Estácio	**UNIVERSIDADE ESTACIO DE SÁ** POLO PARANGABA – FORTALEZA/CE DESENVOLVIMENTO FULL STACK - 22.3 Relatório da Missão Prática | Nível 3 | Mundo 3
Aluno:	Jefferson Ponte Pessoa
Professor:	Simone Ingrid Monteiro Gama
Repositório:	https://github.com/jeffersonkako/M3-nivel2

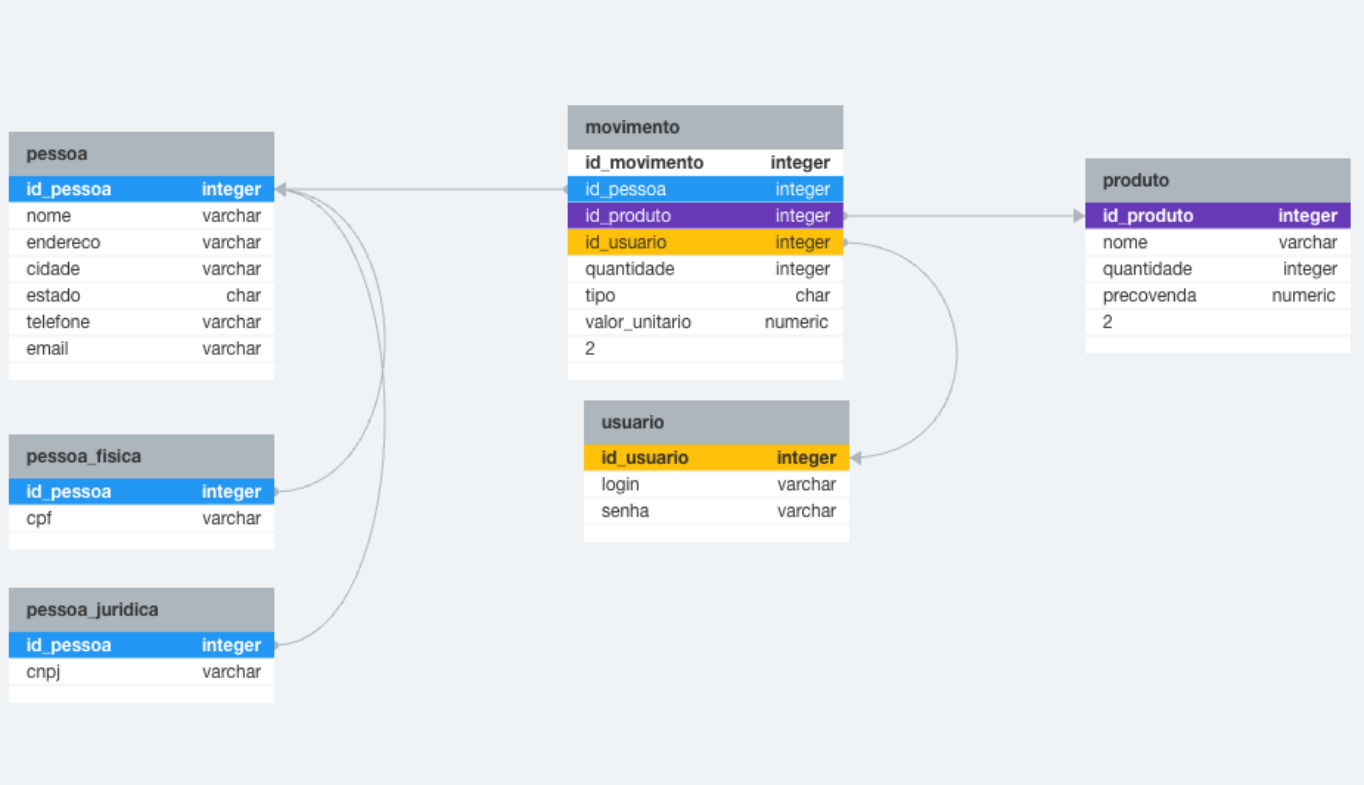
Título da Prática: 1º Procedimento | Criando o Banco de Dados

Objetivos 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)
- 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.

Códigos:

Tabela modelada no DBDesigner WEB
(Uso macOS e o programa não existe para meu sistema operacional.)



