



UNIVERSIDADE
Estácio de Sá

Universidade	Estácio de Sá
Campus	Polo de Cobilândia / Vila – Velha/ES
Nome do Curso	Desenvolvimento Full Stack
Nome da Disciplina	RPG0015 – Vamos Manter as Informações
Turma	9001
Semestre	Primeiro Semestre de 2024
Integrantes do Grupo	Tiago de Jesus Pereira Furtado
Matrícula	202306189045

**VILA VELHA
2024**

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

👉 1º Procedimento | Criando o Banco de Dados

1- Objetivo da Prática:

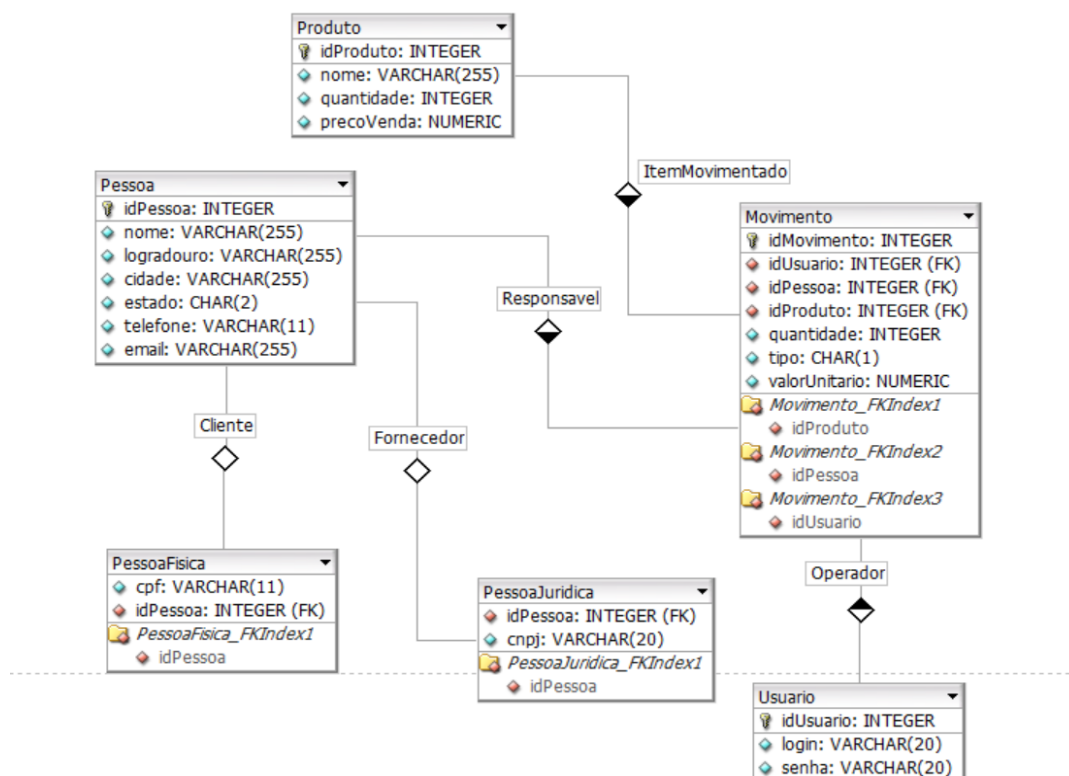
- Identificar os requisitos de um sistema e transformá-los no modelo adequado.
- Utilizar ferramentas de modelagem para bases de dados relacionais.
- Explorar a sintaxe SQL na criação das estruturas do banco (DDL).
- Explorar a sintaxe SQL na consulta e manipulação de dados (DML).
- No final do exercício, 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 - Todos os códigos solicitados neste roteiro de aula:

Modelagem Feita Utilizando o DBDesigner-Fork

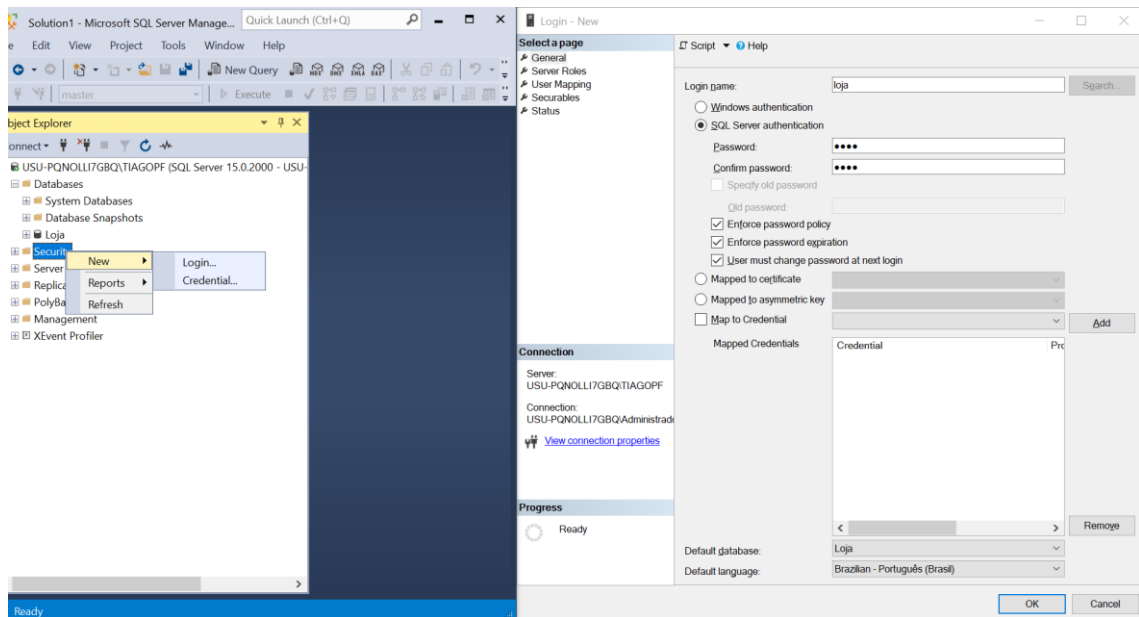
- 2- Definir o modelo de dados para um sistema com as características apresentadas nos tópicos seguintes:
- Deve haver um cadastro de usuários para acesso ao sistema, os quais irão atuar como operadores para a compra e venda de produtos.
 - Deve haver um cadastro de pessoas físicas e pessoas jurídicas, com os dados básicos de identificação, localização e contato, diferenciando-se apenas pelo uso de CPF ou CNPJ.
 - Deve haver um cadastro de produtos, contendo identificador, nome, quantidade e preço de venda.
 - Os operadores (usuários) poderão efetuar movimentos de compra para um determinado produto, sempre de uma pessoa jurídica, indicando a quantidade de produtos e preço unitário.
 - Os operadores (usuários) poderão efetuar movimentos de venda para um determinado produto, sempre para uma pessoa física, utilizando o preço de venda atualmente na base



"Descrição"

