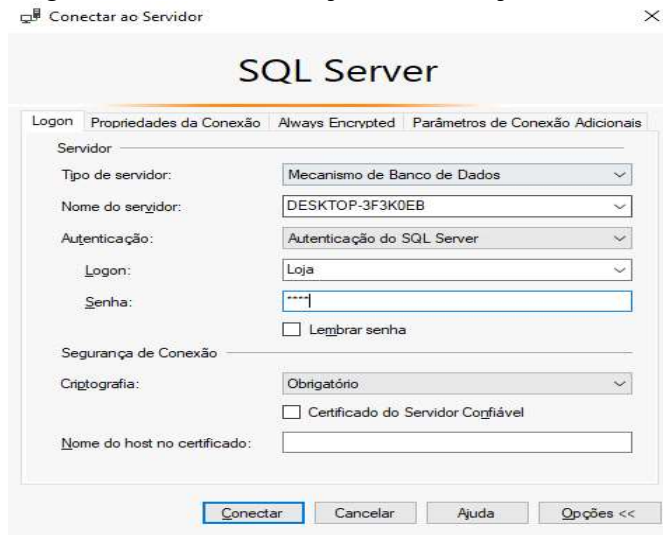
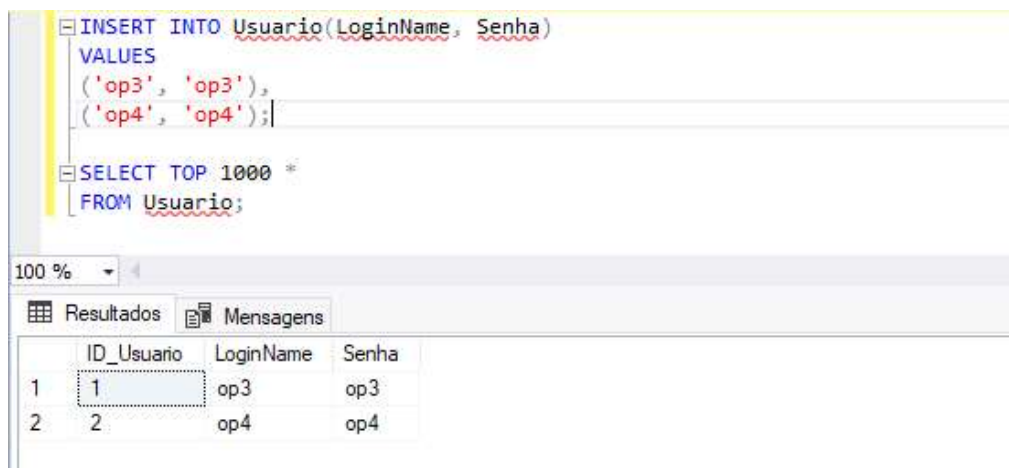


2º Procedimento | Alimentando a Base de dados

- Utilizar o SQL Server Management Studio para alimentar as tabelas com dados básicos do sistema:
- Logar como usuário **loja**, senha **loja**.



The screenshot shows the 'Conectar ao Servidor' (Connect to Server) dialog box in SQL Server Enterprise Manager. The 'Logon' tab is selected. The 'Tipo de servidor' (Server type) is set to 'Mecanismo de Banco de Dados' (Database Engine). The 'Nome do servidor' (Server name) is 'DESKTOP-3F3K0EB'. The 'Autenticação' (Authentication) is set to 'Autenticação do SQL Server' (SQL Server Authentication). The 'Logon' (Username) is 'Loja' and the 'Senha' (Password) is masked with four asterisks. There is an unchecked checkbox for 'Lembrar senha' (Remember password). Under 'Segurança de Conexão' (Connection Security), 'Criptografia' (Encryption) is set to 'Obrigatório' (Required), and there is an unchecked checkbox for 'Certificado do Servidor Confiável' (Trusted Server Certificate). The 'Nome do host no certificado' (Certificate host name) field is empty. At the bottom are buttons for 'Conectar' (Connect), 'Cancelar' (Cancel), 'Ajuda' (Help), and 'Opções <<' (Options <<).



