Estácio - Unidade São pedro
RJ 140 Km 2, 512 loja 1, São Pedro da Aldeia - RJ, 28941-182

Desenvolvimento Full Stack
Classe: Missão Prática | Nível 2 | Mundo 3
3º Semestre
Marvin de Almeida Costa

Título da Prática: 2º Procedimento | Alimentando a Base

Objetivo da Prática:

1. Identificar os requisitos de um sistema e transformá-los no modelo adequado.
2. Utilizar ferramentas de modelagem para bases de dados relacionais.
3. Explorar a sintaxe SQL na criação das estruturas do banco (DDL).
4. Explorar a sintaxe SQL na consulta e manipulação de dados (DML)
5. No final do exercício, o aluno terá vivenciado a experiência de modelar a base de
6. dados para um sistema simples, além de implementá-la, através da sintaxe SQL, na
7. plataforma do SQL Server.

Todos os códigos solicitados neste roteiro de aula:

-- Inserir dados

INSERT INTO Pessoa (nome, logradouro, cidade, estado, telefone, email) VALUES ('nome a', 'logradouro a', 'cidade a','es' ,'telefone a', 'email a');
INSERT INTO Pessoa (nome, logradouro, cidade, estado, telefone, email) VALUES ('Marvin', 'Rua 12', 'Riacho do Sul','PA' ,'1111-1111', 'marvin@gmail.com');
INSERT INTO Pessoa (nome, logradouro, cidade, estado, telefone, email) VALUES ('Chantili', 'Rua 12', 'Riacho do Sul','PA' ,'1112-1111', 'chan@gmail.com');
INSERT INTO Pessoa (nome, logradouro, cidade, estado, telefone, email) VALUES ('Joao', 'Rua 13', 'RJ','PA' ,'1123-1111', 'JOAOaaa@gmail.com');

