



## **RPG0015 - Vamos manter as informações!**

**Ricardo Alves dos Santos Junior - 202208681158**

Universidade Estácio de Sá

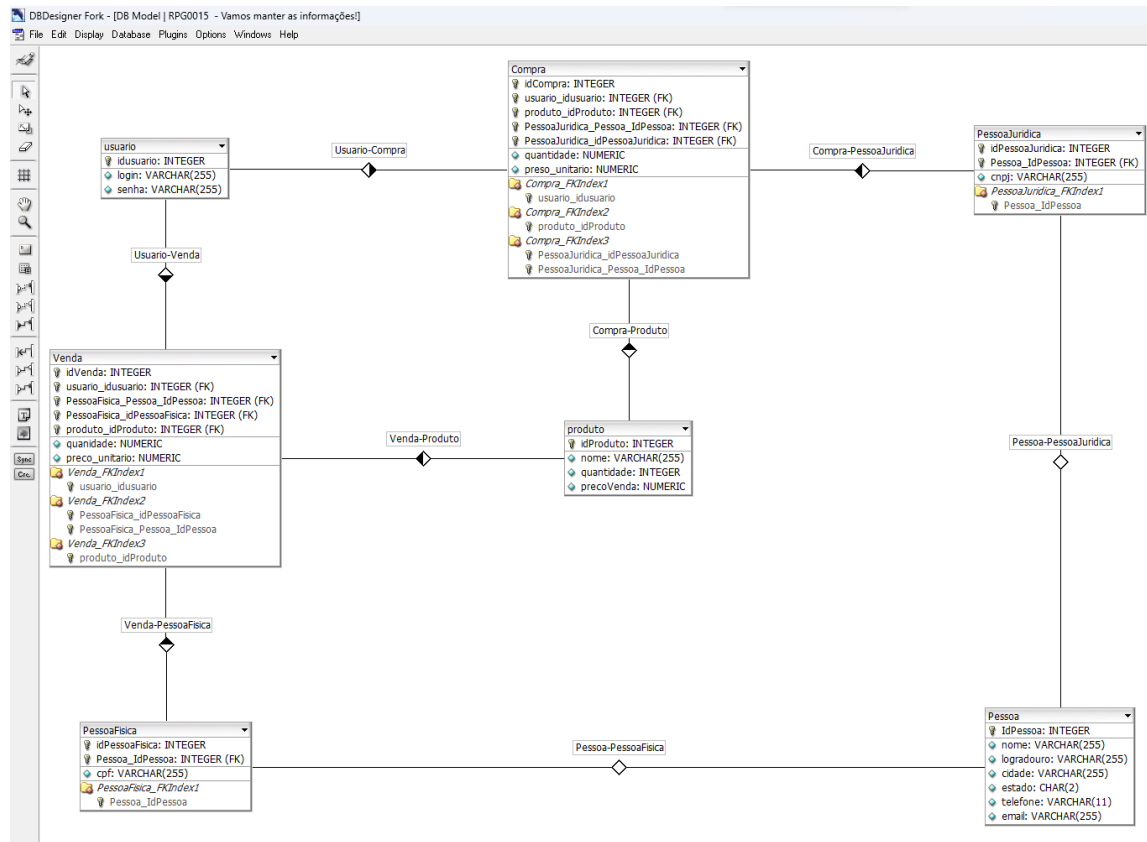
**Nível 2: Vamos Manter as Informações? –2023.3 – Terceiro Semestre**

### **Objetivo 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

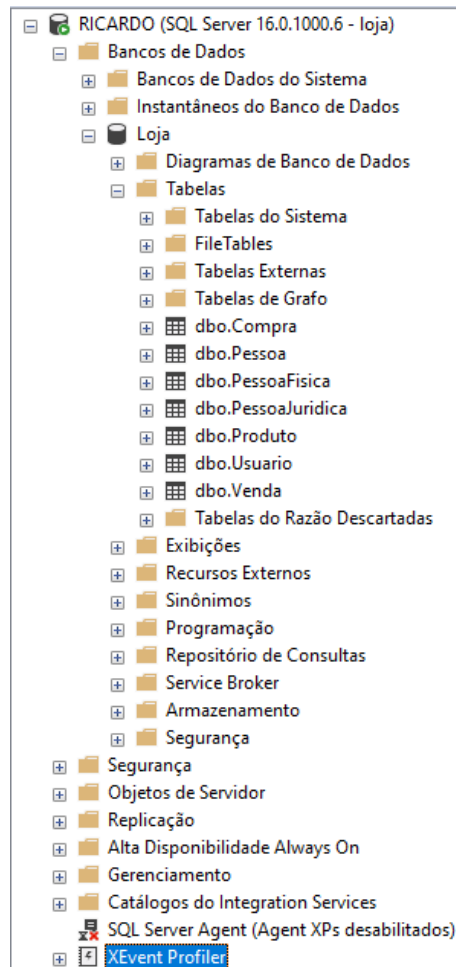
Imagem do Projeto desenvolvido no DBDesigner Fork:



