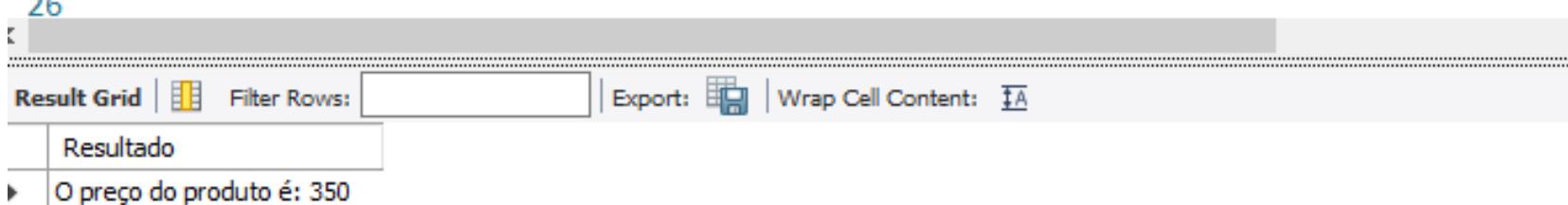
# BEGIN... END: Stored Procedure

Os comandos BEGIN e END têm como objetivo delimitar um bloco de comandos a serem executados por uma Function ou Stored Procedure.

Cada bloco BEGIN / END possui um delimitador (;).

Porém, o delimitador ; pode ser problemático pois, ao ser encontrado em um procedimento ou função, ele a finaliza imediatamente. Para não termos problemas, devemos então mudar esse delimitador e para isso usamos o comando DELIMITER para criar rotinas com declarações compostas.

```
14  -- Criar procedimento para busca de preço do produto
15  DELIMITER $$
16 • CREATE PROCEDURE sp_BuscaPreco(ID INT)
17 • BEGIN
18   SELECT CONCAT('O preço do produto é: ', Preco_Unit) AS 'Resultado'
19   FROM produtos
20   WHERE ID_Produto = ID;
21   SELECT 'Procedimento finalizado com sucesso!!' AS 'Mensagem';
22 END $$
23 DELIMITER ;
24
25 • CALL sp_BuscaPreco(4);
26
```



The screenshot shows the MySQL Workbench interface with the results of the stored procedure execution. The result grid contains one row with the value 'O preço do produto é: 350'. The grid has columns for 'Result Grid' and 'Filter Rows:'. There are also buttons for 'Export:' and 'Wrap Cell Content:'.

# Escopo das variáveis: Revisão

O escopo de uma variável se refere ao local onde essa variável existe, ou seja, o local onde ela pode ser acessada.

Temos os seguintes níveis de escopo:

1. USER-DEFINED VARIABLES
2. LOCAL VARIABLES

# Escopo das variáveis: Revisão

O escopo de uma variável se refere ao local onde essa variável existe, ou seja, o local onde ela pode ser acessada.

Temos os seguintes níveis de escopo:

1. USER-DEFINED VARIABLES
2. LOCAL VARIABLES



USADA DENTRO DE FUNCTIONS E STORED PROCEDURES

# Declaração de variáveis locais

Podemos criar variáveis locais em um procedimento ou função usando uma declaração DECLARE dentro de um bloco BEGIN.

A variável pode ser criada e inicializada com um valor se desejado.

Ficam disponíveis apenas dentro do bloco onde foram criadas, e em blocos que existam dentro do bloco onde a variável foi criada.

Após o bloco ter sido executado e encerrado, a variável é desalocada da memória.

## FUNCTION

```
DELIMITER $$  
CREATE FUNCTION fn_Exemplo(valor INT)  
RETURNS INT  
BEGIN  
    DECLARE var1 INT;  
    DECLARE var2 DECIMAL(10, 2);  
    DECLARE var3 VARCHAR(100);  
    DECLARE var4 DATE;  
  
    SET var1 = 10;  
    SET var2 = 9.98;  
    SET var3 = 'SQL';  
    SET var4 = '2999-12-31';  
END $$
```

## PROCEDURE

```
DELIMITER $$  
CREATE PROCEDURE sp_Exemplo()  
BEGIN  
    DECLARE var1 INT;  
    DECLARE var2 DECIMAL(10, 2);  
    DECLARE var3 VARCHAR(100);  
    DECLARE var4 DATE;  
  
    SET var1 = 10;  
    SET var2 = 9.98;  
    SET var3 = 'SQL';  
    SET var4 = '2999-12-31';  
END $$
```

# Functions vs Stored Procedures

000

## Functions

- Uma function é usada para calcular um valor baseado em alguns inputs.
- Functions não permitem a utilização de instruções INSERT, UPDATE e DELETE para alteração de um banco de dados.
- Usamos functions em conjunto com as instruções SELECT, WHERE, HAVING mas não é possível fazer o mesmo com procedures.
- Uma function pode ser executada dentro de uma procedure, mas o contrário não é válido.



## Procedures

- Uma procedure pode retornar um ou mais valores, ou não retornar nenhum valor.
- Procedures podem ser vistas como programas/scripts. Uma procedure permite alterar o estado global da base de dados (por exemplo, a utilização das instruções INSERT, UPDATE, DELETE).
- Procedures são utilizadas normalmente para juntar várias queries em um único bloco de comando.

# SQL

## Módulo Oracle

ORACLE®



# SUMÁRIO

000

## MÓDULO X NOME DO MÓDULO

01	NOME DO MÓDULO	.....	01
02	NOME DO MÓDULO	.....	02
03	NOME DO MÓDULO	.....	03
04	NOME DO MÓDULO	.....	04
05	NOME DO MÓDULO	.....	05

## MÓDULO X NOME DO MÓDULO

01	NOME DO MÓDULO	.....	01
02	NOME DO MÓDULO	.....	02
03	NOME DO MÓDULO	.....	03
04	NOME DO MÓDULO	.....	04
05	NOME DO MÓDULO	.....	05

## MÓDULO X NOME DO MÓDULO

01	NOME DO MÓDULO	.....	01
02	NOME DO MÓDULO	.....	02
03	NOME DO MÓDULO	.....	03
04	NOME DO MÓDULO	.....	04
05	NOME DO MÓDULO	.....	05

## MÓDULO X NOME DO MÓDULO

01	NOME DO MÓDULO	.....	01
02	NOME DO MÓDULO	.....	02
03	NOME DO MÓDULO	.....	03
04	NOME DO MÓDULO	.....	04
05	NOME DO MÓDULO	.....	05



MÓDULO 1

INTRODUÇÃO

INTRODUÇÃO

INTRODUÇÃO



# A história dos Bancos de Dados

Todos sabemos que os dados/informações foram armazenados durante décadas dentro de bibliotecas, salas de registros, pastas físicas, fichas em papel, enfim, armazenados de maneira impressa para **consultas** posteriores.

Com décadas de informações geradas, é claro que esse volume todo trazia uma série de desafios em sua manipulação e organização.

A partir dos anos 60, os computadores passam a ser um recurso importante dentro das empresas, e junto com eles, veio a possibilidade de armazenar informações em formato digital (deixando de ser exclusivamente impresso).

# A história dos Bancos de Dados

Após alguma evolução nos modelos de bancos de dados, no início dos anos 70, Edgar Frank propõe o **modelo de dados relacional**, um marco na forma de pensar em bancos de dados.

Ainda na década de 70, foram desenvolvidos dois protótipos de sistema relacional.

1. Ingres, desenvolvido pela UCB.
2. System R, desenvolvido pela IBM.

Foi neste período que também foi criado o termo **Sistema de Gerenciamento de Banco de Dados Relacional** (SGBDR, ou RDBMS em inglês).

Mas o que seria um sistema relacional?

# Banco de Dados Relacional

Um Banco de Dados Relacional é um conjunto de tabelas que armazenam informações sobre algum negócio. Essas tabelas se relacionam por meio de colunas em comum, conhecidas como chaves.

Observe o Banco de Dados abaixo, composto por duas tabelas: **EMPLOYEES** (funcionários) e **DEPARTMENTS** (departamentos). Essas duas tabelas possuem uma coluna em comum, a coluna **DEPARTMENT\_ID**.

EMPLOYEES

EMPLOYEE_ID	FIRST_NAME	DEPARTMENT_ID
100	Steven	90
101	Neena	90
102	Lex	90
103	Alexander	60
104	Bruce	60
105	David	60
106	Valli	60
107	Diana	60
108	Nancy	100
109	Daniel	100
110	John	100
111	Ismael	100
112	Jose Manuel	100
113	Luis	100
114	Den	30
115	Alexander	30

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME
10	Administration
20	Marketing
30	Purchasing
40	Human Resources
50	Shipping
60	IT
70	Public Relations
80	Sales
90	Executive

# Banco de Dados Relacional

000

Por meio da coluna **DEPARTMENT\_ID** é possível estabelecer uma relação entre as tabelas. Repare que o Steven, de **EMPLOYEE\_ID** igual a 100 da tabela **EMPLOYEES** é do departamento 90. Mas qual é o departamento 90?

O departamento 90 é o Executive, que podemos verificar na tabela **DEPARTMENTS**.

EMPLOYEES

EMPLOYEE_ID	FIRST_NAME	DEPARTMENT_ID
100	Steven	90

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME
90	Executive

# Banco de Dados Relacional

000

Relacionar as tabelas de um banco de dados relacional é a sua principal característica. Por meio de uma coluna em comum, conhecida como **chave**, é possível conectar as informações de duas ou mais tabelas.

Você deve lembrar que essas chaves podem ser identificadas como **CHAVE PRIMÁRIA** (ou PK, do inglês *primary key*) e **CHAVE ESTRANGEIRA** (ou FK, do inglês *foreign key*). Se você não lembra, pode ficar tranquilo pois mais a frente vamos voltar neste assunto.

EMPLOYEES

EMPLOYEE_ID	FIRST_NAME	DEPARTMENT_ID
100	Steven	90
101	Nneena	90
102	Lex	90
103	Alexander	60
104	Bruce	60
105	David	60
106	Valli	60
107	Diana	60
108	Nancy	100
109	Daniel	100
110	John	100
111	Ismael	100
112	Jose Manuel	100
113	Luis	100
114	Den	30
115	Alexander	30

Chave Estrangeira

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME
10	Administration
20	Marketing
30	Purchasing
40	Human Resources
50	Shipping
60	IT
70	Public Relations
80	Sales
90	Executive

Chave Primária

# Conceitos básicos de bancos de dados relacionais

000

Uma tabela dentro de um banco de dados é dividida colunas e linhas, temos muito mais definições do que se imagina.

Observe a imagem ao lado.

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	SALARY	DEPARTMENT_ID
166	Sundar	Ande	SANDE	6400	80
167	Amit	Banda	ABANDA	6200	80
168	Lisa	Ozer	LOZER	11500	80
169	Harrison	Bloom	HBLOOM	10000	80
170	Tayler	Fox	TFOX	9600	80
171	William	Smith	WSMITH	7400	80
172	Elizabeth	Bates	EBATES	7300	80
173	Sundita	Kumar	SKUMAR	6100	80
174	Ellen	Abel	EABEL	11000	80
175	Alyssa	Hutton	AHUTTON	8800	80
176	Jonathon	Taylor	JTAYLOR	8600	80
177	Jack	Livingston	JLIVINGS	8400	80
178	Kimberely	Grant	KGRANT	7000	(null)
179	Charles	Johnson	CJOHNSON	6200	80
180	Winston	Taylor	WTAYLOR	3200	50
181	Jean	Fleaur	JFLEAUR	3100	50
182	Martha	Sullivan	MSULLIVA	2500	50
183	Girard	Geoni	GGEONI	2800	50
184	Nandita	Sarchand	NSARCHAN	4200	50
185	Alexis	Bull	ABULL	4100	50
186	Julia	Dellinger	JDELLING	3400	50

1

2

3

4

## 1 Chave Primária

Uma coluna que identifica de forma única cada linha da tabela. É composta por valores que nunca se repetem. Ela será fundamental para se criar relações com outras tabelas.

## 2 Campo

É o nome dado para uma coluna de uma tabela. A coluna sempre será de um tipo específico: número, texto, data.

## 3 Chave Estrangeira

Nome dado à coluna que irá se relacionar com a chave primária de uma outra tabela. No exemplo ao lado, a coluna DEPARTMENT\_ID possibilitará a relação com uma tabela de departamentos..

## 4 Registro

É o nome dado para uma linha de uma tabela.

# A história da Linguagem SQL

Agora voltando aos dois protótipos de sistema relacional mencionados anteriormente:

1. Ingres, desenvolvido pela UCB.
2. System R, desenvolvido pela IBM.

Cada um desses protótipos usava linguagens para consultas aos bancos de dados. Respectivamente, as linguagens QUEL e SEQUEL. Mas, com a popularização dos modelos de bancos de dados relacionais e o surgimento de SGBDRs, viu-se a necessidade de se criar um padrão para a linguagem de bancos de dados relacionais.

Foi ai que, na década de 1980, a linguagem foi padronizada pela American National Standards Institute (ANSI), sendo criado o **SQL** (*Structured Query Language* ou, Linguagem de Consulta Estruturada).

# SQL Oracle vs SQL ANSI

000

Apesar da linguagem SQL ter sido padronizada para uso em bancos de dados relacionais, temos várias empresas desenvolvedoras de SGBDRs, tais como Microsoft (SQL Server), IBM (DB2) e a Oracle (Oracle Database e MySQL), assim como SGBDRs de código aberto (como o PostgreSQL, MariaDB e SQLite).

Portanto, é natural que cada SGBD tenha alguma variação do SQL, e é ai que surgem o T-SQL (Transact-SQL, variação do SQL da Microsoft) e o PL-SQL (Procedural Language SQL) variação do SQL para o Oracle).

De qualquer forma, todos os SQLs têm uma mesma base de comandos padrão, proveniente da padronização feita pela ANSI. Por isso, ao começar a aprender SQL, foque em aprender SQL, não se preocupe com as possíveis variações. Conforme você for avançando, você vai entendendo as particularidades existentes entre os diferentes SGBDs.

# MER, SGBD, ACID, CRUD, ... que tal um dicionário?

Ao começar a aprender SQL, nos deparamos com algumas siglas e termos, a começar pela própria sigla SQL. E a cada passo que damos, nos deparamos com outras, como SGBD, ANSI, SQL Server, MySQL, PL-SQL e assim vai.

Vamos agora falar de mais alguns. A ideia é dar uma noção geral, até porque vamos nos aprofundar mais a frente, mas neste momento vale falar brevemente sobre isso.

# SQL

---

Não poderia começar de outro jeito. A primeira sigla é SQL, que significa **Structured Query Language** (Linguagem de Consulta de Dados).

É uma linguagem padrão utilizada em praticamente todos os sistemas de bancos de dados relacionais, tais como: SQL Server, MySQL, PostgreSQL, Oracle, SQLite, Db2, Access, MariaDB, etc.

É conhecida também como **SEQUEL**.

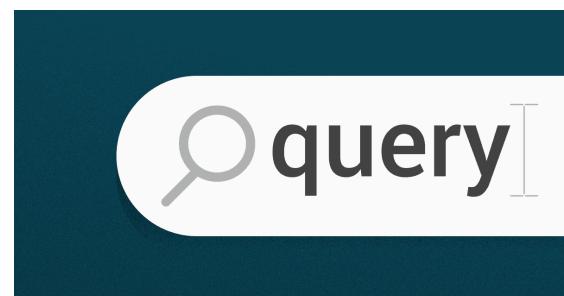


# Query (Consulta)

---

Uma **query** é uma consulta de informações dentro de um banco de dados. É quando a gente “pede” para um banco de dados mostrar as informações que estiverem presentes dentro dele. Esse pedido é feito baseado nos comandos SQL.

Lembre-se: SQL significa *Structured Query Language* ou, traduzindo, Linguagem de **Consulta** Estruturada.



# ANSI e ISO

---

**ANSI** significa *American National Standards Institute* (Instituto Nacional Americano de Padrões). Já a **ISO** significa *International Organization for Standardization* (Organização Internacional de Normalização).

Ambos são responsáveis pela padronização de produtos, sistemas, processos para que possam ser utilizados no mundo todo.



# BD (ou DB)

---

Significa **Banco de Dados** (*Database*). Um banco de dados é um lugar onde armazenamos dados/informações, dentro de tabelas.

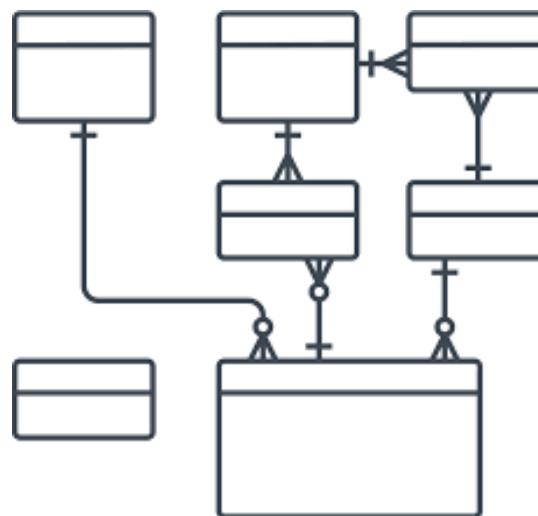


# MER e DER

---

O MER (Modelo Entidade Relacionamento) é um modelo que descreve como vai se dar o relacionamento entre cada entidade (tabela) de um banco de dados. Ou seja, como uma entidade *Produto* se relaciona com uma entidade *Venda*.

Já o DER (Diagrama Entidade Relacionamento) é uma representação gráfica do MER.



# SGBDR

---

SGBDR significa **Sistema de Gerenciamento de Banco de Dados Relacional** (no inglês, RDBMS, *Relational Database Management System*). É um programa que vamos usar para acessar, guardar, criar e gerenciar os Bancos de Dados Relacionais, por meio da linguagem SQL.



# MySQL, SQL Server, PostgreSQL, SQLite, Oracle, Db2, MariaDB, Access

---

São os nomes dos diferentes sistemas de gerenciamento de bancos de dados relacionais (SGBDR/RDBMS) onde podemos usar o SQL.



# T-SQL

---

T-SQL significa Transact-SQL. É uma variação do SQL utilizada no SGBDR da Microsoft: o SQL Server. Possui base na própria linguagem SQL.



# PL-SQL

---

PL-SQL significa Procedural Language-SQL. É uma variação do SQL utilizada no SGBDR da Oracle: o Oracle Database. Possui base na própria linguagem SQL.



# PL/pgSQL

---

PL/pgSQL também significa Procedural Language-SQL. É uma variação do SQL utilizada no SGBDR PostgreSQL. Possui base na própria linguagem SQL.



# DBA

---

DBA significa **Administrador de Bancos de Dados** (*Database Administrator*). Se trata de uma das profissões mais conhecidas na área de bancos de dados.

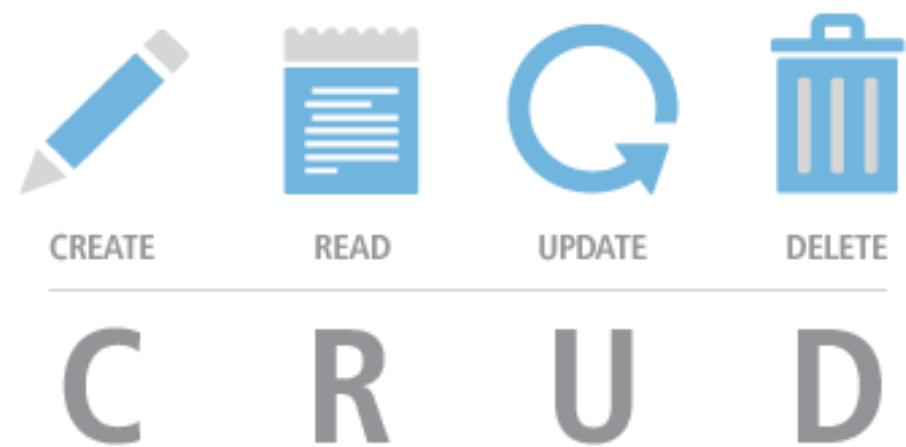


# CRUD

---

Se trata de um conjunto de comandos realizados dentro de bancos de dados, para:

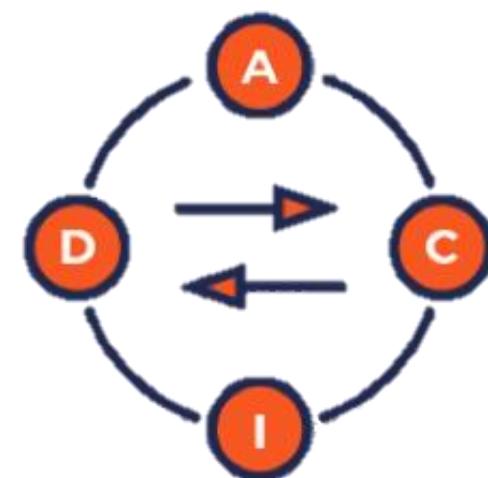
**Criar (Create), Ler (Read), Atualizar (Update) e Deletar (Delete)** dados. Esses nomes não são necessariamente os nomes dos comandos no SQL, apenas resumem a ideia do que esse grupo de comandos faz.



# ACID

---

Significa *Atomicidade, Consistência, Isolamento e Durabilidade*. São propriedades das transações em bancos de dados (como *INSERT*, *UPDATE* e *DELETE*).



# DML, DDL, DQL, DCL e TCL

---

São grupos de comandos dentro do SQL.

- **DML (*Data Manipulation Language*)**: Linguagem de Manipulação de Dados é um conjunto de comandos para manipulação dos dados armazenados dentro das tabelas em um banco de dados:  
Exemplos: INSERT, UPDATE e DELETE.
- **DDL (*Data Definition Language*)**: Linguagem de definição de dados é usada para criar e modificar a estrutura dos objetos armazenados em um banco de dados.  
Exemplos: ALTER, CREATE, DROP.
- **DQL (*Data Query Language*)**: Linguagem de consulta de dados são os comandos de consulta aos dados armazenados em um banco de dados.  
Exemplo: SELECT (Obs: em alguns lugares esse comando está agrupado no DML).

# DML, DDL, DQL, DCL e TCL

---

São grupos de comandos dentro do SQL.

- **DCL (Data Control Language)**: Linguagem de Controle de Dados é usada para controle de acesso e permissões dos usuários em um banco de dados.

Exemplos: GRANT, REVOKE e DENY.

- **TCL (Transaction Control Language)**: Linguagem de controle de transação são comandos usados para gerenciar as mudanças feitas pelos comandos DML.

Exemplos: COMMIT, ROLLBACK e SAVEPOINT.

# Resumo

Neste módulo, aprendemos que:

- 1 Computadores trouxeram a possibilidade de armazenar grandes volumes de dados, dentro dos chamados Bancos de Dados Relacionais.
- 2 Bancos de Dados Relacionais são conjuntos de tabelas que podem se relacionar, a partir de colunas em comum, chamadas de chaves. Podemos acessar, gerenciar e criar esses bancos de dados dentro dos chamados SGBDRs (Sistemas de Gerenciamento de Bancos de Dados Relacionais).
- 3 O SQL é a linguagem padrão para se trabalhar com Bancos de Dados Relacionais.
- 4 A diversidade de SGBDRs (IBM, Microsoft, Oracle) resultou em algumas variações do SQL, como o T-SQL, PL-SQL e PL/pgSQL. Porém, todos tem a mesma base padronizada do SQL.
- 5 Conforme começamos a aprender sobre SQL e Bancos de Dados, nos deparamos com uma série de termos e siglas com muitos significados.

MÓDULO 2

# INSTALAÇÃO

# INSTALAÇÃO

# INSTALAÇÃO



# Criando conta gratuita no site da Oracle

000

Agora...

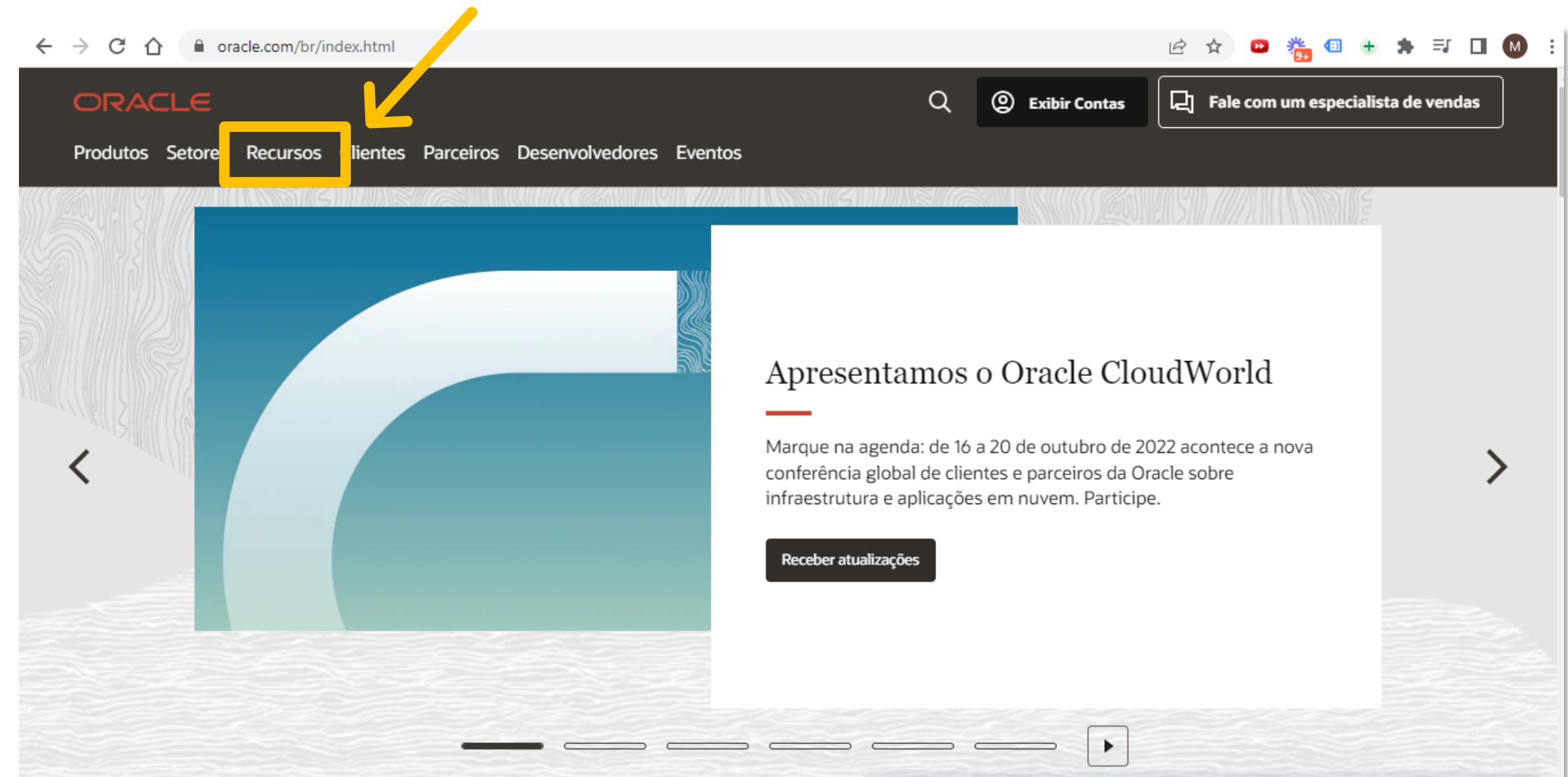
ORACLE®

# Download e instalação do Oracle Database 21c XE

000

Agora que temos uma conta Oracle, vamos acessar o site [oracle.com/br](http://oracle.com/br) para fazer o download do Oracle Database 21c Express Edition.

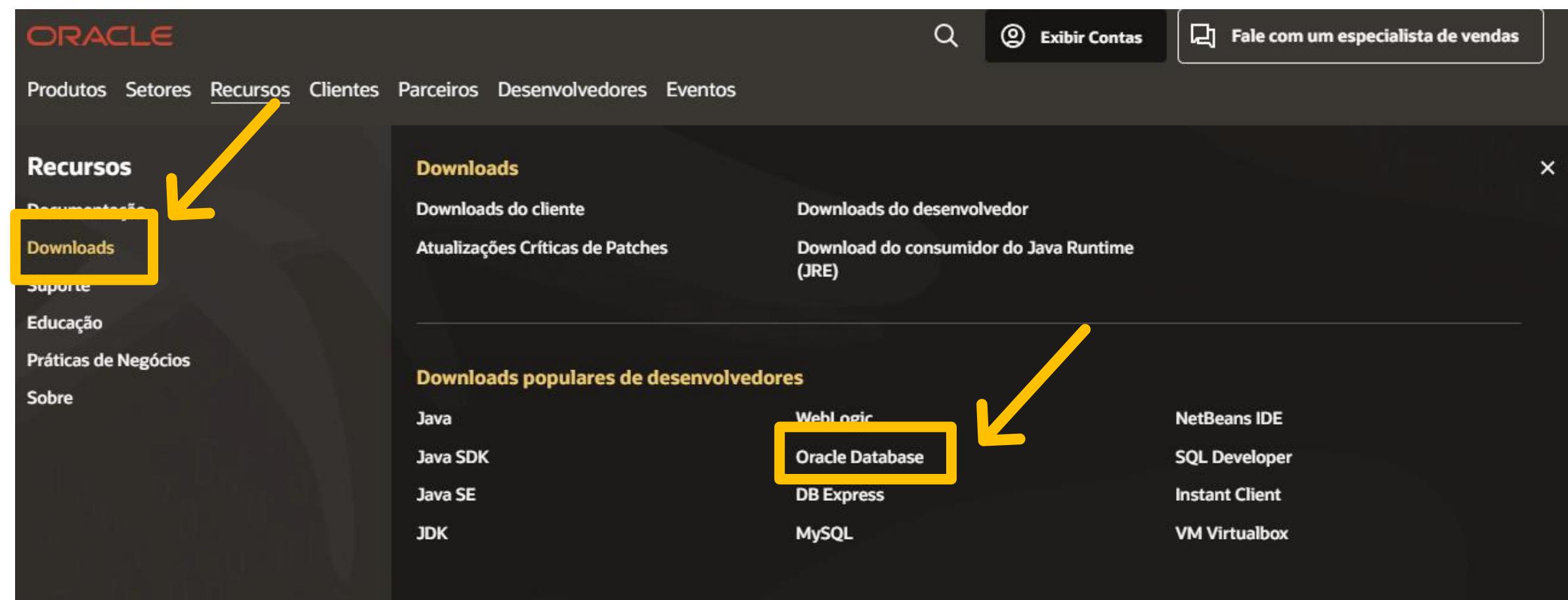
Na página ao lado, vamos clicar em Recursos.



# Download e instalação do Oracle Database 21c XE

000

Em seguida, do lado esquerdo clique em **Downloads** e a seguir em **Oracle Database**.



# Download e instalação do Oracle Database 21c XE

000

Na página seguinte, desça até encontrar a opção Oracle Database Express Edition, que será uma opção gratuita do Oracle. Clique no link destacado na imagem.

The screenshot shows the Oracle Database 21c XE download page. At the top, there's a navigation bar with links for Products, Industries, Resources, Customers, Partners, Developers, and Events. Below the navigation bar, a note states: "Oracle Database 21c is the latest innovation release. For details about database releases and their support timetables, refer to Oracle Support Document 742000.1 (Release Schedule of Current Database Releases) on My Oracle Support." A section titled "21.3 - Enterprise Edition (also includes Standard Edition 2)" is shown, listing three download options: Linux x86-64, HP-UX ia64, and Microsoft Windows x64 (64-bit). The Microsoft Windows entry has a red arrow pointing to it from below. A red box highlights the "Oracle Database Express Edition" link, which is located just above the Microsoft Windows row. This link is also underlined and appears to be the target of the red arrow.

