Estácio	Universidade Estácio Campus Castelo Curso de Desenvolvimento Full Stack Relatório da Missão Prática 2 - Mundo 3
Disciplina:	RPG0015 - Vamos manter as informações!
Nome:	Breno Ambrosim Louzada
Turma:	2023.1

Modelagem e implementação de um banco de dados simples, utilizando como base o SQL Server

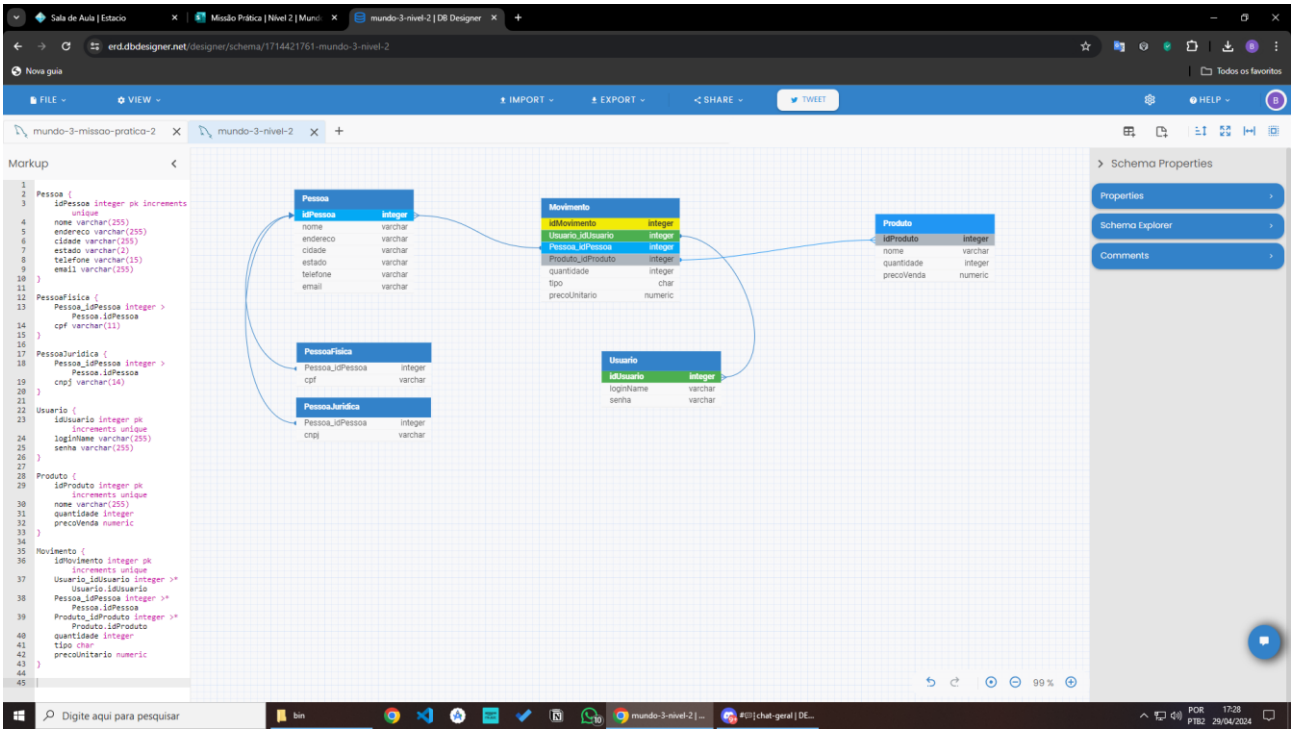
1. Título da Prática: “1º Procedimento | Criando o Banco de Dados”

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. Resultados da execução dos códigos



