Estácio

Campus: Florianópolis
Curso: Desenvolvedor Full Stack
Disciplina: Missão Prática | Nível 2 | Mundo 3
Nome: Sérgio Koerich Pereira

Missão Prática | Nível 2 | Mundo 3

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.

1º Procedimento | Criando o Banco de Dados

Códigos:

```
-- Criação da sequência para IDs
CREATE SEQUENCE seq_pessoa_id
    START WITH 1
    INCREMENT BY 1;

-- Tabela de Usuários
CREATE TABLE Usuarios (
    ID INT PRIMARY KEY IDENTITY,
    Nome NVARCHAR(100),
    Senha NVARCHAR(100)
);

-- Tabela de Pessoas
CREATE TABLE Pessoas (
    ID INT PRIMARY KEY DEFAULT NEXT VALUE FOR seq_pessoa_id,
    Tipo NVARCHAR(10) CHECK (Tipo IN ('Física', 'Jurídica')),
    CPF NVARCHAR(11) NULL,
```

```
    CNPJ NVARCHAR(14) NULL,
    Nome NVARCHAR(100),
    Endereco NVARCHAR(255),
    Contato NVARCHAR(100)
);

-- Tabela de Produtos
CREATE TABLE Produtos (
    ID INT PRIMARY KEY IDENTITY,
    Nome NVARCHAR(100),
    Quantidade INT,
    PrecoVenda DECIMAL(10, 2)
);

-- Tabela de Movimentos de Compra
CREATE TABLE MovimentosCompra (
    ID INT PRIMARY KEY IDENTITY,
    ID_Produto INT FOREIGN KEY REFERENCES Produtos(ID),
    ID_Pessoa_Juridica INT FOREIGN KEY REFERENCES Pessoas(ID),
    Quantidade INT,
    PrecoUnitario DECIMAL(10, 2),
    Data DATETIME
);

-- Tabela de Movimentos de Venda
CREATE TABLE MovimentosVenda (
    ID INT PRIMARY KEY IDENTITY,
    ID_Produto INT FOREIGN KEY REFERENCES Produtos(ID),
    ID_Pessoa_Fisica INT FOREIGN KEY REFERENCES Pessoas(ID),
    Quantidade INT,
    PrecoVenda DECIMAL(10, 2),
    Data DATETIME
);
```

1. Como são implementadas as diferentes cardinalidades (1x1, 1XN ou NxN) em um banco de dados relacional?

- **1x1 (Um para Um):**
    - **Implementação:** Em um relacionamento 1x1, cada registro em uma tabela está associado a no máximo um registro em outra tabela. Para implementar isso, você pode usar uma chave primária em ambas as tabelas e garantir que a chave estrangeira seja única.

Exemplo: Se temos uma tabela `Pessoa` e uma tabela `Passaporte`, onde cada pessoa tem exatamente um passaporte e vice-versa, a tabela `Passaporte` terá uma coluna `PessoaID` que é uma chave estrangeira e também uma chave primária.

```
CREATE TABLE Pessoa (
    ID INT PRIMARY KEY,
    Nome NVARCHAR(100)
);

CREATE TABLE Passaporte (
    ID INT PRIMARY KEY,
    PessoaID INT UNIQUE,
    Numero NVARCHAR(20),
    FOREIGN KEY (PessoaID) REFERENCES Pessoa(ID)
);
```

- **1XN (Um para Muitos):**
  - **Implementação:** Em um relacionamento 1XN, um registro em uma tabela pode estar associado a múltiplos registros em outra tabela. Para implementar isso, você usa uma chave primária na tabela "um" e uma chave estrangeira na tabela "muitos".

Exemplo: Se uma tabela `Categoria` tem vários `Produtos`, a tabela `Produtos` terá uma coluna `CategoriaID` que referencia a chave primária de `Categoria`.

```
CREATE TABLE Categoria (
    ID INT PRIMARY KEY,
    Nome NVARCHAR(100)
);

CREATE TABLE Produtos (
    ID INT PRIMARY KEY,
    Nome NVARCHAR(100),
    CategoriaID INT,
    FOREIGN KEY (CategoriaID) REFERENCES Categoria(ID)
);
```

- **NxN (Muitos para Muitos):**
  - **Implementação:** Em um relacionamento NxN, múltiplos registros em uma tabela estão associados a múltiplos registros em outra tabela. Para implementar isso, você cria uma tabela de junção (ou tabela intermediária) que contém chaves estrangeiras que referenciam as tabelas envolvidas.