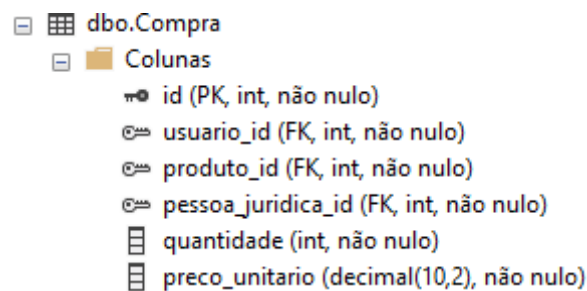
Código usado para a Criação do Banco seguindo os requisitos do Projeto:

```
1  -- Criar DATABASE
2  CREATE DATABASE Loja;
3
4  USE Loja;
5
6
7  -- Entidade Usuário
8  CREATE TABLE Usuario (
9      id INT IDENTITY(1,1) PRIMARY KEY,
10     login VARCHAR(250) NOT NULL,
11     senha VARCHAR(250) NOT NULL
12 );
13
14 -- Entidade Pessoa
15 CREATE TABLE Pessoa (
16     id INT IDENTITY(1,1) PRIMARY KEY,
17     nome VARCHAR(250) NOT NULL,
18     logradouro VARCHAR(250) NOT NULL,
19     cidade VARCHAR(250) NOT NULL,
20     estado VARCHAR(2) NOT NULL,
21     telefone VARCHAR(11) NOT NULL,
22     email VARCHAR(250) NOT NULL
23 );
24
25 -- Entidade PessoaFisica
26 CREATE TABLE PessoaFisica (
27     id INT IDENTITY(1,1) PRIMARY KEY,
28     cpf VARCHAR(14) NOT NULL,
29     pessoa_id INT NOT NULL,
30     FOREIGN KEY (pessoa_id) REFERENCES Pessoa(id)
31 );
32
33 -- Entidade PessoaJuridica
34 CREATE TABLE PessoaJuridica (
35     id INT IDENTITY(1,1) PRIMARY KEY,
36     cnpj VARCHAR(18) NOT NULL,
37     pessoa_id INT NOT NULL,
38     FOREIGN KEY (pessoa_id) REFERENCES Pessoa(id)
39 );
40
41 -- Entidade Produto
42 CREATE TABLE Produto (
43     id INT IDENTITY(1,1) PRIMARY KEY,
44     nome VARCHAR(250) NOT NULL,
45     quantidade INT NOT NULL,
46     preco DECIMAL(10,2) NOT NULL
47 );
48
49 -- Entidade Compra
50 CREATE TABLE Compra (
51     id INT IDENTITY(1,1) PRIMARY KEY,
52     usuario_id INT NOT NULL,
53     produto_id INT NOT NULL,
54     pessoa_juridica_id INT NOT NULL,
55     quantidade INT NOT NULL,
56     preco_unitario DECIMAL(10,2) NOT NULL,
57     FOREIGN KEY (usuario_id) REFERENCES Usuario(id),
58     FOREIGN KEY (produto_id) REFERENCES Produto(id),
59     FOREIGN KEY (pessoa_juridica_id) REFERENCES PessoaJuridica(id)
60 );
61
62 -- Entidade Venda
63 CREATE TABLE Venda (
64     id INT IDENTITY(1,1) PRIMARY KEY,
65     usuario_id INT NOT NULL,
66     produto_id INT NOT NULL,
67     pessoa_fisica_id INT NOT NULL,
68     quantidade INT NOT NULL,
69     preco_unitario DECIMAL(10,2) NOT NULL,
70     FOREIGN KEY (usuario_id) REFERENCES Usuario(id),
71     FOREIGN KEY (produto_id) REFERENCES Produto(id),
72     FOREIGN KEY (pessoa_fisica_id) REFERENCES PessoaFisica(id)
73 );
74
```

