RPG0015 - Vamos manter as informações!

Paulo Wuéliton Horacio Fernandes (202302025919)

507 Polo Tucuruvi – São Paulo/SP
RPG0015 - Vamos manter as informações! – 2024.1 FLEX - 3º Semestre Letivo

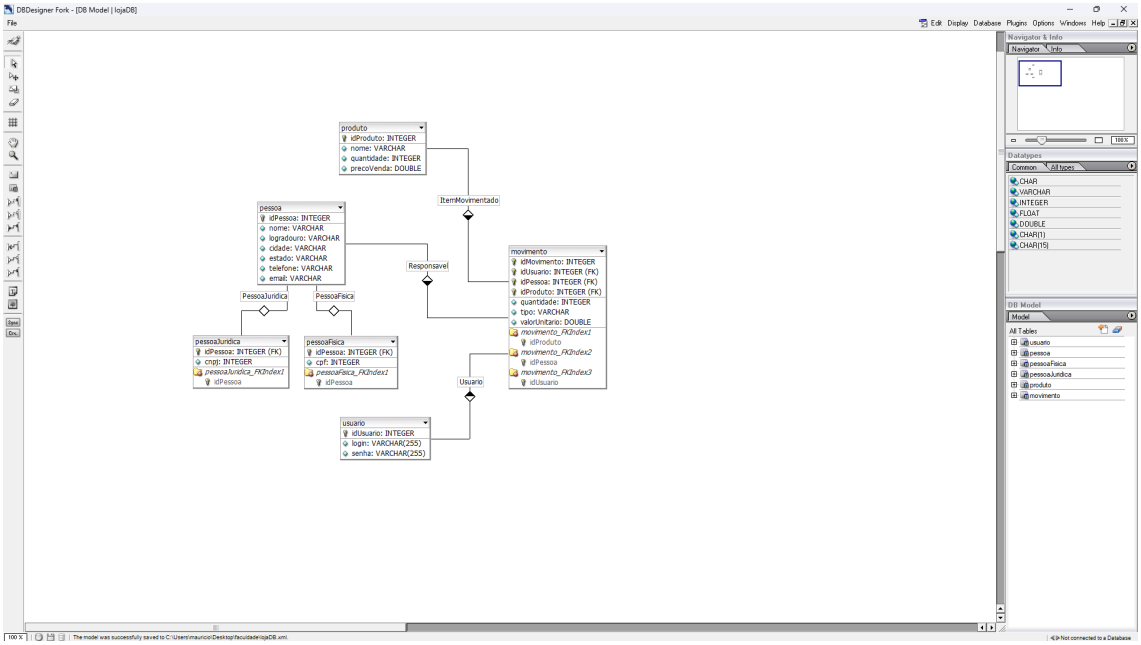
GIT: https://github.com/wueliton/rpg0015-vamos-manter-as-informacoes

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 | Criando o Banco de Dados

Modelo de Dados



Modelagem dos Dados

Tabela de Usuários

```
CREATE TABLE usuario(
    idUsuario INT IDENTITY(1,1) PRIMARY KEY,
    login VARCHAR(100),
    senha VARCHAR(30)
);
```

Sequence para ID de Pessoas, Tabela de Pessoas, Pessoas Físicas e Jurídicas

```sql
CREATE SEQUENCE idPessoa START WITH 1 INCREMENT BY 1;

CREATE TABLE pessoa(
    idPessoa INT PRIMARY KEY DEFAULT NEXT VALUE FOR idPessoa,
    nome VARCHAR(255),
    logradouro VARCHAR(255),
    cidade VARCHAR(255),
    estado VARCHAR(2),
    telefone VARCHAR(255),
    email VARCHAR(255)
);

CREATE TABLE pessoaFisica(
    idPessoa INT PRIMARY KEY NOT NULL,
    cpf VARCHAR(11) NOT NULL,
    FOREIGN KEY (idPessoa) REFERENCES pessoa(idPessoa)
);

CREATE TABLE pessoaJuridica(
    idPessoa INT PRIMARY KEY NOT NULL,
    cnpj VARCHAR(14) NOT NULL,
    FOREIGN KEY (idPessoa) REFERENCES pessoa(idPessoa)
);
```