Name	Download	Note
Linux x86-64	<a href="#">ZIP (2.9 GB)</a>   <a href="#">RPM (2.6 GB)</a>	<a href="#">See All</a>
HP-UX ia64	<a href="#">ZIP (3 GB)</a>	<a href="#">See All</a>
Microsoft Windows x64 (64-bit)	<a href="#">ZIP (2.5 GB)</a>	<a href="#">See All</a>

**Oracle Database Express Edition**  
Oracle Database Express Edition

# Download e instalação do Oracle Database 21c XE

000

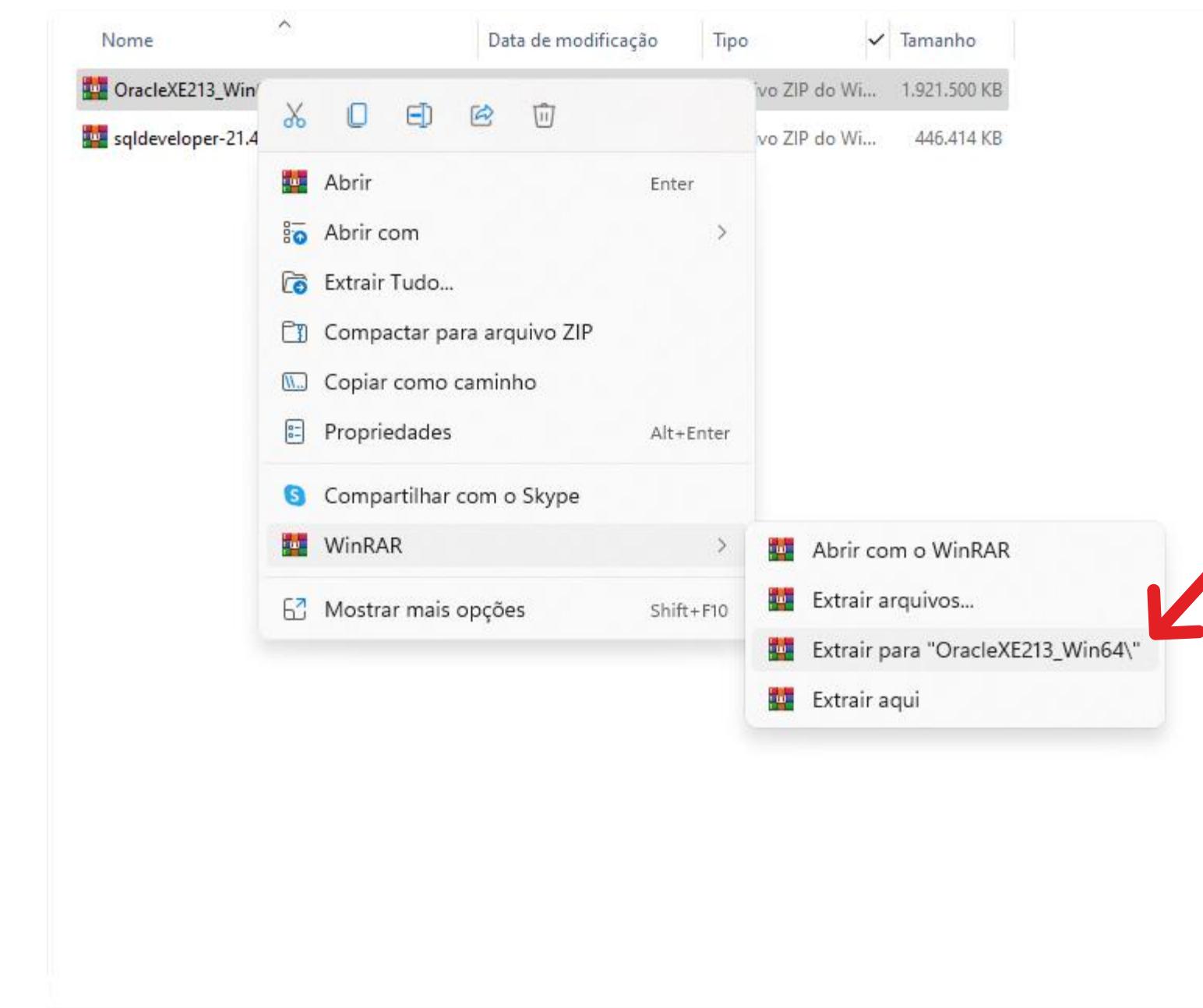
Se a sua versão for Windows, escolha a primeira opção, destacada no print ao lado.

The screenshot shows the Oracle Database Express Edition (XE) Downloads page. At the top, there is a navigation bar with links for Products, Industries, Resources, Customers, Partners, Developers, and Events. Below the navigation bar, the page title is "Database / Technologies / Oracle Database Express Edition (XE) Downloads". The main content area is titled "Oracle Database XE Downloads" and "Oracle Database 21c Express Edition". Under this title, there is a "Download" section containing a button labeled "Oracle Database 21c Express Edition for Windows x64". This button is highlighted with a red box and a red arrow points to it from the left. To the right of the download button, there is a "Description" section which includes file size "(1,967,615,483 bytes - October 08, 2021)" and a Sha256sum value.

# Download e instalação do Oracle Database 21c XE

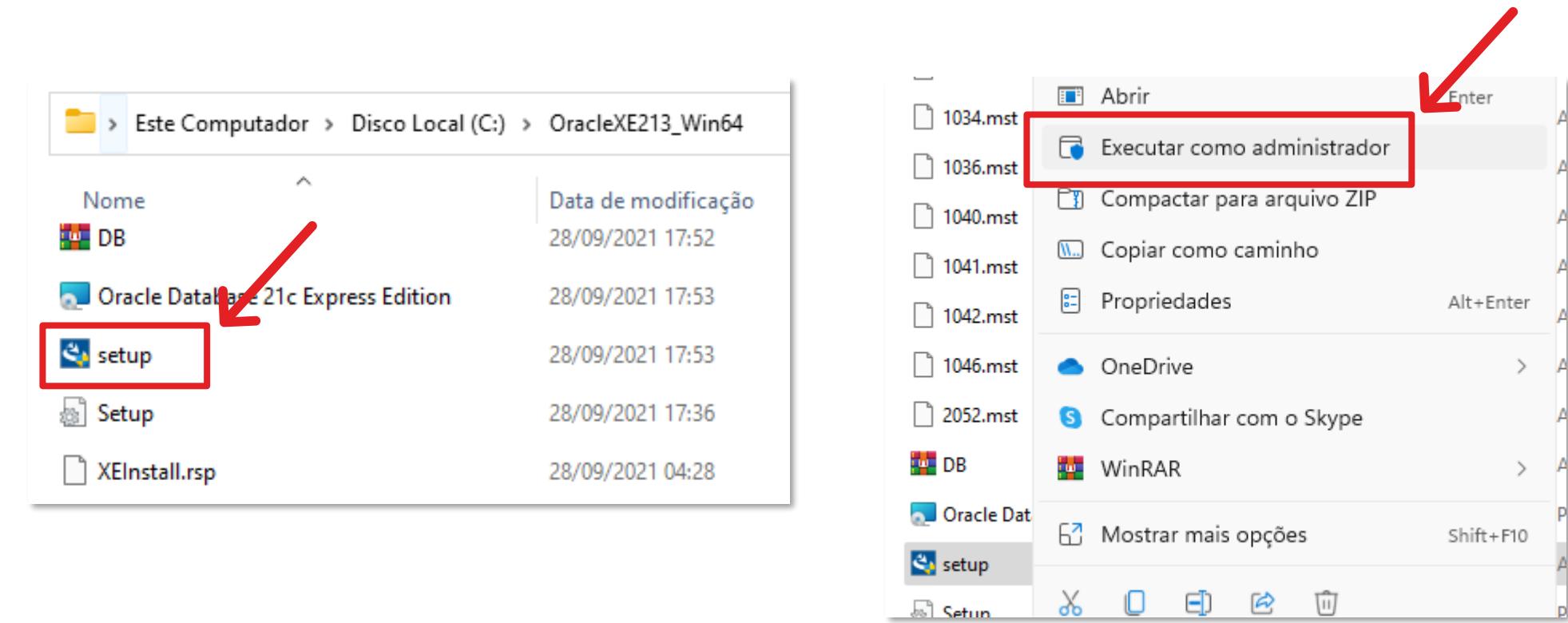
Após o download, precisaremos descompactar o arquivo zip chamado **OracleXE213\_Win64**.

Você pode escolher uma pasta do seu computador para descompactar.



# Download e instalação do Oracle Database 21c XE

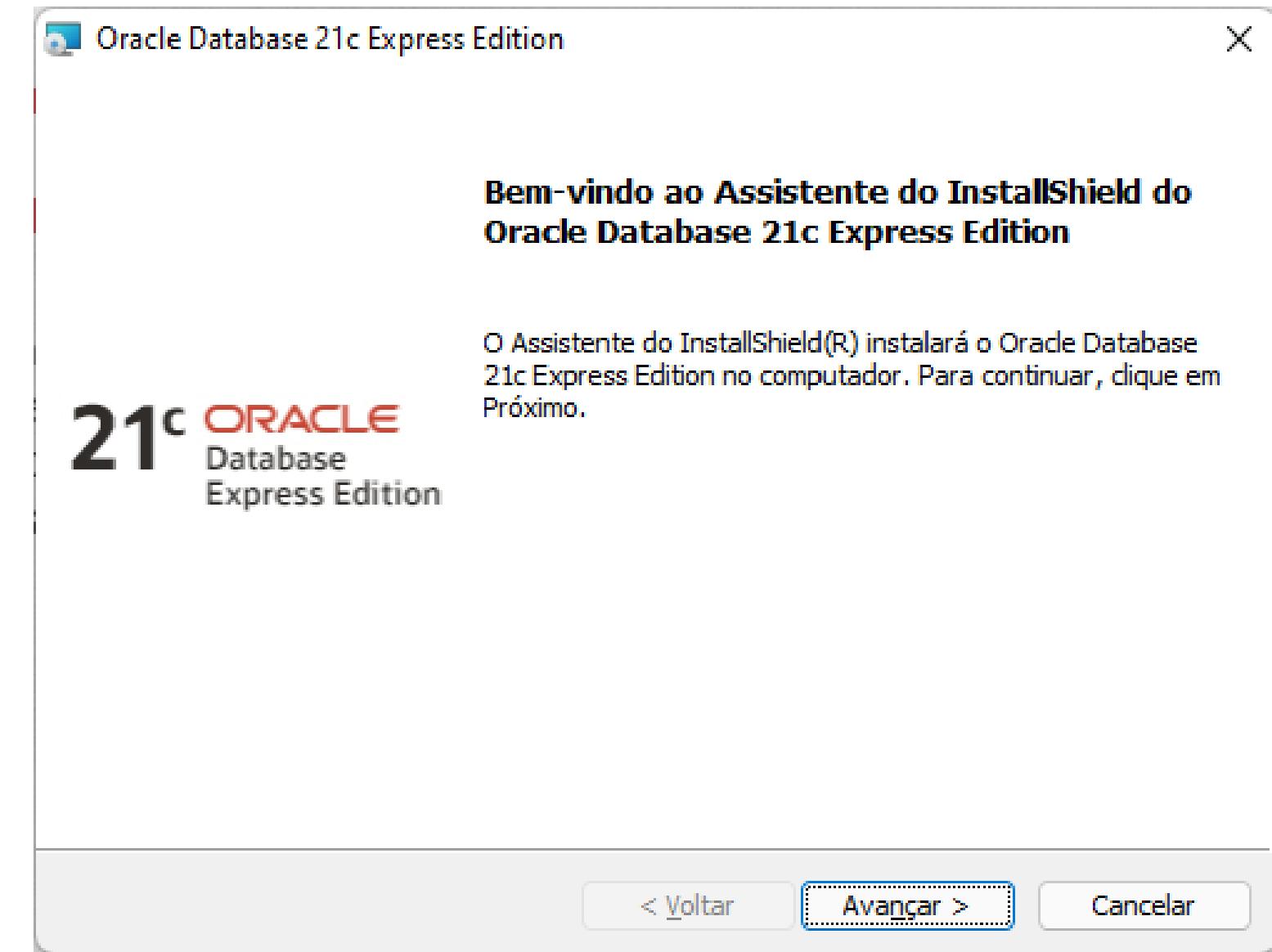
Dentro da pasta descompactada, procure pelo arquivo setup, clique nele com o botão direito e depois em **Executar como administrador**.



# Download e instalação do Oracle Database 21c XE

000

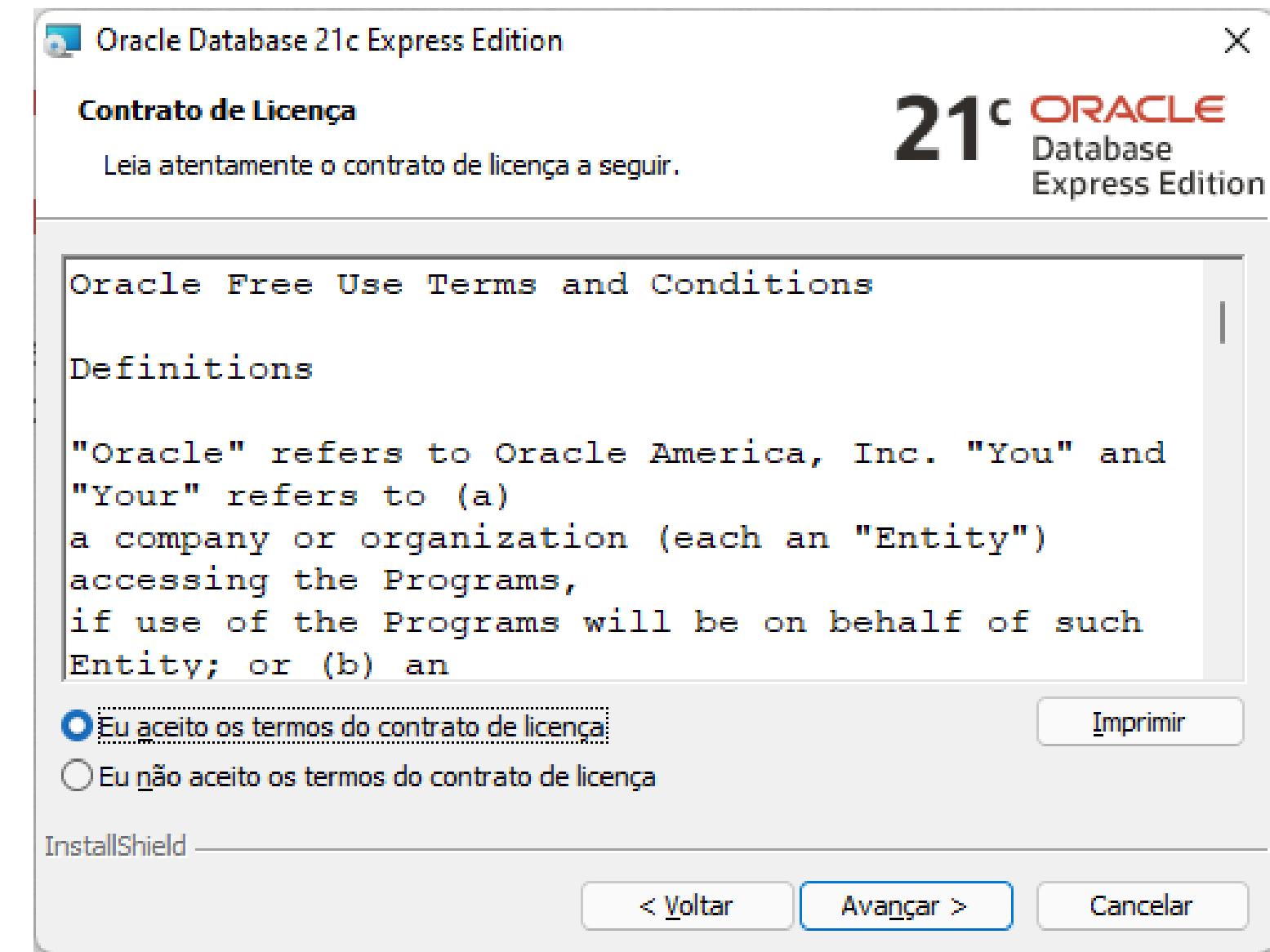
O assistente de instalação será aberto. Clique em Avançar.



# Download e instalação do Oracle Database 21c XE

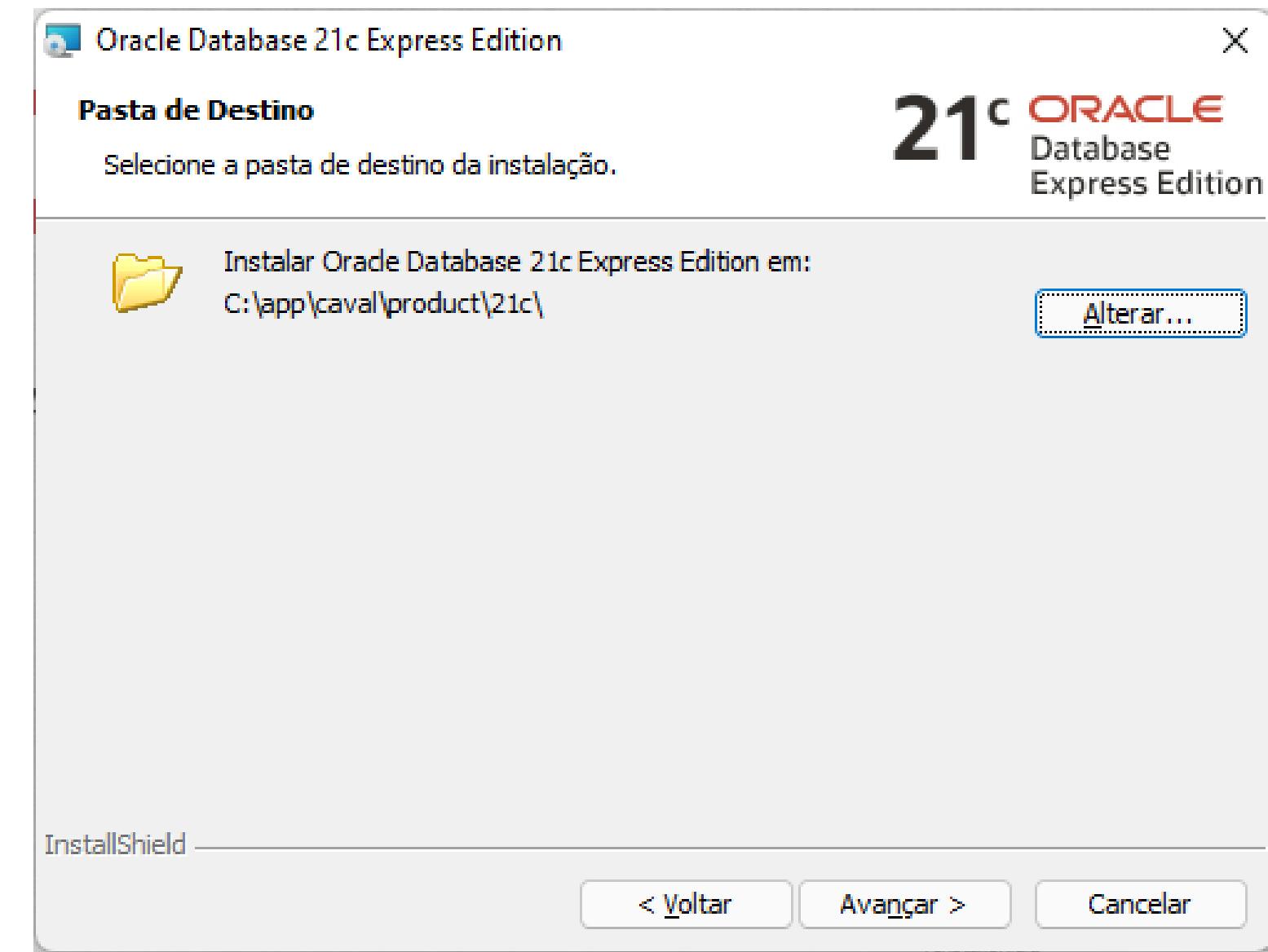
000

Aceite os termos de instalação e clique em **Avançar** novamente.



# Download e instalação do Oracle Database 21c XE

Se desejar, altere a pasta de destino. Aqui vamos deixar o padrão mesmo. Em seguida, clique em Avançar novamente.

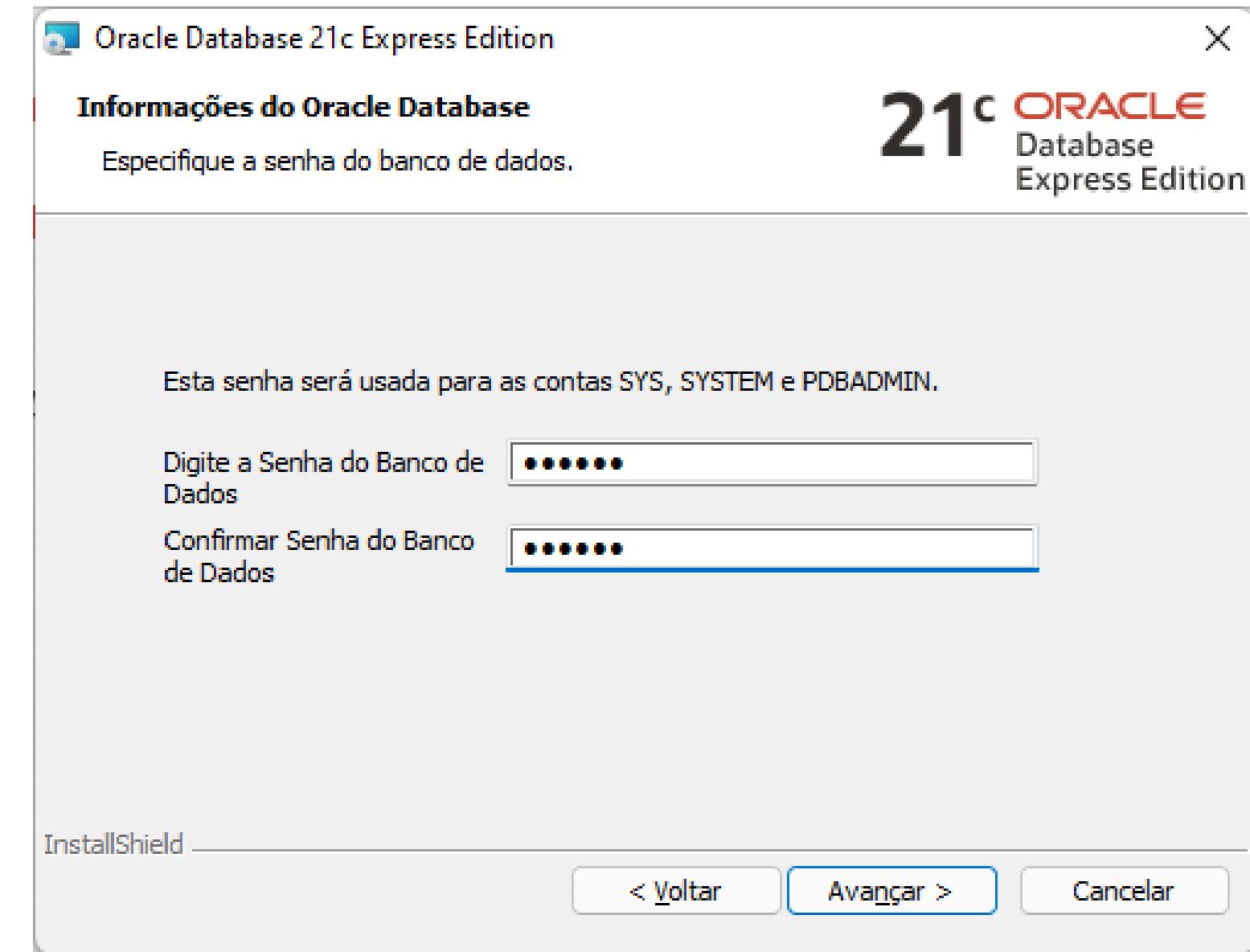


# Download e instalação do Oracle Database 21c XE

Configure uma senha para o banco de dados. Esta senha será usada para as contas SYS, SYSTEM e PDBADMIN.

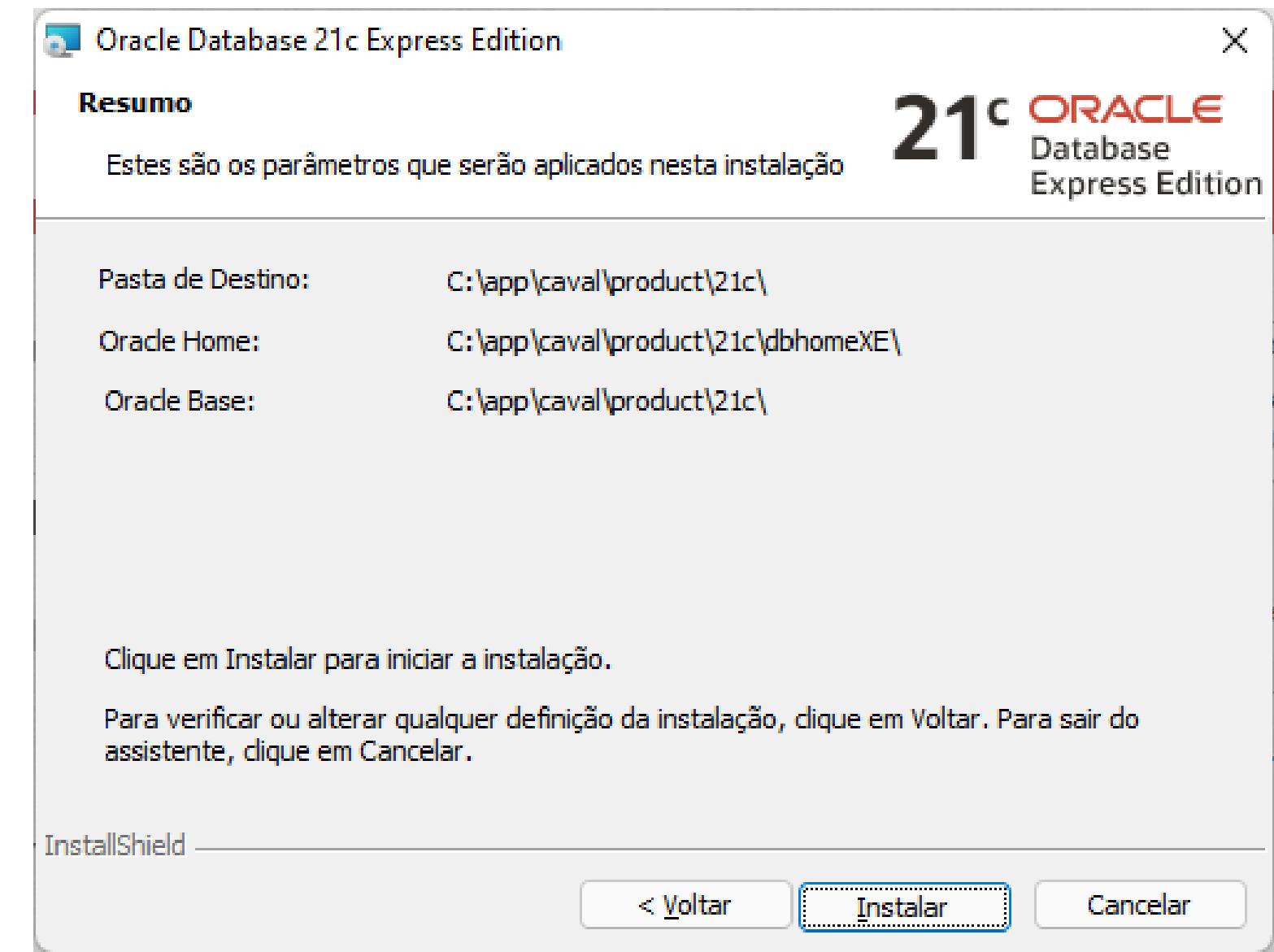
Como aqui estamos em um ambiente para fins didáticos, vamos configurar a senha ‘oracle’. Você pode usar a mesma senha ou escolher uma que desejar. Em seguida, clique em Avançar.

**Importante: anote a sua senha para não correr o risco de esquecê-la e evitar possíveis dores de cabeça no futuro.**



# Download e instalação do Oracle Database 21c XE

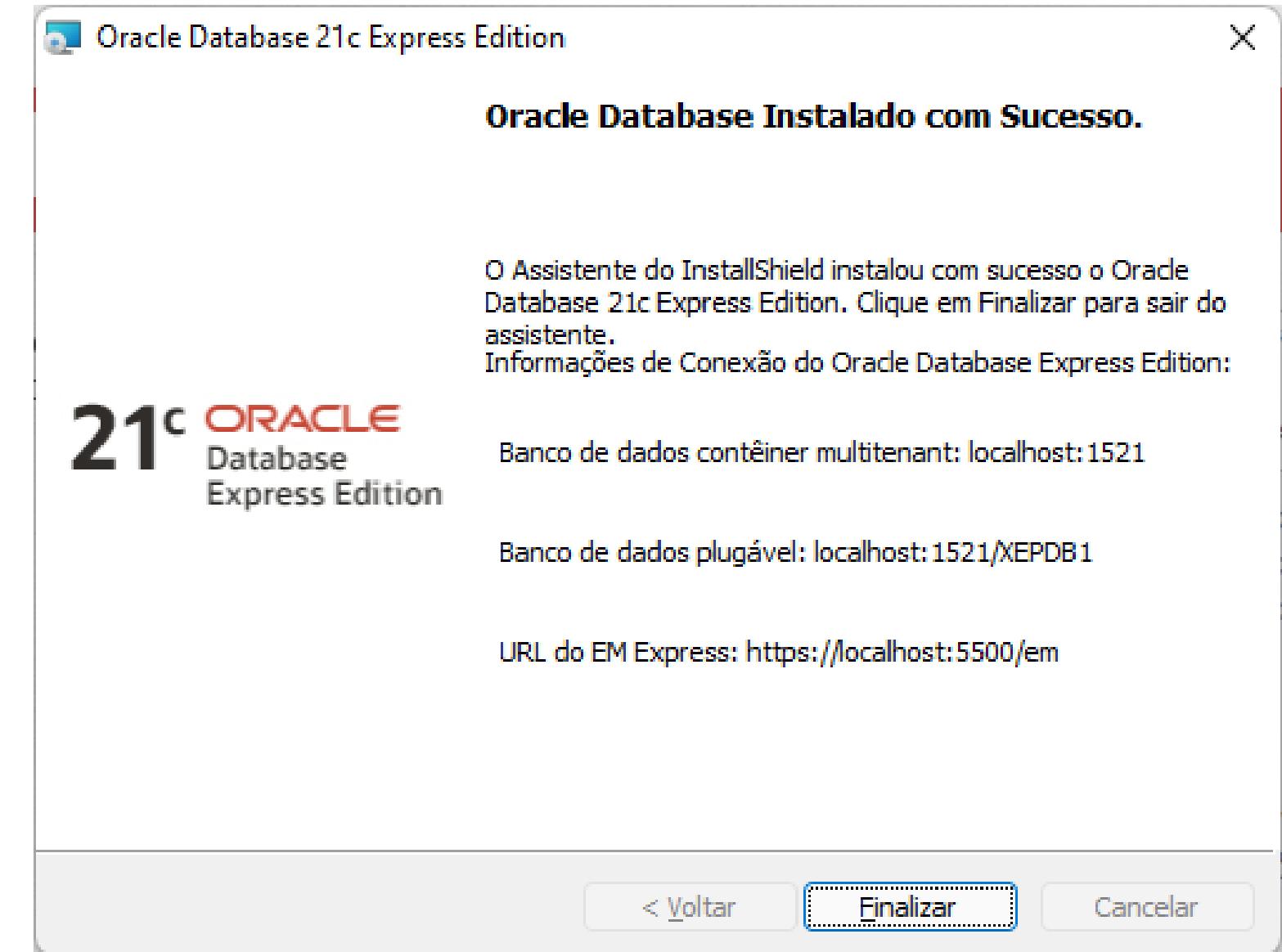
Na janela seguinte, basta clicar em Instalar. A instalação pode levar alguns minutos.



# Download e instalação do Oracle Database 21c XE

000

Por fim, a instalação do Oracle Database 21c XE está concluída. Clique em **Finalizar**.



