



Universidade Estácio

Campus Curitiba-PR

**Curso de Desenvolvimento Full Stack
Relatório da Missão Prática 2 - Mundo 3**

Disciplina: RPG0015 – Vamos manter as informações!

Nome: Marilene Gomes Duarte

Turma: 2023.2

<https://github.com/mari-lene>

Modelagem e implementação de um banco de dados simples com base o SQL Serve

1. Título da Prática: 1º Procedimento | Criando o banco de dados

2. Objetivo da Prática

- Identificar os requisitos de sistema e transformá-los no modelo adequado.
- Utilizar ferramentas de modelagem para bases de dados relacionais.
- Explorar a sintaxe SQL na criação das estruturas de banco (DDL).
- Explorar a sintaxe 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 Serve

3. Códigos solicitados: anexo no final do relatório.

4. Resultados da execução dos códigos

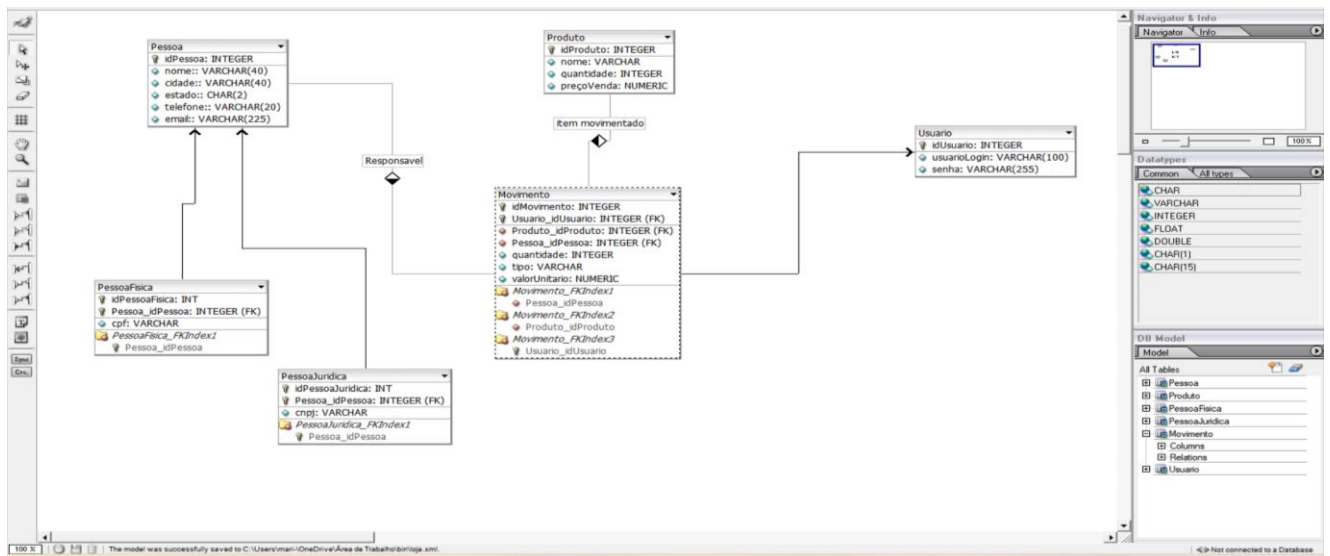


Figura 1: Diagrama entidade-Relacionamento (DER)