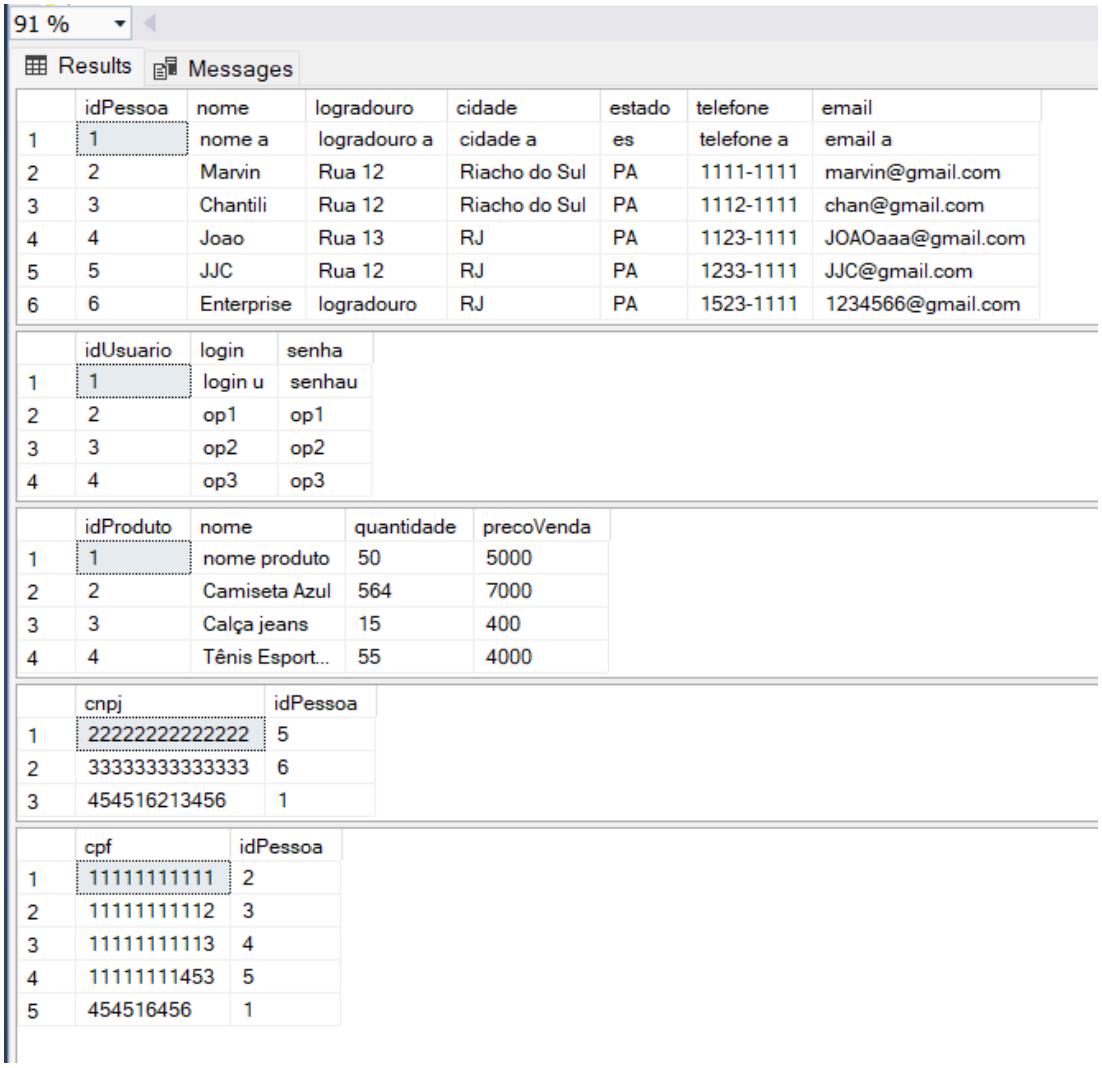
```
INSERT INTO Pessoa (nome, logradouro, cidade, estado, telefone, email) VALUES ('JJC', 'Rua 12', 'RJ','PA' ,'1233-1111', 'JJC@gmail.com');
INSERT INTO Pessoa (nome, logradouro, cidade, estado, telefone, email) VALUES ('Enterprise', 'logradouro', 'RJ','PA' ,'1523-1111', '1234566@gmail.com');

INSERT INTO Produto (nome, quantidade, precoVenda) VALUES ('nome produto', 50, 5000);
INSERT INTO Produto (nome, quantidade, precoVenda) VALUES ('Camiseta Azul', 564, 7000);
INSERT INTO Produto (nome, quantidade, precoVenda) VALUES ('Calça jeans', 15, 400);
INSERT INTO Produto (nome, quantidade, precoVenda) VALUES ('Tênis Esportivo', 55, 4000);

INSERT INTO Usuario (login, senha) VALUES ('login u', 'senhau');
INSERT INTO Usuario (login, senha) VALUES ('op1', 'op1');
INSERT INTO Usuario (login, senha) VALUES ('op2', 'op2');
INSERT INTO Usuario (login, senha) VALUES ('op3', 'op3');

INSERT INTO PessoaFisica (cpf, idPessoa) VALUES ('45451556', 1);
INSERT INTO PessoaFisica (cpf, idPessoa) VALUES ('11111111111', 2);
INSERT INTO PessoaFisica (cpf, idPessoa) VALUES ('11111111112', 3);
INSERT INTO PessoaFisica (cpf, idPessoa) VALUES ('11111111113', 4);
INSERT INTO PessoaFisica (cpf, idPessoa) VALUES ('11111111453', 5);

INSERT INTO PessoaFisica (cnpj, idPessoa) VALUES ('454162533456', 1);
INSERT INTO PessoaJuridica (cnpj, idPessoa) VALUES ('22222222222222', 5);
INSERT INTO PessoaJuridica (cnpj, idPessoa) VALUES ('33333333333333', 6);

INSERT INTO Movimento (idUsuario, idPessoa, idProduto, quantidade, tipo, valorUnitario)
VALUES (1, 1, 1, 2, 'E', 5000);
INSERT INTO Movimento (idUsuario, idPessoa, idProduto, quantidade, tipo, valorUnitario)
VALUES (2, 2, 2, 1, 'E', 1000);
INSERT INTO Movimento (idUsuario, idPessoa, idProduto, quantidade, tipo, valorUnitario)
VALUES (2, 2, 2, 1, 'E', 1000);
INSERT INTO Movimento (idUsuario, idPessoa, idProduto, quantidade, tipo, valorUnitario)
VALUES (2, 2, 2, 1, 'S', 1000);
INSERT INTO Movimento (idUsuario, idPessoa, idProduto, quantidade, tipo, valorUnitario)
VALUES (2, 2, 2, 1, 'S', 1000);
INSERT INTO Movimento (idUsuario, idPessoa, idProduto, quantidade, tipo, valorUnitario)
VALUES (2, 2, 2, 1, 'S', 1000);
INSERT INTO Movimento (idUsuario, idPessoa, idProduto, quantidade, tipo, valorUnitario)
VALUES (3, 3, 2, 1, 'E', 10000);
INSERT INTO Movimento (idUsuario, idPessoa, idProduto, quantidade, tipo, valorUnitario)
VALUES (3, 3, 3, 10, 'E', 50000);
INSERT INTO Movimento (idUsuario, idPessoa, idProduto, quantidade, tipo, valorUnitario)
VALUES (3, 3, 4, 10, 'E', 40000);
```

```
INSERT INTO Movimento (idUsuario, idPessoa, idProduto, quantidade, tipo, valorUnitario)
VALUES (2, 2, 3, 2, 'S', 4000);
