# Missão Prática | Nível 2 | Mundo 3

## RPG0015 - Vamos manter as informações!

**Aluna:** Simone Ramos de Jesus.

**Matricula:** 202208290965.

**Campus:** Polo Prado – Belo Horizonte – MG.

**Curso:** Desenvolvimento Full Stack

**Turma**: 2023.3

**GITHUB:**

## Objetivos 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 dado (DML).

5 – No final do exercício, o aluno terá vivenciado a experiência de modelar a base de dados para um sistema simples, além de implementá-la, através da sintaxe SQL, na plataforma do SQL Server.

## CadrastoBD.sql

```
Execute
CREATE TABLE[pessoa](
    idPessoa INT (1,1) PRIMARY KEY,
    nome VARCHAR(255) ,
    logradouro VARCHAR(255),
    cidade VARCHAR(255),
    estado CHAR(2),
    telefone VARCHAR(11),
    email VARCHAR(255),   );

Execute
CREATE TABLE [pessoa_fisica](
    idPessoaFisica INT (1,1) PRIMARY KEY,
    cpf VARCHAR(11),
    FOREIGN KEY(IdPessoaFisica) REFERENCES Pessoa(idPessoa)
);

Execute
CREATE TABLE [pessoa_juridica](
    idPessoaJuridica INT (1,1) PRIMARY KEY,
    FOREIGN KEY (idPessoaJuridica) REFERENCES Pessoa(idPessoa)
);

Execute
CREATE TABLE[produto](
    idProduto INT (1,1) PRIMARY KEY,
    nome VARCHAR(255),
    quantidade INT,
    precoVenda NUMERIC,
);

Execute
CREATE TABLE[usuario](
    idOperador INT (1,1) PRIMARY KEY,
    nome VARCHAR(25),
    senha VARCHAR(25),
);
```

```sql
CREATE TABLE[MovimentoCompra](
    idMovimentoCopmpra INT (1,1) PRIMARY KEY,
    precoUnitario NUMERIC,
    quantidade INT,
    idOperador INT,
    idFornecedor INT,
    idProduto INT,
    FOREIGN KEY (idOperador) REFERENCES Usuario(idOperador),
    FOREIGN KEY (idProduto) REFERENCES Produto(idProduto),
    FOREIGN KEY (idFornecedor) REFERENCES PessoaJuridica(idPessoaJuridica),
);

CREATE TABLE MovimentoVenda(
    quantidade INT,
    precoUnitario NUMERIC,
    idMovimentoVenda INT PRIMARY KEY,
    idProduto INT,
    idComprador INT,
    idOperador INT,
    FOREIGN KEY (idProduto) REFERENCES Produto(idProduto),
    FOREIGN KEY (idComprador) REFERENCES PessoaFisica(idPessoaFisica),
    FOREIGN KEY (idOperador) REFERENCES Usuario(idOperador),
);

CREATE SEQUENCE IdSequencePessoa
    START WITH 1
    INCREMENT BY 1;

INSERT INTO Usuario(idOperador, nome, senha) VALUES (1, "op1","op1");
INSERT INTO Usuario(idOperador, nome, senha) VALUES (2, "op2","op2");
```


```sql
INSERT INTO Usuario(idOperador, nome, senha) VALUES (1, "op1","op1");
INSERT INTO Usuario(idOperador, nome, senha) VALUES (2, "op2","op2");
```


```sql
INSERT INTO Produto(idProduto, nome, quantidade, precoVenda) VALUES (1,"Banana",100, 5.00);
INSERT INTO Produto(idProduto, nome, quantidade, precoVenda) VALUES (2,"Laranja",500, 2.00);
INSERT INTO Produto(idProduto, nome, quantidade, precoVenda) VALUES (3,"Manga",800, 4.00);

INSERT INTO Pessoa(idPessoa, nome, logradouro, cidade, estado, telefone, email) VALUES (7,"Joao","Rua 12, cas
INSERT INTO PessoaFisica (idPessoaFisica, cpf) VALUES (7,"11111111111");

INSERT INTO Pessoa (idPessoa, nome, logradouro, cidade, estado, telefone, email) VALUES (15,"JJC","Rua 11, Ce
INSERT INTO PessoaJuridica (idPessoaJuridica, cnpj) VALUES (15, "222222222222");


INSERT INTO MovimentoVenda (idMovimentoVenda, idProduto, idComprador, idOperador, quantidade, precoUnitario)
INSERT INTO MovimentoVenda (idMovimentoVenda, idProduto, idComprador, idOperador, quantidade, precoUnitario)
INSERT INTO MovimentoVenda (idMovimentoVenda, idProduto, idComprador, idOperador, quantidade, precoUnitario)



INSERT INTO MovimentoCompra(idMovimentoCompra, idProduto, idFornecedor, idOperador, quantidade, precoUnitario
INSERT INTO MovimentoCompra(idMovimentoCompra, idProduto, idFornecedor, idOperador, quantidade, precoUnitario
```