Exemplo: Se temos `Estudantes` e `Cursos`, onde um estudante pode se inscrever em vários cursos e um curso pode ter vários estudantes, você criaria uma tabela de junção `EstudanteCurso`.

```
CREATE TABLE Estudantes (
    ID INT PRIMARY KEY,
    Nome NVARCHAR(100)
);

CREATE TABLE Cursos (
    ID INT PRIMARY KEY,
    Nome NVARCHAR(100)
);

CREATE TABLE EstudanteCurso (
    EstudanteID INT,
    CursoID INT,
    PRIMARY KEY (EstudanteID, CursoID),
    FOREIGN KEY (EstudanteID) REFERENCES Estudantes(ID),
    FOREIGN KEY (CursoID) REFERENCES Cursos(ID)
);
```

-

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

- **Tipo de Relacionamento para Herança:**
  - **Herança em Bancos de Dados Relacionais:** Para representar herança em um banco de dados relacional, você pode usar um dos seguintes métodos:

Tabela Única de Herança: Toda a hierarquia de classes é representada em uma única tabela. Esta tabela contém colunas para todos os atributos das classes pai e filho.

```
CREATE TABLE Pessoas (
    ID INT PRIMARY KEY,
    Tipo NVARCHAR(10),
    Nome NVARCHAR(100),
    CPF NVARCHAR(11) NULL,
    CNPJ NVARCHAR(14) NULL
);
```

-

Tabela de Superclasse e Tabelas de Subclasse: Usa uma tabela para a superclasse e tabelas separadas para cada subclasse. A tabela da subclasse contém uma chave estrangeira que referencia a tabela da superclasse.

```
CREATE TABLE Pessoas (
    ID INT PRIMARY KEY,
    Nome NVARCHAR(100)
);

CREATE TABLE PessoasFisicas (
    ID INT PRIMARY KEY,
    CPF NVARCHAR(11),
    FOREIGN KEY (ID) REFERENCES Pessoas(ID)
);

CREATE TABLE PessoasJuridicas (
    ID INT PRIMARY KEY,
    CNPJ NVARCHAR(14),
    FOREIGN KEY (ID) REFERENCES Pessoas(ID)
);
```

-
- **Tabela de Subclasse com Herança Completa:** Cria tabelas para cada nível da hierarquia, onde cada subclasse tem sua própria tabela e pode referenciar a tabela pai para atributos comuns.

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

- **Interface Gráfica Intuitiva:**
  - O SSMS fornece uma interface gráfica que facilita a criação, modificação e exclusão de objetos de banco de dados como tabelas, views e stored procedures sem a necessidade de escrever código SQL manualmente.
- **Ferramentas de Design:**
  - O SSMS oferece ferramentas de design de banco de dados que permitem criar diagramas de banco de dados visualmente, o que ajuda a entender e modificar a estrutura do banco de dados de forma mais intuitiva.
- **Assistentes e Modelos:**
  - Assistentes integrados para tarefas comuns, como a criação de novas tabelas, índices e usuários, ajudam a acelerar o processo de desenvolvimento e administração.
- **Execução e Monitoramento:**
  - O SSMS permite executar scripts SQL, monitorar o desempenho do banco de dados e revisar logs de atividades, facilitando a administração e o diagnóstico de problemas.
- **Suporte a Transações:**
  - O gerenciamento de transações é facilitado com suporte a rollback e commit, permitindo a execução segura de operações complexas sem comprometer a integridade dos dados.

- **Automação de Tarefas:**
    - Recursos como o SQL Server Agent permitem agendar e automatizar tarefas administrativas, como backups e manutenção de índices, reduzindo a necessidade de intervenção manual.