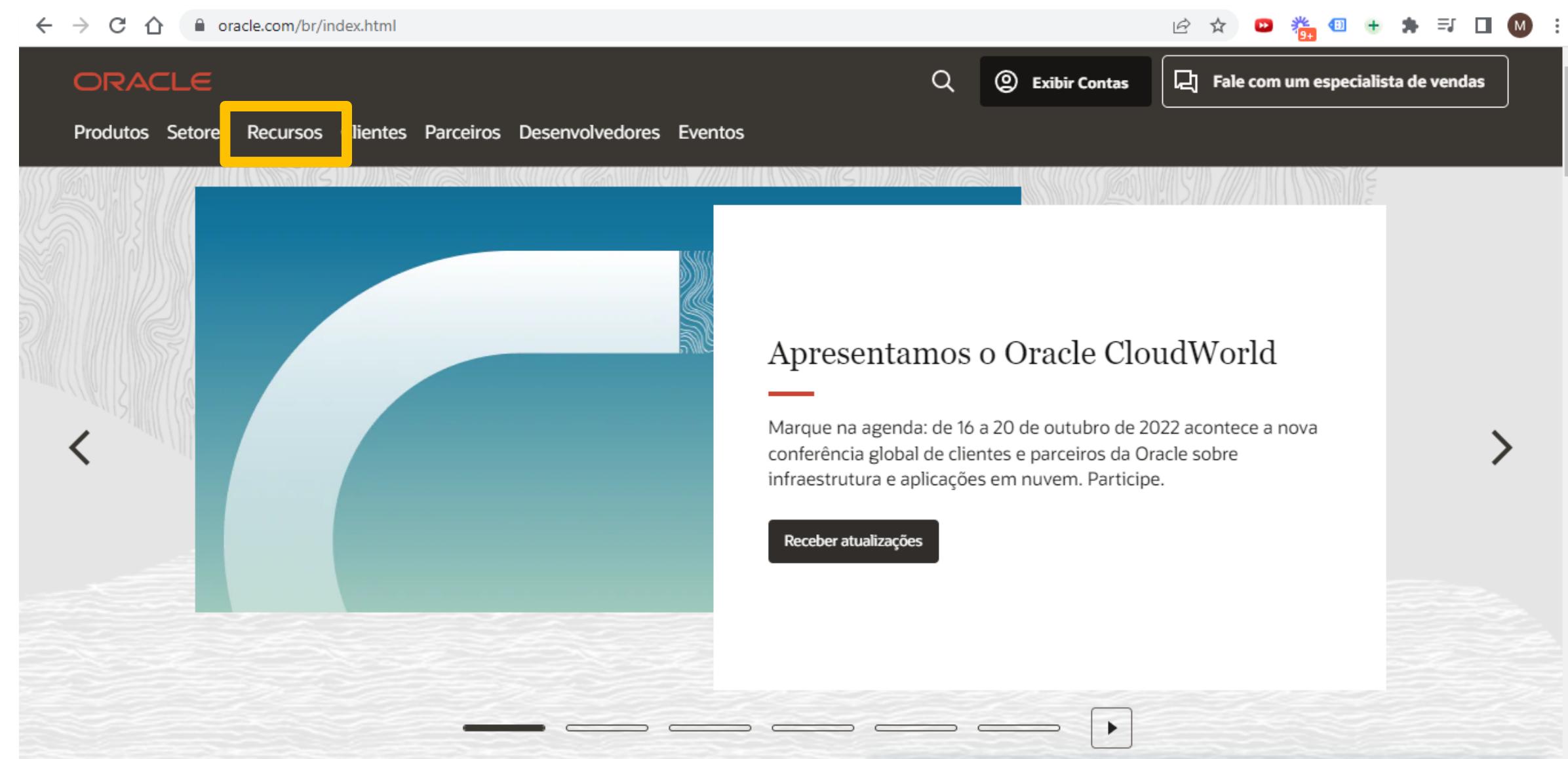
# Download e instalação do Oracle SQL Developer

000

Agora que instalamos o Oracle Database, será necessário instalar a interface onde conseguiremos gerenciar os bancos de dados e utilizar os comandos SQL.

Para isso, vamos acessar o site [oracle.com/br](http://oracle.com/br) para fazer o download desta interface, que no caso será o SQL Developer.

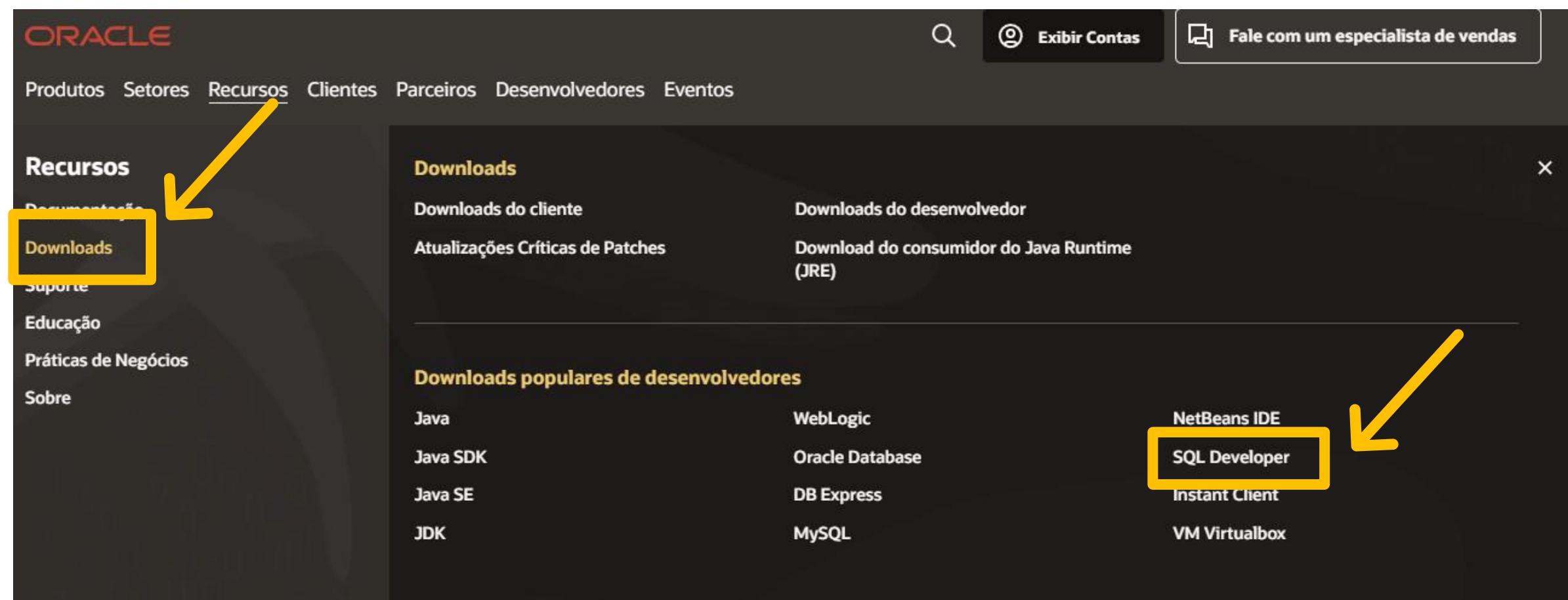
Na página ao lado, vamos clicar em Recursos.



# Download e instalação do Oracle SQL Developer

000

Em seguida, do lado esquerdo clique em **Downloads** e a seguir em **SQL Developer**.



# Download e instalação do Oracle SQL Developer

000

Se a sua versão for Windows, escolha a primeira opção, destacada no print ao lado.

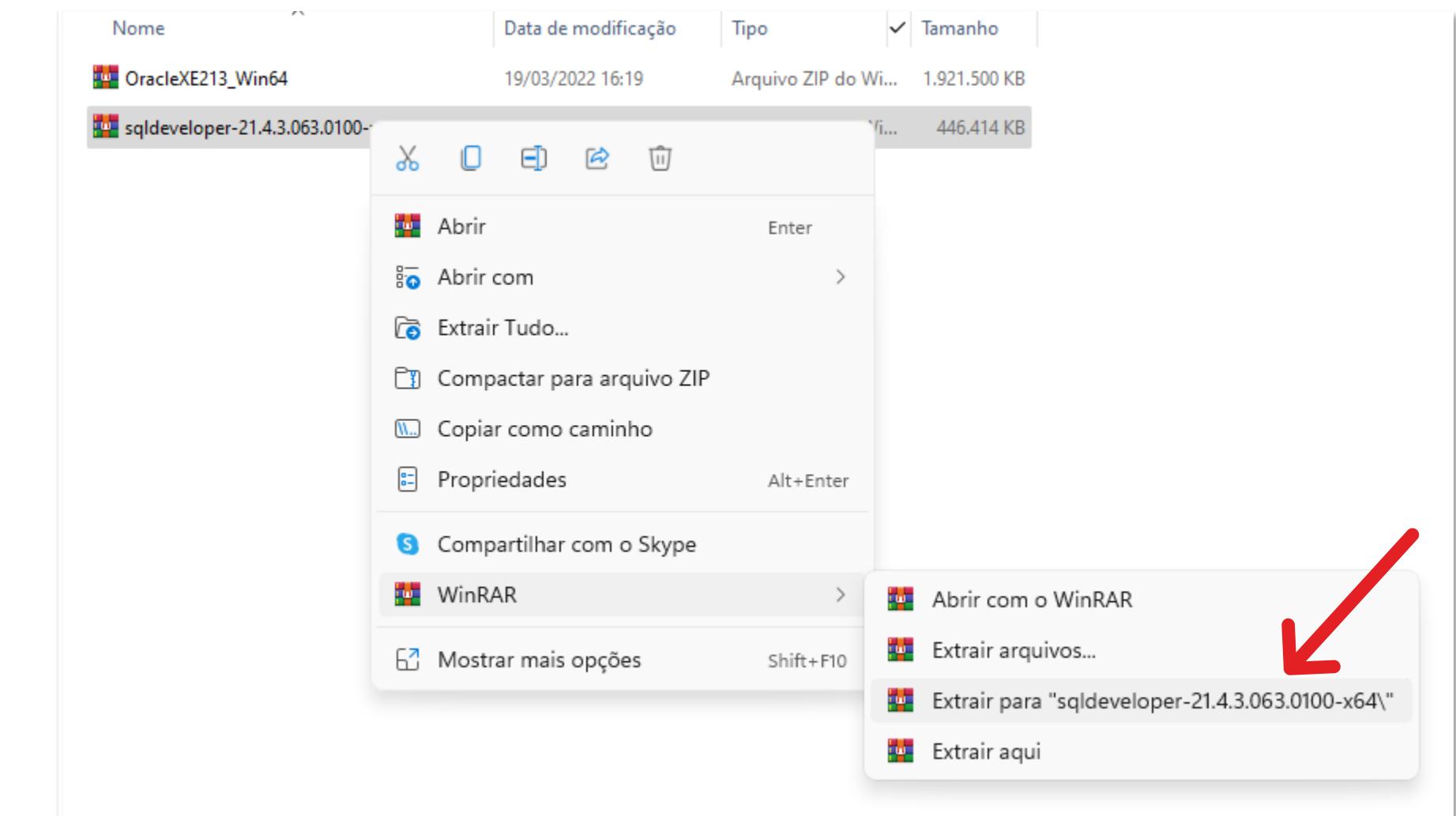
The screenshot shows the Oracle website's download section for SQL Developer 21.4.3. A red box highlights the 'Platform' column for the first row, which specifies 'Windows 64-bit with JDK 8 included'. A red arrow points from this highlighted box towards the 'Download' column, which contains a link labeled 'Download (436 MB)' with a download icon. To the right of the download link is a 'Notes' column containing MD5, SHA1, and Installation Notes information.

Platform	Download	Notes
Windows 64-bit with JDK 8 included	<a href="#">Download (436 MB)</a>	<ul style="list-style-type: none"><li>MD5: 9e091edecad4344e21c5fd0b4844d9a</li><li>SHA1: 3d2b647f24c857d8cc3f3a74a81f66ac4ce58a32</li><li><a href="#">Installation Notes</a></li></ul>

# Download e instalação do Oracle SQL Developer

Após o download, precisaremos descompactar o arquivo zip chamado **sqldeveloper**.

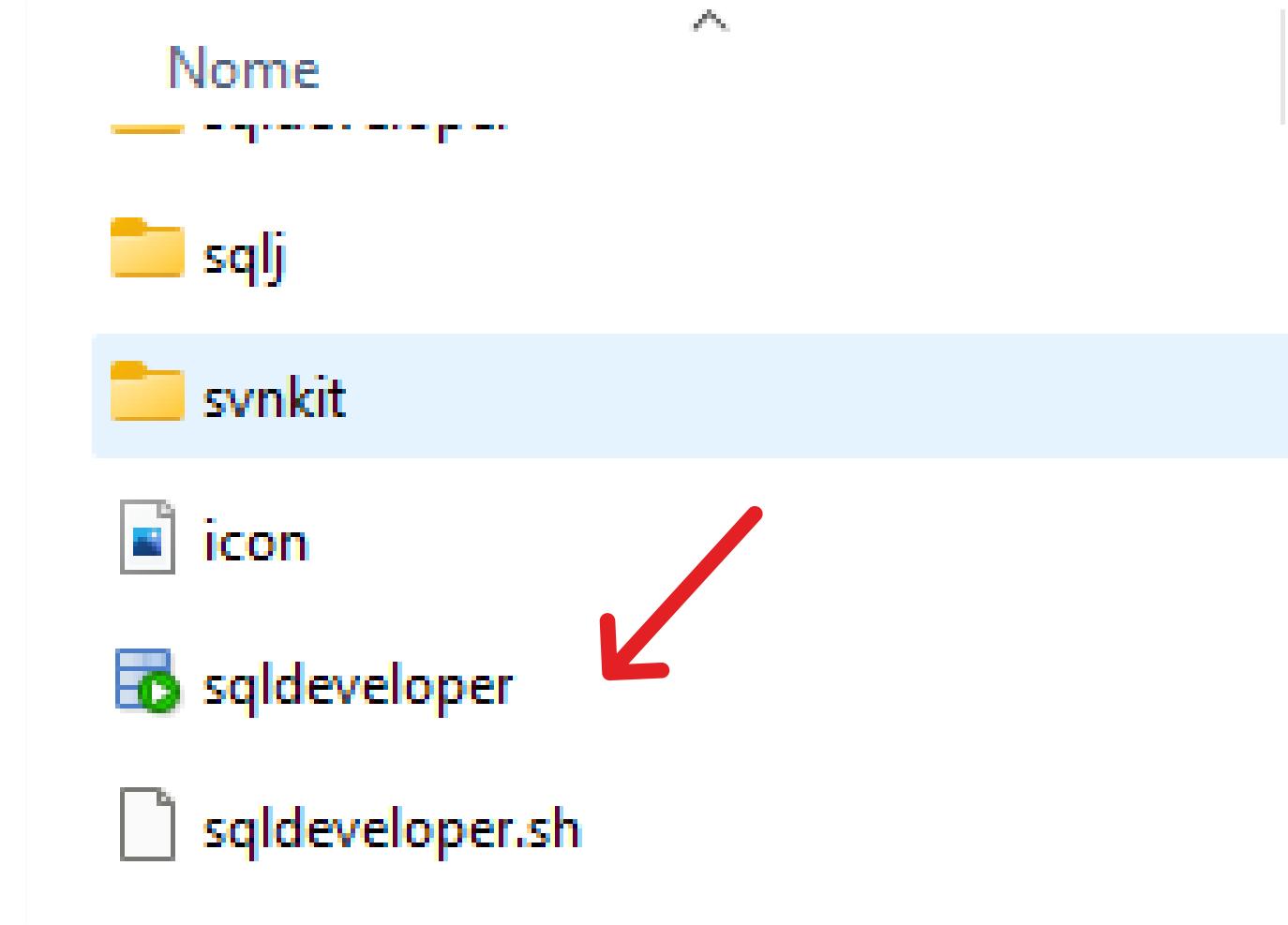
Você pode escolher uma pasta do seu computador para descompactar.



# Download e instalação do Oracle SQL Developer

000

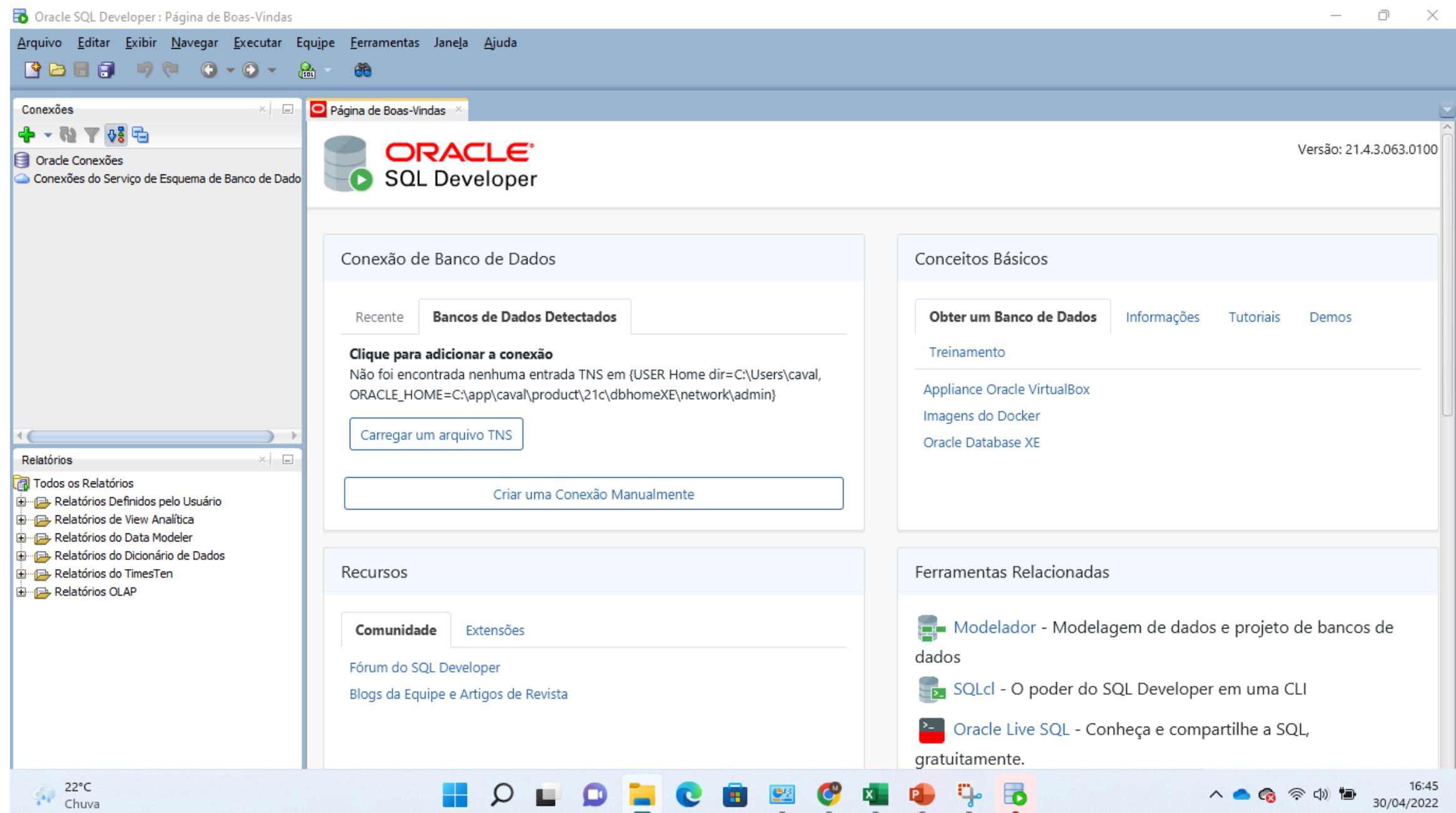
Na pasta que você descompactou os arquivos, procure pelo executável do sqldeveloper.



# Download e instalação do Oracle SQL Developer

000

Pronto! Agora temos tudo o necessário para começar os trabalhos.



# MÓDULO 3

ADMINISTRAÇÃO DE USUÁRIOS

ADMINISTRAÇÃO DE USUÁRIOS

ADMINISTRAÇÃO DE USUÁRIOS

# Sobre contas de usuários

Neste capítulo vamos falar brevemente sobre administração e privilégios de usuários.

Para que as pessoas possam acessar seu banco de dados, você precisa criar usuários e fornecer os devidos privilégios a eles. Imagine que você tem um sistema na sua empresa, e precisa que pessoas possam usá-lo. Dentro desse sistema, será possível fazer uma série de tarefas, porém cada funcionário poderá usar apenas parte desses recursos, ou a sua totalidade, depende da sua função.

Por exemplo, se você tem um sistema de pagamentos, você poderá ter uma série de níveis de privilégios dos usuários que acessam esse sistema. Alguns poderão fazer tudo (normalmente os admins), como criar/editar/excluir usuários, configurar formas de pagamentos, criar novos produtos, visualizar as vendas, reembolsar, e assim vai.

Outros poderão apenas visualizar as vendas e reembolsar, e alguns poderão apenas visualizar as vendas.

Do ponto de vista do Oracle Database, teremos algo semelhante. Na hora de criar um usuário para usar os serviços do banco de dados, é necessário não só dar a ele um login e senha, como também fornecer a ele os privilégios que a sua conta terá.

A Oracle recomenda que você forneça a cada usuário apenas os privilégios necessários para que cada um execute o seu trabalho.

# Usuários pré-definidos

Alguns usuários são criados automaticamente durante a instalação. São eles: **SYS**, **SYSTEM**, **SYSMAN** e **DBSNMP**.

Estas são contas administrativas, ou seja, contas que possuem vários privilégios, como o admin de um sistema. Ao fazer o login no Oracle Database, usamos uma destas contas.

Também veremos que é possível criar novos usuários, configurando diferentes níveis de privilégios a cada um deles.

# Sobre as contas administrativas e privilégios

Contas administrativas permitem a você executar funções administrativas, como gerenciar usuários, gerenciar a memória de bancos de dados, iniciar ou desconectar um banco de dados, dentre outras coisas.

## 1. Usuários SYS e SYSTEM

Estes usuários são criados automaticamente quando você instala o Oracle Database. Ambos são criados com a senha informada durante a instalação e os dois automaticamente recebem os privilégios de um DBA (Administrador de Banco de Dados).

Dentre os dois, o SYS possui mais privilégios sob o ponto de vista do banco de dados.

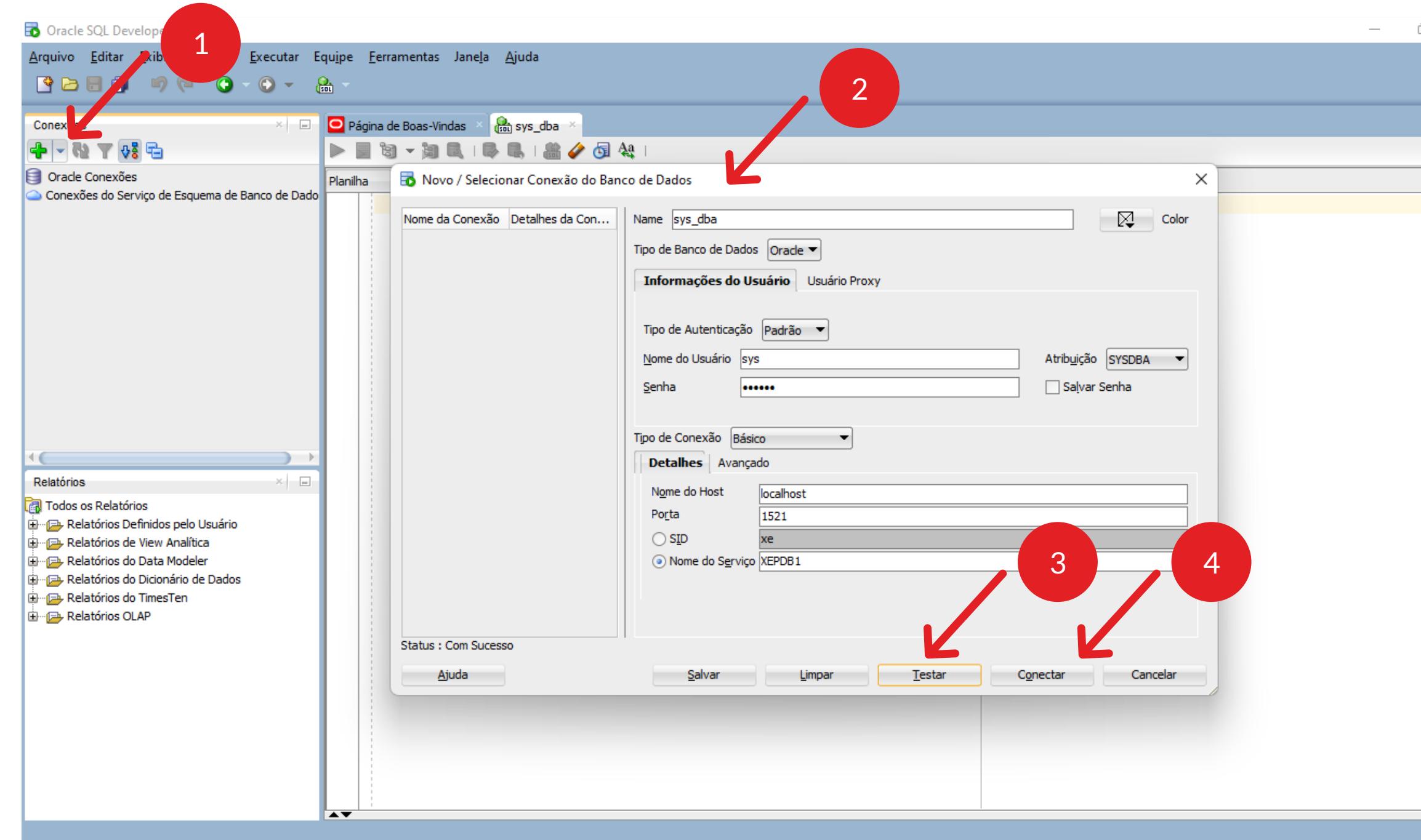
## 2. Privilégios SYSDBA e SYSOPER

Estes são privilégios administrativos, necessários para executar operações de alto nível em um banco de dados, como criar, iniciar, desconectar, criar backup ou recuperar bancos de dados. Dentre os dois privilégios, o SYSDBA é mais completo.

# Criando a conexão sys no Banco de Dados

Vamos então criar a conexão sys dentro do Oracle Database.

- 1 Clique na opção  para criar uma nova conexão para o banco de dados.
- 2 Na janela que se abrir, preencha com os seguintes dados:
  - Nome: sys\_dba
  - Nome do Usuário: sys
  - Senha: oracle (ou a senha que você configurou)
  - Atribuição: SYSDBA
  - Nome do Serviço: XEPDB1
 O resto pode deixar o padrão.
- 3 Clique em testar para testar a conexão. O status deverá aparecer como **Com sucesso**.
- 4 Clique em conectar.



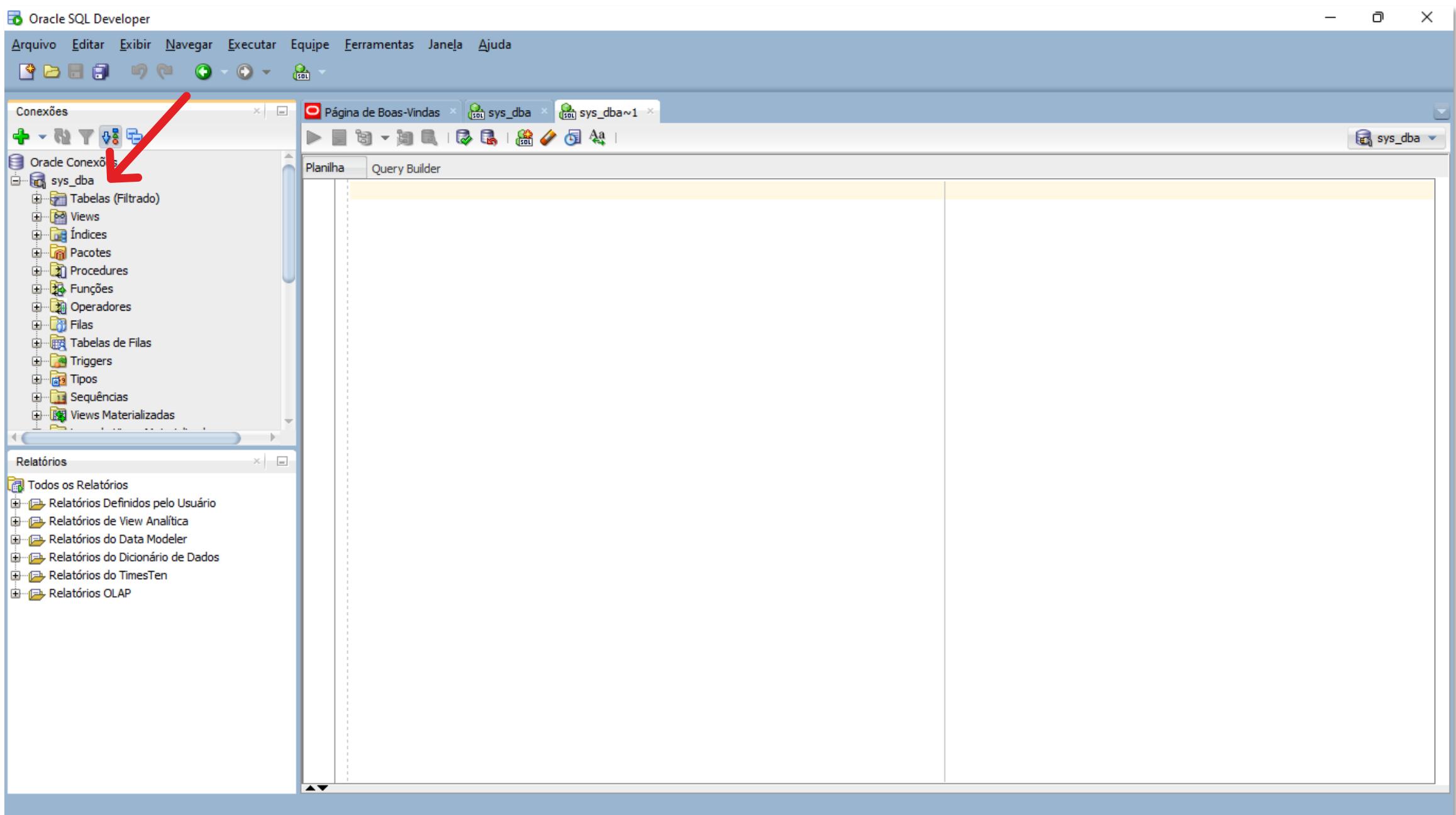
# Criando a conexão sys no Banco de Dados

000

Pronto! Já criamos a nossa conexão sys\_dba que terá os privilégios administrativos.

O próximo passo será criar uma conexão para o banco de dados que vamos usar no curso: o HR.

Mas antes, precisaremos criar um usuário HR, e conceder a ele permissões específicas.



# Privilégios e papéis de usuários

Os privilégios de usuários oferecem níveis de segurança ao que pode ser feito em um banco de dados. Os privilégios são usados para controlar o acesso aos dados e também limitar que tipos de comandos (DDL, DML, DCL, TCL, etc) podem ser executados por aquele usuário.

Quando você cria um usuário, você concede (GRANT) privilégios para permitir o usuário se conectar a um banco de dados, executar queries, criar objetos no banco de dados ou atualizar informações, dentre outras coisas.

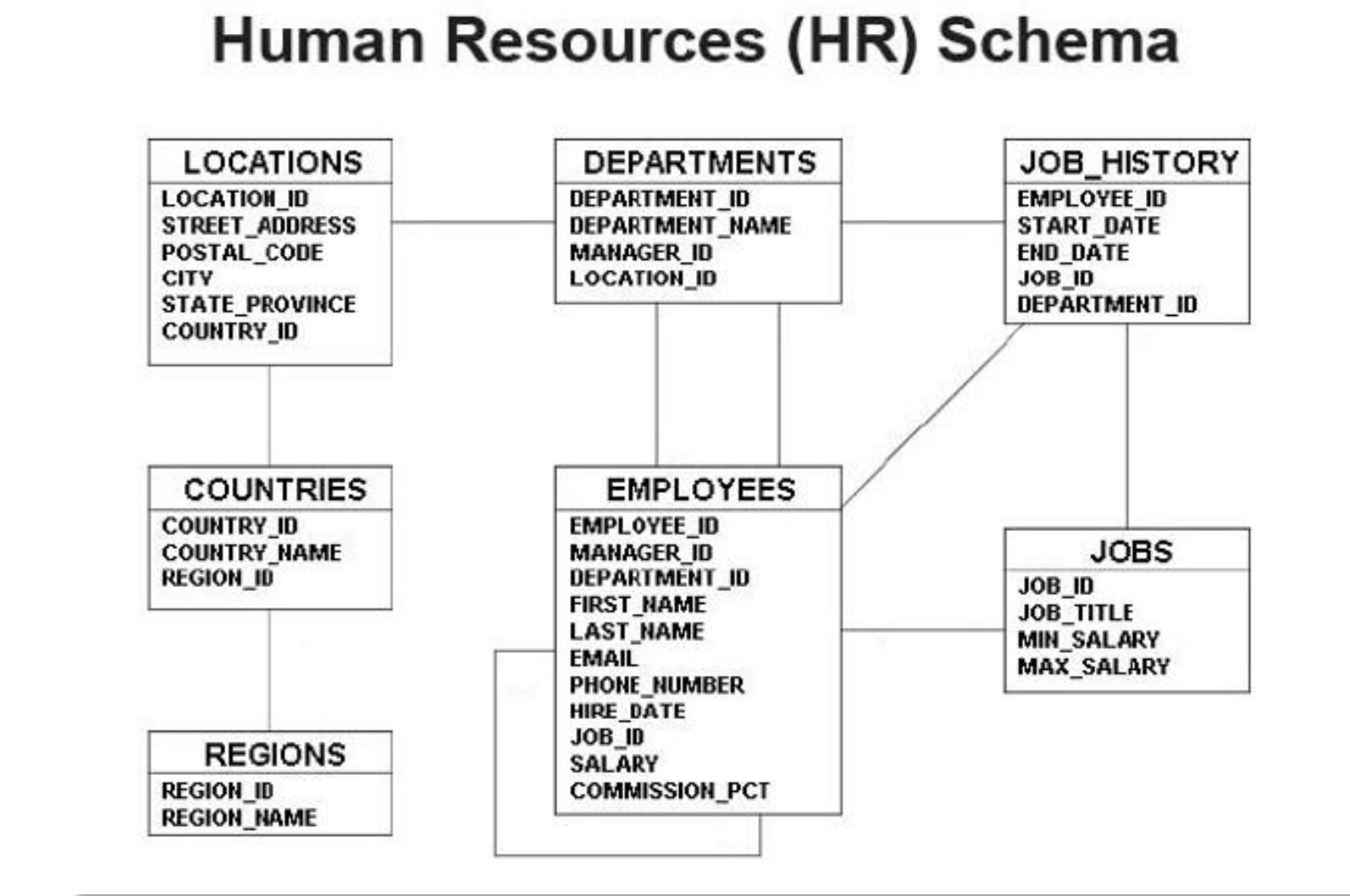
A seguir, vamos criar o usuário HR no Oracle Database. Utilizaremos alguns comandos que nos aprofundaremos apenas mais a frente. Neste primeiro momento, precisaremos apenas criar o usuário para dar continuidade ao curso.

