



01000101 01110011 01100011 01101111 01101100 01100001
00100000 01101000 01100101 00100000 01000100 01100001
01101000 01101111 01110011 00001010

GLOSSÁRIO SQL

BOAS-VINDAS AO GLOSSÁRIO DE SQL!

Este é um guia prático que simplifica os fundamentos da linguagem de consulta estruturada. Este glossário foi criado para proporcionar uma introdução clara e concisa aos comandos mais comuns do SQL, como Select, Insert, Update, entre outros. Explore e aprofunde-se no mundo dos bancos de dados, utilizando este guia para fortalecer sua compreensão e aplicação do SQL. Aproveite o material e, em caso de dúvidas, sinta-se à vontade para enviar perguntas através do fórum do curso.

Um abraço e bons estudos!

SUMÁRIO

01000101 0110011 01100011 01101111 01101100 01100001
00100000 01100100 01100101 00100000 01000100 01100001
01100100 01101111 01100111 00001010

✿ UTILIZANDO O COMANDO SELECT.....	04
✿ FILTROS WHERE E DISTINCT.....	07
✿ CRIANDO TABELAS	11
✿ ALTERANDO E EXCLUINDO TABELAS	14
✿ CHAVE PRIMÁRIA.....	18
✿ CHAVE ESTRANGEIRA.....	21
✿ INSERINDO DADOS – INSERT	27

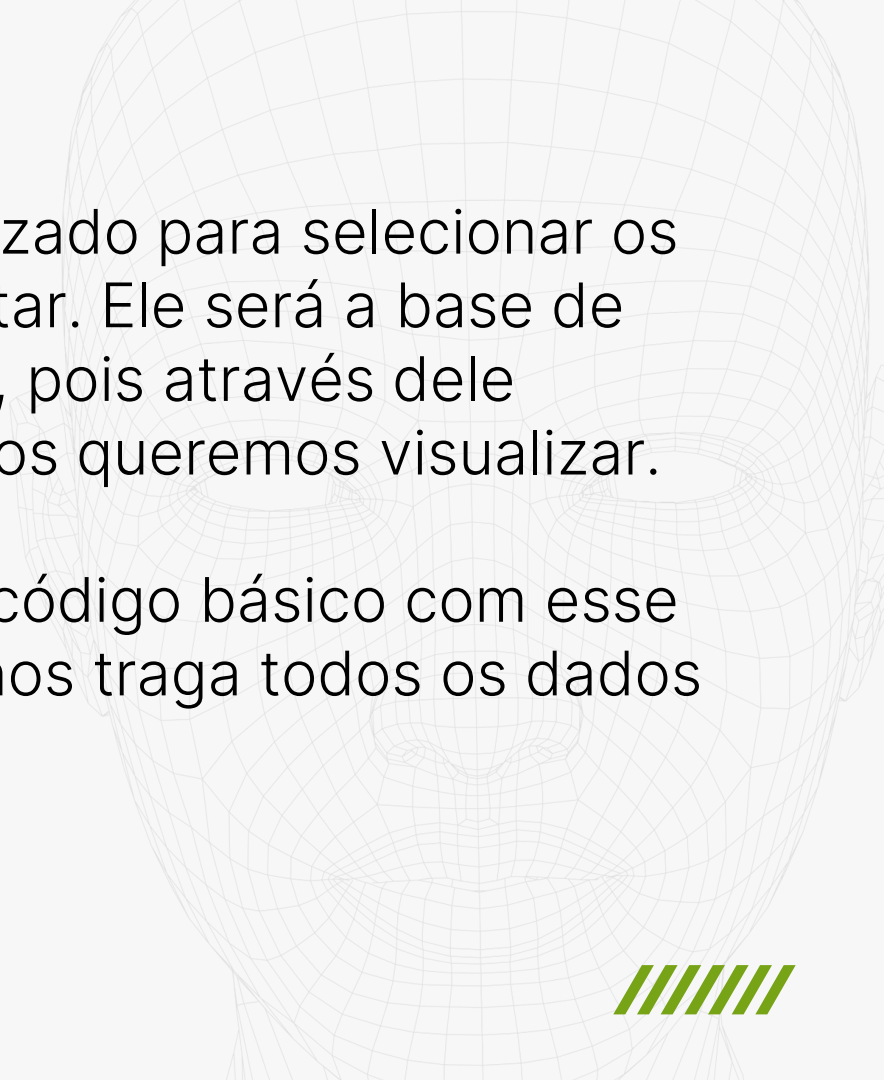
✿ COMBINANDO INSERT E SELECT	32
✿ FILTROS DE MAIOR, MENOR, IGUAL E DIFERENTE	37
✿ FILTROS COMPOSTOS - AND, OR, NOT E BETWEEN	46
✿ ORDENANDO OS DADOS - ORDER BY	52
✿ UTILIZANDO ALIAS	58
✿ ATUALIZANDO E EXCLUINDO DADOS	61

01000101 0110011 01100011 01101111 01101100
01100001 00100000 01100100 01100101 00100000
01000100 01100001 01100100 01101111 0110011
00001010

// Glossário SQL_

UTILIZANDO O COMANDO SELECT



A faint, stylized wireframe of a human face is visible in the background, centered on the right side of the slide.

O comando **SELECT** será utilizado para selecionar os dados que desejamos consultar. Ele será a base de várias consultas que faremos, pois através dele conseguimos dizer quais dados queremos visualizar.

A primeira forma de criar um código básico com esse comando é pedindo que ele nos traga todos os dados de uma tabela.



```
SELECT * FROM nome_da_tabela;
```

O **SELECT** pede para que sejam selecionados os dados, o **asterisco (*)** significa que queremos todos os dados e o **FROM** determina a origem desses dados.

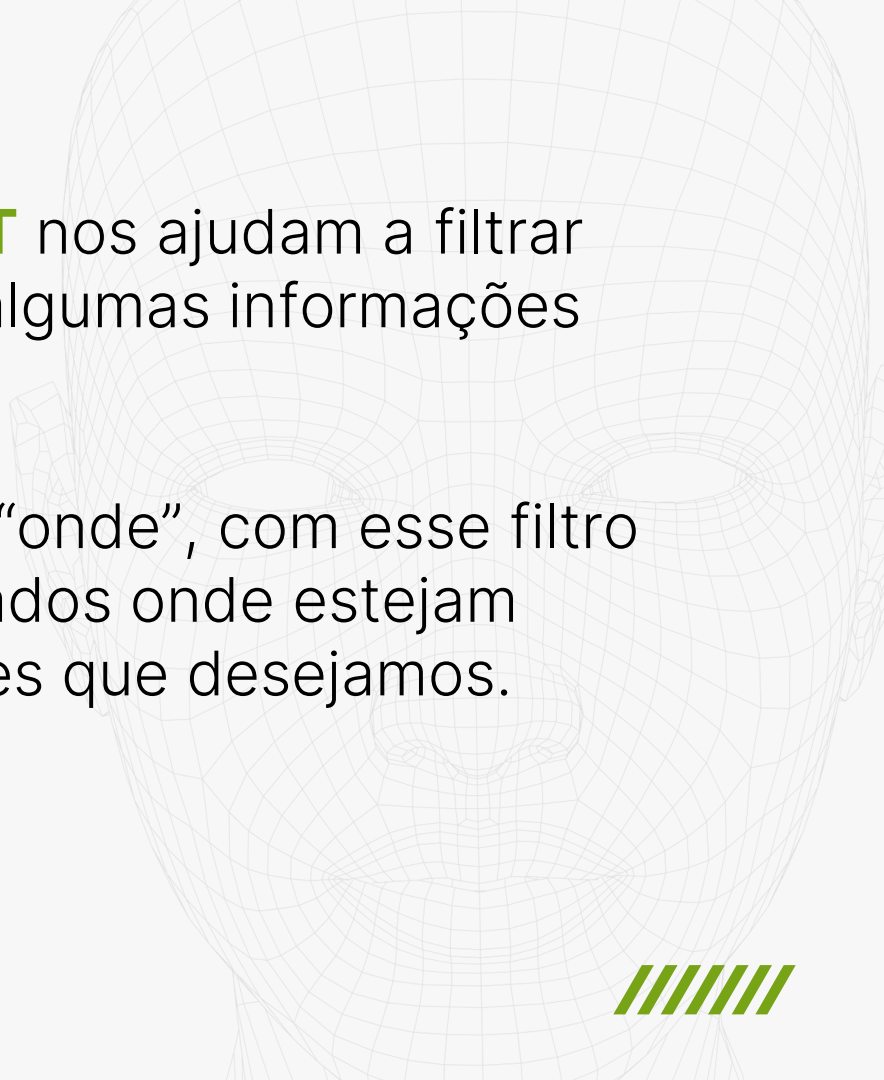


01000101 0110011 01100011 0110111 01101100
01100001 00100000 01100100 01100101 00100000
01000100 01100001 01100100 0110111 0110011
00001010

// Glossário SQL_

FILTROS WHERE E DISTINCT



A faint, light green wireframe of a human face is visible in the background, centered on the right side of the slide.

Os filtros **WHERE** e **DISTINCT** nos ajudam a filtrar nossos dados com base em algumas informações específicas.

WHERE, traduzindo significa “onde”, com esse filtro podemos trazer apenas os dados onde estejam contidas certas especificações que desejamos.