2º Procedimento | Alimentando a Base

1.2 Inserir Dados na Tabela de Usuários

Execute o seguinte script SQL para inserir os usuários com senhas especificadas:

```
-- Inserir dados na tabela de usuários
INSERT INTO Usuarios (Nome, Senha) VALUES ('op1', 'op1');
INSERT INTO Usuarios (Nome, Senha) VALUES ('op2', 'op2');
```

1.3 Inserir Produtos na Base de Dados

Execute o script SQL para inserir produtos na tabela `Produtos`:

```
-- Inserir dados na tabela de produtos
INSERT INTO Produtos (Nome, Quantidade, PrecoVenda) VALUES ('Produto
A', 100, 10.00);
INSERT INTO Produtos (Nome, Quantidade, PrecoVenda) VALUES ('Produto
B', 200, 20.00);
INSERT INTO Produtos (Nome, Quantidade, PrecoVenda) VALUES ('Produto
C', 150, 15.00);
```

1.4 Criar Pessoas Físicas e Jurídicas

Obter o próximo ID de pessoa a partir da sequência:

```
-- Obter o próximo ID de pessoa
DECLARE @next_pessoa_id INT;
SET @next_pessoa_id = NEXT VALUE FOR seq_pessoa_id;
```

Inserir pessoas comuns:

```
-- Inserir dados comuns na tabela de pessoas
```

```
INSERT INTO Pessoas (ID, Tipo, Nome, Endereco, Contato) VALUES
(@next_pessoa_id, 'Física', 'Pessoa Física 1', 'Rua A, 123',
'1234-5678');
INSERT INTO Pessoas (ID, Tipo, Nome, Endereco, Contato) VALUES (NEXT
VALUE FOR seq_pessoa_id, 'Jurídica', 'Pessoa Jurídica 1', 'Avenida
B, 456', '8765-4321');
```

Inserir pessoas físicas:

```
-- Inserir dados na tabela de pessoas físicas
INSERT INTO PessoasFisicas (ID, CPF) VALUES (@next_pessoa_id,
'123.456.789-00');
```

Inserir pessoas jurídicas:

```
-- Inserir dados na tabela de pessoas jurídicas
INSERT INTO PessoasJuridicas (ID, CNPJ) VALUES (NEXT VALUE FOR
seq_pessoa_id, '12.345.678/0001-99');
```

1.5 Criar Movimentações

Inserir movimentações de entrada e saída:

```
-- Inserir movimentações de entrada
INSERT INTO MovimentosCompra (ID_Produto, ID_Pessoa_Juridica,
Quantidade, PrecoUnitario, Data)
VALUES (1, 2, 50, 10.00, GETDATE());

-- Inserir movimentações de saída
INSERT INTO MovimentosVenda (ID_Produto, ID_Pessoa_Fisica,
Quantidade, PrecoVenda, Data)
VALUES (1, 1, 30, 10.00, GETDATE());
```

2. Consultas sobre os Dados Inseridos

2.1 Dados Completos de Pessoas Físicas

```
-- Consultar dados completos de pessoas físicas
SELECT P.ID, P.Nome, P.Endereco, P.Contato, PF.CPF
FROM Pessoas P
```

```
JOIN PessoasFisicas PF ON P.ID = PF.ID
WHERE P.Tipo = 'Física';
```

2.2 Dados Completos de Pessoas Jurídicas

sql
Copiar código

```
-- Consultar dados completos de pessoas jurídicas
SELECT P.ID, P.Nome, P.Endereco, P.Contato, PJ.CNPJ
FROM Pessoas P
JOIN PessoasJuridicas PJ ON P.ID = PJ.ID
WHERE P.Tipo = 'Jurídica';
```

2.3 Movimentações de Entrada

```
-- Consultar movimentações de entrada
SELECT
    P.Nome AS Fornecedor,
    Pr.Nome AS Produto,
    MC.Quantidade,
    MC.PrecoUnitario,
    MC.Quantidade * MC.PrecoUnitario AS ValorTotal,
    MC.Data
FROM MovimentosCompra MC
JOIN Produtos Pr ON MC.ID_Produto = Pr.ID
JOIN Pessoas P ON MC.ID_Pessoa_Juridica = P.ID;
```

2.4 Movimentações de Saída

```
-- Consultar movimentações de saída
SELECT
    P.Nome AS Comprador,
    Pr.Nome AS Produto,
    MV.Quantidade,
    MV.PrecoVenda,
    MV.Quantidade * MV.PrecoVenda AS ValorTotal,
    MV.Data
FROM MovimentosVenda MV
JOIN Produtos Pr ON MV.ID_Produto = Pr.ID
```

```
JOIN Pessoas P ON MV.ID_Pessoa_Fisica = P.ID;
```

