



Estácio

UNIVERSIDADE ESTÁCIO DE SÁ DE RIBEIRÃO PRETO

POLO PARQUE ANDORINHAS

DESENVOLVIMENTO FULL STACK

2023.3 FULL STACK ALUNO

NIVEL 2: Vamos Manter as Informações?

ALESSANDRO SENDI SHIGEMATSU

Título da Prática

RPG0015 - Vamos manter as informações!

Objetivos 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

Todos os códigos solicitados neste roteiro de aula

```
CREATE TABLE Produto (  
    idProduto INTEGER NOT NULL IDENTITY(1,1) PRIMARY KEY,  
    nome VARCHAR(255) NULL ,  
    quantidade INTEGER NULL ,  
    precoVenda NUMERIC NULL ,  
);
```

```
CREATE TABLE Pessoa (  
    idPessoa INTEGER NOT NULL PRIMARY KEY,  
    nome VARCHAR(255) NULL ,  
    logradouro VARCHAR(255) NULL ,  
    cidade VARCHAR(255) NULL ,  
    estado CHAR(2) NULL ,  
    telefone VARCHAR(255) NULL ,  
    email VARCHAR(255) NULL ,  
);
```

```
CREATE TABLE Usuario (  
    idUsuario INTEGER NOT NULL IDENTITY(1,1) PRIMARY KEY,  
    login VARCHAR(255) NULL ,  
    senha VARCHAR(255) NULL ,  
);
```

```
CREATE TABLE Movimento (  
    idMovimento INTEGER IDENTITY(1,1) PRIMARY KEY,  
    Usuario_idUsuario INTEGER FOREIGN KEY REFERENCES Usuario(idUsuario) ,  
    Pessoa_idPessoa INTEGER FOREIGN KEY REFERENCES Pessoa(idPessoa) ,
```

```

    Produto_idProduto INTEGER FOREIGN KEY REFERENCES Produto(idProduto) ,
    quantidade INTEGER NULL ,
    tipo VARCHAR(2) NULL ,
    valorUnitario NUMERIC NULL ,
);






















CREATE TABLE PessoaFisica (
    Pessoa_idPessoa INTEGER FOREIGN KEY REFERENCES Pessoa(idPessoa) ,
    CPF INTEGER NULL ,
);

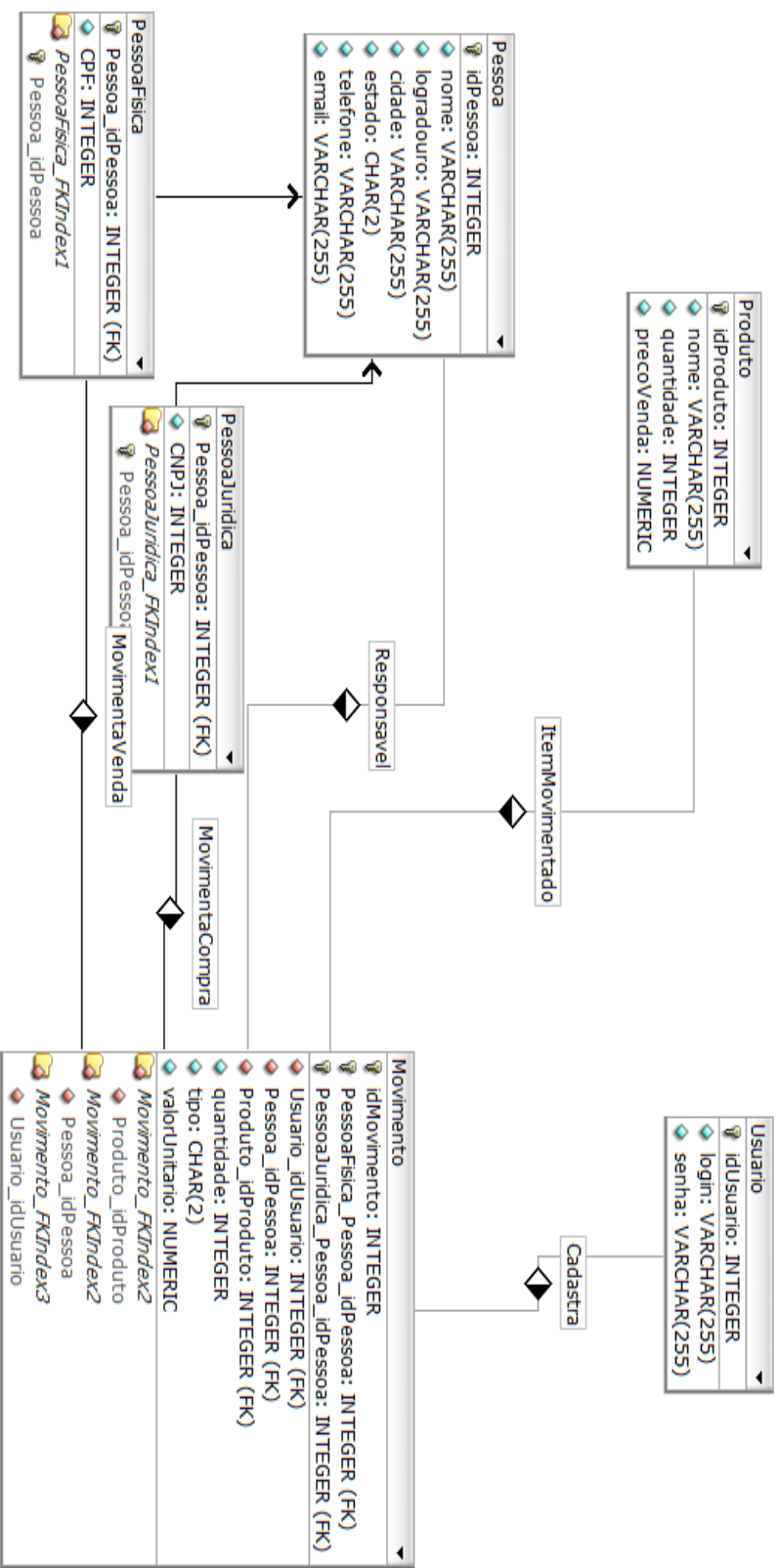
CREATE TABLE PessoaJuridica (
    Pessoa_idPessoa INTEGER FOREIGN KEY REFERENCES Pessoa(idPessoa) ,
    CNPJ BigInt NULL ,
);