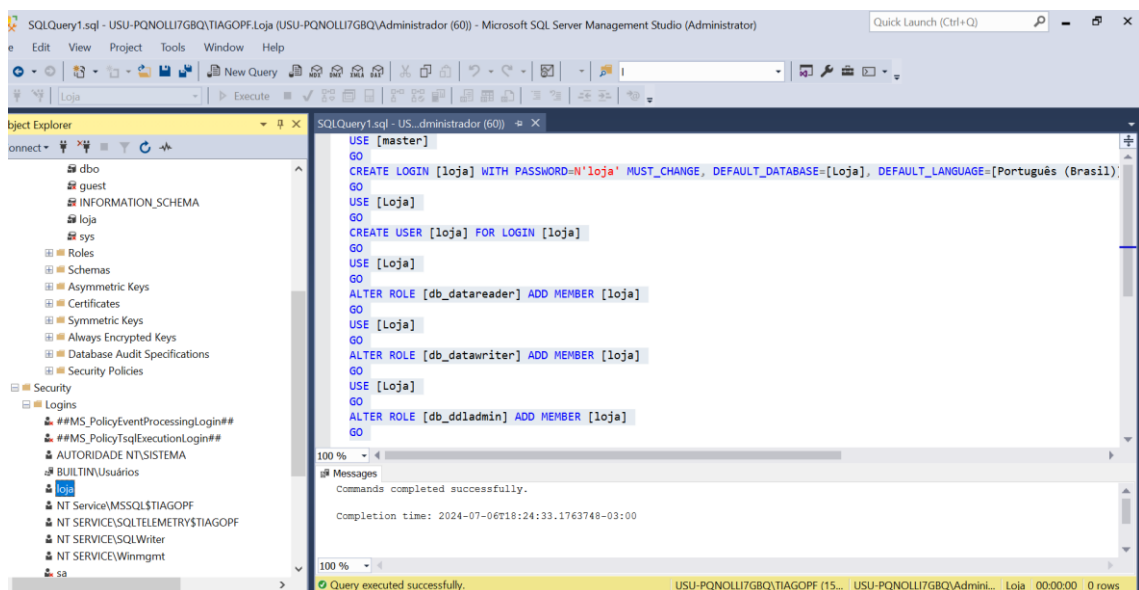
Modelo de dados de um sistema feito pelo DBDesigner-Fork

Criando e Configurando o Login no SQL



“Descrição”

Criando e configurando o Login através do painel do SQL Serve



“Descrição”

Criando e Configurando o Login através de comando SQL

Código completo de Criação do Banco de Dados juntamente com as Tabelas

```
CREATE DATABASE Loja;
GO

USE Loja;
GO

CREATE SEQUENCE ordemPessoaId
START WITH 1
INCREMENT BY 1;

CREATE TABLE Pessoa (
    idPessoa INTEGER NOT NULL CONSTRAINT PK_Pessoa PRIMARY KEY,
    nome VARCHAR(255) NOT NULL,
    logradouro VARCHAR(255),
    cidade VARCHAR(255),
    estado CHAR(2) NOT NULL,
    telefone VARCHAR(11),
    email VARCHAR(255)
);
GO

CREATE TABLE PessoaFisica (
    idPessoa INTEGER NOT NULL CONSTRAINT PK_PessoaFisica PRIMARY KEY,
    cpf VARCHAR(11) NOT NULL,
    CONSTRAINT FK_PessoaFisica_Pessoa FOREIGN KEY (idPessoa) REFERENCES Pessoa (idPessoa)
);
GO

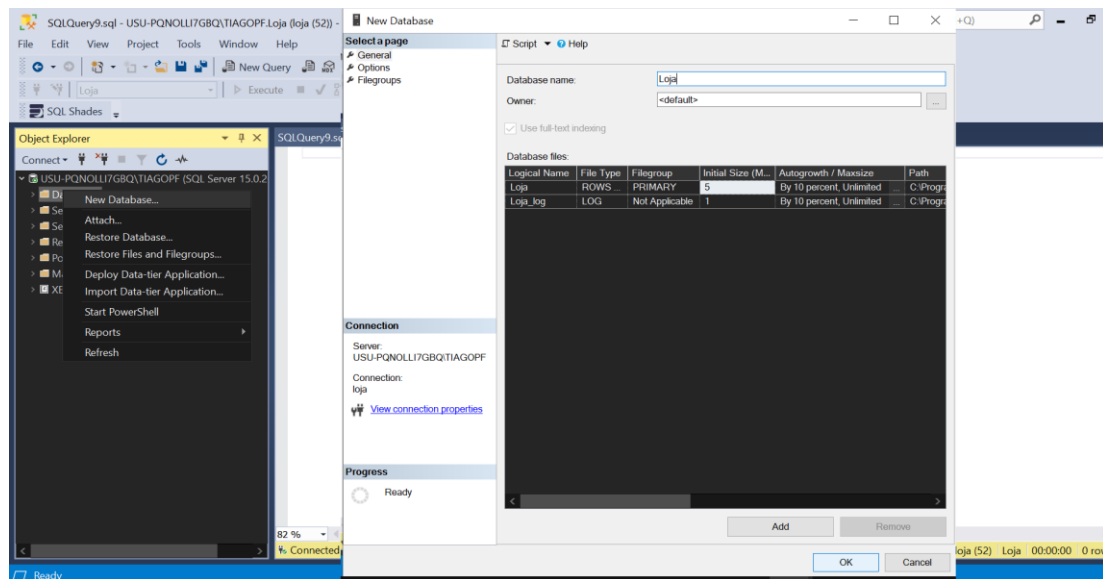
CREATE TABLE PessoaJuridica (
    idPessoa INTEGER NOT NULL CONSTRAINT PK_PessoaJuridica PRIMARY KEY,
    cnpj VARCHAR(14) NOT NULL,
    CONSTRAINT FK_PessoaJuridica_Pessoa FOREIGN KEY (idPessoa) REFERENCES Pessoa (idPessoa)
);
GO

CREATE TABLE Usuario (
    idUsuario INTEGER NOT NULL CONSTRAINT PK_Usuario PRIMARY KEY IDENTITY,
    login VARCHAR(20) NOT NULL,
    senha VARCHAR(20) NOT NULL
);
GO

CREATE TABLE Produto (
    idProduto INTEGER NOT NULL CONSTRAINT PK_Produto PRIMARY KEY,
    nome VARCHAR(255) NOT NULL,
    quantidade INTEGER,
    precoVenda NUMERIC(5, 2)
);
GO

CREATE TABLE Movimento (
    idMovimento INTEGER NOT NULL CONSTRAINT PK_Movimento PRIMARY KEY,
    idUsuario INTEGER NOT NULL,
    idPessoa INTEGER NOT NULL,
    idProduto INTEGER,
    quantidade INTEGER,
    tipo CHAR(1),
    valorUnitario NUMERIC(5, 2),
    CONSTRAINT FK_Movimento_Usuario FOREIGN KEY (idUsuario) REFERENCES Usuario (idUsuario),
    CONSTRAINT FK_Movimento_Pessoa FOREIGN KEY (idPessoa) REFERENCES Pessoa (idPessoa),
    CONSTRAINT FK_Movimento_Produto FOREIGN KEY (idProduto) REFERENCES Produto (idProduto)
);
GO
```

