SQL para Usuários de Python

Asimov Academy

ASIMOV

Conteúdo

01. SQL aplicado além da sintaxe

Há centenas de tutoriais de SQL online para ensinar a sintaxe SQL. Este curso é sobre construir modelos em torno do que o SQL e entender como nos ajuda no dia a dia orientado a soluções.

Spoiler de tópicos

1. Confusão sobre o que é um banco de dados
2. Como inserir dados
3. SQL / Python
4. Como coletar dados
5. Keywords
6. Como funcionam as Joins
7. O que fazer sobre erros

O que é SQL?

SQL, que significa “Structured Query Language” (Linguagem de Consulta Estruturada), é uma linguagem padrão para acessar e manipular bancos de dados.

Características Principais

- **Linguagem de Domínio Específico**: Projetada especificamente para gerenciar dados em um Sistema de Gerenciamento de Banco de Dados Relacional (SGBDR).
- **Leitura e Manipulação de Dados**: Permite consultar, inserir, atualizar e deletar dados.
- **Gestão de Banco de Dados**: Utilizada para criar e modificar a estrutura de bancos de dados e controlar o acesso aos dados.

Como Funciona o SQL?

- **Comandos SQL**: Consistem em declarações ou queries que são usadas para realizar operações específicas. Exemplos incluem `SELECT`, `INSERT`, `UPDATE`, `DELETE`, etc.
- **Interatividade com Bancos de Dados**: O SQL envia comandos para o banco de dados, que executa esses comandos e retorna resultados.
- **Padronização**: Apesar de haver variações entre sistemas de bancos de dados, a maioria segue um padrão SQL comum. Como se fossem os dialetos espalhados pelo Brasil.

Usos Comuns do SQL

- **Consulta de Dados**: Recuperar dados específicos de um banco de dados.
- **Atualização de Registros**: Modificar dados existentes.
- **Inserção de Dados**: Adicionar novos dados.
- **Deleção de Dados**: Remover dados.
- **Criação e Modificação de Estruturas**: Criar e modificar tabelas e outros objetos de banco de dados.

Compreender essas ideias abstratas vem naturalmente para engenheiros e analistas que trabalham com código todos os dias. Pessoas de negócios tendem a ter dificuldades para entender o que o código está fazendo. Eles não conseguem conceituar como os dados estão sendo transformados. Eles veem apenas entrada e saída.

02. O que é um database

Pessoas entendem uma planilha do Excel. Ela é visual, os dados estão ali, e você pode realizar cálculos bem ao lado dos dados. Para muitos cenários, esta é a ferramenta certa a ser usada, inclusive é a ferramenta correta em grande parte do dia-a-dia dos brasileiros nas grandes empresas.

	A	B	C	D	
1	Investimentos e Prazos				
2					
3	Data de Resgate	Valor Atual	Banco	Tipo	
4	Diária	R$ 10.173,39	Inter	CDB	L
5	3/10/23	R$ 4.101,74	Inter	LCI/LCA	9
6	3/11/23	R$ 3.288,41	Inter	LCI/LCA	9
7	31/05/24	R$ 3.108,19	Inter	LCI/LCA	A
8	28/04/25	R$ 1.051,02	Inter	LCI/LCA	B
9	28/04/25	R$ 1.051,02	Inter	LCI/LCA	B
10	28/04/25	R$ 1.051,02	Inter	LCI/LCA	B
11	28/01/25	R$ 3.044,19	Inter	LCI/LCA	B
12	19/04/24	R$ 10.595,00	XP	CDB	M
13	24/04/24	R$ 9.526,00	XP	CDB	M
14	21/03/24	R$ 1.060,00	XP	CDB	P

Figure 1: Representação de uma Tabela Excel

Por que usar uma Database (base de dados) pode ser desafiador?

Para acessar uma database, teremos que utilizar uma linguagem de programação para acessá-la, geralmente SQL. Há uma curva de aprendizado a ser superada.

Outra dor que é recorrente é que o acesso, assim como a estruturação dos dados é bem diferente do que fomos acostumados com planilhas e dados comuns de empresas. Existem credenciais a serem requeridas, maneiras de requisitá-la, maneiras de guarda-la e uma visualização bem diferente dos dados que serão consumidos

Vocabulário básico de Databases - e como isso afeta o Python

Primeiro de tudo, deixaremos claro ao longo desse material que `database == base de dados`. Novos termos e conceitos sempre são complicados de serem absorvidos e faz parte do processo de

aprendizado. Nesse material, assim como no curso completo, utilizaremos bastante dos termos a seguir, assim como é utilizado no âmbito profissional e internacional.

Table (Tabela)

SQL: Uma tabela é onde os dados são armazenados em linhas e colunas.

Pandas: Equivalente a um DataFrame no Pandas. Um DataFrame é uma estrutura de dados bidimensional, com eixos rotulados (linhas e colunas).

Field (Campo)

SQL: Campo refere-se a uma coluna em uma tabela.

Pandas: No Pandas, isso é simplesmente uma coluna do DataFrame. Cada coluna tem um nome e armazena um tipo específico de dado.

Record (Registro)

SQL: Um registro é uma linha individual em uma tabela.

Pandas: Um registro é representado por uma linha no DataFrame. Cada linha pode ser acessada por índices ou condições.

Primary Key (Chave Primária)

SQL: Uma chave primária é um identificador único para cada registro em uma tabela.

Pandas: O conceito mais próximo seria o índice do DataFrame, que identifica de maneira única cada linha. Embora o Pandas não imponha a unicidade dos índices como SQL, é uma prática comum usar índices únicos.

Foreign Key (Chave Estrangeira)

SQL: Chave estrangeira é um campo que estabelece um link para a chave primária de outra tabela.

Pandas: Essa relação é manipulada através de operações como merge ou join, que combinam DataFrames com base em colunas comuns, semelhantes às chaves estrangeiras.

Schema (Esquema)

SQL: Um esquema define a estrutura de uma tabela (campos, tipos de dados, etc.).

Pandas: O equivalente seria o dtypes de um DataFrame, que mostra os tipos de dados de cada coluna. No entanto, o Pandas é mais flexível e não exige a definição de um esquema tão estrito quanto SQL.

Resumindo

Bancos de dados são muito semelhantes a ter várias planilhas, mas os termos são um pouco diferentes.