```

Os resultados da execução.

[-]	[Grid Icon]	dbo.Movimento
	[-]	[Folder Icon] Colunas
		idMovimento (PKintnã nulo)
		Usuario_idUsuario (FKintnulo)
		Pessoa_idPessoa (FKintnulo)
		Produto_idProduto (FKintnulo)
		quantidade (intnulo)
		tipo (intnulo)
		valorUnitario (numeric(18,0)nulo)
[-]	[Grid Icon]	dbo.Pessoa
	[-]	[Folder Icon] Colunas
		idPessoa (PKintnã nulo)
		nome (varchar(255)nulo)
		logradouro (varchar(255)nulo)
		cidade (varchar(255)nulo)
		estado (char(2)nulo)
		telefone (varchar(255)nulo)
		email (varchar(255)nulo)
[-]	[Grid Icon]	dbo.PessoaFisica
	[-]	[Folder Icon] Colunas
		Pessoa_idPessoa (FKintnulo)
		CPF (intnulo)

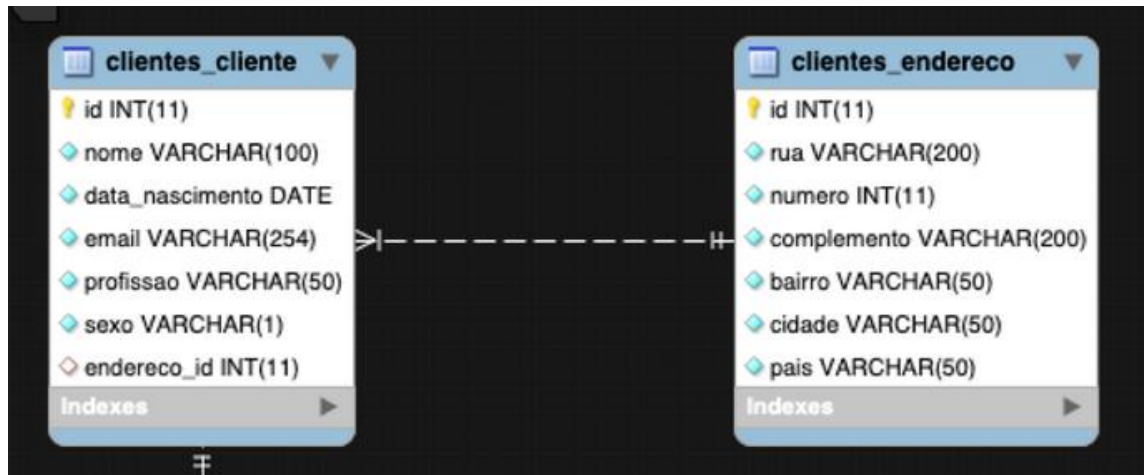
-   **dbo.PessoaJuridica**
 -   **Colunas**
 -  **Pessoa_idPessoa (FKintnulo)**
 -  **CNPJ (biquintnulo)**
-   **dbo.Produto**
 -   **Colunas**
 -  **idProduto (PKintnãõ nulo)**
 -  **nome (varchar(255)nulo)**
 -  **quantidade (intnulo)**
 -  **precoVenda (numeric(18,0)nulo)**
-   **dbo.Usuario**
 -   **Colunas**
 -  **idUsuario (PKintnãõ nulo)**
 -  **login (varchar(255)nulo)**
 -  **senha (varchar(255)nulo)**



Análise

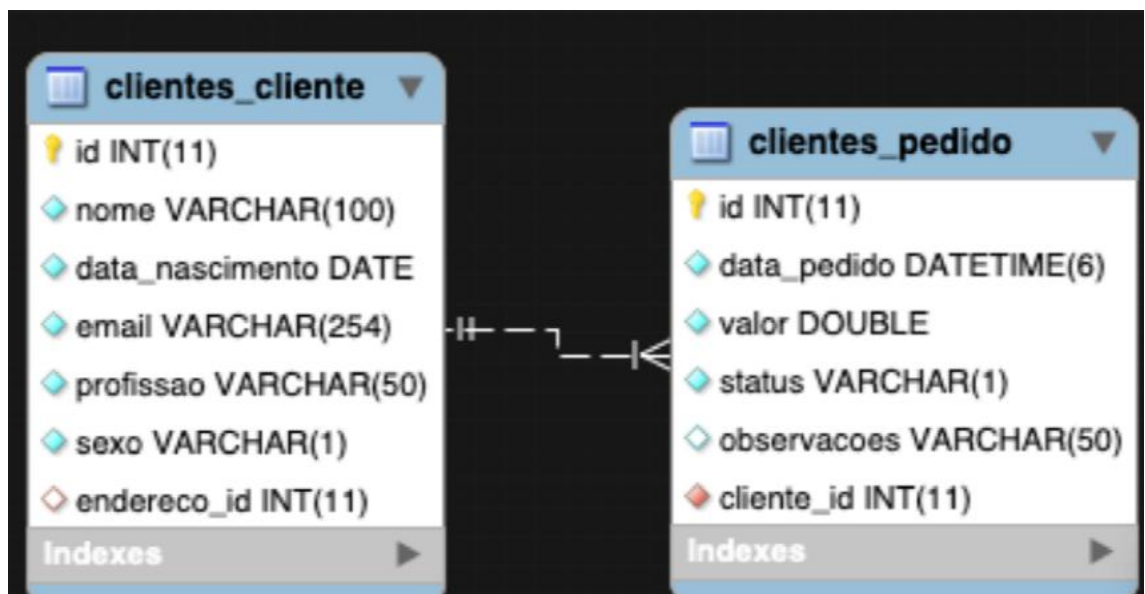
- a) Como são implementadas as diferentes cardinalidades, basicamente 1X1, 1XN ou NxN, em um banco de dados relacional?