(link: https://dbdesigner.page.link/wpM6e4T5NV17cfHv5)

```
USE Loja;
GO


CREATE SEQUENCE orderPessoa
AS INT
START WITH 1
INCREMENT BY 1;


CREATE TABLE Pessoa(
  idPessoa INTEGER NOT NULL,
  nome VARCHAR(255),
  endereco 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

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

1x1 (Um para Um): Você usa duas tabelas, onde uma tem uma chave primária que também é usada como chave estrangeira na outra tabela. Geralmente você faz isso quando quer separar informações que não são sempre necessárias. Por exemplo, informações pessoais em uma tabela e informações sensíveis como dados de login em outra.

1xN (Um para Muitos): Neste caso, você tem uma chave primária em uma tabela e uma chave estrangeira na outra. É o tipo mais comum de relação. Por exemplo, um cliente (`ClienteID` como chave primária) pode fazer vários pedidos (cada pedido com um `ClienteID` como chave estrangeira).

NxN (Muitos para Muitos): Aqui você precisa de uma tabela extra, chamada tabela de junção, que tem duas chaves estrangeiras que juntas formam uma chave primária composta. Isso é usado quando itens de uma tabela podem se relacionar com vários itens de outra tabela. Um bom exemplo é estudantes e cursos: um estudante pode se inscrever em vários cursos e um curso pode ter vários estudantes.

2. Que tipo de relacionamento deve ser utilizado para representar o uso de herança em bancos de dados relacionais?

Para representar herança em bancos de dados relacionais, o tipo de relacionamento mais comum é o "relacionamento de subtipo/supertipo". Nesse modelo, uma tabela base, que representa o supertipo, contém as colunas comuns a todas as entidades, enquanto tabelas adicionais representam os subtipos e contêm colunas específicas para as características únicas de cada subtipo. Cada tabela de subtipo tem uma chave estrangeira que se refere à chave primária da tabela supertipo, estabelecendo uma relação de herança.

3. 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) aumenta a produtividade ao oferecer uma interface gráfica intuitiva para gerenciamento de banco de dados, recursos de autocompletar e coloração de sintaxe para facilitar a escrita de código, ferramentas integradas para monitoramento e otimização de desempenho, e assistentes para simplificar tarefas complexas. Além disso, facilita o debugging, a gestão de segurança e a manutenção do banco de dados, tudo em uma única plataforma. No meu caso, tive que usar o DBeaver, que é uma ferramenta universal que faz o a mesma coisa que o

SSMS faz, porem com algumas melhorias.

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. Resultados 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,'11111111111'),
    (2,'22222222222'),
    (3,'33333333333');


INSERT INTO PessoaJuridica(FK_Pessoa_idPessoa,cnpj)
  VALUES (4,'44444444444444'),
    (5,'55555555555555');


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,precoUnita
rio)
  VALUES (1,1,1,10,'E',5.00),
    (2,2,2,20,'S',2.00),
    (1,3,3,30,'E',4.00);
```

```
SELECT * FROM Pessoa;
```

100 %

Results | Messages

	idPessoa	nome	endereco	cidade	estado	telefone	email
1	1	Andrea	Avenida A, 11	Rio Branco	AC	1111-1111	andre@gmail.com
2	2	Bruna	Avenida B, 22	Salvador	BA	2222-2222	bruna@gmail.com
3	3	Carlos	Avenida C, 33	Fortaleza	CE	3333-3333	carlos@gmail.com
4	4	Distribuidora Delta	Avenida D, 44	Brasilia	DF	4444-4444	delta@gmail.com
5	5	Empresa Echo	Avenida E, 55	Vitoria	ES	5555-5555	echo@gmail.com

.

```
SELECT * FROM PessoaFisica;
```

100 %

Results | Messages

	FK_Pessoa_idPessoa	cpf
1	1	11111111111
2	2	22222222222
3	3	33333333333

```
SELECT * FROM PessoaJuridica;
```

100 %

Results | Messages

	FK_Pessoa_idPessoa	cnpj
1	4	44444444444444
2	5	55555555555555

```
SELECT * FROM Usuario;
```

100 %

Results | Messages

	idUsuario	loginName	senha
1	1	op1	op1
2	2	op2	op2

```sql
SELECT * FROM Produto;
```

100 %

Results | Messages

	idProduto	nome	quantidade	precoVenda
1	1	Banana	100	5
2	2	Laranja	500	2
3	3	Manga	800	4

```sql
SELECT * FROM Movimento;
```

100 %

Results | Messages

	idMovimento	FK_Usuario_idUsuario	FK_Pessoa_idPessoa	FK_Produto_idProduto	quantidade	tipo	precoUnitario
1	1	1	1	1	10	E	5
2	2	2	2	2	20	S	2
3	3	1	3	3	30	E	4

```sql
SELECT p.*, pf.cpf
FROM Pessoa p
INNER JOIN PessoaFisica pf ON p.idPessoa = pf.FK_Pessoa_idPessoa;
```

100 %

Results | Messages

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

```sql
SELECT p.*, pj.cnpj
FROM Pessoa p
INNER JOIN PessoaJuridica pj ON p.idPessoa = pj.FK_Pessoa_idPessoa;
```

100 %

Results | Messages

	idPessoa	nome	endereco	cidade	estado	telefone	email	cnpj
1	4	Distribuidora Delta	Avenida D, 44	Brasilia	DF	4444-4444	delta@gmail.com	44444444444444
2	5	Empresa Echo	Avenida E, 55	Vitoria	ES	5555-5555	echo@gmail.com	55555555555555

```sql
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';
```

100 %

Results | Messages

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

```sql
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';
```