Figura 2: script de criação das tabelas do banco de dados, com uso de sequence orderPessoa.

```
USE Loja;

GO

CREATE SEQUENCE orderPessoa

AS INT

START WITH 1

INCREMENT BY 1;


CREATE TABLE Pessoa(
    idPessoa INTEGER NOT NULL,
    nome VARCHAR(255),
    endereço VARCHAR(255),
    cidade VARCHAR(255),
    estado CHAR(2),
    telefone VARCHAR(15),
    email VARCHAR(255),
    CONSTRAINT CPK_Pessoa PRIMARY KEY CLUSTERED(idPessoa ASC)
);

GO

CREATE TABLE PessoaFisica(
    FK_Pessoa_idPessoa INTEGER NOT NULL,
    cpf VARCHAR(11) NOT NULL,
    CONSTRAINT CPK_PessoaFisica PRIMARY KEY CLUSTERED(FK_Pessoa_idPessoa ASC),
    CONSTRAINT CFK_Pessoa_PessoaFisica FOREIGN KEY(FK_Pessoa_idPessoa) REFERENCES Pessoa(idPessoa)
);
```

```
        ON UPDATE CASCADE
        ON DELETE CASCADE
    );
GO

CREATE TABLE PessoaJuridica(
    FK_Pessoa_idPessoa INTEGER NOT NULL,
    cnpj VARCHAR(14) NOT NULL,
    CONSTRAINT CPK_PessoaJuridica PRIMARY KEY CLUSTERED(FK_Pessoa_idPessoa ASC),
    CONSTRAINT CFK_Pessoa_PessoaJuridica FOREIGN KEY(FK_Pessoa_idPessoa) REFERENCES Pessoa(idPessoa)
        ON UPDATE CASCADE
        ON DELETE CASCADE
    );
GO

CREATE TABLE Usuario(
    idUsuario INTEGER NOT NULL IDENTITY,
    loginName VARCHAR(20) NOT NULL,
    senha VARCHAR(20) NOT NULL,
    CONSTRAINT CPK_Usuario PRIMARY KEY CLUSTERED(idUsuario ASC)
    );
GO

CREATE TABLE Produto(
    idProduto INTEGER NOT NULL IDENTITY,
    nome VARCHAR(255) NOT NULL,
    quantidade INTEGER,
    precoVenda NUMERIC,
    CONSTRAINT CPK_Produto PRIMARY KEY CLUSTERED(idProduto ASC)
    );
GO

CREATE TABLE Movimento(
    idMovimento INTEGER NOT NULL IDENTITY,
    FK_Usuario_idUsuario INTEGER NOT NULL,
    FK_Pessoa_idPessoa INTEGER NOT NULL,
    FK_Produto_idProduto INTEGER NOT NULL,
    quantidade INTEGER,
```

```

tipo CHAR(1),
precoUnitario NUMERIC,
CONSTRAINT CPK_Movimento PRIMARY KEY CLUSTERED(idMovimento ASC),
CONSTRAINT CFK_Usuario_Movimento FOREIGN KEY(FK_Usuario_idUsuario) REFERENCES Usuario(idUsuario)
ON UPDATE CASCADE
ON DELETE CASCADE,
CONSTRAINT CFK_Pessoa_Movimento FOREIGN KEY(FK_Pessoa_idPessoa) REFERENCES Pessoa(idPessoa)
ON UPDATE CASCADE
ON DELETE CASCADE,
CONSTRAINT CFK_Produto_Movimento FOREIGN KEY(FK_Produto_idProduto) REFERENCES Produto(idProduto)
ON UPDATE CASCADE
ON DELETE CASCADE
);
GO

```

5. Análise e Conclusão

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

Para a definição da cardinalidade de um-para-um (1X1), é necessário criar 2 tabelas, estabelecer uma chave primária na tabela 1 e uma chave estrangeira na tabela 2, com referência para chave primária da tabela 1.



Figura 3: exemplo de cardinalidade um-para-um (1X1).

Por exemplo [1], como mostrado na fig. 3, a coluna ID da tabela Empregado é uma chave primária, que serve como referência para a coluna FK_Empregado_EmpregadoID da tabela Salário que é uma chave estrangeira. Ambas devem ser chaves únicas, ou seja, os valores da coluna ID não se repetem, assim como os valores da coluna FK_Empregado_EmpregadoID.

Para a definição da cardinalidade de um-para-muitos (1XN), é necessário criar 2 tabelas, estabelecer chaves primárias distintas para cada tabela e criar uma chave estrangeira na tabela 2 com referência para a chave primária da tabela 1.

A diferença aqui, em relação à cardinalidade um-para-um (1X1), é que os valores contidos na coluna da chave estrangeira podem se repetir.

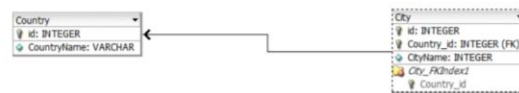


Figura 4: exemplo de cardinalidade um-para-muitos (1XN).