Tabela de Produtos

```sql
CREATE TABLE produto(
    idProduto INT PRIMARY KEY NOT NULL,
    nome VARCHAR(255),
    quantidade INT,
    precoVenda DECIMAL(10, 2)
);
```

Tabela de Movimento

```
CREATE TABLE movimento(
    idMovimento INT IDENTITY(1,1) PRIMARY KEY,
    idUsuario INT NOT NULL,
    idPessoa INT NOT NULL,
    idProduto INT NOT NULL,
    quantidade INT,
    tipo VARCHAR(1),
    valorUnitario DECIMAL(10,2),
    FOREIGN KEY (idUsuario) REFERENCES usuario(idUsuario),
    FOREIGN KEY (idPessoa) REFERENCES pessoa(idPessoa),
    FOREIGN KEY (idProduto) REFERENCES produto(idProduto),
    CONSTRAINT TIPO_MOVIMENTO_INVALIDO CHECK (tipo IN ('E', 'S'))
);
```

Análise e Conclusão

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

O relacionamento entre tabelas possui sempre dois tipos de chaves, as primárias (PK) e as estrangeiras (FK) que criam o vínculo entre as tabelas, garantindo a integridade dos dados.

Na cardinalidade 1X1 o relacionamento é realizado por uma tabela que possui chave primária (PK) e outra tabela que possui chave estrangeira (FK) definida como única e relacionada a chave primária da primeira tabela, isso faz com que um identificador da primeira tabela só esteja relacionado a uma linha da segunda tabela, pois não poderão haver duas chaves estrangeiras com o mesmo valor.

Na cardinalidade 1XN o relacionamento é construído por uma tabela com uma chave primária (FK) e outra tabela que possui chave estrangeira (FK) relacionada com a chave primária da primeira tabela,

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

Há diversas maneiras de representar a herança, a depender da estrutura

desejada, mas, o mais comum é utilizar um relacionamento 1x1 ou 1x0.

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

O Gerenciador facilita a conexão com o banco de dados, a visualização da estrutura e execução de diversas operações que otimizam o tempo de trabalho, como criação de consultas para exibição de todos os dados da tabela, editor de consultas SQL com autocomplete das informações digitadas, correção da sintaxe SQL , exportação das consultas digitadas para um arquivo externo, além de permitir gerenciar também os bancos de dados em nuvem.

d) **Como a herança é refletida no banco de dados, quando lidamos com um modelo estritamente relacional?**

Na modelagem de herança em um banco de dados relacional, a superclasse e a classe são representadas por duas tabelas distintas, relacionadas por meio de uma chave primária e uma chave estrangeira, estabelecendo um relacionamento 1 para 1. Durante a inserção de dados, a tabela que representa a classe depende de uma chave primária existente na tabela que representa a superclasse. Esse tipo de relacionamento simplifica a recuperação de dados e se assemelha muito à estrutura de herança encontrada em Java.

2º Procedimento | Alimentando a Base

Inserção de Usuários

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

Inserção de Produtos

```
INSERT INTO produto (idProduto, nome, quantidade, precoVenda) VALUES (1,
'Banana', 100, 5.00);
INSERT INTO produto (idProduto, nome, quantidade, precoVenda) VALUES (3,
'Laranja', 500, 2.00);
INSERT INTO produto (idProduto, nome, quantidade, precoVenda) VALUES (4,
'Manga', 100, 4.00);
```

