



Missão Prática | Nível 1 | Mundo 3

Eduardo Eugênio de Araujo e Silva Domingues 202302579043

1172 POLO CASA CAIADA – OLINDA – PE

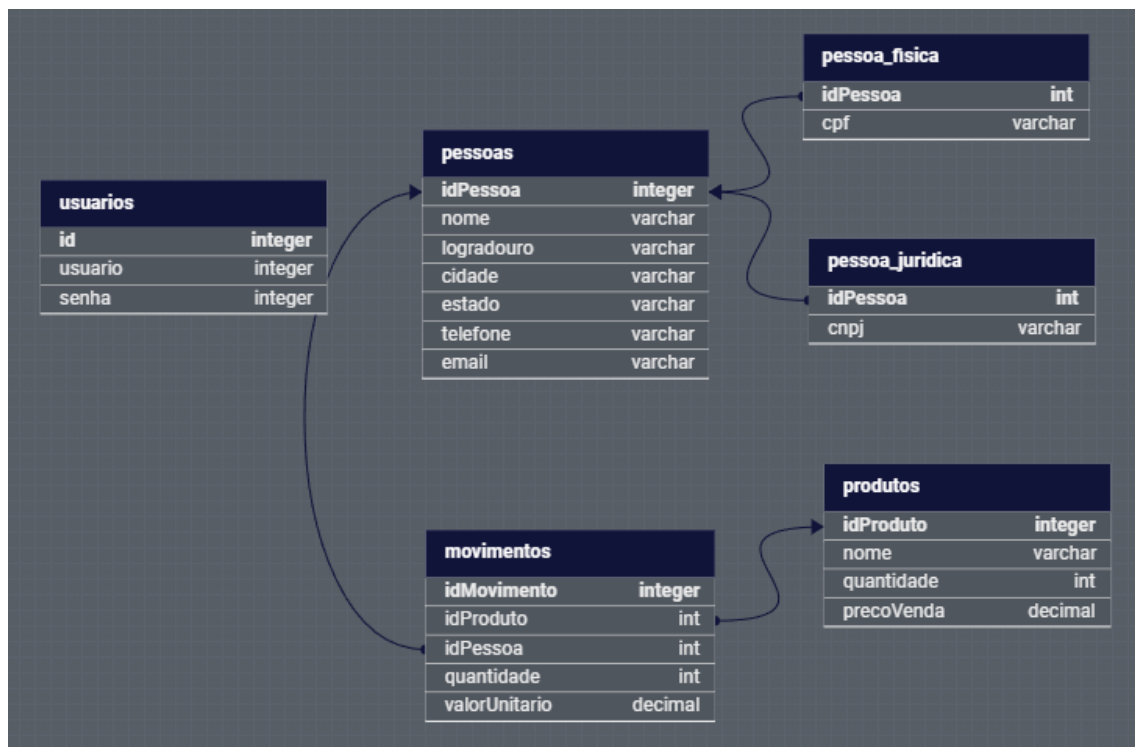
Vamos manter as informações! – RPG0015 – 2024.1

<https://github.com/eduardoduud/Missao-Pratica-Nivel-2-Mundo-3>

Objetivo da Prática

- Identificar os requisitos de um sistema e transformá-los no modelo adequado.
- Utilizar ferramentas de modelagem para bases de dados relacionais.
- Explorar a sintaxe SQL na criação das estruturas do banco (DDL).
- Explorar a sintaxe SQL na consulta e manipulação de dados (DML)