100 %

Results | Messages

	idMovimento	FK_Usuario_idUsuario	FK_Pessoa_idPessoa	FK_Produto_idProduto	quantidade	tipo	precoUnitario	comprador	Produto	quantidade	precoUnitario	total
1	4	2	2	2	20	S	2	Bruna	Laranja	20	2	40

```sql
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;
```

100 %

Results | Messages

	nome	compras
1	Banana	50
2	Manga	120

```sql
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;
```

100 %

Results | Messages

	nome	vendas
1	Laranja	40

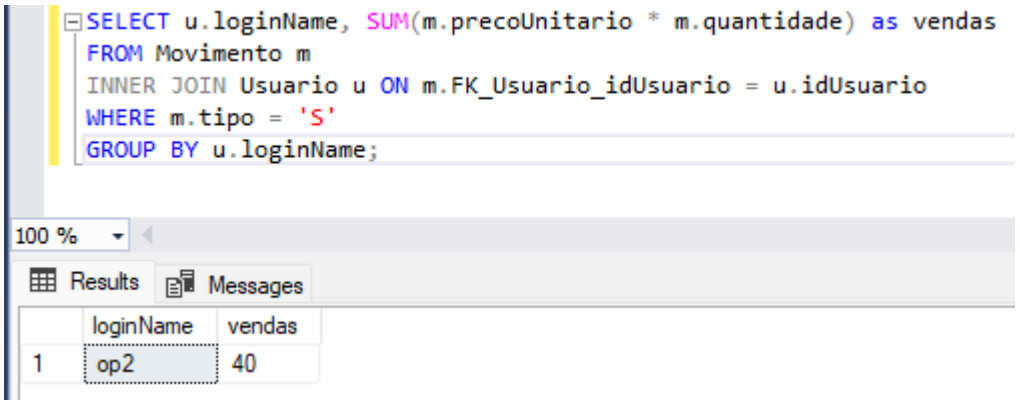
```sql
SELECT u.*
SELECT u.loginName, SUM(m.precoUnitario * m.quantidade) as compras
FROM Movimento m
INNER JOIN Usuario u ON m.FK_Usuario_idUsuario = u.idUsuario
WHERE m.tipo = 'E'
GROUP BY u.loginName;
```

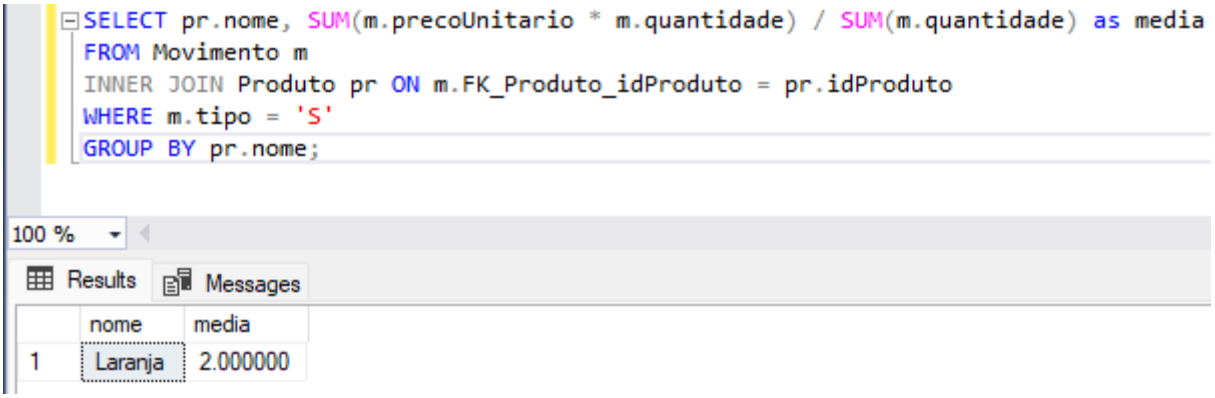
100 %

100 %

Results | Messages

	loginName	compras
1	op1	170





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 que *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 da 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 | COND(t)}
```

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

Anexo I: códigos dos scripts SQL

script01_create.sql

```
USE Loja;
GO

CREATE SEQUENCE orderPessoa
AS INT
START WITH 1
INCREMENT BY 1;

CREATE TABLE Pessoa(
  idPessoa INTEGER NOT NULL,
  nome VARCHAR(255),
  endereco 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
```

script02_insert.sql

```
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,'44444444444444'),
    (5,'55555555555555');

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,precoUnita
rio)
  VALUES (1,1,1,10,'E',5.00),
    (2,2,2,20,'S',2.00),
    (1,3,3,30,'E',4.00);
```

script03_select.sql

```
-- 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';
```

```sql
-- 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;
```