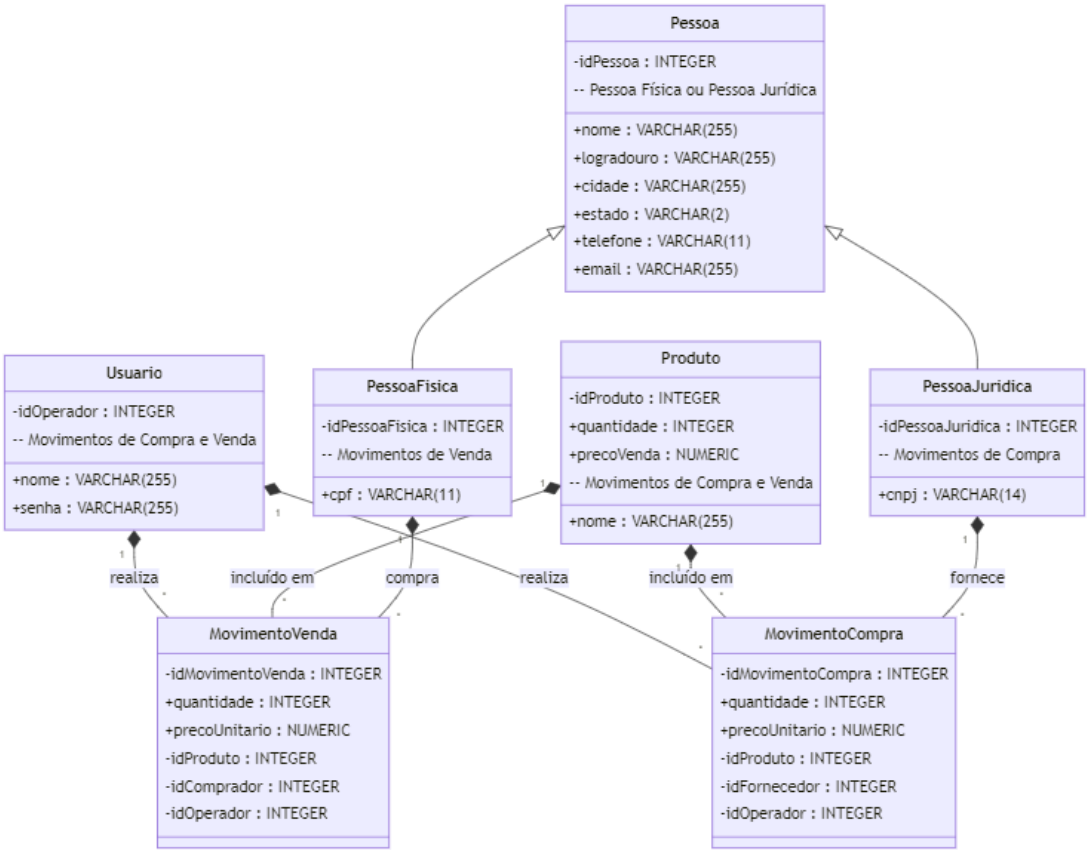
```
87
    ▷ Execute
88  SELECT * FROM Pessoa INNER JOIN PessoaFisica ON Pessoa.idPessoa = PessoaFisica.idPessoaFisica;
    ▷ Execute
89  SELECT * FROM Pessoa INNER JOIN PessoaJuridica ON Pessoa.idPessoa = PessoaJuridica.idPessoaJuridica;
    ▷ Execute
90  SELECT idProduto, SUM(quantidade*precoUnitario) AS TotalEntradas FROM MovimentoCompra GROUP BY idProduto;
```