Inserção de Pessoas Físicas e Jurídicas e Movimentações

```
DECLARE @PFId INT;
SET @PFId = NEXT VALUE FOR idPessoa;

INSERT INTO pessoa (idPessoa, nome, logradouro, cidade, estado, telefone,
email)
VALUES (@PFId, 'Joao', 'Rua 12, casa 3, Quitanda', 'Riacho do Sul', 'PA',
'1111-1111', 'joao@riacho.com');

INSERT INTO pessoaFisica (idPessoa, cpf) VALUES (@PFId, '11111111111');

SELECT * FROM pessoa AS p INNER JOIN pessoaFisica AS pf ON p.idPessoa =
pf.idPessoa;

-- CRIAÇÃO DE PESSOA JURÍDICA
DECLARE @PJId INT;
SET @PJId = NEXT VALUE FOR idPessoa;

INSERT INTO pessoa (idPessoa, nome, logradouro, cidade, estado, telefone,
email)
VALUES (@PJId, 'JJC', 'Rua 11. Centro', 'Riacho do Norte', 'PA', '1212-
1212', 'jjc@riacho.com');

INSERT INTO pessoaJuridica (idPessoa, cnpj) VALUES (@PJId,
'22222222222222');

SELECT * FROM pessoa AS p INNER JOIN pessoaJuridica AS pj ON p.idPessoa =
pj.idPessoa;

-- CRIAÇÃO DA MOVIMENTAÇÃO
INSERT INTO movimento (idUsuario, idPessoa, idProduto, quantidade, tipo,
valorUnitario) VALUES
    (1, @PFId, 1, 20, 'S', 4.00),
    (1, @PFId, 3, 15, 'S', 2.00),
    (2, @PFId, 3, 10, 'S', 3.00),
    (1, @PJId, 3, 15, 'E', 5.00),
    (1, @PJId, 4, 20, 'E', 4.00);
```

Consultas solicitadas

a. Dados completos de pessoas físicas

```
SELECT * FROM pessoa AS p INNER JOIN pessoaFisica AS pf ON p.idPessoa =
pf.idPessoa;
```

Results Messages

	idPessoa	nome	logradouro	cidade	estado	telefone	email	idPessoa	cpf
1	1	Joao	Rua 12, casa 3, Quitanda	Riacho do Sul	PA	1111-1111	joao@riacho.com	1	11111111111

b. Dados completos de pessoas jurídicas

```
SELECT * FROM pessoa AS p INNER JOIN pessoaJuridica AS pj ON p.idPessoa =
pj.idPessoa;
```

Results Messages

	idPessoa	nome	logradouro	cidade	estado	telefone	email	idPessoa	cnpj
1	2	JJC	Rua 11. Centro	Riacho do Norte	PA	1212-1212	jjc@riacho.com	2	22222222222222

c. Movimentações de entrada, com produto, fornecedor, quantidade, preço unitário e valor total.

```
SELECT m.idMovimento,
    m.tipo,
    produto.nome AS Produto,
    p.nome as Fornecedor,
    m.quantidade AS Quantidade,
    m.valorUnitario AS 'Preço Unitário',
    m.quantidade * m.valorUnitario AS 'valorTotal'
FROM movimento AS m
INNER JOIN pessoa AS p ON m.idPessoa = p.idPessoa
INNER JOIN produto ON m.idProduto = produto.idProduto
WHERE m.tipo = 'E';
```

Results Messages

	idMovimento	tipo	Produto	Fornecedor	Quantidade	Preço Unitário	valorTotal
1	7	E	Laranja	JJC	15	5.00	75.00
2	8	E	Manga	JJC	20	4.00	80.00

d. Movimentações de saída, com produto, comprador, quantidade,preço unitário e valor total.

```sql
SELECT m.idMovimento,
    m.tipo,
    produto.nome AS Produto,
    p.nome AS Comprador,
    m.quantidade AS Quantidade,
    m.valorUnitario AS 'Preço Unitário',
    m.quantidade * m.valorUnitario AS 'Valor Total'
FROM movimento AS m
INNER JOIN produto ON m.idProduto = produto.idProduto
INNER JOIN pessoa AS p ON m.idPessoa = p.idPessoa
WHERE m.tipo = 'S';
```

Results Messages

	idMovimento	tipo	Produto	Comprador	Quantidade	Preço Unitário	Valor Total
1	1	S	Banana	Joao	20	4.00	80.00
2	4	S	Laranja	Joao	15	2.00	30.00
3	5	S	Laranja	Joao	10	3.00	30.00

e. Valor total das entradas agrupadas por produto.

```sql
SELECT produto.idProduto, produto.nome AS produto,
    SUM(m.quantidade * m.valorUnitario) AS total
FROM movimento AS m
INNER JOIN produto ON m.idProduto = produto.idProduto
WHERE m.tipo = 'E'
GROUP BY produto.nome, produto.idProduto;
```

Results Messages

	idProduto	produto	total
1	3	Laranja	75.00
2	4	Manga	80.00

f. Valor total das saídas agrupadas por produto.

```sql
SELECT produto.idProduto, produto.nome AS produto,
    SUM(m.quantidade * m.valorUnitario) AS total
FROM movimento AS m
INNER JOIN produto ON m.idProduto = produto.idProduto
WHERE m.tipo = 'S'
GROUP BY produto.nome, produto.idProduto;
```