Por exemplo, a coluna ID da tabela Country é uma chave primária, que serve como referência para a coluna FK_Country_ID da tabela City, que é uma chave estrangeira. Na prática, isto pode ser imaginado como um país possuir várias cidades

A cardinalidade de muitos-para-muitos (NxN) ocorre quando múltiplas linhas de uma tabela estão relacionadas com múltiplas linhas de outra tabela. Por exemplo, clientes e produtos de um supermercado. Sistemas gerenciadores de banco de dados (SGBD) convencionais não suportam relações diretas de muitos-para-muitos (NxN) entre 2 tabelas. A solução para relacionar estas tabelas é a criação de uma 3ª tabela intermediária entre as 2 tabelas, de modo que seja criada uma relação de muitos-para-um (Nx1) da primeira com a terceira tabela, e outra relação de um-para-muitos (1XN) da terceira com a segunda tabela. Por exemplo, vários clientes possuem notas fiscais distintas, cujo conteúdo de cada nota fiscal contém vários produtos; portanto, a nota fiscal comporia, desta maneira, a tabela intermediária.

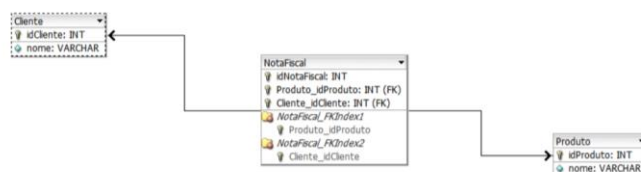


Figura 5: exemplo de cardinalidade muitos-para-muitos (NXN).

Em termos de chaves, as tabelas externas possuem chaves primárias, e a tabela intermediária possui 2 chaves estrangeiras que possuem como referências as respectivas chaves primárias das tabelas externas.

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

Em bancos de dados relacionais, o tipo de relacionamento utilizado para representar herança é a **generalização/especialização**. Essa técnica permite organizar as entidades em uma hierarquia, onde uma entidade superclasse (geralmente mais abrangente) pode ter uma ou mais entidades subclasses (mais específicas). Existem três tipos principais de generalização/especialização:

- **Generalização total:**

- A superclasse pode existir independentemente das subclasses.
- Todas as entidades da superclasse devem pertencer a pelo menos uma subclasse.
- Exemplo: Pessoa (superclasse) → Cliente e Funcionário (subclasses).

- **Generalização parcial:**

- A superclasse não pode existir independentemente das subclasses.
- Algumas entidades da superclasse podem não pertencer a nenhuma subclasse. ○ Exemplo: Animal (superclasse) → Cachorro e Gato (subclasses).

- **Herança múltipla:**

- Uma subclasse pode herdar mais de uma superclasse.
- Pode ser mais complexa de implementar e gerenciar.
- Exemplo: Aluno (subclasse) herda de Pessoa e Curso (superclasses).

O modo de representação no Diagrama de Entidades e Relacionamentos (ER), a generalização/especialização é representada por setas e triângulos. A superclasse fica no topo da hierarquia, com as subclasses abaixo dela. As setas indicam a relação de herança.

Na implementação em bancos de dados relacionais, a herança é geralmente implementada usando chaves primárias e estrangeiras. A chave primária da superclasse é usada como chave estrangeira nas subclasses. Isso garante que os dados das subclasses sejam consistentes com os da superclasse.

Possíveis vantagens do uso de herança incluem a redução da redundância de dados, a facilidade de atualização e gerenciamento de dados, e a melhoria da legibilidade do modelo de dados. Como desvantagens, é possível citar ser mais complexa de implementar e gerenciar, o que pode afetar o desempenho do banco de dados.

Algumas considerações sobre a escolha do tipo de generalização/especialização dependem das necessidades específicas do modelo de dados. Assim como é importante avaliar as vantagens e desvantagens antes de implementar a herança em um banco de dados relacional.

A herança também permite a uma tabela herdar a estrutura (colunas, chaves) e o comportamento (restrições, opções de armazenamento, gatilhos) de uma super-tabela acima dela na hierarquia [2].

Alguns bancos de dados como PostgreSQL [3] e Oracle [4], possuem palavras-chaves reservadas, como "INHERITS" para explicitamente implementar a herança. Por exemplo:

```
CREATE TABLE cities (  
    name          text,  
    population    float,  
    Elevation     int  
);
```

```
CREATE TABLE capitals (  
    state      char(2)  
) INHERITS (cities);
```

define uma tabela capitals, a qual herda os atributos da tabela cities.