Criando o Banco de Dados com o Nome “Loja”



“Descrição”

Criando e Configurando o Banco de Dados através do painel do SQL

```
CREATE DATABASE [Loja]
CONTAINMENT = NONE
ON PRIMARY
( NAME = N'Loja', FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL15.TIAGOPF\MSSQL\DATA\Loja.mdf' , SIZE = 5120KB , FILEGROWTH = 10%)
LOG ON
( NAME = N'Loja_log', FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL15.TIAGOPF\MSSQL\DATA\Loja_log.ldf' , SIZE = 1024KB , FILEGROWTH = 10)
GO
ALTER DATABASE [Loja] SET ANSI_NULL_DEFAULT OFF
GO
ALTER DATABASE [Loja] SET ANSI_NULLS OFF
GO
ALTER DATABASE [Loja] SET ANSI_PADDING OFF
GO
ALTER DATABASE [Loja] SET ANSI_WARNINGS OFF
GO
ALTER DATABASE [Loja] SET ARITHABORT OFF
GO
ALTER DATABASE [Loja] SET AUTO_CLOSE OFF
GO
ALTER DATABASE [Loja] SET AUTO_SHRINK OFF
GO
ALTER DATABASE [Loja] SET AUTO_CREATE_STATISTICS ON(INCREMENTAL = OFF)
GO
ALTER DATABASE [Loja] SET AUTO_UPDATE_STATISTICS ON
GO
ALTER DATABASE [Loja] SET CURSOR_CLOSE_ON_COMMIT OFF
GO
ALTER DATABASE [Loja] SET CURSOR_DEFAULT GLOBAL
GO
ALTER DATABASE [Loja] SET CONCAT_NULL_YIELDS_NULL OFF
GO
ALTER DATABASE [Loja] SET NUMERIC_ROUNDABORT OFF
GO
ALTER DATABASE [Loja] SET QUOTED_IDENTIFIER OFF
GO
ALTER DATABASE [Loja] SET RECURSIVE_TRIGGERS OFF
GO
ALTER DATABASE [Loja] SET DISABLE_BROKER
GO
ALTER DATABASE [Loja] SET AUTO_UPDATE_STATISTICS_ASYNC OFF
GO
ALTER DATABASE [Loja] SET DATE_CORRELATION_OPTIMIZATION OFF
GO
ALTER DATABASE [Loja] SET PARAMETERIZATION SIMPLE
GO
ALTER DATABASE [Loja] SET READ_COMMITTED_SNAPSHOT OFF
GO
ALTER DATABASE [Loja] SET READ_WRITE
GO
ALTER DATABASE [Loja] SET RECOVERY SIMPLE
GO
ALTER DATABASE [Loja] SET MULTI_USER
GO
ALTER DATABASE [Loja] SET PAGE_VERIFY CHECKSUM
GO
ALTER DATABASE [Loja] SET TARGET_RECOVERY_TIME = 0 SECONDS
GO
ALTER DATABASE [Loja] SET DELAYED_DURABILITY = DISABLED
GO
USE [Loja]
GO
ALTER DATABASE SCOPED CONFIGURATION SET LEGACY_CARDINALITY_ESTIMATION = OFF;
GO
ALTER DATABASE SCOPED CONFIGURATION SET MAXDOP = 0;
GO
ALTER DATABASE SCOPED CONFIGURATION FOR SECONDARY SET MAXDOP = PRIMARY;
GO
ALTER DATABASE SCOPED CONFIGURATION SET PARAMETER_SNIFFING = On;
GO
ALTER DATABASE SCOPED CONFIGURATION FOR SECONDARY SET PARAMETER_SNIFFING = Primary;
GO
ALTER DATABASE SCOPED CONFIGURATION SET QUERY_OPTIMIZER_HOTFIXES = OFF;
GO
ALTER DATABASE SCOPED CONFIGURATION FOR SECONDARY SET QUERY_OPTIMIZER_HOTFIXES = Primary;
GO
USE [Loja]
GO
IF NOT EXISTS (SELECT name FROM sys.filegroups WHERE is_default=1 AND name = N'PRIMARY') ALTER DATABASE [Loja] MODIFY FILEGROUP [PRIMARY] DEFAULT
GO
```

“Descrição”

Código da Criação e Configuração do Banco de Dados Utilizando comandos SQL

4 - Os resultados da execução dos códigos também devem ser apresentados:

Criando a tabela Movimento

```
CREATE TABLE Movimento (  
    idMovimento INTEGER NOT NULL CONSTRAINT PK_Movimento PRIMARY KEY,  
    idUsuario INTEGER NOT NULL,  
    idPessoa INTEGER NOT NULL,  
    idProduto INTEGER,  
    quantidade INTEGER,  
    tipo CHAR(1),  
    valorUnitario NUMERIC (5, 2),  
    CONSTRAINT FK_Movimento_Usuario FOREIGN KEY (idUsuario) REFERENCES Usuario (idUsuario),  
    CONSTRAINT FK_Movimento_Pessoa FOREIGN KEY (idPessoa) REFERENCES Pessoa (idPessoa),  
    CONSTRAINT FK_Movimento_Produto FOREIGN KEY (idProduto) REFERENCES Produto (idProduto)  
);  
GO
```

Criando a Tabela Produto

```
CREATE TABLE Produto (  
    idProduto INTEGER NOT NULL CONSTRAINT PK_Produto PRIMARY KEY,  
    nome VARCHAR(255) NOT NULL,  
    quantidade INTEGER,  
    precoVenda NUMERIC(5, 2)  
);  
GO
```

Criando a Tabela Usuário

```
CREATE TABLE Usuario (  
    idUsuario INTEGER NOT NULL CONSTRAINT PK_Usuario PRIMARY KEY IDENTITY,  
    login VARCHAR(20) NOT NULL,  
    senha VARCHAR(20) NOT NULL  
);  
GO
```