2.5 Valor Total das Entradas Agrupadas por Produto

```
-- Valor total das entradas agrupadas por produto
SELECT
    Pr.Nome AS Produto,
    SUM(MC.Quantidade * MC.PrecoUnitario) AS ValorTotal
FROM MovimentosCompra MC
JOIN Produtos Pr ON MC.ID_Produto = Pr.ID
GROUP BY Pr.Nome;
```

2.6 Valor Total das Saídas Agrupadas por Produto

```
-- Valor total das saídas agrupadas por produto
SELECT
    Pr.Nome AS Produto,
    SUM(MV.Quantidade * MV.PrecoVenda) AS ValorTotal
FROM MovimentosVenda MV
JOIN Produtos Pr ON MV.ID_Produto = Pr.ID
GROUP BY Pr.Nome;
```

2.7 Operadores que Não Efetuarão Movimentações de Entrada

```
-- Operadores que não efetuaram movimentações de entrada
SELECT U.Nome
FROM Usuarios U
LEFT JOIN MovimentosCompra MC ON U.ID = MC.ID_Usuario
WHERE MC.ID_Usuario IS NULL;
```

2.8 Valor Total de Entrada Agrupado por Operador

```
-- Valor total de entrada agrupado por operador
SELECT
    U.Nome AS Operador,
    SUM(MC.Quantidade * MC.PrecoUnitario) AS ValorTotal
FROM MovimentosCompra MC
JOIN Usuarios U ON MC.ID_Usuario = U.ID
```

```
GROUP BY U.Nome;
```

2.9 Valor Total de Saída Agrupado por Operador

```
-- Valor total de saída agrupado por operador
SELECT
    U.Nome AS Operador,
    SUM(MV.Quantidade * MV.PrecoVenda) AS ValorTotal
FROM MovimentosVenda MV
JOIN Usuarios U ON MV.ID_Usuario = U.ID
GROUP BY U.Nome;
```

2.10 Valor Médio de Venda por Produto (Média Ponderada)

```
-- Valor médio de venda por produto, utilizando média ponderada
SELECT
    Pr.Nome AS Produto,
    SUM(MV.Quantidade * MV.PrecoVenda) / NULLIF(SUM(MV.Quantidade),
0) AS ValorMedio
FROM MovimentosVenda MV
JOIN Produtos Pr ON MV.ID_Produto = Pr.ID
GROUP BY Pr.Nome;
```

Análise e Conclusão

1. Quais as diferenças no uso de SEQUENCE e IDENTITY?

- **SEQUENCE:**
  - **Descrição:** SEQUENCE é um objeto de banco de dados que gera uma sequência de números únicos que podem ser utilizados para criar valores de colunas ou outros identificadores.
  - **Flexibilidade:** Oferece flexibilidade na geração de números, podendo ser incrementada em qualquer ponto e não está atrelada diretamente a uma tabela específica.
  - **Uso:** Pode ser usada em várias tabelas e pode ser reiniciada, ter ciclos definidos, e ajustar o valor inicial e o incremento.

Exemplo de Criação:

```
CREATE SEQUENCE seq_pessoa_id
  START WITH 1
  INCREMENT BY 1;
```

    -
    - **Uso:** NEXT VALUE FOR seq_pessoa_id obtém o próximo valor da sequência.
- **IDENTITY:**
    - **Descrição:** IDENTITY é uma propriedade de coluna que gera valores únicos automaticamente quando uma nova linha é inserida na tabela. É específico para uma coluna dentro de uma tabela.
    - **Simplicidade:** Simples de usar para tabelas onde a coluna precisa gerar valores únicos automaticamente. O controle sobre o incremento e o valor inicial é mais restrito em comparação com SEQUENCE.
    - **Uso:** Associada a uma tabela durante a criação da coluna. Os valores são gerados automaticamente e não podem ser ajustados ou reiniciados facilmente.