R: A cardinalidades 1X1 define que um item de uma entidade só poderá se relacionar com um item de outra entidade.



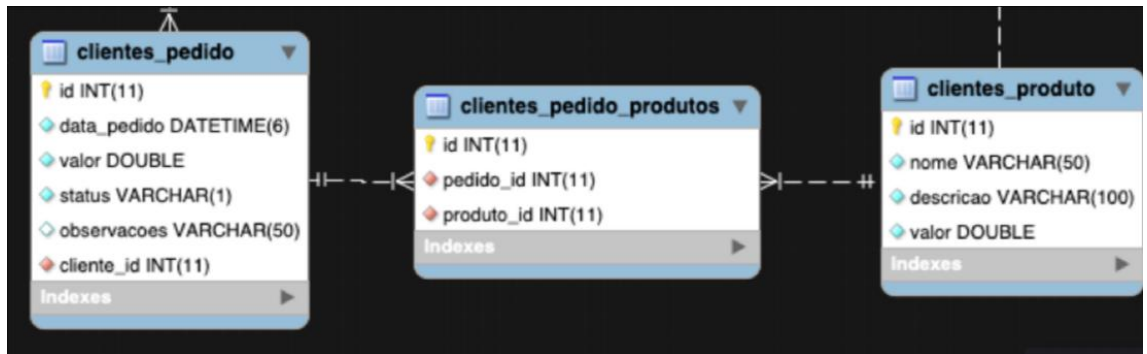
Na imagem acima, podemos ver que o **id** da tabela **clientes_endereco** está na tabela **clientes_cliente**.

A cardinalidade 1XN determina que um item de uma tabela pode se relacionar com vários itens de uma outra tabela.



Na imagem acima, podemos ver que o **id** da tabela **clientes_cliente** está na tabela **clientes_pedido**.

A cardinalidade NxN define que um item de uma tabela pode se relacionar com vários itens de uma outra tabela e vice-versa.



Para o relacionamento de `clientes_pedido` e `clientes_produto` é necessário criar uma tabela que contém Chave primária de ambos.

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

R: Deve ser usado **Generalização e Especialização** e a cardinalidade é 1X1.

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

R: As melhorias do uso do SMSS são:

- Criação e modificação rápida no Banco de dados.
- Adicionar tabelas, views e stored procedures.
- Modificação de tabelas, views e stored procedures.
- Teste dos objetos do banco de dados utilizando ferramentas externas de testes
- Implementar banco de dados para seus respectivo infra, remoto ou nuvem.
- Consulta facilitada no banco de dados.
- Importar e exportar dados.
- Backup do banco de dados.
- Restauração do Banco de dados.

Conclusão

A modelagem de dados auxilia na construção do banco de dados relacional como o SQL SERVER, permitindo a visualização do fluxo de dados e suas responsabilidades.

O banco de dados relacional deve ser pensado em como uma tabela irá influenciar outras como por exemplo um cliente pode ter apenas 1 ou mais telefones e como irei buscar tal informação se estão em tabelas diferentes.

A ferramenta DBDesigner não foi explicada, como se dá seu uso e como realizar os relacionamentos, dificultando o iniciante na modelagem de dados.

No download e instalação do SQL server deveria ter uma explicação detalhada do que realizar download.

O banco de dados poderia ter erros de normalização na tabela Pessoa , endereço, telefone e e mail deveriam ter tabelas próprias, e poderia ter uma flag com identificador de tipo de pessoa (Pessoa Fisica ou Pessoa Juridica) ao invés de criar 2 tabelas.

Tive dificuldades na criação e permissões de um novo logon, tive que configurar tanto o servidor quanto o logon.