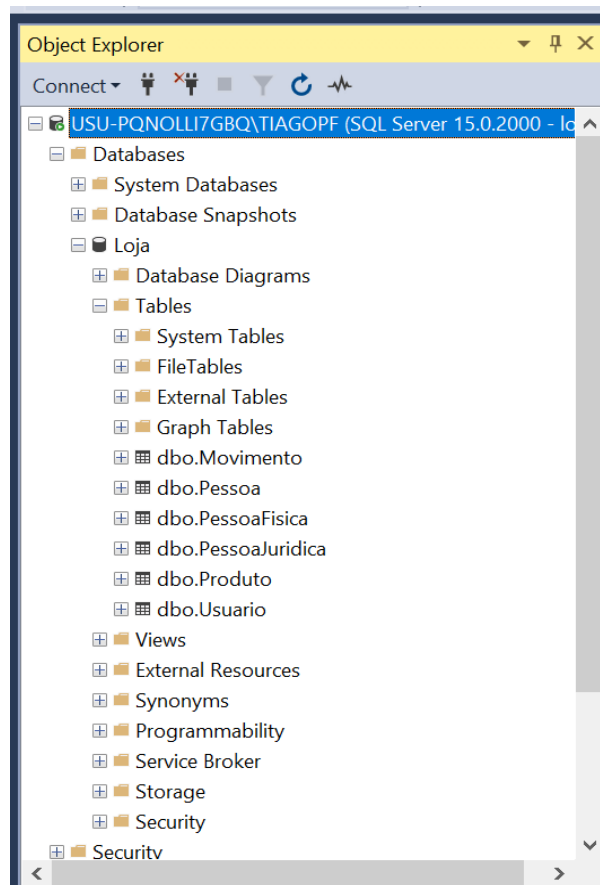
Criando a Tabela Pessoa Física / Jurídica

```
CREATE TABLE PessoaFisica (  
    idPessoa INTEGER NOT NULL CONSTRAINT PK_PessoaFisica PRIMARY KEY,  
    cpf VARCHAR(11) NOT NULL,  
    CONSTRAINT FK_PessoaFisica_Pessoa FOREIGN KEY (idPessoa) REFERENCES Pessoa (idPessoa)  
);  
GO  
  
CREATE TABLE PessoaJuridica (  
    idPessoa INTEGER NOT NULL CONSTRAINT PK_PessoaJuridica PRIMARY KEY,  
    cnpj VARCHAR(14) NOT NULL,  
    CONSTRAINT FK_PessoaJuridica_Pessoa FOREIGN KEY (idPessoa) REFERENCES Pessoa (idPessoa)  
);  
GO  
|
```

Criando a Tabela Pessoa

```
CREATE TABLE Pessoa (  
    idPessoa INTEGER NOT NULL CONSTRAINT PK_Pessoa PRIMARY KEY,  
    nome VARCHAR(255) NOT NULL,  
    logradouro VARCHAR(255),  
    cidade VARCHAR(255),  
    estado CHAR(2) NOT NULL,  
    telefone VARCHAR(11),  
    email VARCHAR(255)  
);  
GO
```

Resultado da Criação das Tabelas



5 – Análise e Conclusão:

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

Em um banco de dados relacional, as cardinalidades são implementadas através de chaves primárias e chaves estrangeiras. Aqui está uma explicação básica de como cada tipo é tratado:

❖ Cardinalidade 1:1 (Um para Um)

- ✓ **Implementação:** Ambas as tabelas têm uma chave primária que também atua como chave estrangeira. Por exemplo, se temos duas tabelas, Pessoa e Passaporte, cada Pessoa pode ter um único Passaporte, e vice-versa. A tabela Passaporte terá uma coluna que é uma chave estrangeira referenciando a chave primária de Pessoa.

❖ Cardinalidade 1

(Um para Muitos)

- ✓ **Implementação:** A tabela do lado "um" possui uma chave primária, enquanto a tabela do lado "muitos" tem uma chave estrangeira que aponta para a chave primária da tabela "um". Por exemplo, na relação entre Cliente e Pedido, um Cliente pode ter vários Pedidos, então a tabela Pedido incluirá uma coluna com a chave estrangeira que referência a chave primária da tabela Cliente.

❖ Cardinalidade N

(Muitos para Muitos)

- ✓ **Implementação:** Esta relação é implementada através de uma tabela intermediária (ou de junção), que contém chaves estrangeiras que referenciam as chaves primárias das duas tabelas envolvidas. Por exemplo, se temos Aluno e Curso, e um aluno pode estar em muitos cursos e um curso pode ter muitos alunos, criaríamos uma tabela AlunoCurso com duas colunas: uma para a chave estrangeira AlunoID e outra para CursoID. Essa tabela atuará como um vínculo entre Aluno e Curso.

❖ Resumo

- ✓ **1:1:** Chave estrangeira na tabela dependente.
- ✓ **1:N:** Chave estrangeira na tabela do lado "muitos".
- ✓ **N:M:** Tabela de junção com chaves estrangeiras de ambas as tabelas.

Essas implementações permitem garantir a integridade referencial e facilitam consultas complexas em bancos de dados relacionais.

❖ 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, você pode utilizar os seguintes tipos de relacionamento:

❖ Tabela Única

- ✓ **Descrição:** Uma única tabela armazena todos os atributos das classes base e derivadas. Um campo indica o tipo de classe.
- ✓ **Vantagens:** Simplicidade e consultas mais rápidas.
- ✓ **Desvantagens:** Pode resultar em muitos campos nulos.

❖ Tabela por Classe

- ✓ **Descrição:** Uma tabela para a classe base e tabelas separadas para cada subclasse, onde as subclasses contêm uma chave estrangeira referenciando a classe base.
- ✓ **Vantagens:** Estrutura clara e evita campos nulos.
- ✓ **Desvantagens:** Requer mais joins para consultas.

❖ Tabela de Junção

- ✓ **Descrição:** Uma tabela para a classe base e tabelas separadas para subclasses, com as subclasses armazenando os atributos específicos.
- ✓ **Vantagens:** Mantém a integridade e a estrutura sem campos nulos.
- ✓ **Desvantagens:** Pode ser complexa, devido a múltiplos joins.

❖ Resumo

- ✓ A escolha depende das necessidades do seu projeto. Se a simplicidade é priorizada, use a tabela única. Para maior clareza e estrutura, opte por tabelas por classe ou tabela de junção.

❖ 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) oferece várias funcionalidades que melhoram a produtividade nas tarefas de gerenciamento de bancos de dados. Aqui estão algumas das principais características:

❖ Interface Gráfica Intuitiva

- ✓ **Descrição:** A interface visual facilita a navegação entre objetos do banco de dados, como tabelas, views, procedimentos armazenados, etc., permitindo uma gestão mais rápida e eficiente.

❖ Editor de Consultas Avançado

- ✓ **Descrição:** O editor de consultas oferece recursos como destaque de sintaxe, autocompletar, e sugestões de código, o que ajuda a escrever consultas mais rapidamente e com menos erros.

❖ Intelli Sense

- ✓ **Descrição:** O Intelli Sense fornece sugestões automáticas enquanto você digita, incluindo nomes de tabelas, colunas e funções, agilizando o desenvolvimento de queries.

❖ Modelagem de Dados

- ✓ **Descrição:** Ferramentas de modelagem permitem criar e visualizar diagramas de banco de dados, facilitando a compreensão das relações entre tabelas.

❖ Gerenciamento de Tarefas e Scripts

- ✓ **Descrição:** Você pode criar, salvar e agendar scripts para tarefas rotineiras, como backups e manutenção de índices, melhorando a eficiência e a consistência.

❖ **Relatórios e Monitoramento**

- ✓ **Descrição:** O SSMS permite monitorar o desempenho do servidor e gerar relatórios sobre o uso do banco de dados, ajudando na identificação de gargalos e na otimização de recursos.

❖ **Integração com SQL Server Profiler**

- ✓ **Descrição:** Ferramentas como o Profiler ajudam a rastrear e analisar a performance de consultas, permitindo otimizações baseadas em dados reais.

❖ **Suporte a Extensões e Add-ins**

- ✓ **Descrição:** O SSMS suporta extensões que podem adicionar funcionalidades específicas, adaptando o ambiente às necessidades do usuário.

❖ **Execução de Consultas em Batch**

- ✓ **Descrição:** Permite executar múltiplas instruções SQL em um único script, economizando tempo e facilitando operações complexas.

