Primeira etapa: DBDesigner

Diagrama ER (Entidade-Relacionamento) que representa o modelo de dados:

==========

```
-- Tabela de Usuários
CREATE TABLE Usuarios (
    id INT PRIMARY KEY,
    nome VARCHAR(100),
    email VARCHAR(100),
    senha VARCHAR(100)
);

-- Tabela de Pessoas
CREATE TABLE Pessoas (
    id INT PRIMARY KEY,
    nome VARCHAR(100),
    tipo ENUM('Física', 'Jurídica'),
    cpf VARCHAR(14),
    cnpj VARCHAR(18),
    endereco VARCHAR(255),
    telefone VARCHAR(20)
);

-- Tabela de Produtos
CREATE TABLE Produtos (
    id INT PRIMARY KEY,
    nome VARCHAR(100),
    quantidade INT,
    preco DECIMAL(10, 2)
);

-- Tabela de Compras
CREATE TABLE Compras (
    id INT PRIMARY KEY,
    id_operador INT,
```

```
    id_fornecedor INT,
    id_produto INT,
    quantidade INT,
    preco_unitario DECIMAL(10, 2),
    FOREIGN KEY (id_operador) REFERENCES Usuarios(id),
    FOREIGN KEY (id_fornecedor) REFERENCES Pessoas(id),
    FOREIGN KEY (id_produto) REFERENCES Produtos(id)
);

-- Tabela de Vendas
CREATE TABLE Vendas (
    id INT PRIMARY KEY,
    id_operador INT,
    id_cliente INT,
    id_produto INT,
    quantidade INT,
    preco_unitario DECIMAL(10, 2),
    FOREIGN KEY (id_operador) REFERENCES Usuarios(id),
    FOREIGN KEY (id_cliente) REFERENCES Pessoas(id),
    FOREIGN KEY (id_produto) REFERENCES Produtos(id)
);
```

==========

Este código SQL cria cinco tabelas: Usuarios, Pessoas, Produtos, Compras e Vendas. A tabela Usuarios armazena informações dos usuários do sistema, como nome, e-mail e senha. A tabela Pessoas armazena informações de pessoas físicas e jurídicas, diferenciando-se pelo tipo e pelo CPF ou CNPJ.

A tabela Produtos armazena informações dos produtos, como nome, quantidade e preço. As tabelas Compras e Vendas registram movimentos de compra e venda, respectivamente, associando o operador, o cliente/fornecedor e o produto, juntamente com a quantidade e o preço unitário.

Segunda etapa: SQL Server Management Studio.

1) Utilização do editor de SQL para a criação das estruturas do modelo

```
-- Criar a tabela de Usuários
CREATE TABLE Usuarios (
    id INT PRIMARY KEY,
    nome VARCHAR(100),
    email VARCHAR(100),
    senha VARCHAR(100)
);

-- Criar a tabela de Pessoas
CREATE TABLE Pessoas (
    id INT PRIMARY KEY,
    nome VARCHAR(100),
    tipo VARCHAR(20),
    cpf VARCHAR(14),
    cnpj VARCHAR(18),
    endereco VARCHAR(255),
    telefone VARCHAR(20)
);

-- Criar a tabela de Produtos
CREATE TABLE Produtos (
    id INT PRIMARY KEY,
    nome VARCHAR(100),
    quantidade INT,
    preco DECIMAL(10, 2)
);

-- Criar a tabela de Compras
CREATE TABLE Compras (
    id INT PRIMARY KEY,
    id_operador INT,
    id_fornecedor INT,
    id_produto INT,
    quantidade INT,
    preco_unitario DECIMAL(10, 2),
    FOREIGN KEY (id_operador) REFERENCES Usuarios(id),
    FOREIGN KEY (id_fornecedor) REFERENCES Pessoas(id),
```

```
    FOREIGN KEY (id_produto) REFERENCES Produtos(id)
);

-- Criar a tabela de Vendas
CREATE TABLE Vendas (
    id INT PRIMARY KEY,
    id_operador INT,
    id_cliente INT,
    id_produto INT,
    quantidade INT,
    preco_unitario DECIMAL(10, 2),
    FOREIGN KEY (id_operador) REFERENCES Usuarios(id),
    FOREIGN KEY (id_cliente) REFERENCES Pessoas(id),
    FOREIGN KEY (id_produto) REFERENCES Produtos(id)
);
```

==========

Argumentações

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

As diferentes cardinalidades em um banco de dados relacional são aplicadas por meio das relações entre tabelas, considerando que para representar uma cardinalidade 1x1, onde uma entidade está associada a apenas outra entidade, é comum utilizar uma chave estrangeira em uma das tabelas que referencia a chave primária da outra tabela. Isso cria uma relação direta entre as duas entidades, garantindo que cada registro em uma tabela esteja associado a exatamente um registro na outra tabela. Por outro lado, para uma cardinalidade 1xN, onde uma entidade está associada a muitas outras entidades, novamente utilizamos chaves estrangeiras.