1º Procedimento | Criando o Banco de Dados






```
CREATE TABLE usuarios(  
    idUsuario INT IDENTITY(1,1) PRIMARY KEY,  
    usuario VARCHAR(100),  
    senha VARCHAR(30)  
);  
  
CREATE SEQUENCE SEQ_idPessoa  
    AS INT  
    START WITH 7  
    INCREMENT BY 8;  
  
CREATE TABLE pessoa (  
    idPessoa INT PRIMARY KEY NOT NULL DEFAULT NEXT VALUE FOR  
SEQ_idPessoa,  
    nome VARCHAR(255) not null,  
    logradouro VARCHAR(255) not null,  
    cidade VARCHAR(255) not null,  
    estado VARCHAR(2) not null,  
    telefone VARCHAR(11) not null,  
    email VARCHAR(255) not null UNIQUE  
);  
  
CREATE TABLE pessoa_juridica (  
    idPessoa INT FOREIGN KEY REFERENCES pessoa(idPessoa) not null PRIMARY  
KEY,  
    cnpj varchar(14) not null  
);  
  
CREATE TABLE pessoa_fisica (  
    idPessoa INT FOREIGN KEY REFERENCES pessoa(idPessoa) PRIMARY KEY,  
    cpf VARCHAR(11) not null  
);  
  
CREATE TABLE produto(  
    idProduto INT PRIMARY KEY NOT NULL,  
    nome VARCHAR(255),  
    quantidade INT,  
    precoVenda DECIMAL(10, 2)  
);
```



Estácio

```
CREATE TABLE movimento(  
    idMovimento INT IDENTITY(1,1) PRIMARY KEY,  
    idUsuario INT NOT NULL,  
    idPessoa INT NOT NULL,  
    idProduto INT NOT NULL,  
    quantidade INT,  
    tipo VARCHAR(1),  
    valorUnitario DECIMAL(10,2),  
    FOREIGN KEY (idUsuario) REFERENCES usuarios(idUsuario),  
    FOREIGN KEY (idPessoa) REFERENCES pessoa(idPessoa),  
    FOREIGN KEY (idProduto) REFERENCES produto(idProduto),  
    CONSTRAINT TIPO_MOVIMENTO_INVALIDO CHECK (tipo IN ('E', 'S'))  
);
```

 Mensagens

Comandos concluídos com êxito.

Horário de conclusão: 2024-05-01T01:08:15.7970355-03:00

- a) Cardinalidades 1x1 são implementadas usando chaves estrangeiras. 1xN usa chave estrangeira em uma tabela. NxN requer uma tabela de associação.
- b) O relacionamento de herança é representado usando a tabela por classe ou a tabela por hierarquia.
- c) O SQL Server Management Studio oferece recursos como edição visual de esquemas, consultas SQL, geração de scripts e monitoramento de desempenho, aumentando a eficiência e produtividade.

Observe que os tópicos acima seguem exatamente o que está na Atividade Prática exigida.



2º Procedimento | Alimentando a Base

```
INSERT INTO usuario (login, senha) VALUES ('op1', 'op1');
INSERT INTO usuario (login, senha) VALUES ('op2', 'op2');

SELECT * FROM usuario;

INSERT INTO produto (idProduto, nome, quantidade, precoVenda) VALUES (1,
'Banana', 100, 5.00);
INSERT INTO produto (idProduto, nome, quantidade, precoVenda) VALUES (3,
'Laranja', 500, 2.00);
INSERT INTO produto (idProduto, nome, quantidade, precoVenda) VALUES (4,
'Manga', 100, 4.00);

SELECT * FROM produto;

DECLARE @idFisica INT;
SET @idFisica = NEXT VALUE FOR idPessoa;

INSERT INTO pessoa (idPessoa, nome, logradouro, cidade, estado, telefone,
email)
VALUES (@idFisica, 'Joao', 'Rua 12, casa 3, Quitanda', 'Riacho do Sul',
'PA', '1111-1111', 'joao@riacho.com');

INSERT INTO pessoa_fisica (idPessoa, cpf) VALUES (@idFisica,
'11111111111');

SELECT * FROM pessoa AS p INNER JOIN pessoa_fisica pf ON p.idPessoa =
pf.idPessoa;

DECLARE @idJuridica INT;
SET @idJuridica = NEXT VALUE FOR idPessoa;

INSERT INTO pessoa (idPessoa, nome, logradouro, cidade, estado, telefone,
email)
VALUES (@idJuridica, 'JJC', 'Rua 11. Centro', 'Riacho do Norte', 'PA',
'1212-1212', 'jjc@riacho.com');

INSERT INTO pessoa_juridica (idPessoa, cnpj) VALUES (@idJuridica,
'22222222222222');
```