❖ **Segurança e Gerenciamento de Usuários**

- ✓ **Descrição:** Ferramentas para gerenciamento de permissões e usuários facilitam a administração da segurança do banco de dados.

Essas funcionalidades tornam o SSMS uma ferramenta poderosa para administradores de banco de dados e desenvolvedores, aumentando a eficiência e reduzindo o tempo necessário para realizar tarefas administrativas e de desenvolvimento.

2º Procedimento | Alimentando a Base

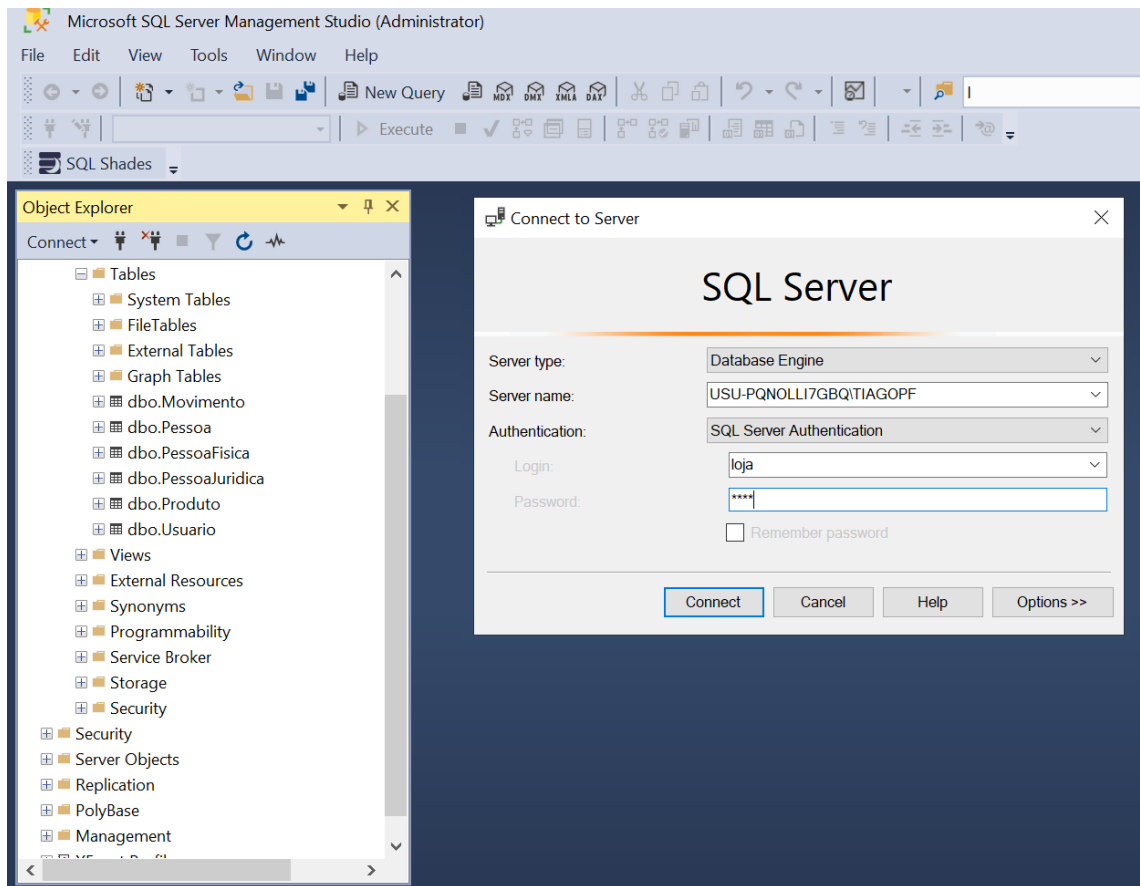
Objetivo da Prática:

- 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.

Todos os códigos solicitados neste roteiro de aula:

1- Utilizar o SQL Server Management Studio para alimentar as tabelas com dados básicos do sistema:

a. Logar. como usuário **loja**, senha **loja**.



“Descrição”

Logando utilizando o usuário loja e a senha loja
Que foram criados anteriormente

Utilizar o editor de SQL para incluir dados na tabela de usuários, de forma a obter um conjunto como o apresentado a seguir:

Inserindo login e senha na tabela Usuários:

```
USE Loja;  
  
INSERT INTO Usuario (login, senha)  
VALUES ('op1', 'op1'),  
('op2', 'op2'),  
('op3', 'op3'),  
('op4', 'op4');
```

“Descrição”

Inserindo através do Editor SQL alguns logins e senhas na tabela usuário

Resultado:

SQLQuery1.sql - USU...PF.Loja (loja (53))*

```
SELECT TOP (3) [idUsuario]  
              , [login]  
              , [senha]  
FROM [Loja].[dbo].[Usuario]
```

81 %

Results Messages

	idUsuario	login	senha
1	1	op1	op1
2	2	op2	op2
3	3	op3	op3

“Descrição”

Resultado da inclusão dos login e senha na tabela usuário

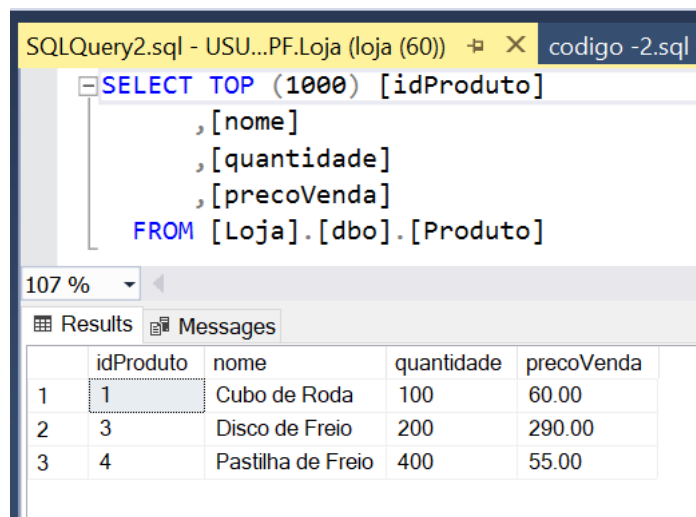
Inserindo Produtos na tabela Produtos:

```
INSERT INTO Produto (idProduto, nome, quantidade, precoVenda)
VALUES ('1', 'Cubo de Roda', '100', '60.00'),
('3', 'Disco de Freio', '200', '290.00'),
('4', 'Pastilha de Freio', '400', '55.00');
```

“Descrição”

Incluindo através do Editor SQL alguns Produtos na tabela Produto

Resultado:



The screenshot shows a SQL query window titled 'SQLQuery2.sql - USU...PF.Loja (loja (60))' with a query that selects the top 1000 records from the 'Produto' table. Below the query, the 'Results' tab is active, displaying a table with 5 columns: 'idProduto', 'nome', 'quantidade', and 'precoVenda'. The table contains 3 rows of data.

	idProduto	nome	quantidade	precoVenda
1	1	Cubo de Roda	100	60.00
2	3	Disco de Freio	200	290.00
3	4	Pastilha de Freio	400	55.00

