



Centro Universitário Estácio

Curso: Desenvolvimento Full Stack

Disciplina: Vamos Manter as Informações

Semestre: 3º

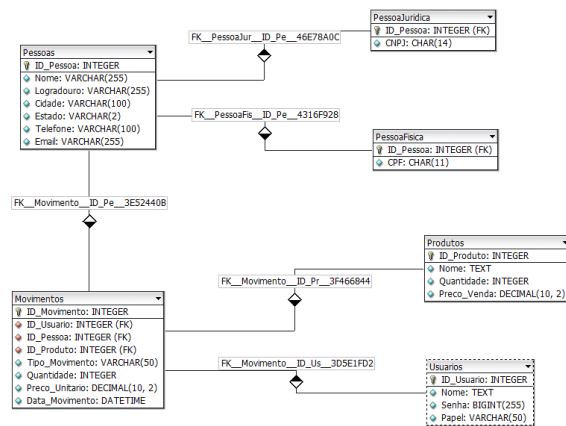
Aluno: Jonnatha Walben Saldanha da Silva

1º Procedimento - Criando o Banco de Dados

Objetivo da Prática:

O objetivo desta prática é demonstrar habilidades básicas de modelagem de banco de dados em um sistema utilizando o SQL Server Management Studio (SSMS) e o DB Designer Fork. Os principais pontos abordados incluem a criação de tabelas, definição de relacionamentos entre tabelas, uso de chaves primárias e estrangeiras, e criação de sequences para geração de identificadores. Além disso, é importante explorar as funcionalidades das ferramentas de modelagem e do SSMS para melhorar a produtividade no gerenciamento do banco de dados.

DBDesigner Fork – Modelo de Dados



Ativar o Win
Acesse Configura

Códigos Solicitados:

Aqui está um resumo dos códigos solicitados e apresentados ao longo do procedimento:

1. Criação do Logon e Usuário.

```
USE master;  
CREATE LOGIN loja WITH PASSWORD = 'loja';  
GO  
USE seu_banco_de_dados;  
CREATE USER loja FOR LOGIN loja;  
GRANT CREATE TABLE TO loja;  
GO
```