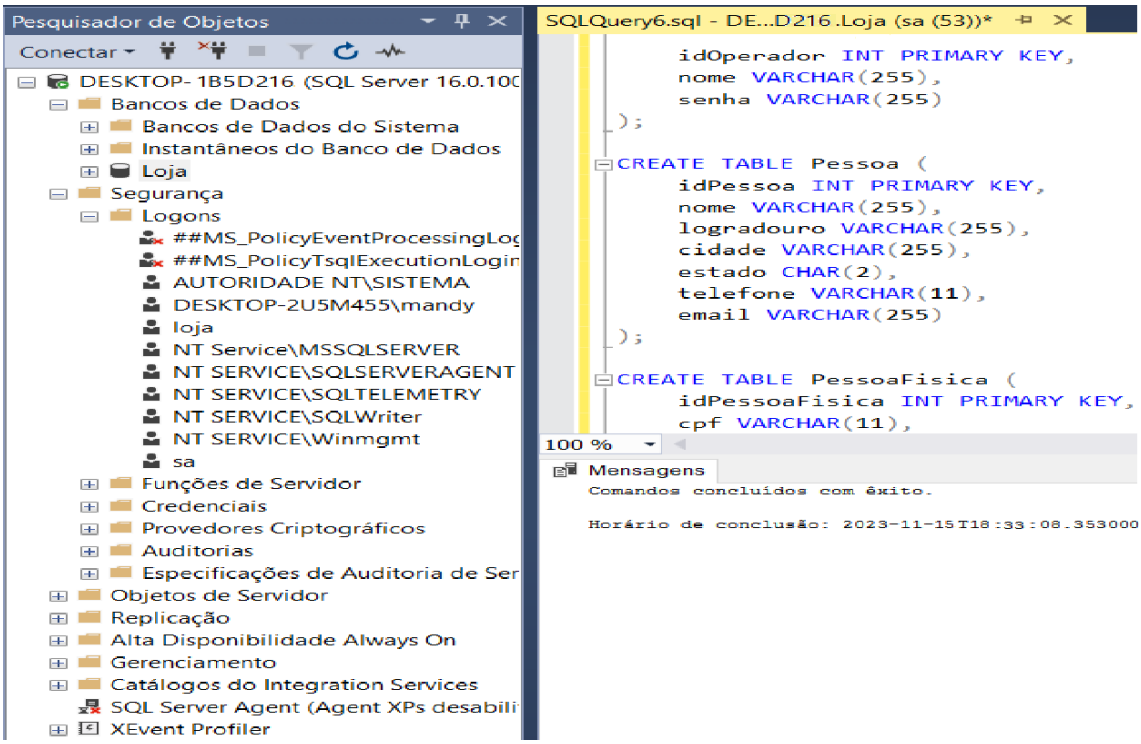
## ResultadoBD.sql:

```
2   SELECT p.*,pj.cnpj
3   FROM Pessoa p
4   INNER JOIN PessoaJuridica pj ON p.idPessoa = pj.idPessoaJuridica;
5
    ▷ Execute
6   SELECT p.*,pf.cpf
7   FROM Pessoa p
8   INNER JOIN PessoaFisica pf ON p.idPessoa = pf.idPessoaFisica;
9
10
11
    ▷ Execute
12  SELECT
13      mv.idMovimentoVenda,
14      pr.nome AS Produto,
15      p.nome AS Comprador,
16      mv.quantidade,
17      mv.precoUnitario,
18      (mv.quantidade * mv.precoUnitario) AS valorTotal
19  FROM
20      MovimentoVenda mv
21  INNER JOIN
22      Produto pr ON mv.idProduto = pr.idProduto
23  INNER JOIN
24      PessoaFisica pf ON mv.idComprador = pf.idPessoaFisica
25  INNER JOIN
26      Pessoa p ON pf.idPessoaFisica = p.idPessoa;
27
28
    ▷ Execute
29  SELECT
30      mv.idMovimentoCompra,
31      pr.nome AS NomeProduto,
32      p.nome AS NomeFornecedor,
33      mc.quantidade,
34      mc.precoUnitario,
35      (mc.quantidade * mc.precoUnitario) AS valorTotal
```

```
FROM
    MovimentoCompra mc
INNER JOIN
    Produto pr ON mc.idProduto = pr.idProduto
INNER JOIN
    PessoaJuridica pj ON mc.idFornecedor = pj.idPessoaJuridica