Por exemplo, se quisermos filtrar na tabela de fornecedores apenas os dados que possuam como país de origem a China, ficaria dessa forma:

```
SELECT * FROM tabelaforneecedores  
WHERE _país_de_origem = 'China';
```



Já o filtro **DISTINCT**, traduzindo significa “distinto”, com esse filtro conseguimos trazer apenas os dados distintos de uma certa coluna, por exemplo, para consultar apenas o número de clientes únicos contidos na base de dados vamos usar o DISTINCT da seguinte forma:

```
SELECT DISTINCT ID_Cliente_pedidos  
FROM tabelapedidos;
```

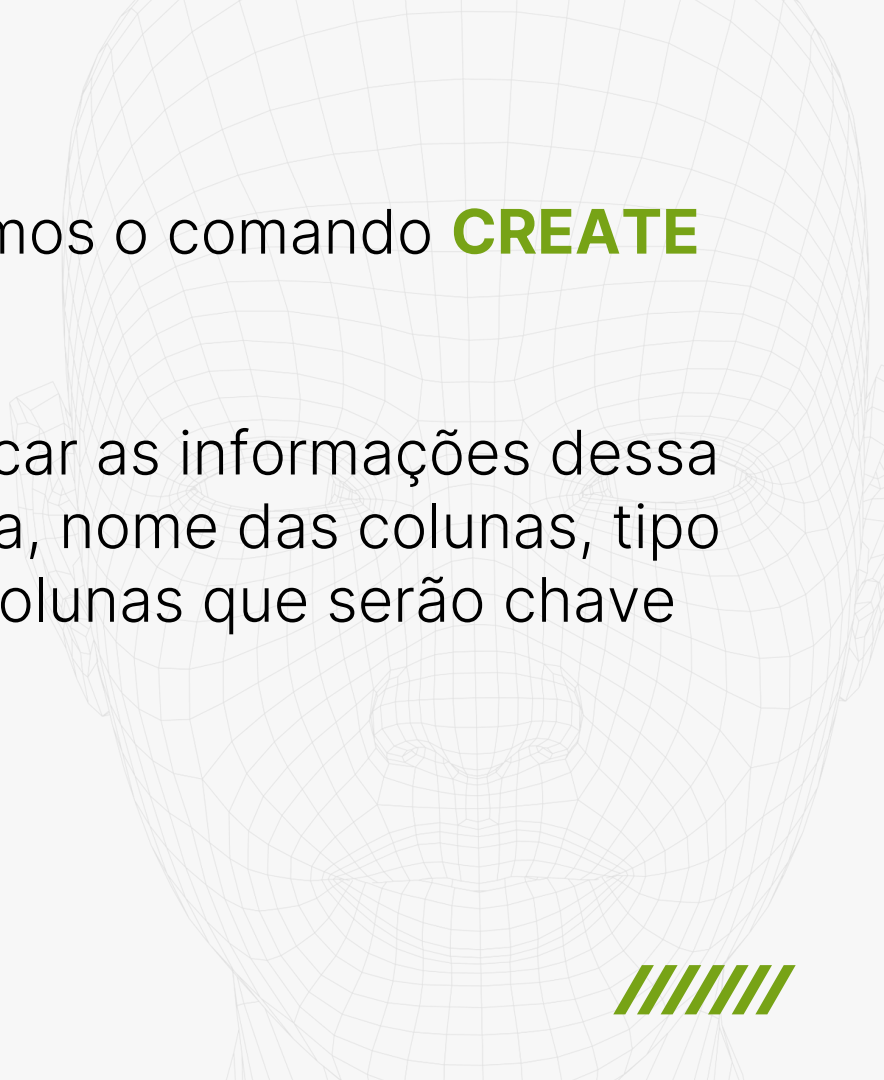


01000101 0110011 01100011 0110111 01101100
01100001 01000000 01100100 01100101 00100000
01000100 01100001 01100100 0110111 0110011
00001010

// Glossário SQL_

CRIANDO TABELAS



A faint, light green wireframe of a human face is visible in the background on the right side of the slide.

Para criarmos tabelas, utilizamos o comando **CREATE TABLE**.

Além disso devemos especificar as informações dessa tabela como o nome da tabela, nome das colunas, tipo de dado de cada coluna, as colunas que serão chave primária ou estrangeira, etc.



Por exemplo, a criação da tabela de clientes, contendo três colunas, ficaria assim:

```
CREATE TABLE tabelaclientes (  
    ID_Cliente INT PRIMARY KEY,  
    Nome_Cliente VARCHAR(250),  
    Informacoes_De_Contato VARCHAR(250)  
);
```



01000101 0110011 01100011 01101111 01101100
01100001 00100000 01100100 01100101 00100000
01000100 01100001 01100100 01101111 0110011
00001010

// Glossário SQL_

ALTERANDO E EXCLUINDO TABELAS



Para alterarmos a estrutura de uma tabela utilizamos o comando **ALTER TABLE**, com esse comando conseguimos modificar a estrutura de uma tabela mesmo após sua criação, por exemplo, se quisermos acrescentar uma nova coluna a uma tabela já criada, no caso acrescentar a coluna de endereço do cliente na tabela de clientes, ficaria dessa forma:

```
ALTER TABLE tabelaclientes ADD  
Endereço_Cliente VARCHAR(250);
```


Já para excluir uma tabela utilizamos o comando **DROP TABLE**, com esse comando excluimos a estrutura da tabela, inteira, não apenas os dados contidos nela. Por exemplo, caso fosse necessário excluir a tabela de clientes do nosso banco de dados, o código seria esse:

```
DROP TABLE tabelaclientes;
```



// ATENÇÃO_

É importante utilizarmos esse comando com **cautela**, pois ele excluirá a estrutura da tabela **por completo** do seu banco de dados!

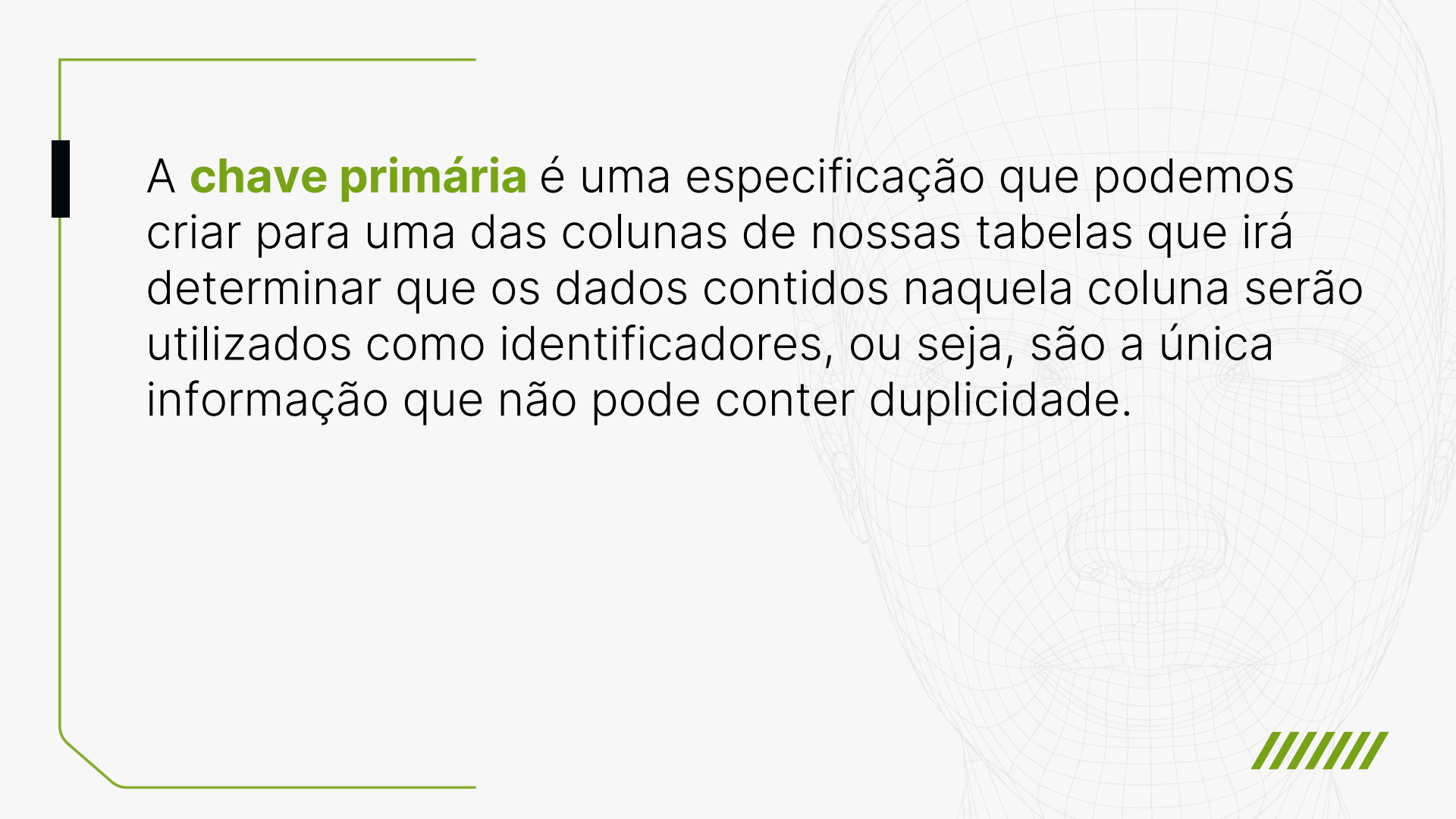


01000101 0110011 01100011 0110111 01101100
01100001 01000000 01100100 01100101 00100000
01000100 01100001 01100100 0110111 0110011
00001010

// Glossário SQL_

CHAVE PRIMÁRIA





A **chave primária** é uma especificação que podemos criar para uma das colunas de nossas tabelas que irá determinar que os dados contidos naquela coluna serão utilizados como identificadores, ou seja, são a única informação que não pode conter duplicidade.

Por exemplo, vamos criar uma tabela de categorias dos produto, formada pelas colunas: nome, descrição e ID, que será a chave primária da tabela e servirá para **identificar** cada linha de dado contido na tabela.