03. Paralelos entre Python e SQL

Python

```python
import pandas as pd

# Table
# Criando um DataFrame que é equivalente a uma tabela SQL
clientes = pd.DataFrame({
    'ID': [],
    'Nome': [],
    'Endereco': []
})

# Field
# Selecionando colunas específicas do DataFrame
colunas_selecionadas = clientes[['Nome', 'Endereco']]

# Record
# Adicionando um registro ao DataFrame
clientes = clientes.append({'ID': 1, 'Nome': 'João Silva', 'Endereco': 'Rua das Flores'},
↪  ignore_index=True)

# Primary Key
# Definindo uma coluna como índice (similar a uma chave primária)
clientes = clientes.set_index('ID')

# Foreign Key
# Supondo que exista um DataFrame 'pedidos' com uma coluna 'ClienteID'
pedidos = pd.merge(pedidos, clientes, left_on='ClienteID', right_on='ID')

# Schema
# Visualizando os tipos de dados de cada coluna do DataFrame
clientes.dtypes
```

SQL

```sql
-- Não é necessário compreender de fato o material, é apenas um comparativo paralelo

-- Table
CREATE TABLE Clientes (
    ID INT,
    Nome VARCHAR(100),
    Endereco VARCHAR(100)
);

-- Field
SELECT Nome, Endereco FROM Clientes;

-- Record
INSERT INTO Clientes (ID, Nome, Endereco) VALUES (1, 'João Silva', 'Rua das Flores');
```

```sql
-- Primary Key
ALTER TABLE Clientes ADD PRIMARY KEY (ID);

-- Foreign Key
SELECT * FROM Pedidos JOIN Clientes ON Pedidos.ClienteID = Clientes.ID;

-- Schema
DESCRIBE Clientes;
```

04. Como iremos praticar: PostgreSQL e PGAdmin

O conceito do curso envolve aplicar SQL em Python. Para isso temos uma gama de ferramentas como SQLite3, SQLAlchemy, etc. Porém no primeiro momento é importante que possamos consolidar o que aprendemos sobre a linguagem, de alguma maneira. Para isso utilizaremos o PostgreSQL no PGAdmin para que possamos testar os conceitos.

Para isso temos que:

1. Baixar o PostgreSQL: pesquisar “PostgreSQL download” no Google e baixar a versão mais atualizada para o seu OS
2. Baixar o PGAdmin: pesquisar “PGAdmin download” no Google e baixar a versão mais atualizada para o seu OS
3. Importar tabelas para o PGAdmin
    - Especificar tipos de colunas para cada coluna de cada tabela (`assets/data_specs.sql`)

Em um segundo momento do curso, aplicaremos SQL em Python, escrevendo os códigos no VS Code (ou na IDE de preferência), nos especializando em transformar o conhecimento inicial em soluções integradas.

05. Troubleshooting PGAdmin

Para que serve este capítulo

O capítulo a seguir é para aqueles que não conseguiram iniciar o PGAdmin por diversas maneiras. Existem diversos motivos pelos quais o PGAdmin pode não se inicializar corretamente e geralmente não são culpa de quem os instalou, afinal o software é conhecido por ser um excelente geranciador de bases de dados, porém tem uma pobre instalação que não corrige ou informa da maneira mais eficiente os erros.

Figure 2: Encontrando erros no PGAdmin

Encorajamos os alunos deste material a pesquisarem os erros quando surgirem e preferencialmente na língua inglesa, pois encontraram mais material. A partir deste cenário gerei este capítulo para tratarmos dos erros mais comuns que encontrei no PGAdmin 4, os porquês e como solucionar-los.

Erros e motivações

- **pgAdmin Runtime Environment**: Falha de conexão entre o servidor e o software logo após a instalação.

Este problema está associado às permissões do seu usuário em relação ao programa e potencialmente a estabelecer a conexão entre o Path e o PostegreSQL instalado posteriormente na máquina. Note que ao utilizar PostgreSQL em mais de um usuário no mesmo computador podem haver problemas de conexão, pois ambos estarão tentando conectar no mesmo servidor pelo mesmo ponto de acesso.

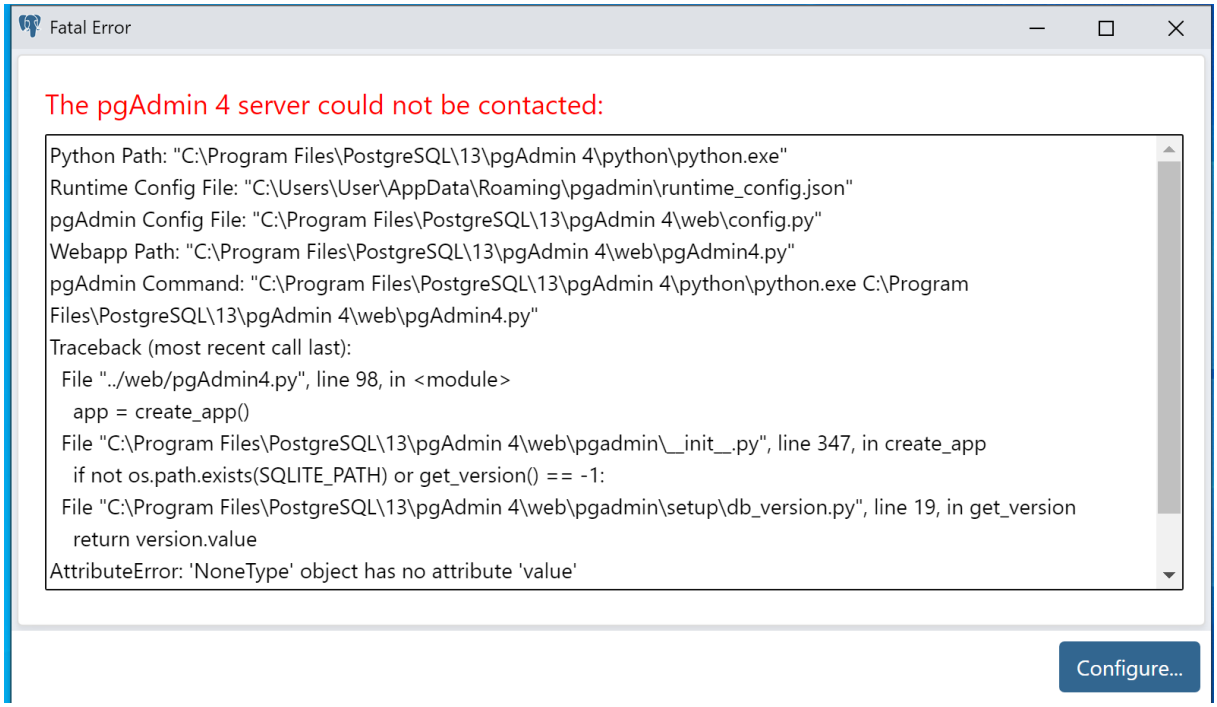

Figure 3: Erro de Runtime Environment do PGAdmin

- **Servers não aparecem no software**: Ao inicializar o software, os servidores na aba da direita não se inicializam.

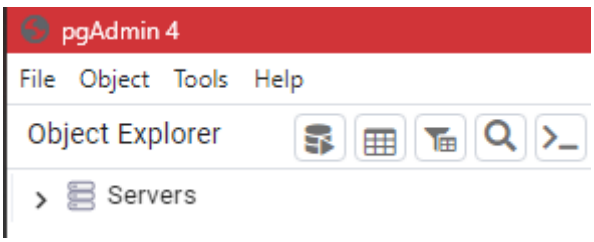

Figure 4: Erro de servidores não aparecendo na interface

Neste caso o problema é que o processo de geração de servidores do PostgreSQL não está rodando no background do seu computador, impossibilitando o PGAdmin4 a conectar-se com estes servidores. Infelizmente nesse caso o software não gera mensagem de erro, apenas não dispõe da mensagem de carregamento, como se os botões responsáveis não fossem responsivos.

- **Carregamento eterno**: Na tela de inicialização do PGAdmin4, o processo nunca inicializa o programa, apenas carrega.