INSERT INTO Movimento (idUsuario, idPessoa, idProduto, quantidade, tipo, valorUnitario)
VALUES (2, 2, 3, 2, 'S', 4000);
INSERT INTO Movimento (idUsuario, idPessoa, idProduto, quantidade, tipo, valorUnitario)
VALUES (2, 2, 4, 2, 'S', 4000);
INSERT INTO Movimento (idUsuario, idPessoa, idProduto, quantidade, tipo, valorUnitario)
VALUES (3, 3, 4, 2, 'S', 4000);

-- Dados Completos de Pessoas Físicas
SELECT * FROM PessoaFisica pf
JOIN Pessoa p ON pf.idPessoa = p.idPessoa;


-- Dados Completos de Pessoas Jurídicas
SELECT * FROM PessoaJuridica pj
JOIN Pessoa p ON pj.idPessoa = p.idPessoa;


-- Movimentações de entrada, com produto, fornecedor, quantidade, preço unitário e valor total.
SELECT m.idMovimento, p.nome AS Produto, u.login AS Fornecedor, m.quantidade,
m.valorUnitario, (m.quantidade * m.valorUnitario) AS ValorTotal
FROM Movimento m
JOIN Produto p ON m.idProduto = p.idProduto
JOIN Usuario u ON m.idUsuario = u.idUsuario
WHERE m.tipo = 'E';


-- Movimentações de saída, com produto, comprador, quantidade, preço unitário e valor total.
SELECT m.idMovimento, p.nome AS Produto, c.nome AS Comprador, m.quantidade,
m.valorUnitario, (m.quantidade * m.valorUnitario) AS ValorTotal
FROM Movimento m
JOIN Produto p ON m.idProduto = p.idProduto
JOIN Pessoa c ON m.idPessoa = c.idPessoa
WHERE m.tipo = 'S';


-- Valor total das entradas agrupadas por produto.
SELECT p.nome AS Produto, SUM((m.quantidade * m.valorUnitario)) AS ValorTotalEntrada
FROM Movimento m
JOIN Produto p ON m.idProduto = p.idProduto
WHERE m.tipo = 'E'
GROUP BY p.nome;
```

```
-- Valor total das saídas agrupadas por produto.
SELECT p.nome AS Produto, SUM((m.quantidade * m.valorUnitario)) AS ValorTotalSaida
FROM Movimento m
JOIN Produto p ON m.idProduto = p.idProduto
WHERE m.tipo = 'S'
GROUP BY p.nome;


-- Operadores que não efetuaram movimentações de entrada (compra).
SELECT u.login
FROM Usuario u
LEFT JOIN Movimento m ON u.idUsuario = m.idUsuario AND m.tipo = 'E'
WHERE m.idMovimento IS NULL;


-- Valor total de entrada, agrupado por operador.
SELECT u.login, SUM((m.quantidade * m.valorUnitario)) AS ValorTotalEntrada
FROM Movimento m
JOIN Usuario u ON m.idUsuario = u.idUsuario
WHERE m.tipo = 'E'
GROUP BY u.login;


-- Valor total de saída, agrupado por operador.
SELECT u.login, SUM((m.quantidade * m.valorUnitario)) AS ValorTotalSaida
FROM Movimento m
JOIN Usuario u ON m.idUsuario = u.idUsuario
WHERE m.tipo = 'S'
GROUP BY u.login;


-- Valor médio de venda por produto, utilizando média ponderada.
SELECT p.nome AS Produto, AVG(m.quantidade * p.precoVenda) AS MediaPonderada
FROM Movimento m
JOIN Produto p ON m.idProduto = p.idProduto
WHERE m.tipo = 'S'
GROUP BY p.nome;
```

