Implementação de Banco de Dados para Gerenciamento de Movimentos de Compra e Venda

Missão Prática | Nível 2 | Mundo 3

- **Aluna:** Amanda Duque Kawauchi
- **Matrícula:** 202209170254
- **Campus:** Morumbi

- **Curso:** Desenvolvimento Full-Stack
- **Disciplina:** Nível 2: Vamos Manter as Informações?
- **Turma:** 2023.3
- **Semestre Letivo:** 3º Semestre
- **Repositório GitHub:** Link do Repositório GitHub

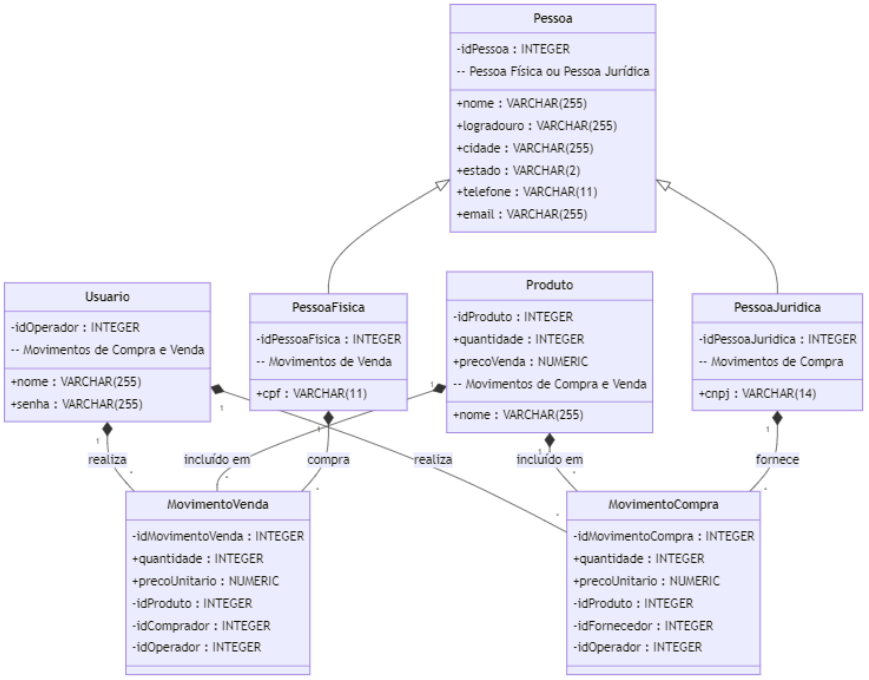
Objetivo da Prática

Esta prática tem como objetivo a criação de um banco de dados relacional para o gerenciamento de movimentos de compra e venda dentro de uma plataforma de negociações, utilizando SQL Server Management Studio e aplicando conceitos de modelagem UML e SQL.

👉 1º Procedimento – Criando o Banco de Dados

Definição do Modelo de Dados

A estrutura do banco de dados é projetada seguindo o diagrama de classe UML, definindo as entidades `Usuario`, `Pessoa`, `PessoaFisica`, `PessoaJuridica`, `Produto`, `MovimentoCompra`, e `MovimentoVenda`, cada uma com seus atributos e relações, exemplificando a organização e relacionamentos entre as tabelas.