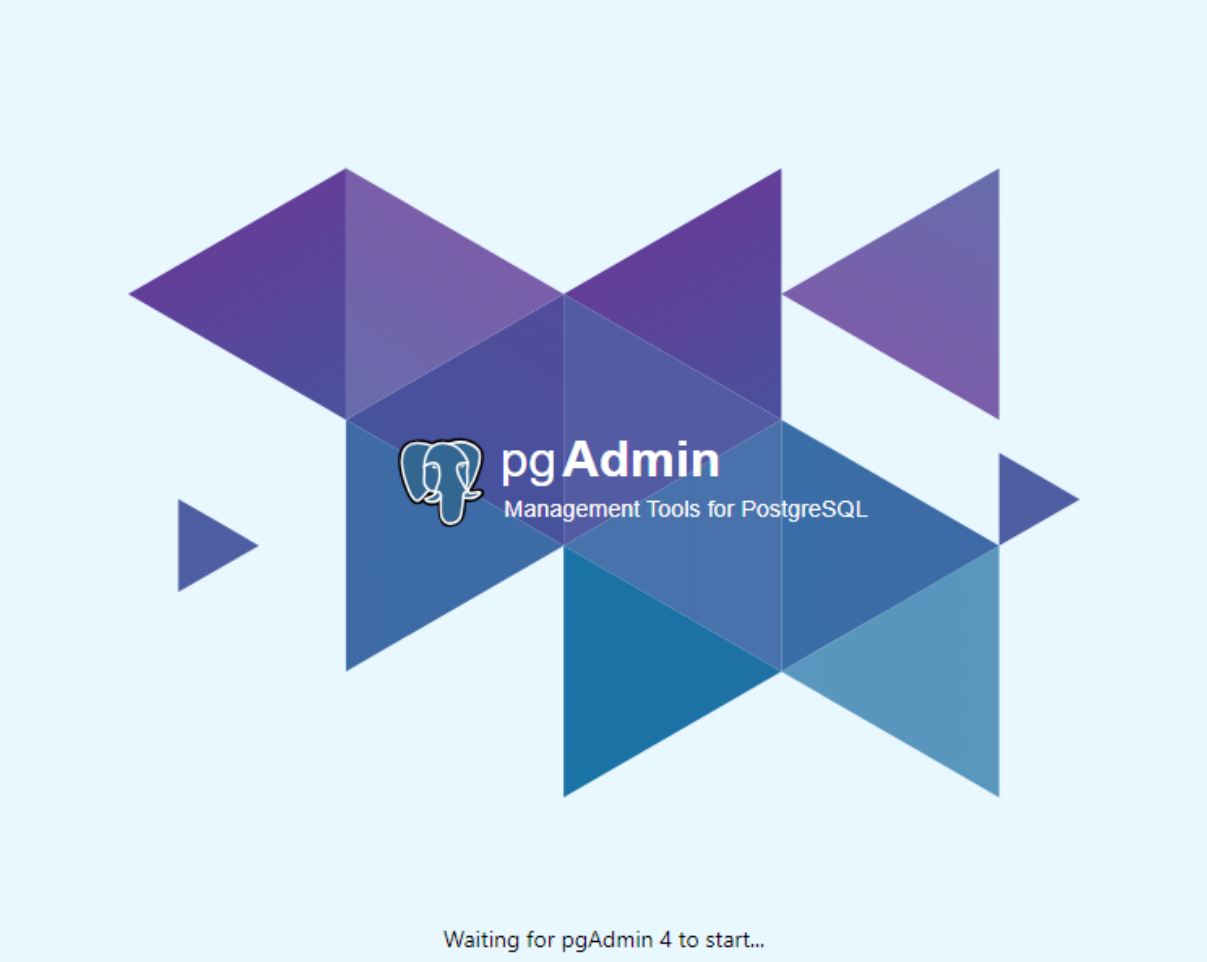

Figure 5: Tela de carregamento do PGAdmin, onde fica sem entrar no programa

Soluções e Troubleshooting

1. Tente inicializar o PGAdmin como administrador, para isso clique com o botão direito no programa e selecione “executar como administrador”.

2. Acesse o gerenciador de tarefas (no Windows pode utilizar o atalho CTRL+SHIFT+ESC), vá na aba Usuários acima e desconecte os outros usuários, para caso estejam conectados no PostgreSQL.
3. Se as soluções acima não funcionarem, delete a pasta no diretório `C:\Usuários\SeuNomeDeUsuario\AppData\Roaming\pasta\pgAdmin`. Note que a pasta `AppData` é uma pasta oculta. No Windows para vê-la no gerenciador de arquivos você terá que ir em:

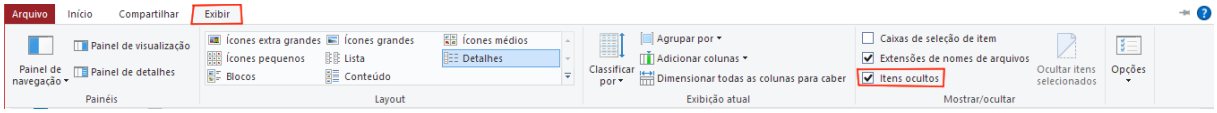

Figure 6: Habilitando pastas ocultas no Windows Explorer

4. Caso as soluções acima não funcionem, desinstale o PostgreSQL e PGAdmin pelo gerenciador de programas e lembre-se de apagar as pastas referentes dos arquivos de programas.

06. Inserindo os dados no PGAdmin

Primeiro de tudo, certifique-se que tem os dados baixados do material dos cursos. Com o material do curso baixado, os dados a serem importados devem ser encontrados na pasta `assets/data`. Note que há apenas um arquivo .SQL que vamos seguir e adicionar no PGAdmin. Para isso, siga o passo-a-passo abaixo:

1. Conecte-se ao Banco de Dados Conecte-se ao banco de dados no qual deseja inserir os dados.
2. Crie a tabela Devemos especificar o tipo de cada coluna antes.
3. Navegue até a Tabela Vá até a tabela específica onde deseja inserir os dados.
4. Mapeie as Colunas Mapeie as colunas do arquivo CSV para as colunas correspondentes na tabela do banco de dados.
5. Configure as Opções de Importação Configure outras opções, como delimitador de colunas, formato de data, etc.

07. Acessando dados

O objetivo de código SQL no geral é acessar, alterar e gerar dados de maneira otimizada.

Para isso iremos testar uma série de comandos:

Consumo de uma table

Tabela: consumidores

Nome	Estado	Idade	Vegetariano
Rodrigo	RS	22	Não
Luana	SC	42	Não
Carolina	SP	17	Sim
Jorge	ES	30	Sim

Existes `keywords` no SQL que nos permitem consumir dados de tabelas de maneiras diversas:

- `SELECT`: extrai dados de um banco de dados (`SELECT DISTINCT` para exclusivo)

```
SELECT Nome, Idade, Vegetariano FROM consumidores
```

Filtros de pesquisa

- `WHERE`: filtra os records, pode ser usado no select, delete, update [...]

```
SELECT Nome, Idade, Vegetariano FROM consumidores
WHERE Vegetariano = 'Não'
```

- `ORDER BY`: ASC ou DESC, define a ordem do retorno dos records

```
SELECT Nome, Idade, Vegetariano FROM consumidores
WHERE Vegetariano = 'Não'
ORDER BY Nome
```

- `GROUP BY`: agrupa as informações dada uma coluna específica, idêntico ao groupby do `Pandas`.

```
SELECT COUNT(Ids_consumidor), Estado
FROM consumidores
GROUP BY Estado
```

Note no exemplo acima que temos funções no SQL assim como no Python, então o `COUNT()` nos retorna a contagem de uma coluna.