The screenshot shows the SQL Server Enterprise Manager interface. The SQL Editor window contains the following query:

```
INSERT INTO Usuario(LoginName, Senha)
VALUES
('op3', 'op3'),
('op4', 'op4');

SELECT TOP 1000 *
FROM Usuario;
```

The query has been executed successfully. The 'Resultados' (Results) tab is active, showing a table with 2 rows and 4 columns: ID_Usuario, LoginName, Senha, and an unnamed column. The data is as follows:

	ID_Usuario	LoginName	Senha	
1	1	op3	op3	
2	2	op4	op4	

- Utilizar o editor de SQL para incluir dados na tabela de usuários, de

forma a obter um conjunto como o apresentado a seguir:

```

INSERT INTO Produto(nome, quantidade, precoVenda)
VALUES
('Martelo', 150, '15.00'),
('Chave de Fenda', 200, '7.50'),
('Serra', 100, '25.00');

SELECT TOP 1000 *
FROM Produto;

```

	ID_Produto	nome	quantidade	precoVenda
1	1	Martelo	150	15
2	2	Chave de Fenda	200	8
3	3	Serra	100	25

- Inserir alguns produtos na base de dados, obtendo um conjunto como o que é apresentado a seguir:

- Criar pessoas físicas e jurídicas na base de dados:
- Obter o próximo id de pessoa a partir da sequence.

```

CREATE SEQUENCE orderPessoa
AS INT
START WITH 1
INCREMENT BY 1;

```

- Incluir na tabela pessoa os dados comuns

```

INSERT INTO Pessoa(ID_Pessoa, nome, logradouro, cidade, estado, telefone, email)
VALUES
(NEXT VALUE FOR orderPessoa, 'Lukas Cauã', 'Rua Cilon Cunha Brum, 724', 'Rio de Janeiro', 'RJ', '3157-0406', 'lukascaua@gmail.com'),
(NEXT VALUE FOR orderPessoa, 'Maria Silva', 'Rua das Flores, 123', 'São Paulo', 'SP', '4002-8922', 'maria.silva@gmail.com'),
(NEXT VALUE FOR orderPessoa, 'José Santos', 'Avenida Paulista, 1000', 'São Paulo', 'SP', '4500-1234', 'jose.santos@gmail.com'),
(NEXT VALUE FOR orderPessoa, 'SuperCompras', 'Rua do Comércio, 500', 'Belo Horizonte', 'MG', '3500-1234', 'contato@supercompras.com'),
(NEXT VALUE FOR orderPessoa, 'Três Irmãos', 'Avenida Central, 900', 'Porto Alegre', 'RS', '3100-5678', 'contato@tresi.com');

SELECT TOP 1000 *
FROM Pessoa;

```

	ID_Pessoa	nome	logradouro	cidade	estado	telefone	email
1	1	Lukas Cauã	Rua Cilon Cunha Brum, 724	Rio de Janeiro	RJ	3157-0406	lukascaua@gmail.com
2	2	Maria Silva	Rua das Flores, 123	São Paulo	SP	4002-8922	maria.silva@gmail.com
3	3	José Santos	Avenida Paulista, 1000	São Paulo	SP	4500-1234	jose.santos@gmail.com
4	4	SuperCompras	Rua do Comércio, 500	Belo Horizonte	MG	3500-1234	contato@supercompras.com
5	5	Três Irmãos	Avenida Central, 900	Porto Alegre	RS	3100-5678	contato@tresi.com

```

INSERT INTO PessoaFisica(ID_PessoaFisica, Pessoa_ID_Pessoa, CPF)
VALUES
((NEXT VALUE FOR orderPessoa), (SELECT TOP 1 ID_Pessoa FROM Pessoa WHERE nome = 'Lukas Cauã'), '12345678901'),
((NEXT VALUE FOR orderPessoa), (SELECT TOP 1 ID_Pessoa FROM Pessoa WHERE nome = 'Maria Silva'), '98765432100'),
((NEXT VALUE FOR orderPessoa), (SELECT TOP 1 ID_Pessoa FROM Pessoa WHERE nome = 'José Santos'), '19283746509');
GO
SELECT TOP 1000 *
FROM PessoaFisica;

```

	ID_PessoaFisica	Pessoa_ID_Pessoa	CPF
1	1	1	12345678901
2	2	2	98765432100
3	3	3	19283746509

- Incluir em pessoa física o CPF...,

```

SELECT p.*, pf.CPF
FROM Pessoa p
INNER JOIN PessoaFisica pf ON p.ID_Pessoa = pf.Pessoa_ID_Pessoa;

