Nome: Wallace Felipe Tavares Moreira

Matrícula: 202109237331

Universidade: UNIVERSIDADE ESTÁCIO DE SÁ

Curso: Desenvolvimento Full Stack

Campus: Jardim América – Itaguaí/RJ

Disciplina: Nível 2 – Vamos manter as informaçoes!

Semestre Letivo: Terceiro Semestre

Objetivo da Prática

1 -Identificar os requisitos de um sistema e transformá-los no modelo adequado.

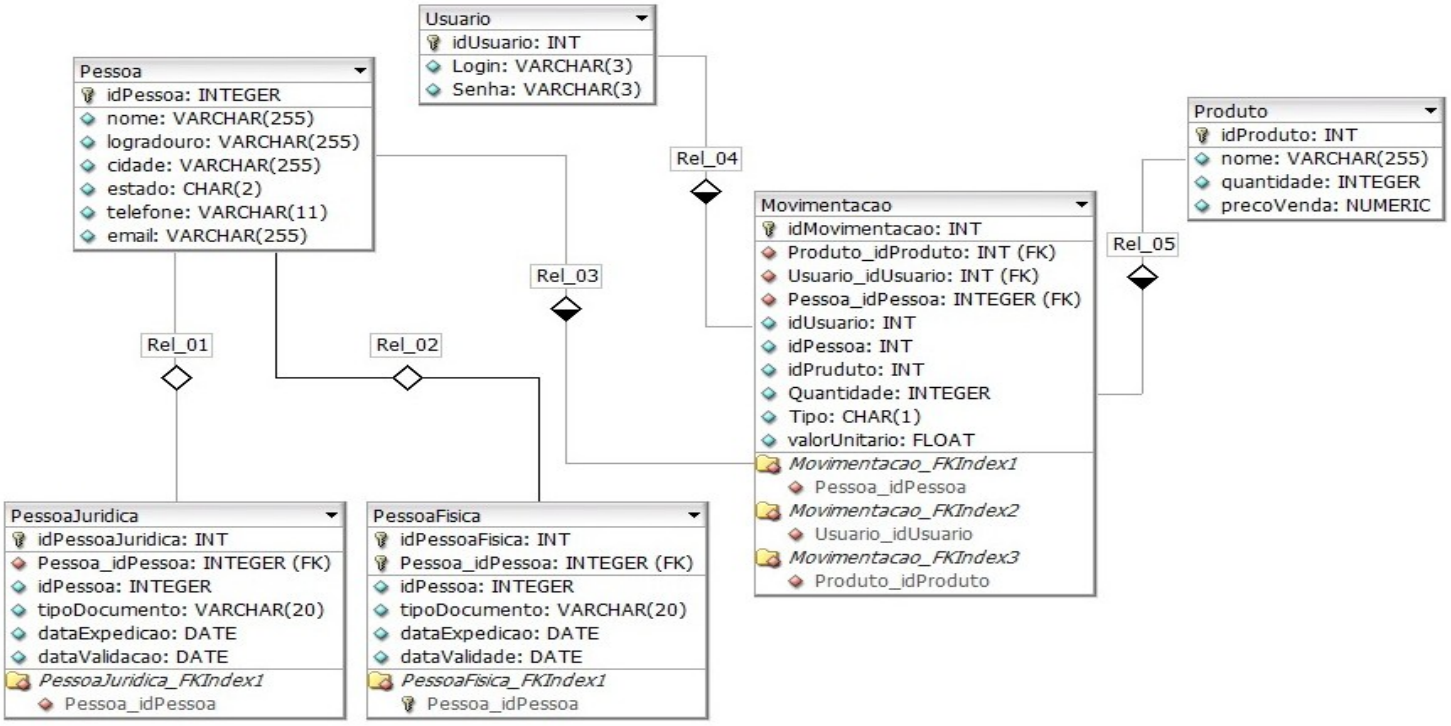
2 - Utilizar ferramentas de modelagem para bases de dados relacionais.

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 dados para um sistema simples, além de implementá-la, através da sintaxe SQL, na plataforma do SQL Server.

