UNIVERSIDADE ESTÁCIO DE SÁ

DESENVOLVIMENTO FULL STACK

MUNDO 3 – NÍVEL 2

RPG0015 - VAMOS MANTER AS INFORMAÇÕES!

ALUNO: ERB LUCIANO MARIZ DE BARROS FILHO

Objetivos da prática:

Por meio da criação de um banco de dados para um projeto fictício de uma loja, tem como objetivo principal desta prática, aplicar os conceitos teóricos de modelagem de banco de dados, utilizando o Microsoft SQL Server.

Criando um banco de dados para um projeto fictício de uma loja, Querendo alcançar o entendimento sobre a estruturação de dados em ambientes relacionais.

Tendo como objetivos: Realizar a criação do banco de dados “Loja” contemplando tabelas para armazenar informações sobre pessoas, usuários, produtos e movimentações; Inserir dados fictícios nas tabelas criadas, abrangendo aspectos como pessoas físicas e jurídicas, usuários, produtos, e movimentações de entrada e saída.

Resultados:

Consegui como resultados da prática de modelagem e implementação do banco de dados para o projeto da loja. Apresentando a modelagem do banco de dados utilizando a ferramenta DB Designer Online, criação da estrutura do banco no SQL Server Management Studio, alimentação do banco com dados totalmente fictícios e execução de consultas SQL.

O início deste projeto teve como objetivo à modelagem conceitual do banco de dados, crucial para definir a estrutura que sustentaria o sistema da loja fictícia.

A entidade Pessoa, por exemplo, engloba informações como nome, logradouro, cidade, estado, telefone e email. A relação entre Pessoa e Usuario foi estabelecida para incorporar dados de login e senha às informações das pessoas.

Uma abordagem de herança foi aplicada para diferenciar entre Pessoas Físicas e Jurídicas. As entidades PessoaFisica e PessoaJuridica herdam características da entidade base Pessoa, enquanto suas particularidades, como CPF e CNPJ, são representadas em tabelas específicas.

A modelagem visual permitiu a definição clara de chaves primárias, estrangeiras e índices, garantindo a integridade referencial do banco de dados.

A relação entre as entidades, estabelece a base para a construção de consultas complexas e a recuperação eficiente de dados durante a fase de implementação.

Essa representação gráfica foi fundamental para validar conceitualmente a estrutura do banco antes da implementação no SQL Server Management Studio.

A visualização prévia proporcionou uma compreensão abrangente da interconexão entre as entidades e seus relacionamentos, tornando a transição da modelagem para a implementação mais fluida e consistente.

A fase de criação e estruturação do banco de dados foi realizada para definir a arquitetura necessária para suportar as operações da loja fictícia. O processo abrange desde a criação do banco no SQL Server até a definição das tabelas, chaves primárias e estrangeiras, estabelecendo relações coesas entre os dados.

Código:

1 - Criação de usuário e banco de dados.

```
USE master;
CREATE LOGIN loja WITH PASSWORD = 'loja', CHECK_POLICY = OFF;
CREATE USER loja FOR LOGIN loja;
CREATE DATABASE Loja;
USE Loja;
```