2. Criação das tabelas no banco de dados 'loja'

```
-- Tabela Usuários
CREATE TABLE Usuarios (
  ID_Usuario INT PRIMARY KEY,
  Nome TEXT,
  Senha BIGINT(255),
  Papel VARCHAR(50)
);

-- Tabela Pessoas
CREATE TABLE Pessoas (
  ID_Pessoa INT PRIMARY KEY,
  Nome VARCHAR(255),
  Logradouro VARCHAR(255),
  Cidade VARCHAR(100),
  Estado VARCHAR(2),
  Telefone VARCHAR(100),
  Email VARCHAR(100)
);

-- Tabela Produtos
CREATE TABLE Produtos (
  ID_Produto INT PRIMARY KEY,
  Nome TEXT,
  Quantidade INT,
  Preço_Venda DECIMAL(10, 2)
);

-- Tabela Movimentos
CREATE TABLE Movimentos (
  ID_Movimento INT PRIMARY KEY,
  ID_Usuario INT,
  ID_Pessoa INT,
  ID_Produto INT,
  Tipo_Movimento VARCHAR(50),
  Quantidade INT,
  Preço_Unitario DECIMAL(10, 2),
  Data_Movimento DATETIME,
  FOREIGN KEY (ID_Usuario) REFERENCES Usuarios(ID_Usuario),
  FOREIGN KEY (ID_Pessoa) REFERENCES Pessoas(ID_Pessoa),
  FOREIGN KEY (ID_Produto) REFERENCES Produtos(ID_Produto)
);

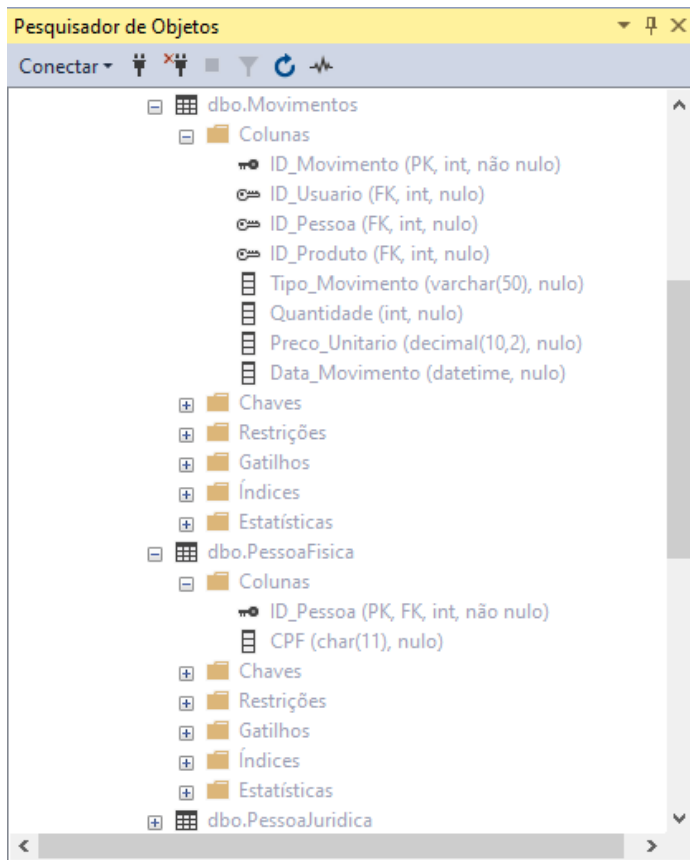
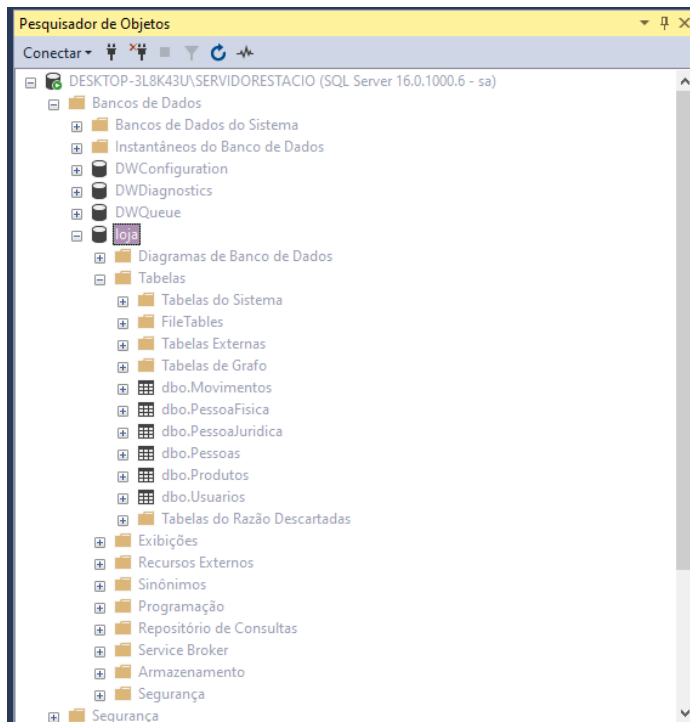
-- Tabela PessoaFisica
CREATE TABLE PessoaFisica (
  ID_Pessoa INT PRIMARY KEY,
  CPF CHAR(11) UNIQUE,
  FOREIGN KEY (ID_Pessoa) REFERENCES Pessoas(ID_Pessoa)
);

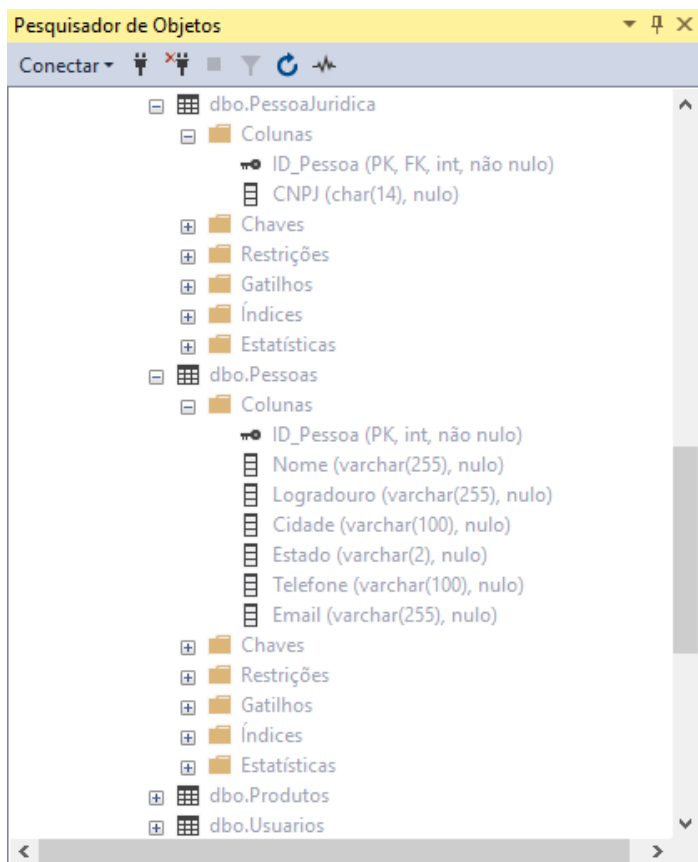
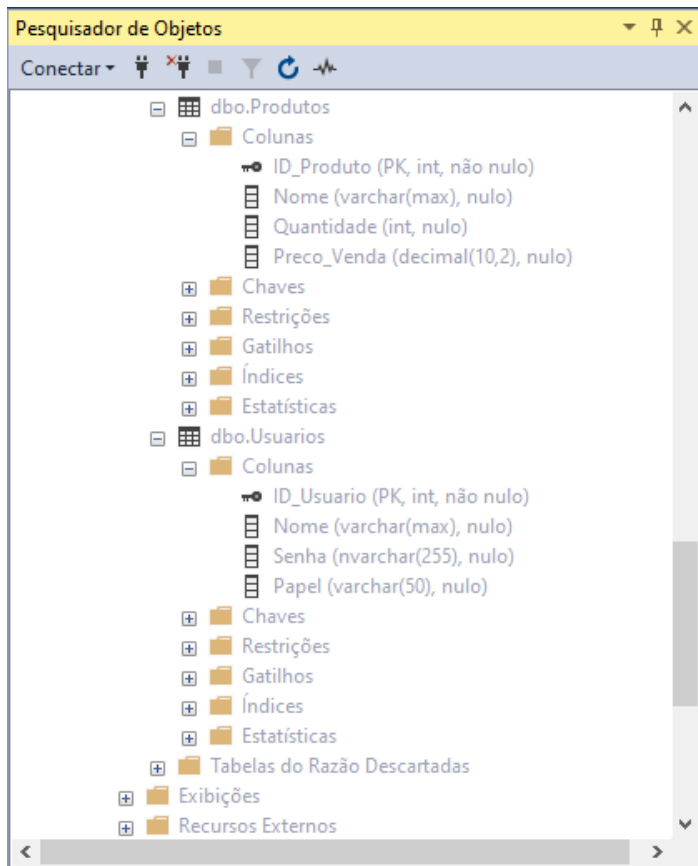
-- Tabela PessoaJuridica
CREATE TABLE PessoaJuridica (
  ID_Pessoa INT PRIMARY KEY,
  CNPJ CHAR(14) UNIQUE,
  FOREIGN KEY (ID_Pessoa) REFERENCES Pessoas(ID_Pessoa)
);
```