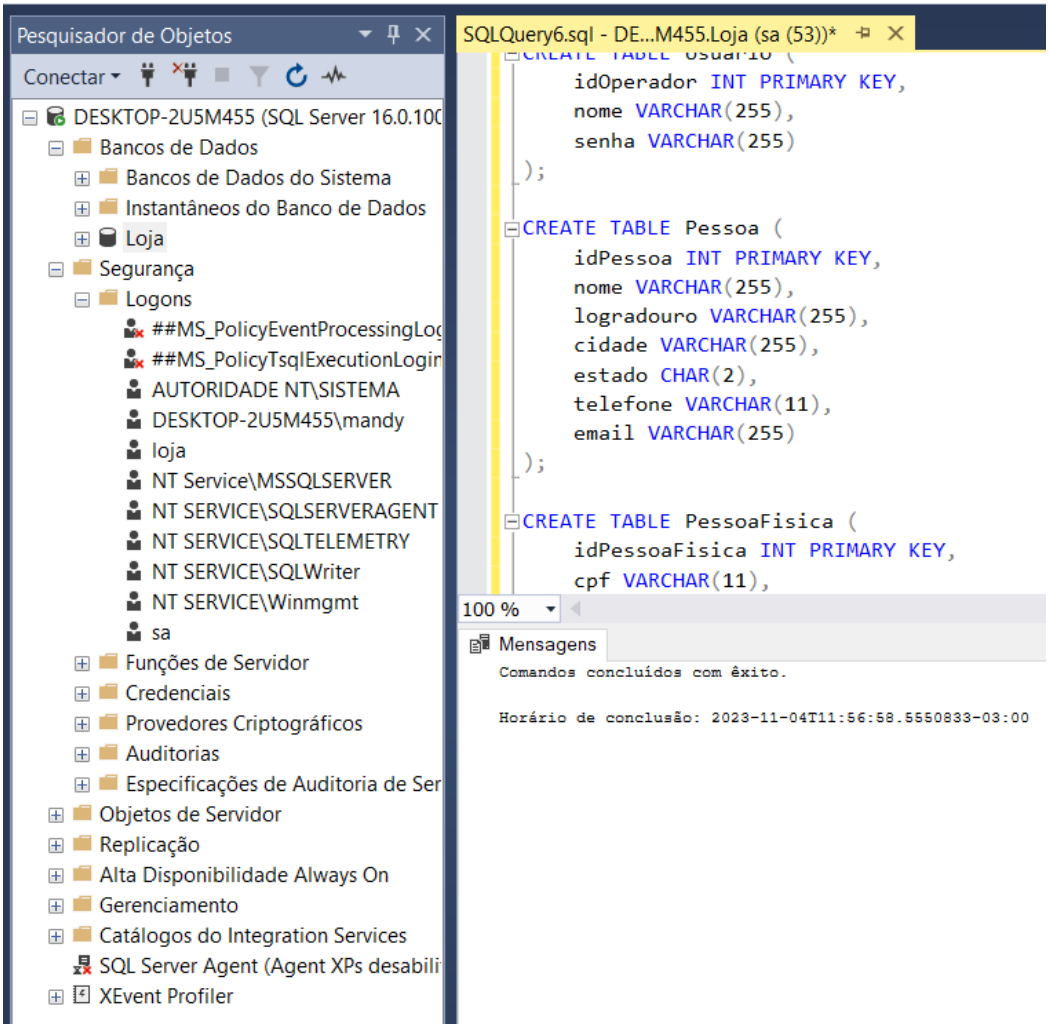
Criação da Base de Dados

```
CREATE  TABLE  Usuario (
idOperador INT  PRIMARY KEY,
nome VARCHAR(255),
senha VARCHAR(255)
);

CREATE  TABLE  Pessoa (
idPessoa INT  PRIMARY KEY,
nome VARCHAR(255),
logradouro VARCHAR(255),
cidade VARCHAR(255),
estado CHAR(2),
telefone VARCHAR(11),
email VARCHAR(255)
);
```

```
CREATE  TABLE  PessoaFisica (
idPessoaFisica INT  PRIMARY KEY,
cpf VARCHAR(11),
FOREIGN KEY (idPessoaFisica) REFERENCES Pessoa(idPessoa)
);

CREATE  TABLE  PessoaJuridica (
idPessoaJuridica INT  PRIMARY KEY,
cnpj VARCHAR(14),
FOREIGN KEY (idPessoaJuridica) REFERENCES Pessoa(idPessoa)
);

CREATE  TABLE  Produto (
idProduto INT  PRIMARY KEY,
nome VARCHAR(255),
quantidade INT,
precoVenda NUMERIC
);

CREATE  TABLE  MovimentoCompra (
idMovimentoCompra INT  PRIMARY KEY,
quantidade INT,
precoUnitario NUMERIC,
idProduto INT,
idFornecedor INT,
idOperador INT,
FOREIGN KEY (idProduto) REFERENCES Produto(idProduto),
FOREIGN KEY (idFornecedor) REFERENCES PessoaJuridica(idPessoaJuridic
FOREIGN KEY (idOperador) REFERENCES Usuario(idOperador)
);

CREATE  TABLE  MovimentoVenda (
idMovimentoVenda INT  PRIMARY KEY,
quantidade INT,
precoUnitario NUMERIC,
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  PessoaIdSequence
START  WITH  1
```

```
INCREMENT BY  1;
```