Dica para exploração de dados

Se você estiver explorando os dados para descobrir o que eles contêm, sugerimos limitar a quantidade de registros recebidos para melhorar a velocidade da consulta. Você faz isso em uma instrução LIMIT que vem no final da consulta e especifica o número de linhas que deseja ver.

```
SELECT * FROM tabela
LIMIT 5
```

08. Desafios de Solicitações ao Banco e Condicionais

Emulador

Como configurado anteriormente, utilizaremos o PGAdmin para consumir os dados que inserimos anteriormente.

Siga o passo a passo da aula para entender como funciona o emulador.

Para começar vamos trabalhar apenas com a Table `Customers` que registra uma série de colunas:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Rodrigo	João	Protásio Alves 1442	Porto Alegre	789123	Brasil
2	Pedro	Carlos	Carlo Gomes 123	Porto Alegre	123456	Brasil

Questionário 1

1. Como você seleciona todas as colunas da tabela `Customers`?
2. Como mostrar apenas o `CustomerName` e `City` da tabela `Customers`?
3. Como encontrar clientes na tabela `Customers` que estão localizados no país `Germany`?
4. Como listar clientes da tabela `Customers` com `CustomerID` menor ou igual a 5?
5. Como selecionar clientes da `Germany` que também estão na cidade de `Berlin`?
6. Quais são os títulos de contato únicos dos clientes da `Germany`?
7. Retorne apenas os primeiros 5 registros da tabela `Customers`.

09. Resolução Questionário 1

Exercício 1

```
SELECT * FROM public."Customers";
```

- **O que faz**: Esta consulta seleciona todas as colunas (`*`) da tabela `Customers`.
- **Como funciona**: O `*` é um coringa que representa todas as colunas disponíveis na tabela. Portanto, essa consulta retorna todos os registros (linhas) e todas as suas respectivas colunas (campos) da tabela `Customers`.
- **Uso típico**: Usado quando você quer visualizar todos os dados de uma tabela sem restrição.

Exercício 2

```
SELECT "CustomerName", "City" FROM public."Customers";
```

- **O que faz**: Seleciona apenas as colunas `CustomerName` e `City` da tabela `Customers`.
- **Como funciona**: Ao contrário da primeira consulta que usa `*`, esta especifica exatamente quais colunas queremos ver. Apenas os dados dessas duas colunas (nome do cliente e cidade) serão retornados para todos os registros da tabela.
- **Uso típico**: Ideal quando você precisa de informações específicas de uma tabela, sem necessidade de ver todas as colunas.

Exercício 3

```
SELECT * FROM public."Customers" WHERE "Country" = 'Germany';
```

- **O que faz**: Seleciona todos os registros da tabela `Customers` onde a coluna `City` é igual a 'Berlin'.
- **Como funciona**: A cláusula `WHERE` filtra os registros. Esta consulta retorna todos os campos (`*`) dos registros cuja cidade é especificamente 'Berlin'.
- **Uso típico**: Usado para filtrar registros que atendem a uma condição específica.

Exercício 4

```
SELECT * FROM public."Customers" WHERE "CustomerID" <= 5;
```

- **O que faz**: Seleciona todos os registros da tabela `Customers` onde o `CustomerID` é menor ou igual a 5.

- **Como funciona**: A cláusula `WHERE` é utilizada novamente para filtrar, mas desta vez com uma condição numérica. Retorna todos os campos dos clientes cujo `CustomerID` é 5 ou menos.
- **Uso típico**: Útil para filtrar registros com base em valores numéricos, como IDs, preços, quantidades, etc.

Exercício 5

```
SELECT * FROM public."Customers" WHERE "Country" = 'Germany' AND
"City" = 'Berlin';
```

- **O que faz**: Seleciona todos os registros da tabela `Customers` onde o `Country` é 'Germany' e a `City` é 'Berlin'.
- **Como funciona**: Utiliza a cláusula `WHERE` com um operador lógico `AND` para combinar duas condições. Apenas os registros que satisfazem ambas as condições (localizados na Alemanha e na cidade de Berlim) serão retornados.
- **Uso típico**: Ideal para situações em que você precisa de uma filtragem mais específica, combinando múltiplas condições.

Exercício 6

```
SELECT DISTINCT "City" FROM public."Customers";
```

- **O que faz**: Seleciona todas as cidades distintas (únicas) da tabela `Customers`.
- **Como funciona**: A palavra-chave `DISTINCT` é usada para remover duplicatas na coluna especificada (`City`). Portanto, esta consulta retorna uma lista de cidades únicas presentes na tabela, sem repetições.
- **Uso típico**: Usado quando você precisa saber quais são os valores únicos em uma coluna, como uma lista de categorias, cidades, países, etc.

Exercício 7

```
SELECT * FROM public."Customers" LIMIT 5;
```

- **O que faz**: Esta consulta seleciona os primeiros 5 registros da tabela `Customers`.
- **Como funciona**:
- `SELECT *`: Seleciona todas as colunas da tabela `Customers`.
- `FROM Customers`: Indica que a consulta será realizada na tabela `Customers`.

- `LIMIT 5`: Limita o número de registros retornados pela consulta a 5. Isso significa que apenas os primeiros 5 registros da tabela serão exibidos.
- **Uso típico**: Útil quando você precisa de uma amostra dos dados ou quer limitar o número de registros retornados por uma consulta, seja para melhorar a performance ou para evitar sobrecarregar a interface do usuário com muitos dados.

10. Novas Keywords de filtro

Anteriormente vimos algumas `keywords` que nos permitiam filtrar a nossa consulta SQL como `WHERE`. Veremos que existem outras maneiras mais avançadas de realizar os filtros e que vão enriquecer as nossas consultas.

Filtrando com Condições Específicas

Em SQL, você pode filtrar dados com base em condições específicas usando a cláusula `WHERE`. Esta cláusula permite especificar critérios para escolher quais linhas (registros) você quer ver nos resultados.

- **Keyword**: `LIKE`
  - Usada para filtrar dados com base em padrões de texto.
  - Exemplo: `SELECT * FROM Customers WHERE CustomerName LIKE 'A%';`
  - Este comando seleciona todos os clientes cujos nomes começam com 'A'.
  - Paralelo em Python: similar a usar expressões regulares ou checagem de strings com `startswith`.

Operadores Lógicos

Operadores lógicos são usados para refinar as consultas SQL, combinando ou modificando condições.