3. Definição de uma sequence para geração de identificadores de pessoa.

```
CREATE SEQUENCE Seq_ID_Pessoa START WITH 1 INCREMENT BY 1;
GO
```

Resultados da Execução:





Análise e Conclusão:

4. Cardinalidades em Banco de Dados Relacional:

- As diferentes cardinalidades, como 1X1, 1XN ou NxN, são implementadas por meio do uso de chaves primárias e estrangeiras para estabelecer relacionamentos entre tabelas.
- Um relacionamento 1X1 é implementado quando uma tabela possui uma chave estrangeira que faz referência a uma chave primária em outra tabela, e vice-versa.
- Um relacionamento 1XN é implementado quando uma tabela possui uma chave estrangeira que faz referência à chave primária em outra tabela, mas essa outra tabela pode ter várias linhas associadas a uma única linha na primeira tabela.
- Um relacionamento NxN é implementado por meio de uma tabela de junção que contém chaves estrangeiras referentes a outras duas tabelas, permitindo associações múltiplas entre as entidades.

5. Herança em Bancos de Dados Relacionais:

- Para representar o conceito de herança em bancos de dados relacionais, geralmente é utilizado um modelo de tabela única (Single Table Inheritance) ou um modelo de tabela por classe (Class Table Inheritance).
- No modelo de tabela única, todos os atributos de todas as classes são combinados em uma única tabela, com um atributo adicional para indicar o tipo de objeto.
- No modelo de tabela por classe, cada classe é representada por uma tabela separada, com uma tabela adicional para as características comuns a todas as classes.