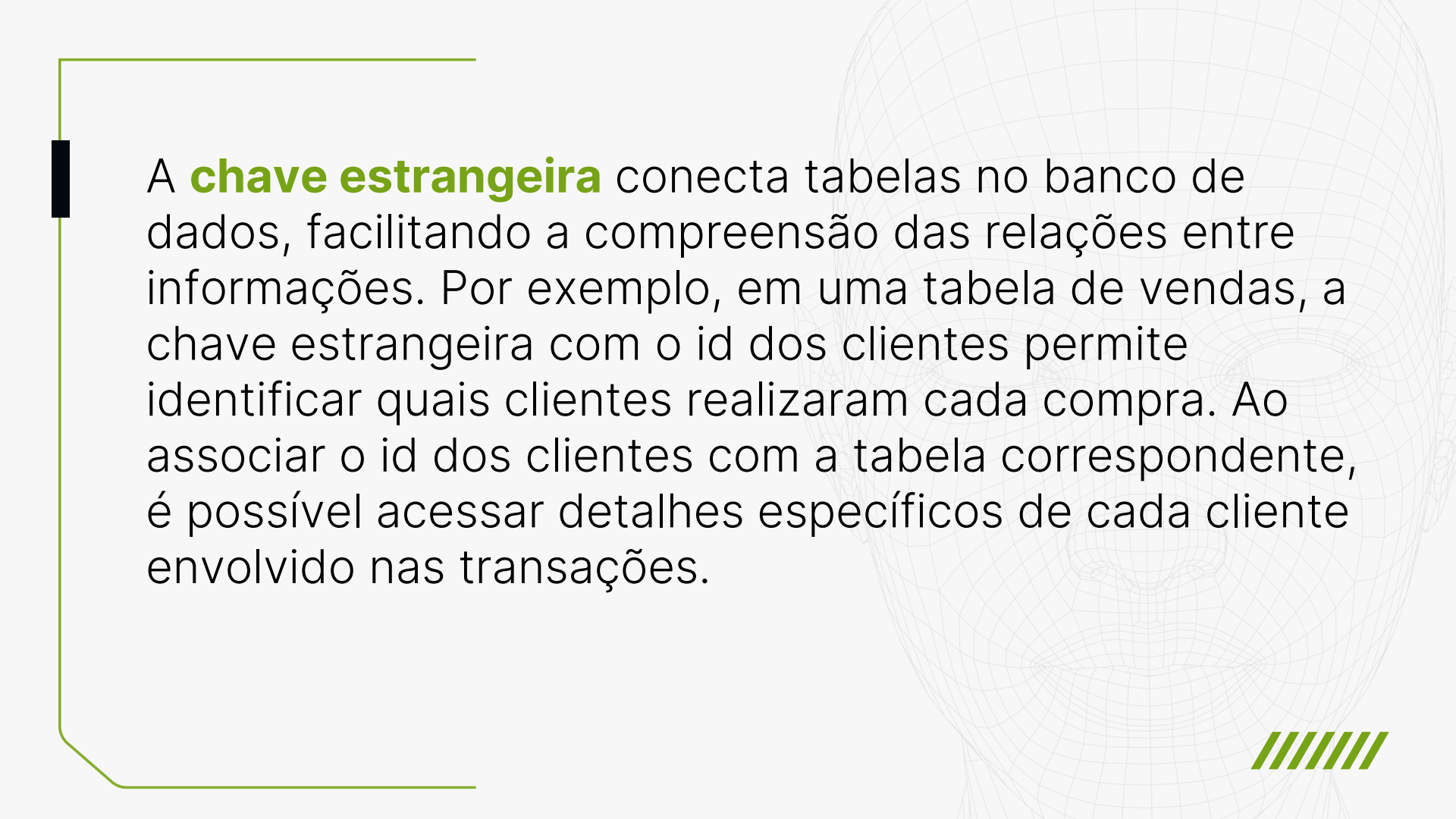
```
CREATE TABLE tabelacategorias (  
    ID_Categoria INT PRIMARY KEY,  
    Nome_Categoria VARCHAR(250),  
    Descricao_Categoria TEXT  
);
```

01000101 0110011 01100011 01101111 01101100
01100001 00100000 01100100 01100101 00100000
01000100 01100001 01100100 01101111 0110011
00001010

// Glossário SQL_

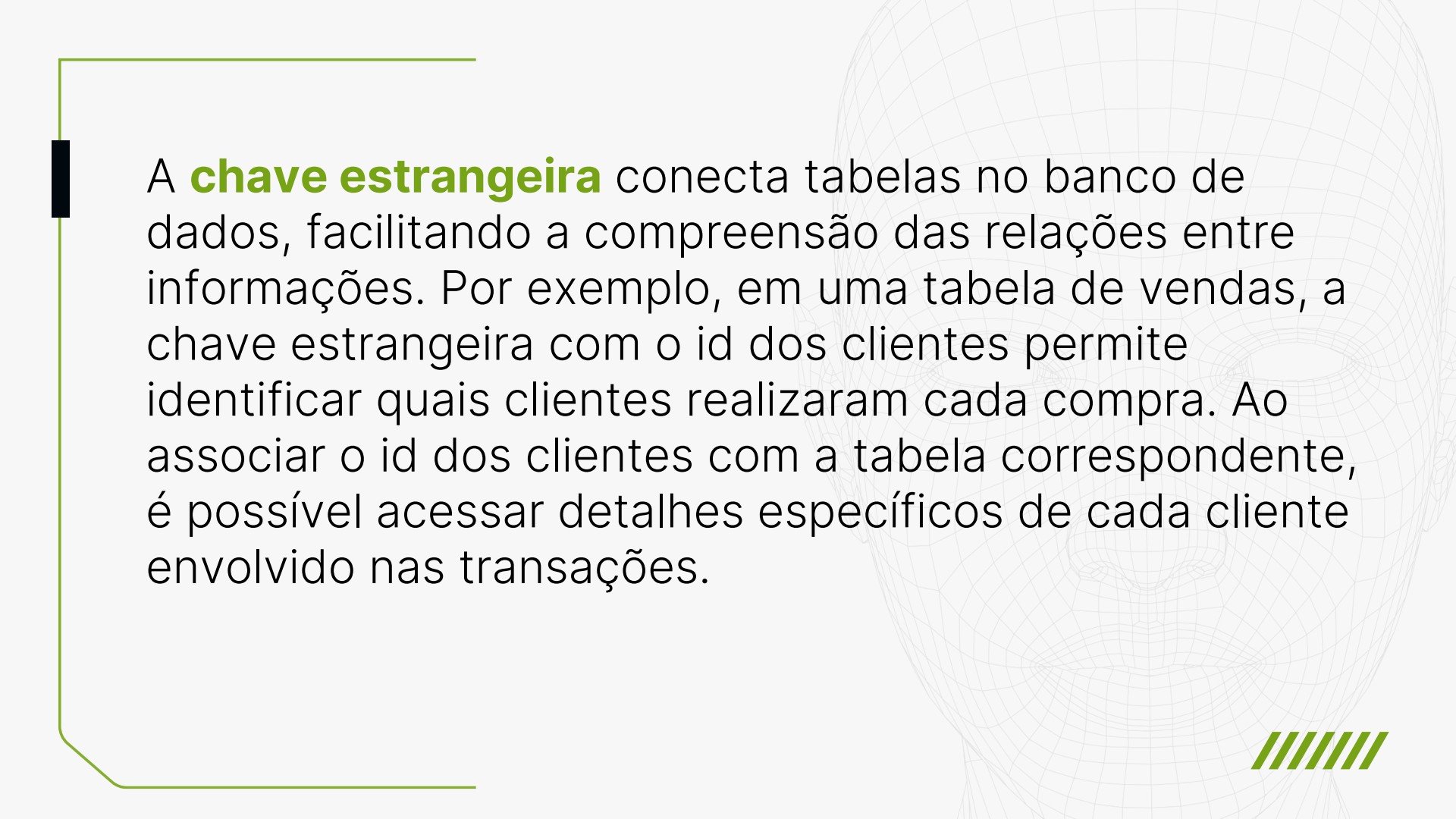
CHAVE ESTRANGEIRA





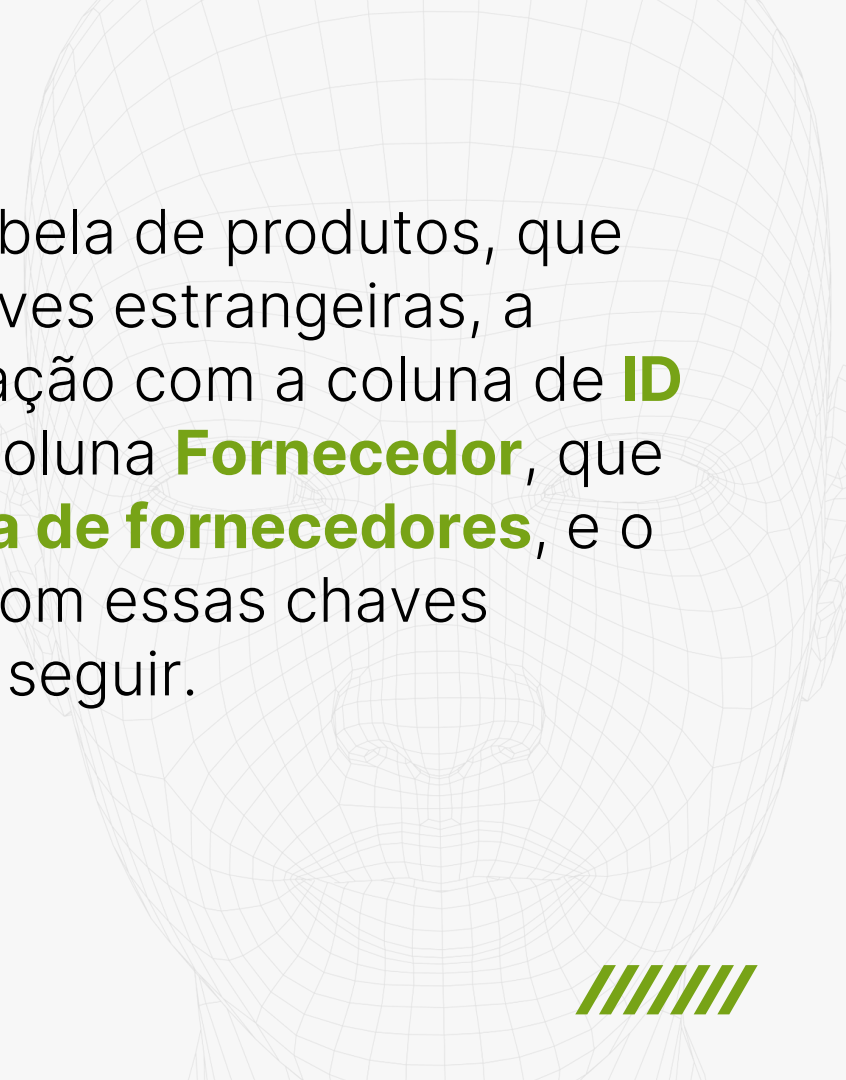
A **chave estrangeira** conecta tabelas no banco de dados, facilitando a compreensão das relações entre informações. Por exemplo, em uma tabela de vendas, a chave estrangeira com o id dos clientes permite identificar quais clientes realizaram cada compra. Ao associar o id dos clientes com a tabela correspondente, é possível acessar detalhes específicos de cada cliente envolvido nas transações.



A decorative background featuring a faint, light green wireframe of a human face. A solid green line runs vertically on the left side, with a short horizontal segment at the top and bottom, forming a partial frame around the text.

A **chave estrangeira** conecta tabelas no banco de dados, facilitando a compreensão das relações entre informações. Por exemplo, em uma tabela de vendas, a chave estrangeira com o id dos clientes permite identificar quais clientes realizaram cada compra. Ao associar o id dos clientes com a tabela correspondente, é possível acessar detalhes específicos de cada cliente envolvido nas transações.





Para exemplificar, criamos a tabela de produtos, que possui duas colunas como chaves estrangeiras, a coluna **Categoria**, que terá ligação com a coluna de **ID da tabela de Categorias**, e a coluna **Fornecedor**, que se liga a coluna de **ID da tabela de fornecedores**, e o código para criar essa tabela com essas chaves estrangeiras ficaria da forma a seguir.



```
CREATE TABLE tabelaprodutos (  
    ID INT PRIMARY KEY,  
    NomeDoProduto VARCHAR(255),  
    Descricao TEXT,  
    Categoria INT,  
    PrecoDeCompra DECIMAL(10, 2),  
    Unidade VARCHAR(50),  
    Fornecedor INT,  
    DataDeInclusao DATE,  
    FOREIGN KEY (Categoria) REFERENCES  
Categorias(ID),  
    FOREIGN KEY (Fornecedor) REFERENCES  
Fornecedores(ID);
```

// ATENÇÃO_

A chave estrangeira **sempre estará ligada**
a uma chave primária da outra tabela.