- **Keywords**: `AND`, `OR`
  - `AND`: Usado para garantir que duas ou mais condições sejam verdadeiras.
    - Exemplo: `SELECT * FROM Customers WHERE Country = 'Germany' AND City = 'Berlin';`
    - Seleciona clientes que estão na Alemanha e em Berlim.
    - Paralelo em Python: similar ao uso de `and` em condições if.
  - `OR`: Usado quando qualquer uma das condições deve ser verdadeira.
    - Exemplo: `SELECT * FROM Customers WHERE Country = 'Germany' OR Country = 'France';`
    - Seleciona clientes que estão na Alemanha ou na França.
    - Paralelo em Python: similar ao uso de `or` em condições if.

Agrupar Resultados e Usar Having

SQL permite agrupar registros e aplicar funções agregadas, como contar, somar, etc.

- **Keyword**: `GROUP BY`
  - Usado para agrupar registros em subconjuntos.
  - Exemplo: `SELECT Country, COUNT(*) FROM Customers GROUP BY Country;`
  - Este comando conta quantos clientes existem em cada país.
  - Paralelo em Python: similar a agrupar dados em um DataFrame do Pandas usando `groupby`.
- **Keyword**: `HAVING`
  - Usada para filtrar grupos criados pelo `GROUP BY`.
  - Exemplo: `SELECT Country, COUNT(*) FROM Customers GROUP BY Country HAVING COUNT(*) > 5;`
  - Seleciona países que têm mais de 5 clientes.
  - Paralelo em Python: similar a aplicar filtros em grupos no Pandas após `groupby`.

Estes conceitos são a base para explorar e analisar grandes conjuntos de dados em SQL. Entender como funcionam e como são semelhantes às operações em Python ajuda a criar uma fundação sólida para aprendizado e aplicação prática.

11. Desafios AND, OR, WHERE

Questionário 2

1. Selecione todos os clientes que estão localizados em “Rio de Janeiro” ou “São Paulo”.
2. Selecione o nome do cliente, o nome do contato e a cidade para todos os clientes que não estão no Brasil.
3. Liste os países com pelo menos um cliente e ordene alfabeticamente.
4. Selecione o nome do cliente e a cidade para todos os clientes cujo nome do contato é “John Doe”.

12. Resolução Questionário 2

Exercício 1

```
SELECT * FROM public."Customers" WHERE "City" = 'Rio de Janeiro' OR
"City" = 'So Paulo';
```

- **Pergunta**: Selecione todos os clientes que estão localizados em “Rio de Janeiro” ou “São Paulo”.
- **Como funciona**: Utiliza a cláusula `WHERE` para filtrar os resultados com base nas cidades do Rio de Janeiro ou São Paulo.
- **Uso típico**: Útil para identificar clientes em cidades específicas.

Exercício 2

```
SELECT "CustomerName", "ContactName", "City" FROM public."Customers"
WHERE "Country" <> 'Brazil';
```

- **Pergunta**: Selecione o nome do cliente, o nome do contato e a cidade para todos os clientes que não estão no Brasil.
- **Como funciona**: Utiliza a cláusula `WHERE` para filtrar os resultados onde o país não é o Brasil.
- **Uso típico**: Útil para obter informações específicas de clientes fora do Brasil.

Exercício 3

```
SELECT DISTINCT "Country" FROM public."Customers" ORDER BY "Coun-
try";
```

- **Pergunta**: Liste os países com pelo menos um cliente e ordene alfabeticamente.
- **Como funciona**: Utiliza `DISTINCT` para obter valores únicos de países e `ORDER BY` para classificá-los alfabeticamente.
- **Uso típico**: Útil para identificar os países presentes na base de dados de clientes.

Exercício 4

```
SELECT "CustomerName", "City" FROM public."Customers" WHERE "Contact-
Name" = 'John Doe';
```

- **Pergunta**: Selecione o nome do cliente e a cidade para todos os clientes cujo nome do contato é "John Doe".
- **Como funciona**: Utiliza a cláusula `WHERE` para filtrar os resultados onde o nome do contato é "John Doe".
- **Uso típico**: Útil para identificar clientes específicos com um contato chamado "John Doe".

13. Um pouco mais sobre LIKE

A keyword LIKE é usada em consultas SQL para realizar correspondências parciais em strings. Seguem exemplos:

1. Correspondência Exata

```sql
SELECT * FROM tabela WHERE coluna LIKE 'exemplo';
```

Retorna todas as linhas onde a coluna tem exatamente o valor 'exemplo'.

2. Correspondência Parcial no Início

```sql
SELECT * FROM tabela WHERE coluna LIKE 'prefixo%';
```

Retorna todas as linhas onde a coluna começa com 'prefixo'.

3. Correspondência Parcial no Final

```sql
SELECT * FROM tabela WHERE coluna LIKE '%sufixo';
```

Retorna todas as linhas onde a coluna termina com 'sufixo'.

4. Correspondência em Qualquer Lugar

```sql
SELECT * FROM tabela WHERE coluna LIKE '%conteudo%';
```

Retorna todas as linhas onde a coluna contém 'conteudo' em qualquer lugar.

5. Uso de Caracteres Coringa

```sql
SELECT * FROM tabela WHERE coluna LIKE '_xemplo';
```

Retorna todas as linhas onde o valor da coluna termina com "xemplo" e o primeiro é desconhecido.

6. Correspondência usando caracteres de escape

```sql
SELECT * FROM tabela WHERE coluna LIKE '20\%';
```

Retorna todas as linhas onde a coluna contém '20%'.

7. Correspondência com múltiplos caracteres coringa

```sql
SELECT * FROM tabela WHERE coluna LIKE 'abcd%efg';
```

Retorna todas as linhas onde a coluna começa com ‘abcd’ e termina com ‘efg’, podendo ter qualquer conteúdo no meio.

8. Correspondência com conjunto de caracteres

```sql
SELECT * FROM tabela WHERE coluna LIKE '[a-c]%';
```

Retorna todas as linhas onde a primeira letra da coluna é ‘a’, ‘b’ ou ‘c’.

14. Desafios LIKE, HAVING, GROUP BY

Questionário 3

1. Quais países têm mais de 5 clientes e quantos clientes eles têm?
2. Conte quantos clientes existem em cada país.
3. Selecione todos os clientes que têm “A” no início do nome do cliente ou “B” no início do nome do contato.
4. Liste o número de clientes em cada cidade que tenha mais de 2 clientes.
5. Selecione todos os clientes que estão localizados em “Brazil” e cujo código postal não está vazio.

15. Resolução Questionário 3

Exercício 1

```
SELECT "Country", COUNT("CustomerID") FROM public."Customers" GROUP
BY "Country" HAVING COUNT("CustomerID") > 5;
```

- **Pergunta**: Quais países têm mais de 5 clientes e quantos clientes eles têm?
- **Como funciona**: A cláusula `GROUP BY` é empregada para agrupar os resultados por país. A cláusula `HAVING` filtra os grupos com mais de 5 clientes, e `COUNT(CustomerID)` conta o número de clientes em cada grupo.
- **Uso típico**: Útil para analisar a distribuição de clientes por país, identificando aqueles com um número significativo de clientes.

Exercício 2

```
SELECT "Country", COUNT("CustomerID") FROM public."Customers" GROUP
BY "Country";
```

- **Pergunta**: Quantos clientes existem em cada país?
- **Como funciona**: A cláusula `GROUP BY` agrupa os resultados por país, e `COUNT(CustomerID)` conta o número de clientes em cada grupo.
- **Uso típico**: Útil para obter uma contagem de clientes por país, oferecendo insights sobre a presença da clientela em diferentes regiões.