6. Melhoria da Produtividade com o SQL Server Management Studio (SSMS):

- O SSMS oferece uma interface gráfica e recursos avançados que permitem realizar tarefas de gerenciamento de banco de dados de forma eficiente e produtiva.
- Ele oferece recursos como IntelliSense para autocompletar código, assistentes para a criação de estruturas de banco de dados, visualizações gráficas de esquemas de banco de dados, ferramentas de depuração e otimização de consultas, e muito mais.
- Com o SSMS, é possível executar consultas SQL, administrar e monitorar o banco de dados, criar e modificar estruturas de banco de dados, e realizar várias outras tarefas relacionadas ao gerenciamento do banco de dados.

Conclusão:

Nesta prática, foram demonstradas habilidades básicas de modelagem de banco de dados usando o SQL Server Management Studio (SSMS) e o DB Designer Fork. Além disso, foi discutido como implementar diferentes tipos de relacionamentos e representar herança em bancos de dados relacionais, bem como como o SSMS pode melhorar a produtividade nas tarefas de gerenciamento do banco de dados. Essas habilidades são fundamentais para desenvolvedores e administradores de banco de dados que trabalham com sistemas baseados em banco de dados relacionais.

2º Procedimento – Alimentando a Base

- Conectar-se como usuário 'loja'.
- Incluir dados na tabela de usuários.

```
-- Conectar-se como usuário 'loja'
USE loja;
GO
-- Incluir dados na tabela de usuários
INSERT INTO Usuarios (ID_Usuario, Nome, Senha, Papel) VALUES
(1, 'op1', 'op1', 'Operador'),
(2, 'op2', 'op2', 'Operador');
```

- Produtos inseridos na tabela produtos.

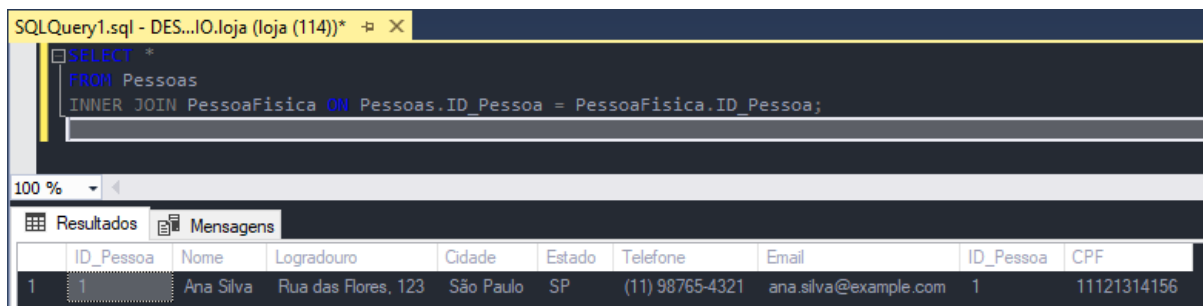
```
-- Inserir produtos na tabela Produtos
INSERT INTO Produtos (ID_Produto, Nome, Quantidade, Preco_Venda) VALUES
(1, 'Camiseta', 50, 29.99),
(2, 'Calça Jeans', 30, 69.99),
(3, 'Tênis Esportivo', 20, 99.99),
(4, 'Bermuda', 40, 39.99),
(5, 'Jaqueta de Couro', 10, 149.99);
```

- Criação de movimento base de dados.

```
-- Movimentações na tabela Movimentos
INSERT INTO Movimentos (ID_Movimento, ID_Usuario, ID_Pessoa, ID_Produto,
Tipo_Movimento, Quantidade, Preco_Unitario, Data_Movimento)
VALUES
(1, 1, 1, 1, 'E', 20, 25.99, '2024-04-22'), -- Entrada de 20 unidades do Produto 1
(2, 1, 2, 2, 'E', 10, 59.99, '2024-04-22'), -- Entrada de 10 unidades do Produto 2
(3, 2, 3, 3, 'S', 5, 99.99, '2024-04-22'), -- Saída de 5 unidades do Produto 3
(4, 2, 4, 1, 'S', 15, 29.99, '2024-04-22'); -- Saída de 15 unidades do Produto 1
```

Resultados das efetuações nas seguintes consultas sobre os dados inseridos:

- Dados completos de pessoas físicas.