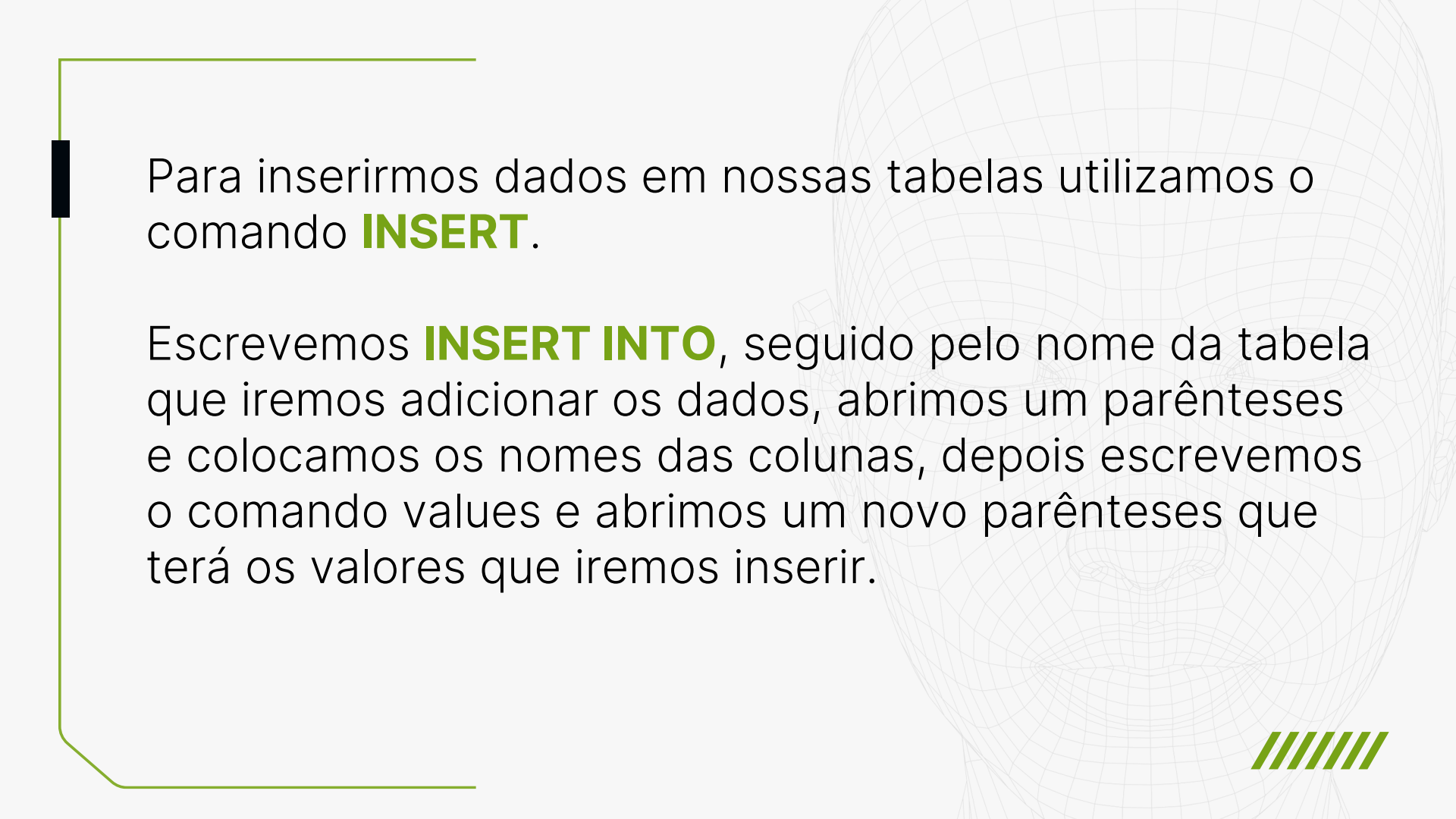


01000101 0110011 01100011 01101111 01101100
01100001 00100000 01100100 01100101 00100000
01000100 01100001 01100100 01101111 0110011
00001010

// Glossário SQL_

INSERINDO DADOS - INSERT

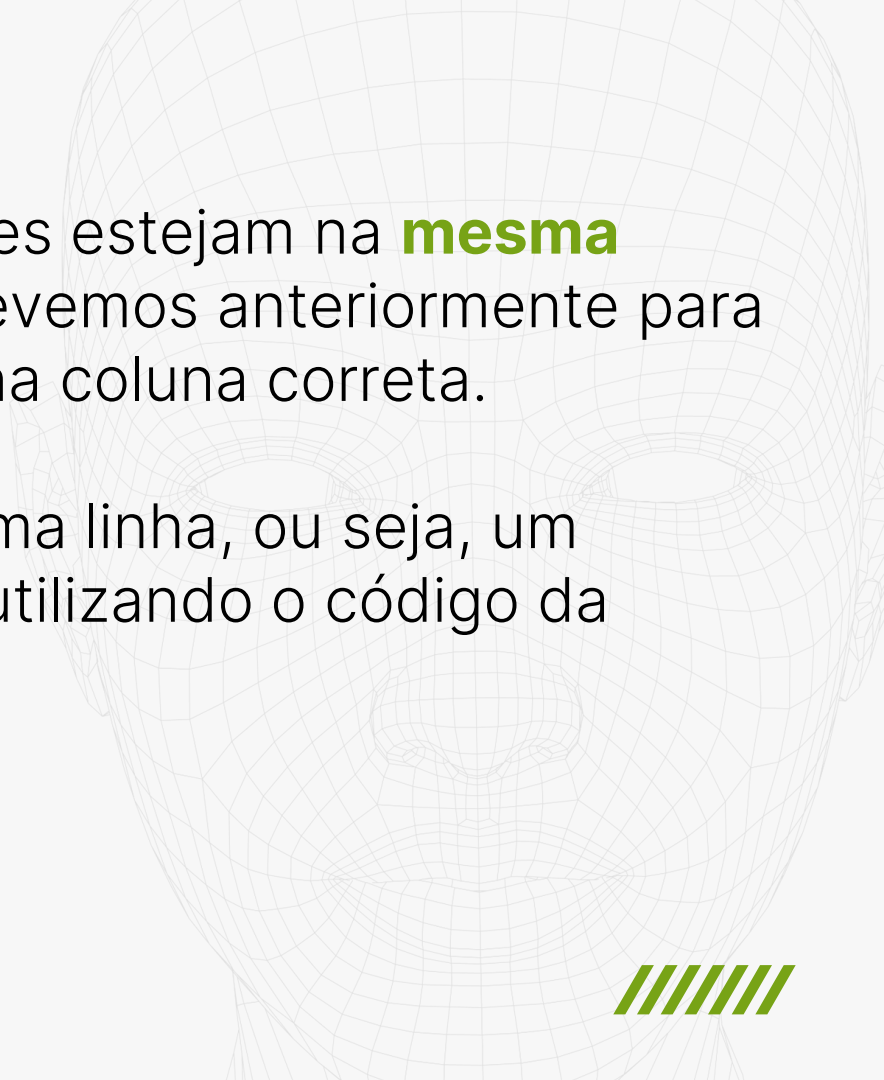




Para inserirmos dados em nossas tabelas utilizamos o comando **INSERT**.

Escrevemos **INSERT INTO**, seguido pelo nome da tabela que iremos adicionar os dados, abrimos um parênteses e colocamos os nomes das colunas, depois escrevemos o comando values e abrimos um novo parênteses que terá os valores que iremos inserir.



A faint, light gray wireframe of a human face is visible in the background on the right side of the slide.

É importante que esses valores estejam na **mesma ordem** das colunas que escrevemos anteriormente para que cada valor seja inserido na coluna correta.

Por exemplo, vamos inserir uma linha, ou seja, um cliente na tabela de clientes utilizando o código da forma a seguir.




```
INSERT INTO tabelaclientes  
(id_cliente, nome_cliente, informacoes_de_contato,  
endereco_cliente)  
VALUES  
('1', 'Ana Silva', 'ana.silva@email.com', 'rua  
flores - casa 1');
```

Para inserir mais de uma linha de uma só vez utilizamos o **mesmo código**, apenas vamos acrescentando mais parênteses com as informações de cada linha, como veremos na próxima página.

```
INSERT INTO tabelaclientes  
(id_cliente,nome_cliente, informacoes_de_contato,  
endereço_cliente)
```

```
VALUES
```

```
('2', 'João Santos', 'joao.santos@provedor.com',  
'Rua dos pinheiros, 25'),
```

```
('3', 'Maria Fernandes', 'maria.fernandes@email.com',  
'Rua Santo Antonio, 10'),
```