Resultado obtido:



Análise e Conclusão

Como são implementadas as diferentes cardinalidades, basicamente 1X1, 1XN ou NxN, em um banco de dados relacional?

Cardinalidades em bancos de dados relacionais são implementadas da seguinte forma:

- **1X1 (Um para Um):** Este tipo de relacionamento é estabelecido quando uma entidade está associada a no máximo uma outra

entidade. Geralmente, isso é implementado com chaves primárias e estrangeiras que são iguais ou com restrições de unicidade.

- **1XN (Um para Muitos):** Um relacionamento um para muitos ocorre quando uma entidade pode estar associada a várias entidades de outro tipo. Isso é implementado com uma chave estrangeira na entidade do lado "muitos" que aponta para a chave primária da entidade do lado "um".
- **NxN (Muitos para Muitos):** Um relacionamento muitos para muitos é representado por uma tabela de junção que contém chaves estrangeiras referenciando as chaves primárias das entidades envolvidas.

Que tipo de relacionamento deve ser utilizado para representar o uso de herança em bancos de dados relacionais?

A herança em bancos de dados relacionais é frequentemente representada por meio de uma estrutura de tabelas que reflete a relação "é-um" entre entidades. Isso é feito utilizando uma tabela principal que armazena os atributos comuns a todas as entidades e tabelas secundárias para cada subclasse, que contêm atributos específicos e uma chave estrangeira que referencia a tabela principal. Assim, as tabelas secundárias "herdam" os dados da tabela principal, simulando a herança de classes em programação orientada a objetos.

Como o SQL Server Management Studio permite a melhoria da produtividade nas tarefas relacionadas ao gerenciamento do banco de dados?

O SQL Server Management Studio (SSMS) melhora a produtividade ao fornecer uma interface gráfica intuitiva que facilita a administração do banco de dados, a execução de scripts SQL, a configuração de segurança, o monitoramento do desempenho e a manutenção de bancos de dados. Além disso, oferece ferramentas para automatizar tarefas rotineiras e complexas, aumentando a eficiência e reduzindo a possibilidade de erros.

👉 2º Procedimento – Alimentando a Base

Alimentação Inicial das Tabelas

Incluindo dados nas tabelas:

Inserção de Usuários:

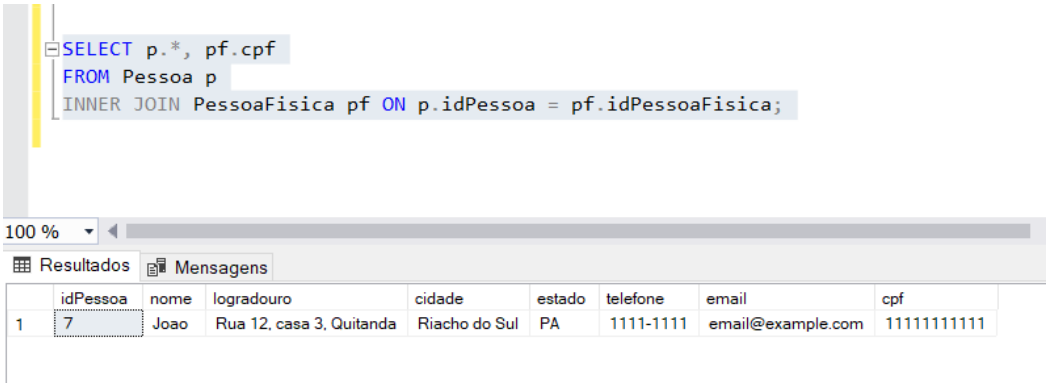
```
INSERT INTO Usuario (nome, senha) VALUES ('op1', 'op1'), ('op2', 'op
```