1º Procedimento



Análise e Conclusão - 1º Procedimento:

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

1- Cardinalidade 1:1 (Um para Um): Use uma chave estrangeira em uma tabela para se relacionar com a chave primária de outra tabela.

2- Cardinalidade 1:N (Um para Muitos): Adicione uma chave estrangeira na tabela "muitos" para referenciar a chave primária da tabela "um".

3- Cardinalidade N:N (Muitos para Muitos): Use uma tabela de junção para conectar duas tabelas principais, incluindo suas chaves primárias.

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

Para representar herança em bancos de dados relacionais, é recomendado o uso de um relacionamento do tipo "Tabelas Separadas" (ou "Tabelas Filhas"), onde cada subclasse tem sua própria tabela, contendo atributos específicos, e compartilha uma chave comum com a tabela da superclasse.

c) 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 melhora a produtividade no gerenciamento do banco de dados oferecendo uma interface intuitiva para executar tarefas, como consultas SQL, administração de servidores, criação de esquemas e visualização de dados, facilitando o desenvolvimento e a manutenção do banco de dados.

2º Procedimento

```
use [Loja];
GO
-- Tabela Usuario
CREATE TABLE [Usuario] (
  idUsuario INT IDENTITY(1,1) PRIMARY KEY,
  Login VARCHAR(20) NULL,
  Senha VARCHAR(20) NULL
);
GO
-- Inserindo dados na tabela Usuario
INSERT INTO [Usuario] (Login, Senha)
VALUES ('op1', 'op1'), ('op2', 'op2');

SELECT * FROM [Usuario];
GO
================================================================================

-- Tabela Produto
CREATE TABLE [Produto] (
  idProduto INT IDENTITY(1,1) NOT NULL,
  nome VARCHAR(255) NULL,
  quantidade INT NULL,
  precoVenda FLOAT,
  PRIMARY KEY (idProduto)
);
-- Inserindo dados na tabela Produto
INSERT INTO [Produto] (nome, quantidade, precoVenda)
VALUES ('Banana', 100, 5.01),
       ('Laranja', 500, 2.02),
       ('Manga', 800, 4.03);
SELECT * FROM [Produto];
GO
```

```
==========================================================================
-- Tabela Pessoa
CREATE TABLE [Pessoa] (
  idPessoa INT IDENTITY(1,1) NOT NULL,
  nome VARCHAR(255) NULL,
  logradouro VARCHAR(255) NULL,
  cidade VARCHAR(255) NULL,
  estado CHAR(2) NULL,
  telefone VARCHAR(11) NULL,
  email VARCHAR(255) NULL,
  PRIMARY KEY (idPessoa)
);

-- Inserindo dados na tabela Pessoa
INSERT INTO [Pessoa] (nome, logradouro, cidade, estado, telefone, email)
VALUES ('Wallace Felipe Tavares', 'Rua Chile', 'Rio de Janeiro', 'RJ',
'11991876543', 'wfelipe@gmail.com'),
            ('Felipe Wallace Tavares', 'Rua Chile', 'Rio de Janeiro', 'RJ',
'12345678909', 'wfeli@gmail.com'),
             ('Bingo Trêsbolas', 'Rua Ana Augusta', 'Rio de Janeiro', 'RJ',
'11991098789', 'wfee@gmail.com'),
         ('Padaria Norte', 'Rua México', 'Rio de Janeiro', 'RJ', '11909878654',
'lipe@gmail.com');


SELECT * FROM [Pessoa];
GO
```

```
==========================================================================
-- Tabela PessoaJuridica
CREATE TABLE [PessoaJuridica] (
  idPessoaJuridica INT IDENTITY(1,1) NOT NULL,
  idPessoa INT NOT NULL,