INNER JOIN
    Pessoa p ON pj.idPessoaJuridica = p.idPessoa;



▷ Execute
SELECT pr.nome AS Produto, SUM(mv.quantidade* mv.precoUnitario) AS ValorTotalSaídas
FROM MovimentoVenda mv
INNER JOIN Produto pr ON mv.idProduto = pr.idProduto
GROUP BY pr.nome;

▷ Execute
SELECT pr.nome AS Produto, SUM(mc.quantidade* mc.precoUnitario) AS ValorTotalEntradas
FROM MovimentoCompra mc
INNER JOIN Produto pr ON mc.idProduto = pr.idProduto
GROUP BY pr.nome;

▷ Execute
SELECT u.*
FROM Usuario u
LEFT JOIN MovimentoCompra mc ON u.idOperador = mc.idOperador
WHERE mc.idMovimentoCompra IS NULL;

▷ Execute
SELECT u.nome AS Operador, SUM(mc.quantidade*mc.precounitario) AS ValorTotalSaida
FROM MovimentoVenda mv
INNER JOIN Usuario u ON mv.idOperador = u.idOperador
GROUP BY u.nome;

▷ Execute
SELECT u.nome AS Operador, SUM(mc.quantidade*mc.precounitario) AS ValorTotalEntrada
FROM MovimentoCompra mc
INNER JOIN Usuario u ON mc.idOperador = u.idOperador
GROUP BY u.nome;


▷ Execute
SELECT pr.nome AS Produto, SUM(mv.quantidade*mv.precounitario) AS PrecoMedioM
FROM MovimentoVenda mv
INNER JOIN Produto pr ON mv.idProduto = pr.idProduto
GROUP BY pr.nome;
```