```

	ID_Pessoa	nome	logradouro	cidade	estado	telefone	email	CPF
1	1	Lukas Cauã	Rua Cilon Cunha Brum, 724	Rio de Janeiro	RJ	3157-0406	lukascaua@gmail.com	12345678901
2	2	Maria Silva	Rua das Flores, 123	São Paulo	SP	4002-8922	maria.silva@gmail.com	98765432100
3	3	José Santos	Avenida Paulista, 1000	São Paulo	SP	4500-1234	jose.santos@gmail.com	19283746509

...efetuando o relacionamento com pessoa.

```

INSERT INTO PessoaJuridica(ID_PessoaJuridica, Pessoa_ID_Pessoa, CNPJ)
VALUES
((NEXT VALUE FOR orderPessoa), (SELECT TOP 1 ID_Pessoa FROM Pessoa WHERE nome = 'SuperCompras'), '11222333000145'),
((NEXT VALUE FOR orderPessoa), (SELECT TOP 1 ID_Pessoa FROM Pessoa WHERE nome = 'Três Irmãos'), '22334455000167');
GO
SELECT TOP 1000 *
FROM PessoaJuridica;

```

	ID_PessoaJuridica	Pessoa_ID_Pessoa	CNPJ
1	1	4	11222333000145
2	2	5	22334455000167

- Incluir em pessoa jurídica o CNPJ...,

...efetuando o relacionando com pessoa.

```

SELECT p.*, pj.CNPJ
FROM Pessoa p
INNER JOIN PessoaJuridica pj ON p.ID_Pessoa = pj.Pessoa_ID_Pessoa;

SELECT TOP 1000 *
FROM PessoaJuridica;

```

	ID_Pessoa	nome	logradouro	cidade	estado	telefone	email	CNPJ
1	4	SuperCompras	Rua do Comércio, 500	Belo Horizonte	MG	3500-1234	contato@supercompras.com	11222333000145
2	5	Três Irmãos	Avenida Central, 900	Porto Alegre	RS	3100-5678	contato@tresi.com	22334455000167

- Criar algumas movimentações na base de dados, obtendo um conjunto como o que é apresentado a seguir, onde E representa Entrada e S representa Saída.

```

INSERT INTO Movimento(Pessoa_ID_Pessoa, Usuario_ID_Usuario, Produto_ID_Produto, quantidade, tipo, valorUnitario)
VALUES
((SELECT TOP 1 ID_Pessoa FROM Pessoa WHERE nome = 'Lukas Cauã'), (SELECT TOP 1 ID_Usuario FROM Usuario WHERE LoginName = 'LukasCaua'), (SELECT TOP 1 ID_Produto FROM Produto WHERE nome = 'Martelo'), 15, 'E', 10.00),
((SELECT TOP 1 ID_Pessoa FROM Pessoa WHERE nome = 'Maria Silva'), (SELECT TOP 1 ID_Usuario FROM Usuario WHERE LoginName = 'MariaSilva'), (SELECT TOP 1 ID_Produto FROM Produto WHERE nome = 'Chave de Fenda'), 10, 'S', 5.00),
((SELECT TOP 1 ID_Pessoa FROM Pessoa WHERE nome = 'José Santos'), (SELECT TOP 1 ID_Usuario FROM Usuario WHERE LoginName = 'JoseSantos'), (SELECT TOP 1 ID_Produto FROM Produto WHERE nome = 'Serra'), 5, 'E', 20.00);

SELECT TOP 1000 *
FROM Movimento;

```

	ID_Movimento	Pessoa_ID_Pessoa	Usuario_ID_Usuario	Produto_ID_Produto	quantidade	tipo	valorUnitario
1	4	1	1	1	10	E	15.00
2	5	1	1	1	15	E	10.00
3	6	2	2	2	10	S	5.00
4	7	3	1	3	5	E	20.00

- Efetuar as seguintes consultas sobre os dados inseridos:

```

SELECT p.*, pf.CPF
FROM Pessoa p
INNER JOIN PessoaFisica pf ON p.ID_Pessoa = pf.Pessoa_ID_Pessoa;

```

	ID_Pessoa	nome	logradouro	cidade	estado	telefone	email
1	1	Lukas Cauã	Rua Cilon Cunha Brum, 724	Rio de Janeiro	RJ	3157-0406	lukascaua@gmail.co
2	2	Maria Silva	Rua das Flores, 123	São Paulo	SP	4002-8922	maria.silva@gmail.co
3	3	José Santos	Avenida Paulista, 1000	São Paulo	SP	4500-1234	jose.santos@gmail.co

Dados completos de pessoas físicas.

```
SELECT p.*, pj.CNPJ
FROM Pessoa p
INNER JOIN PessoaJuridica pj ON p.ID_Pessoa = pj.Pessoa_ID_Pessoa;
```

	ID_Pessoa	nome	logradouro	cidade	estado	telefone	email
1	4	SuperCompras	Rua do Comércio, 500	Belo Horizonte	MG	3500-1234	contato@supercompr
2	5	Três Irmãos	Avenida Central, 900	Porto Alegre	RS	3100-5678	contato@tresi.com

- Dados completos de pessoas jurídicas.