2 - Criação da estrutura de tabelas, campos e chaves.

```
CREATE TABLE [Pessoa] (
idPessoa int NOT NULL UNIQUE,
nome varchar(50) NOT NULL,
logradouro varchar(50) NOT NULL,
cidade varchar(50) NOT NULL,
estado char(2) NOT NULL,
telefone varchar(11) NOT NULL,
email varchar(50) NOT NULL,
CONSTRAINT [PK_PESSOA] PRIMARY KEY CLUSTERED
(
[idPessoa] ASC
) WITH (IGNORE_DUP_KEY = OFF)
)
CREATE TABLE [Usuario] (
idUsuario int NOT NULL UNIQUE,
login varchar(25) NOT NULL UNIQUE,
senha varchar(16) NOT NULL,
CONSTRAINT [PK_USUARIO] PRIMARY KEY CLUSTERED
(
[idUsuario] ASC
) WITH (IGNORE_DUP_KEY = OFF)
)
CREATE TABLE [PessoaFisica] (
idPessoaFisica int NOT NULL UNIQUE,
cpf varchar(11) NOT NULL UNIQUE,
CONSTRAINT [PK_PESSOAFISICA] PRIMARY KEY CLUSTERED
(
[idPessoaFisica] ASC
) WITH (IGNORE_DUP_KEY = OFF)
)
CREATE TABLE [PessoaJuridica] (
idPessoaJuridica int NOT NULL UNIQUE,
cnpj varchar(14) NOT NULL UNIQUE,
CONSTRAINT [PK_PESSOAJURIDICA] PRIMARY KEY CLUSTERED
```

```
(
[idPessoaJuridica] ASC
) WITH (IGNORE_DUP_KEY = OFF)
)
CREATE TABLE [Produto] (
idProduto int NOT NULL UNIQUE,
nome varchar(100) NOT NULL,
quantidade int NOT NULL,
precoVenda decimal (10, 2) NOT NULL,
CONSTRAINT [PK_PRODUTO] PRIMARY KEY CLUSTERED
(
[idProduto] ASC
) WITH (IGNORE_DUP_KEY = OFF)
)
CREATE TABLE [Movimento] (
idMovimento int NOT NULL UNIQUE,
idProduto int NOT NULL,
idPessoa int NOT NULL,
idUsuario int NOT NULL,
tipo char(1) NOT NULL,
quantidade int NOT NULL,
precoUnitario decimal(10, 2) NOT NULL,
CONSTRAINT [PK_MOVIMENTO] PRIMARY KEY CLUSTERED
(
[idMovimento] ASC
) WITH (IGNORE_DUP_KEY = OFF)
)
```

3 - Criação de relações/dependências referenciais e integrativas.

```
ALTER TABLE [PessoaFisica] WITH CHECK ADD CONSTRAINT [PessoaFisica_fk0]
FOREIGN
KEY ([idPessoaFisica]) REFERENCES [Pessoa]([idPessoa])
ON UPDATE CASCADE
ALTER TABLE [PessoaFisica] CHECK CONSTRAINT [PessoaFisica_fk0]
ALTER TABLE [PessoaJuridica] WITH CHECK ADD CONSTRAINT [PessoaJuridica_fk0]
FOREIGN KEY ([idPessoaJuridica]) REFERENCES [Pessoa]([idPessoa])
ON UPDATE CASCADE
ALTER TABLE [PessoaJuridica] CHECK CONSTRAINT [PessoaJuridica_fk0]
ALTER TABLE [Movimento] WITH CHECK ADD CONSTRAINT [Movimento_fk0] FOREIGN KEY
([idProduto]) REFERENCES [Produto]([idProduto])
ON UPDATE CASCADE
ALTER TABLE [Movimento] CHECK CONSTRAINT [Movimento_fk0]
ALTER TABLE [Movimento] WITH CHECK ADD CONSTRAINT [Movimento_fk1] FOREIGN KEY
([idPessoa]) REFERENCES [Pessoa]([idPessoa])
ON UPDATE CASCADE
ALTER TABLE [Movimento] CHECK CONSTRAINT [Movimento_fk1]
ALTER TABLE [Movimento] WITH CHECK ADD CONSTRAINT [Movimento_fk2] FOREIGN KEY
([idUsuario]) REFERENCES [Usuario]([idUsuario])
ON UPDATE CASCADE
ALTER TABLE [Movimento] CHECK CONSTRAINT [Movimento_fk2]
```