2º Procedimento | Alimentando a Base

INSERTS

```
CREATE SEQUENCE LOJA_SEQUENCE
AS INT
START WITH 1
INCREMENT BY 1;

INSERT INTO [dbo].[Pessoa]
(idPessoa
, nome
, [logradouro]
, [cidade]
, [estado]
, [telefone]
, [email])
VALUES
(
    NEXT VALUE FOR LOJA_SEQUENCE
    , 'Joao'
    , 'Rua 12, Casa 3,Quitanda'
    , 'Riacho do Sul'
    , 'PA'
    , '1111-1111'
    , 'joao@riacho.com'
)
, (NEXT VALUE FOR LOJA_SEQUENCE
    , 'JJC'
    , 'Rua 11, Centro'
    , 'Riacho do Norte'
    , 'PA'
    , '1212-1212'
    , 'jjc@riacho.com' )

INSERT INTO [dbo].[PessoaFisica]
([Pessoa_idPessoa]
, [CPF])
VALUES
(1
, 111111111)

INSERT INTO [dbo].[PessoaJuridica]
([Pessoa_idPessoa]
, [CNPJ])
VALUES
( 2
, 22222222222222)
```



```

INSERT INTO [dbo].[Produto]
    ([nome]
    ,[quantidade]
    ,[precoVenda])
VALUES
    ('Banana',100,5)
    ,('Laranja',500,2)
    ,('Manga',800,4)
INSERT INTO [dbo].[Usuario]
    ([login]
    ,[senha])
VALUES
    ('op1','op2')
    ,('op2','op2')

INSERT INTO [dbo].[Movimento]
    ([Usuario_idUsuario]
    ,[Pessoa_idPessoa]
    ,[Produto_idProduto]
    ,[quantidade]
    ,[tipo]
    ,[valorUnitario])
VALUES
    (1,1,1,20,'S',4)
    ,(1,1,2,15,'S',2)
    ,(2,1,2,10,'S',3)
    ,(1,2,2,15,'E',5)
    ,(1,2,3,20,'E',4)

```

SELECTS

Dados completos de pessoas físicas.

```

SELECT * FROM Pessoa
right join PessoaFisica on idPessoa = Pessoa_idPessoa

```

	idPessoa	nome	logradouro	cidade	estado	telefone	email	Pessoa_idPessoa	CPF
1	1	Joao	Rua 12, Casa 3,Quitanda	Riacho do Sul	PA	1111-1111	joao@riacho.com	1	111111111

Dados completos de pessoas jurídicas.

```

SELECT * FROM Pessoa
right join PessoaJuridica on idPessoa = Pessoa_idPessoa

```

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

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

```

SELECT tipo as movimentacao
,pdt.nome as produto
,p.nome as fornecedor
, MV.quantidade

```

```

, valorUnitario as preco_unitario
, valorUnitario * MV.quantidade as valor_total
FROM Movimento as MV
join Produto as pdt on idProduto = Produto_idProduto
join Pessoa as p on idPessoa = MV.Pessoa_idPessoa
right join PessoaJuridica as PJ on idPessoa = PJ.Pessoa_idPessoa
where tipo = 'E'

```

	movimentacao	produto	fornecedor	quantidade	preco_unitario	valor_total
1	E	Laranja	JJC	15	5	75
2	E	Manga	JJC	20	4	80

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

```

SELECT tipo as movimentacao
,pdt.nome as produto
,p.nome as comprador
, MV.quantidade
, valorUnitario as preco_unitario
, valorUnitario * MV.quantidade as valor_total
FROM Movimento as MV
join Produto as pdt on idProduto = Produto_idProduto
join Pessoa as p on idPessoa = MV.Pessoa_idPessoa
right join PessoaFisica as PF on idPessoa = PF.Pessoa_idPessoa
where tipo = 'S'

```

	movimentacao	produto	comprador	quantidade	preco_unitario	valor_total
1	S	Banana	Joao	20	4	80
2	S	Laranja	Joao	15	2	30
3	S	Laranja	Joao	10	3	30