Exemplo de Criação:

```
CREATE TABLE Pessoa (
    ID INT IDENTITY(1,1) PRIMARY KEY,
    Nome NVARCHAR(100)
);
```

    -
    - **Uso:** O valor da coluna ID é gerado automaticamente quando uma nova linha é inserida.

Principais Diferenças:

- SEQUENCE é mais flexível e pode ser usado em vários contextos, enquanto IDENTITY é restrito à coluna e tabela específicas.
- SEQUENCE pode ser manipulada diretamente para alterar valores, enquanto IDENTITY é gerada automaticamente pelo sistema.

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

- **Integridade Referencial:** Chaves estrangeiras garantem que os valores em uma coluna de uma tabela (chave estrangeira) correspondam aos valores de uma coluna de outra tabela (chave primária), mantendo a integridade dos dados.
- **Evita Dados Orfãos:** Impede a inserção de dados que não têm uma correspondência válida em outra tabela, evitando registros órfãos (referências inválidas).
- **Restrições de Exclusão e Atualização:** Permite definir ações a serem tomadas quando um registro referenciado é excluído ou atualizado, como cascata ou restrição, garantindo que a consistência dos dados seja mantida.

Exemplo:

```
CREATE TABLE Pedido (
    ID INT PRIMARY KEY,
    ClienteID INT,
```

```
    FOREIGN KEY (ClienteID) REFERENCES Cliente(ID)
);
```

- Neste exemplo, a tabela Pedido tem uma chave estrangeira ClienteID que deve corresponder a um valor existente na tabela Cliente.

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

- **Álgebra Relacional:** Define operações básicas para manipular e consultar dados em tabelas.

Seleção (σ): Filtra linhas com base em uma condição.

```
SELECT * FROM Tabela WHERE Condicao;
```

-

Projeção (π): Seleciona colunas específicas.

```
SELECT Coluna1, Coluna2 FROM Tabela;
```

-

União (∪): Combina resultados de duas consultas.

```
SELECT Coluna FROM Tabela1
UNION
SELECT Coluna FROM Tabela2;
```

-

Diferença (−): Retorna resultados que estão em uma tabela, mas não na outra.

```
SELECT Coluna FROM Tabela1
EXCEPT
SELECT Coluna FROM Tabela2;
```

-

Produto Cartesiano (×): Combina todas as linhas de duas tabelas.

```
SELECT * FROM Tabela1, Tabela2;
```

-

Junção (⋈): Combina linhas com base em uma condição.

```
SELECT * FROM Tabela1
JOIN Tabela2 ON Tabela1.Coluna = Tabela2.Coluna;
```

○

Renomeação (ρ): Renomeia tabelas ou colunas.

SELECT Coluna AS NovoNome FROM Tabela;

○

- **Cálculo Relacional:** Define operações baseadas em fórmulas lógicas e predicados para consultas.

Cálculo Relacional de Tuplas: Baseado em predicados sobre tuplas.

{ t | t ∈ Tabela ∧ Condicao(t) }

○

Cálculo Relacional de Domínios: Baseado em predicados sobre domínios de atributos.

{ <a1, a2, ..., an> | Condicao(a1, a2, ..., an) }

○

- O SQL combina elementos de ambos os paradigmas, utilizando a álgebra relacional para definir a estrutura das consultas e o cálculo relacional para descrever os predicados e condições.

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

- **Agrupamento em Consultas:**

Uso do GROUP BY: Para agregar dados e calcular funções agregadas (como SUM, COUNT, AVG, etc.) para grupos de registros.

SELECT Coluna, COUNT(*)
FROM Tabela
GROUP BY Coluna;

○

Funções Agregadas: Calcula valores agregados para cada grupo.
SELECT Coluna, SUM(Quantidade)
FROM Tabela
GROUP BY Coluna;

○

- **Requisito Obrigatório:**

Incluir Colunas no GROUP BY: Qualquer coluna na cláusula SELECT que não esteja dentro de uma função agregada deve ser incluída na cláusula GROUP BY. Caso contrário, ocorrerá um erro de execução.
-- Correto

```
SELECT Coluna, SUM(Quantidade)
FROM Tabela
GROUP BY Coluna;

-- Incorreto (se Coluna não estiver em GROUP BY)
SELECT Coluna, SUM(Quantidade)

FROM Tabela;
```