Os resultados da execução dos códigos também devem ser apresentados;

Inserir dados



91 %

Results | Messages

	idPessoa	nome	logradouro	cidade	estado	telefone	email
1	1	nome a	logradouro a	cidade a	es	telefone a	email a
2	2	Marvin	Rua 12	Riacho do Sul	PA	1111-1111	marvin@gmail.com
3	3	Chantili	Rua 12	Riacho do Sul	PA	1112-1111	chan@gmail.com
4	4	Joao	Rua 13	RJ	PA	1123-1111	JOAOaaa@gmail.com
5	5	JJC	Rua 12	RJ	PA	1233-1111	JJC@gmail.com
6	6	Enterprise	logradouro	RJ	PA	1523-1111	1234566@gmail.com

	idUsuario	login	senha
1	1	login u	senhau
2	2	op1	op1
3	3	op2	op2
4	4	op3	op3

	idProduto	nome	quantidade	precoVenda
1	1	nome produto	50	5000
2	2	Camiseta Azul	564	7000
3	3	Calça jeans	15	400
4	4	Tênis Esport...	55	4000

	cnpj	idPessoa
1	22222222222222	5
2	33333333333333	6
3	454516213456	1

	cpf	idPessoa
1	11111111111	2
2	11111111112	3
3	11111111113	4
4	11111111453	5
5	454516456	1

	idMovimento	idUsuario	idPessoa	idProduto	quantidade	tipo	valorUnitario
1	1	1	1	1	2	E	5000
2	2	2	2	2	1	E	1000
3	3	2	2	2	1	E	1000
4	4	2	3	2	1	S	1000
5	5	2	4	2	1	S	1000
6	6	2	4	2	1	S	1000
7	7	3	6	2	10	E	10000
8	8	3	6	3	10	E	50000
9	9	3	6	4	10	E	40000
10	10	2	3	3	2	S	4000
11	11	2	4	3	2	S	4000
12	12	2	4	4	2	S	4000
13	13	3	4	4	2	S	4000

1. Dados completos de pessoas físicas.

```
--SELECT * FROM Movimento;

SELECT * FROM PessoaFisica pf JOIN Pessoa p ON pf.idPessoa = p.idPessoa;
```

91 %

Results | Messages

	cpf	idPessoa	idPessoa	nome	logradouro	cidade	estado	telefone	email
1	11111111111	2	2	Marvin	Rua 12	Riacho do Sul	PA	1111-1111	marvin@gmail.com
2	11111111112	3	3	Chantili	Rua 12	Riacho do Sul	PA	1112-1111	chan@gmail.com
3	11111111113	4	4	Joao	Rua 13	RJ	PA	1123-1111	JOAOaaa@gmail.com
4	11111111453	5	5	JJC	Rua 12	RJ	PA	1233-1111	JJC@gmail.com
5	454516456	1	1	nome a	logradouro a	cidade a	es	telefone a	email a

2. Dados completos de pessoas jurídicas.

```
SELECT * FROM PessoaJuridica pj JOIN Pessoa p ON pj.idPessoa = p.idPessoa;
```