# O schema HR

A Oracle disponibiliza um banco de dados (schema) para fins educacionais. O HR se trata de um banco com informações de uma empresa de Recursos Humanos (Human Resources – HR). Este é um banco de dados relacional, onde as diferentes tabelas possuem alguma relação entre si.

O banco de dados HR possui 7 tabelas, chamadas:

- COUNTRIES
- DEPARTMENTS
- EMPLOYEES
- JOB\_HISTORY
- JOBS
- LOCATIONS
- REGIONS



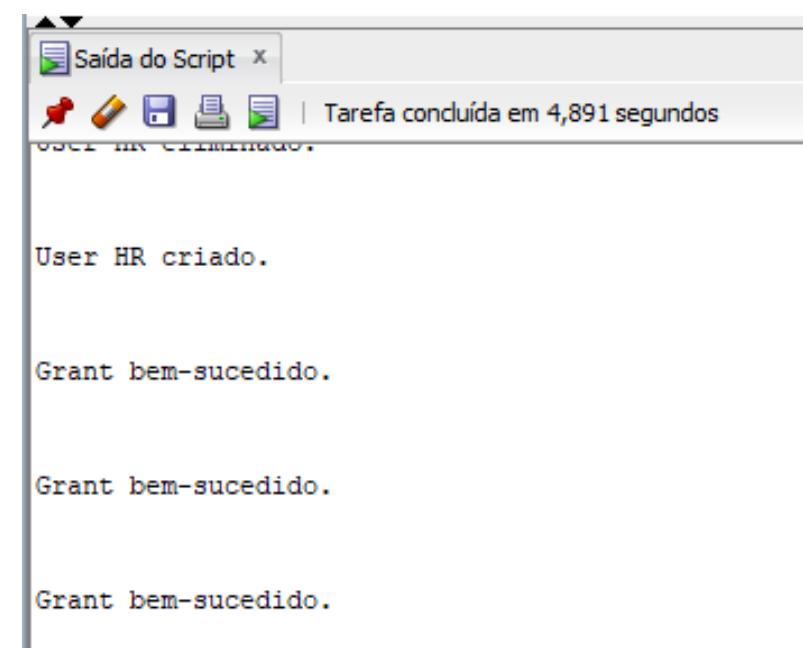
As linhas que ligam as tabelas indicam a relação que elas têm entre si. Por exemplo, a tabela **EMPLOYEES** (funcionários) possui uma relação entre as tabelas **DEPARTMENTS**, **JOB\_HISTORY**, **JOBS** e com ela mesma.

# Criando o usuário HR

Usaremos o código do lado direito para criar o usuário HR.

Clique no botão indicado ao lado para executar este código. Caso ele peça para selecionar a conexão, escolha nossa conexão sys\_dba e clique em Ok.

Após a execução dos comandos, teremos criado o usuário HR.



```

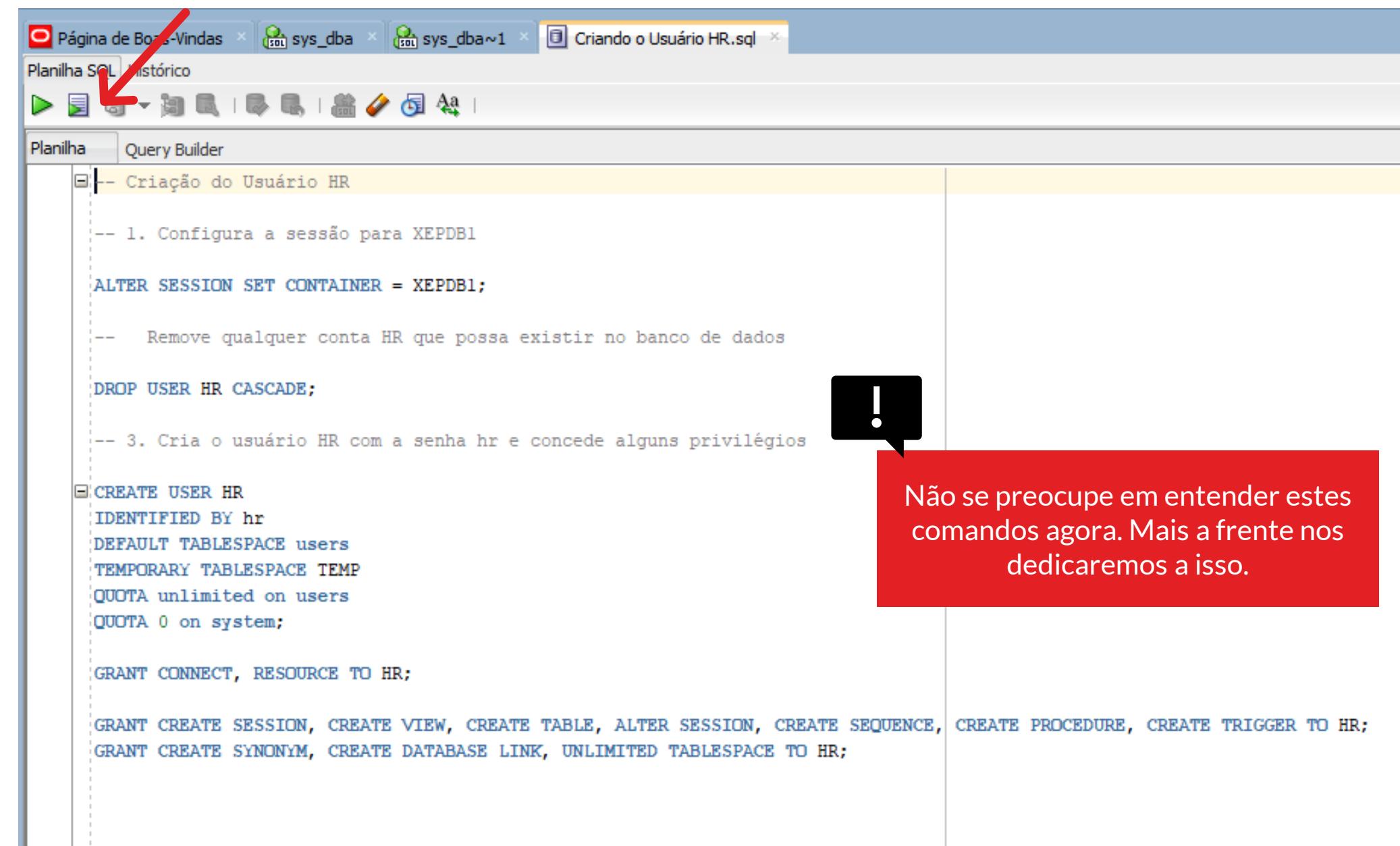
Saída do Script x
Tarefa concluída em 4,891 segundos
User HR criado.

Grant bem-sucedido.

Grant bem-sucedido.

Grant bem-sucedido.

```



```

-- Criação do Usuário HR

-- 1. Configura a sessão para XEPDB1
ALTER SESSION SET CONTAINER = XEPDB1;

-- Remove qualquer conta HR que possa existir no banco de dados
DROP USER HR CASCADE;

-- 3. Cria o usuário HR com a senha hr e concede alguns privilégios
CREATE USER HR
IDENTIFIED BY hr
DEFAULT TABLESPACE users
TEMPORARY TABLESPACE TEMP
QUOTA unlimited on users
QUOTA 0 on system;

GRANT CONNECT, RESOURCE TO HR;

GRANT CREATE SESSION, CREATE VIEW, CREATE TABLE, ALTER SESSION, CREATE SEQUENCE,
CREATE PROCEDURE, CREATE TRIGGER TO HR;
GRANT CREATE SYNONYM, CREATE DATABASE LINK, UNLIMITED TABLESPACE TO HR;

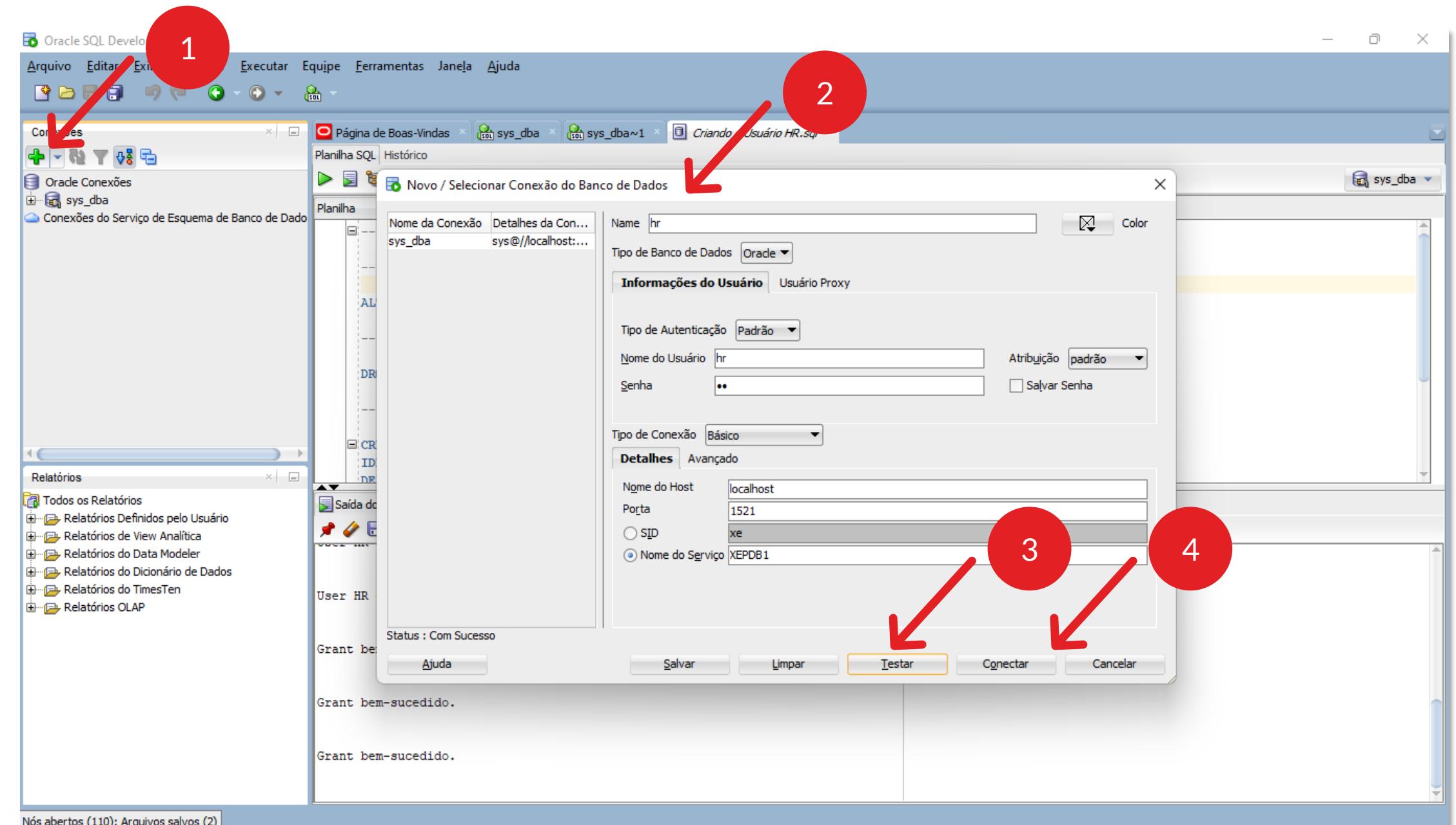
```

Não se preocupe em entender estes comandos agora. Mais a frente nos dedicaremos a isso.

# Criando uma conexão para o schema HR

Vamos então criar o nosso banco de dados HR dentro do Oracle Database.

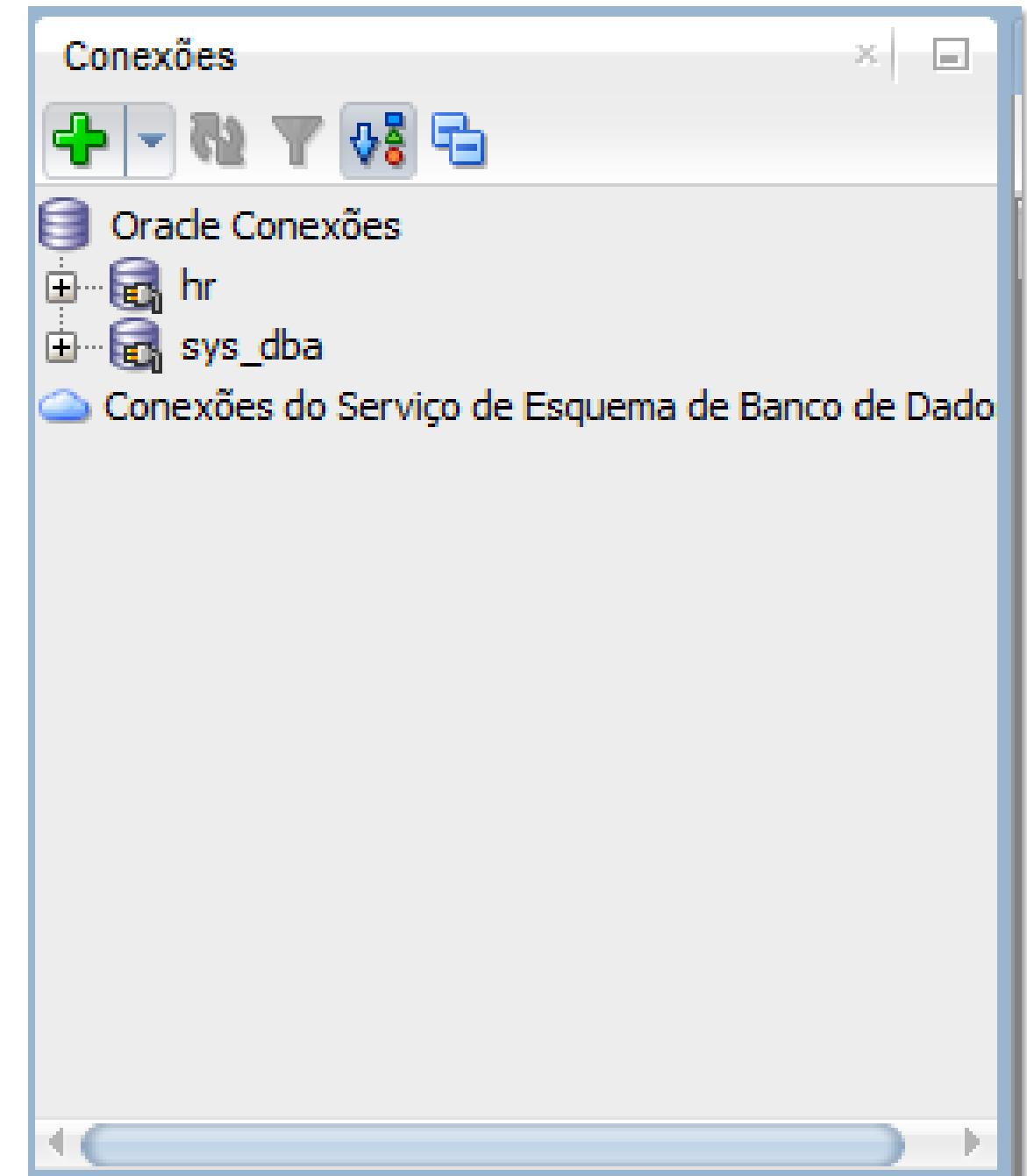
- 1 Clique na opção  para criar uma nova conexão para o banco de dados.
- 2 Na janela que se abrir, preencha com os seguintes dados:
  - Nome: HR
  - Nome do Usuário: hr
  - Senha: hr
  - Atribuição: SYSDBA
  - Nome do Serviço: XEPDB1
 O resto pode deixar o padrão.
- 3 Clique em testar para testar a conexão. O status deverá aparecer como **Com sucesso**.
- 4 Clique em conectar.



# Criando uma conexão para o schema HR

Pronto! Agora temos a nossa conexão HR.

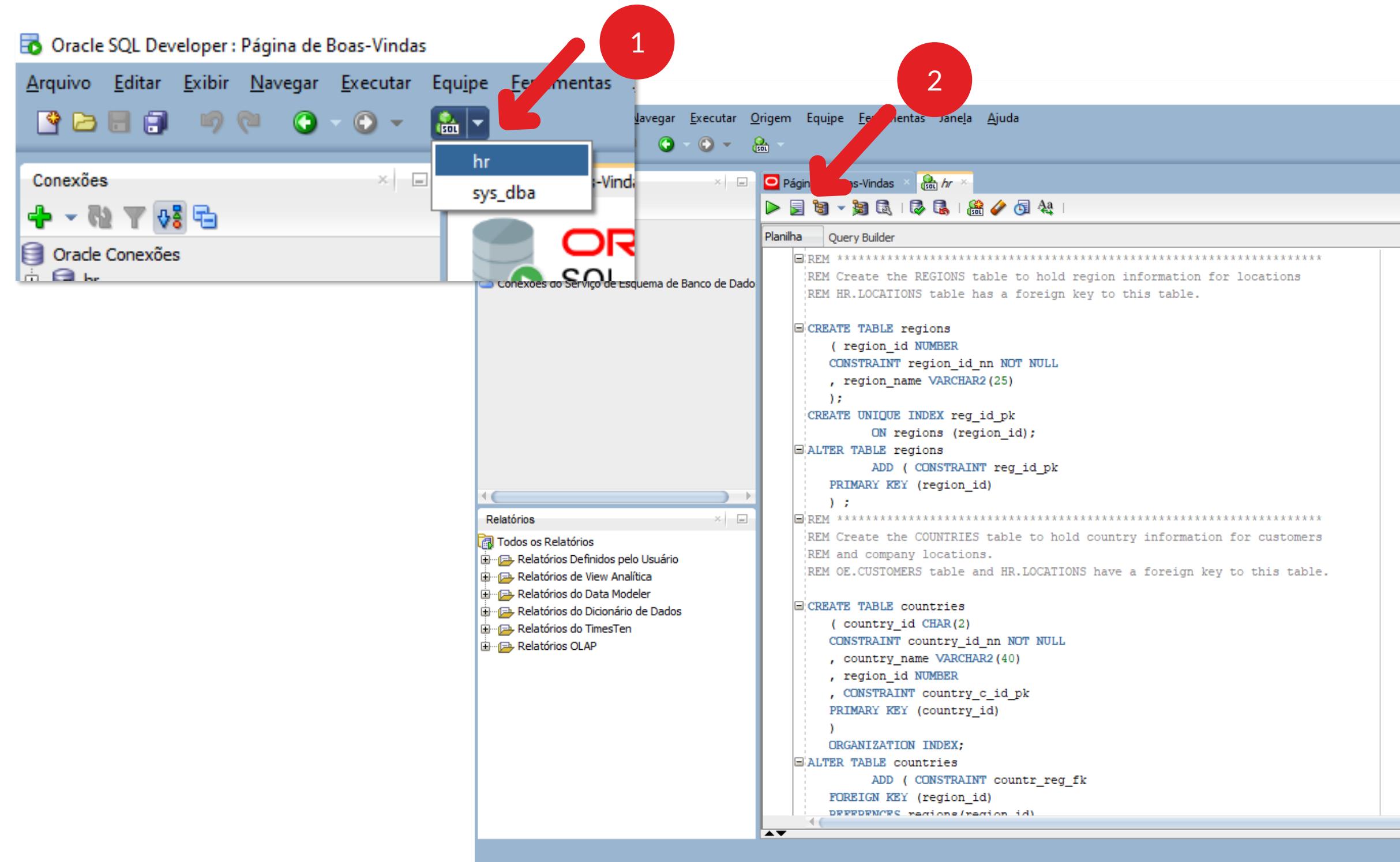
Podemos partir para a criação das tabelas.



# Criando as tabelas do usuário HR

Para criar as tabelas do banco de dados HR, o primeiro passo é criar uma janela de query e escolher a conexão que vamos usar para criar as tabelas. No caso, será a HR.

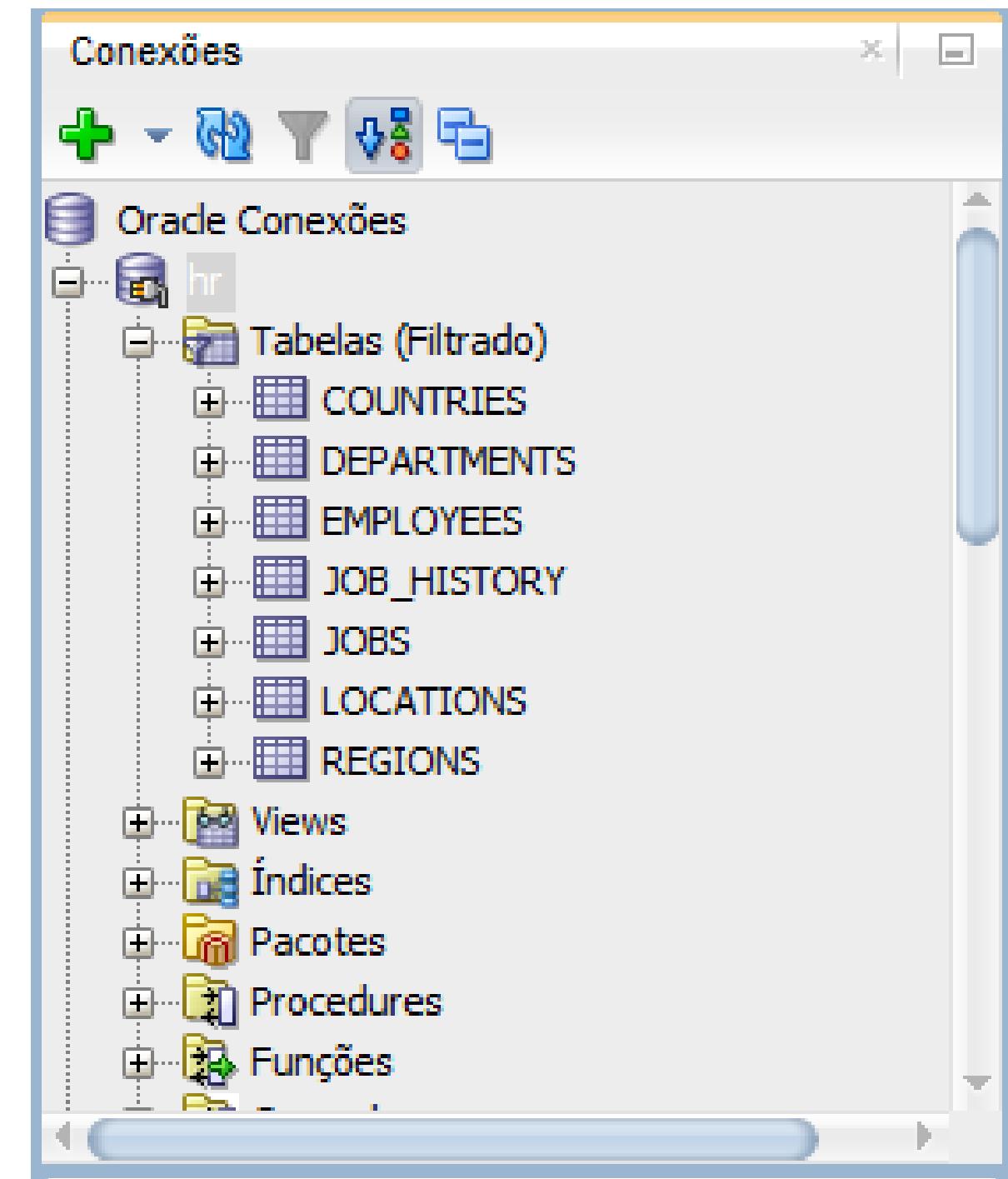
Feito isso, vamos colar o código do arquivo hr\_schema disponibilizado na aula na janela de query e depois clicar em executar.



# Criando as tabelas do usuário HR

Feito isso, podemos ir até o banco de dados HR e expandir as tabelas. Veremos que as nossas 7 tabelas foram criadas com sucesso.

Agora, estamos prontos para começar o curso!



## MÓDULO 4

# INTRODUÇÃO AO SQL

# INTRODUÇÃO AO SQL

# INTRODUÇÃO AO SQL

# Introdução ao SQL

O SQL é a linguagem padrão para se trabalhar com bancos de dados relacionais. Ou seja, conseguimos criar, ler, atualizar, excluir, enfim, manipular dados de forma geral dentro de bancos de dados e tabelas através de comandos em SQL.

Utilizamos o SQL dentro de programas específicos, chamados de SGBDRs. Apenas a título de recordação, no SQL Server usamos o SSMS, no MySQL usamos o Workbench, no PostgreSQL usamos o pgAdmin. No caso da Oracle, o nome do SGBDR utilizado será o **SQL Developer**.

# Introdução ao SQL

000

A linguagem SQL é composta por uma série de grupos de comandos, que inclusive já até introduzimos na seção de dicionário aqui da apostila, no módulo 1. Esses grupos de comando seriam os seguintes:

**DDL**

**Data Definition Language**

**CREATE**

Cria uma nova tabela, view ou outro objeto dentro do banco de dados.

**ALTER**

Modifica um objeto dentro do banco de dados (tabela, view, etc).

**DROP**

Exclui um objeto dentro do banco de dados (tabela, view, etc).

**DML**

**Data Manipulation Language**

**INSERT**

Adiciona uma nova linha em uma tabela.

**UPDATE**

Atualiza os valores das linhas de uma tabela.

**DELETE**

Exclui linhas de uma tabela.

**DCL**

**Data Control Language**

**GRANT**

Dá privilégios a um usuário.

**REVOKE**

Retira privilégios de um usuário.

**DQL**

**Data Query Language**

**SELECT**

Comando de seleção de linhas de uma tabela.

Obs: em algumas referências, o SELECT pode entrar na categoria DML.

**TCL**

**Transaction Control Language**

**COMMIT**

Grava definitivamente no banco os efeitos dos comandos de uma transação (insert, delete e update).

**ROLLBACK**

Desfaz os efeitos dos comandos de transação.

**SAVEPOINT**

É um ponto de salvamento até onde uma transação pode ser revertida.

# Documentação Oracle SQL

000

É possível encontrar uma documentação completa do SQL Oracle neste site: <https://otn.oracle.com>

The screenshot shows the Oracle website homepage. At the top, there is a promotional message about visiting a local Oracle site, with buttons for 'Visite Oracle.com Brasil' and 'Não obrigado, prefiro ficar aqui'. Below this is a navigation bar with links for ORACLE, Products, Industries, Resources, Customers, Partners, Developers, Events, View Accounts, Contact Sales, and Try Oracle Cloud Free Tier. The main content area features three columns: 'Downloads' (Database Downloads, Java Downloads, SQL Developer, Oracle Instant Client, Oracle WebLogic Server), 'Documentation' (Cloud Documentation, Java Documentation, Solution Architectures, Tutorials, Technical Articles), and 'Support' (MyOracle Support Login, Support Policies, Advanced Customer Services, Support Contacts Directory, Partner Support). A red arrow points to the 'All Documentation' button in the Documentation column, which is highlighted with a red border. At the bottom, there are links for Chat with sales and Contact or call, along with the Oracle logo.

# Documentação Oracle SQL

000

Clique em Oracle Database para acessar toda a documentação disponível.

The screenshot shows the Oracle homepage with a grid of icons at the top representing various product categories. Below this is a section titled 'Featured Products' displaying twelve items, each with an icon and a name. The 'Oracle Database' item is highlighted with a red border and a red arrow points to it from the bottom right. The other products listed are Advertising, Application Express (APEX), Autonomous Database, Exadata Cloud at Customer, Financials, GraalVM Enterprise, Human Resources, NetSuite Applications, Integration, Oracle Analytics, Planning, Oracle Database, Supply Chain Management, Verrazzano, and Visual Builder Studio.

Fusion Cloud Applications	Cloud Infrastructure	On-Premises Applications	Middleware	Database	Engineered Systems	Java	Hardware	Operating Systems	Virtualization	Industries	Architecture Center
---------------------------	----------------------	--------------------------	------------	----------	--------------------	------	----------	-------------------	----------------	------------	---------------------

**Featured Products**

- Advertising
- Application Express (APEX)
- Autonomous Database
- Exadata Cloud at Customer
- Financials
- GraalVM Enterprise
- Human Resources
- NetSuite Applications
- Integration
- Oracle Analytics
- Planning
- Oracle Database
- Supply Chain Management
- Verrazzano
- Visual Builder Studio

ORACLE®

 SQL IMPRESSIONADOR | HASHTAG TREINAMENTOS

# Documentação Oracle SQL

000

Vá até a opção Development (como mostrado em 1) e depois em SQL Language Reference (em 2) para acessar um arquivo com um manual de referência da linguagem SQL no Oracle.

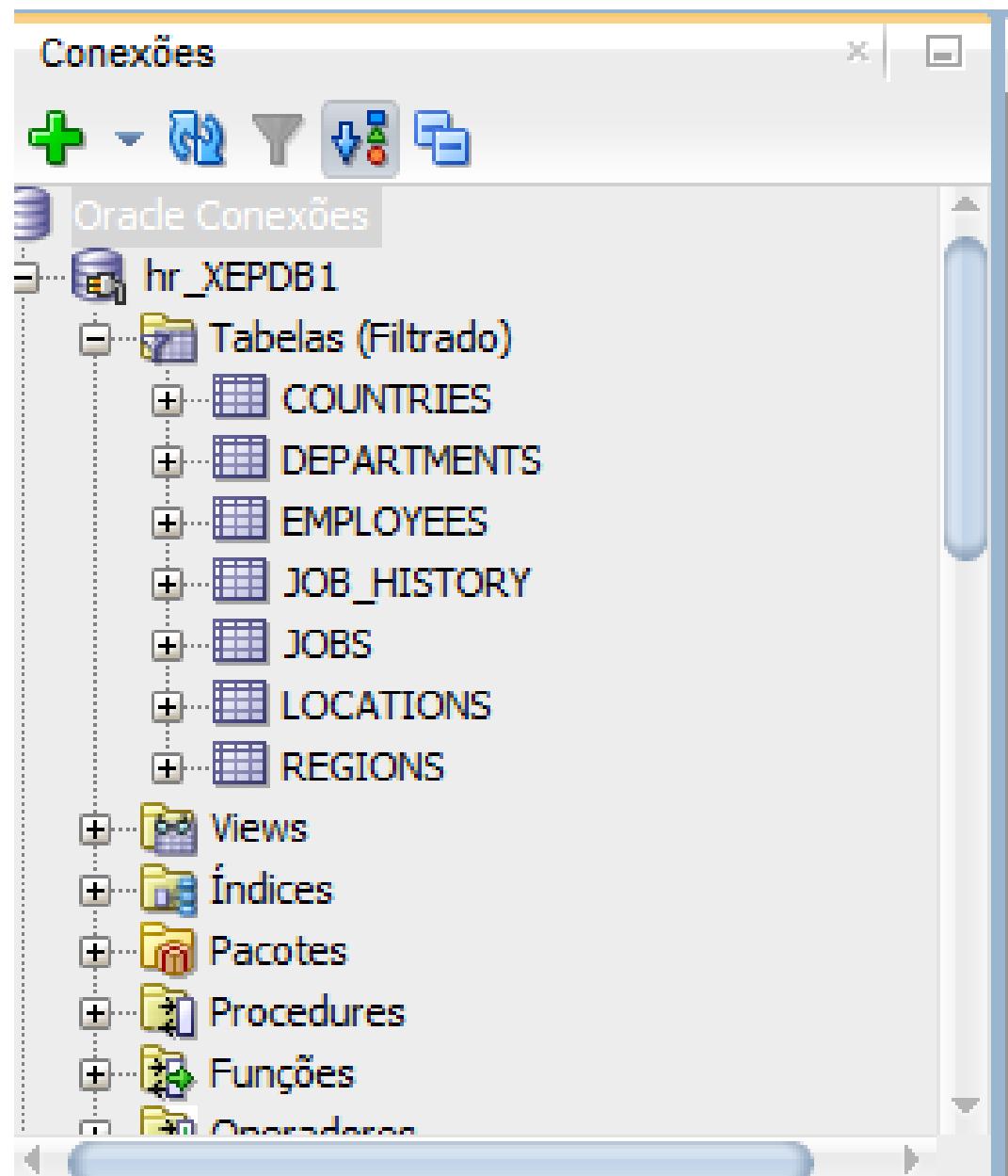
The screenshot shows the Oracle Database Documentation homepage. The left sidebar has a 'Development' section highlighted with a red circle labeled '1'. Below it is a 'Topics' section with links like 'Install and Upgrade', 'Administration', 'Development' (which is also highlighted with a red circle labeled '2'), 'Security', 'Performance', and 'Clustering'. The main content area features a large heading 'Oracle Database Documentation' and a 'SQL and PL/SQL' section. In the 'SQL and PL/SQL' section, there is a link to 'SQL Language Reference' which is highlighted with a red box and a red arrow pointing to it from the 'Development' link in the sidebar.