Exercício 3

```
SELECT * FROM public."Customers" WHERE "CustomerName" LIKE 'A%' OR
"ContactName" LIKE 'B%';
```

- **Pergunta**: Quais clientes têm nomes que começam com “A” ou cujo nome de contato começa com “B”?
- **Como funciona**: O operador `LIKE` com `%` é usado para buscar registros onde o nome do cliente começa com “A” ou o nome do contato começa com “B”.
- **Uso típico**: Útil para segmentar clientes com base nos primeiros caracteres de seus nomes.

Exercício 4

```
SELECT "City", COUNT("CustomerID") FROM public."Customers" GROUP BY
"City" HAVING COUNT("CustomerID") > 2;
```

- **Pergunta**: Quantos clientes existem em cada cidade que tem mais de 2 clientes?
- **Como funciona**: A cláusula `GROUP BY` agrupa os resultados por cidade. A cláusula `HAVING` filtra os grupos com mais de 2 clientes, e `COUNT(CustomerID)` conta o número de clientes em cada grupo.
- **Uso típico**: Útil para identificar cidades com uma concentração significativa de clientes.

Exercício 5

```
SELECT * FROM public."Customers" WHERE "Country" = 'Brazil' AND
"PostalCode" IS NOT NULL;
```

- **Pergunta**: Quais clientes estão localizados no Brasil e têm código postal não nulo?
- **Como funciona**: A cláusula `WHERE` combina duas condições: `Country = 'Brazil'` e `PostalCode IS NOT NULL`.
- **Uso típico**: Útil para obter informações específicas sobre clientes no Brasil com códigos postais válidos.

16. Testando novas tabelas no emulador

Se vocês notaram, existem outras tabelas que não somente `Costumers`. Elas também fazem parte do que chamamos de base de dados (pode ser referenciada como “bd”/“db”/“database”).

Podemos acessá-las individualmente apenas alterando um pouco a sintaxe dos desafios propostos anteriormente Veja bem, em um dos desafios anteriores acessamos os primeiros 5 *records* da tabela Customers com o seguinte código:

```
SELECT * FROM Customers LIMIT 5;
```

Para acessar outra tabela, basta alterar uma porção da sintaxe.

```
SELECT * FROM Tabela LIMIT 5;
```

Como por exemplo:

```
SELECT * FROM Orders LIMIT 5;
```

E aos poucos vamos notando um padrão e estrutura de sintaxe de SQL que se pauta em:

```
SELECIONAR (o que) DE (qual tabela) CONDICIONAIS
```

Observe que todos os exercícios anteriores poderiam ser refeitos utilizando as outras tabelas, se adaptadas as condicionais às respectivas colunas e seus questionamentos.

Mas como podemos fazer para abordar tabelas de maneira simultâneas, para que as nossas pesquisas a uma base de dados sejam feitas de maneira eficiente e prática?

Nos próximos tópicos abordaremos o que chamamos de JOINS no SQL, uma maneira de pivotar dados semelhante ao MERGE do Pandas.

17. Joins

Um JOIN (junção) em SQL é como combinar informações de diferentes tabelas para obter um conjunto de dados mais completo. Imagine duas tabelas como peças de um quebra-cabeça, onde o JOIN é o processo de juntar essas peças pela correspondência de valores em colunas específicas. O INNER JOIN, por exemplo, resulta apenas nas peças que se encaixam perfeitamente, enquanto o LEFT JOIN traz todas as peças da tabela à esquerda e as peças correspondentes da tabela à direita, e assim por diante. Essa abordagem é essencial para unir dados relacionados e extrair insights mais abrangentes ao realizar consultas em bancos de dados relacionais.

Existem quatro tipos principais de junções: INNER JOIN, LEFT JOIN, RIGHT JOIN e FULL JOIN.

A estrutura básica de uma query SQL para realizar um Join pode ser escrita da seguinte maneira:

```
SELECT colunas
FROM tabela1
JOIN tabela2 ON tabela1.coluna = tabela2.coluna;
```

- INNER JOIN: Retorna registros que possuem valores correspondentes em ambas as tabelas.
- LEFT JOIN: Retorna todos os registros da tabela à esquerda (tabela da esquerda da instrução JOIN).
- RIGHT JOIN: Retorna todos os registros da tabela à direita (tabela da direita da instrução JOIN).
- FULL JOIN: Retorna todos os registros quando há uma correspondência em qualquer uma das tabelas, incluindo registros da tabela à esquerda que não têm correspondência na tabela à direita e vice-versa.

INNER JOIN

A junção INNER JOIN é usada para combinar registros de duas tabelas que têm valores correspondentes nas colunas especificadas.

Exemplo:

```
SELECT clientes.nome, pedidos.numero_pedido
FROM clientes
INNER JOIN pedidos ON clientes.id = pedidos.id_cliente;
```

LEFT JOIN

A junção LEFT JOIN retorna todos os registros da tabela da esquerda (tabela à esquerda da instrução JOIN).

Exemplo:

```
SELECT clientes.nome, pedidos.numero_pedido
FROM clientes
LEFT JOIN pedidos ON clientes.id = pedidos.id_cliente;
```

RIGHT JOIN

A junção `RIGHT JOIN` é semelhante ao `LEFT JOIN`, mas retorna todos os registros da tabela da direita.

Exemplo:

```
SELECT clientes.nome, pedidos.numero_pedido
FROM clientes
RIGHT JOIN pedidos ON clientes.id = pedidos.id_cliente;
```

FULL JOIN

A junção `FULL JOIN` retorna todos os registros quando há uma correspondência em qualquer uma das tabelas. Ele inclui registros da tabela esquerda que não têm correspondência na tabela direita e vice-versa.

Exemplo:

```
SELECT clientes.nome, pedidos.numero_pedido
FROM clientes
FULL JOIN pedidos ON clientes.id = pedidos.id_cliente;
```