1 %

Results | Messages

	cnpj	idPessoa	idPessoa	nome	logradouro	cidade	estado	telefone	email
1	22222222222222	5	5	JJC	Rua 12	RJ	PA	1233-1111	JJC@gmail.com
2	33333333333333	6	6	Enterprise	logradouro	RJ	PA	1523-1111	1234566@gmail.com
3	454516213456	1	1	nome a	logradouro a	cidade a	es	telefone a	email a

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

```
SELECT m.idMovimento, p.nome AS Produto, u.login AS Fornecedor, m.quantidade, m.valorUnitario, (m.quantidade * m.valorUnitario) AS ValorTotal
FROM Movimento m
JOIN Produto p ON m.idProduto = p.idProduto
JOIN Usuario u ON m.idUsuario = u.idUsuario
WHERE m.tipo = 'E';
```

1 %

Results | Messages

	idMovimento	Produto	Fornecedor	quantidade	valorUnitario	ValorTotal
1	1	nome produto	login u	2	5000	10000
2	2	Camiseta Azul	op1	1	1000	1000
3	3	Camiseta Azul	op1	1	1000	1000
4	7	Camiseta Azul	op2	10	10000	100000
5	8	Calça jeans	op2	10	50000	500000
6	9	Tênis Esportivo	op2	10	40000	400000

4. Movimentações de saída, com produto, comprador, quantidade, preço unitário e valor total.

```sql
SELECT m.idMovimento, p.nome AS Produto, c.nome AS Comprador, m.quantidade, m.valorUnitario, (m.quantidade * m.valorUnitario) AS ValorTotal
FROM Movimento m
JOIN Produto p ON m.idProduto = p.idProduto
JOIN Pessoa c ON m.idPessoa = c.idPessoa
WHERE m.tipo = 'S';
```

91 %

Results | Messages

	idMovimento	Produto	Comprador	quantidade	valorUnitario	ValorTotal
1	4	Camiseta Azul	Chantili	1	1000	1000
2	5	Camiseta Azul	Joao	1	1000	1000
3	6	Camiseta Azul	Joao	1	1000	1000
4	10	Calça jeans	Chantili	2	4000	8000
5	11	Calça jeans	Joao	2	4000	8000
6	12	Tênis Esportivo	Joao	2	4000	8000
7	13	Tênis Esportivo	Joao	2	4000	8000

5. Valor total das entradas agrupadas por produto.

```sql
SELECT p.nome AS Produto, SUM((m.quantidade * m.valorUnitario)) AS ValorTotalEntrada
FROM Movimento m
JOIN Produto p ON m.idProduto = p.idProduto
WHERE m.tipo = 'E'
GROUP BY p.nome;
```

91 %

Results | Messages

	Produto	ValorTotalEntrada
1	Calça jeans	500000
2	Camiseta Azul	102000
3	nome produto	10000
4	Tênis Esportivo	400000

6. Valor total das saídas agrupadas por produto.

```sql
SELECT p.nome AS Produto, SUM((m.quantidade * m.valorUnitario)) AS ValorTotalSaida
FROM Movimento m
JOIN Produto p ON m.idProduto = p.idProduto
WHERE m.tipo = 'S'
GROUP BY p.nome;
```

91 %

Results | Messages

	Produto	ValorTotalSaida
1	Calça jeans	16000
2	Camiseta Azul	3000
3	Tênis Esportivo	16000

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

```sql
SELECT u.login
FROM Usuario u
LEFT JOIN Movimento m ON u.idUsuario = m.idUsuario AND m.tipo = 'E'
WHERE m.idMovimento IS NULL;
```

91 %

Results | Messages

	login
1	op3

8. Valor total de entrada, agrupado por operador.

```sql
SELECT u.login, SUM((m.quantidade * m.valorUnitario)) AS ValorTotalEntrada
FROM Movimento m
JOIN Usuario u ON m.idUsuario = u.idUsuario
WHERE m.tipo = 'E'
GROUP BY u.login;
```