# Explorando as tabelas do banco HR

O banco de dados HR possui informações sobre os Recursos Humanos de uma empresa, e é composto por 7 tabelas:

1. COUNTRIES
2. DEPARTMENTS
3. EMPLOYEES
4. JOB\_HISTORY
5. JOBS
6. LOCATIONS
7. REGIONS

Se a gente clicar em cima do nome da tabela, será aberta uma nova janela onde conseguimos ver mais informações sobre essa tabela, além dos dados armazenados. Vamos fazer esse exercício.



# Tabela COUNTRIES

Ao clicar na tabela COUNTRIES, é aberta uma janela com a informação dos nomes das **Colunas**, além de uma outra janela chamada **Dados**, onde é possível visualizar os dados da tabela.

Esta tabela armazena informações de países e possui 3 colunas:

- COUNTRY\_ID
- COUNTRY\_NAME
- REGION\_ID

The screenshot shows two windows of Oracle SQL Developer displaying the COUNTRIES table. The top window, titled 'COUNTRIES', has its tab bar highlighted with red boxes around 'Colunas' and 'Dados'. The 'Colunas' tab is active, showing the table structure with columns: COLUMN\_NAME, DATA\_TYPE, NULLABLE, DATA\_DEFAULT, COLUMN\_ID, and COMMENTS. The 'Dados' tab is also visible. The bottom window, also titled 'COUNTRIES', has its tab bar highlighted with red boxes around 'Colunas' and 'Dados'. The 'Dados' tab is active, showing the table data with columns: COUNTRY\_ID, COUNTRY\_NAME, and REGION\_ID. Red arrows point from the text in the left column to the corresponding tabs in both windows.

COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1 COUNTRY_ID	CHAR (2 BYTE)	No	(null)	1	Primary key of countries table.
2 COUNTRY_NAME	VARCHAR2 (40 BYTE)	Yes	(null)	2	Country name
3 REGION_ID	NUMBER	Yes	(null)	3	Region ID for the country. Foreign

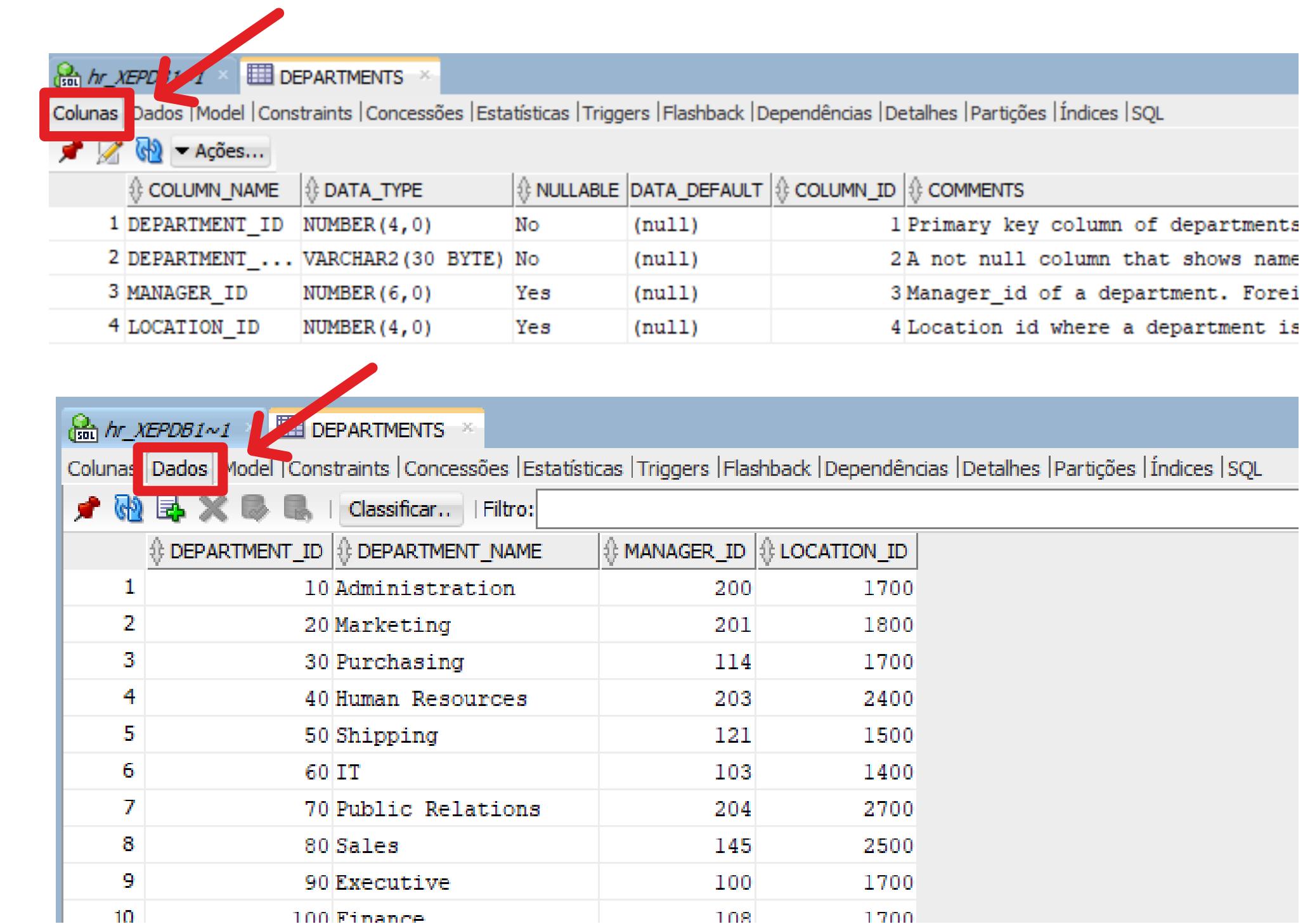
COUNTRY_ID	COUNTRY_NAME	REGION_ID
1 AR	Argentina	2
2 AU	Australia	3
3 BE	Belgium	1
4 BR	Brazil	2
5 CA	Canada	2
6 CH	Switzerland	1
7 CN	China	3
8 DE	Germany	1
9 DK	Denmark	1
10 EG	Egypt	4
11 FR	France	1
12 IL	Israel	4
13 ...	- ..	-

# Tabela DEPARTMENTS

A tabela DEPARTMENTS armazena as informações dos departamentos da empresa, e possui 4 colunas:

- DEPARTMENT\_ID
- DEPARTMENT\_NAME
- MANAGER\_ID
- LOCATION\_ID

Fica de exercício para você explorar as demais tabelas do banco de dados.



The screenshot shows two views of the DEPARTMENTS table in Oracle SQL Developer. The top window displays the table structure (Columns view) with four columns: DEPARTMENT\_ID, DEPARTMENT\_NAME, MANAGER\_ID, and LOCATION\_ID. The bottom window displays the actual data (Dados view) with ten rows of department information. Both windows have red arrows pointing to the 'Dados' tab in their respective tabs.

COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1 DEPARTMENT_ID	NUMBER (4, 0)	No	(null)		1 Primary key column of departments
2 DEPARTMENT_...	VARCHAR2 (30 BYTE)	No	(null)		2 A not null column that shows name
3 MANAGER_ID	NUMBER (6, 0)	Yes	(null)		3 Manager_id of a department. Forei
4 LOCATION_ID	NUMBER (4, 0)	Yes	(null)		4 Location id where a department is

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10 Administration	200	1700
2	20 Marketing	201	1800
3	30 Purchasing	114	1700
4	40 Human Resources	203	2400
5	50 Shipping	121	1500
6	60 IT	103	1400
7	70 Public Relations	204	2700
8	80 Sales	145	2500
9	90 Executive	100	1700
10	100 Finance	108	1700

# DESCRIBE/DESC - Exibindo a estrutura de tabelas

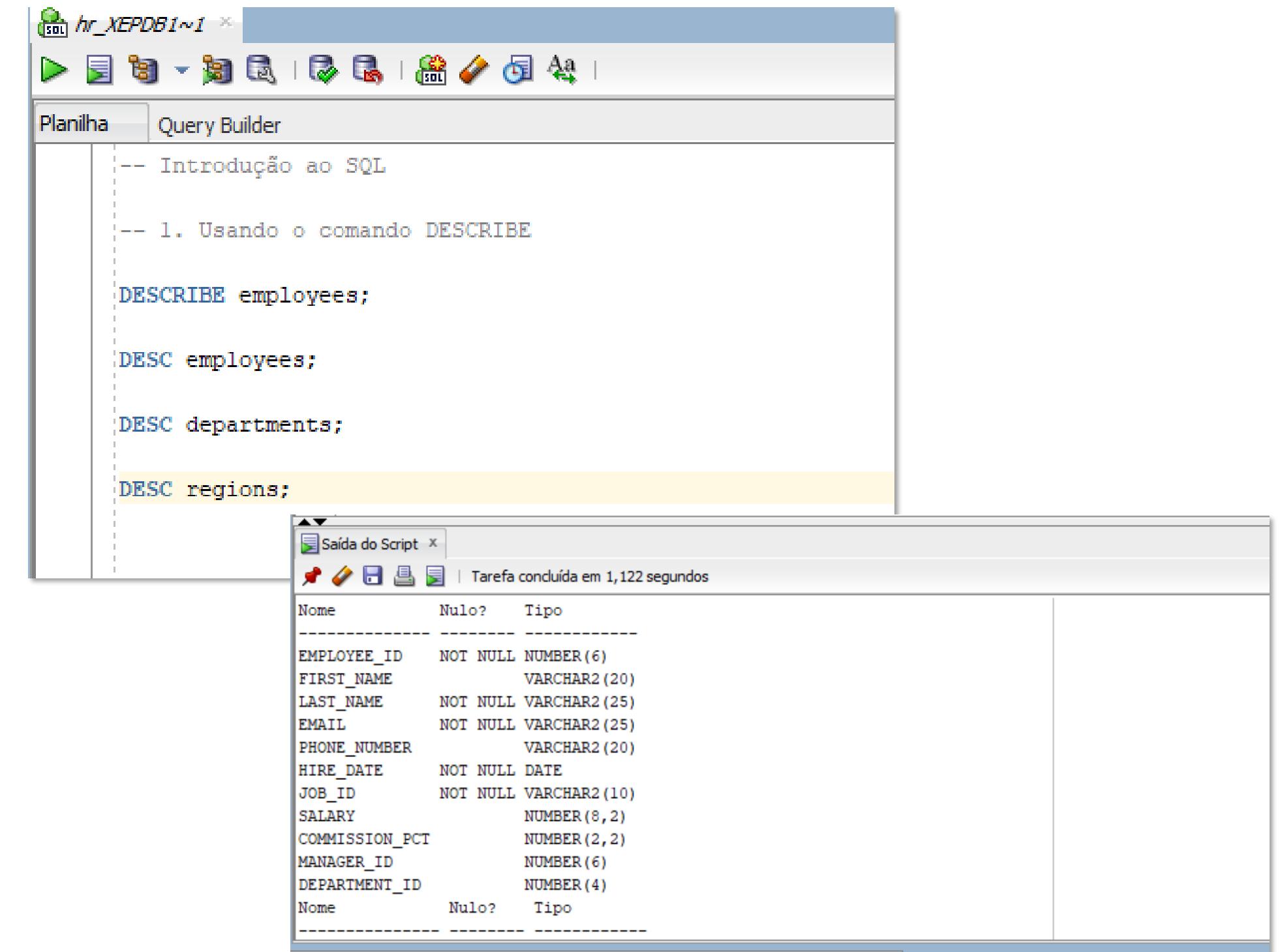
Vamos agora começar com os primeiros comandos em SQL dentro do Oracle.

Podemos usar o comando DESCRIBE (ou simplesmente DESC) para visualizar a estrutura das tabelas do nosso banco de dados:

- Nomes das colunas
- Constraints (Restrições)
- Tipos dos dados das colunas

Na imagem ao lado, temos a aplicação do comando.

Para executá-lo, basta selecionar a linha do código e clicar em  , no canto superior direito da janela de consulta.



The screenshot shows the Oracle SQL Developer interface. In the top-left window, there is a code editor with the following content:

```
-- Introdução ao SQL
-- 1. Usando o comando DESCRIBE
DESCRIBE employees;
DESC employees;
DESC departments;
DESC regions;
```

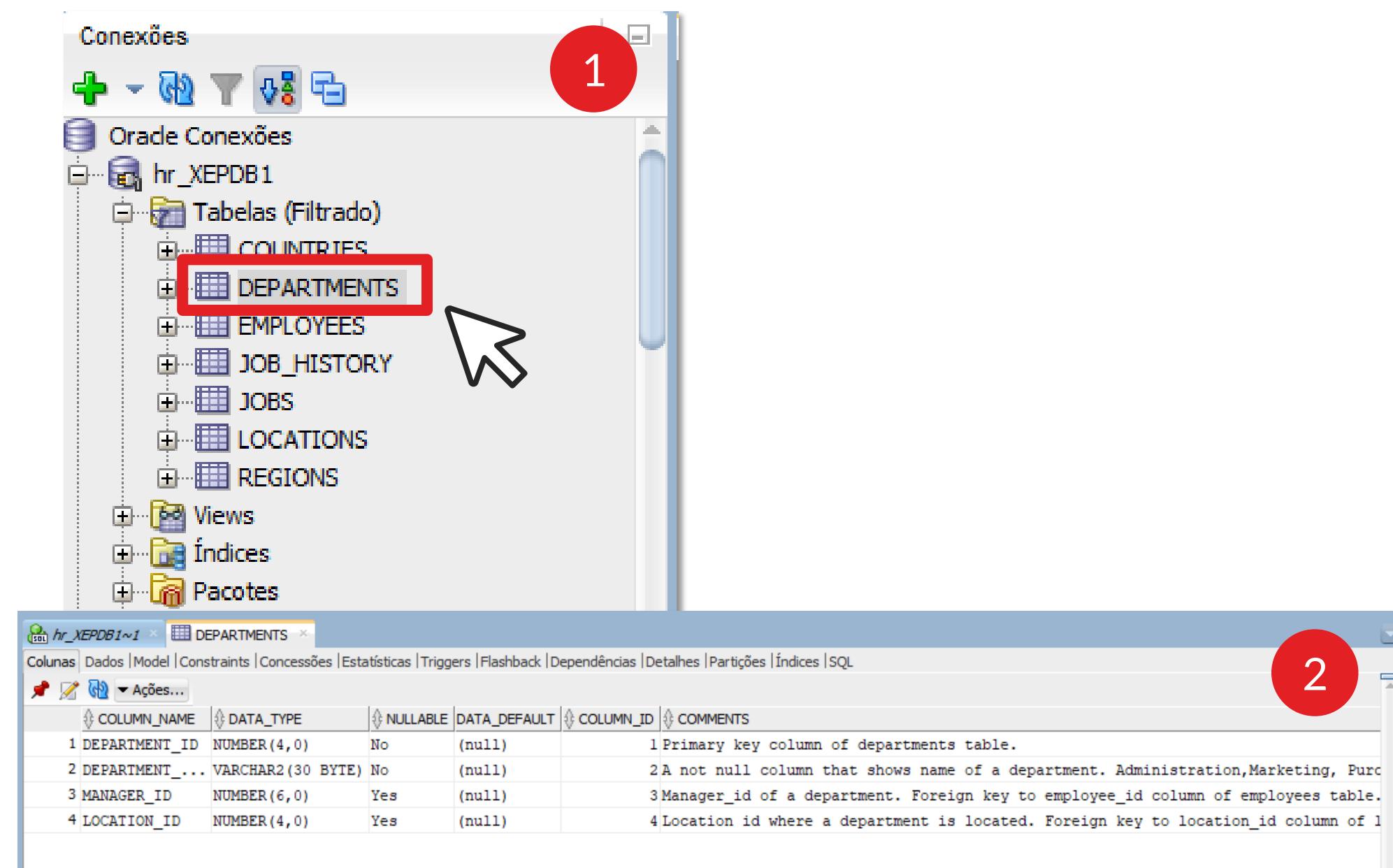
The line `DESC regions;` is highlighted with a yellow background. Below the code editor, there is a "Saída do Script" (Script Output) window showing the results of the DESCRIBE command for the employees table:

Nome	Nulo?	Tipo
EMPLOYEE_ID	NOT NULL	NUMBER(6)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
EMAIL	NOT NULL	VARCHAR2(25)
PHONE_NUMBER		VARCHAR2(20)
HIRE_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
SALARY		NUMBER(8,2)
COMMISSION_PCT		NUMBER(2,2)
MANAGER_ID		NUMBER(6)
DEPARTMENT_ID		NUMBER(4)
Nome	Nulo?	Tipo

# DESCRIBE/DESC - Exibindo a estrutura de tabelas

Uma outra forma de visualizar a estrutura das tabelas é clicando no nome da tabela que queremos.

Automaticamente será aberta uma guia com as informações sobre aquela tabela, como pode ser visto na figura 2.



# DESCRIBE/DESC - Exibindo a estrutura de tabelas

Para visualizar os dados daquela tabela, podemos clicar na opção **Dados**.

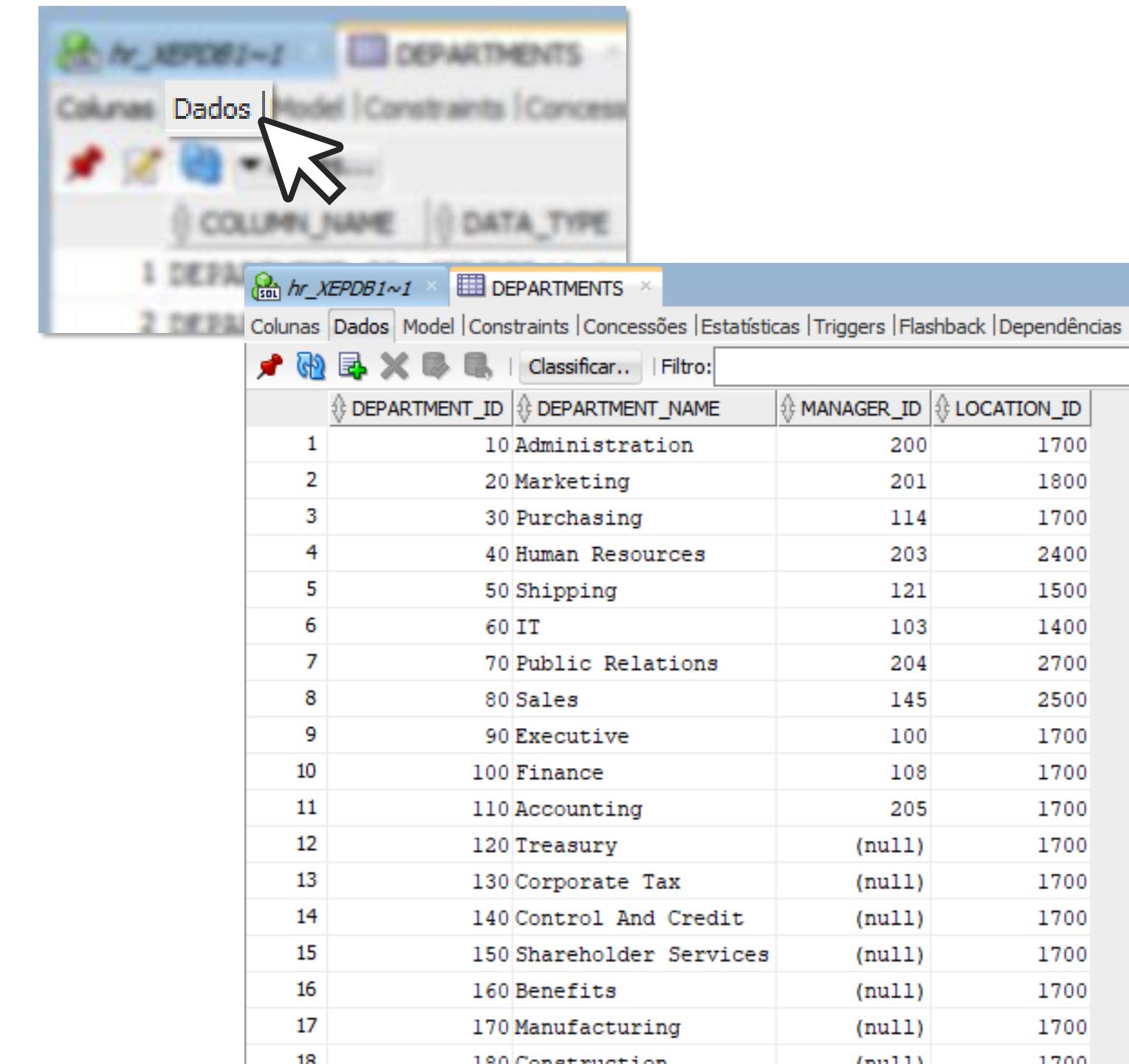
The screenshot shows two windows of Oracle SQL Developer. The top window is titled 'hr\_XEPDB1~1' and displays the 'DEPARTMENTS' table structure. The bottom window is also titled 'hr\_XEPDB1~1' and displays the 'DEPARTMENTS' table data. A red circle labeled '1' is over the 'Dados' tab in the top window's toolbar, and a red circle labeled '2' is over the 'Dados' tab in the bottom window's toolbar. Both windows show the same data:

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	Administration	200	1700
2	Marketing	201	1800
3	Purchasing	114	1700
4	Human Resources	203	2400
5	Shipping	121	1500
6	IT	103	1400
7	Public Relations	204	2700
8	Sales	145	2500
9	Executive	100	1700
10	Finance	108	1700
11	Accounting	205	1700
12	Treasury	(null)	1700
13	Corporate Tax	(null)	1700
14	Control And Credit	(null)	1700
15	Shareholder Services	(null)	1700
16	Benefits	(null)	1700
17	Manufacturing	(null)	1700
18	Construction	(null)	1700

# SELECT – Selecionando os dados da tabela

Vimos anteriormente que é possível visualizar os dados de uma tabela simplesmente clicando no nome dela e automaticamente e na nova janela que abrir, ao clicar em DADOS, são mostrados todas as linhas e colunas daquela tabela.

Podemos também usar um comando em SQL para fazer essa seleção. O comando é o **SELECT**. O comando SELECT pode ser usado de duas maneiras. A seguir, vamos ver cada uma delas.



The screenshot shows two windows of Oracle Database SQL Developer. The top window is titled 'hr\_XEPDB1~1' and displays the 'DEPARTMENTS' table structure with tabs for 'Colunas', 'Dados' (which is selected), 'Model', 'Constraints', 'Concessões', 'Estatísticas', 'Triggers', 'Flashback', and 'Dependências'. The bottom window also has the same title and shows the 'DEPARTMENTS' table data. The 'Dados' tab is selected here as well. The table contains 18 rows of department information:

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	Administration	200	1700
2	Marketing	201	1800
3	Purchasing	114	1700
4	Human Resources	203	2400
5	Shipping	121	1500
6	IT	103	1400
7	Public Relations	204	2700
8	Sales	145	2500
9	Executive	100	1700
10	Finance	108	1700
11	Accounting	205	1700
12	Treasury	(null)	1700
13	Corporate Tax	(null)	1700
14	Control And Credit	(null)	1700
15	Shareholder Services	(null)	1700
16	Benefits	(null)	1700
17	Manufacturing	(null)	1700
18	Construction	(null)	1700

# SELECT – Selecionando os dados da tabela

1

Selecionando todas as colunas  
da tabela

```
SELECT *
FROM tabela;
```

A primeira maneira é selecionar todas as colunas de uma determinada tabela.

Para isso, usamos o comando **SELECT**, seguido do caractere \* (asterisco). Este asterisco significa que queremos exibir todas as colunas.

A seguir, após o comando **FROM**, informamos o nome da tabela de onde queremos visualizar aquelas colunas.

# SELECT – Selecionando os dados da tabela

# 2

## Selecionando colunas específicas da tabela

```
SELECT  
    coluna1,  
    coluna2,  
    coluna3  
FROM tabela;
```

A segunda maneira é selecionar apenas algumas colunas da tabela, dado que nem sempre queremos visualizar todas.

Neste caso, usamos o comando **SELECT**, seguido de cada uma das colunas da tabela que queremos visualizar.

A seguir, após o comando **FROM**, informamos o nome da tabela de onde queremos visualizar aquelas colunas.

# SELECT – Selecionando os dados da tabela

Em resumo, temos o seguinte:

**SELECT** identifica as colunas ou expressões a serem exibidas. O \* (asterisco) significa que queremos visualizar todas as colunas da tabela.

**FROM** identifica as tabelas que contêm as colunas selecionadas no SELECT.

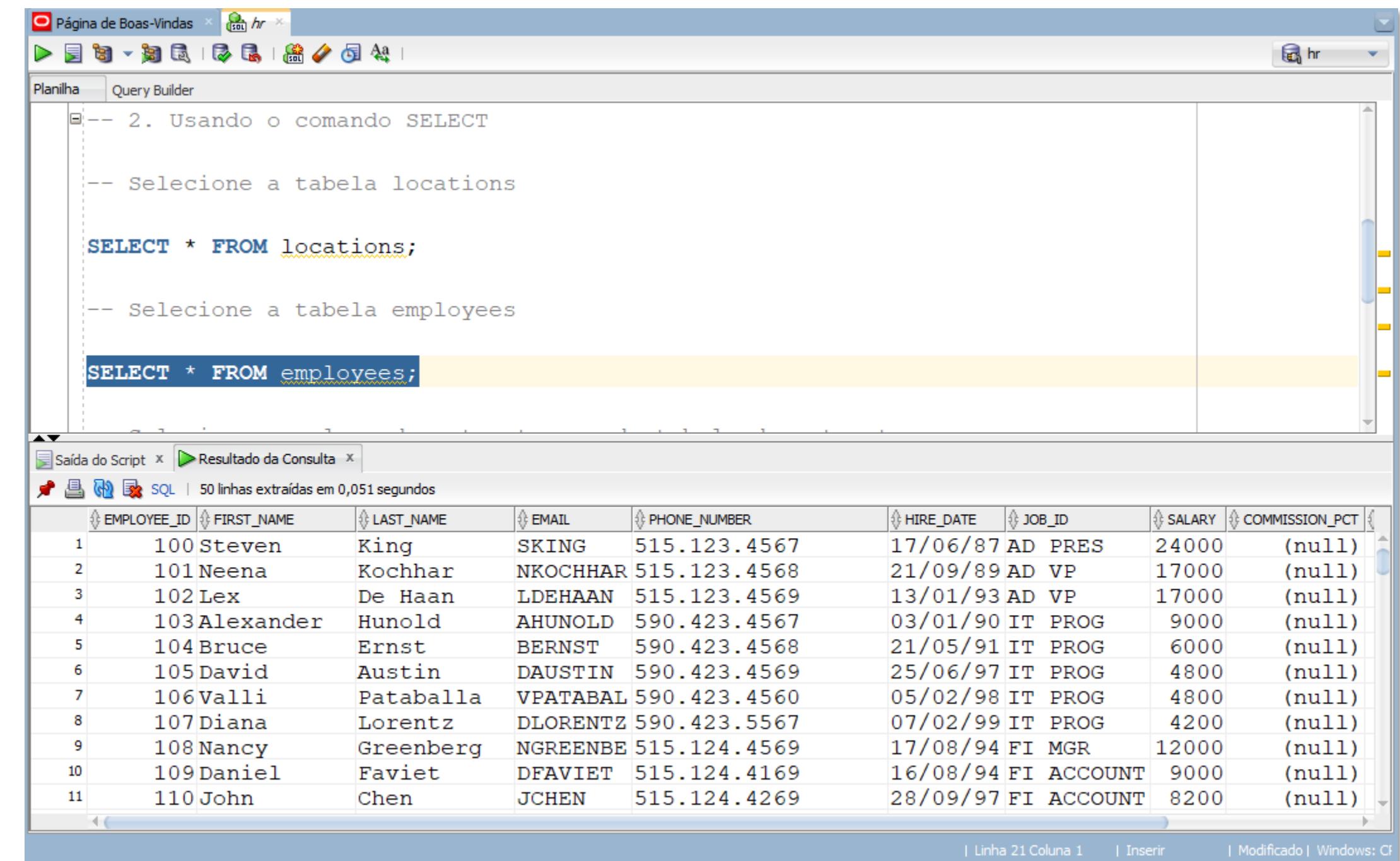
# SELECT – Selecionando os dados da tabela

Ao lado, temos uma aplicação do comando SELECT para selecionar a tabela de EMPLOYEES.

Para executar a linha, posicione o cursor no final dela (após o ponto e vírgula) e depois clique em .

Logo abaixo, será retornada a tabela resultante.

Observe que todas as colunas da tabela foram retornadas, pois usamos o comando SELECT \*. Lembrando que o \* significa que queremos retornar **todas as colunas**.



The screenshot shows a SQL development environment with two tabs: 'Planilha' (Worksheet) and 'Resultado da Consulta' (Query Result). In the 'Planilha' tab, there is a script with the following content:

```

-- 2. Usando o comando SELECT

-- Selecione a tabela locations

SELECT * FROM locations;

-- Selecione a tabela employees

SELECT * FROM employees;

```

The last line of the script, `SELECT \* FROM employees;`, is highlighted with a yellow background. In the 'Resultado da Consulta' tab, the results of the query are displayed in a grid:

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT
1	Steven	King	SKING	515.123.4567	17/06/87	AD PRES	24000	(null)
2	Neena	Kochhar	NKOCHHAR	515.123.4568	21/09/89	AD VP	17000	(null)
3	Lex	De Haan	LDEHAAN	515.123.4569	13/01/93	AD VP	17000	(null)
4	Alexander	Hunold	AHUNOLD	590.423.4567	03/01/90	IT PROG	9000	(null)
5	Bruce	Ernst	BERNST	590.423.4568	21/05/91	IT PROG	6000	(null)
6	David	Austin	DAUSTIN	590.423.4569	25/06/97	IT PROG	4800	(null)
7	Valli	Pataballa	VPATABAL	590.423.4560	05/02/98	IT PROG	4800	(null)
8	Diana	Lorentz	DLORENTZ	590.423.5567	07/02/99	IT PROG	4200	(null)
9	Nancy	Greenberg	NGREENBE	515.124.4569	17/08/94	FI MGR	12000	(null)
10	Daniel	Faviet	DFAVIET	515.124.4169	16/08/94	FI ACCOUNT	9000	(null)
11	John	Chen	JCHEN	515.124.4269	28/09/97	FI ACCOUNT	8200	(null)

Below the grid, status information is shown: 'Saída do Script x Resultado da Consulta x', 'SQL | 50 linhas extraídas em 0,051 segundos', and 'Linha 21 Coluna 1 | Inserir | Modificado | Windows: Cf'.

# SELECT – Selecionando os dados da tabela

Pode ser que a gente não queira visualizar todas as colunas da tabela. Desta forma, especificamos quais colunas desejamos visualizar, em vez de simplesmente usar o asterisco.