- Movimentações de entrada, com produto, fornecedor, quantidade, preço unitário e valor total.

```
SELECT m.*, p.nome AS fornecedor, pr.nome AS Produto, m.quantidade, m.valorUnitario, (m.quantidade * m.valorUnitario) AS total
FROM Movimento m INNER JOIN Pessoa p ON p.ID_Pessoa = m.Pessoa_ID_Pessoa
INNER JOIN Produto pr ON pr.ID_Produto = m.Produto_ID_Produto WHERE m.tipo = 'E';
```

	ID_Movimento	Pessoa_ID_Pessoa	Usuario_ID_Usuario	Produto_ID_Produto	quantidade	tipo	valorUnitario	fornecedor	Produto	quantidade
1	4	1	1	1	10	E	15.00	Lukas Cauã	Martelo	10
2	5	1	1	1	15	E	10.00	Lukas Cauã	Martelo	15
3	7	3	1	3	5	E	20.00	José Santos	Serra	5

```
SELECT m.*, p.nome AS comprador, pr.nome AS Produto, m.quantidade, m.valorUnitario, (m.quantidade * m.valorUnitario) AS total
FROM Movimento m
INNER JOIN Pessoa p ON p.ID_Pessoa = m.Pessoa_ID_Pessoa
INNER JOIN Produto pr ON pr.ID_Produto = m.Produto_ID_Produto WHERE m.tipo = 'S';
```

	ID_Movimento	Pessoa_ID_Pessoa	Usuario_ID_Usuario	Produto_ID_Produto	quantidade	tipo	valorUnitario	comprador	Produto
1	6	2	2	2	10	S	5.00	Maria Silva	Chave de

- Movimentações de saída, com produto, comprador, quantidade, preço unitário e valor total.
- Valor total das entradas agrupadas por produto.

```
SELECT pr.nome AS Produto, SUM(m.quantidade * m.valorUnitario) AS ValorTotalEntradas
FROM Movimento m
INNER JOIN Produto pr ON pr.ID_Produto = m.Produto_ID_Produto
WHERE m.tipo = 'E' GROUP BY pr.nome;
```

	Produto	ValorTotalEntradas
1	Martelo	300.00
2	Serra	100.00

- Valor total das saídas agrupadas por produto.


```

SELECT pr.nome AS Produto, SUM(m.quantidade * m.valorUnitario)
AS ValorTotalSaidas FROM Movimento m
INNER JOIN Produto pr ON pr.ID_Produto = m.Produto_ID_Produto
WHERE m.tipo = 'S' GROUP BY pr.nome;

```

	Produto	ValorTotalSaidas
1	Chave de Fenda	50.00

- Operadores que não efetuaram movimentações de entrada (compra).

```

SELECT u.ID_Usuario, u.LoginName FROM Usuario u
LEFT JOIN Movimento m ON u.ID_Usuario = m.Usuario_ID_Usuario AND m.tipo = 'E'
WHERE m.ID_Movimento IS NULL;

```

	ID_Usuario	LoginName
1	2	op4

- Valor total de entrada, agrupado por operador.

```

SELECT u.ID_Usuario, u.LoginName, SUM(m.quantidade * m.valorUnitario) AS
ValorTotalEntradas FROM Movimento m
INNER JOIN Usuario u ON u.ID_Usuario = m.Usuario_ID_Usuario
WHERE m.tipo = 'E'
GROUP BY u.ID_Usuario, u.LoginName;

```

	ID_Usuario	LoginName	ValorTotalEntradas
1	1	op3	400.00

- Valor total de saída, agrupado por operador.

```

SELECT u.ID_Usuario,u.LoginName,
SUM(m.quantidade * m.precoUnitario) AS ValorTotalSaidas
FROM Movimento m
INNER JOIN Usuario u ON u.ID_Usuario = m.Usuario_ID_Usuario
WHERE m.tipo = 'S' GROUP BY u.ID_Usuario, u.LoginName;