(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) [5] é uma ferramenta poderosa que melhora significativamente a produtividade nas tarefas relacionadas ao gerenciamento do banco de dados, pelos seguintes motivos:

- **Automação de tarefas:** o SSMS permite automatizar tarefas repetitivas por meio de scripts, que economizam tempo e reduzem o risco de erros. É possível criar scripts para tarefas como backups, restaurações e criação de base de dados.
- **Interface gráfica intuitiva:** o SSMS oferece uma interface gráfica amigável que facilita a navegação e o gerenciamento do banco. A interface também fornece visualizações de dados, como diagramas de relacionamento de entidades e consultas de desempenho.
- **Ferramentas de desenvolvimento de consultas:** o SSMS inclui um editor de consultas completo que facilita a escrita e execução de consultas SQL. Esse editor oferece recursos como realce de sintaxe, autocompletar e IntelliSense, que podem ajudar a escrever consultas mais rapidamente e com menos erros. Também possui ferramentas para depuração de consultas e análise de desempenho.
- **Monitoramento e gerenciamento de desempenho:** o SSMS fornece várias ferramentas para monitorar e gerenciar o desempenho do banco de dados, para identificar gargalos, otimizar consultas e solucionar problemas de desempenho. Também oferece recursos para monitorar a saúde geral do banco de dados, como espaço em disco disponível e uso da CPU.
- **Gerenciamento de segurança:** o SSMS fornece ferramentas para gerenciar a segurança do banco de dados, para criar e gerenciar usuários, definir permissões e configurar auditoria. Também oferece recursos para criptografar dados e backups.
- **Integração com outras ferramentas:** o SSMS pode ser integrado com outras ferramentas da Microsoft, como o Visual Studio e o Azure Data Studio. Essa integração auxilia a ser mais produtivo ao trabalhar com bancos de dados SQL Server.
- **Outras vantagens:** o SSMS é gratuito para baixar e usar; é uma ferramenta multiplataforma que pode ser usada em Windows, MacOS e Linux; possui uma comunidade online ativa para auxiliar a sanar problemas e dúvidas.

1. Título da Prática: “2º Procedimento | Alimentando a Base” 2. 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.

3. Códigos solicitados: anexo no final do relatório

4. Resultado da execução dos códigos

```
INSERT INTO Pessoa(idPessoa,nome,endereco,cidade,estado,telefone,email)
VALUES (NEXT VALUE FOR orderPessoa, 'Andrea','Avenida A, 11','Rio Branco','AC','1111-1111','andre@gmail.com'),
(NEXT VALUE FOR orderPessoa, 'Bruna','Avenida B, 22','Salvador','BA','2222-2222','bruna@gmail.com'),
(NEXT VALUE FOR orderPessoa, 'Carlos','Avenida C, 33','Fortaleza','CE','3333-3333','carlos@gmail.com'),
(NEXT VALUE FOR orderPessoa, 'Distribuidora Delta','Avenida D, 44','Brasilia','DF','4444-4444','delta@gmail.com'),
(NEXT VALUE FOR orderPessoa, 'Empresa Echo','Avenida E, 55','Vitoria','ES','5555-5555','echo@gmail.com');

INSERT INTO PessoaFisica(FK_Pessoa_idPessoa,cpf)
VALUES (1,'111111111111'),
(2,'222222222222'),
(3,'333333333333');

INSERT INTO PessoaJuridica(FK_Pessoa_idPessoa,cnpj)
VALUES (4,'4444444444444444'),
(5,'5555555555555555');

INSERT INTO Usuario(loginName,senha)
VALUES ('op1','op1'),
('op2','op2');

INSERT INTO Produto(nome,quantidade,precoVenda)
VALUES ('Banana',100,'5.00'),
('Laranja',500,'2.00'),
('Manga',800,'4.00');

INSERT INTO Movimento(FK_Usuario_idUsuario,FK_Pessoa_idPessoa,FK_Produto_idProduto,quantidade,tipo,precoUnitario)
VALUES (1,1,1,10,'E',5.00),
(2,2,2,20,'S',2.00),
(1,3,3,30,'E',4.00);
```

Figura 6: script de inserção de dados, com uso de sequence orderPessoa.