SQL do banco de dados:

```sql
CREATE TABLE [pessoa] (
    id_pessoa integer NOT NULL IDENTITY(1,1),
    nome varchar(255) NOT NULL,
    endereco varchar(255) NOT NULL,
    cidade varchar(255) NOT NULL,
    estado char(2) NOT NULL,
    telefone varchar(11) NOT NULL,
    email varchar(255) NOT NULL,
    CONSTRAINT [PK_PESSOA] PRIMARY KEY CLUSTERED ([id_pessoa] ASC)
)
GO


CREATE TABLE [pessoa_fisica] (
    id_pessoa integer NOT NULL,
    cpf varchar(11) NOT NULL,
    CONSTRAINT [PK_PESSOA_FISICA] PRIMARY KEY CLUSTERED ([id_pessoa] ASC),
    CONSTRAINT [FK_PESSOA_FISICA_PESSOA] FOREIGN KEY ([id_pessoa]) REFERENCES [pessoa] ([id_pessoa])
    ON UPDATE CASCADE
    ON DELETE CASCADE
)
GO


CREATE TABLE [pessoa_juridica] (
    id_pessoa integer NOT NULL,
    cnpj varchar(20) NOT NULL,
    CONSTRAINT [PK_PESSOA_JURIDICA] PRIMARY KEY CLUSTERED ([id_pessoa] ASC),
    CONSTRAINT [FK_PESSOA_JURIDICA_PESSOA] FOREIGN KEY ([id_pessoa]) REFERENCES [pessoa] ([id_pessoa])
    ON UPDATE CASCADE
    ON DELETE CASCADE
)
GO


CREATE TABLE [produto] (
    id_produto integer NOT NULL IDENTITY(1,1),
    nome varchar(255) NOT NULL,
    quantidade integer NOT NULL,
    precoVenda numeric(10,2) NOT NULL,
    CONSTRAINT [PK_PRODUTO] PRIMARY KEY CLUSTERED ([id_produto] ASC)
)
GO


CREATE TABLE [usuario] (
    id_usuario integer NOT NULL IDENTITY(1,1),
    login varchar(25) NOT NULL,
    senha varchar(25) NOT NULL,
    CONSTRAINT [PK_USUARIO] PRIMARY KEY CLUSTERED ([id_usuario] ASC)
)
GO


CREATE TABLE [movimento] (
    id_movimento integer NOT NULL IDENTITY(1,1),
    id_pessoa integer NOT NULL,
    id_produto integer NOT NULL,
    id_usuario integer NOT NULL,
    quantidade integer NOT NULL,
    tipo char(1) NOT NULL,
    valor_unitario numeric(10,2) NOT NULL,
    CONSTRAINT [PK_MOVIMENTO] PRIMARY KEY CLUSTERED ([id_movimento] ASC)
)
GO

ALTER TABLE [movimento] WITH CHECK ADD CONSTRAINT [movimento_fk0] FOREIGN KEY ([id_pessoa])
REFERENCES [pessoa] ([id_pessoa])
ON UPDATE CASCADE
ON DELETE CASCADE
GO
ALTER TABLE [movimento] CHECK CONSTRAINT [movimento_fk0]
GO

ALTER TABLE [movimento] WITH CHECK ADD CONSTRAINT [movimento_fk1] FOREIGN KEY ([id_produto])
REFERENCES [produto] ([id_produto])
ON UPDATE CASCADE
GO
ALTER TABLE [movimento] CHECK CONSTRAINT [movimento_fk1]
GO

ALTER TABLE [movimento] WITH CHECK ADD CONSTRAINT [movimento_fk2] FOREIGN KEY ([id_usuario])
REFERENCES [usuario] ([id_usuario])
ON UPDATE CASCADE
GO
ALTER TABLE [movimento] CHECK CONSTRAINT [movimento_fk2]
GO
```

Análise e Conclusão:

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

   1x1 (Um para Um): Você usa duas tabelas, onde uma tem uma chave primária que também é usada como chave estrangeira na outra tabela. Geralmente você faz isso quando quer separar informações que não são sempre necessárias. Por exemplo, informações pessoais em uma tabela e informações sensíveis como dados de login em outra.

   1xN (Um para Muitos): Neste caso, você tem uma chave primária em uma tabela e uma chave estrangeira na outra. É o tipo mais comum de relação. Por exemplo, um cliente (`ClienteID` como chave primária) pode fazer vários pedidos (cada pedido com um `ClienteID` como chave estrangeira).