91 %

Results | Messages

	login	ValorTotalEntrada
1	login u	10000
2	op1	2000
3	op2	1000000

9. Valor total de saída, agrupado por operador.

```sql
SELECT u.login, SUM((m.quantidade * m.valorUnitario)) AS ValorTotalSaida
FROM Movimento m
JOIN Usuario u ON m.idUsuario = u.idUsuario
WHERE m.tipo = 'S'
GROUP BY u.login;
```

1 %

Results | Messages

	login	ValorTotalSaida
1	op1	27000
2	op2	8000

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

```sql
SELECT p.nome AS Produto, AVG(m.quantidade * p.precoVenda) AS MediaPonderada
FROM Movimento m
JOIN Produto p ON m.idProduto = p.idProduto
WHERE m.tipo = 'S'
GROUP BY p.nome;
```

91 %

Results | Messages

	Produto	MediaPonderada
1	Calça jeans	800.000000
2	Camiseta Azul	7000.000000
3	Tênis Esportivo	8000.000000

Análise e Conclusão:

Quais as diferenças no uso de sequence e identity?

No contexto do SQL, tanto SEQUENCE quanto IDENTITY são usados para gerar valores únicos automaticamente para colunas em tabelas, mas eles operam de maneiras ligeiramente diferentes e têm aplicações específicas dependendo das necessidades do seu banco de dados.

- SEQUENCE

Definição: Uma sequência é um objeto de banco de dados que gera números únicos cada vez que é chamado. É independente da tabela e pode ser usado em várias tabelas ou até mesmo fora delas.

Uso: Sequências são úteis quando você precisa de controle mais fino sobre o gerenciamento de identificadores únicos, como a capacidade de reverter para um valor anterior se necessário.

- IDENTITY

Definição: A propriedade IDENTITY é aplicada diretamente à definição de uma coluna em uma tabela. Ela especifica que essa coluna deve ser autoincrementada, garantindo que cada novo registro tenha um valor único para essa coluna.

Uso: Identidades são comumente usadas para colunas de chave primária que precisam de valores únicos e consecutivos sem intervenção manual.

- Diferenças Principais:

Escopo: Sequências são objetos independentes que podem ser referenciados por várias tabelas, enquanto identidades são atribuídas a colunas específicas dentro de uma única tabela.

Controle: Sequências oferecem mais controle sobre o gerenciamento dos valores, incluindo a capacidade de redefinir o valor atual. Identidades são mais simples e adequadas para casos de uso padrão onde não há necessidade de controle adicional.

Portabilidade: Embora ambas sejam amplamente suportadas, a sintaxe e os recursos específicos podem variar entre diferentes sistemas de gerenciamento de banco de dados (SGBDs). Por exemplo, a propriedade IDENTITY é comumente encontrada em SQL Server, Oracle e outros SGBDs, enquanto a palavra-chave SEQUENCE é parte do padrão SQL:2008 e é suportada em PostgreSQL, MySQL (a partir da versão 5.7), e outros.

Qual a importância das chaves estrangerias para a consistência do banco?

As chaves estrangeiras (foreign keys) desempenham um papel crucial na manutenção da integridade e consistência dos dados em um banco de dados relacional. Elas são uma ferramenta fundamental para garantir que os dados estejam corretos, completos e consistentes entre diferentes tabelas. Aqui estão alguns pontos importantes sobre a importância das chaves estrangeiras:

1. Integridade Referencial

A principal função das chaves estrangeiras é garantir a integridade referencial. Isso significa que cada valor em uma coluna de chave estrangeira deve corresponder a um valor existente na coluna primária da tabela relacionada. Se você tentar inserir ou atualizar um valor em uma coluna de chave estrangeira que não existe na tabela referenciada, o banco de dados rejeitará a operação, evitando assim inconsistências nos dados.

2. Restrições de Chave Estrangeira

As restrições de chave estrangeira podem ser definidas como CASCADE, SET NULL, SET DEFAULT ou NO ACTION, permitindo controlar o comportamento quando uma linha referenciada é excluída ou modificada. Por exemplo, se uma restrição CASCADE DELETE for usada, a exclusão de uma linha na tabela referenciada automaticamente excluirá todas as linhas correspondentes na tabela que contém a chave estrangeira.