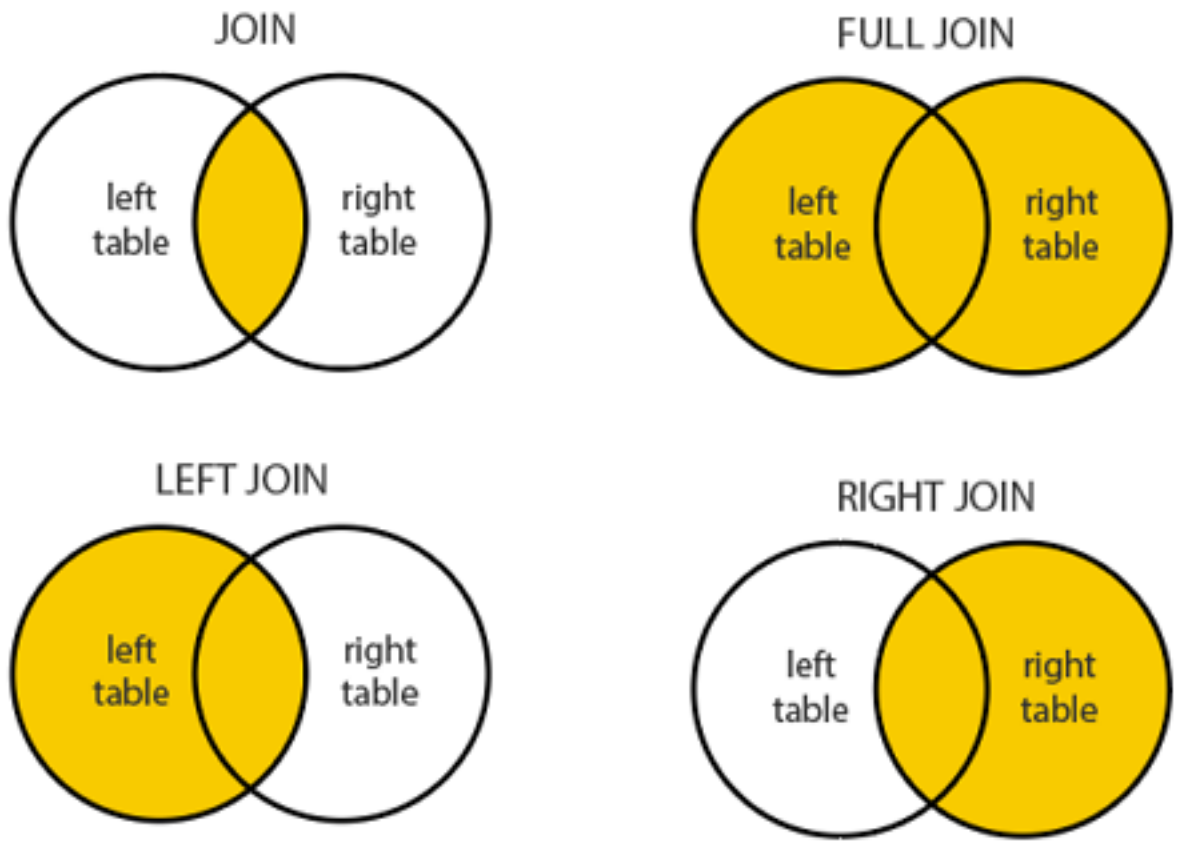

Figure 7: Representação visual dos tipos de Joins de SQL

Esses são os conceitos básicos das junções em SQL. Cada tipo de junção tem seu propósito, e a escolha depende dos requisitos específicos de sua consulta.

18. Dados para testes de Joins e Aliases

Nesta parte do material adicionaremos tabelas mais simples ao PGAdmin 4 para que possamos testar com mais assertividade os conhecimentos sobre Joins e Aliases.

Abaixo temos o material disponível para adição das tabelas no administrador de PostgreSQL. A explicação detalhada está contida nos vídeos da plataforma da Asimov Academy.

Criação das duas tabelas:

```
CREATE TABLE departamentos (
    id SERIAL PRIMARY KEY,
    nome VARCHAR(255) NOT NULL
);

CREATE TABLE funcionarios (
    id SERIAL PRIMARY KEY,
    nome VARCHAR(255) NOT NULL,
    departamento_id INTEGER,
    gerente_id INTEGER,
    FOREIGN KEY (departamento_id) REFERENCES departamentos (id),
    FOREIGN KEY (gerente_id) REFERENCES funcionarios (id)
);
```

Notas

`SERIAL` é usado para criar uma coluna com autoincremento, útil para IDs.

`VARCHAR(255)` define o tipo de dado para strings com um limite de 255 caracteres.

As chaves estrangeiras (`FOREIGN KEY`) criam uma ligação entre as colunas de diferentes tabelas, assegurando a integridade referencial.

Este exemplo assume que você deseja estabelecer uma relação hierárquica dentro da tabela funcionarios onde um funcionário pode ser gerente de outro

Para inserir os dados:

```
INSERT INTO departamentos (nome)
VALUES
('Recursos Humanos'),
('TI'),
('Edição'),
('Vendas'),
('Marketing');

INSERT INTO funcionarios (nome, departamento_id)
VALUES
('Rodrigo Vanzelotti', 2),
('Deborah St', 3),
```

```
('Rodrigo Tadewald', 5),
('Renata Lopes', 5),
('Roberta de Souza', 4),
('Henrique Façanha', 4)
;
```

19. Aliases em SQL

Aliases são apelidos temporários (variáveis) atribuídos a tabelas ou colunas em consultas SQL. Sua utilização pode simplificar consultas, especialmente aquelas que envolvem múltiplas tabelas e operações de `JOIN`.

Por Que Usar Aliases?

- **Legibilidade:** Tornam as consultas mais legíveis, especialmente quando lidamos com nomes de tabelas ou colunas longos e complexos.
- **Concisão:** Reduzem a quantidade de texto necessário, facilitando a escrita e compreensão de consultas.
- **Necessidade:** Em algumas situações, como nos `JOIN` auto-referenciais, os aliases são indispensáveis para diferenciar as tabelas.

Sintaxe Básica

```
SELECT coluna AS alias
FROM tabela AS alias_tabela
WHERE condição;
```

- **AS** é opcional no PostgreSQL. Você pode omiti-lo se preferir, mas sua presença melhora a legibilidade.

Exemplo Simples

Considere a tabela `funcionarios`:

id	nome	departamento_id
1	João Silva	3
2	Maria Lima	2

Uma consulta simples com alias:

```
SELECT nome AS NomeFuncionario
FROM funcionarios AS f
WHERE f.departamento_id = 2;
```

Aliases em Joins

Joins são fundamentais em SQL para combinar registros de duas ou mais tabelas. Aliases se tornam particularmente úteis aqui, permitindo consultas mais claras e concisas.

Tipos de Join

- **INNER JOIN:** Retorna registros com correspondência em ambas as tabelas.
- **LEFT JOIN (ou LEFT OUTER JOIN):** Retorna todos os registros da tabela à esquerda, e os registros correspondentes da tabela à direita.
- **RIGHT JOIN (ou RIGHT OUTER JOIN):** O inverso do LEFT JOIN.
- **FULL JOIN (ou FULL OUTER JOIN):** Combina todos os registros de ambas as tabelas, com correspondências onde disponíveis.

Exemplo Prático com Join

Considere uma segunda tabela, `departamentos`:

id	nome
2	Recursos Humanos
3	TI

Para combinar `funcionarios` e `departamentos`, usaremos um INNER JOIN:

```
SELECT f.nome AS NomeFuncionario, d.nome AS Departamento
FROM funcionarios f
INNER JOIN departamentos d ON f.departamento_id = d.id;
```

Aliases e Joins Complexos

Aliases são essenciais em joins mais complexos, especialmente quando uma tabela é referenciada múltiplas vezes.

Auto-Referência com Alias Imagine que `funcionarios` tenha uma coluna `gerente_id`, apontando para o `id` de outro funcionário na mesma tabela. Para buscar funcionários e seus gerentes por nome:

```sql
SELECT f1.nome AS Funcionario, f2.nome AS Gerente
FROM funcionarios f1
LEFT JOIN funcionarios f2 ON f1.gerente_id = f2.id;
```

Conclusão

Aliases e Joins são ferramentas poderosas em SQL, trabalhando juntas para tornar as consultas mais eficientes, legíveis e concisas. A prática desses conceitos eleva sua habilidade de manipulação de dados, permitindo construir consultas complexas de maneira eficaz. Experimente, explore variações e observe como seu código se torna mais claro e manutenível.