   NxN (Muitos para Muitos): Aqui você precisa de uma tabela extra, chamada tabela de junção, que tem duas chaves estrangeiras que juntas formam uma chave primária composta. Isso é usado quando itens de uma tabela podem se relacionar com vários itens de outra tabela. Um bom exemplo é estudantes e cursos: um estudante pode se inscrever em vários cursos e um curso pode ter vários estudantes.

2. **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, o tipo de relacionamento mais comum é o "relacionamento de subtipo/supertipo". Nesse modelo, uma tabela base, que representa o supertipo, contém as colunas comuns a todas as entidades, enquanto tabelas adicionais representam os subtipos e contêm colunas específicas para as características únicas de cada subtipo. Cada tabela de subtipo tem uma chave estrangeira que se refere à chave primária da tabela supertipo, estabelecendo uma relação de herança.

3. **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) aumenta a produtividade ao oferecer uma interface gráfica intuitiva para gerenciamento de banco de dados, recursos de autocompletar e coloração de sintaxe para facilitar a escrita de código, ferramentas integradas para monitoramento e otimização de desempenho, e assistentes para simplificar tarefas complexas. Além disso, facilita o debugging, a gestão de segurança e a manutenção do banco de dados, tudo em uma única plataforma.

   No meu caso, tive que usar o DBeaver, que é uma ferramenta universal que faz o a mesma coisa que o SSMS faz, porem com algumas melhorias.

Título da Prática: 2º Procedimento | Alimentando a Base

Login: loja
Senha: R2t@7bM9
(tive que por essa senha pois no dbeaver não aceita a senha loja.)

Esquema de inserção de dados nas tabelas:

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

INSERT INTO produto (nome, quantidade, precoVenda)
VALUES ('Banana', 100, 5.00), ('Laranja', 500, 2.00), ('Manga', 800, 4.00);

INSERT INTO pessoa (nome, endereco, cidade, estado, telefone, email)
VALUES
('Pedro Cardoso', 'Avenida Brasil', 'Rio de Janeiro', 'RJ', '21988887777', 'pedro.cardoso@email.com'),
('Fernanda Souza', 'Travessa Joaquim', 'Salvador', 'BA', '71977776666', 'fernanda.souza@email.com'),
('COMPANY', 'Rua das Laranjeiras', 'Curitiba', 'PR', '41966665555', 'company@email.com');

INSERT INTO pessoa_fisica (id_pessoa, cpf)
VALUES (1, '11122233344'), (2, '44433322211');

INSERT INTO pessoa_juridica (id_pessoa, cnpj)
VALUES (3, '11223344000155');

INSERT INTO movimento (id_pessoa, id_produto, id_usuario, quantidade, tipo, valor_unitario)
VALUES
(1, 1, 1, 10, 'E', 15.50),
(1, 2, 1, 5, 'S', 45.00);
```

Esquema de buscas de dados nas tabelas:

```sql
--1
SELECT p.*, pf.cpf
FROM pessoa p
INNER JOIN pessoa_fisica pf ON p.id_pessoa = pf.id_pessoa;

--2
SELECT p.*, pj.cnpj
FROM pessoa p
INNER JOIN pessoa_juridica pj ON p.id_pessoa = pj.id_pessoa;

--3
SELECT m.*, p.nome as fornecedor, pr.nome as produto, m.quantidade, m.valor_unitario, (m.quantidade * m.valor_unitario) as valor_total
FROM movimento m
INNER JOIN pessoa p ON m.id_pessoa = p.id_pessoa
INNER JOIN produto pr ON m.id_produto = pr.id_produto
WHERE m.tipo = 'E';

--4
SELECT m.*, p.nome as comprador, pr.nome as produto, m.quantidade, m.valor_unitario, (m.quantidade * m.valor_unitario) as valor_total
FROM movimento m
INNER JOIN pessoa p ON m.id_pessoa = p.id_pessoa
INNER JOIN produto pr ON m.id_produto = pr.id_produto
WHERE m.tipo = 'S';

--5
SELECT pr.nome, SUM(m.quantidade * m.valor_unitario) as valor_total_entradas
FROM movimento m
INNER JOIN produto pr ON m.id_produto = pr.id_produto
WHERE m.tipo = 'E'
GROUP BY pr.nome;