Inserção de Produtos:

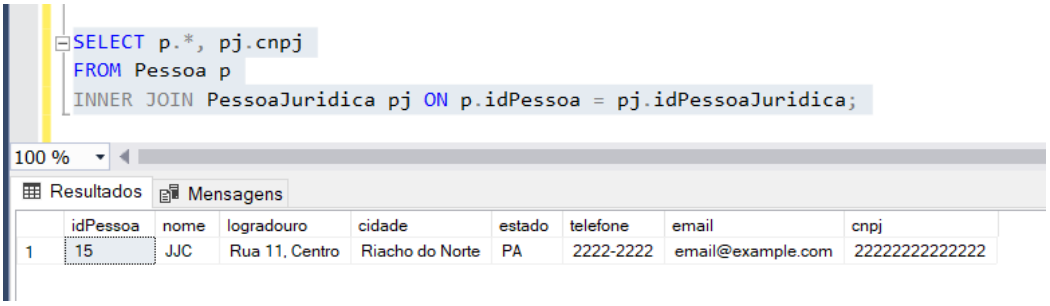
```
INSERT INTO Produto (idProduto, nome, quantidade, precoVenda) VALUES
INSERT INTO Produto (idProduto, nome, quantidade, precoVenda) VALUES
INSERT INTO Produto (idProduto, nome, quantidade, precoVenda) VALUES
```

Consultas Realizadas:

- Dados completos de pessoas físicas.



- Dados completos de pessoas jurídicas.



- Movimentações de entrada, com detalhes relevantes.

```sql
SELECT
    mc.idMovimentoCompra,
    pr.nome AS NomeProduto,
    p.nome AS NomeFornecedor,
    mc.quantidade,
    mc.precoUnitario,
    (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;
```

100 %

Resultados | Mensagens

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

- Movimentações de saída, com detalhes relevantes.

```sql
SELECT
    mv.idMovimentoVenda,
    pr.nome AS Produto,
    p.nome AS Comprador,
    mv.quantidade,
    mv.precoUnitario,
    (mv.quantidade * mv.precoUnitario) AS ValorTotal
FROM
    MovimentoVenda mv
INNER JOIN
    Produto pr ON mv.idProduto = pr.idProduto
INNER JOIN
    PessoaFisica pf ON mv.idComprador = pf.idPessoaFisica
INNER JOIN
    Pessoa p ON pf.idPessoaFisica = p.idPessoa;
```

100 %

Resultados | Mensagens

	idMovimentoVenda	Produto	Comprador	quantidade	precoUnitario	ValorTotal
1	1	Banana	Joao	20	4	80
2	4	Banana	Joao	15	2	30
3	5	Banana	Joao	10	3	30

- Valor total das entradas e saídas, agrupadas por produto.

```sql
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;
```

100 %

Resultados | Mensagens

	Produto	ValorTotalEntradas
1	Banana	155

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

100 %

Resultados | Mensagens

	Produto	ValorTotalSaidas
1	Banana	140

- Operadores que não efetuaram movimentações de entrada.

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

100 %

Resultados | Mensagens

	idOperador	nome	senha
1	2	op2	op2

- Valor total de entrada e saída, agrupado por operador.

```sql
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;
```

100 %

Resultados | Mensagens

	Operador	ValorTotalEntrada
1	op1	155

```sql
SELECT u.nome AS Operador, SUM(mv.quantidade * mv.precoUnitario) AS ValorTotalSaida
FROM MovimentoVenda mv
INNER JOIN Usuario u ON mv.idOperador = u.idOperador
GROUP BY u.nome;
```

100 %

Resultados | Mensagens

	Operador	ValorTotalSaida
1	op1	110
2	op2	30

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

```sql
SELECT pr.nome AS Produto, SUM(mv.quantidade * mv.precoUnitario) / SUM(mv.quantidade) AS PrecoMedioPonderado
FROM MovimentoVenda mv
INNER JOIN Produto pr ON mv.idProduto = pr.idProduto
GROUP BY pr.nome;
```

100 %

Resultados | Mensagens

	Produto	PrecoMedioPonderado
1	Banana	3.111111

Análise e Conclusão

a. Quais as diferenças no uso de sequence e

identity?