“Descrição”

Mostrando o Resultado da inclusão de alguns produtos na tabela Produto

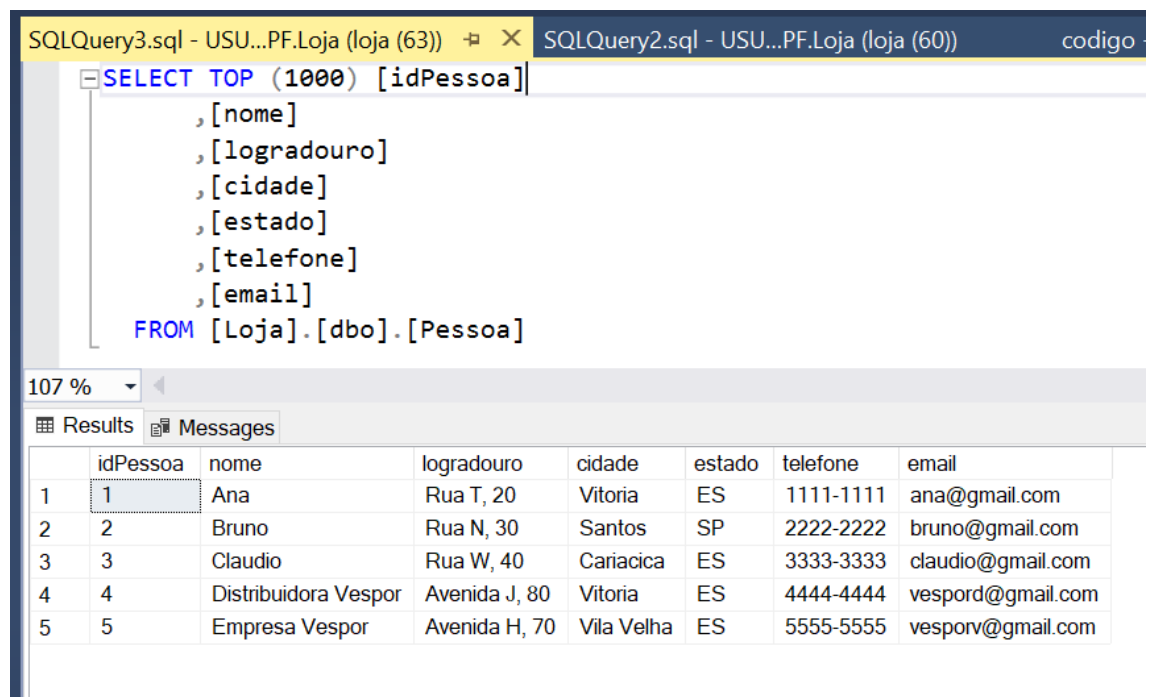
Inserindo Pessoas na Tabela Pessoa:

```
INSERT INTO Pessoa (idPessoa, nome, logradouro, cidade, estado, telefone, email)
VALUES
(NEXT VALUE FOR ordemPessoaId, 'Ana', 'Rua T, 20', 'Vitoria', 'ES', '1111-1111', 'ana@gmail.com'),
(NEXT VALUE FOR ordemPessoaId, 'Bruno', 'Rua N, 30', 'Santos', 'SP', '2222-2222', 'bruno@gmail.com'),
(NEXT VALUE FOR ordemPessoaId, 'Claudio', 'Rua W, 40', 'Cariacica', 'ES', '3333-3333', 'claudio@gmail.com'),
(NEXT VALUE FOR ordemPessoaId, 'Distribuidora Vespor', 'Avenida J, 80', 'Vitoria', 'ES', '4444-4444', 'vespord@gmail.com'),
(NEXT VALUE FOR ordemPessoaId, 'Empresa Vespor', 'Avenida H, 70', 'Vila Velha', 'ES', '5555-5555', 'vesporv@gmail.com');
```

“Descrição”

Inserindo algumas pessoas pôr ordem na Tabela Pessoa.

Resultado:



The screenshot shows a SQL Server Enterprise Manager interface. At the top, there are two tabs: 'SQLQuery3.sql - USU...PF.Loja (loja (63))' and 'SQLQuery2.sql - USU...PF.Loja (loja (60))'. The active tab is 'SQLQuery3.sql'. Below the tabs, the SQL query is displayed: `SELECT TOP (1000) [idPessoa]`. Below the query, the 'Results' tab is selected, showing a table with 5 rows and 7 columns: idPessoa, nome, logradouro, cidade, estado, telefone, and email. The data is as follows:

	idPessoa	nome	logradouro	cidade	estado	telefone	email
1	1	Ana	Rua T, 20	Vitoria	ES	1111-1111	ana@gmail.com
2	2	Bruno	Rua N, 30	Santos	SP	2222-2222	bruno@gmail.com
3	3	Claudio	Rua W, 40	Cariacica	ES	3333-3333	claudio@gmail.com
4	4	Distribuidora Vespor	Avenida J, 80	Vitoria	ES	4444-4444	vespord@gmail.com
5	5	Empresa Vespor	Avenida H, 70	Vila Velha	ES	5555-5555	vesporv@gmail.com

“Descrição”

Mostrando Resultado da inclusão de algumas pessoas na Tabela Pessoa
Repare que estão em ordem de acordo com os dados inseridos anteriormente
Nota que Ana está em primeiro.

Inserindo Pessoa Física e Pessoa Jurídica:

```
INSERT INTO PessoaFisica (idPessoa, cpf)
VALUES (1, '11111111111'),
(2, '22222222222'),
(3, '33333333333');

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

“Descrição”

Inserindo CPF em pessoa Física e CNPJ em Pessoa Jurídica

Resultado:

codigo -2.sql - USU...OPF.Loja (loja (61)) codigo -3.sql - USU...OPF.Loja (loja (53))

```
SELECT *
FROM PessoaFisica
INNER JOIN Pessoa ON PessoaFisica.idPessoa = Pessoa.idPessoa

SELECT *
FROM PessoaJuridica
INNER JOIN Pessoa ON PessoaJuridica.idPessoa = Pessoa.idPessoa
```

107 %

Results Messages

	idPessoa	cpf	idPessoa	nome	logradouro	cidade	estado	telefone	email
1	1	11111111111	1	Ana	Rua T, 20	Vitoria	ES	1111-1111	ana@gmail.com
2	2	22222222222	2	Bruno	Rua N, 30	Santos	SP	2222-2222	bruno@gmail.com
3	3	33333333333	3	Claudio	Rua W, 40	Cariacica	ES	3333-3333	claudio@gmail.com

	idPessoa	cnpj	idPessoa	nome	logradouro	cidade	estado	telefone	email
1	4	44444444444444	4	Distribuidora Vespor	Avenida J, 80	Vitoria	ES	4444-4444	vespord@gmail.com
2	5	55555555555555	5	Empresa Vespor	Avenida H, 70	Vila Velha	ES	5555-5555	vesporv@gmail.com

“Descrição”

Mostrando o Resultado da inclusão de CPF para pessoa Física e CNPJ para Pessoa Jurídica.

Inserindo dados na Tabela Movimento:

```
INSERT INTO Movimento (idMovimento, idUsuario, idPessoa, idProduto, quantidade, tipo, valorUnitario)
VALUES (1, 1, 5, 1, 40, 'E', 40.00),
(3, 2, 3, 3, 20, 'S', 30.00),
(5, 1, 4, 4, 60, 'E', 55.00),
(6, 2, 1, 1, 15, 'S', 40.00),
(7, 4, 2, 4, 25, 'S', 35.00),
(9, 3, 5, 3, 50, 'E', 45.00);
```

“Descrição”

Inserindo dados de movimentações dentro da Tabela Movimento

Resultado:

SQLQuery4.sql - USU...PF.Loja (loja (54)) X codigo -2.sql - USU...OPF.Loja (loja (61))

```
SELECT TOP (1000) [idMovimento]
,[idUsuario]
,[idPessoa]
,[idProduto]
,[quantidade]
,[tipo]
,[valorUnitario]
FROM [Loja].[dbo].[Movimento]
```

107 %

Results Messages

	idMovimento	idUsuario	idPessoa	idProduto	quantidade	tipo	valorUnitario
1	1	1	5	1	40	E	40.00
2	3	2	3	3	20	S	30.00
3	5	1	4	4	60	E	55.00
4	6	2	1	1	15	S	40.00
5	7	4	2	4	25	S	35.00
6	9	3	5	3	50	E	45.00

“Descrição”

Mostrando o Resultado da Inclusão dos dados dentro da Tabela Movimento

4. Efetuar as seguintes consultas sobre os dados inseridos:

Dados completos de pessoas físicas:

codigo -3.sql - USU...OPF.Loja (loja (54))

```
SELECT *  
FROM PessoaFisica  
INNER JOIN Pessoa ON PessoaFisica.idPessoa = Pessoa.idPessoa
```

107 %

Results Messages

	idPessoa	cpf	idPessoa	nome	logradouro	cidade	estado	telefone	email
1	1	11111111111	1	Ana	Rua T, 20	Vitoria	ES	1111-1111	ana@gmail.com
2	2	22222222222	2	Bruno	Rua N, 30	Santos	SP	2222-2222	bruno@gmail.com
3	3	33333333333	3	Claudio	Rua W, 40	Cariacica	ES	3333-3333	claudio@gmail.com

Dados completos de pessoas jurídicas:

SQLQuery1.sql - USU...PF.Loja (loja (53))* codigo -3.sql - USU...OPF.Loja (loja (54))

```
SELECT *  
FROM PessoaJuridica  
INNER JOIN Pessoa ON PessoaJuridica.idPessoa = Pessoa.idPessoa
```

107 %

Results Messages

	idPessoa	cnpj	idPessoa	nome	logradouro	cidade	estado	telefone	email
1	4	4444444444444	4	Distribuidora Vespord	Avenida J, 80	Vitoria	ES	4444-4444	vespord@gmail.com
2	5	5555555555555	5	Empresa Vespord	Avenida H, 70	Vila Velha	ES	5555-5555	vespordv@gmail.com

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

codigo -3.sql - USU...OPF.Loja (loja (54))

```
SELECT  
Produto.nome AS 'produto', Pessoa.nome AS 'fornecedor',  
Movimento.quantidade, Movimento.valorUnitario,  
Movimento.quantidade * Movimento.valorUnitario AS 'valorTotal'  
FROM Movimento  
INNER JOIN Produto ON Movimento.idProduto = Produto.idProduto  
INNER JOIN Pessoa ON Movimento.idPessoa = Pessoa.idPessoa  
WHERE Movimento.tipo = 'E';
```

107 %

Results Messages

	produto	fornecedor	quantidade	valorUnitario	valorTotal
1	Cubo de Roda	Empresa Vespord	40	40.00	1600.00
2	Pastilha de Freio	Distribuidora Vespord	60	55.00	3300.00
3	Disco de Freio	Empresa Vespord	50	45.00	2250.00

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

codigo -3.sql - USU...OPF.Loja (loja (54))

```
SELECT
    Produto.nome AS 'produto', Pessoa.nome AS 'comprador',
    Movimento.quantidade, Movimento.valorUnitario,
    Movimento.quantidade * Movimento.valorUnitario AS 'valorTotal'
FROM Movimento
INNER JOIN Produto ON Movimento.idProduto = Produto.idProduto
INNER JOIN Pessoa ON Movimento.idPessoa = Pessoa.idPessoa
WHERE Movimento.tipo = 'S';
```

107 %

Results Messages

	produto	comprador	quantidade	valorUnitario	valorTotal
1	Disco de Freio	Claudio	20	30.00	600.00
2	Cubo de Roda	Ana	15	40.00	600.00
3	Pastilha de Freio	Bruno	25	35.00	875.00

Valor total das entradas agrupadas por produto:

codigo -3.sql - USU...OPF.Loja (loja (54))

```
SELECT
    Produto.nome AS 'produto',
    SUM(Movimento.quantidade * Movimento.valorUnitario) AS 'valorTotalEntrada'
FROM Movimento
JOIN Produto ON Movimento.idProduto = Produto.idProduto
WHERE Movimento.tipo = 'E'
GROUP BY Produto.nome;
```

107 %

Results Messages

	produto	valorTotalEntrada
1	Cubo de Roda	1600.00
2	Disco de Freio	2250.00
3	Pastilha de Freio	3300.00

Valor total das saídas agrupadas por produto:

codigo -3.sql - USU...OPF.Loja (loja (54))

```
SELECT
    Produto.nome AS 'produto',
    SUM(Movimento.quantidade * Movimento.valorUnitario) AS 'valorTotalSaida'
FROM Movimento
JOIN Produto ON Movimento.idProduto = Produto.idProduto
WHERE Movimento.tipo = 'S'
GROUP BY Produto.nome;
```

107 %

Results Messages

	produto	valorTotalSaida
1	Cubo de Roda	600.00
2	Disco de Freio	600.00
3	Pastilha de Freio	875.00

Operadores que não efetuaram movimentações de entrada (compra):

codigo -3.sql - USU...OPF.Loja (loja (54))

```
SELECT DISTINCT
  Usuario.idUsuario, Usuario.login, Movimento.idMovimento
FROM Usuario
LEFT JOIN Movimento ON Usuario.idUsuario = Movimento.idUsuario AND Movimento.tipo = 'E'
WHERE idMovimento IS NULL;
```

107 %

Results Messages

	idUsuario	login	idMovimento
1	2	op2	NULL
2	4	op4	NULL

Valor total de entrada, agrupado por operador:

codigo -3.sql - USU...OPF.Loja (loja (54))

```
SELECT
  Usuario.login AS 'operador',
  SUM(Movimento.quantidade * Movimento.valorUnitario) AS 'valorTotalEntrada'
FROM Movimento
JOIN Usuario ON Movimento.idUsuario = Usuario.idUsuario
WHERE Movimento.tipo = 'E'
GROUP BY Usuario.login;
```

107 %

Results Messages

	operador	valorTotalEntrada
1	op1	4900.00
2	op3	2250.00

Valor total de saída, agrupado por operador:

codigo -3.sql - USU...OPF.Loja (loja (54))