The screenshot shows a SQL query window titled 'SQLQuery1.sql - DES...IO.loja (loja (114))'. The query is: `SELECT * FROM Pessoas INNER JOIN PessoaFisica ON Pessoas.ID_Pessoa = PessoaFisica.ID_Pessoa;`. The results are displayed in a table with columns: ID_Pessoa, Nome, Logradouro, Cidade, Estado, Telefone, Email, ID_Pessoa, and CPF. The first row shows data for ID_Pessoa 1: Ana Silva, Rua das Flores, 123, São Paulo, SP, (11) 98765-4321, ana.silva@example.com, 1, 11121314156.

ID_Pessoa	Nome	Logradouro	Cidade	Estado	Telefone	Email	ID_Pessoa	CPF
1	Ana Silva	Rua das Flores, 123	São Paulo	SP	(11) 98765-4321	ana.silva@example.com	1	11121314156

- Dados completos de pessoas jurídicas.

SQLQuery2.sql - DES...CIO.loja (loja (62))*

```

SELECT *
FROM Pessoas
INNER JOIN PessoaJuridica ON Pessoas.ID_Pessoa = PessoaJuridica.ID_Pessoa;

```

100 %

Resultados Mensagens

	ID_Pessoa	Nome	Logradouro	Cidade	Estado	Telefone	Email	ID_Pessoa	CNPJ
1	3	Pedro Souza	Rua dos Girassóis, 987	Brasília	DF	(61) 98765-4321	pedro.souza@example.com	3	12345678000190

Movimentações de entrada.

SQLQuery3.sql - DES...CIO.loja (loja (62))*

```

SELECT M.ID_Movimento, P.Nome AS Fornecedor, Pr.Nome AS Produto, M.Quantidade, M.Preco_Unitario, M.Quantidade * M.Preco_Unitario AS Valor_Total
FROM Movimentos M
JOIN Pessoas P ON M.ID_Pessoa = P.ID_Pessoa
JOIN Produtos Pr ON M.ID_Produto = Pr.ID_Produto
WHERE M.Tipo_Movimento = 'E';

```

100 %

Resultados Mensagens

	ID_Movimento	Fornecedor	Produto	Quantidade	Preco_Unitario	Valor_Total
1	1	Ana Silva	Camiseta	20	25.99	519.80
2	2	João Oliveira	Calça Jeans	10	59.99	599.90

- Movimentações de saída.

SQLQuery4.sql - DES...CIO.loja (loja (62))*

```

SELECT M.ID_Movimento, P.Nome AS Comprador, Pr.Nome AS Produto, M.Quantidade, M.Preco_Unitario, M.Quantidade * M.Preco_Unitario AS Valor_Total
FROM Movimentos M
JOIN Pessoas P ON M.ID_Pessoa = P.ID_Pessoa
JOIN Produtos Pr ON M.ID_Produto = Pr.ID_Produto
WHERE M.Tipo_Movimento = 'S';

```

100 %

Resultados Mensagens

	ID_Movimento	Comprador	Produto	Quantidade	Preco_Unitario	Valor_Total
1	3	Pedro Souza	Tênis Esportivo	5	99.99	499.95
2	4	João Silva	Camiseta	15	29.99	449.85

- Valor total das entradas agrupadas por produto.

SQLQuery5.sql - DES...CIO.loja (loja (62))*

```

SELECT Pr.Nome AS Produto, SUM(M.Quantidade * M.Preco_Unitario) AS Valor_Total_Entradas
FROM Movimentos M
JOIN Produtos Pr ON M.ID_Produto = Pr.ID_Produto
WHERE M.Tipo_Movimento = 'E'
GROUP BY Pr.Nome;

```

100 %

Resultados Mensagens

	Produto	Valor_Total_Entradas
1	Calça Jeans	599.90
2	Camiseta	519.80

- Valor total das saídas agrupadas por produto:

SQLQuery6.sql - DES...CIO.loja (loja (62))*

```

SELECT Pr.Nome AS Produto, SUM(M.Quantidade * M.Preco_Unitario) AS Valor_Total_Saidas
FROM Movimentos M
JOIN Produtos Pr ON M.ID_Produto = Pr.ID_Produto
WHERE M.Tipo_Movimento = 'S'
GROUP BY Pr.Nome;

```

100 %

Resultados Mensagens

	Produto	Valor_Total_Saidas
1	Camiseta	449.85
2	Tênis Esportivo	499.95

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

SQLQuery7.sql - DES...CIO.loja (loja (62))*

```

SELECT U.Nome AS Operador
FROM Usuarios U
LEFT JOIN Movimentos M ON U.ID_Usuario = M.ID_Usuario AND M.Tipo_Movimento = 'E'
WHERE M.ID_Movimento IS NULL;