As principais diferenças entre `SEQUENCE` e `IDENTITY` são:

SEQUENCE: É um objeto criado e gerenciado pelo banco de dados que gera números sequenciais, não atrelados a uma tabela específica. Pode ser usado em múltiplas tabelas e não é reiniciado quando os registros são removidos.

- **Flexibilidade:** Uma `SEQUENCE` é um objeto separado que gera números sequenciais, não atrelado a uma tabela específica.
- **Reutilização:** Pode ser usada por múltiplas tabelas e colunas.
- **Controle:** Permite maior controle sobre o processo de geração de números, como definir o valor inicial, incremento, valor mínimo e máximo, e se a sequência deve reciclar.

IDENTITY: É uma propriedade de coluna específica de uma tabela que gera automaticamente valores numéricos sequenciais. É restrito a uma coluna em uma tabela e é geralmente reiniciado quando os registros são deletados e a tabela é recriada.

- **Simplicidade:** A propriedade `IDENTITY` é usada para gerar automaticamente valores numéricos sequenciais diretamente em uma coluna de uma tabela.
- **Especificidade:** Está diretamente ligada a uma coluna específica em uma tabela.
- **Facilidade de uso:** É mais fácil de configurar, pois requer menos parâmetros.

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

Chaves estrangeiras são essenciais para:

- **Integridade Referencial:** As chaves estrangeiras são essenciais para manter a integridade referencial entre tabelas, garantindo que as relações entre elas sejam preservadas.

- **Prevenção de Orfãos:** Evitam que registros "órfãos" existam em tabelas que dependem de outras para terem sentido.
- **Consistência de Dados:** Asseguram que apenas dados válidos sejam inseridos na tabela que possui a chave estrangeira.

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

Na **Álgebra Relacional**, operadores como `SELECT`, `PROJECT`, `JOIN`, `UNION`, `INTERSECT`, e `DIFFERENCE` são usados para manipular e consultar dados em bancos de dados relacionais.

- **Seleção (σ):** Filtra linhas.
- **Projeção (π):** Filtra colunas.
- **União (∪):** Combina resultados de duas consultas.
- **Diferença (-):** Retorna diferenças entre duas consultas.
- **Produto Cartesiano (X):** Combina todas as linhas de duas tabelas.
- **Junção (⋈):** Combina linhas baseadas em condições de junção.

No **Cálculo Relacional**, utiliza-se uma coleção de operadores lógicos como `AND`, `OR`, `NOT`, e `EXISTS`, que permitem a formulação de consultas baseadas em predicados e condições.

- **Predicados:** Expressões que retornam verdadeiro ou falso.
- **Quantificadores Universais e Existenciais (∀, ∃):** Usados para expressar consultas com condições "para todos" ou "existe".

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

O agrupamento em consultas SQL é feito com o comando `GROUP BY`, que é usado para agrupar linhas que têm os mesmos valores em colunas especificadas.

- **Requisito Obrigatório**: Quando utilizado, todas as colunas listadas na cláusula `SELECT` que não estão incluídas nas funções agregadas

(`COUNT` , `MAX` , `MIN` , `SUM` , `AVG`) devem estar presentes na cláusula `GROUP BY` .

Conclusão

> "A análise da Missão Prática proporcionou um aprendizado inicial sobre operações de bancos de dados, incluindo a criação de tabelas, a inserção de dados e o estabelecimento de relacionamentos."
>
> ***Lições principais:***
>
> - **Modelagem de Dados:** A aplicação prática na definição de tabelas e relacionamentos ressalta a importância de uma modelagem de dados eficaz.
> - **Chaves Estrangeiras:** O emprego de chaves estrangeiras evidenciou seu papel crucial na preservação da integridade dos dados