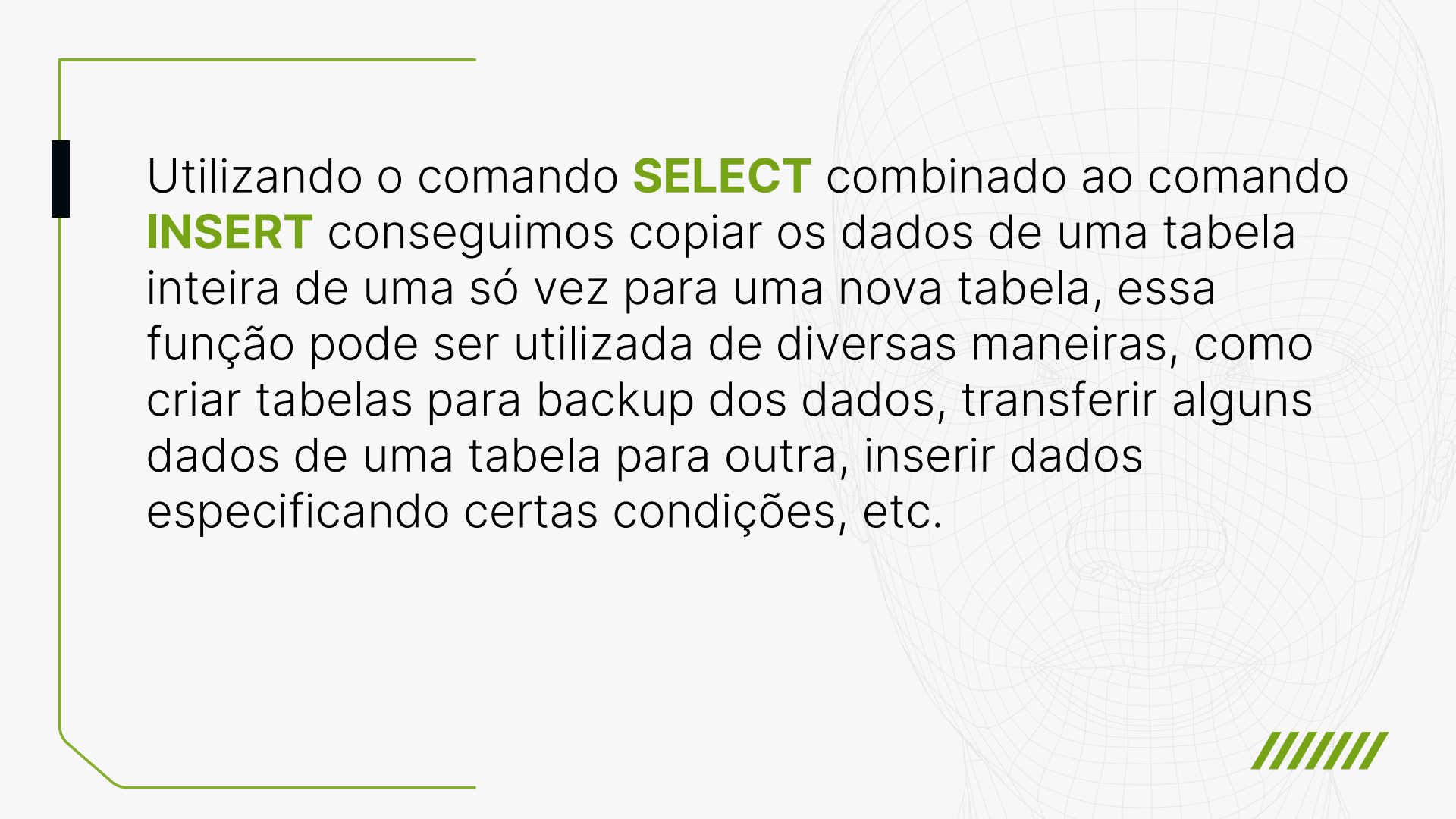
```
('4', 'Carlos Pereira', 'carlos.pereira@email.com',  
'Avenida rio, 67');
```

01000101 0110011 01100011 01101111 01101100
01100001 00100000 01100100 01100101 00100000
01000100 01100001 01100100 01101111 0110011
00001010

// Glossário SQL_

INSERINDO DADOS COMBINANDO OS COMANDOS INSERT E SELECT



A decorative background featuring a faint, light green wireframe of a human face. A solid green line runs vertically on the left side, with a short horizontal segment at the top and a longer one at the bottom, forming a partial frame around the text.

Utilizando o comando **SELECT** combinado ao comando **INSERT** conseguimos copiar os dados de uma tabela inteira de uma só vez para uma nova tabela, essa função pode ser utilizada de diversas maneiras, como criar tabelas para backup dos dados, transferir alguns dados de uma tabela para outra, inserir dados especificando certas condições, etc.



Por exemplo, criamos uma nova tabela de pedidos **gold**, que irá conter apenas os pedidos com valores iguais ou acima de \$400 reais, com ajuda do **SELECT** conseguimos fazer isso de uma forma bem prática.

Utilizando o comando **INSERT INTO**, seguido pelo nome da nossa nova tabela, entre parênteses especificamos os campos da nossa nova tabela e abaixo vamos colocar o **SELECT** seguido dos campos da tabela original de pedidos, que serão os campos que vamos copiar os dados.



A faint, stylized wireframe of a human head is visible in the background, facing forward. It is composed of a grid of lines forming the facial structure.

Mas como vamos explicar que queremos trazer apenas os pedidos com valores iguais ou maiores que R\$400?

Acrescentando a cláusula **WHERE**, que já aprendemos por aqui também. Então, a gente coloca **WHERE**, ou seja, onde os valores da coluna `total_do_pedido` sejam maiores ou iguais a 400, o código fica da forma a seguir.



```
INSERT into tabelapedidosgold (  
    ID_pedido_gold,  
    Data_Do_Pedido_gold,  
    Status_gold,  
    Total_Do_Pedido_gold,  
    Cliente_gold,  
    Data_De_Envio_Estimada_gold)  
SELECT  
    id,  
    data_do_pedido,  
    status,  
    total_do_pedido,  
    id_cliente_pedidos,  
    data_de_envio_estimada  
from tabelapedidos  
WHERE total_do_pedido >= 400;
```


01000101 0110011 01100011 01101111 01101100
01100001 00100000 01100100 01100101 00100000
01000100 01100001 01100100 01101111 0110011
00001010

// Glossário SQL_

FILTROS DE MAIOR, MENOR, IGUAL E DIFERENTE



Podemos usar os filtros de **maior (>)**, **menor (<)**, **igual (=)** e **diferente (<>)** para consultarmos valores específicos no total do nosso pedido.

Por exemplo, quero selecionar apenas as vendas acima de 200 reais, então fazemos:

```
SELECT * FROM tabelapedidos WHERE  
total_do_pedido > 200;
```



Podemos utilizar o sinal de igual caso queiramos que os pedidos no valor de 200 também sejam filtrados, ficando da seguinte forma:

```
SELECT * FROM tabelapedidos WHERE  
total_do_pedido >= 200;
```

O mesmo vale para o sinal de menor e menor ou igual:

```
SELECT * FROM tabelapedidos WHERE  
total_do_pedido <= 200;
```

// PROBLEMÁTICA_

E se a gente quiser ver todos os pedidos
menos os pedidos com o valor de 200 reais?



Aí utilizamos o símbolo de diferente, que no caso do SQL é representado pelo sinal de menor e maior um seguido do outro, da seguinte forma:

```
SELECT * FROM tabelapedidos WHERE  
total_do_pedido <> 200;
```

Dessa forma teremos todos os valores diferentes de 200 reais na coluna total do pedido.



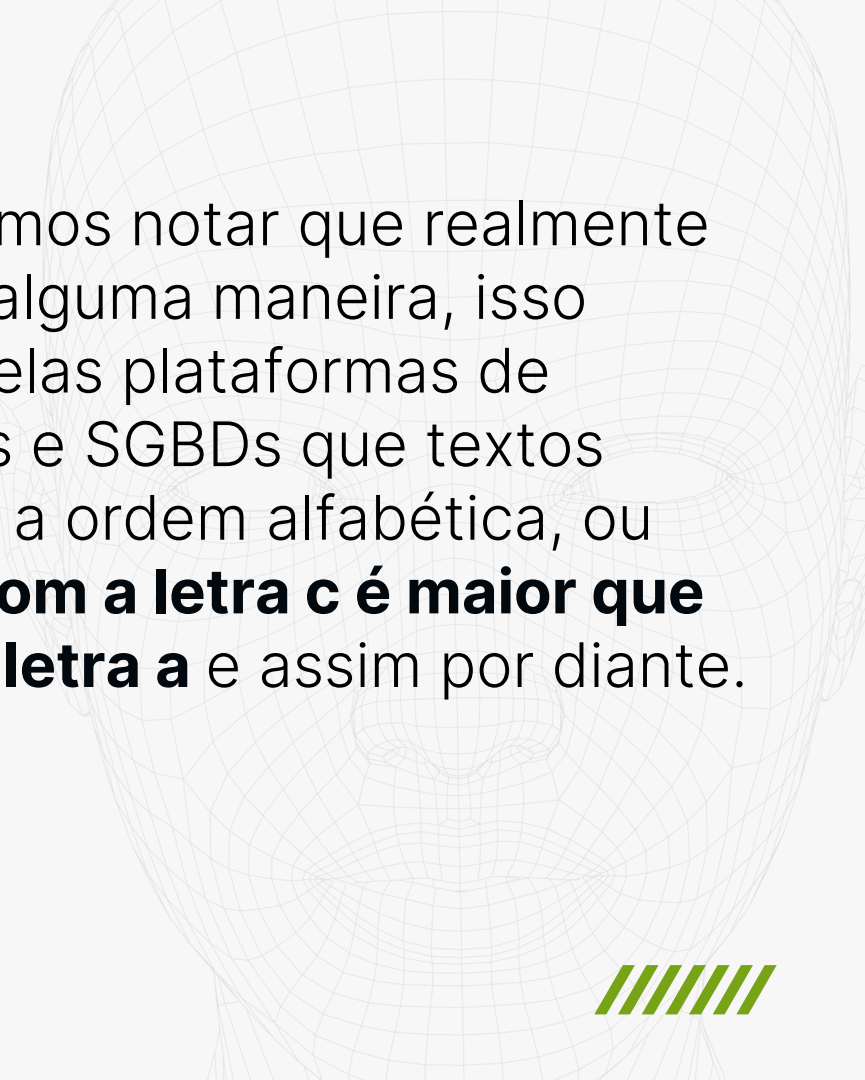
// PROBLEMÁTICA_

E como será que esses filtros se comportam caso nós os **aplicássemos em uma string ao invés de um valor numérico?**



Por exemplo, se nós quisermos filtrar os clientes que tenham como iniciais do seu nome da letra C para frente, podemos utilizar esses filtros, pois eles consideram como grandeza as letras na ordem alfabética. Vamos testar na tabela de clientes?

```
SELECT * FROM tabelapedidos WHERE  
nome_cliente > 'Carlos Pereira';
```



Rodando esse comando podemos notar que realmente os clientes foram filtrados de alguma maneira, isso acontece, pois é entendido, pelas plataformas de consultas de bancos de dados e SGBDs que textos serão filtrados de acordo com a ordem alfabética, ou seja, **um nome que comece com a letra c é maior que um nome que comece com a letra a** e assim por diante.



Da mesma forma podemos filtrar o campo de datas e o próprio SGBD entende quais as datas que são maiores, ou seja, que vem depois e menores, que vem antes.

```
SELECT * FROM tabelapedidos WHERE  
data_do_pedido > '2023-09-19';
```

Dessa forma, filtramos apenas os pedidos que foram feitos após o dia 19/09/2023. Então seja qual for o tipo de dado esses filtros terão bastante utilidade.