The screenshot shows the Oracle SQL Developer interface. In the top window (Planilha), a SQL query is written:

```
-- Selecione apenas as colunas first_name, email e salary da tabela employees
SELECT first_name, email, salary
FROM employees;
```

In the bottom window (Resultado da Consulta), the results of the query are displayed in a grid:

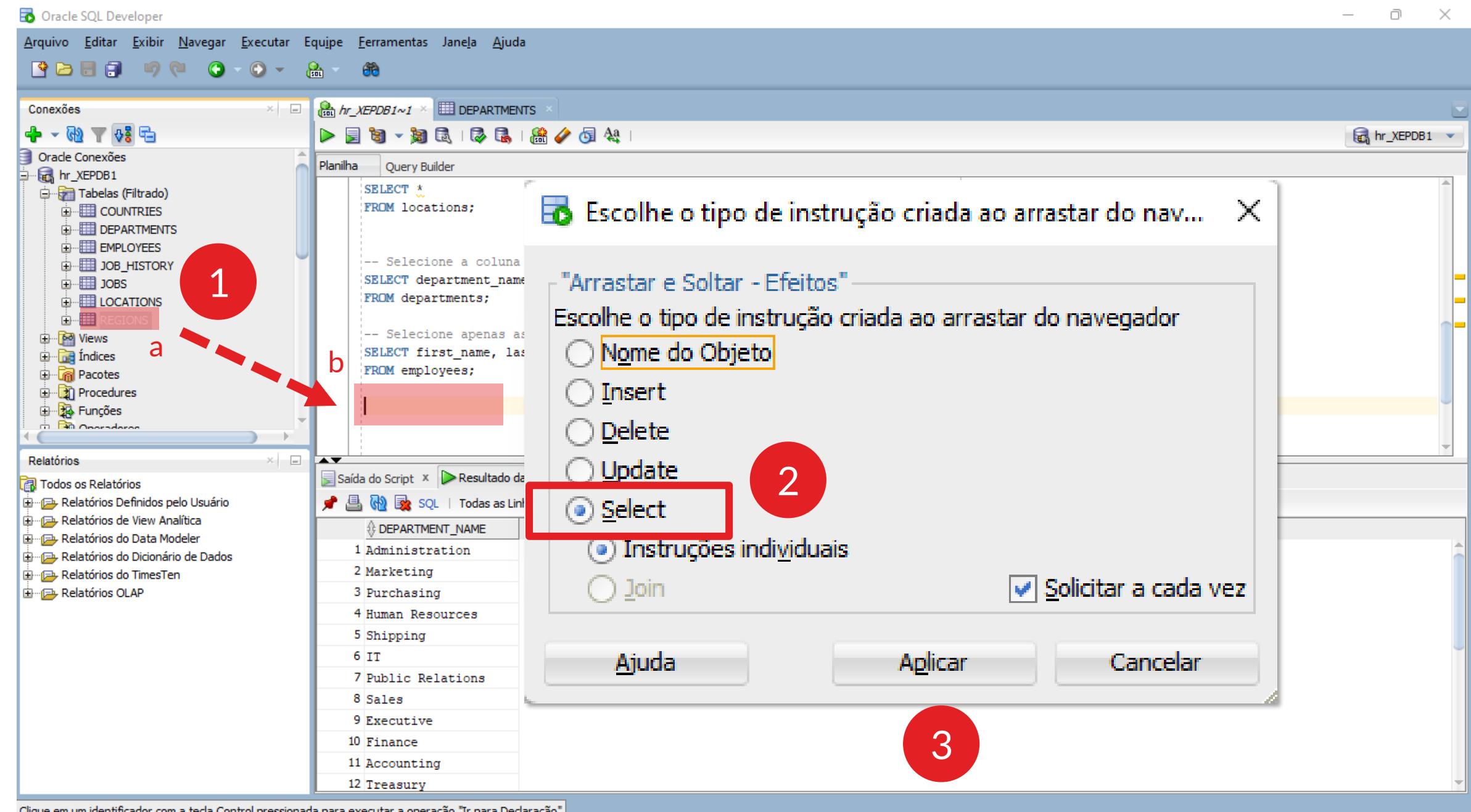
	FIRST_NAME	EMAIL	SALARY
1	Steven	SKING	24000
2	Neena	NKOCHHAR	17000
3	Lex	LDEHAAN	17000
4	Alexander	AHUNOLD	9000
5	Bruce	BERNST	6000
6	David	DAUSTIN	4800
7	Valli	VPATABAL	4800
8	Diana	DLORENTZ	4200
9	Nancy	NGREENBE	12000
10	Daniel	DFAVIET	9000
11	John	JCHEN	8200
12	Ismael	ISCIARRA	7700

# SELECT – Selecionando os dados da tabela

000

Além de selecionar tabelas clicando no nome delas ou então usando um código SQL para isso, temos também uma outra possibilidade no Oracle.

Se você clicar com o mouse na tabela, depois arrastar até a janela de consulta e soltar (1), vai aparecer uma janela onde você pode informar que deseja fazer um SELECT (2). Depois é só clicar em Aplicar (3) que a janela estará feita.



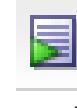
A seta tracejada significa que estamos clicando no ponto a, arrastando e soltando no ponto b.

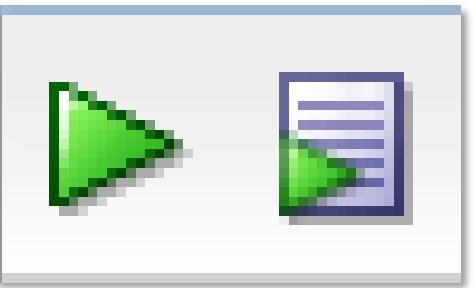
ORACLE®

# Formas de executar um código

Temos duas formas de executar códigos no Oracle. Os botões de execução são identificados pelos ícones ao lado.

O ícone da **esquerda**  executa apenas a linha onde o cursor está selecionado. Podemos usar o atalho CTRL + ENTER.

Já o da **direita**  executa toda a janela de código. Ou seja, todos os comandos que estiverem ali serão executados. Diferente do anterior, que executará apenas a linha onde o cursor do mouse estiver posicionado. Para esta opção de execução, temos o bom e velho atalho F5.



# Comentários

Quando começamos a programar, uma das coisas mais comuns é a gente se deparar com **comentários** dentro dos códigos.

Comentários são muito usados por alguns motivos:

- 1) Documentar e explicar o que está sendo feito no código.
- 2) Documentar quem foi o desenvolvedor do script e quando foi desenvolvido, para o caso de consultas futuras.

Comentários geralmente trazem alguma clareza e pode ser uma boa prática em diversas situações. Apenas tome cuidado para não fazer redações do ENEM nos seus códigos. Utilize os comentários de forma concisa e resumida.

No Oracle, podemos criar comentários de duas formas:

- 1) Comentários de uma linha, com o `--`.
- 2) Comentários em blocos, começando com `/*` e terminando com `*/`, o que permite comentar mais de 1 linha de uma vez.

```
-- Comentando o código
-- É um comentário de 1 linha.
SELECT * FROM employees;

/*
SELECT *
FROM employees;

permite
comentar
múltiplas
linhas
*/

```

# Alias de Colunas

Por padrão, quando criamos uma consulta a uma tabela do banco de dados, o cabeçalho das colunas resultantes têm o mesmo nome das colunas do banco de dados. Observe em 1 que a tabela foi retornada com os nomes originais (em inglês) das colunas nos bancos de dados.

Já em 2, utilizando o alias (comando AS) conseguimos renomear estes cabeçalhos. Importante dizer que o AS não altera os nomes dos cabeçalhos no banco de dados. No banco de dados eles permanecem iguais.

1

```
-- 3. Criando Alias (apelidos para as colunas)

SELECT
    first_name,
    email,
    salary
FROM employees;
```

FIRST_NAME	EMAIL	SALARY
1 Steven	SKING	24000
2 Neena	NKOCHHAR	17000
3 Lex	LDEHAAN	17000
4 Alexander	AHUNOLD	9000
5 Bruce	BERNST	6000

2

```
-- 3. Criando Alias (apelidos para as colunas)

SELECT
    first_name AS nome,
    email AS email,
    salary AS salario
FROM employees;
```

NOME	EMAIL	SALARIO
1 Steven	SKING	24000
2 Neena	NKOCHHAR	17000
3 Lex	LDEHAAN	17000
4 Alexander	AHUNOLD	9000
5 Bruce	BERNST	6000
6 David	DAUSTIN	4800
7 Valli	VPATABAL	4800

# Alias de Colunas

Também poderíamos utilizar o alias sem necessariamente colocar o AS, da forma como está mostrado ao lado.

Um detalhe importante sobre a nomeação das colunas. Podemos também utilizar nomes compostos, mas uma boa prática é evitar espaços nos nomes, acentos e caracteres especiais. Busque usar um nome com o texto escrito todo junto, pra evitar qualquer problema.

Então por exemplo, se você quisesse traduzir literalmente o `first_name`, poderia escrever `primeiro_nome`. Ou seja, com um underline (\_) separando a primeira da segunda palavra.

```
SELECT
    first_name nome,
    email email,
    salary salario
FROM employees;
```

The screenshot shows the Oracle SQL Developer interface. At the top, there's a code editor with the query:

```
SELECT
    first_name nome,
    email email,
    salary salario
FROM employees;
```

Below the code editor is a toolbar with icons for script output, result set, and SQL. The status bar indicates "50 linhas extraídas em 0,006 segundos". The main area displays the results in a table:

	NOME	EMAIL	SALARIO
1	Steven	SKING	24000
2	Neena	NKOCHHAR	17000
3	Lex	LDEHAAN	17000
4	Alexander	AHUNOLD	9000
5	Bruce	BERNST	6000
6	David	DAUSTIN	4800
7	Valli	VPATABAL	4800
8	Diana	DLORENTZ	4200
9	Nancy	NGREENBE	12000
10	Daniel	DFAVIET	9000
11	John	JCHEN	8200
12	Ismael	ISCIARRA	7700
13	Tiger	TMITOMASI	7800

# Alias de Colunas

Se ainda assim você quiser usar espaços, caracteres especiais e acentuação, você pode colocar os nomes entre aspas duplas, assim como mostrado ao lado.

```
-- Selecione as colunas first_name, email e salary da tabela employees.  
-- Dê um nome para as colunas da tabela.  
  
SELECT  
    first_name "primeiro nome",  
    email "e-mail",  
    salary "salário (R$)"  
FROM employees;
```

	primeiro nome	e-mail	salário (R\$)
1	Steven	SKING	24000
2	Neena	NKOCHHAR	17000
3	Lex	LDEHAAN	17000
4	Alexander	AHUNOLD	9000
5	Bruce	BERNST	6000
6	David	DAUSTIN	4800
7	Valli	VPATABAL	4800
8	Diana	DLORENTZ	4200
9	Nancy	NGREENBE	12000

# Alias de Tabelas

Uma aplicação do Alias bem importante será a de renomear tabelas dentro dos nossos códigos, para torná-los mais reduzidos.

No exemplo ao lado, fizemos uma relação entre as tabelas **employees** e **departments**. Observe que logo depois do nome **employees** utilizamos um **e**, e logo depois do **departments** usamos um **d**. Em seguida, em todos os lugares onde precisamos referenciar essas tabelas, utilizamos os seus apelidos, em vez de seus nomes completos.

Voltaremos nesse tipo de alias mais a frente, quando entrarmos na parte de relacionamentos entre as tabelas por meio dos JOINs. Por isso, não precise se preocupar em entender isso agora.

```
-- Relaciona as tabelas employees e departments por meio de um LEFT JOIN.
-- Além disso, as tabelas são renomeadas com o aliasing.
SELECT
    e.employee_id,
    e.first_name,
    e.email,
    e.department_id,
    d.department_name
FROM employees e
LEFT JOIN departments d
ON e.department_id = d.department_id;
```

Saída do Script x Resultado da Consulta x

SQL | 50 linhas extraídas em 0,009 segundos

	EMPLOYEE_ID	FIRST_NAME	EMAIL	DEPARTMENT_ID	DEPARTMENT_NAME
1	100	Steven	SKING	90	Executive
2	101	Neena	NKOCHHAR	90	Executive
3	102	Lex	LDEHAAN	90	Executive
4	103	Alexander	AHUNOLD	60	IT
5	104	Bruce	BERNST	60	IT
6	105	David	DAUSTIN	60	IT
7	106	Valli	VPATABAL	60	IT
8	107	Diana	DLORENTZ	60	IT
9	108	Nancy	NGREENBE	100	Finance

# Operador de concatenação ||

Existe um operador especial no Oracle que permite concatenar textos, que seria a barra vertical dupla ||.

Com ela, podemos concatenar diversos textos, como mostrado ao lado.

Duas observações:

1- Textos no código são escritos com aspas simples (exceto no alias, que usamos aspas duplas)

2- Para concatenar os nomes, colocamos um espaço entre first\_name e last\_name para que os textos não ficassem colados.

```
-- 4. Operador de concatenação  
-- Faça uma consulta que retorne o nome completo dos funcionários  
  
SELECT  
    first_name || ' ' || last_name "Nome Completo"  
FROM employees;
```

	Nome Completo
1	Ellen Abel
2	Sundar Ande
3	Mozhe Atkinson
4	David Austin
5	Hermann Baer
6	Shelli Baida
7	Amit Banda
8	Elizabeth Bates
9	Sarah Bell

# Resolvendo problema do ‘dentro de uma string’

Em algumas situações, precisamos ter uma aspa simples dentro do texto. Porém, a aspa simples serve para delimitar os textos dentro do SQL Oracle.

O que fazer?

Neste caso, vamos colocar o texto que possui uma aspa simples dentro de colchetes, iniciados por um q (q de quote, do inglês ‘aspas’).

O resultado é mostrado ao lado.

```
-- Faça uma consulta que retorne a lista de funcionários e seus respectivos gerentes

SELECT
    first_name || q'[. Employee's manager id: ]' || manager_id "Funcionário e Gerente"
FROM employees;
```

	Funcionário e Gerente
1	Steven. Employee's manager id:
2	Neena. Employee's manager id: 100
3	Lex. Employee's manager id: 100
4	Alexander. Employee's manager id: 102
5	Bruce. Employee's manager id: 103
6	David. Employee's manager id: 103
7	Valli. Employee's manager id: 103
8	Diana. Employee's manager id: 103
9	Nancy. Employee's manager id: 101

# Cálculos simples no SQL

É possível realizar cálculos básicos de soma, subtração, multiplicação e divisão no SQL.

Utilizamos os seguintes operadores:

- + para realizar a soma
- para realizar subtração
- \* para realizar multiplicação
- / para realizar divisão

Ao lado, temos um exemplo bem simples. Aplicamos um bônus de R\$100 ao salário de todos os funcionários.

```
-- 5. Cálculos simples no SQL
-- Adicione um bônus de R$100 aos salários de todos os funcionários

SELECT
    first_name,
    last_name,
    salary,
    salary + 100 "salário + bônus"
FROM employees;
```

The screenshot shows a SQL query being run against the 'employees' table. The query adds a bonus of 100 to each employee's salary. The results are displayed in a grid with columns: FIRST\_NAME, LAST\_NAME, SALARY, and salário + bônus. The data includes 11 rows of employees with their names, current salaries, and the new salaries after the bonus is applied.

	FIRST_NAME	LAST_NAME	SALARY	salário + bônus
1	Steven	King	24000	24100
2	Neena	Kochhar	17000	17100
3	Lex	De Haan	17000	17100
4	Alexander	Hunold	9000	9100
5	Bruce	Ernst	6000	6100
6	David	Austin	4800	4900
7	Valli	Pataballa	4800	4900
8	Diana	Lorentz	4200	4300
9	Nancy	Greenberg	12000	12100
10	Daniel	Faviet	9000	9100
11	Tiger	Chen	6200	6200

# Cálculos simples no SQL

Podemos incluir parênteses para fazer cálculos mais avançados, como no exemplo ao lado.

Aplicamos um bônus de 25% sobre o salário de todos os funcionários.

```
-- Adicione um bônus de 25% do salário para todos os funcionários

SELECT
    first_name,
    last_name,
    salary,
    salary * (1 + 0.25) "salário + bônus (25%)"
FROM employees;
```

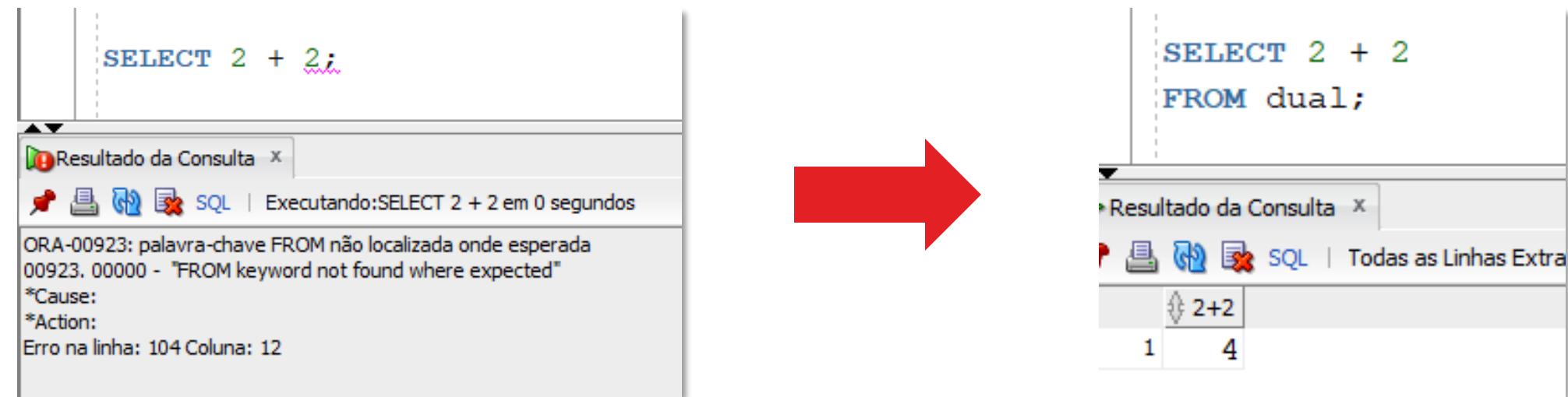
	FIRST_NAME	LAST_NAME	SALARY	salário + bônus (25%)
1	Steven	King	24000	30000
2	Neena	Kochhar	17000	21250
3	Lex	De Haan	17000	21250
4	Alexander	Hunold	9000	11250
5	Bruce	Ernst	6000	7500
6	David	Austin	4800	6000
7	Valli	Pataballa	4800	6000
8	Diana	Lorentz	4200	5250
9	Nancy	Greenberg	12000	15000
10	Daniel	Faviet	9000	11250
11	John	Chen	8200	10250
12	Ismael	Sciarra	7700	9625
13	Jose Manuel	Urman	7800	9750

# Tabela Dual

A tabela Dual do Oracle é usada para fazer operações com o SELECT aonde não é necessário fazer uma seleção de dados de uma tabela.

Ela ajuda a manter a sintaxe de um SELECT aonde não teremos uma tabela na consulta, mantendo a cláusula FROM.

Por exemplo, imagina que a gente queira fazer um cálculo simples de  $2 + 2$ . Em outro banco de dados, usariamos um SELECT  $2 + 2$  e já seria suficiente. Porém, no Oracle não é possível omitir a informação do FROM com a tabela. Ele retorna um erro. Para contornar isso, usamos a tabela dual.



Usaremos o FROM dual sempre que precisarmos fazer qualquer operação onde não seja necessário selecionar uma tabela de um banco de dados.

# Selecionando valores distintos

Quando selecionamos uma coluna de uma tabela, o SQL retorna todas as linhas dessa tabela.

Observe na imagem ao lado. Criamos um código para trazer a coluna department\_id da tabela EMPLOYEES.

Porém, como todas as linhas foram retornadas, tivemos vários ids se repetindo. Isso porque podemos ter vários funcionários de um mesmo departamento.

Como fazer então para retornar apenas os ids **distintos**?

```
-- Qual é o total de departamentos? Faça esta análise em cima da tabela  
  
SELECT  
    department_id  
FROM employees;
```

Saída do Script x Resultado da Consulta x  
SQL | 50 linhas extraídas em 0,01 segundos

DEPARTMENT_ID	
1	90
2	90
3	90
4	60
5	60
6	60
7	60
8	60
9	100
10	100
11	100
12	100
13	100
14	100

# Selecionando valores distintos

É ai que entra o SELECT DISTINCT. Este comando nos permite retornar apenas os valores distintos de uma coluna, e sua aplicação é mostrada na imagem ao lado.

Observe que no resultado temos apenas os department\_ids distintos, nenhum está repetido.

```
-- Qual é o total de departamentos? Faça esta análise em cima da tabela  
  
SELECT  
    DISTINCT department_id  
FROM employees;
```



DEPARTMENT_ID	90
1	90
2	60
3	100
4	30
5	50
6	80
7	(null)
8	10
9	20
10	40
11	70
12	110

# Selecionando valores distintos (mais de uma coluna)

Podemos querer saber também dos valores distintos considerando mais de uma coluna por vez.

No exemplo ao lado, queremos os distintos considerando tanto o `first_name` quanto o `last_name`. Dessa maneira, será feita uma espécie de concatenação dos valores para então retornar com os valores distintos.

Por exemplo, neste caso, os nomes Ellen Abel e Ellen Bloom seriam nomes distintos e portanto estariam presentes em linhas diferentes da tabela.

```
-- Retorne os valores distintos de nomes completos dos funcionários

SELECT
    DISTINCT first_name, last_name
FROM employees;
```

Saída do Script | Resultado da Consulta | SQL | 50 linhas extraídas em 0,011 segundos

	FIRST_NAME	LAST_NAME
1	Ellen	Abel
2	Sundar	Ande
3	Mozhe	Atkinson
4	David	Austin
5	Hermann	Baer
6	Shelli	Baida
7	Amit	Banda
8	Elizabeth	Bates
9	Sarah	Bell
10	David	Bernstein
11	Laura	Bissot
12	Harrison	Bloom
13	Alexis	Bull

# Resumo do Módulo e Boas Práticas

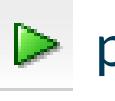
Algumas dicas, boas práticas e cuidados ao realizar as consultas:

- 1 Os comandos em SQL não diferenciam maiúsculas de minúsculas. Usar SELECT ou select funcionará do mesmo jeito. Mas, uma prática comum é colocar os nomes de comandos em MAIÚSCULA, já os nomes de colunas, tabelas, escrever em minúscula.
- 2 Podemos escrever todos o código em 1 linha só, ou em várias linhas.
- 3 Geralmente, colocamos os comandos (SELECT, FROM, WHERE, GROUP BY, etc) em linhas separadas.
- 4 Indentar um código não é obrigatório, mas facilita no entendimento.
- 5 Terminamos os comandos em SQL usando o ponto e vírgula.

# Resumo do Módulo e Boas Práticas

Algumas dicas, boas práticas e cuidados ao realizar as consultas:

6

Podemos executar os códigos de duas formas: com o  para rodar apenas a linha onde o cursor está posicionado ou com o  para executar o código por completo.

7

Comentários nos códigos são uma boa prática. Podemos comentar uma única linha com o -- ou comentar várias linhas de uma vez com o /\* \*/. Apenas tome cuidado para não abusar muito dos comentários.

8

É possível personalizar os nomes das colunas e tabelas na hora de criar consultas por meio do aliasing. Lembrando que o AS é opcional para as colunas e não é necessário para as tabelas.

9

Utilize o operador || para concatenar colunas do banco de dados.

10

O comando DISTINCT retorna os valores distintos de uma coluna. Se utilizarmos mais de uma coluna na hora de usar o DISTINCT, ele irá retornar os valores distintos considerando uma concatenação das colunas que você tiver informado.

# Boas práticas e cuidados ao criar consultas

Algumas dicas, boas práticas e cuidados ao realizar as consultas:

1

Os comandos em SQL não diferenciam maiúsculas de minúsculas. Usar SELECT ou select funcionará do mesmo jeito. Mas, uma prática comum é colocar os nomes de comandos em MAIÚSCULA, já os nomes de colunas, tabelas, escrever em minúscula.

```
-- Selecione a tabela de funcionários (EMPLOYEES)
SELECT *
FROM employees;
```

```
-- Selecione a tabela de funcionários (EMPLOYEES)
select *
from employees;
```

**Não faz diferença usar os comandos em maiúsculas ou minúsculas.**

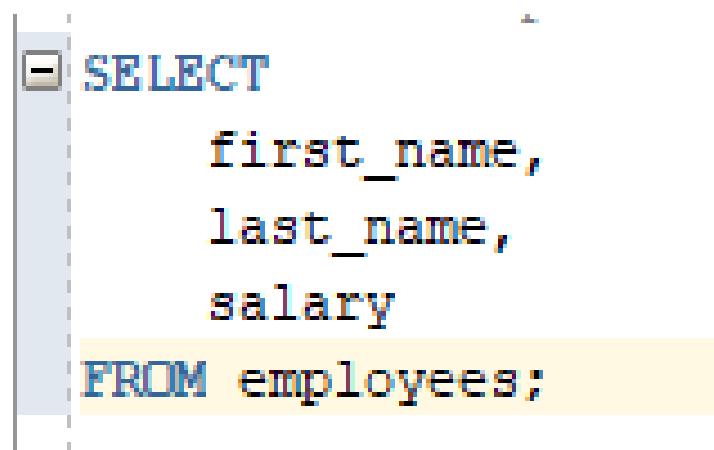
# Boas práticas e cuidados ao criar consultas

Algumas dicas, boas práticas e cuidados ao realizar as consultas:

2

Podemos escrever todos o código em 1 linha só, ou em várias linhas.

```
-- Selecione apenas as colunas first_name, last_name e salary da tabela EMPLOYEES  
SELECT first_name, last_name, salary FROM employees;
```



```
SELECT  
    first_name,  
    last_name,  
    salary  
FROM employees;
```

Os dois códigos acima fazem a mesma coisa. Porém, o código da imagem superior é bem mais intuitivo e organizado do que o código da imagem inferior

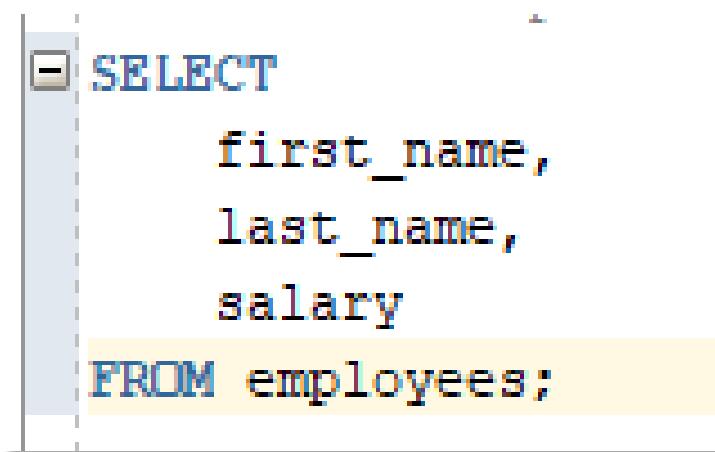
# Boas práticas e cuidados ao criar consultas

Algumas dicas, boas práticas e cuidados ao realizar as consultas:

3

Geralmente, colocamos os comandos (SELECT, FROM, WHERE, GROUP BY, etc) em linhas separadas.

```
-- Selecione apenas as colunas first_name, last_name e salary da tabela EMPLOYEES  
SELECT first_name, last_name, salary FROM employees;
```



A screenshot of a code editor showing a collapsed SQL query. The code is identical to the one above, but the entire statement is preceded by a minus sign (-) and enclosed in brackets, indicating it is collapsed or minimized.

```
[ -] SELECT  
    first_name,  
    last_name,  
    salary  
FROM employees;
```

Os dois códigos acima fazem a mesma coisa. Porém, o código da imagem superior é bem mais intuitivo e organizado do que o código da imagem inferior.

# Boas práticas e cuidados ao criar consultas

Algumas dicas, boas práticas e cuidados ao realizar as consultas:

4

Indentar um código não é obrigatório, mas facilita no entendimento.

```
SELECT  
    first_name,  
    last_name,  
    salary  
FROM employees;
```

```
SELECT  
    first_name,  
    last_name,  
    salary  
FROM employees;
```

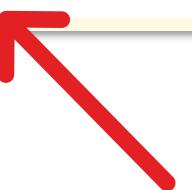
Os dois códigos acima fazem a mesma coisa. Porém, no código da imagem à direita fica muito mais fácil de entender que as colunas fazem parte do comando SELECT. Esses espaçamentos são chamados de indentação e facilitam muito o entendimento do código.

# Boas práticas e cuidados ao criar consultas

Algumas dicas, boas práticas e cuidados ao realizar as consultas:

- 5 Terminamos os comandos em SQL usando o ponto e vírgula.

```
-- Selecione apenas as colunas first_name, last_name e salary da tabela EMPLOYEES  
SELECT first_name, last_name, salary FROM employees;
```



# MÓDULO 5

ORDENANDO E FILTRANDO DADOS

ORDENANDO E FILTRANDO DADOS

ORDENANDO E FILTRANDO DADOS

# ORDER BY: Ordenando os dados de uma tabela

Vamos agora falar sobre o ORDER BY. O objetivo do ORDER BY é ordenar uma ou mais colunas de uma tabela de forma ascendente ou descendente. Mas o que significa ascendente ou descendente?

Bom, vai depender do tipo de dado que a coluna a ser ordenada possui.

Se a coluna a ser ordenada for do tipo **NÚMERO**, a ordenação ASCENDENTE significará uma ordenação CRESCENTE, enquanto uma ordenação DESCENDENTE significará uma ordenação DECRESCENTE.

Caso a coluna a ser ordenada seja do tipo **TEXTO**, a ordenação ASCENDENTE significará uma ordenação ALFABÉTICA (A-Z), enquanto uma ordenação DESCENDENTE significará uma ordenação “ANTI”-ALFABÉTICA (Z-A).

Já para uma coluna do tipo **DATA**, a ordenação ASCENDENTE significará uma ordenação CRESCENTE de data (data mais antiga para a data mais recente), enquanto uma ordenação DESCENDENTE significará uma ordenação DECRESCENTE de data (data mais recente para a data mais antiga).

- Usamos a palavra-chave **ASC** para ordenar de forma ASCENDENTE.
- Usamos a palavra-chave **DESC** para ordenar de forma DESCENDENTE.
- Caso o ORDER BY seja usado sem especificar uma das duas formas anteriores, a ordenação padrão é ASCENDENTE (ASC).

# ORDER BY: Sintaxe

A sintaxe usada para o comando ORDER BY é a seguinte:

- **SELECT \* FROM Tabela ORDER BY coluna1;**

Seleciona a tabela ordenando pela **coluna1** em ordem crescente (ASC). Como não especificamos se é ASC ou DESC, a ordenação padrão será ASC.

- **SELECT \* FROM Tabela ORDER BY coluna1 ASC;**

Seleciona a tabela ordenando pela **coluna1** em ordem crescente (ASC). Aqui especificamos que a ordenação será ASC.

- **SELECT \* FROM Tabela ORDER BY coluna1 DESC;**

Seleciona a tabela ordenando pela **coluna1** em ordem decrescente (DESC). Aqui especificamos que a ordenação será DESC.

- **SELECT \* FROM Tabela ORDER BY coluna1 ASC, coluna2 DESC;**

Seleciona a tabela ordenando por duas colunas: **coluna1** em ordem crescente (ASC) e **coluna2** em ordem decrescente (DESC).

# ORDER BY: NULLS FIRST e NULLS LAST

Também é possível ordenar uma coluna de forma que os valores nulos (NULL) sejam mostrados no começo ou no final da coluna.

- **SELECT \* FROM Tabela ORDER BY coluna1 NULLS FIRST;**

Seleciona a tabela ordenando pela **coluna1** em ordem crescente, de forma que os valores nulos apareçam nas **primeiras linhas**.

- **SELECT \* FROM Tabela ORDER BY coluna1 NULLS LAST;**

Seleciona a tabela ordenando pela **coluna1** em ordem crescente, de forma que os valores nulos apareçam nas **últimas linhas**.

- **SELECT \* FROM Tabela ORDER BY coluna1 DESC NULLS FIRST;**

Seleciona a tabela ordenando pela **coluna1** em ordem decrescente, de forma que os valores nulos apareçam nas **primeiras linhas**.

- **SELECT \* FROM Tabela ORDER BY coluna1 DESC NULLS LAST;**

Seleciona a tabela ordenando pela **coluna1** em ordem decrescente, de forma que os valores nulos apareçam nas **últimas linhas**.

# ORDER BY: NULLS FIRST e NULLS LAST

Também é possível ordenar uma coluna de forma que os valores nulos (NULL) sejam mostrados no começo ou no final da coluna.

- **SELECT \* FROM Tabela ORDER BY coluna1 NULLS FIRST;**

Seleciona a tabela ordenando pela **coluna1** em ordem crescente, de forma que os valores nulos apareçam nas **primeiras linhas**.

- **SELECT \* FROM Tabela ORDER BY coluna1 NULLS LAST;**

Seleciona a tabela ordenando pela **coluna1** em ordem crescente, de forma que os valores nulos apareçam nas **últimas linhas**.

- **SELECT \* FROM Tabela ORDER BY coluna1 DESC NULLS FIRST;**

Seleciona a tabela ordenando pela **coluna1** em ordem decrescente, de forma que os valores nulos apareçam nas **primeiras linhas**.

- **SELECT \* FROM Tabela ORDER BY coluna1 DESC NULLS LAST;**

Seleciona a tabela ordenando pela **coluna1** em ordem decrescente, de forma que os valores nulos apareçam nas **últimas linhas**.

# ORDER BY ASC: Aplicação

Agora que já sabemos a lógica por trás do ORDER BY, podemos praticar algumas aplicações.

Podemos selecionar a tabela EMPLOYEES e ordenar de acordo com a coluna salary, que é uma coluna de **números**, em ordem crescente (do menor para o maior).