Loja1.sql - MARLOJOA (MARI\mari- (57))

Loja1.sql - MARLOJOA (MARI\mari- (56))

SQLQuery4.sql - MA...(MARI\mari- (51))

100 %

Resultados Mensagens

idPessoa	nome	endereco	cidade	estado	telefone	email	cpf
1	Andrea	Avenida A, 11	Rio Branco	AC	1111-1111	andre@gmail.com	11111111111
2	Bruna	Avenida B, 22	Salvador	BA	2222-2222	bruna@gmail.com	22222222222
3	Carlos	Avenida C, 33	Fortaleza	CE	3333-3333	carlos@gmail.com	33333333333

idPessoa	nome	endereco	cidade	estado	telefone	email	cnj
1	4	Distribuidora Delta	Avenida D, 44	Brasilia	DF	4444-4444	444444444444444
2	5	Empresa Echo	Avenida E, 55	Vitoria	ES	5555-5555	555555555555555

idMovimento	FK_Usuario_idUsuario	FK_Pessoa_idPessoa	FK_Produto_idProduto	quantidade	tipo	precoUnitario	fornecedor	Produto	quantidade	precoUnitario	total
2	3	1	3	30	E	4	Carlos	Manga	30	4	120
3	4	1	1	10	E	5	Andrea	Banana	10	5	50
4	6	1	3	30	E	4	Carlos	Manga	30	4	120
5	7	1	1	10	E	5	Andrea	Banana	10	5	50
6	9	1	3	30	E	4	Carlos	Manga	30	4	120
7	10	1	1	10	E	5	Andrea	Banana	10	5	50
8	12	1	3	30	E	4	Carlos	Manga	30	4	120
9	13	1	1	10	E	5	Andrea	Banana	10	5	50

idMovimento	FK_Usuario_idUsuario	FK_Pessoa_idPessoa	FK_Produto_idProduto	quantidade	tipo	precoUnitario	comprador	Produto	quantidade	precoUnitario	total
1	2	2	2	20	S	2	Bruna	Laranja	20	2	40
2	5	2	2	20	S	2	Bruna	Laranja	20	2	40
3	8	2	2	20	S	2	Bruna	Laranja	20	2	40
4	11	2	2	20	S	2	Bruna	Laranja	20	2	40
5	14	2	2	20	S	2	Bruna	Laranja	20	2	40
6	17	2	2	20	S	2	Bruna	Laranja	20	2	40

Figura 7: resultado da prática.

Loja1.sql - MARLOJOA (MARI\mari- (57))

Loja1.sql - MARLOJOA (MARI\mari- (56))

SQLQuery4.sql - MA... (MARI\mari- (51))

100 %

Resultados Mensagens

6	9	1	3	30	E	4	Carlos	Manga	30	4	120
7	10	1	1	10	E	5	Andrea	Banana	10	5	50
8	12	1	3	30	E	4	Carlos	Manga	30	4	120
9	13	1	1	10	E	5	Andrea	Banana	10	5	50
10	15	1	3	30	E	4	Carlos	Manga	30	4	120
11	16	1	1	10	E	5	Andrea	Banana	10	5	50
12	18	1	3	30	E	4	Carlos	Manga	30	4	120

idMovimento	FK_Usuario_idUsuario	FK_Pessoa_idPessoa	FK_Produto_idProduto	quantidade	tipo	precoUnitario	comprador	Produto	quantidade	precoUnitario	total
1	2	2	2	20	S	2	Bruna	Laranja	20	2	40
2	5	2	2	20	S	2	Bruna	Laranja	20	2	40
3	8	2	2	20	S	2	Bruna	Laranja	20	2	40
4	11	2	2	20	S	2	Bruna	Laranja	20	2	40
5	14	2	2	20	S	2	Bruna	Laranja	20	2	40
6	17	2	2	20	S	2	Bruna	Laranja	20	2	40

nome	compras
1	Banana 300
2	Manga 720

nome	vendas
1	Laranja 240

idUsuario	loginName	senha
1	2	op2
2	3	op1
3	4	op2
4	5	op1
5	6	op2
6	7	op1
7	8	op2
8	9	op1

loginName	compras
1	op1 1020

loginName	vendas
1	op2 240

nome	media
1	Laranja 2.000000

Consulta executada com êxito.

MARI (16.0 RTM) MARI\mari- (57) LOJA 00:00:00 38 linhas

LI 6

Col 12

Car 12

INS

Figura 8: resultado prática

5. Análise e Conclusão

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

De acordo com o artigo [6], ambas são utilizadas para geração de numeração automática, mas a diferença é que identity é dependente da coluna da tabela onde é aplicada, enquanto sequence é independente da tabela. Se houver alguma situação na qual seja necessário manter uma numeração automática global (em múltiplas tabelas), então o uso de sequence é indicado.