01000101 0110011 01100011 01101111 01101100
01100001 01000000 01100100 01100101 00100000
01000100 01100001 01100100 01101111 0110011
00001010

// Glossário SQL_

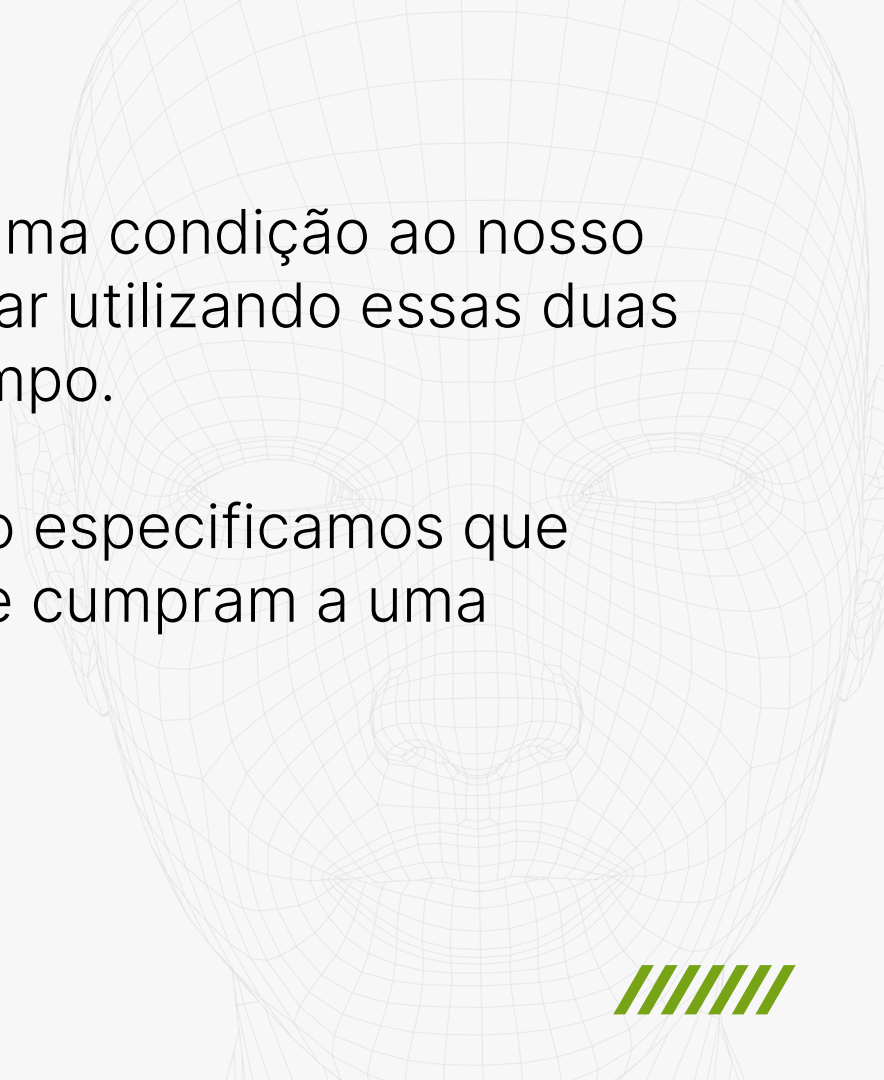
FILTROS COMPOSTOS - AND, OR, NOT E BETWEEN



Os filtros compostos nos ajudam a detalhar ainda mais nossas consultas e trazer mais especificações ao mesmo tempo. Por exemplo, como podemos fazer caso a gente queira saber apenas os pedidos com valor **maior ou igual a 200 reais** e com o **status pendente**?

```
SELECT * FROM tabelapedidos WHERE  
total_do_pedido >= 200 AND  
status = 'Pendente';
```



A faint, light gray wireframe of a human face is visible in the background on the right side of the slide.

Então, o **AND** adiciona mais uma condição ao nosso filtro, assim conseguimos filtrar utilizando essas duas especificações ao mesmo tempo.

Já o **OR** adicionarmos quando especificamos que queremos filtrar os dados que cumpram a uma especificação ou a outra.



Por exemplo, vamos filtrar todos os pedidos que ainda não foram enviados, ou seja, que estejam com o status **pendente** ou **processando**:

```
SELECT * FROM tabelapedidos WHERE  
status = 'Pendente' OR status = 'Processando' ;
```

Todos os pedidos com esses dois tipos de status foram filtrados.



Agora, se eu acrescentar o **NOT** após o **WHERE** ele irá me retornar todos os pedidos menos os pedidos com o status pendente.

```
SELECT * FROM tabelapedidos WHERE NOT  
status = 'Pendente';
```



Também temos a função **BETWEEN**, a tradução seria “entre”, então utilizamos ela quando queremos selecionar um intervalo inteiro, vamos ver na prática:

```
SELECT * FROM tabelapedidos WHERE  
data_de_envio_estimada BETWEEN  
'2023-08-01' AND '2023-09-01';
```

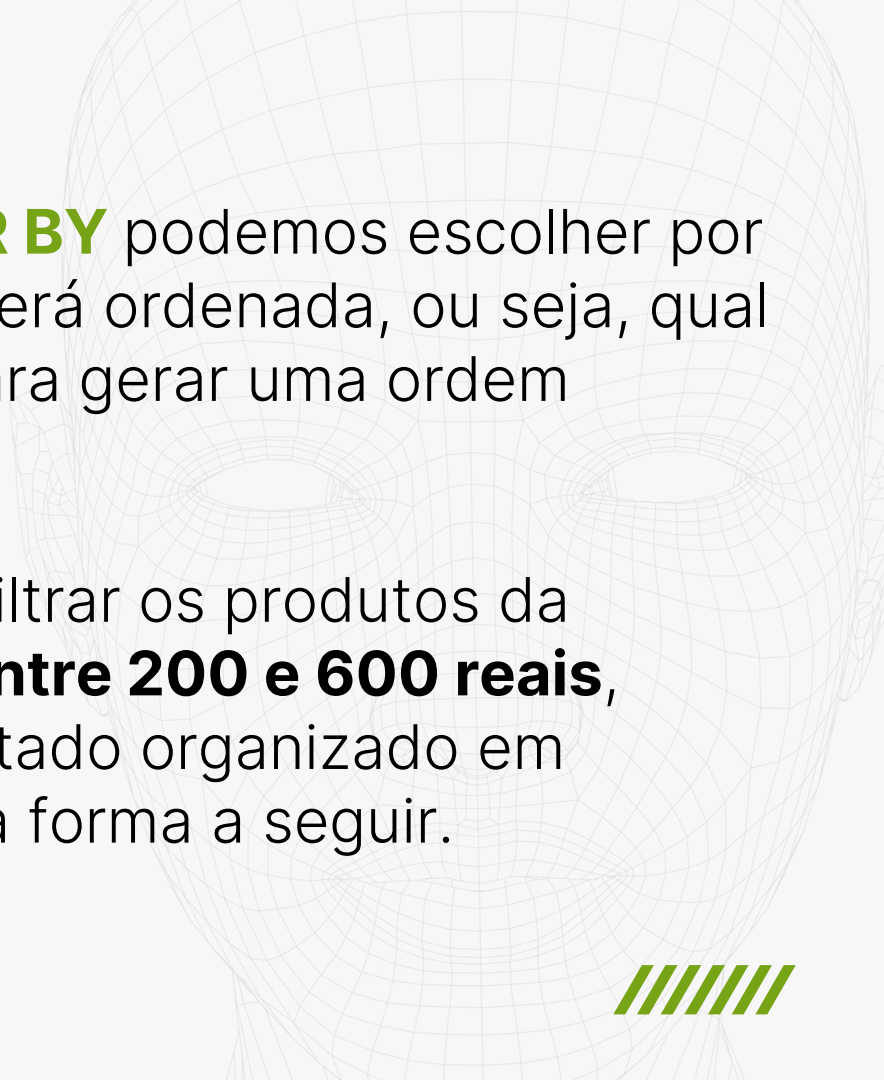
Dessa forma serão filtrados todos os pedidos que foram feitos nesse intervalo de tempo, ou seja, que estão **entre** o dia 01/08/23 e 01/09/23.

01000101 0110011 01100011 0110111 01101100
01100001 01000000 01100100 01100101 00100000
01000100 01100001 01100100 0110111 0110011
00001010

// Glossário SQL_

ORDENANDO OS DADOS - ORDER BY



A faint, light gray wireframe of a human face is visible in the background, centered on the right side of the slide.

Utilizando o comando **ORDER BY** podemos escolher por qual coluna, nossa consulta será ordenada, ou seja, qual tipo de dado será utilizado para gerar uma ordem naquela consulta.

Por exemplo, nós queremos filtrar os produtos da Hermex Import que custam **entre 200 e 600 reais**, mas queremos ver esse resultado organizado em ordem alfabética. Faremos da forma a seguir.



Selecione, da tabela de produtos, todos os produtos que tenham o preço de compra entre R\$200 e R\$600 e ordene esse resultado pelo nome do produto.

```
SELECT * FROM tabelaprodutos  
WHERE preco_de_compra BETWEEN 200 AND 600  
ORDER BY nome_produto;
```

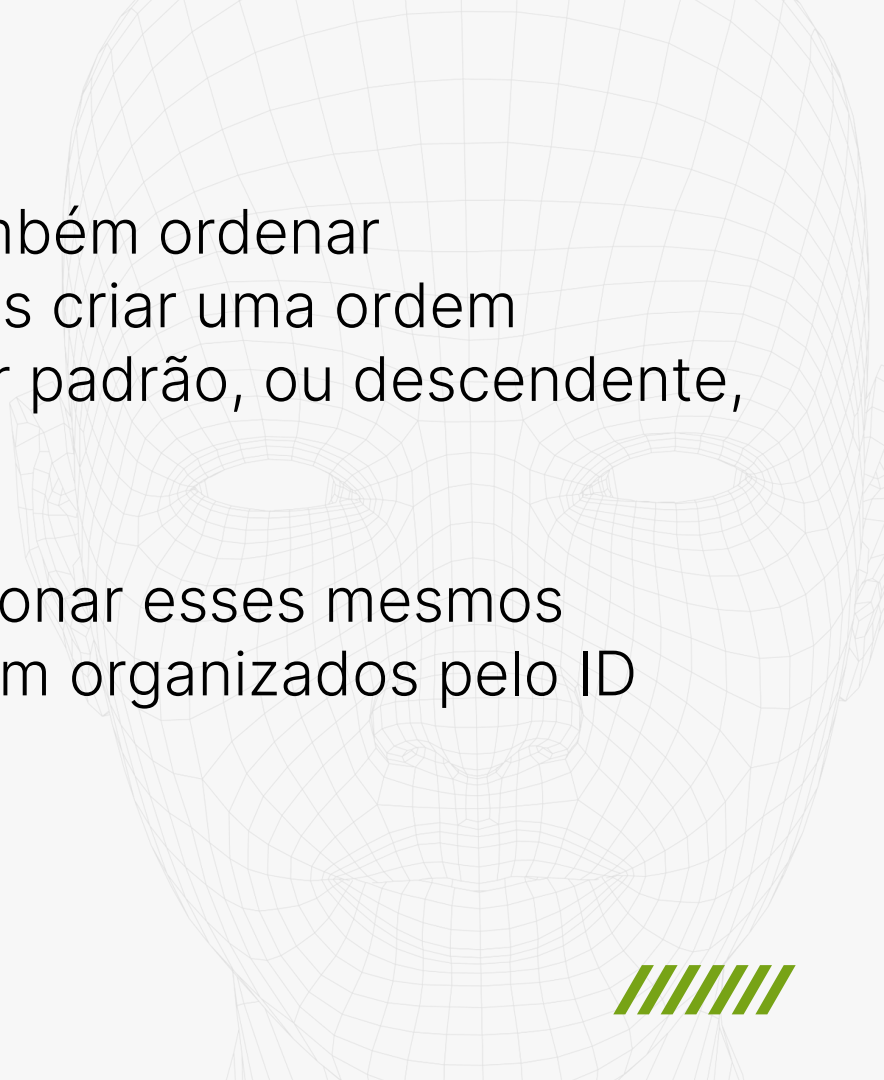


Poderíamos ordenar esses produtos também pela sua data de inclusão:

```
SELECT * FROM tabelaprodutos  
WHERE preco_de_compra BETWEEN 200 AND 600  
ORDER BY data_de_inclusa;
```

Assim, eles seriam retornados de acordo com a ordem em que foram incluídos.