No exemplo ao lado, observe que a coluna SALARY está ordenada do menor salário para o maior salário, ou seja, em ordem crescente (ASC).

```
-- 1. Ordenando os dados de uma tabela
-- 1.1. Comando ORDER BY com coluna de NÚMEROS

-- a) Faça uma ordenação na tabela EMPLOYEES para visualizar os funcionários do menor
-- para o maior salário.

SELECT * FROM employees
ORDER BY salary;
```

**Resultado da Consulta**

SQL | 50 linhas extraídas em 0,011 segundos

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	TJ	Olson	TJOLSON	650.124.8234	10/04/99	ST_CLERK	2100	(null)	121	5
2	Steven	Markle	SMARKLE	650.124.1434	08/03/00	ST_CLERK	2200	(null)	120	5
3	Hazel	Philtanker	HPHILTAN	650.127.1634	06/02/00	ST_CLERK	2200	(null)	122	5
4	James	Landry	JLANDRY	650.124.1334	14/01/99	ST_CLERK	2400	(null)	120	5
5	Ki	Gee	KGEE	650.127.1734	12/12/99	ST_CLERK	2400	(null)	122	5
6	Karen	Colmenares	KCOLMENA	515.127.4566	10/08/99	PU_CLERK	2500	(null)	114	3
7	James	Marlow	JAMRLOW	650.124.7234	16/02/97	ST_CLERK	2500	(null)	121	5
8	Joshua	Patel	JPATEL	650.121.1834	06/04/98	ST_CLERK	2500	(null)	123	5
9	Peter	Vargas	PVARGAS	650.121.2004	09/07/98	ST_CLERK	2500	(null)	124	5
10	Martha	Sullivan	MSULLIVA	650.507.9878	21/06/99	SH_CLERK	2500	(null)	120	5
11	Randall	Perkins	RPERKINS	650.505.4876	19/12/99	SH_CLERK	2500	(null)	122	5

# ORDER BY ASC: Aplicação

Seguindo a mesma lógica, podemos ordenar uma coluna de **textos** em ordem ascendente (alfabética).

No exemplo ao lado, ordenamos a tabela EMPLOYEES de acordo com a coluna de nome.

Observe que a coluna FIRST\_NAME está ordenando os textos de A até Z.

```
-- a) Ordene a tabela de EMPLOYEES para mostrar os funcionários em ordem alfabética.

SELECT * FROM employees
ORDER BY first_name;
```

Resultado da Consulta x

SQL | 50 linhas extraídas em 0,007 segundos

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID
1	Adam	Tripp	AFRIPP	650.123.2234	10/04/97	ST_MAN	8200	(null)	101
2	Alana	Walsh	AWALSH	650.507.9811	24/04/98	SH_CLERK	3100	(null)	125
3	Alberto	Irrazuriz	AERRAZUR	011.44.1344.429278	10/03/97	SA_MAN	12000	0,3	100
4	Alexander	Hunold	AHUNOLD	590.423.4567	03/01/90	IT_PROG	9000	(null)	102
5	Alexander	Khoo	AKHOO	515.127.4562	18/05/95	PU_CLERK	3100	(null)	111
6	Alexis	Bull	ABULL	650.509.2876	20/02/97	SH_CLERK	4100	(null)	122
7	Allan	McEwen	AMCEWEN	011.44.1345.829268	01/08/96	SA_REP	9000	0,35	140
8	Alyssa	Hutton	AHUTTON	011.44.1644.429266	19/03/97	SA_REP	8800	0,25	141
9	Amit	Banda	ABANDA	011.44.1346.729268	21/04/00	SA_REP	6200	0,1	142
10	Anthony	Cabrio	ACABRIO	650.509.4876	07/02/99	SH_CLERK	3000	(null)	123
11	Britney	Everett	BEVERETT	650.501.2876	03/03/97	SH_CLERK	3900	(null)	124
12	Bruce	Ernst	BERNST	590.423.4568	21/05/91	IT_PROG	6000	(null)	103
13	Charles	Johnson	CJOHNSON	011.44.1644.429262	04/01/00	SA_REP	6200	0,1	143
14	Christopher	Olsen	COLSEN	011.44.1344.498718	30/03/98	SA_REP	8000	0,2	144
15	Clara	Vishney	CVISHNEY	011.44.1346.129268	11/11/97	SA_REP	10500	0,25	145
16	Curtis	Davies	CDAVIES	650.121.2994	29/01/97	ST_CLERK	3100	(null)	124
17	Daniel	Faviet	DFAVIET	515.124.4169	16/08/94	FI_ACCOUNT	9000	(null)	104

# ORDER BY ASC: Aplicação

Por fim, podemos ordenar uma coluna de **datas** em ordem ascendente (data mais antiga para a mais recente).

No exemplo ao lado, ordenamos a tabela EMPLOYEES de acordo com a coluna `hire_date` (data de contratação).

Observe que a coluna `HIRE_DATE` mostra as datas da mais antiga para a mais recente.

```
-- a) Ordene a tabela de EMPLOYEES para mostrar os funcionários de acordo com a data de
-- contratação: da mais antiga pra mais recente.

SELECT * FROM employees
ORDER BY hire_date;
```

Resultado da Consulta | SQL | 50 linhas extraídas em 0,006 segundos

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEF
1	Steven	King	SKING	515.123.4567	17/06/87	AD_PRES	24000	(null)	(null)	
2	Jennifer	Whalen	JWHALEN	515.123.4444	17/09/87	AD_ASST	4400	(null)	101	
3	Neena	Kochhar	NKOCHHAR	515.123.4568	21/09/89	AD_VP	17000	(null)	100	
4	Alexander	Hunold	AHUNOLD	590.423.4567	03/01/90	IT_PROG	9000	(null)	102	
5	Bruce	Ernst	BERNST	590.423.4568	21/05/91	IT_PROG	6000	(null)	103	
6	Lex	De Haan	LDEHAAN	515.123.4569	13/01/93	AD_VP	17000	(null)	100	
7	Susan	Mavris	SMAVRIS	515.123.7777	07/06/94	FI REP	6500	(null)	101	
8	Hermann	Baer	HBAER	515.123.8888	07/06/94	FI REP	10000	(null)	101	
9	Shelley	Higgins	SHIGGINS	515.123.8080	07/06/94	AC_MGR	12000	(null)	101	
10	William	Gietz	WGIETZ	515.123.8181	07/06/94	AC_ACCOUNT	8300	(null)	205	
11	Daniel	Faviet	DFAVIET	515.124.4169	16/08/94	FI ACCOUNT	9000	(null)	108	
12	Nancy	Greenberg	NGREENBE	515.124.4569	17/08/94	FI_MGR	12000	(null)	101	
13	Den	Raphaely	DRAPHEAL	515.127.4561	07/12/94	PU MAN	11000	(null)	100	
14	Payam	Kaufling	PKAUFLIN	650.123.3234	01/05/95	PU MAN	7900	(null)	100	
15	Alexander	Khoo	AKHOO	515.127.4562	18/05/95	PU_CLERK	3100	(null)	114	
16	Renske	Ladwig	RLADWIG	650.121.1234	14/07/95	PU_CLERK	3600	(null)	123	

# ORDER BY DESC: Aplicação

Agora vamos ver como fica a ordenação contrária: uma ordenação DESC (descendente).

Podemos selecionar a tabela EMPLOYEES e ordenar de acordo com a coluna salary, que é uma coluna de **números**, em ordem decrescente (do maior para o menor).

No exemplo ao lado, observe que a coluna SALARY está ordenada do maior salário para o menor salário, ou seja, em ordem decrescente (DESC).

```
-- b) Faça uma ordenação na tabela EMPLOYEES para visualizar os funcionários do maior
-- para o menor salário.

SELECT * FROM employees
ORDER BY salary DESC;
```

**Resultado da Consulta**

SQL | 50 linhas extraídas em 0,011 segundos

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID
1	Steven	King	SKING	515.123.4567	17/06/87	AD_PRES	24000	(null)	(null)
2	Neena	Kochhar	NKOCHHAR	515.123.4568	21/09/89	AD_VP	17000	(null)	100
3	Lex	De Haan	LDEHAAN	515.123.4569	13/01/93	AD_VP	17000	(null)	100
4	John	Russell	JRUSSEL	011.44.1344.429268	01/10/96	SA_MAN	14000	0,4	100
5	Karen	Partners	KPARTNER	011.44.1344.467268	05/01/97	SA_MAN	13500	0,3	100
6	Michael	Hartstein	MHARTSTE	515.123.5555	17/02/96	MK_MAN	13000	(null)	100
7	Nancy	Greenberg	NGREENBE	515.124.4569	17/08/94	FI_MGR	12000	(null)	101
8	Alberto	Errazuriz	AERRAZUR	011.44.1344.429278	10/03/97	SA_MAN	12000	0,3	100
9	Shelley	Higgins	SHIGGINS	515.123.8080	07/06/94	AC_MGR	12000	(null)	101
10	Lisa	Ozer	LOZER	011.44.1343.929268	11/03/97	SA REP	11500	0,25	148
11	Ellen	Abel	EABEL	011.44.1644.429267	11/05/96	SA REP	11000	0,3	149
12	Gerald	Cambrault	GCAMBRAU	011.44.1344.619268	15/10/99	SA_MAN	11000	0,3	100
13	Den	Raphaely	DRAPHEAL	515.127.4561	07/12/94	PU_MAN	11000	(null)	100
14	Eleni	Zlotkey	EZLOTKEY	011.44.1344.429018	29/01/00	SA_MAN	10500	0,2	100
15	Clara	Vishney	CVISHNEY	011.44.1346.129268	11/11/97	SA REP	10500	0,25	147
16	Tiger	Tomasi	TTING	011.44.1345.120268	20/01/96	SA REP	10000	0,25	146

# ORDER BY DESC: Aplicação

Seguindo a mesma lógica, podemos ordenar uma coluna de **textos** em ordem decrescente (“anti”-alfabética).

No exemplo ao lado, ordenamos a tabela EMPLOYEES de acordo com a coluna de nome.

Observe que a coluna FIRST\_NAME está ordenando os textos de Z até A.

```
-- b) Ordene a tabela de EMPLOYEES para mostrar os funcionários em ordem "anti"-alfabética.

SELECT * FROM employees
ORDER BY first_name DESC;
```

Resultado da Consulta x

SQL | 50 linhas extraídas em 0,007 segundos

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEP
1	18	Winston	Taylor	WTAYLOR	650.507.9876	24/01/98	SH_CLERK	3200	(null)	120	
2	17	William	Smith	WSMITH	011.44.1343.629268	23/02/99	SA_REP	7400	0,15	148	
3	20	William	Gietz	WGIETZ	515.123.8181	07/06/94	AC_ACCOUNT	8300	(null)	205	
4	19	Vance	Jones	VJONES	650.501.4876	17/03/99	SH_CLERK	2800	(null)	123	
5	10	Valli	Pataballa	VPATABAL	590.423.4560	05/02/98	IT_PROG	4800	(null)	103	
6	14	Trenna	Rajs	TRAJS	650.121.8009	17/10/95	ST_CLERK	3500	(null)	124	
7	13	TJ	Olson	TJOLSON	650.124.8234	10/04/99	ST_CLERK	2100	(null)	121	
8	19	Timothy	Gates	TGATES	650.505.3876	11/07/98	SH_CLERK	2900	(null)	122	
9	17	Tayler	Fox	TFOX	011.44.1343.729268	24/01/98	SA_REP	9600	0,2	148	
10	20	Susan	Mavris	SMAVRIS	515.123.7777	07/06/94	HR_REP	6500	(null)	101	
11	17	Sundita	Kumar	SKUMAR	011.44.1343.329268	21/04/00	SA_REP	6100	0,1	148	
12	16	Sundar	Ande	SANDE	011.44.1346.629268	24/03/00	SA_REP	6400	0,1	147	
13	10	Steven	King	SKING	515.123.4567	17/06/87	AD_PRE	24000	(null)	(null)	
14	12	Steven	Markle	SMARKLE	650.124.1434	08/03/00	ST_CLERK	2200	(null)	120	
15	13	Stephen	Stiles	SSTILES	650.121.2034	26/10/97	ST_CLERK	3200	(null)	123	
16	11	-----	Montgomery	EMONTGOMERY	515.127.4564	24/07/07	PU_STAFF	2000	(null)	114	

# ORDER BY DESC: Aplicação

Por fim, podemos ordenar uma coluna de **datas** em ordem decrescente (data mais recente para a mais antiga).

No exemplo ao lado, ordenamos a tabela EMPLOYEES de acordo com a coluna hire\_date (data de contratação).

Observe que a coluna HIRE\_DATE mostra as datas da mais recente para a mais antiga, ou seja, em ordem descendente.

```
-- b) Ordene a tabela de EMPLOYEES para mostrar os funcionários de acordo com a data de
-- contratação: da mais recente pra mais antiga.

SELECT * FROM employees
ORDER BY hire_date DESC;
```

**Resultado da Consulta**

SQL | 50 linhas extraídas em 0,01 segundos

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID
1	Sundita	Kumar	SKUMAR	011.44.1343.329268	21/04/00	S_A REP	6100	0,1	148
2	Amit	Banda	ABANDA	011.44.1346.729268	21/04/00	S_A REP	6200	0,1	147
3	Sundar	Ande	SANDE	011.44.1346.629268	24/03/00	S_A REP	6400	0,1	146
4	Steven	Markle	SMARKLE	650.124.1434	08/03/00	S_IT CLERK	2200	(null)	120
5	David	Lee	DLEE	011.44.1346.529268	23/02/00	S_A REP	6800	0,1	145
6	Hazel	Philtanker	PHILLTAN	650.127.1634	06/02/00	S_IT CLERK	2200	(null)	121
7	Girard	Geoni	GGEONI	650.507.9879	03/02/00	S_H CLERK	2800	(null)	120
8	Eleni	Zlotkey	EZLOTKEY	011.44.1344.429018	29/01/00	S_A MAN	10500	0,2	100
9	Mattea	Marvins	MMARVINS	011.44.1346.329268	24/01/00	S_A REP	7200	0,1	144
10	Douglas	Grant	DGRANT	650.507.9844	13/01/00	S_H CLERK	2600	(null)	121
11	Charles	Johnson	CJOHNSON	011.44.1644.429262	04/01/00	S_A REP	6200	0,1	149
12	Randall	Perkins	RPERKINS	650.505.4876	19/12/99	S_H CLERK	2500	(null)	121
13	Ki	Gee	KGEE	650.127.1734	12/12/99	S_IT CLERK	2400	(null)	122
14	Luis	Popp	LPOPP	515.124.4567	07/12/99	S_IT ACCOUNT	6900	(null)	108
15	Oliver	Tuvault	OTUVVAULT	011.44.1344.486508	23/11/99	S_A REP	7000	0,15	149
16	Yannick	Mayer	TMAYER	650.122.5024	16/11/99	S_IT MAN	5000	(null)	121

# ORDER BY: Ordenando mais de uma coluna

Podemos ordenar uma tabela a partir de mais de uma coluna. No exemplo ao lado, ordenamos a tabela EMPLOYEES de acordo com as colunas first\_name e last\_name, ambas em ordem ASC (de A até Z).

Isso significa que, caso tenhamos dois funcionários com o mesmo first\_name, o que vai decidir a ordenação final dos nomes será o last\_name.

Observe por exemplo o nome JAMES. Temos dois funcionários com esses first\_name, mas como ordenamos também pela coluna last\_name em ordem ASC, o JAMES LANDRY vem primeiro.

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME
38	127	James	Landry
39	131	James	Marlow

```
-- a) Alguns funcionários possuem o mesmo first_name (por exemplo, o JAMES).
-- Ordene a tabela EMPLOYEES de acordo com first_name e last_name para fazer o desempate.

SELECT * FROM employees
ORDER BY first_name ASC, last_name ASC;

-- 1.5. Comando ORDER BY com NULLS FIRST e NULLS LAST.

Resultado da Consulta x
SQL | 50 linhas extraídas em 0,127 segundos


| EMPLOYEE_ID | FIRST_NAME  | LAST_NAME | EMAIL       | PHONE_NUMBER       | HIRE_DATE | JOB_ID   | SALARY | COMMISSION_PCT | MANAGER_ID |
|-------------|-------------|-----------|-------------|--------------------|-----------|----------|--------|----------------|------------|
| 1           | Adam        | Fripp     | A.FRIPP     | 650.123.2234       | 10/04/97  | ST_MAN   | 8200   | (null)         | 101        |
| 2           | Alana       | Walsh     | A.WALSH     | 650.507.9811       | 24/04/98  | SH_CLERK | 3100   | (null)         | 121        |
| 3           | Alberto     | Errazuriz | A.ERRAZURIZ | 011.44.1344.429278 | 10/03/97  | SA_MAN   | 12000  | 0,3            | 100        |
| 4           | Alexander   | Hunold    | A.HUNOLD    | 590.423.4567       | 03/01/90  | IT_PROG  | 9000   | (null)         | 102        |
| 5           | Alexander   | Khoo      | A.KHOO      | 515.127.4562       | 18/05/95  | PU_CLERK | 3100   | (null)         | 111        |
| 6           | Alexis      | Bull      | A.BULL      | 650.509.2876       | 20/02/97  | SH_CLERK | 4100   | (null)         | 122        |
| 7           | Allan       | McEwen    | A.MCEWEN    | 011.44.1345.829268 | 01/08/96  | SA_REP   | 9000   | 0,35           | 141        |
| 8           | Alyssa      | Hutton    | A.HUTTON    | 011.44.1644.429266 | 19/03/97  | SA_REP   | 8800   | 0,25           | 142        |
| 9           | Amit        | Banda     | A.BANDA     | 011.44.1346.729268 | 21/04/00  | SA_REP   | 6200   | 0,1            | 143        |
| 10          | Anthony     | Cabrio    | A.CABRIO    | 650.509.4876       | 07/02/99  | SH_CLERK | 3000   | (null)         | 122        |
| 11          | Britney     | Everett   | B.EVERETT   | 650.501.2876       | 03/03/97  | SH_CLERK | 3900   | (null)         | 123        |
| 12          | Bruce       | Ernst     | B.ERNST     | 590.423.4568       | 21/05/91  | IT_PROG  | 6000   | (null)         | 102        |
| 13          | Charles     | Johnson   | C.JOHNSON   | 011.44.1644.429262 | 04/01/00  | SA_REP   | 6200   | 0,1            | 141        |
| 14          | Christopher | Olsen     | C.OLSEN     | 011.44.1344.498718 | 30/03/98  | SA_REP   | 8000   | 0,2            | 142        |
| 15          | Clara       | Vishney   | C.VISHNEY   | 011.44.1346.129268 | 11/11/97  | SA_REP   | 10500  | 0,25           | 143        |
| 16          | David       | Richter   | D.RICHTER   | 650.121.2001       | 20/01/07  | ST_BYRD  | 2100   | (null)         | 122        |


```

# NULLS FIRST e NULLS LAST: Aplicação

A aplicação do NULLS FIRST/NULLS LAST é bem simples.

No exemplo ao lado, usamos o NULLS FIRST para ordenar a coluna manager\_id da tabela DEPARTMENTS para que os valores (null) apareçam primeiro na coluna.

```
-- a) A coluna MANAGER_ID da tabela DEPARTMENTS possui valores nulos. Faça uma ordenação dessa
-- coluna utilizando NULLS FIRST e NULLS LAST.

SELECT * FROM departments
ORDER BY manager_id NULLS FIRST;
```

Resultado da Consulta | SQL | Todas as Linhas Extraídas: 27 em 0,022 segundos

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	140 Control And Credit	(null)	1700
2	260 Recruiting	(null)	1700
3	250 Retail Sales	(null)	1700
4	240 Government Sales	(null)	1700
5	230 IT Helpdesk	(null)	1700
6	220 NOC	(null)	1700
7	210 IT Support	(null)	1700
8	200 Operations	(null)	1700
9	190 Contracting	(null)	1700
10	180 Construction	(null)	1700
11	170 Manufacturing	(null)	1700
12	120 Treasury	(null)	1700
13	130 Corporate Tax	(null)	1700
14	270 Payroll	(null)	1700
15	150 Shareholder Services	(null)	1700

# NULLS FIRST e NULLS LAST: Aplicação

Já o NULLS LAST será usado para ordenar uma coluna de forma que os valores (null) apareçam por último.

No exemplo ao lado, usamos o NULLS LAST de forma que os valores (null) são mostrados no final da tabela.

```
-- a) A coluna MANAGER_ID da tabela DEPARTMENTS possui valores nulos. Faça uma ordenação dessa coluna utilizando NULLS FIRST e NULLS LAST.

SELECT * FROM departments
ORDER BY manager_id NULLS LAST;
```

Resultado da Consulta | SQL | Todas as Linhas Extraídas: 27 em 0,006 segundos

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	90 Executive	100	1700
2	60 IT	103	1400
3	100 Finance	108	1700
4	30 Purchasing	114	1700
5	50 Shipping	121	1500
6	80 Sales	145	2500
7	10 Administration	200	1700
8	20 Marketing	201	1800
9	40 Human Resources	203	2400
10	70 Public Relations	204	2700
11	110 Accounting	205	1700
12	230 IT Helpdesk	(null)	1700
13	240 Government Sales	(null)	1700
14	250 Retail Sales	(null)	1700
15	260 Recruiting	(null)	1700
16	220 NOC	(null)	1700

# NULLS FIRST e NULLS LAST: Aplicação

Podemos também combinar o NULLS FIRST/NULLS LAST com o DESC, para escolher a forma de ordenação (ascendente ou descendente).

Por padrão, NULLS FIRST/NULLS LAST sozinhos ordenarão os valores em ordem ascendente.

Para ordenar os valores em ordem descendente, incluímos o DESC logo antes do NULLS FIRST/NULLS LAST, como mostrado na imagem ao lado.

O exemplo ao lado é bem semelhante que o do slide anterior (volte no slide anterior para comparar), com a diferença que a coluna manager\_id da tabela DEPARTMENTS agora é organizada em ordem decrescente.

```
-- a) A coluna MANAGER_ID da tabela DEPARTMENTS possui valores nulos. Faça uma ordenação dessa
-- coluna utilizando NULLS FIRST e NULLS LAST.

SELECT * FROM departments
ORDER BY manager_id DESC NULLS LAST;
```

Resultado da Consulta | SQL | Todas as Linhas Extraídas: 27 em 0,048 segundos

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	110 Accounting	205	1700
2	70 Public Relations	204	2700
3	40 Human Resources	203	2400
4	20 Marketing	201	1800
5	10 Administration	200	1700
6	80 Sales	145	2500
7	50 Shipping	121	1500
8	30 Purchasing	114	1700
9	100 Finance	108	1700
10	60 IT	103	1400
11	90 Executive	100	1700
12	230 IT Helpdesk	(null)	1700
13	240 Government Sales	(null)	1700
14	250 Retail Sales	(null)	1700
15	260 Recruiting	(null)	1700
16	220 NOC	(null)	1700

# WHERE: Introdução

O comando WHERE nos permite criar filtros nas tabelas dos bancos de dados.

Imagine que tenhamos a tabela ao lado. Ela possui 4 colunas, dentre elas a coluna JOB\_ID, que identifica qual é o ID do cargo de cada funcionário.

Nessa tabela temos todos os funcionários, incluindo todos os cargos. Mas e se a gente quisesse analisar apenas os funcionários com JOB\_ID igual a IT\_PROG?

Neste caso, não faria sentido a gente ter todos os dados na tabela, precisaríamos apenas dos funcionários com JOB\_ID='IT\_PROG'.

Para isso, teríamos que criar um filtro na nossa coluna JOB\_ID.

EMPLOYEE_ID	FIRST_NAME	JOB_ID	SALARY
100	Steven	AD_PRES	24000
101	Neena	AD_VP	17000
102	Lex	AD_VP	17000
103	Alexander	IT_PROG	9000
104	Bruce	IT_PROG	6000
105	David	IT_PROG	4800
106	Valli	IT_PROG	4800
107	Diana	IT_PROG	4200
108	Nancy	FI_MGR	12000
109	Daniel	FI_ACCOUNT	9000
110	John	FI_ACCOUNT	8200
111	Ismael	FI_ACCOUNT	7700
112	Jose Manuel	FI_ACCOUNT	7800
113	Luis	FI_ACCOUNT	6900
114	Den	PU_MAN	11000
115	Alexander	PU_CLERK	3100
116	Shelli	PU_CLERK	2900
117	Sigal	PU_CLERK	2800
118	Guy	PU_CLERK	2600
119	Karen	PU_CLERK	2500
120	Matthew	ST_MAN	8000
121	Adam	ST_MAN	8200
122	Pavam	ST_MAN	7900

# WHERE: Introdução

Observe que de todos os funcionários da tabela EMPLOYEES, temos apenas 5 que são IT\_PROG.

Conseguimos agora visualizar apenas os dados que nos interessam, deixando a tabela muito mais simplificada e de fácil entendimento.

Para conseguir chegar nesse resultado temos que fazer um filtro na nossa tabela. Este filtro conseguiremos fazer utilizando o comando **WHERE**.

O WHERE nos permitirá fazer uma série de filtros nas nossas tabelas, desde os filtros mais básicos até os filtros mais avançados.

Vamos então aprender a usar estes filtros na prática.

EMPLOYEE_ID	FIRST_NAME	JOB_ID	SALARY
103	Alexander	IT_PROG	9000
104	Bruce	IT_PROG	6000
105	David	IT_PROG	4800
106	Valli	IT_PROG	4800
107	Diana	IT_PROG	4200

# WHERE: Sintaxe

A sintaxe do WHERE é bem simples e é mostrada na figura 1 ao lado.

Fazemos um SELECT FROM normal e ao final adicionamos o WHERE seguido de uma **condição**.

Por exemplo, para chegar na tabela do slide anterior, precisamos filtrar a nossa tabela EMPLOYEES para que mostrasse apenas os funcionários cujo JOB\_ID é igual a 'IT\_PROG'.

Para isso, usamos o código ao lado, onde a nossa **condição** é **JOB\_ID = 'IT\_PROG'**.

```
SELECT coluna1, coluna2, ...  
FROM tabela  
WHERE condicao;
```

1

```
SELECT employee_id, first_name, job_id, salary  
FROM employees  
WHERE job_id = 'IT_PROG';
```

2

# Operadores básicos de filtros

Podemos utilizar o WHERE em conjunto com uma série de operadores, listados a seguir.

Operador	Tradução	Finalidade
=	Igual a	Filtrar todos os valores que são iguais a um determinado valor.
>	Maior que	Filtrar todos os valores que são maiores que um determinado valor.
<	Menor que	Filtrar todos os valores que são menores que um determinado valor.
>=	Maior ou igual a	Filtrar todos os valores que são maiores ou iguais a um determinado valor.
<=	Menor ou igual a	Filtrar todos os valores que são menores ou iguais a um determinado valor.
<>	Diferente de	Filtrar todos os valores que são diferentes de determinado valor.
AND	E	Usado para filtrar múltiplas condições usando a lógica E. O AND mostra as linhas se todas as condições forem verdadeiras.
OR	OU	Usado para filtrar múltiplas condições usando a lógica OU. O OR mostra as linhas se uma ou mais condições forem verdadeiras.
LIKE	COMO, SEMELHANTE	Usado para pesquisar um padrão especificado em uma coluna.
IN	EM	Permite especificar vários valores dentro do WHERE. Pode ser uma alternativa ao OR.
BETWEEN	ENTRE	Permite selecionar valores dentro de um determinado intervalo de forma inclusiva: os valores inicial e final são incluídos.
IS NULL	É NULO	Permite filtrar valores nulos.
NOT	NÃO	Permite fazer a negação dos operadores anteriores.

# Operadores básicos de filtros

Vamos começar com os operadores básicos listados a seguir.

Operador	Tradução	Finalidade
=	Igual a	Filtrar todos os valores que são iguais a um determinado valor.
>	Maior que	Filtrar todos os valores que são maiores que um determinado valor.
<	Menor que	Filtrar todos os valores que são menores que um determinado valor.
>=	Maior ou igual a	Filtrar todos os valores que são maiores ou iguais a um determinado valor.
<=	Menor ou igual a	Filtrar todos os valores que são menores ou iguais a um determinado valor.
<>	Diferente de	Filtrar todos os valores que são diferentes de determinado valor.
AND	E	Usado para filtrar múltiplas condições usando a lógica E. O AND mostra as linhas se todas as condições forem verdadeiras.
OR	OU	Usado para filtrar múltiplas condições usando a lógica OU. O OR mostra as linhas se uma ou mais condições forem verdadeiras.
LIKE	COMO, SEMELHANTE	Usado para pesquisar um padrão especificado em uma coluna.
IN	EM	Permite especificar vários valores dentro do WHERE. Pode ser uma alternativa ao OR.
BETWEEN	ENTRE	Permite selecionar valores dentro de um determinado intervalo de forma inclusiva: os valores inicial e final são incluídos.
IS NULL	É NULO	Permite filtrar valores nulos.
NOT	NÃO	Permite fazer a negação dos operadores anteriores.

# WHERE: Filtros de Números

Nossa primeira aplicação do WHERE será com filtros de **números**. De acordo com a tabela anterior, temos diversos comparadores que podemos usar: =, <, >, <=, >=, <>.

No exemplo ao lado, filtramos a tabela EMPLOYEES para mostrar apenas os funcionários que são do departamento igual a 100.

```
-- a) Filtre a tabela EMPLOYEES e retorne apenas  
-- os funcionários do departamento ID = 100  
  
SELECT employee_id, first_name, salary, department_id  
FROM employees  
WHERE department_id = 100;  
  
-- b) Filtre a tabela JOBS e retorne aqueles que tem um sal
```

Resultado da Consulta | Todas as Linhas Extraídas: 6 em 0,009 segundos

EMPLOYEE_ID	FIRST_NAME	SALARY	DEPARTMENT_ID
1	Nancy	12000	100
2	Daniel	9000	100
3	John	8200	100
4	Ismael	7700	100
5	Jose Manuel	7800	100
6	Luis	6900	100

# WHERE: Filtros de Números

Já no exemplo ao lado, filtramos a tabela JOBS para mostrar apenas os cargos com salário mínimo maior que 6.000.

```
-- b) Filtre a tabela JOBS e retorne aqueles que
-- tem um salário mínimo maior que 6000

SELECT *
FROM jobs
WHERE min_salary > 6000;

-- c) Filtre a tabela EMPLOYEES para retornar apenas o job_id = ST_MAN

SELECT
```

Resultado da Consulta | Todas as Linhas Extraídas: 7 em 0,022 segundos

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
1	AD_PRES President	20000	40000
2	AD_VP Administration Vice President	15000	30000
3	FI_MGR Finance Manager	8200	16000
4	AC_MGR Accounting Manager	8200	16000
5	SA_MAN Sales Manager	10000	20000
6	PU_MAN Purchasing Manager	8000	15000
7	MK_MAN Marketing Manager	9000	15000

# WHERE: Filtros de Texto

Para filtrar colunas de **texto**, basta seguir o exemplo ao lado.

Queremos todos os job\_id iguais a 'ST\_MAN'.

Observe que o texto ST\_MAN é escrito entre aspas simples. Sempre que quisermos filtrar uma tabela a partir de uma coluna de textos, a palavra deve estar entre aspas simples.

**OBS:** o Oracle é case sensitive, ou seja:

- WHERE job\_id = 'ST\_MAN'
- WHERE job\_id = 'st\_man'

são coisas diferentes.

```
-- c) Filtre a tabela EMPLOYEES para retornar apenas o job_id = ST_MAN

SELECT
    first_name,
    last_name,
    salary,
    job_id
FROM employees
WHERE job_id = 'ST_MAN';



Resultado da Consulta



Todas as Linhas Extraídas: 5 em 0,009 segundos



|   | FIRST_NAME | LAST_NAME | SALARY | JOB_ID |
|---|------------|-----------|--------|--------|
| 1 | Matthew    | Weiss     | 8000   | ST_MAN |
| 2 | Adam       | Fripp     | 8200   | ST_MAN |
| 3 | Payam      | Kaufling  | 7900   | ST_MAN |
| 4 | Shanta     | Vollman   | 6500   | ST_MAN |
| 5 | Kevin      | Mourgos   | 5800   | ST_MAN |


```

# WHERE: Filtros de Data

Para filtrar colunas de **data**, também colocamos essa data entre aspas simples, como se fosse um texto.