```

ID_Usuario	LoginName	ValorTotalEntradas
1	op3	400.00

- Valor médio de venda por produto, utilizando média ponderada.

```

SELECT pr.nome AS Produto,SUM(m.quantidade * m.valorUnitario) / SUM(m.quantidade)
AS ValorMedioVenda FROM Movimento m
INNER JOIN Produto pr ON pr.ID_Produto = m.Produto_ID_Produto
WHERE m.tipo = 'S' GROUP BY pr.nome;

```

Produto	ValorMedioVenda
1 Chave de Fenda	5.000000

Análise e Conclusão do 2º Procedimento:

- Quais as diferenças no uso de sequence e identity?

As diferenças principais no uso de *SEQUENCE* e *IDENTITY* em SQL estão relacionadas ao seu funcionamento, sintaxe e suporte em diferentes sistemas de gerenciamento de banco de dados (SGBDs). Aqui estão os pontos chave:

SEQUENCE: Oferece mais flexibilidade porque pode ser compartilhada por várias tabelas e permite mais controle sobre como os valores são obtidos e utilizados.

IDENTITY: É mais simples de implementar e usar em comparação com sequence, especialmente quando se trata de tabelas simples que precisam de uma chave primária auto-incremental.

As escolha entre *SEQUENCE* e *IDENTITY* depende das necessidades

específicas do banco de dados e das características de implementação desejadas. Por exemplo, se você precisa de valores únicos que possam ser compartilhados entre várias tabelas, *SEQUENCE* é mais adequada. Se você precisa de uma chave primária auto-incremental simples em uma única tabela, *IDENTITY* pode ser mais conveniente.

Em resumo, ambas as opções são úteis para a geração automática de valores únicos em bancos de dados relacionais, mas cada uma tem suas próprias características e casos de uso específicos.

- **Qual a importância das chaves estrangeiras para a consistência do banco?**

As chaves estrangeiras (*foreign keys*) desempenham um papel crucial na garantia da consistência e integridade dos dados em um banco de dados relacional. Aqui estão as principais razões pelas quais as chaves estrangeiras são importantes:

- *Manutenção da Integridade Referencial:*
- *Evita Inconsistências de Dados:*
- *Facilita a Manutenção e Gestão de Dados:*
- *Suporte a Operações Transacionais:*
- *Documentação e Entendimento do Modelo de Dados:*

Em resumo, as chaves estrangeiras são fundamentais para garantir que as relações entre dados em um banco de dados relacional sejam consistentes, corretas e seguras. Elas ajudam a evitar erros de integridade referencial, simplificam operações de manutenção e consultas complexas, e contribuem para a confiabilidade e eficiência do sistema de gerenciamento de banco de dados como um todo.

- **Quais operadores do SQL pertencem à álgebra relacional e quais são definidos no cálculo relacional?**

No contexto de bancos de dados relacionais, os operadores podem ser classificados em duas principais abordagens teóricas: álgebra relacional e cálculo relacional. Aqui estão os operadores típicos de cada uma dessas abordagens:

Álgebra Relacional:

- **Seleção (Selection):**
 - Operador representado por σ .
 - Seleciona linhas da tabela que satisfazem uma condição específica.
- **Projeção (Projection):**

- Operador representado por π .
- Seleciona colunas específicas da tabela.
- **União (Union):**
 - Operador representado por \cup .
 - Combina duas relações (tabelas) que têm o mesmo conjunto de colunas.
- **Interseção (Intersection):**
 - Operador representado por \cap .
 - Retorna as linhas que estão presentes em ambas as relações.
- **Diferença (Difference):**
 - Operador representado por $-$ (ou \setminus).
 - Retorna as linhas que estão na primeira relação, mas não na segunda.
- **Produto Cartesiano (Cartesian Product):**
 - Operador representado por \times .
 - Combina cada linha de uma relação com cada linha de outra relação.
- **Junção (Join):**
 - Operador que combina colunas de duas ou mais tabelas com base em uma condição relacionada.