Valor total das entradas agrupadas por produto.

```

select nome as produto,sum( MV.quantidade * MV.valorUnitario ) as valor_total
from Movimento as MV
join Produto on idProduto = Produto_idProduto
where tipo = 'E'
group by nome

```

	nome	valor_total
1	Laranja	75
2	Manga	80

Valor total das saídas agrupadas por produto.

```

select nome as produto,sum( MV.quantidade * MV.valorUnitario ) as valor_total
from Movimento as MV

```

```

join Produto on idProduto = Produto_idProduto
where tipo = 'S'
group by nome

```

	produto	valor_total
1	Banana	80
2	Laranja	60

Operadores que não efetuaram movimentações de entrada

```

Select p.nome as operadores from pessoa as p
cross apply(
    SELECT count(1) as movimentacao FROM movimento as MV
    WHERE idPessoa = MV.Pessoa_idPessoa AND tipo = 'E'
) as t1
where movimentacao = 0

```

	operadores
1	Joao

Valor total de entrada, agrupado por operador.

```

SELECT nome,
sum( valorUnitario * MV.quantidade ) as valor_total
FROM Movimento as MV
join Pessoa as p on p.idPessoa = MV.Pessoa_idPessoa
WHERE tipo = 'E'
group by nome

```

	nome	valor_total
1	JJC	155

Valor total de saída, agrupado por operador.

```

SELECT nome,
sum( valorUnitario * MV.quantidade ) as valor_total
FROM Movimento as MV
join Pessoa as p on p.idPessoa = MV.Pessoa_idPessoa
WHERE tipo = 'S'
group by nome

```

nome	valor_total
Joao	140

Valor médio de venda por produto, utilizando média ponderada.

```
select idProduto, nome, dividendo/divisor as media_ponderada from (
    select Produto_idProduto,
        sum( valorUnitario * MV.quantidade ) as dividendo,
        sum( MV.quantidade ) as divisor
    from Movimento as MV
    where tipo = 'S'
    group by Produto_idProduto
) as t1
left join Produto on idProduto = Produto_idProduto
```

Resultados		Mensagens	
	idProduto	nome	media_ponderada
1	1	Banana	4.000000
2	2	Laranja	2.400000

Análise

a) Quais as diferenças no uso de sequence e identity?

SEQUENCE e IDENTITY está no fato de que as SEQUENCES são acionadas sempre quando forem necessárias, sem dependência de tabelas e campos no banco, onde pode ser chamada diretamente por aplicativos.

Outra diferença está que nas SEQUENCES, nós podemos obter o novo valor antes de usá-lo em um comando, diferente do IDENTITY, onde não podemos obter um novo valor. Além disso, com o IDENTITY não podemos gerar novos valores em uma instrução UPDATE, enquanto que com SEQUENCE, já podemos.

Com SEQUENCES, podemos definir valores máximos e mínimos, além de termos a possibilidade de informar que a mesma irá trabalhar de forma cíclica e com cache, além de podemos obter mais valores em sequencia de um só vez, utilizando para isso a procedure SP_SEQUENCE_GET_RANGE, onde então é permitido atribuímos os valores individuais para aumentar então o desempenho no uso da SEQUENCE.

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

A **chave estrangeira** é uma coluna em uma tabela que faz referência à **chave** primária de outra tabela. Ela estabelece relacionamentos entre as tabelas e permite a integridade referencial, garantindo que os registros relacionados existam em ambas as tabelas

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

Os operadores da álgebra relacional são:

- Select
- project
- theta join
- equijoin
- natural Join
- union
- intersection
- difference
- produto cartesiano
- divisão

calculo relacional:

- AND
- OR
- NOT

- d) Como é feito o agrupamento em consultas, e qual requisito é obrigatório?
Através do Group by, precisamos que as colunas a serem apresentadas estejam no tanto no select, tanto no group by.

Conclusão

O SSMS é um poderoso gerenciador de banco de dados relacional que facilita o dia a dia do programador.

Se bem estruturado pode se proteger de erros que eventualmente podem acontecer.