  nome VARCHAR(255) NULL,
  logradouro VARCHAR(255) NULL,
  cidade VARCHAR(255) NULL,
  estado CHAR(2) NULL,
  telefone VARCHAR(11) NULL,
  email VARCHAR(255) NULL,
  CNPJ VARCHAR(14) NULL,
  PRIMARY KEY (idPessoaJuridica),
  CONSTRAINT FK_PessoaJuridica_Pessoa
  FOREIGN KEY (idPessoa)
  REFERENCES [Pessoa] (idPessoa)
);
-- Inserindo dados na tabela PessoaJuridica
INSERT INTO [PessoaJuridica] (idPessoa, nome, logradouro, cidade, estado,
telefone, email, CNPJ)
VALUES (1, 'Nome PJ 1', 'Rua PJ 1', 'Cidade PJ 1', 'RJ', '12345678901',
'emailpj1@gmail.com', '18263258000146'),
           (2, 'Nome PJ 2', 'Rua PJ 2', 'Cidade PJ 2', 'RJ', '12345678902',
'emailpj2@gmail.com', '18263258000146');


-- Atualização dos dados na PessoaJuridica com os CNPJ
UPDATE [PessoaJuridica]
SET CNPJ = '18263258000146'
WHERE idPessoaJuridica = 1;


UPDATE [PessoaJuridica]
SET CNPJ = '18263258000146'
WHERE idPessoaJuridica = 2;


DROP TABLE IF EXISTS [PessoaJuridica];
DROP TABLE [PessoaJuridica];
SELECT * FROM [PessoaJuridica];
GO
```

```
==========================================================================
-- Tabela PessoaFisica
CREATE TABLE [PessoaFisica] (
  idPessoaFisica INT IDENTITY(1,1) NOT NULL,
  idPessoa INT NOT NULL,
  nome VARCHAR(255) NULL,
  logradouro VARCHAR(255) NULL,
  cidade VARCHAR(255) NULL,
  estado CHAR(2) NULL,
  telefone VARCHAR(11) NULL,
  email VARCHAR(255) NULL,
  CPF VARCHAR(14) NULL,
  PRIMARY KEY(idPessoaFisica),
  CONSTRAINT FK_PessoaFisica_Pessoa
  FOREIGN KEY (idPessoa)
  REFERENCES [Pessoa] (idPessoa)
);
INSERT INTO [PessoaFisica] (idPessoa, nome, logradouro, cidade, estado, telefone, email, CPF)
VALUES (3, 'Nome PF 1', 'Rua PF 1', 'Cidade PF 1', 'RJ', '12345678903', 'emailpf1@gmail.com', '022094775-40'),
           (4, 'Nome PF 2', 'Rua PF 2', 'Cidade PF 2', 'RJ', '12345678904', 'emailpf2@gmail.com', '022055785-50');


-- Atualização dos dados na tabela PessoaFisica com os CPF
UPDATE [PessoaFisica]
SET CPF = '022094775-40'
WHERE idPessoaFisica = 1;
UPDATE [PessoaFisica]
SET CPF = '022055785-50'
WHERE idPessoaFisica = 2;
DROP TABLE IF EXISTS [PessoaFisica];
DROP TABLE [PessoaFisica];
SELECT * FROM [PessoaFisica];
GO
```

```
================================================================================
-- Tabela Movimentacao
CREATE TABLE [Movimentacao] (
  id_movimento INT IDENTITY(1,1) NOT NULL,
  idUsuario INT NULL,
  idPessoa INT NULL,
  idProduto INT NULL,
  quantidade INT NOT NULL,
  tipo CHAR(1) NULL,
  valor_unitario FLOAT NULL,
  CONSTRAINT PK_Movimentacao PRIMARY KEY (id_movimento),
  CONSTRAINT FK_Usuario FOREIGN KEY (idUsuario)
    REFERENCES [Usuario] (idUsuario),
  CONSTRAINT FK_Pessoa FOREIGN KEY (idPessoa)
    REFERENCES [Pessoa] (idPessoa),
  CONSTRAINT FK_Produto FOREIGN KEY (idProduto)
    REFERENCES [Produto] (idProduto)
);

-- Inserindo dados na tabela Movimentacao
INSERT INTO [Movimentacao] (idUsuario, idPessoa, idProduto, quantidade, tipo,
valor_unitario)
VALUES (1, 1, 1, 150, 'E', 8.90),
       (2, 2, 2, 150, 'S', 8.90);

DROP TABLE IF EXISTS [Movimentacao];
DROP TABLE [Movimentacao];
SELECT * FROM [Movimentacao];
```