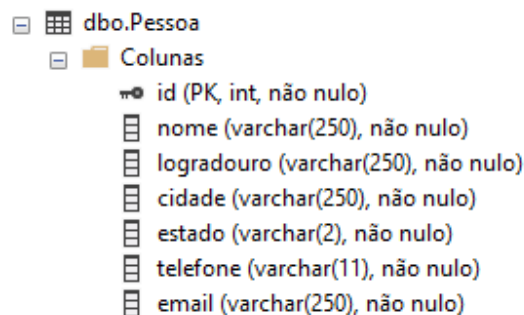
Resultado obtido após criar o Banco de dados Loja no SQL Server Management Studio:








Detalhando tabela Compra:








Detalhando tabela Pessoa:









Detalhando tabela PessoaFisica:

- [-]  **dbo.PessoaFisica**
  - [-]  **Colunas**
    -  **id** (PK, int, não nulo)
    -  **cpf** (varchar(14), não nulo)
    -  **pessoa\_id** (FK, int, não nulo)






Detalhando tabela PessoaJuridica:

- [-]  **dbo.PessoaJuridica**
  - [-]  **Colunas**
    -  **id** (PK, int, não nulo)
    -  **cnpj** (varchar(17), não nulo)
    -  **pessoa\_id** (FK, int, não nulo)









Detalhando tabela Produto:

- [-]  **dbo.Produto**
  - [-]  **Colunas**
    -  **id** (PK, int, não nulo)
    -  **nome** (varchar(250), não nulo)
    -  **quantidade** (int, não nulo)
    -  **preco** (decimal(10,2), não nulo)

Detalhando tabela Usuario:

- [-]  **dbo.Usuario**
  - [-]  **Colunas**
    -  **id** (PK, int, não nulo)
    -  **login** (varchar(250), não nulo)
    -  **senha** (varchar(250), não nulo)

Detalhando tabela Venda:

- [-]  **dbo.Venda**
  - [-]  **Colunas**
    -  **id** (PK, int, não nulo)
    -  **usuario\_id** (FK, int, não nulo)
    -  **produto\_id** (FK, int, não nulo)
    -  **pessoa\_fisica\_id** (FK, int, não nulo)
    -  **quantidade** (int, não nulo)
    -  **preco\_unitario** (decimal(10,2), não nulo)

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

Em um banco de dados relacional, as diferentes cardinalidades, como 1:1, 1:N e N:N, são implementadas através de tabelas e chaves estrangeiras.

Cardinalidade 1:1 é implementada quando cada registro em uma tabela está associado a exatamente um registro em outra tabela. Em um banco de dados relacional, isso é geralmente implementado através de uma chave estrangeira em uma das tabelas que referencia a chave primária da outra tabela.

Cardinalidade 1:N é implementada quando um registro em uma tabela pode estar associado a vários registros em outra tabela, mas cada registro na segunda tabela está associado a exatamente um registro na primeira tabela. Em um banco de dados relacional, isso é geralmente implementado através de uma chave estrangeira na tabela que representa o "N" lado do relacionamento.

Cardinalidade N:N é implementada quando os registros em duas tabelas podem estar associados de várias maneiras. Em um banco de dados relacional, isso é geralmente implementado através de uma tabela de junção que contém chaves estrangeiras que referenciam as chaves primárias das duas tabelas. Porém na sua estrutura de banco de dados, não há uma relação N:N explícita.

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

No modelo de banco de dados relacional, a herança é geralmente representada usando o relacionamento 1x1. Isso é feito através de uma estrutura de tabelas onde uma tabela pai contém informações comuns a todos os subtipos, e tabelas filhas contêm informações específicas para cada subtipo.

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

O SQL Server Management Studio (SSMS) é uma ferramenta que permite a melhoria da produtividade nas tarefas relacionadas ao gerenciamento do banco de dados. Lá você pode conectar-se à diferentes instâncias do SQL Server, pode executar consultas SQL, usando recursos como IntelliSense, formatação de código, realce de sintaxe, execução parcial, execução de plano, pode editar e projetar objetos do banco de dados, como tabelas, índices, procedimentos armazenados, pode explorar e gerenciar os dados do banco de dados, usando recursos como edição de dados, seleção de dados, filtragem de dados, pode monitorar e otimizar o desempenho do banco de dados, usando recursos como relatórios de desempenho, rastreamento de atividades, análise de consultas. Também pode automatizar e agendar tarefas do banco de dados, usando recursos como o Agente SQL Server, o SQL Server Integration Services, o SQL Server Analysis Services e pode personalizar e configurar o ambiente do SSMS, usando recursos como cores personalizadas, atalhos de teclado, opções de layout, opções de designer, entre outros recursos não citados