No entanto, nesse caso, a chave estrangeira é colocada na tabela que está "do lado muitos" da relação, referenciando a chave primária da tabela "do lado um", o que permite que um único registro na tabela "do lado um" esteja associado a vários registros na tabela "do lado muitos". Já para a cardinalidade NxN, onde muitas entidades estão associadas a muitas outras entidades, é necessário introduzir uma tabela intermediária,

também conhecida como tabela de associação ou tabela de junção, que contém chaves estrangeiras que referenciam as chaves primárias das duas tabelas envolvidas, estabelecendo assim a relação ‘muitos-para-muitos’ entre elas.

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

Para representar o uso de herança em bancos de dados relacionais, o tipo de relacionamento mais adequado é o de Tabela por Subtipo ou Tabela por Classe. Neste modelo, cada subtipo de uma superclasse é representado por uma tabela separada, onde as tabelas de subtipos possuem uma relação de ‘um-para-um’ com a tabela da superclasse, o que permite que cada instância de um subtipo herde os atributos e métodos da superclasse, garantindo a integridade e consistência dos dados.

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

O SSMS permite a melhoria da produtividade nas tarefas relacionadas ao gerenciamento do banco de dados de várias maneiras ao oferecer uma interface gráfica intuitiva e amigável que facilita a criação, edição e visualização de esquemas de banco de dados, tabelas, consultas SQL, entre outros objetos, e possuir recursos avançados de autocompletar e realce de sintaxe, que ajudam os desenvolvedores a escrever consultas SQL de forma mais rápida e precisa.

Outras funcionalidades úteis incluem a capacidade de executar e depurar consultas SQL, gerenciar permissões de acesso, monitorar o desempenho do servidor e realizar backups e restaurações de banco de dados de maneira eficiente. Essas características combinadas ajudam a simplificar e agilizar o processo de desenvolvimento e administração de bancos de dados usando o SQL Server.

Terceira Etapa

1) Inserir alguns produtos na base de dados:

```sql
```

```
INSERT INTO Produtos (nome, quantidade, preco)
VALUES ('Produto A', 100, 10.99),
       ('Produto B', 50, 20.50),
       ('Produto C', 200, 5.75);
```

==========

2) Criar pessoas físicas e jurídicas na base de dados:

```sql
-- Obter o próximo id de pessoa a partir da sequence
DECLARE @proxId INT;
SET @proxId = NEXT VALUE FOR PessoaSequence;

-- Incluir na tabela pessoa os dados comuns para pessoa física
INSERT INTO Pessoas (id, nome, tipo, endereco, telefone)
VALUES (@proxId, 'João da Silva', 'Física', 'Rua A, 123',
'12345678901');

-- Incluir em pessoa física o CPF, efetuando o relacionamento
com pessoa
INSERT INTO PessoasFisicas (id, cpf)
VALUES (@proxId, '123.456.789-01');
```

```sql
-- Obter o próximo id de pessoa a partir da sequence
DECLARE @proxId INT;
SET @proxId = NEXT VALUE FOR PessoaSequence;

-- Incluir na tabela pessoa os dados comuns para pessoa jurídica
INSERT INTO Pessoas (id, nome, tipo, endereco, telefone)
VALUES (@proxId, 'Empresa XYZ', 'Jurídica', 'Av. Principal,
456', '98765432109876');

-- Incluir em pessoa jurídica o CNPJ, relacionando com pessoa
INSERT INTO PessoasJuridicas (id, cnpj)
```

```
VALUES (@proxId, '98.765.432/1098-76');
```

==========

3) Criar algumas movimentações na base de dados:

```sql
-- Exemplo de movimentação de entrada (Compra)
INSERT INTO Compras (id_operador, id_fornecedor, id_produto, quantidade, preco_unitario)
VALUES (1, 2, 1, 50, 8.99); -- Operador 1 (op1) faz uma compra de 50 unidades do Produto A da Empresa XYZ

-- Exemplo de movimentação de saída (Venda)
INSERT INTO Vendas (id_operador, id_cliente, id_produto, quantidade, preco_unitario)
VALUES (2, 3, 2, 20, 25.99); -- Operador 2 (op2) realiza uma venda de 20 unidades do Produto B para João da Silva
```

==========

4) Efetuar as consultas solicitadas sobre os dados inseridos:

==========

```sql
-- 6) Dados completos de pessoas físicas
SELECT * FROM Pessoas WHERE tipo = 'Física';

-- 7) Dados completos de pessoas jurídicas
SELECT * FROM Pessoas WHERE tipo = 'Jurídica';

-- 8) Movimentações de entrada
SELECT p.nome AS Produto, pj.nome AS Fornecedor, c.quantidade, c.preco_unitario, (c.quantidade * c.preco_unitario) AS Total
```

```
FROM Compras c
JOIN Produtos p ON c.id_produto = p.id
JOIN Pessoas pj ON c.id_fornecedor = pj.id;

-- 9) Movimentações de saída
SELECT p.nome AS Produto, pf.nome AS Comprador, v.quantidade,
v.preco_unitario, (v.quantidade * v.preco_unitario) AS Total
FROM Vendas v
JOIN Produtos p ON v.id_produto = p.id
JOIN Pessoas pf ON v.id_cliente = pf.id;

-- 10) Valor total das entradas agrupadas por produto
SELECT p.nome AS Produto, SUM(c.quantidade * c.preco_unitario)
AS TotalEntradas
FROM Compras c
JOIN Produtos p ON c.id_produto = p.id
GROUP BY p.nome;

-- 11) Valor total das saídas agrupadas por produto
SELECT p.nome AS Produto, SUM(v.quantidade * v.preco_unitario)
AS TotalSaidas
FROM Vendas v
JOIN Produtos p ON v.id_produto = p.id
GROUP BY p.nome;

-- 12) Operadores que não efetuaram movimentações de entrada
(compra)
SELECT u.nome AS Operador
FROM Usuarios u
LEFT JOIN Compras c ON u.id = c.id_operador
WHERE c.id_operador IS NULL;

-- 13) Valor total de entrada, agrupado por operador
SELECT u.nome AS Operador, SUM(c.quantidade * c.preco_unitario)
AS TotalEntradas
FROM Compras c
JOIN Usuarios u ON c.id_operador = u.id
```

```
GROUP BY u.nome;

-- 14) Valor total de saída, agrupado por operador
SELECT u.nome AS Operador, SUM(v.quantidade * v.preco_unitario)
AS TotalSaidas
FROM Vendas v
JOIN Usuarios u ON v.id_operador = u.id
GROUP BY u.nome;

-- 15) Valor médio de venda por produto, utilizando média
ponderada
SELECT p.nome AS Produto, SUM(v.quantidade * v.preco_unitario) /
SUM(v.quantidade) AS MediaVenda
FROM Vendas v
JOIN Produtos p ON v.id_produto = p.id
GROUP BY p.nome;
```

==========

Argumentações

Quais as diferenças no uso de sequence e identity?

No SQL Server, uma sequence é um objeto que gera uma sequência de números exclusivos em uma ordem específica. Ao usá-lo, é possível controlar manualmente a geração de valores para colunas, fornecendo flexibilidade na atribuição de identificadores exclusivos às linhas de uma tabela. Quanto ao Identity, se trata de uma propriedade de coluna que automaticamente gera valores numéricos únicos para cada linha inserida em uma tabela.

Ao definir uma coluna como identity, o SQL Server automaticamente atribui e incrementa os valores para essa coluna conforme novas linhas são adicionadas à tabela. Portanto, a diferença fundamental entre sequence e identity é que o primeiro oferece mais controle sobre a geração de valores únicos, enquanto o segundo é uma opção mais simplificada e automática para atribuir identificadores únicos às linhas de uma tabela.

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

As chaves estrangeiras são fundamentais para garantir a integridade referencial e a consistência dos dados em um banco de dados relacional, pois ao definir uma chave estrangeira em uma tabela, estamos estabelecendo uma relação entre duas tabelas, onde a tabela referenciada contém a chave primária que corresponde à chave estrangeira na tabela que a referencia. Isso significa que cada valor na coluna referenciada deve refletir na coluna referenciadora, garantindo que não haja referências órfãs ou inexistentes. Também é possível considerar que as chaves estrangeiras ajudam a manter a integridade dos dados durante operações de inserção, atualização e exclusão, evitando que ações indesejadas causem inconsistências nos dados.

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

Alguns dos operadores da álgebra relacional incluem união, interseção, diferença, projeção, seleção, junção e renomeação, sendo usados para realizar operações sobre conjuntos de dados relacionais. Quanto ao cálculo relacional, os operadores que podem ser empregados são seleção, projeção e junção. No entanto, é necessário considerar que esses operadores são expressos de forma diferente do que na álgebra relacional, ao ser aplicada a lógica de predicados para descrever as operações efetuadas sobre os dados.

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

O agrupamento em consultas SQL é realizado utilizando a cláusula `GROUP BY`, que agrupa as linhas de uma consulta em conjuntos baseados nos valores de uma ou mais colunas. Quando usamos `GROUP BY`, precisamos também usar funções de agregação, como `SUM`, `AVG`, `COUNT`, `MAX` e `MIN`, para calcular valores agregados para cada grupo de linhas.

Um requisito obrigatório ao utilizar a cláusula `GROUP BY` é que todas as colunas que não estão sendo usadas em funções de agregação devem estar incluídas na cláusula `GROUP BY`, garantindo que o SQL saiba como agrupar as linhas corretamente e como aplicar as funções de agregação aos grupos de linhas correspondentes, considerando que se uma coluna estiver presente na lista de seleção,

mas não estiver presente na cláusula `GROUP BY` ou em uma função de agregação, um erro será gerado.