Estácio

Efetuar as seguintes consultas sobre os dados inseridos

```sql
-- a) Dados completos de pessoas físicas.
SELECT * FROM PessoaFisica;
```

```sql
-- b) Dados completos de pessoas jurídicas.
SELECT * FROM PessoaJuridica;
```

```sql
-- c) Movimentações de entrada, com produto, fornecedor, quantidade, preço unitário e valor total.
SELECT p.nome, m.idProduto, m.idPessoa, m.quantidade, m.valor_unitario, (m.quantidade * m.valor_unitario) AS valor_total
FROM Movimentacao AS m
JOIN Pessoa AS p ON m.idPessoa = p.idPessoa
WHERE m.tipo = 'E';
```

```sql
-- d) Movimentações de saída, com produto, cliente, quantidade, preço unitário e valor total.
SELECT p.nome, m.idProduto, m.idPessoa, m.quantidade, m.valor_unitario, (m.quantidade * m.valor_unitario) AS valor_total
FROM Movimentacao AS m
JOIN Pessoa AS p ON m.idPessoa = p.idPessoa
WHERE m.tipo = 'S';
```

```sql
-- e) Valor total das entradas agrupadas por produto.
SELECT idProduto, SUM(quantidade * valor_unitario) AS ValorTotalEntrada
FROM Movimentacao
WHERE tipo = 'E'
GROUP BY idProduto;
```

```sql
-- f) Valor total das saídas agrupadas por produto.
SELECT idProduto, SUM(quantidade * valor_unitario) AS ValorTotalSaida
FROM Movimentacao
WHERE tipo = 'S'
GROUP BY idProduto;
```

```sql
-- g) Operadores que não efetuaram movimentações de entrada (compra).
SELECT *
FROM Usuario
WHERE idUsuario NOT IN (SELECT DISTINCT idUsuario FROM Movimentacao WHERE tipo = 'E');
```

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

```sql
-- i) Valor total de saída, agrupado por operador.
SELECT p.nome AS operador, SUM(m.quantidade * m.valor_unitario) AS valor_total_saida
FROM Movimentacao AS m
JOIN Pessoa AS p ON m.idPessoa = p.idPessoa
WHERE m.tipo = 'S'
GROUP BY p.nome;
```

```sql
---J) Valor médio de venda por produto, utilizando média ponderada.
SELECT idProduto, SUM(quantidade * valor_unitario) / SUM(quantidade) AS ValorMedioVenda
FROM Movimentacao
WHERE tipo = 'S'
GROUP BY idProduto;
```

Análise e Conclusão - 2º Procedimento:

a) Diferenças no Uso de Sequence e Identity:

Sequence: É um objeto genérico de banco de dados que gera valores exclusivos em ordem sequencial. Pode ser usado em várias tabelas e em diferentes sistemas de gerenciamento de banco de dados (DBMS).

Identity: É um recurso específico de alguns DBMS, como o SQL Server e o MySQL, usado para criar colunas que geram automaticamente valores exclusivos e crescentes. Geralmente é usado como chave primária em uma tabela.

b) Importância das Chaves Estrangeiras para a Consistência do Banco:

As chaves estrangeiras são vitais para garantir a consistência dos dados em um banco de dados. Elas estabelecem relacionamentos entre tabelas, evitam dados "órfãos", facilitam consultas complexas e ajudam na manutenção do banco de dados.

c) Operadores do SQL Pertencentes à Álgebra Relacional e Definidos no Cálculo Relacional:

Na linguagem SQL, a maioria dos operadores pertence à álgebra relacional, enquanto o cálculo relacional é mais usado para especificar consultas. Operadores comuns da álgebra relacional em SQL incluem SELECT, JOIN, UNION, entre outros, enquanto o cálculo relacional é usado de maneira mais teórica para descrever consultas. Em resumo, a álgebra relacional é a base das operações práticas em SQL.

d) Agrupamento em Consultas e Requisito Obrigatório:

Em consultas SQL, o agrupamento é realizado com a cláusula "GROUP BY". O requisito obrigatório é listar todas as colunas não agregadas na cláusula "GROUP BY". Isso permite agrupar registros com base em colunas específicas e aplicar funções de agregação, como AVG, COUNT, MAX, MIN ou SUM, aos grupos resultantes. Isso é útil para resumir dados em consultas.