20. Desafio Joins - parte 1

Estas perguntas são projetadas para praticar o uso de diferentes tipos de `JOIN` em SQL, incluindo `INNER JOIN`, `LEFT JOIN`, `RIGHT JOIN`, e `FULL JOIN`, bem como combinações mais complexas e funções de agregação.

Tabelas utilizadas: `Customers`, `Orders` e `Employees`.

1. Como você consultaria o nome do cliente (`CustomerName`) e o nome do funcionário (`FirstName`, `LastName`) para cada pedido?
2. Como você encontraria todos os pedidos feitos por clientes de um determinado país, por exemplo, "Brasil"?
3. Como você listaria todos os clientes e os pedidos associados a eles, incluindo clientes que não fizeram nenhum pedido?
4. Como você mostraria todos os pedidos e os nomes dos clientes associados, incluindo pedidos que não têm um cliente associado?

21. Solução desafios de Joins - parte 1

Exercício 1

```sql
SELECT c."CustomerName", e."FirstName", e."LastName"
FROM "Orders" o
INNER JOIN "Customers" c ON o."CustomerID" = c."CustomerID"
INNER JOIN "Employees" e ON o."EmployeeID" = e."EmployeeID"
```

- **Pergunta**: Como você consultaria o nome do cliente (`CustomerName`) e o nome do funcionário (`FirstName`, `LastName`) para cada pedido?
  - **Como funciona**: Esta consulta usa `JOIN` para combinar linhas da tabela `orders` com as tabelas `customers` e `employees`, baseando-se nas chaves estrangeiras `CustomerID` e `EmployeeID`, respectivamente.
  - **Uso típico**: Em aplicações de e-commerce, essa consulta pode ser usada para gerar relatórios de pedidos, mostrando quem fez o pedido e quem é responsável por processá-lo, facilitando a gestão de pedidos e atendimento ao cliente.

Exercício 2

```sql
SELECT o."OrderID", c."CustomerName"
FROM "Orders" o
INNER JOIN "Customers" c ON o."CustomerID" = c."CustomerID"
WHERE c."Country" = 'Brazil'
```

- **Pergunta**: Como você encontraria todos os pedidos feitos por clientes de um determinado país, por exemplo, “Brasil”?
  - **Como funciona**: A consulta usa `JOIN` para ligar `orders` a `customers` e então aplica um filtro `WHERE` sobre a coluna `Country` da tabela `customers`, selecionando apenas os pedidos dos clientes desse país.
  - **Uso típico**: Importante para análise de mercado, identificando demandas de produtos ou serviços em regiões específicas, o que pode orientar estratégias de marketing e expansão de negócios.

Exercício 3

```sql
SELECT c."CustomerName", o."OrderID"
FROM "Customers" c
LEFT JOIN "Orders" o ON o."CustomerID" = c."CustomerID"
```

- **Pergunta**: Como você listaria todos os clientes e os pedidos associados a eles, incluindo clientes que não fizeram nenhum pedido?

  - **Como funciona**: Utilizando `LEFT JOIN` entre `customers` e `orders`, essa consulta inclui todos os clientes, mesmo aqueles sem pedidos associados, mostrando `NULL` nos campos do pedido para esses clientes.
  - **Uso típico**: Essencial para análises de engajamento do cliente, permitindo identificar clientes inativos que podem ser alvo de campanhas de marketing específicas para reengajá-los.

Exercício 4

```sql
SELECT c."CustomerName", o."OrderID"
FROM "Orders" o
LEFT JOIN "Customers" c ON o."CustomerID" = c."CustomerID"
```

- **Pergunta**: Como você mostraria todos os pedidos e os nomes dos clientes associados, incluindo pedidos que não têm um cliente associado?

  - **Como funciona**: A consulta usa `RIGHT JOIN` para garantir que todos os pedidos sejam listados, inclusive aqueles sem um cliente correspondente, preenchendo com `NULL` as informações do cliente quando não houver correspondência.
  - **Uso típico**: Útil para auditorias de integridade de dados, onde é necessário verificar se existem pedidos que não estão corretamente associados a clientes.

22. Desafio Joins - parte 2

5. Como você listaria todos os clientes e todos os pedidos, mostrando as correspondências onde existem e os registros sem correspondência de ambas as tabelas?
6. Como você consultaria o nome do cliente, o nome do funcionário responsável pelo pedido e os detalhes do pedido (ID do pedido e quantidade)?
7. Como você calcularia o número total de pedidos por cliente?

23. Solução desafios de Joins - parte 2

Exercício 5

```sql
SELECT customers.CustomerName, orders.OrderID
FROM customers
FULL JOIN orders ON customers.CustomerID = orders.CustomerID;
```

- **Pergunta**: Como você listaria todos os clientes e todos os pedidos, mostrando as correspondências onde existem e os registros sem correspondência de ambas as tabelas?
  - **Como funciona**: `FULL JOIN` combina `customers` e `orders` mostrando todas as correspondências e preenchendo com `NULL` onde não há correspondência, seja de cliente sem pedido ou pedido sem cliente.
  - **Uso típico**: Essa abordagem é útil para uma visão completa de atividades de clientes e pedidos, permitindo análises detalhadas de desempenho de vendas e engajamento de clientes.

Exercício 6

```sql
SELECT customers.CustomerName, employees.FirstName, employees.LastName, order_details.OrderID,
↪  order_details.Quantity
FROM order_details
JOIN orders ON order_details.OrderID = orders.OrderID
JOIN customers ON orders.CustomerID = customers.CustomerID
JOIN employees ON orders.EmployeeID = employees.EmployeeID;
```

- **Pergunta**: Como você consultaria o nome do cliente, o nome do funcionário responsável pelo pedido e os detalhes do pedido (ID do pedido e quantidade)?
  - **Como funciona**: A consulta faz múltiplos `JOIN`s para conectar `order_details` a `orders`, e então a `customers` e `employees`, permitindo a visualização detalhada de cada pedido, quem fez, quem processou e os detalhes do pedido.
  - **Uso típico**: Ideal para geração de relatórios detalhados de pedidos, que são fundamentais para análises operacionais, logísticas e de atendimento ao cliente em negócios de varejo ou e-commerce.

Exercício 7

```sql
SELECT customers.CustomerName, COUNT(orders.OrderID) AS TotalOrders
FROM customers
LEFT JOIN orders ON customers.CustomerID = orders.CustomerID
GROUP BY customers.CustomerName;
```

- **Pergunta**: Como você calcularia o número total de pedidos por cliente?
  - **Como funciona**: A consulta faz um `LEFT JOIN` de `customers` com `orders` e usa `GROUP BY` para agrupar os resultados por cliente, utilizando a função de agregação `COUNT` para contar os pedidos de cada cliente.
  - **Uso típico**: Essencial para análises de fidelidade do cliente, permitindo identificar clientes mais ativos que podem ser recompensados ou focados em campanhas de retenção, bem como identificar aqueles com baixa atividade para ações de reengajamento.