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

Em Sistemas de Gerenciamento de Banco de Dados (SGBD) relacionais, as chaves estrangeiras (Foreign Key - FK) são fundamentais para manter a integridade referencial entre as tabelas, de maneira que a inserção, exclusão ou alteração de linhas em uma tabela primária seja imediatamente

refletida em outra tabela na qual a chave estrangeira está vinculada. Isto previne inconsistências e perdas de referências nos dados armazenados.

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

Segundo o artigo [7], as principais diferenças entre as duas abordagens podem ser mostradas na tabela abaixo:

Álgebra Relacional	Cálculo Relacional
É uma linguagem procedural.	É uma linguagem formal declarativa
Define como obter um resultado.	Define qual resultado obter.
A ordem das operações é especificada.	A ordem não é especificada
É independente de domínio.	Pode pertencer a um domínio específico
Sintaxe próxima da programação.	Sintaxe matemática abstrata.

Os operadores de Álgebra Relacional são seleção, projeção, união, diferença, produto cartesiano e junção [8]. O Cálculo Relacional possui poder expressivo idêntico à Álgebra Relacional [9], ou seja, todos os operadores possuem equivalência; de modo que uma expressão do Cálculo Relacional é igualmente uma relação que representa o resultado de uma consulta à base de dados. Uma representação do Cálculo Relacional por Tuplas é

$\{t \mid \text{COND}(t)\}$

na qual t é uma variável que representa as tuplas de uma relação e $\text{COND}(t)$ é uma condição sobre t . O resultado desta expressão é o conjunto das tuplas t que satisfaz $\text{COND}(t)$.

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

O agrupamento em consultas é realizado através da cláusula “GROUP BY”, utilizada para agrupar linhas baseada em uma função aplicada sobre uma coluna. O requisito obrigatório, além do uso da cláusula GROUP BY, é incluir alguma função de agrupamento tal como: SUM, COUNT, AVG, MAX, MIN, entre outras. Por exemplo, dada a tabela Alunos:

Tabela alunos

ID	Nome	curso
1	André	Direito
2	Bruna	Medicina
3	Carlos	Engenharia
4	Daniela	Direito
5	Eduardo	Medicina
6	Fernanda	Direito

Para determinar a quantidade de alunos por curso, a query de agrupamento é:

```
SELECT COUNT(ID), Curso FROM Alunos GROUP BY Curso;
```

Cujo resultado, é mostrado abaixo:

COUNT (ID)	CURSO
3	Direito
2	Medicina
1	Engenharia

Referencias

[1] “Relationships in SQL – Complete Guide With Examples”

Disponível em <https://blog.devart.com/types-of-relationships-in-sql-server-database.htm>

Acesso em 28 de julho de 2024.

[2] “Table inheritance”

Disponível em <https://www.ibm.com/docs/en/informix-servers/14.10?topic=inheritance-table>

Acesso em 28 de julho de 2024.

[3] “PostgreSQL documentation - Inheritance”

Disponível em <https://www.postgresql.org/docs/current/ddl-inherit.html>

Acesso em 28 de julho de 2024.

[4] “Inheritance in SQL Object Types”

Disponível em <https://docs.oracle.com/en/database/oracle/oracle-database/19/adobj/inheritance-in-sql-object-types.html>

Acesso em 28 de julho de 2024.

[5] “O que é o SSMS (SQL Server Management Studio)?”

Disponível em <https://learn.microsoft.com/pt-br/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver16>

Acesso em 28 de julho de 2024.

[6] “Sequences compared to identity columns”

Disponível em <https://www.ibm.com/docs/en/ias?topic=sequences-compared-identity-columns>

Acesso em 28 de julho de 2024.

[7] Diferença entre Álgebra Relacional e Cálculo Relacional

Disponível em: <https://acervolima.com/diferenca-entre-algebra-relacional-e-calculo-relacional/>

Acesso em 28 de julho de 2024.

[8] Álgebra e Cálculo Relacional

Disponível em [Sistemas de Banco de Dados \(ufu.br\)](#)

Acesso em 28 de julho de 2024.

[9] Álgebra e Cálculo Relacional

Disponível em [Microsoft PowerPoint - parteIV \[Modo de Compatibilidade\] \(up.pt\)](#)

Acesso em 28 de julho de 2024.