No exemplo ao lado, estamos retornando todos os funcionários da tabela EMPLOYEES que têm uma data de contratação maior ou igual a '01/01/2000'.

```
-- d) Filtre a tabela EMPLOYEES para mostrar apenas os funcionários que foram contratados
-- após o dia 01/01/00

SELECT
  *
FROM employees
WHERE hire_date >= '01/01/00';


```

Resultado da Consulta | SQL | Todas as Linhas Extraídas: 11 em 0,011 segundos

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID
1	128	Steven	Markle	SMARKLE	650.124.1434	08/03/00	ST_CLERK	2200	(null)
2	136	Hazel	Philtanker	HPHILTAN	650.127.1634	06/02/00	ST_CLERK	2200	(null)
3	149	Eleni	Zlotkey	EZLOTKEY	011.44.1344.429018	29/01/00	SA_MAN	10500	0,2
4	164	Mattea	Marvins	MMARVINS	011.44.1346.329268	24/01/00	SA REP	7200	0,1
5	165	David	Lee	DLEE	011.44.1346.529268	23/02/00	SA REP	6800	0,1
6	166	Sundar	Ande	SANDE	011.44.1346.629268	24/03/00	SA REP	6400	0,1
7	167	Amit	Banda	ABANDA	011.44.1346.729268	21/04/00	SA REP	6200	0,1
8	173	Sundita	Kumar	SKUMAR	011.44.1343.329268	21/04/00	SA REP	6100	0,1
9	179	Charles	Johnson	CJOHNSON	011.44.1644.429262	04/01/00	SA REP	6200	0,1
10	183	Girard	Geoni	GGEONI	650.507.9879	03/02/00	SH_CLERK	2800	(null)

# Operadores AND, OR e NOT

Vamos agora usar os operadores AND, OR e NOT.

Operador	Tradução	Finalidade
=	Igual a	Filtrar todos os valores que são iguais a um determinado valor.
>	Maior que	Filtrar todos os valores que são maiores que um determinado valor.
<	Menor que	Filtrar todos os valores que são menores que um determinado valor.
>=	Maior ou igual a	Filtrar todos os valores que são maiores ou iguais a um determinado valor.
<=	Menor ou igual a	Filtrar todos os valores que são menores ou iguais a um determinado valor.
<>	Diferente de	Filtrar todos os valores que são diferentes de determinado valor.
AND	E	Usado para filtrar múltiplas condições usando a lógica E. O AND mostra as linhas se todas as condições forem verdadeiras.
OR	OU	Usado para filtrar múltiplas condições usando a lógica OU. O OR mostra as linhas se uma ou mais condições forem verdadeiras.
LIKE	COMO, SEMELHANTE	Usado para pesquisar um padrão especificado em uma coluna.
IN	EM	Permite especificar vários valores dentro do WHERE. Pode ser uma alternativa ao OR.
BETWEEN	ENTRE	Permite selecionar valores dentro de um determinado intervalo de forma inclusiva: os valores inicial e final são incluídos.
IS NULL	É NULO	Permite filtrar valores nulos.
NOT	NÃO	Permite fazer a negação dos operadores anteriores.

# Operador AND

Até agora, vimo como fazer filtros usando um critério em apenas uma coluna (department\_id=100, salary>6000, job\_id='ST\_MAN').

O que fazemos quando precisamos filtrar a partir de mais de uma coluna? Imagine que você queria uma lista com todos os funcionários com job\_id='IT\_PROG', mas especificamente aquelas que tem o salary $\geq$ 5000. Neste caso, precisamos filtrar a partir de duas colunas: job\_id e salary. A solução é muito simples, basta usarmos o operador AND para fazer estes múltiplos testes, como mostrado na imagem abaixo.

```
-- 1.2. Filtro WHERE combinado com AND e OR.  
-- a) Quais funcionários têm o JOB_ID = 'IT_PROG' e SALARY >= 5000?  
  
SELECT * FROM employees  
WHERE job_id = 'IT_PROG' AND salary >= 5000;
```

Resultado da Consulta x

SQL | Todas as Linhas Extraídas: 2 em 0,14 segundos

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	Alexander	Hunold	AHUNOLD	590.423.4567	03/01/90	IT_PROG	9000	(null)	102	60
2	Bruce	Ernst	BERNST	590.423.4568	21/05/91	IT_PROG	6000	(null)	103	60

# Operador OR

Agora observe a seguinte situação: queremos todos os funcionários que são do department\_id igual a 90 ou igual a 100. Como resolver nesse caso?

O AND não vai nos ajudar porque ele pressupõe que todos os testes sejam verdade para que as linhas da tabela sejam retornadas. Só que dessa vez, basta que uma das duas condições sejam atendidas (department\_id=90 OU department\_id=100) e pra gente já é o suficiente. Se as duas forem verdadeiras então, melhor ainda. O código e o resultado são mostrados abaixo.

```
-- b) Quais funcionários são do departamento 90 ou 100?

SELECT * FROM employees
WHERE department_id = 90 OR department_id = 100;

-- 1.3. Filtros LIKE, BETWEEN, IN, IS NULL e NOT.

-- a) Quais funcionários possuem um JOB_ID que comecam com o texto: 'ST'?
```

**Resultado da Consulta**

Todas as Linhas Extraídas: 9 em 0,013 segundos

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	Steven	King	SKING	515.123.4567	17/06/87	AD_PRES	24000	(null)	(null)	90
2	Neena	Kochhar	NKOCHHAR	515.123.4568	21/09/89	AD_VP	17000	(null)	10	90
3	Lex	De Haan	LDEHAAN	515.123.4569	13/01/93	AD_VP	17000	(null)	10	90
4	Nancy	Greenberg	NGREENBE	515.124.4569	17/08/94	FI_MGR	12000	(null)	10	100
5	Daniel	Faviet	DFAVIET	515.124.4169	16/08/94	FI_ACCOUNT	9000	(null)	10	100
6	John	Chen	JCHEN	515.124.4269	28/09/97	FI_ACCOUNT	8200	(null)	10	100
7	Ismael	Sciarra	ISCIARRA	515.124.4369	30/09/97	FI_ACCOUNT	7700	(null)	10	100
8	Jose Manuel	Urman	JMURMAN	515.124.4469	07/03/98	FI_ACCOUNT	7800	(null)	10	100
9	Luis	Popp	LPOPP	515.124.4567	07/12/99	FI_ACCOUNT	6900	(null)	10	100

# Operador NOT

O operador NOT é um operador de negação, que sempre vai trazer o oposto do que estiver no filtro. Por exemplo, se quisermos todos os funcionários que são do department\_id que não seja o 90, podemos usar a solução da imagem abaixo (também poderíamos usar o operador <> para fazer o filtro de ‘diferente de’, mas com o NOT conseguiremos negar qualquer tipo de filtro, como veremos em aplicações mais a frente).

```
-- b) Quais funcionários são do departamento 90 ou 100?

SELECT * FROM employees
WHERE NOT department_id = 90
ORDER BY department_id DESC;

-- 1.3. Filtros LIKE, BETWEEN, IN, IS NULL e NOT.
```

FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
Shelley	Higgins	SHIGGINS	515.123.8080	07/06/94	AC_MGR	12000	(null)	101	110
William	Gietz	WGIETZ	515.123.8181	07/06/94	AC_ACCOUNT	8300	(null)	205	110
John	Chen	JCHEN	515.124.4269	28/09/97	FI_ACCOUNT	8200	(null)	103	100
Jose Manuel	Urman	JMURMAN	515.124.4469	07/03/98	FI_ACCOUNT	7800	(null)	103	100
Luis	Popp	LPOPP	515.124.4567	07/12/99	FI_ACCOUNT	6900	(null)	103	100
Daniel	Faviet	DFAVIET	515.124.4169	16/08/94	FI_ACCOUNT	9000	(null)	103	100
Nancy	Greenberg	NGREENBE	515.124.4569	17/08/94	FI_MGR	12000	(null)	101	100
Ismael	Sciarra	ISCIARRA	515.124.4369	30/09/97	FI_ACCOUNT	7700	(null)	103	100
John	Russell	JRUSSEL	011.44.1344.429268	01/10/96	SA_MAN	14000	0,4	100	80
Karen	Partners	KPARTNER	011.44.1344.467268	05/01/97	SA_MAN	13500	0,3	100	80

# Operador LIKE e Wildcards

O operador LIKE é usado junto com o WHERE para procurar por algum padrão específico de texto em uma coluna.

Associado ao LIKE, usamos também os chamados *wildcards* (caracteres curinga) que vão dar essa flexibilidade de busca dos padrões de texto.

Existem dois *wildcards* frequentemente usados junto com o LIKE:

- O sinal de porcentagem ( % ) representa zero, um ou múltiplos caracteres.
- O underline ( \_ ) um único caractere

Podemos usar tanto o % quanto o \_ em conjunto para criar filtros ainda mais personalizados.

A sintaxe do LIKE é mostrada na imagem ao lado.

```
SELECT coluna1, coluna2 ...
FROM tabela
WHERE colunaN LIKE padrao;
```

```
SELECT coluna1, coluna2, ...
FROM tabela
WHERE colunaN NOT LIKE padrao;
```

# Operador LIKE e Wildcards

Vamos a alguns exemplos mostrando os diferentes casos do operador LIKE em conjunto com os wildcards ‘%’ e ‘\_’.

Aplicação do LIKE	Descrição
WHERE first_name LIKE 'a%'	Encontra valores que começam com “a”
WHERE first_name LIKE '%a'	Encontra valores que terminam com “a”
WHERE first_name LIKE '%ao%	Encontra valores que têm o “ao” em qualquer posição do texto
WHERE first_name LIKE '_a%	Encontra valores que têm começam com uma letra qualquer e que têm a letra “a” na segunda posição
WHERE first_name LIKE 'a_%'	Encontra valores que começam com a letra “a” e têm pelo menos 2 caracteres
WHERE first_name LIKE 'a__%	Encontra valores que começam com a letra “a” e têm pelo menos 3 caracteres
WHERE first_name LIKE 'a%o'	Encontra valores que começam com a letra “a” e terminam com a letra “o”

Veremos na prática.

# Operador LIKE e Wildcards

Começamos com o exemplo abaixo. Vamos retornar a lista de funcionários que possuem um JOB\_ID que **começa** com o texto 'ST'.

De acordo com o que vimos na tabela anterior, a solução é mostrada abaixo.

```
-- a) Quais funcionários possuem um JOB_ID que começam com o texto: 'ST'?

SELECT * FROM employees
WHERE job_id LIKE 'ST%';

Resultado da Consulta x
SQL | Todas as Linhas Extraídas: 25 em 0,052 segundos
```

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	125	Julia	Nayer	JNAYER	650.124.1214	16/07/97	ST_CLERK	3200	(null)	120	50
2	126	Irene	Mikkilineni	IMIKKILI	650.124.1224	28/09/98	ST_CLERK	2700	(null)	120	50
3	127	James	Landry	JLANDRY	650.124.1334	14/01/99	ST_CLERK	2400	(null)	120	50
4	128	Steven	Markle	SMARKLE	650.124.1434	08/03/00	ST_CLERK	2200	(null)	120	50
5	129	Laura	Bissot	LBISSOT	650.124.5234	20/08/97	ST_CLERK	3300	(null)	121	50
6	130	Mozhe	Atkinson	MATKINSO	650.124.6234	30/10/97	ST_CLERK	2800	(null)	121	50
7	131	James	Marlow	JAMRLOW	650.124.7234	16/02/97	ST_CLERK	2500	(null)	121	50
8	132	TJ	Olson	TJOLSON	650.124.8234	10/04/99	ST_CLERK	2100	(null)	121	50
9	133	Jason	Mallin	JMALLIN	650.127.1934	14/06/96	ST_CLERK	3300	(null)	122	50
10	134	Michael	Rogers	MROGERS	650.127.1834	26/08/98	ST_CLERK	2900	(null)	122	50
11	135	Ki	Gee	KGEE	650.127.1734	12/12/99	ST_CLERK	2400	(null)	122	50

# Operador LIKE e Wildcards

Neste outro exemplo, queremos todos os funcionários que têm o número '123' na segunda parte do telefone. Como o padrão dos números é sempre ter 3 caracteres no começo + ponto + a segunda parte + ponto + 4 caracteres, podemos usar o wildcard \_, como mostrado na imagem abaixo. Ou seja, começamos repetindo o \_ 4 vezes no começo, e depois do 123, repetimos mais 5 vezes (lembmando que o ponto também conta como um caractere).

```
-- c) O telefone é formado por 3 partes: AAA.BBB.CCCC. Quais telefones têm a 2ª parte igual a 123?

SELECT * FROM employees
WHERE phone_number LIKE '____123_____';
```

**Resultado da Consulta** | Todas as Linhas Extraídas: 15 em 0,009 segundos

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	100	Steven King	SKING	515.123.4567	17/06/87	AD_PRES	24000	(null)	(null)	90
2	101	Neena Kochhar	NKOCHHAR	515.123.4568	21/09/89	AD_VP	17000	(null)	100	90
3	102	Lex De Haan	LDEHAAN	515.123.4569	13/01/93	AD_VP	17000	(null)	100	90
4	120	Matthew Weiss	MWEISS	650.123.1234	18/07/96	ST_MAN	8000	(null)	100	50
5	121	Adam Fripp	AFRIPP	650.123.2234	10/04/97	ST_MAN	8200	(null)	100	50
6	122	Payam Kaufling	PKAUFLING	650.123.3234	01/05/95	ST_MAN	7900	(null)	100	50
7	123	Shanta Vollman	SVOLLMA	650.123.4234	10/10/97	ST_MAN	6500	(null)	100	50
8	124	Kevin Mourgos	KMOURGO	650.123.5234	16/11/99	ST_MAN	5800	(null)	100	50
9	200	Jennifer Whalen	JWHALEN	515.123.4444	17/09/87	AD_ASST	4400	(null)	101	10
10	201	Michael Hartstein	MHARTST	515.123.5555	17/02/96	MK_MAN	13000	(null)	100	20
...										

# Operador LIKE e Wildcards

No exemplo anterior você pode ter ficado com a seguinte dúvida: **por que não usar o '%123%' pra encontrar a sequência de números no meio?**

Lembre-se que o *wildcard* % significa zero, um ou múltiplos caracteres. Então quando colocamos o % no começo e % no fim, estamos dizendo que o 123 pode estar em qualquer lugar dentro do texto.

Se usássemos o '%123%', observe na linha 9, o funcionário Renske. A segunda parte do telefone dele não é 123 e sim 121. E porque ele retornou essa linha? Porque na terceira parte, a sequência 123 está presente (...1234). Por isso essa linha foi retornada.

Portanto, avalie a situação e veja qual dos wildcards fazem mais sentido de usar: o % ou o \_.

```
-- c) O telefone é formado por 3 partes: AAA.BBB.CCCC. Quais telefones têm a 2ª parte igual a 123?

SELECT * FROM employees
WHERE phone_number LIKE '%123%';

Resultado da Consulta
```

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER
1	100	Steven	King	SKING	515.123.4567
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568
3	102	Lex	De Haan	LDEHAAN	515.123.4569
4	120	Matthew	Weiss	MWEISS	650.123.1234
5	121	Adam	Fripp	AFRIPP	650.123.2234
6	122	Payam	Kaufling	PKAUFLIN	650.123.3234
7	123	Shanta	Vollman	SVOLLMAN	650.123.4234
8	124	Kevin	Mourgos	KMOURGOS	650.123.5234
9	137	Renske	Ladwig	RLADWIG	650.121.1234
10	200	Jennifer	Whalen	JWHALEN	515.123.4444
...	...	...	...	...	...

# Operador BETWEEN

O operador BETWEEN é usado junto com o WHERE para selecionar valores dentro de um intervalo. Esses valores podem ser números, textos ou datas.

O operador BETWEEN é inclusivo, ou seja, os valores do começo e do fim do intervalo também estão incluídos.

A sintaxe desse operador é mostrada ao lado.

Vamos agora ver alguns exemplos.

```
SELECT coluna1, coluna2, ...
FROM tabela
WHERE colunaN BETWEEN valor1 AND valor2;
```

```
SELECT coluna1, coluna2, ...
FROM tabela
WHERE colunaN NOT BETWEEN valor1 AND valor2;
```

# Operador BETWEEN: números

No exemplo ao lado queremos selecionar todos os funcionários que têm um salário entre 10 mil e 30 mil. A aplicação do BETWEEN é bem intuitiva.

```
-- a) Selecione os funcionários que recebam um salário ENTRE 10.000 e 30.000.  
SELECT * FROM employees  
WHERE salary BETWEEN 10000 AND 30000;
```

Resultado da Consulta | SQL | Todas as Linhas Extraídas: 19 em 0,008 segundos

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	100	Steven	King	SKING	515.123.4567	17/06/87	AD_PRE	24000	(null)	(null)
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	21/09/89	AD_VP	17000	(null)	100
3	102	Lex	De Haan	LDEHAAN	515.123.4569	13/01/93	AD_VP	17000	(null)	100
4	108	Nancy	Greenberg	NGREENBE	515.124.4569	17/08/94	FI_MGR	12000	(null)	101
5	114	Den	Raphaely	DRAPHEAL	515.127.4561	07/12/94	PU_MAN	11000	(null)	100
6	145	John	Russell	JRUSSEL	011.44.1344.429268	01/10/96	SA_MAN	14000	0,4	100
7	146	Karen	Partners	KPARTNER	011.44.1344.467268	05/01/97	SA_MAN	13500	0,3	100
8	147	Alberto	Errazuriz	AERRAZUR	011.44.1344.429278	10/03/97	SA_MAN	12000	0,3	100
9	148	Gerald	Cambrault	GCAMBRAU	011.44.1344.619268	15/10/99	SA_MAN	11000	0,3	100
10	149	Eleni	Zlotkey	EZLOTKEY	011.44.1344.429018	29/01/00	SA_MAN	10500	0,2	100
11	150	-	-	-	-	-	-	-	-	-

# Operador BETWEEN: textos

Agora em um caso de textos: podemos usar o BETWEEN para mostrar a lista de funcionários que têm o primeiro nome começando em 'A' até os funcionários que têm o primeiro nome começando em 'D'.

Lembre-se que o Oracle é case sensitive, então como os nomes começam com letra minúscula, as letras A e D do filtro também devem estar em maiúscula.

```
-- b) Selecione os funcionários que têm nomes começando com a letra 'A' até a letra 'D'.
SELECT * FROM employees
WHERE first_name BETWEEN 'A' AND 'D';
```

Resultado da Consulta | Todas as Linhas Extraídas: 16 em 0,012 segundos

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEP
1	Amit	Banda	ABANDA	011.44.1346.729268	21/04/00	SA_REP	6200	0,1	147	
2	Alexis	Bull	ABULL	650.509.2876	20/02/97	SH_CLERK	4100	(null)	121	
3	Anthony	Cabrio	ACABRIO	650.509.4876	07/02/99	SH_CLERK	3000	(null)	121	
4	Curtis	Davies	CDAVIES	650.121.2994	29/01/97	ST_CLERK	3100	(null)	124	
5	Bruce	Ernst	BERNST	590.423.4568	21/05/91	IT_PROG	6000	(null)	103	
6	Alberto	Irrazuriz	AERRAZUR	011.44.1344.429278	10/03/97	SA_MAN	12000	0,3	100	
7	Britney	Iverett	BEVERETT	650.501.2876	03/03/97	SH_CLERK	3900	(null)	123	
8	Adam	Tripp	AFRIPP	650.123.2234	10/04/97	ST_MAN	8200	(null)	100	
9	Alexander	Junold	AHUNOLD	590.423.4567	03/01/90	IT_PROG	9000	(null)	102	
10	Alyssa	Button	AHUTTON	011.44.1644.429266	19/03/97	SA_REP	8800	0,25	149	
11										
12										
13										
14										
15										
16										
17										
18										
19										
20										

# Operador BETWEEN: datas

Também podemos usar o BETWEEN com datas. No exemplo ao lado, queremos retornar todos os funcionários que foram contratados entre os dias '01/01/1999' e '31/12/2000'.

```
-- c) Selecione os funcionários que foram contratados entre as datas '01/01/99' e '31/12/00'

SELECT * FROM employees
WHERE hire_date BETWEEN '01/01/99' AND '31/12/00';


```

**Resultado da Consulta**

Todas as Linhas Extraídas: 29 em 0,004 segundos

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID
1	Diana	Lorentz	DLORENTZ	590.423.5567	07/02/99	IT_PROG	4200	(null)	103
2	Luis	Popp	LPOPP	515.124.4567	07/12/99	TI_ACCOUNT	6900	(null)	108
3	Karen	Colmenares	KCOLMENA	515.127.4566	10/08/99	PU_CLERK	2500	(null)	114
4	Kevin	Mourgos	KMOURGOS	650.123.5234	16/11/99	ST_MAN	5800	(null)	100
5	James	Landry	JLANDRY	650.124.1334	14/01/99	ST_CLERK	2400	(null)	120
6	Steven	Markle	SMARKLE	650.124.1434	08/03/00	ST_CLERK	2200	(null)	120
7	TJ	Olson	TJOLSON	650.124.8234	10/04/99	ST_CLERK	2100	(null)	121
8	Ki	Gee	KGEE	650.127.1734	12/12/99	ST_CLERK	2400	(null)	122
9	Hazel	Philtanker	PHILLTAN	650.127.1634	06/02/00	ST_CLERK	2200	(null)	122
10	Gerald	Cambrault	GCAMBRAU	011.44.1344.61926	15/10/99	SA_MAN	11000	0,3	100

# Operador IN

O operador IN é usado junto com o WHERE e permite especificar múltiplos valores no filtro.

A sintaxe desse operador é mostrada ao lado.

Vamos agora ver alguns exemplos.

```
SELECT coluna1, coluna2, ...
FROM tabela
WHERE colunaN IN (valor1, valor2, ...);
```

```
SELECT coluna1, coluna2, ...
FROM tabela
WHERE colunaN NOT IN (valor1, valor2, ...);
```

# Operador IN

No exemplo ao lado, queremos todos os funcionários dos departamentos 30, 50 ou 80.

```
-- a) Selecione os funcionários que são de um dos seguintes departamentos: 30, 50, 80.  
SELECT * FROM employees  
WHERE department_id IN (30, 50, 80);
```

Resultado da Consulta | SQL | 50 linhas extraídas em 0,016 segundos

FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1 Den	Raphaely	DRAPHEAL	515.127.4561	07/12/94	PU_MAN	11000	(null)	10	30
2 Alexander	Khoo	AKHOO	515.127.4562	18/05/95	PU_CLERK	3100	(null)	11	30
3 Shelli	Baida	SBAIDA	515.127.4563	24/12/97	PU_CLERK	2900	(null)	11	30
4 Sigal	Tobias	STOBIAS	515.127.4564	24/07/97	PU_CLERK	2800	(null)	11	30
5 Guy	Himuro	GHIMURO	515.127.4565	15/11/98	PU_CLERK	2600	(null)	11	30
6 Karen	Colmenares	KCOLMENA	515.127.4566	10/08/99	PU_CLERK	2500	(null)	11	30
7 Matthew	Weiss	MWEISS	650.123.1234	18/07/96	ST_MAN	8000	(null)	10	50
8 Adam	Fripp	AFRIPP	650.123.2234	10/04/97	ST_MAN	8200	(null)	10	50
9 Payam	Kaufling	PKAUFLIN	650.123.3234	01/05/95	ST_MAN	7900	(null)	10	50
10 Shanta	Vollman	SVOLLMAN	650.123.4234	10/10/97	ST_MAN	6500	(null)	10	50
11									

# Operador IS NULL

O operador IS NULL é usado junto com o WHERE e permite filtrar valores nulos (que são diferentes de zero e diferentes de vazio).

A sintaxe desse operador é mostrada ao lado.

Vamos agora ver alguns exemplos.

```
SELECT coluna1, coluna2, ...
FROM tabela
WHERE colunaN IS NULL;
```

```
SELECT coluna1, coluna2, ...
FROM tabela
WHERE colunaN IS NOT NULL;
```

# Operador IS NULL

No exemplo ao lado, filtramos a tabela EMPLOYEES para mostrar todos os funcionários que possuem um percentual de comissão nulo. Utilizamos para isso o operador NULL.

```
-- a) Quais funcionários têm o percentual de comissão igual a (null)?
SELECT * FROM employees
WHERE commission_pct IS NULL;

-- b) Quais funcionários têm o percentual de comissão diferente de null.
```

Resultado da Consulta x

SQL | 50 linhas extraídas em 0,012 segundos

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPART
1	100	Steven	King	SKING	515.123.4567	17/06/87	AD_PRES	24000	(null)	(null)	
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	21/09/89	AD_VP	17000	(null)	100	
3	102	Lex	De Haan	LDEHAAN	515.123.4569	13/01/93	AD_VP	17000	(null)	100	
4	103	Alexander	Hunold	AHUNOLD	590.423.4567	03/01/90	IT_PROG	9000	(null)	102	
5	104	Bruce	Ernst	BERNST	590.423.4568	21/05/91	IT_PROG	6000	(null)	103	
6	105	David	Austin	DAUSTIN	590.423.4569	25/06/97	IT_PROG	4800	(null)	103	
7	106	Valli	Pataballa	VPATABAL	590.423.4560	05/02/98	IT_PROG	4800	(null)	103	
8	107	Diana	Lorentz	DLORENTZ	590.423.5567	07/02/99	IT_PROG	4200	(null)	103	
9	108	Nancy	Greenberg	NGREENBE	515.124.4569	17/08/94	FI_MGR	12000	(null)	101	
10	109	Daniel	Faviet	DFAVIET	515.124.4169	16/08/94	FI_ACCOUNT	9000	(null)	108	
11	110	Pat	Chen	PCHE	515.124.4560	08/08/95	FI_ACCOUNT	6000	(null)	100	

# Regras de precedência

Até agora vimos diversos operadores aplicados juntos ao WHERE, e a dúvida que pode surgir é: quando temos mais de um sendo usado ao mesmo tempo, quem “ganha” de quem? Quem será executado primeiro?

A seguir, listamos a ordem de prioridade de cada um dos operadores:

1. Condições de comparação
2. IS NULL, LIKE, IN
3. BETWEEN
4. NOT
5. AND
6. OR

O que mais gera dúvida é quando usamos múltiplos AND e OR no mesmo filtro, que é o tipo de combinação mais comum.

Vejamos a seguir um exemplo.

# Regras de precedência

No exemplo abaixo, temos 3 testes lógicos. Qual você acha que será o resultado?

- (1) Será que primeiro ele vai ver quais são os funcionários de job\_id = 'IT\_PROG' ou 'ST\_MAN' e só depois vai trazer quem tem salário acima de 5000 dentre esses dois?
- (2) Ou primeiro ele vai trazer os funcionários 'ST\_MAN' com salário acima de 5000 junto também de todos os funcionários 'IT\_PROG'?

```
SELECT first_name, job_id, salary
FROM employees
WHERE job_id = 'IT_PROG' OR job_id = 'ST_MAN' AND salary > 5000;
```

# Regras de precedência

Se você respondeu a opção (2) então acertou. De acordo com as regras de precedência, o operador AND vem primeiro que o operador OR. Portanto, no exemplo anterior, teremos a seguinte ordem:

```
SELECT first_name, job_id, salary
FROM employees
WHERE job_id = 'IT_PROG' OR job_id = 'ST_MAN' AND salary > 5000;
```

Primeiro, é feito o filtro de funcionários com `JOB_ID='ST_MAN'` AND `salary > 5000`

Em seguida, é feito o filtro de funcionários com `JOB_ID='IT_PROG'` OR o resultado anterior.

Observe, portanto, que no resultado temos: funcionários IT\_PROG OU funcionários ST\_MAN com salário acima de 5000.

Os funcionários que têm o salário abaixo de 5000 são necessariamente IT\_PROG.

#	FIRST_NAME	JOB_ID	SALARY
1	Alexander	IT_PROG	9000
2	Bruce	IT_PROG	6000
3	David	IT_PROG	4800
4	Valli	IT_PROG	4800
5	Diana	IT_PROG	4200
6	Matthew	ST_MAN	8000
7	Adam	ST_MAN	8200
8	Payam	ST_MAN	7900
9	Shanta	ST_MAN	6500
10	Kevin	ST_MAN	5800

# Regras de precedência

E se quiséssemos os funcionários IT\_PROG ou ST\_MAN, mas apenas os que tem salário acima de 5000 (agora, considerando ambos). Como ficaria?

Então teríamos que usar parênteses para especificar o que queremos que seja feito primeiro.

Isso porque nesse novo caso precisamos que o OR seja executado primeiro, e só depois o AND. O código da solução é mostrado ao lado.

Dessa vez, o filtro de salário só será aplicado depois que a gente tiver apenas os funcionários IT\_PROG ou ST\_MAN.

Observe que a consulta trás apenas os funcionários IT\_PROG ou ST\_MAN, mas que que necessariamente têm um salário acima de 5000.

```
SELECT first_name, job_id, salary
FROM employees
WHERE (job_id = 'IT_PROG' OR job_id = 'ST_MAN') AND salary > 5000;
```

The screenshot shows a SQL query results window titled "Resultado da Consulta". The query is:

```
SELECT first_name, job_id, salary
FROM employees
WHERE (job_id = 'IT_PROG' OR job_id = 'ST_MAN') AND salary > 5000;
```

The results table has columns: FIRST\_NAME, JOB\_ID, and SALARY. The data is:

	FIRST_NAME	JOB_ID	SALARY
1	Alexander	IT_PROG	9000
2	Bruce	IT_PROG	6000
3	Matthew	ST_MAN	8000
4	Adam	ST_MAN	8200
5	Payam	ST_MAN	7900
6	Shanta	ST_MAN	6500
7	Kevin	ST_MAN	5800

SQL | Todas as Linhas Extraídas: 7 em 0,021 segundos

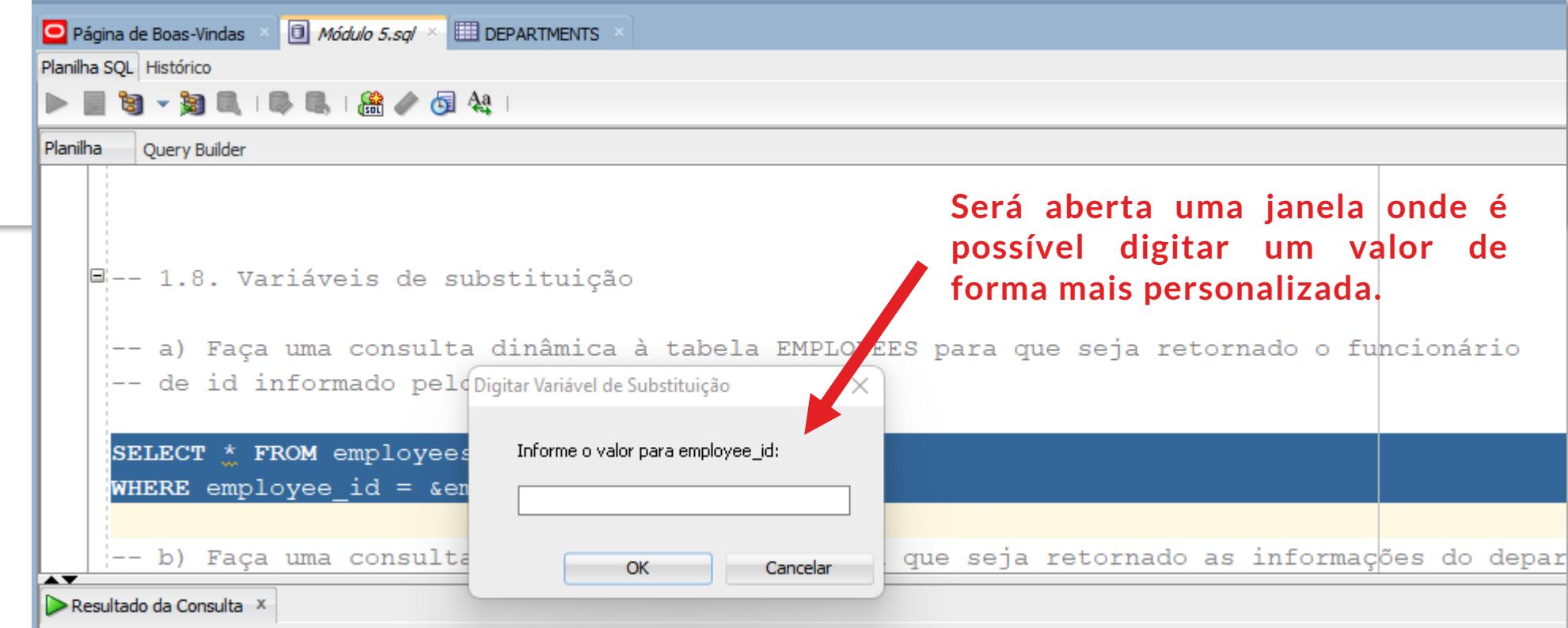
# Variáveis de substituição

Variáveis de substituição permitem que a gente interaja com o Oracle com algum input do usuário, abrindo uma caixinha onde podemos digitar um valor. Por exemplo, imagine que você quer retornar o funcionário de um ID que você quer informar de maneira dinâmica.

Para isso, você vai usar o & da seguinte forma como é mostrado no print abaixo.

```
-- a) Faça uma consulta dinâmica à tabela EMPLOYEES para que seja retornado o funcionário
-- de id informado pelo usuário.
```

```
SELECT * FROM employees
WHERE employee_id = &employee_id;
```



# Variáveis de substituição

Caso o seu filtro seja sobre uma coluna de texto ou de data, é necessário colocar entre aspas simples.

```
SELECT * FROM departments  
WHERE department_name = '&department_name';
```

# SQL IMPRESSIONADOR

Apostila completa