## 2º Procedimento – Alimentando a Base

Inserindo dados ao Banco:

Inserção de usuários:

```
--Inserindo Usuários:
INSERT INTO Usuario (login, senha)
VALUES ('op1', 'op1');

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

Inserção de Produtos:

```
--Inserindo Produtos:
INSERT INTO Produto (nome, quantidade, preco)
VALUES ('Banana' ,100 , 5.00);

INSERT INTO Produto (nome, quantidade, preco)
VALUES ('Laranja' ,500 , 2.00);

INSERT INTO Produto (nome, quantidade, preco)
VALUES ('Manga' ,800 , 4.00);
```

Inserção de Pessoa Física:

```
--Inserindo Pessoa Física:
INSERT INTO Pessoa (nome, logradouro, cidade, estado, telefone, email)
VALUES ('João da Silva', 'Rua das Flores, 123', 'Fortaleza', 'CE', '85987654321', 'joao@gmail.com');
INSERT INTO PessoaFisica (cpf, pessoa_id)
VALUES ('123.456.789-10', SCOPE_IDENTITY());
```

Inserção de Pessoa Jurídica:

```
--Inserindo Pessoa Jurídica:
INSERT INTO Pessoa (nome, logradouro, cidade, estado, telefone, email)
VALUES ('Empresa ABC', 'Avenida Brasil, 456', 'Fortaleza', 'CE', '85912345678', 'empresa@abc.com');
INSERT INTO PessoaJuridica (cnpj, pessoa_id)
VALUES ('12345.678/0001-90', SCOPE_IDENTITY());
```

Inserção de Compra:

```
-- Inserindo uma compra (Entrada):
INSERT INTO Compra (usuario_id, produto_id, pessoa_juridica_id, quantidade, preco_unitario)
VALUES (1, 1, 2, 10, 5.00);
```

Inserção de Venda

```
-- Inserindo uma venda (Saída):
INSERT INTO Venda (usuario_id, produto_id, pessoa_fisica_id, quantidade, preco_unitario)
VALUES (1, 1, 1, 5, 50.00);
```

Efetando consultas sobre os dados inseridos:

Selecionando todos os Usuários:

```
SELECT * FROM Usuario;
```

	id	login	senha
1	1	op1	op1
2	2	op2	op2

Selecionando todos os Produtos:

```
SELECT * FROM Produto;
```

	id	nome	quantidade	preco
1	1	Banana	100	5.00
2	2	Laranja	500	2.00
3	3	Manga	800	4.00

Selecionado Pessoas Físicas:

```
SELECT * FROM Pessoa JOIN PessoaFisica ON Pessoa.id = PessoaFisica.pessoa_id;
```

	id	nome	logradouro	cidade	estado	telefone	email	id	cpf	pessoa_id
1	1	João da Silva	Rua das Flores, 123	Fortaleza	CE	85987654321	joao@gmail.com	1	123.456.789-10	1

Selecionando Pessoas Jurídicas:

```
SELECT * FROM Pessoa JOIN PessoaJuridica ON Pessoa.id = PessoaJuridica.pessoa_id;
```

	id	nome	logradouro	cidade	estado	telefone	email	id	cnpj	pessoa_id
1	3	Empresa ABC	Avenida Brasil, 456	Fortaleza	CE	85912345678	empresa@abc.com	2	12345.678/0001-90	3

Selecionando Compras:

```
SELECT * FROM Compra;
```

	id	usuario_id	produto_id	pessoa_juridica_id	quantidade	preco_unitario
1	4	1	1	2	10	5.00

Selecionando Vendas:

```
SELECT * FROM Venda;
```

	id	usuario_id	produto_id	pessoa_fisica_id	quantidade	preco_unitario
1	1	1	1	1	5	50.00

Selecionando entrada e saída de produtos:

```
SELECT C.usuario_id, C.pessoa_juridica_id AS pessoa_id, C.produto_id, C.quantidade, 'E' AS tipo, C.preco_unitario
FROM Compra C
UNION ALL
SELECT V.usuario_id, V.pessoa_fisica_id AS pessoa_id, V.produto_id, V.quantidade, 'S' AS tipo, V.preco_unitario
FROM Venda V
ORDER BY usuario_id, pessoa_id, produto_id;
```



	usuario_id	pessoa_id	produto_id	quantidade	tipo	preco_unitario
1	1	1	1	5	S	50.00
2	1	2	1	10	E	5.00

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

IDENTITY é uma propriedade que pode ser adicionada a uma coluna numérica em uma tabela. Quando uma nova linha é inserida na tabela sem um valor para a coluna IDENTITY, o SQL Server automaticamente gera um valor único seguindo as regras definidas para a coluna IDENTITY.

SEQUENCE é um objeto no banco de dados que gera um valor numérico único. Ao contrário de IDENTITY, SEQUENCE não está associada a uma coluna específica e pode ser usada para gerar valores para várias colunas ou tabelas.

Você pode definir a semente e o incremento para um SEQUENCE, assim como você pode fazer com uma coluna IDENTITY.

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

As chaves estrangeiras são muito importantes, pois elas garantem que os dados em tabelas relacionadas permaneçam consistentes. Elas evitam a criação de registros órfãos, ou seja, registros que não têm correspondência em outra tabela, ao impor a integridade referencial. Isso significa que elas criam uma ligação entre as tabelas, de forma que qualquer alteração ou exclusão de valores na tabela primária seja refletida adequadamente na tabela estrangeira. Além de garantir a consistência dos dados, as chaves estrangeiras também trazem outros benefícios para o banco de dados, como facilitação da realização de consultas que envolvem várias tabelas, melhoram o desempenho das consultas, aumentam a segurança dos dados.

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

Álgebra Relacional:

A álgebra relacional inclui operadores como SELECT, PROJECT, JOIN, DIVIDE, SET e UNION.

Cálculo Relacional:

O cálculo relacional inclui operadores como  $\pi$  (projeção),  $\sigma$  (seleção),  $\rho$  (renomeação),  $\cup$  (união),  $\cap$  (interseção) e  $-$  (diferença).

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

A cláusula GROUP BY no SQL é usada para agrupar linhas que têm os mesmos valores em colunas específicas em um conjunto de resultados. É frequentemente usada com funções de agregação (COUNT, MAX, MIN, SUM, AVG) para agrupar os resultados por um ou mais campos.

Existem algumas regras que você precisa seguir ao usar GROUP BY:

Colunas na cláusula SELECT: Todas as colunas que não estão usando uma função de agregação devem estar na cláusula GROUP BY. Por exemplo, se você tem SELECT categoria, nome FROM produto GROUP BY categoria;, você precisará adicionar nome à cláusula GROUP BY, assim: SELECT categoria, nome FROM produto GROUP BY categoria, nome;

Colunas na cláusula WHERE: As colunas que estão sendo usadas na cláusula WHERE devem estar na cláusula GROUP BY, a menos que você esteja usando uma função de agregação.

## **Conclusão**

Primeiro, realizei uma análise detalhada dos requisitos do usuário e do domínio do problema para entender as entidades, atributos e relacionamentos necessários. Com base nessa análise, desenvolvi um design de banco de dados que incluiu tabelas para diferentes entidades, como Usuario, Pessoa, PessoaFisica, PessoaJuridica, Produto, Compra e Venda. Em seguida, usei um script SQL para criar o banco de dados, incluindo todas as tabelas e seus respectivos atributos. Também defini chaves primárias e estrangeiras para garantir a integridade dos dados e estabelecer relacionamentos entre as tabelas.

No design do banco de dados, utilizei ligações 1x1 e 1xN para representar diferentes tipos de relacionamentos. Por exemplo, a tabela PessoaFisica tem uma ligação 1x1 com a tabela Pessoa, e a tabela Compra tem uma ligação 1xN com a tabela Produto.

Além disso, usei funções como AI (Auto Increment) e NN (Not Null) para garantir que os dados sejam inseridos corretamente nas tabelas. A função AI foi usada para gerar automaticamente valores únicos para as chaves primárias, enquanto a função NN foi usada para garantir que certos campos não possam ser nulos.

Finalmente, utilizei várias funções SQL para exibir conjuntos de tabelas e analisar os dados. Por exemplo, usei a função SELECT para selecionar dados de várias tabelas, a função JOIN para combinar dados de várias tabelas com base em uma coluna comum, e a função GROUP BY para agrupar dados por um ou mais campos.

Em conclusão, o desenvolvimento deste banco de dados relacional demonstrou a importância da análise crítica, do projeto de banco de dados e da implementação de práticas SQL para garantir que os dados sejam armazenados de forma eficiente e eficaz. Ao seguir essas etapas, consegui criar um banco de dados que atende às necessidades do usuário e que pode ser facilmente expandido e modificado conforme necessário.