Results Messages

	idProduto	produto	total
1	1	Banana	80.00
2	3	Laranja	60.00

g. Operadores que não efetuaram movimentações de entrada (compra).

```
SELECT u.* FROM usuario AS u
LEFT JOIN movimento AS m ON m.idUsuario = u.idUsuario AND m.tipo = 'E'
WHERE m.idMovimento IS NULL;
```

Results Messages

	idUsuario	login	senha
1	2	op2	op2

h. Valor total de entrada, agrupado por operador.

```
SELECT u.idUsuario, u.login AS operador,
    SUM(m.quantidade * m.valorUnitario) AS total
FROM movimento AS m
INNER JOIN usuario as u ON m.idUsuario = u.idUsuario
WHERE m.tipo = 'E'
GROUP BY u.login, u.idUsuario;
```

Results Messages

	idUsuario	operador	total
1	1	op1	155.00

i. Valor total de saída, agrupado por operador.

```
SELECT u.idUsuario, u.login AS operador,
    SUM(m.quantidade * m.valorUnitario) AS total
FROM movimento AS m
INNER JOIN usuario as u ON m.idUsuario = u.idUsuario
WHERE m.tipo = 'S'
GROUP BY u.login, u.idUsuario;
```

Results Messages

	idUsuario	operador	total
1	1	op1	110.00
2	2	op2	30.00

j. Valor médio de venda por produto, utilizando média ponderada.

```sql
SELECT p.idProduto,
       p.nome AS produto,
       SUM(m.quantidade * m.valorUnitario) / SUM(m.quantidade) AS
valorMedio
FROM movimento m
INNER JOIN produto p ON m.idProduto = p.idProduto
WHERE m.tipo = 'S'
GROUP BY p.idProduto, p.nome;
```

Results Messages

	idProduto	produto	valorMedio
1	1	Banana	4.000000
2	3	Laranja	2.400000

Análise e Conclusão

a) **Quais as diferenças no uso de sequence e identity?**
Sequence é uma sequencia independente de qualquer tabela, que pode ser utilizada para inserir registros em qualquer tabela do banco de dados, permitindo que o próximo número da sequencia seja recuperado em qualquer operação de inserção no banco de dados, já o Identity está vinculado a uma coluna de uma tabela que irá gerar a sequencia automaticamente a cada inserção de linhas da tabela vinculada.

b) **Qual a importância das chaves estrangerias para a consistência do banco?**
As chaves estrangeiras garantem a consistência e integridade dos dados vinculados, pois obriga que para que o relacionamento seja criado, editado ou excluído, o valor na coluna relacionada exista e seja do tipo definido, prevenindo a criação de chaves estrangeiras sem valores na coluna de relacionamento.

c) **Quais operadores do SQL pertencem à álgebra relacional e quais são definidos no cálculo relacional?**
Operadores que pertencem à álgebra relacional:
a. Seleção: Seleciona linhas que satisfazem uma determinada condição indicada pelo WHERE.

b. Projeção: Gera relações excluindo alguns atributo, isso é feito através do SELECT.

c. Produto Cartesiano: Permite a combinação de informações de duas relações através do JOIN.

d. União: Cria uma relação a partir de duas outras relações, estabelecendo a união de todas as linhas da nova relação, realizada pelo operador UNION.

e. Diferença: Cria uma relação que contém todas as linhas que pertencem a primeira relação e não pertencem a segunda, realizada pelo operador EXCEPT.

f. Renomear: Permite renomear o esquema de uma relação utilizando o operador AS.

Operadores definidos no cálculo relacional:

a. Operadores lógicos: Ajudam a combinar condições usadas para filtrar os registros obtidos, são eles AND, OR e NOT.

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

A função GROUP BY é usada para agrupar linhas de dados com valores semelhantes em uma ou mais colunas. Requer a presença de funções de agregação (como SUM, MAX, MIN, AVG, COUNT) aplicadas a uma ou mais colunas para gerar um resultado agrupado. Além disso, é necessário listar as colunas utilizadas nas funções de agregação na cláusula GROUP BY.