A faint, light green wireframe of a human face is visible in the background, centered on the right side of the slide.

Além disso, nós podemos também ordenar numericamente. Nós podemos criar uma ordem ascendente, que já é feita por padrão, ou decendente, utilizando o **DESC**.

Por exemplo, podemos selecionar esses mesmos produtos mas pedir para serem organizados pelo ID do fornecedor. Veja a seguir.



```
SELECT * FROM tabelaprodutos  
WHERE preco_de_compra BETWEEN 200 AND 600  
ORDER BY fornecedor_produto;
```

A mesma consulta acrescentando o **DESC**, para que os IDs venham organizados em ordem decrescente:

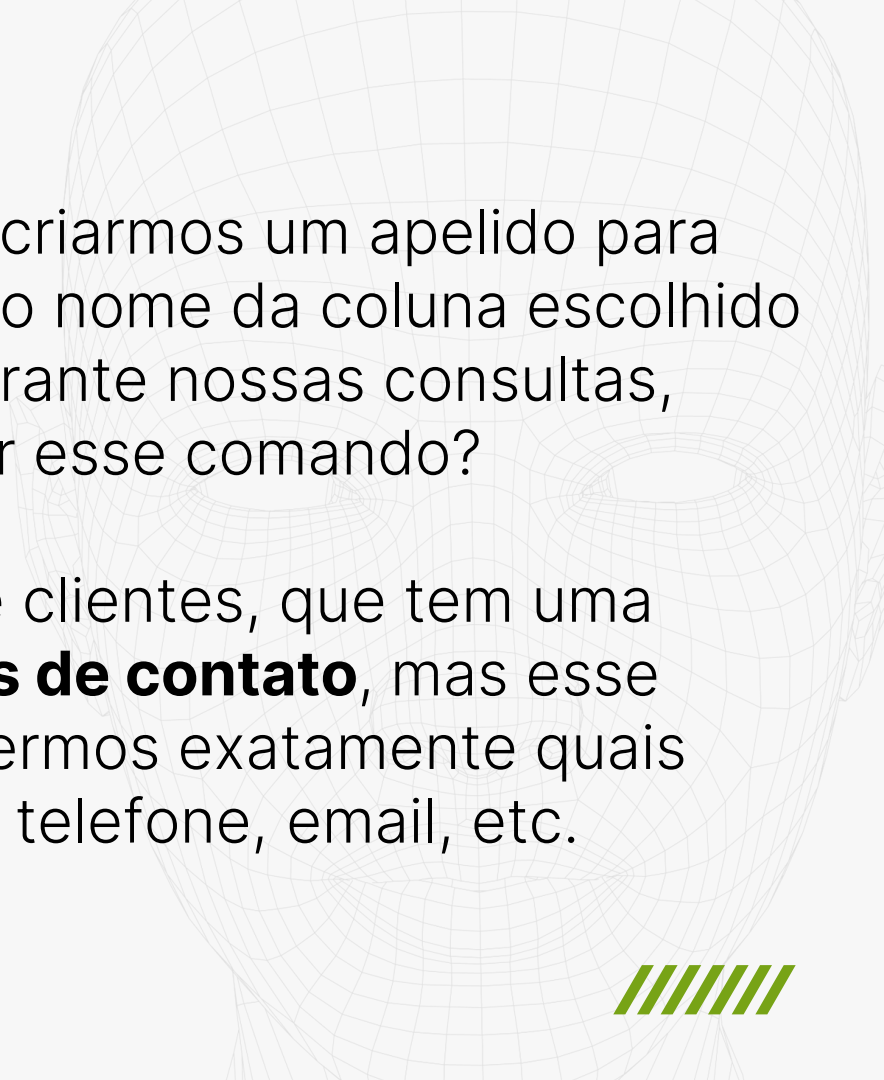
```
SELECT * FROM tabelaprodutos  
WHERE preco_de_compra BETWEEN 200 AND 600  
ORDER BY fornecedor_produto DESC;
```

01000101 0110011 01100011 0110111 01101100
01100001 01000000 01100100 01100101 00100000
01000100 01100001 01100100 0110111 0110011
00001010

// Glossário SQL_

UTILIZANDO ALIAS



A faint, light gray wireframe of a human face is visible in the background, centered on the right side of the slide.

O **ALIAS** nada mais é do que criarmos um apelido para nossas colunas, dessa forma o nome da coluna escolhido por nós é o que aparecerá durante nossas consultas, então vamos ver como utilizar esse comando?

Vamos pegar nossa tabela de clientes, que tem uma coluna chamada **informações de contato**, mas esse nome fica meio vago pra sabermos exatamente quais informações ela tem, se é um telefone, email, etc.



Mas observando os dados a gente vê que é o e-mail, então vamos criar um **ALIAS**, ou seja, um apelido pra essa coluna. Colocaremos o apelido de **email_cliente**, assim fica mais tranquilo pra sabermos qual tipo de informação de contato temos nessa coluna.

```
SELECT informacoes_de_contato AS  
email_cliente FROM tabelaclientes;
```



01000101 0110011 01100011 01101111 01101100
01100001 01000000 01100100 01100101 00100000
01000100 01100001 01100100 01101111 0110011
00001010

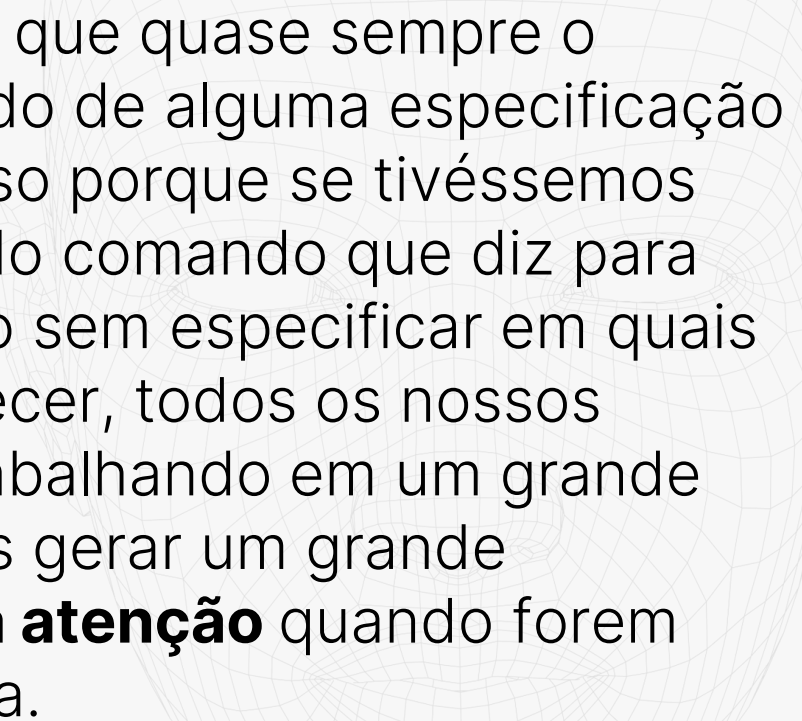
// Glossário SQL_

ATUALIZANDO E EXCLUINDO DADOS



Utilizamos o comando **UPDATE** para atualizar dados contidos em nossas tabelas, por exemplo, na tabela de pedidos, temos alguns pedidos que estão com o **status processando**, mas agora precisamos atualizar todos os campos com essa informação para o **status enviado**, o código vai ficar da seguinte maneira:

```
UPDATE tabelapedidos SET status =  
'Enviado' WHERE status = 'Processando';
```