```

100 %

Resultados Mensagens

	Operador
1	op2

- Valor total de entrada, agrupado por operador

SQLQuery8.sql - DES...CIO.loja (loja (62))*

```

SELECT U.Nome AS Operador, SUM(M.Quantidade * M.Preco_Unitario) AS Valor_Total_Entradas
FROM Usuarios U
LEFT JOIN Movimentos M ON U.ID_Usuario = M.ID_Usuario AND M.Tipo_Movimento = 'E'
GROUP BY U.Nome;

```

100 %

Resultados Mensagens

	Operador	Valor_Total_Entradas
1	op1	1119.70
2	op2	NULL

- Valor total de saída, agrupado por operador.

SQLQuery9.sql - DES...CIO.loja (loja (62))*

```

SELECT U.Nome AS Operador, SUM(M.Quantidade * M.Preco_Unitario) AS Valor_Total_Saidas
FROM Usuarios U
LEFT JOIN Movimentos M ON U.ID_Usuario = M.ID_Usuario AND M.Tipo_Movimento = 'S'
GROUP BY U.Nome;

```

100 %

Resultados Mensagens

	Operador	Valor_Total_Saidas
1	op1	NULL
2	op2	949.80

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

SQLQuery10.sql - DE...CIO.loja (loja (62))*

```

SELECT Pr.Nome AS Produto, SUM(M.Quantidade * M.Preco_Unitario) / SUM(M.Quantidade) AS Valor_Medio_Venda
FROM Movimentos M
JOIN Produtos Pr ON M.ID_Produto = Pr.ID_Produto
WHERE M.Tipo_Movimento = 'S'
GROUP BY Pr.Nome;

```

100 %

Resultados Mensagens

	Produto	Valor_Medio_Venda
1	Camiseta	29.990000
2	Tênis Esportivo	99.990000

Análise e Conclusão

Diferenças no uso de sequence e identity:

Identity: No SQL Server, **IDENTITY** é uma propriedade de coluna que automaticamente gera valores numéricos únicos à medida que novas linhas são inseridas em uma tabela. Os valores são gerenciados internamente pelo banco de dados e são incrementados automaticamente. O **IDENTITY** é específico para colunas individuais em uma tabela.

- **Sequence:** Uma sequence é um objeto de banco de dados que gera uma sequência de valores numéricos. Ela pode ser usada para fornecer valores para colunas em tabelas ou em outras situações onde valores sequenciais são necessários. Ao contrário do **IDENTITY**, uma sequence é um objeto separado e pode ser compartilhada por várias tabelas ou colunas em um banco de dados.

Importância das chaves estrangeiras para a consistência do banco:

- Chaves estrangeiras são fundamentais para garantir a integridade referencial em um banco de dados. Elas estabelecem uma relação entre duas tabelas, onde a chave estrangeira em uma tabela faz referência à chave primária (ou a uma coluna única) em outra tabela. Isso garante que cada valor na coluna de chave estrangeira corresponda a um valor válido na tabela relacionada, mantendo a consistência dos dados e evitando relações inválidas.

Operadores do SQL pertencentes à álgebra relacional e ao cálculo relacional:

- **Álgebra Relacional:** Inclui operadores como **SELECT**, **PROJECT**, **JOIN**, **UNION**, **INTERSECT**, **DIFFERENCE**, entre outros. Estes operadores são usados para manipular conjuntos de dados de tabelas.
- **Cálculo Relacional:** Não há uma correspondência direta entre operadores SQL e operadores do cálculo relacional. O cálculo relacional é uma abordagem mais descritiva, onde as consultas são expressas em termos de predicados lógicos. Alguns conceitos do cálculo relacional podem ser implementados em SQL usando cláusulas como **WHERE**, **HAVING**, **EXISTS**, **ALL**, **ANY**, entre outras.

Agrupamento em consultas e requisitos obrigatórios:

- O agrupamento em consultas é feito usando a cláusula **GROUP BY**, que agrupa as linhas de um conjunto de resultados com base em uma ou mais colunas. O requisito obrigatório para usar **GROUP BY** é que qualquer coluna que não esteja sendo agregada na consulta deve ser incluída na cláusula **GROUP BY**. Isso significa que cada coluna no **SELECT** que não seja uma função de agregação (como **SUM**, **AVG**, **COUNT**, etc.) deve estar presente na cláusula **GROUP BY**.