4 - Inserção de dados nas tabelas.

```
INSERT INTO [Pessoa] (idPessoa, nome, logradouro, cidade, estado, telefone,
email)
VALUES
```

```
    (1, 'Audenice Barros', 'Avenida Visconde de Albuquerque 53', 'Recife',
'PE', '81988885555', 'aobarros@email.com'),
    (2, 'Pedro Silva', 'Avenida Central 456', 'João Pessoa', 'PB',
'21987654321', 'pedro@email.com'),
    (3, 'Camila Oliveira', 'Travessa Principal 789', 'Belo Horizonte', 'MG',
'31987654321', 'camila@email.com'),
    (4, 'Paulo Pereira', 'Rua Principal 321', 'Rio de Janeiro', 'RJ',
'51987654321', 'paulo@email.com');

INSERT INTO [Usuario] (idUsuario, login, senha)
VALUES
(1, 'funcionario1', 'senha123'),
(2, 'funcionario2', 'senha456'),
(3, 'gerente1', 'senha789');
INSERT INTO [PessoaFisica] (idPessoaFisica, cpf)
VALUES
(1, '11122233344'),
(2, '55566677788');
INSERT INTO [PessoaJuridica] (idPessoaJuridica, cnpj)
VALUES
(1, '12345678901234'),
(2, '98765432109876');
INSERT INTO [Produto] (idProduto, nome, quantidade, precoVenda)
VALUES
(1, 'Maçã', 200, 2.50),
(2, 'Abacaxi', 150, 3.00),
(3, 'Pera', 120, 2.75),
(4, 'Uva', 180, 4.50);
INSERT INTO [Movimento] (idMovimento, idProduto, idPessoa, idUsuario, tipo,
quantidade, precoUnitario)
VALUES
(1, 1, 1, 1, 'E', 20, 2.50),
(2, 2, 2, 2, 'E', 15, 3.00),
(3, 3, 3, 3, 'S', 10, 2.75),
(4, 4, 4, 1, 'E', 25, 4.50),
(5, 1, 2, 2, 'S', 5, 3.00);
```

5 - Consultas.

```
SELECT Pessoa.idPessoa, Pessoa.nome, Pessoa.logradouro, Pessoa.cidade,
Pessoa.estado, Pessoa.telefone, Pessoa.email, PF.cpf
FROM Pessoa
JOIN PessoaFisica PF ON Pessoa.idPessoa = PF.idPessoaFisica;
SELECT Pessoa.idPessoa, Pessoa.nome, Pessoa.logradouro, Pessoa.cidade,
Pessoa.estado, Pessoa.telefone, Pessoa.email, PJ.cnpj
FROM Pessoa
JOIN PessoaJuridica PJ ON Pessoa.idPessoa = PJ.idPessoaJuridica;
SELECT
P.nome AS Fornecedor,
Prod.nome AS Produto,
Mov.quantidade,
Mov.precoUnitario AS PrecoUnitario,
Mov.quantidade * Mov.precoUnitario AS ValorTotal
FROM Movimento Mov
JOIN Produto Prod ON Mov.idProduto = Prod.idProduto
JOIN Pessoa P ON Mov.idPessoa = P.idPessoa
```

```
WHERE Mov.tipo = 'E';
SELECT
P.nome AS Comprador,
Prod.nome AS Produto,
Mov.quantidade,
Mov.precoUnitario AS PrecoUnitario,
Mov.quantidade * Mov.precoUnitario AS ValorTotal
FROM Movimento Mov
JOIN Produto Prod ON Mov.idProduto = Prod.idProduto
JOIN Pessoa P ON Mov.idPessoa = P.idPessoa
WHERE Mov.tipo = 'S';
SELECT Mov.idProduto, SUM(Mov.quantidade * Mov.precoUnitario) AS
TotalEntradas
FROM Movimento Mov
WHERE Mov.tipo = 'E'
GROUP BY Mov.idProduto;
SELECT Mov.idProduto, SUM(Mov.quantidade * Mov.precoUnitario) AS TotalSaidas
FROM Movimento Mov
WHERE Mov.tipo = 'S'
GROUP BY Mov.idProduto;
SELECT DISTINCT U.*
FROM Usuario U
LEFT JOIN Movimento M ON U.idUsuario = M.idUsuario AND M.tipo = 'E'
WHERE M.idUsuario IS NULL;
SELECT M.idUsuario, SUM(M.quantidade * M.precoUnitario) AS TotalEntradas
FROM Movimento M
WHERE M.tipo = 'E'
GROUP BY M.idUsuario;
SELECT M.idUsuario, SUM(M.quantidade * M.precoUnitario) AS TotalSaidas
FROM Movimento M
WHERE M.tipo = 'S'
GROUP BY M.idUsuario;
SELECT
Prod.nome AS Produto,
Mov.idProduto,
SUM(Mov.quantidade * Mov.precoUnitario) / SUM(Mov.quantidade) AS
MediaPonderada
FROM Movimento Mov
JOIN Produto Prod ON Mov.idProduto = Prod.idProduto
WHERE Mov.tipo = 'S'
GROUP BY Mov.idProduto, Prod.nome;
```