# Resultado da execução dos códigos:



100 %

Resultados | Mensagens

| | Operador | ValorTotalSaida |
|---|---|---|
| 1 | op1 | 10 |
| 2 | op2 | 30 |

100 %

Resultados | Mensagens

| | idOperador | nome | senha |
|---|---|---|---|
| 1 | 2 | op2 | op2 |

00 %

Resultados | Mensagens

| | idMovimentoVenda | Produto | Comprador | quantidade | precoUnitario | ValorTotal |
|---|---|---|---|---|---|---|
| 1 | 1 | Banana | Joao | 20 | 4 | 80 |

100 %

Resultados | Mensagens

| | Produto | ValorTotalEntradas |
|---|---|---|
| 1 | Banana | 15 |

100 %

Resultados | Mensagens

| | idMovimentoCompra | NomeProduto | NomeFornecedor | quantidade | precoUnitario | ValorTotal |
|---|---|---|---|---|---|---|
| 1 | 7 | Banana | JJC | 15 | 5 | 75 |
| 2 | 8 | Banana | JJC | 20 | 4 | 80 |

100 %

Resultados | Mensagens

| | idPessoa | nome | logradouro | cidade | estado | telefone | email | cnpj |
|---|---|---|---|---|---|---|---|---|
| 1 | 15 | JJC | Rua 11, Centro | Riacho do Norte | PA | 2222-2222 | email@example.com | 22222222222222 |

100 %

Resultados | Mensagens

| | idPessoa | nome | logradouro | cidade | estado | telefone | email | cpf |
|---|---|---|---|---|---|---|---|---|
| 1 | 7 | Joao | Rua 12, casa 3, Quitanda | Riacho do Sul | PA | 1111-1111 | email@example.com | 11111111111 |

**1 - Implementação de Cardinalidades:**

1x1 "Um-Para-Um": é uma única ocorrência que pode agregar com apenas uma única ocorrência de outro elemento.

1xN "Um-Para-Muitos": é uma ocorrência de uma entidade onde pode agregar com várias outros elementos.

NxN "Muitos-Para-Muitos": são várias ocorrências de uma entidade pode agregar com muitas outros elementos de outra entidade.

**2 - Representação de Herança em Bancos de Dados Relacionais:**

É sistema de armazenamento que permite a persistência de dados e implementar funcionalidades, uma tabela principal "Matriz" vai armazenar os atributos comuns a todas as entidades e tabelas secundárias para cada subclasse, ou seja, as tabelas secundárias "herdam" os dados da tabela principal, simulando a herança de classes em POO.

**3 - SQL Server Management Studio (SSMS) para Melhoria da Produtividade:**

SQL Server Management Studio é uma excelente ferramenta, que aumenta a produtividade e eficiência para o programador, este programa permite a administração e gerenciamento do SQL Server. Este programa de alta performance permite criar e alterar tabelas, gerenciar esquema, interface gráfica para fazer consultas, interação com o banco de dados e várias outras funcionalidades que torna acessível para o usuário.

**4 - Diferenças entre Sequence e Identity:**

Sequence: é um objeto de banco de dados que gera uma sequência de valores únicos. “Flexibilidade, Reutilização e Controle.”

Identity: é uma propriedade da tabela, que gera automaticamente valores incrementais. “Simplicidade, Especificidade e Facilidade de uso.”

**5 - Importância das Chaves Estrangeiras para a Consistência do Banco:**

Garantem a integridade referencial no banco de dados, no intuito que os relacionamentos entre tabelas. Um dos objetivos é impedir a inserção de dados que preservam as relações definidas, e a consistência e evitando inconsistência nos dados. “ Integridade Referencial, Prevenção de Órfãos e Consistência de Dados”.

**6- Operadores SQL e Álgebra Relacional:**

**Álgebra Relacional:** operadores Select, Project, Join, Union, Intersect e Difference.

**Seleção:** Filtrar linhas.

**Projeção:** Filtra colunas.

**União:** Combina resultados de duas consultas

**Diferença:** Retorna deferenças entre duas consultas.

**Operadores SQL:** SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY, JOIN, ETC.

## 7. Agrupamento em Consultas SQL e Requisito Obrigatório: Operadores SQL:

A junção das consultas SQL é feita com comando Group BY, é utilizado para agrupar linhas e colunas especificas.

**Requisitos Obrigatório**: Quando for utilizado e todas as colunas listadas na cláusula SELECT exceto (MAX, SUM, COUNT, AVG, GROUP BY E MIN).

## 8 - Conclusão:

A análise da Missão Prática trouxe uma valiosa introdução ao universo das operações de bancos de dados, abordando temas como a criação de tabelas, inserção de dados e estabelecimento de relacionamentos. Este conhecimento prático é fundamental para compreender e aplicar efetivamente conceitos cruciais na administração de sistemas de banco de dados relacionais**.**