Cálculo Relacional:

O cálculo relacional não possui operadores tão explicitamente definidos quanto a álgebra relacional. Em vez disso, ele utiliza expressões e predicados para especificar o que deve ser recuperado da base de dados. Alguns conceitos no cálculo relacional incluem:

- **Cálculo de Tupla (Tuple Calculus):**
 - Especifica as tuplas (linhas) que satisfazem uma determinada condição.
 - Utiliza variáveis para descrever as condições e expressões para filtrar e selecionar as tuplas.
- **Cálculo de Domínio (Domain Calculus):**
 - Especifica os valores de atributo (coluna) que satisfazem uma determinada condição.
 - Também utiliza variáveis e expressões para definir as condições de seleção.

O cálculo relacional, por outro lado, define uma linguagem mais abrangente para descrever e manipular dados em um banco de dados relacional. Ele utiliza fórmulas matemáticas para expressar consultas complexas, incluindo quantificadores e variáveis. No SQL, alguns operadores são influenciados pelo cálculo relacional, mas a implementação completa não se limita a ele.

- *Agregação: Funções como COUNT, SUM, AVG, MIN e MAX são utilizadas para resumir dados em grupos de tuplas, retornando valores agregados. No SQL, essas funções são geralmente usadas em conjunto com a cláusula GROUP BY.*
- *Aninhamento de Consultas: O SQL permite aninhar consultas umas dentro das outras, utilizando subconsultas, o que possibilita realizar operações mais complexas e combinações de dados de diferentes relações.*

Nem todos os operadores da álgebra relacional possuem uma representação direta no SQL. Algumas operações podem ser implementadas através de combinações de outros operadores ou utilizando recursos mais avançados da linguagem.

O cálculo relacional fornece uma base teórica mais poderosa para a manipulação de dados, mas o SQL oferece uma sintaxe mais prática e intuitiva para a maioria dos usuários.

• **Como é feito o agrupamento em consultas, e qual requisito é obrigatório?**

O agrupamento em consultas SQL é feito usando a cláusula GROUP BY. Esta cláusula permite agrupar linhas que possuem valores iguais em uma ou mais colunas especificadas, geralmente para aplicar funções de agregação como SUM, COUNT, AVG, MAX, MIN sobre grupos de dados. Sintaxe Básica do "GROUP BY":

```
SELECT coluna1, coluna2, ..., funcao_agregacao(coluna)
FROM tabela
GROUP BY coluna1, coluna2, ...
```

Requisitos Obrigatórios para o GROUP BY:

Para utilizar o GROUP BY de forma correta e eficaz, é fundamental seguir dois requisitos:

- **Especificar Colunas de Agrupamento:**
 - *Ao usar GROUP BY, você precisa especificar quais colunas serão usadas para agrupar os dados. Isso significa que todas as colunas na lista SELECT que não são funções de agregação devem estar presentes na lista GROUP BY.*
 - *Por exemplo, se você selecionar coluna1, coluna2 e aplicar SUM(coluna3), então coluna1 e coluna2 devem estar presentes na cláusula GROUP BY.*
- **Restrições na Seleção de Colunas:**
 - *Quando você usa funções de agregação (SUM, COUNT, AVG, etc.) em uma consulta com GROUP BY, geralmente não pode selecionar colunas individuais que não estão incluídas na cláusula GROUP BY (a menos que elas estejam dentro de uma função de agregação).*

- Por exemplo, se você agrupa por *coluna1* e *coluna2*, e deseja contar quantos registros existem em cada grupo, você pode fazer *COUNT(*)* ou *COUNT(coluna_qualquer)*, mas não pode selecionar *coluna3* diretamente a menos que seja uma função de agregação.

Exemplo Prático:

Considere uma tabela *pedidos* com as colunas *id_pedido*, *id_cliente*, *data_pedido* e *valor_pedido*. Se quisermos calcular o total de pedidos por cliente:

```
SELECT id_cliente, COUNT(*) AS total_pedidos  
FROM pedidos  
GROUP BY id_cliente;
```

Neste exemplo:

- Estamos agrupando os pedidos pelo *id_cliente*.
- A função de agregação *COUNT(*)* conta quantos registros (pedidos) existem para cada *id_cliente*.
- *id_cliente* é especificado na cláusula *GROUP BY*, pois estamos agrupando pelos valores únicos dessa coluna.

Portanto, o requisito obrigatório ao usar *GROUP BY* em consultas SQL é especificar quais colunas estão sendo usadas para agrupar os dados na cláusula *GROUP BY*. Isso garante que a operação de agrupamento seja feita de maneira correta e que as funções de agregação sejam aplicadas aos grupos de dados desejados.