Códigos solicitados

```
USE Loja;  
GO  
CREATE SEQUENCE orderPessoa  
AS INT  
START WITH 1
```

INCREMENT BY 1;

```
CREATE TABLE Pessoa(  
    idPessoa INTEGER NOT NULL,  
    nome VARCHAR(255),  
    endereço VARCHAR(255),  
    cidade VARCHAR(255),  
    estado CHAR(2),  
    telefone VARCHAR(15),  
    email VARCHAR(255),  
    CONSTRAINT CPK_Pessoa PRIMARY KEY CLUSTERED(idPessoa ASC)  
);  
GO
```

```
CREATE TABLE PessoaFisica(  
    FK_Pessoa_idPessoa INTEGER NOT NULL,  
    cpf VARCHAR(11) NOT NULL,  
    CONSTRAINT CPK_PessoaFisica PRIMARY KEY CLUSTERED(FK_Pessoa_idPessoa ASC),  
    CONSTRAINT CFK_Pessoa_PessoaFisica FOREIGN KEY(FK_Pessoa_idPessoa) REFERENCES Pessoa(idPessoa)  
        ON UPDATE CASCADE  
        ON DELETE CASCADE  
);  
GO
```

```
CREATE TABLE PessoaJuridica(  
    FK_Pessoa_idPessoa INTEGER NOT NULL,  
    cnpj VARCHAR(14) NOT NULL,  
    CONSTRAINT CPK_PessoaJuridica PRIMARY KEY CLUSTERED(FK_Pessoa_idPessoa ASC),  
    CONSTRAINT CFK_Pessoa_PessoaJuridica FOREIGN KEY(FK_Pessoa_idPessoa) REFERENCES Pessoa(idPessoa)  
        ON UPDATE CASCADE  
        ON DELETE CASCADE  
);  
GO
```

```
CREATE TABLE Usuario(  
    idUsuario INTEGER NOT NULL IDENTITY,  
    loginName VARCHAR(20) NOT NULL,  
    senha VARCHAR(20) NOT NULL,  
    CONSTRAINT CPK_Usuario PRIMARY KEY CLUSTERED(idUsuario ASC)  
);  
GO
```

```

CREATE TABLE Produto(
    idProduto INTEGER NOT NULL IDENTITY,
    nome VARCHAR(255) NOT NULL,
    quantidade INTEGER,
    precoVenda NUMERIC,
    CONSTRAINT CPK_Produto PRIMARY KEY CLUSTERED(idProduto ASC)
);
GO

CREATE TABLE Movimento(
    idMovimento INTEGER NOT NULL IDENTITY,
    FK_Usuario_idUsuario INTEGER NOT NULL,
    FK_Pessoa_idPessoa INTEGER NOT NULL,
    FK_Produto_idProduto INTEGER NOT NULL,
    quantidade INTEGER,
    tipo CHAR(1),
    precoUnitario NUMERIC,
    CONSTRAINT CPK_Movimento PRIMARY KEY CLUSTERED(idMovimento ASC),
    CONSTRAINT CFK_Usuario_Movimento FOREIGN KEY(FK_Usuario_idUsuario) REFERENCES Usuario(idUsuario)
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT CFK_Pessoa_Movimento FOREIGN KEY(FK_Pessoa_idPessoa) REFERENCES Pessoa(idPessoa)
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT CFK_Produto_Movimento FOREIGN KEY(FK_Produto_idProduto) REFERENCES Produto(idProduto)
        ON UPDATE CASCADE
        ON DELETE CASCADE
);
GO

```

```

INSERT INTO Pessoa(idPessoa,nome,endereco,cidade,estado,telefone,email)
VALUES (NEXT VALUE FOR orderPessoa, 'Andrea','Avenida A, 11','Rio Branco','AC','1111-1111','andre@gmail.com'),
(NEXT VALUE FOR orderPessoa, 'Bruna','Avenida B, 22','Salvador','BA','2222-2222','bruna@gmail.com'),
(NEXT VALUE FOR orderPessoa, 'Carlos','Avenida C, 33','Fortaleza','CE','3333-3333','carlos@gmail.com'),
(NEXT VALUE FOR orderPessoa, 'Distribuidora Delta','Avenida D, 44','Brasilia','DF','4444-4444','delta@gmail.com'),
(NEXT VALUE FOR orderPessoa, 'Empresa Echo','Avenida E, 55','Vitoria','ES','5555-5555','echo@gmail.com');