Análise e Conclusão:

1- **Diferenças entre sequence e identity**

Identity e Sequence são usados para gerar um número automático. A principal diferença entre eles é que o identity depende da tabela e o sequence é independente da tabela.

2- **Importância das chaves estrangerias para a consistência do banco**

A chave estrangeira é a representação de um relacionamento entre tabelas, por isso é muito importante, haja vista que a utilização da chave

estrangeira possibilita a implementação da integridade de dados diretamente no banco de dados, conhecida como integridade referencial.

3- Operadores do SQL pertencem à álgebra relacional e os que são definidos no cálculo relacional

Na verdade, tanto a álgebra relacional quanto o cálculo relacional são linguagens de consulta para bancos de dados relacionais, mas elas têm abordagens diferentes para expressar consultas.

Na álgebra relacional, os operadores principais incluem:

Seleção (σ): Usado para selecionar tuplas de uma relação que atendem a uma condição específica.

Projeção (π): Usado para selecionar colunas específicas de uma relação.

União (∪): Combina duas relações para produzir uma nova relação que contém todas as tuplas de ambas, eliminando duplicatas.

Interseção (∩): Retorna uma relação que contém apenas as tuplas que estão presentes em ambas as relações de entrada.

Diferença (-): Retorna uma relação que contém todas as tuplas da primeira relação que não estão presentes na segunda relação.

Produto Cartesiano (×): Combina todas as tuplas de uma relação com todas as tuplas de outra relação, produzindo uma nova relação com todas as combinações possíveis de tuplas.

Junção (⋈): Combina duas relações com base em uma condição de junção específica.

Por outro lado, no cálculo relacional, os operadores são expressos em termos de lógica de predicados e incluem:

Projeção: Assim como na álgebra, permite a seleção de colunas específicas.

Seleção: Também semelhante à álgebra, para selecionar tuplas que atendam a uma condição específica.

Join: No cálculo relacional, a junção é geralmente expressa usando predicados de igualdade entre atributos de duas relações.

Portanto, a seleção e a projeção são comuns tanto à álgebra quanto ao cálculo relacional. Enquanto a álgebra relacional possui operadores específicos para união, interseção, diferença e produto cartesiano, o cálculo relacional trata essas operações de maneira mais indireta, geralmente usando predicados de igualdade e lógica booleana.

4- Agrupamento em consultas

O agrupamento em consultas SQL é realizado usando a cláusula GROUP BY, seguida por uma lista de colunas pelas quais você deseja agrupar os dados. Quando uma consulta inclui uma cláusula GROUP BY, as linhas com valores idênticos nas colunas especificadas são agrupadas em conjuntos.

Para agrupar os dados por produto e calcular a soma da quantidade vendida de cada produto (Por exemplo), você pode usar a seguinte consulta:

```
SELECT produto, SUM(quantidade) AS total_quantidade
FROM pedidos
GROUP BY produto;
```

É importante notar que ao usar a cláusula GROUP BY, todas as colunas selecionadas na consulta que não estão envolvidas em funções de agregação (como SUM, COUNT, AVG, etc.) devem ser incluídas na cláusula GROUP BY. Isso garante que cada linha no resultado esteja associada a um grupo definido pelas colunas especificadas. Se uma coluna não estiver incluída na cláusula GROUP BY e não estiver envolvida em uma função de agregação, ela causará um erro de sintaxe.