3. Facilitação de Consultas

Chaves estrangeiras facilitam a realização de consultas complexas e a junção de tabelas. Eles permitem que você relate facilmente informações de várias tabelas com base em relações lógicas, tornando mais fácil extrair insights valiosos dos seus dados.

4. Melhoria da Performance

Embora nem sempre seja o caso, especialmente em bancos de dados modernos otimizados, chaves estrangeiras podem ajudar a melhorar a performance das consultas ao reduzir a necessidade de pesquisas completas em todas as tabelas envolvidas. Além disso, elas podem ser indexadas separadamente para acelerar ainda mais as consultas.

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

A álgebra relacional e o cálculo relacional são conceitos fundamentais na teoria dos bancos de dados relacionais, cada um com seus próprios conjuntos de operadores específicos que permitem manipular e transformar dados em tabelas.

- Álgebra Relacional

Os operadores da álgebra relacional são usados para realizar operações básicas sobre relações (tabelas). Eles incluem:

Seleção (σ): Filtra os registros de uma relação baseado em uma condição.
Projeção (π): Seleciona colunas específicas de uma relação.
Junção (⋈): Combina duas ou mais relações com base em uma condição de igualdade entre atributos correspondentes.

Diferença (−): Subtrai os registros de uma relação da outra.
União (∪): Combina os registros de duas relações sem duplicatas.
Interseção (∩): Combina os registros de duas relações mantendo apenas os registros que aparecem em ambas.
Divisão (÷): Semelhante à interseção, mas considera também as chaves estrangeiras.

- Cálculo Relacional

O cálculo relacional é uma extensão da álgebra relacional que permite expressões mais complexas e condicionais. Os operadores do cálculo relacional incluem:

Operadores Relacionais: São operadores como =, <, >, <=, >=, <> (ou !=) que permitem comparar valores entre colunas ou entre valores literais e colunas.
Operadores Lógicos: Incluem AND, OR, NOT que permitem combinar condições de maneira lógica.
Operadores de Agregação: Como SUM, COUNT, AVG, MAX, MIN que realizam cálculos em um conjunto de valores.
Funções de Data e Hora: Como CURRENT_DATE, DATEADD, DATEDIFF que manipulam datas e horas.
Funções de String: Como CONCAT, SUBSTRING, LENGTH que manipulam strings.

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

O agrupamento (ou grouping) em consultas SQL é uma técnica usada para combinar linhas que têm valores semelhantes em colunas específicas em um único grupo. Isso é útil quando você deseja realizar operações agregadas, como soma, média, contagem, etc., em grupos de dados.

- Como Realizar Agrupamento

Para realizar o agrupamento em uma consulta SQL, você usa a cláusula GROUP BY. A sintaxe básica é:

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s);
```

- column_name(s) são as colunas pelas quais você quer agrupar os resultados.
- table_name é o nome da tabela de onde você está selecionando os dados.
- condition é opcional e pode ser usado para filtrar os registros antes do agrupamento.

- Requisitos Obrigatórios

Para que uma consulta SQL funcione corretamente, especialmente quando se trata de agrupamento, existem alguns requisitos importantes:

1. Colunas na Cláusula SELECT: Todas as colunas listadas na cláusula SELECT que não estão envolvidas em funções agregadas devem estar incluídas na cláusula GROUP BY.

2. Funções Agregadas: Quando você usa funções agregadas (como SUM, COUNT, AVG, etc.), todas as colunas listadas no SELECT que não são usadas diretamente nessas funções devem ser incluídas na cláusula GROUP BY.

3. Ordem das Colunas: A ordem das colunas na cláusula GROUP BY deve corresponder à ordem em que aparecem nas colunas listadas após qualquer função agregada na cláusula SELECT.

4. Tipos de Dados: As colunas usadas na cláusula GROUP BY devem ter tipos de dados compatíveis entre si e com as funções agregadas utilizadas.

5. Cláusula WHERE: Se aplicável, a cláusula WHERE pode ser usada para filtrar os registros antes do agrupamento, mas ela não substitui a necessidade de agrupar por colunas relevantes.