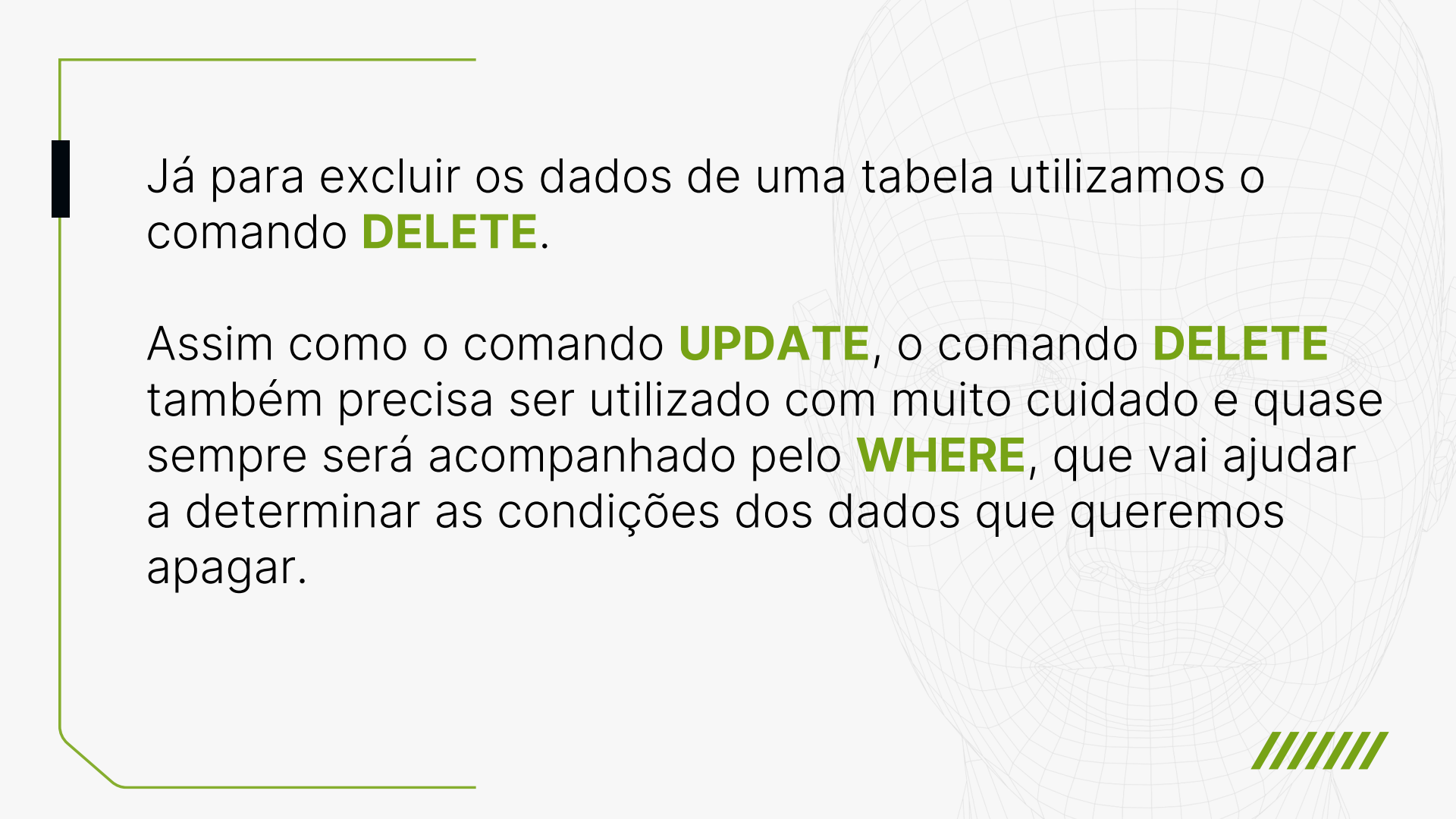


Uma coisa muito importante é que quase sempre o comando **UPDATE** vem seguido de alguma especificação condicionada pelo **WHERE**, isso porque se tivéssemos colocado apenas até a parte do comando que diz para atualizar o status para enviado sem especificar em quais condições isso deveria acontecer, todos os nossos pedidos seriam alterados e trabalhando em um grande banco de dados isso pode nos gerar um grande problema então sempre **muita atenção** quando forem alterar os dados de uma tabela.

Também é possível **alterar mais de um campo** de uma só vez. Imagine que temos um cliente que mudou de e-mail e endereço e precisamos fazer essa alteração:

```
UPDATE tabelaclientes  
SET informacoes_de_contato = 'j.santos@email.com',  
    endereço_cliente = 'Rua dos paralelepípedos, 30'  
WHERE id_cliente = 2;
```

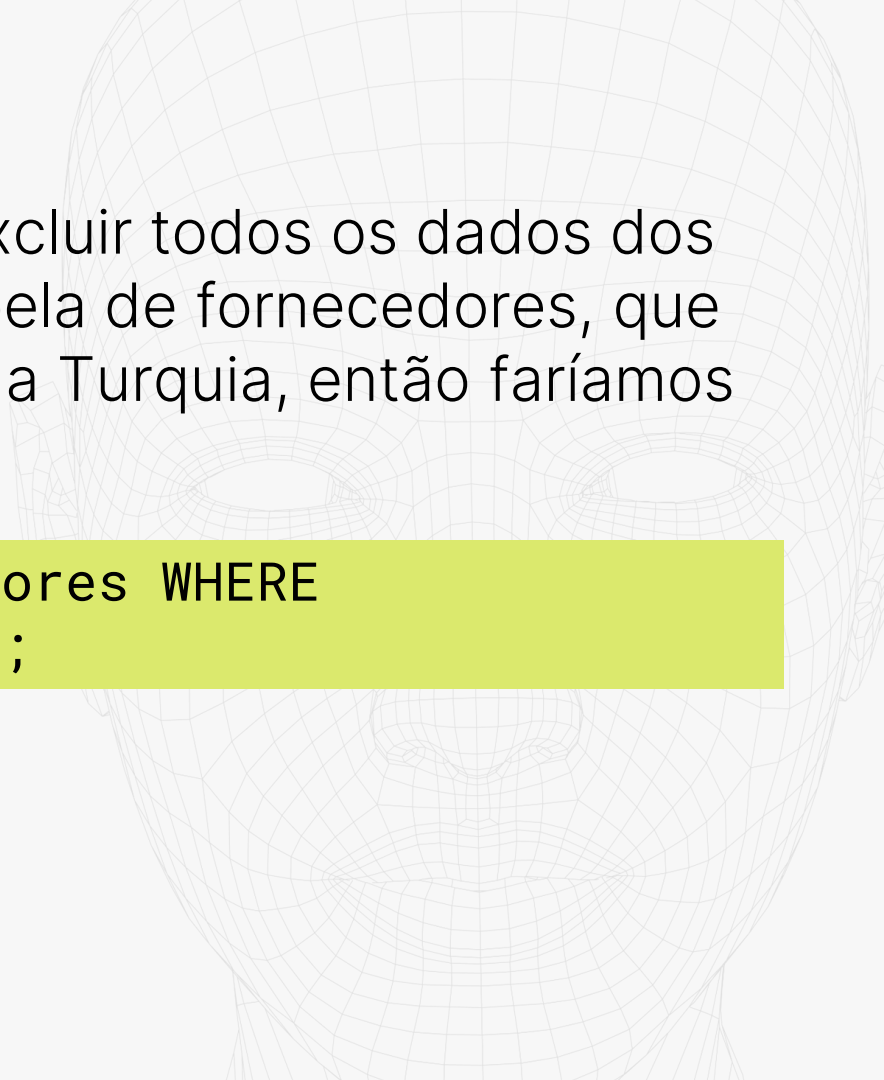
Utilizamos o **ID do cliente** para identificar em qual cliente gostaríamos de fazer a alteração.

A faint, light green wireframe of a human face is visible in the background, centered and slightly to the right. It consists of a grid of lines forming the facial structure.

Já para excluir os dados de uma tabela utilizamos o comando **DELETE**.

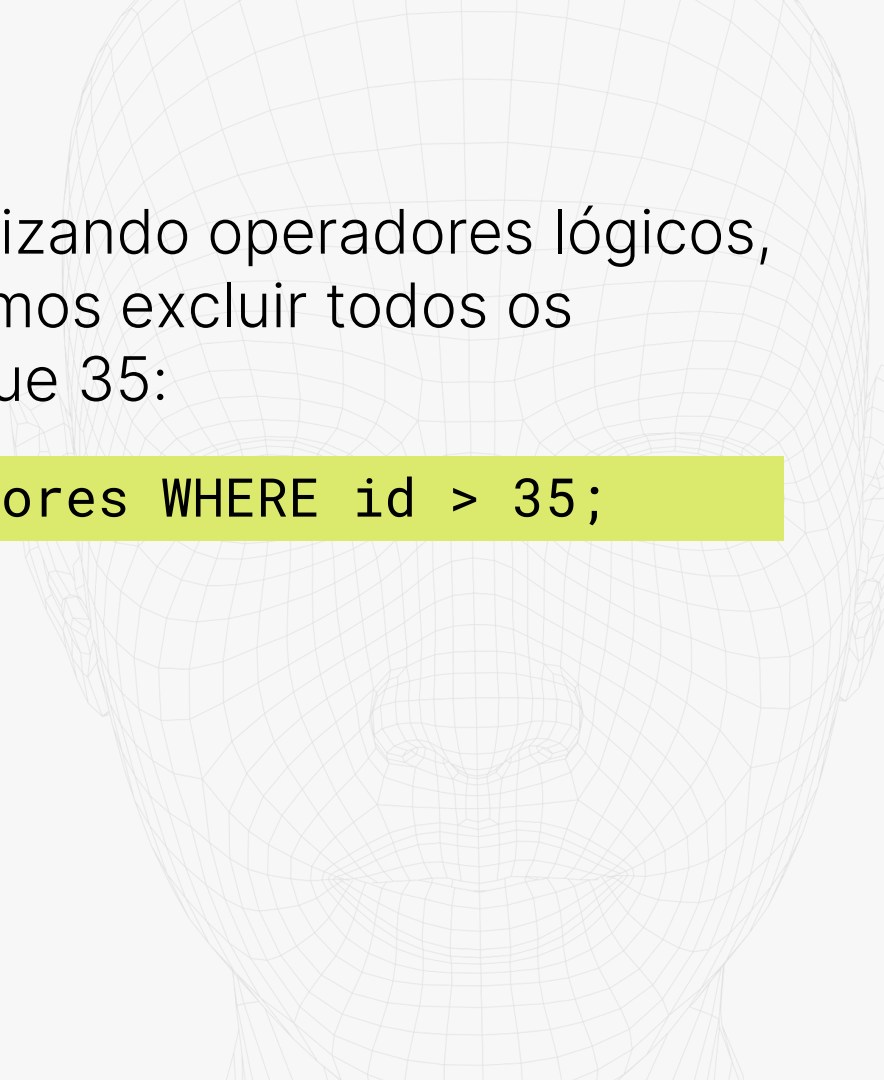
Assim como o comando **UPDATE**, o comando **DELETE** também precisa ser utilizado com muito cuidado e quase sempre será acompanhado pelo **WHERE**, que vai ajudar a determinar as condições dos dados que queremos apagar.



A faint, light gray wireframe of a human face is visible in the background, centered on the right side of the slide. It consists of a grid of lines forming the facial structure, including the eyes, nose, and mouth.

Por exemplo, se quisermos excluir todos os dados dos fornecedores contidos na tabela de fornecedores, que tenham como país de origem a Turquia, então faríamos da seguinte maneira:

```
DELETE FROM tabelaforneecedores WHERE  
país_de_origem = 'Turquia';
```

A faint, light gray wireframe of a human face is visible in the background, centered on the right side of the slide. It consists of a grid of lines forming the facial structure.

Também podemos excluir utilizando operadores lógicos, como por exemplo, se quisermos excluir todos os fornecedores com ID maior que 35:

```
DELETE FROM tabelaforneecedores WHERE id > 35;
```

UTILIZE E DOMINE O SQL!

Parabéns por explorar o Glossário de SQL! Agora que você adquiriu os fundamentos essenciais, é hora de aplicar esse conhecimento. Utilize este material como referência em seus projetos e desafios, praticando para aprimorar suas habilidades na manipulação de bancos de dados. Ao se tornar mais confiante na linguagem SQL, você estará preparado para enfrentar novos desafios.

Muito obrigado por chegar até aqui e nos vemos nos próximos cursos da formação em SQL da Alura. Até mais!