Estácio

```
SELECT * FROM pessoa a INNER JOIN pessoa_juridica pj ON a.idPessoa =  
pj.idPessoa;
```

```
INSERT INTO movimento (idUserario, idPessoa, idProduto, quantidade, tipo,  
valorUnitario) VALUES  
  (1, @idFisica, 1, 20, 'S', 4.00),  
  (1, @idFisica, 3, 15, 'S', 2.00),  
  (2, @idFisica, 3, 10, 'S', 3.00),  
  (1, @idJuridica, 3, 15, 'E', 5.00),  
  (1, @idJuridica, 4, 20, 'E', 4.00);
```

```
SELECT * FROM movimento;
```

```
SELECT * FROM pessoa a INNER JOIN pessoa_fisica pf ON a.idPessoa =  
pf.idPessoa;
```

```
SELECT * FROM pessoa a INNER JOIN pessoa_juridica pj ON a.idPessoa =  
pj.idPessoa;
```

```
SELECT a.idMovimento,  
  a.tipo,  
  b.nome AS Produto,  
  c.nome as Fornecedor,  
  a.quantidade AS Quantidade,  
  a.valorUnitario AS Preco_Unitario,  
  a.quantidade * a.valorUnitario AS Valor_Total  
FROM movimento a  
INNER JOIN pessoa b ON a.idPessoa = b.idPessoa  
INNER JOIN produto c ON a.idProduto = c.idProduto  
WHERE a.tipo = 'E';
```

```
SELECT a.idMovimento,  
  a.tipo,  
  b.nome AS Produto,  
  c.nome AS Comprador,  
  a.quantidade AS Quantidade,  
  a.valorUnitario AS Preco_Unitario,  
  a.quantidade * a.valorUnitario AS Valor_Total  
FROM movimento a  
INNER JOIN produto b ON a.idProduto = b.idProduto  
INNER JOIN pessoa c ON a.idPessoa = c.idPessoa  
WHERE a.tipo = 'S';
```



Estácio

```
SELECT b.idProduto, b.nome AS produto,  
       SUM(a.quantidade * a.valorUnitario) AS total  
FROM movimento a  
INNER JOIN produto b ON a.idProduto = b.idProduto  
WHERE a.tipo = 'E'  
GROUP BY b.nome, b.idProduto;
```

```
SELECT b.idProduto, b.nome AS produto,  
       SUM(a.quantidade * a.valorUnitario) AS total  
FROM movimento a  
INNER JOIN produto b ON a.idProduto = b.idProduto  
WHERE a.tipo = 'S'  
GROUP BY b.nome, b.idProduto;
```

```
SELECT a.* FROM usuario a  
LEFT JOIN movimento b ON b.idUsuario = a.idUsuario AND b.tipo = 'E'  
WHERE b.idMovimento IS NULL;
```

```
SELECT b.idUsuario, b.login AS operador,  
       SUM(a.quantidade * a.valorUnitario) AS total  
FROM movimento a  
INNER JOIN usuario b ON a.idUsuario = b.idUsuario  
WHERE a.tipo = 'E'  
GROUP BY b.login, b.idUsuario;
```

```
SELECT b.idUsuario, b.login AS operador,  
       SUM(a.quantidade * a.valorUnitario) AS total  
FROM movimento a  
INNER JOIN usuario b ON a.idUsuario = b.idUsuario  
WHERE a.tipo = 'S'  
GROUP BY b.login, b.idUsuario;
```

```
SELECT b.idProduto,  
       b.nome AS produto,  
       SUM(a.quantidade * a.valorUnitario) / SUM(a.quantidade) AS  
valorMedio  
FROM movimento a  
INNER JOIN produto b ON a.idProduto = b.idProduto  
WHERE a.tipo = 'S'  
GROUP BY b.idProduto, b.nome;
```



✔ Consulta executada com êxito.



Conclusão:

- a) Sequence é independente da tabela, enquanto Identity é específico de uma coluna.
- b) Chaves estrangeiras garantem integridade referencial, mantendo a consistência dos dados.
- c) Operadores JOIN, UNION, SELECT pertencem à álgebra relacional; EXISTS, FORALL são do cálculo relacional.
- d) Agrupamento é feito com GROUP BY. Colunas selecionadas precisam ser agregadas ou incluídas na cláusula GROUP BY.

Observe que os tópicos acima seguem exatamente o que está na Atividade Prática exigida.