--6
SELECT pr.nome, SUM(m.quantidade * m.valor_unitario) as valor_total_saidas
FROM movimento m
INNER JOIN produto pr ON m.id_produto = pr.id_produto
WHERE m.tipo = 'S'
GROUP BY pr.nome;

--7
SELECT u.*
FROM usuario u
LEFT JOIN movimento m ON u.id_usuario = m.id_usuario AND m.tipo = 'E'
WHERE m.id_movimento IS NULL;

--8
SELECT u.login, SUM(m.valor_unitario * m.quantidade) as valor_total_entradas
FROM movimento m
INNER JOIN usuario u ON m.id_usuario = u.id_usuario
WHERE m.tipo = 'E'
GROUP BY u.login;


--9
SELECT u.login, SUM(m.valor_unitario * m.quantidade) as valor_total_saidas
FROM movimento m
INNER JOIN usuario u ON m.id_usuario = u.id_usuario
WHERE m.tipo = 'S'
GROUP BY u.login;

--10
SELECT pr.nome, SUM(m.valor_unitario * m.quantidade) / SUM(m.quantidade) as media_ponderada_venda
FROM movimento m
INNER JOIN produto pr ON m.id_produto = pr.id_produto
WHERE m.tipo = 'S'
GROUP BY pr.nome;
```

Análise e Conclusão

1. Quais as diferenças no uso de sequence e identity?

Sequence (Sequência): Refere-se a uma ordem específica de elementos ou uma série de valores que seguem uma regra particular. Em bancos de dados, uma sequência pode ser usada para gerar números únicos consecutivos, geralmente utilizados como chaves primárias. Na programação, uma sequência pode ser uma lista, um array, ou qualquer outro tipo de coleção ordenada de elementos.

Identity (Identidade): Em matemática, a identidade é um valor que, quando usado em operações, não altera os outros elementos. Por exemplo, em multiplicação, o número 1 é a identidade porque qualquer número multiplicado por 1 permanece inalterado. Em bancos de dados, um campo de identidade é geralmente um campo que tem um valor único para cada registro, como uma chave primária autoincrementável. Em programação, uma função identidade é aquela que sempre retorna o mesmo valor que foi usado como argumento.

2. Qual a importância das chaves estrangeiras para a consistência do banco?

As chaves estrangeiras são essenciais em bancos de dados relacionais, pois garantem a integridade referencial ao vincular tabelas de maneira que as alterações em uma tabela se reflitam em outras relacionadas, prevenindo inconsistências e dados órfãos, e facilitam a manutenção e a compreensão das relações entre os dados.

3. Quais operadores do SQL pertencem a álgebra relacional e quais são definidos no cálculo relacional?

A álgebra relacional e o cálculo relacional são duas linguagens de consulta fundamentais para o modelo relacional em banco de dados, mas elas operam de maneiras distintas. A álgebra relacional utiliza operadores como seleção, projeção, união, diferença, produto cartesiano e junção, que são procedimentos para obter novas relações a partir de relações existentes. Por outro lado, o cálculo relacional é baseado em fórmulas lógicas e utiliza quantificadores universais e existenciais, assim como predicados que descrevem as propriedades que as tuplas devem satisfazer, em vez de operadores explícitos para manipulação de tuplas. Enquanto a álgebra relacional diz "como" obter o resultado, o cálculo relacional define "o que" se deseja obter como resultado, deixando o "como" para o sistema de gerenciamento do banco de dados resolver.

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

O agrupamento em consultas, especialmente em bancos de dados SQL, é feito utilizando a cláusula `GROUP BY`, que agrupa linhas que têm os mesmos valores em colunas específicas em conjuntos resumidos. O requisito obrigatório para utilizar `GROUP BY` é que, para cada coluna que você selecionar na sua consulta que não está contida na cláusula `GROUP BY`, você deve aplicar uma função de agregação, como `COUNT()`, `SUM()`, `AVG()`, `MAX()`, ou `MIN()`. Essas funções de agregação são usadas para realizar cálculos sobre um conjunto de linhas agrupadas, fornecendo um único resultado por grupo. Por exemplo, se você quiser saber a quantidade de vendas por produto, você agruparia as linhas da tabela de vendas pelo identificador do produto e usaria a função `COUNT()` para contar o número de linhas em cada grupo.