```

```
INSERT INTO PessoaFisica(FK_Pessoa_idPessoa,cpf)
```

```
VALUES (1,'11111111111'),
```

```
(2,'22222222222'),
```

```
(3,'33333333333');
```

```
INSERT INTO PessoaJuridica(FK_Pessoa_idPessoa,cnpj)
```

```
VALUES (4,'444444444444444'),
```

```
(5,'555555555555555');
```

```
INSERT INTO Usuario(loginName,senha)
```

```
VALUES ('op1','op1'),
```

```
('op2','op2');
```

```
INSERT INTO Produto(nome,quantidade,precoVenda)
```

```
VALUES ('Banana',100,'5.00'),
```

```
('Laranja',500,'2.00'),
```

```
('Manga',800,'4.00');
```

```
INSERT INTO Movimento(FK_Usuario_idUsuario,FK_Pessoa_idPessoa,FK_Produto_idProduto,quantidade,tipo,precoUnitario)
```

```
VALUES (1,1,1,10,'E',5.00),
```

```
(2,2,2,20,'S',2.00),
```

```
(1,3,3,30,'E',4.00);
```

```
-- item (a)
```

```
SELECT p.*, pf.cpf
```

```
FROM Pessoa p
```

```
INNER JOIN PessoaFisica pf ON p.idPessoa = pf.FK_Pessoa_idPessoa;
```

```
-- item (b)
```

```
SELECT p.*, pj.cnpj
```

```
FROM Pessoa p
```

```
INNER JOIN PessoaJuridica pj ON p.idPessoa = pj.FK_Pessoa_idPessoa;
```

```
-- item (c)
```

```
SELECT m.*, p.nome as fornecedor, pr.nome as Produto, m.quantidade, m.precoUnitario, (m.quantidade * m.precoUnitario) as total
```

```
FROM Movimento m
```

```
INNER JOIN Pessoa p ON p.idPessoa = m.FK_Pessoa_idPessoa
```

```
INNER JOIN Produto pr ON pr.idProduto = m.FK_Produto_idProduto
```

```
WHERE m.tipo = 'E';
```

```
-- item (d)
```

```
SELECT m.*, p.nome as comprador, pr.nome as Produto, m.quantidade, m.precoUnitario, (m.quantidade * m.precoUnitario) as total
```

```
FROM Movimento m
```

```
INNER JOIN Pessoa p ON m.FK_Pessoa_idPessoa = p.idPessoa
```

```
INNER JOIN Produto pr ON m.FK_Produto_idProduto = pr.idProduto
```

```
WHERE m.tipo = 'S';
```

```
-- item (e)
```

```
SELECT pr.nome, SUM(m.quantidade * m.precoUnitario) as compras
```

```
FROM Movimento m
```

```
INNER JOIN Produto pr ON m.FK_Produto_idProduto = pr.idProduto
```

```
WHERE m.tipo = 'E'
```

```
GROUP BY pr.nome;
```

```
-- item (f)
```

```
SELECT pr.nome, SUM(m.quantidade * m.precoUnitario) as vendas
```

```
FROM Movimento m
```

```
INNER JOIN Produto pr ON m.FK_Produto_idProduto = pr.idProduto
```

```
WHERE m.tipo = 'S'
```

```
GROUP BY pr.nome;
```

```
-- item (g)
```

```
SELECT u.*
```

```
FROM Usuario u
```

```
LEFT JOIN Movimento m ON u.idUsuario = m.FK_Usuario_idUsuario AND m.tipo = 'E'
```

```
WHERE m.idMovimento IS NULL;
```

```
-- item (h)
```

```
SELECT u.loginName, SUM(m.precoUnitario * m.quantidade) as compras
```

```
FROM Movimento m
```

```
INNER JOIN Usuario u ON m.FK_Usuario_idUsuario = u.idUsuario
```

```
WHERE m.tipo = 'E'
```

```
GROUP BY u.loginName;
```

```
-- item (i)
```

```
SELECT u.loginName, SUM(m.precoUnitario * m.quantidade) as vendas
```

```
FROM Movimento m
```

```
INNER JOIN Usuario u ON m.FK_Usuario_idUsuario = u.idUsuario
```

```
WHERE m.tipo = 'S'
```



```
GROUP BY u.loginName;

-- item (j)
SELECT pr.nome, SUM(m.precoUnitario * m.quantidade) / SUM(m.quantidade) as media
FROM Movimento m
INNER JOIN Produto pr ON m.FK_Produto_idProduto = pr.idProduto
WHERE m.tipo = 'S'
GROUP BY pr.nome;
```