```
SELECT
  Usuario.login AS 'operador',
  SUM(Movimento.quantidade * Movimento.valorUnitario) AS 'valorTotalSaida'
FROM Movimento
JOIN Usuario ON Movimento.idUsuario = Usuario.idUsuario
WHERE Movimento.tipo = 'S'
GROUP BY Usuario.login;
```

107 %

Results Messages

	operador	valorTotalSaida
1	op2	1200.00
2	op4	875.00

Valor médio de venda por produto, utilizando média ponderada:

codigo -3.sql - USU...OPF.Loja (loja (54))

```
SELECT
  SUM(Movimento.valorUnitario * Movimento.quantidade) / SUM(Movimento.quantidade) AS 'médiaPonderada'
FROM Movimento
WHERE Movimento.tipo = 'S';
```

107 %

Results Messages

	médiaPonderada
1	34.583333

5 - Análise e Conclusão:

❖ Quais as diferenças no uso de sequence e identity?

❖ SEQUENCE

- ✓ **Objeto Independente:** Criado separadamente da tabela.
- ✓ **Uso Explícito:** Requer chamadas explícitas para obter o próximo valor (NEXTVAL).
- ✓ **Flexibilidade:** Pode ser compartilhado entre múltiplas tabelas e personalizado (valor inicial, incremento, etc.).
- ✓ **Exemplo:**

```
CREATE SEQUENCE my_sequence START WITH 1 INCREMENT BY 1;  
INSERT INTO my_table (id, name) VALUES (NEXTVAL('my_sequence'), 'John Doe');
```

❖ IDENTITY

- ✓ **Parte da Tabela:** Definido diretamente na coluna da tabela.
- ✓ **Uso Automático:** Valores gerados automaticamente ao inserir registros.
- ✓ **Simplicidade:** Fácil de usar e configurar, específico para cada tabela.
- ✓ **Exemplo:**

```
sql Copiar código  
  
CREATE TABLE my_table (  
    id INT IDENTITY(1,1) PRIMARY KEY,  
    name VARCHAR(50)  
);  
INSERT INTO my_table (name) VALUES ('John Doe');
```

❖ Resumo Prático

- ✓ **SEQUENCE:** Ideal para quando você precisa de flexibilidade e compartilhamento de sequências entre tabelas.
- ✓ **IDENTITY:** Ideal para simplicidade e uso direto na tabela sem necessidade de gerenciamento adicional.

❖ Qual a importância das chaves estrangeiras para a consistência do banco?

As chaves estrangeiras são importantes para a consistência do banco de dados por várias razões:

- ✓ **Integridade Referencial:** Garantem que os relacionamentos entre tabelas sejam válidos, impedindo referências a registros inexistentes.
 - ✓ **Prevenção de Dados Órfãos:** Evitam a existência de registros em tabelas filhas que não têm correspondência na tabela pai.
 - ✓ **Manutenção de Coerência dos Dados:** Permitem ações de cascata (ON DELETE CASCADE, ON UPDATE CASCADE), que propagam automaticamente alterações entre tabelas relacionadas.
 - ✓ **Validação Automática:** O banco de dados valida automaticamente inserções e atualizações, garantindo que as referências sejam consistentes.
 - ✓ **Melhoria da Qualidade dos Dados:** Reduzem inconsistências e duplicidade de informações, mantendo a qualidade dos dados.
-
- ❖ **Resumo**
 - ✓ as chaves estrangeiras asseguram que os dados entre tabelas relacionadas sejam consistentes e válidos, preservando a integridade do banco de dados.

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

❖ Álgebra Relacional

Os operadores de álgebra relacional são usados para manipular e consultar dados em bancos de dados relacionais. Aqui estão os principais operadores:

✓ Seleção (Selection): $\sigma_{\text{condition}}(R)$

- Seleciona tuplas que satisfazem uma condição específica.
- Exemplo em SQL: **SELECT * FROM R WHERE condition;**

✓ Projeção (Projection): $\pi_{\text{attributes}}(R)$

- Seleciona determinadas colunas de uma tabela.
- Exemplo em SQL: **SELECT attribute1, attribute2 FROM R;**

✓ União (Union): $R \cup S$

- Combina os resultados de duas tabelas, removendo duplicatas.
- Exemplo em SQL: **SELECT * FROM R UNION SELECT * FROM S;**

✓ Interseção (Intersection): $R \cap S$

- Seleciona tuplas que são comuns a ambas as tabelas.
- Exemplo em SQL: **SELECT * FROM R INTERSECT SELECT * FROM S;**

✓ Diferença (Difference): $R - S$

- Seleciona tuplas que estão em uma tabela, mas não na outra.
- Exemplo em SQL: **SELECT * FROM R EXCEPT SELECT * FROM S;**

✓ Produto Cartesiano (Cartesian Product): $R \times S$

- Combina todas as tuplas de duas tabelas.
- Exemplo em SQL: **SELECT * FROM R, S;**

✓ Junção (Join): $R \bowtie S$

- Combina tuplas de duas tabelas com base em uma condição.
- Exemplo em SQL: **SELECT * FROM R JOIN S ON R.common_field = S.common_field;**

❖ Cálculo Relacional

O cálculo relacional é uma linguagem declarativa que expressa consultas em termos de lógica de predicados. Existem duas formas principais de cálculo relacional:

✓ Cálculo Relacional de Tuplas (Tuple Relational Calculus - TRC):

- Especifica as tuplas desejadas sem indicar explicitamente os passos para obtê-las.
- Exemplo: $\{t \mid t \in R \wedge t.A = \text{'value'}\}$
- Equivalente em SQL: `SELECT * FROM R WHERE A = 'value';`

✓ Cálculo Relacional de Domínios (Domain Relational Calculus - DRC):

- Semelhante ao TRC, mas trabalha com domínios de atributos em vez de tuplas inteiras.
- Exemplo: $\{ \langle x, y \rangle \mid \exists t (t \in R \wedge t.A = x \wedge t.B = y) \}$
- Equivalente em SQL: `SELECT A, B FROM R;`

❖ Resumo

- ✓ **Álgebra Relacional:** Inclui operadores como seleção, projeção, união, interseção, diferença, produto cartesiano e junção.
- ✓ **Cálculo Relacional:** Expressa consultas em termos de lógica de predicados, incluindo cálculos de tuplas e de domínios.

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

O agrupamento em consultas SQL é feito utilizando a cláusula GROUP BY, que agrupa linhas que têm valores iguais em colunas específicas em uma única linha de resumo. Isso é frequentemente usado em combinação com funções de agregação, como SUM, AVG, COUNT, MIN, e MAX, para realizar cálculos em grupos de dados.

❖ Como é Feito o Agrupamento

✓ Uso da Cláusula GROUP BY:

- A cláusula GROUP BY vem após a cláusula WHERE e antes da cláusula ORDER BY.
- Agrupa as linhas que têm os mesmos valores nas colunas especificadas.

✓ Funções de Agregação:

- As funções de agregação realizam cálculos em grupos de linhas.
- Exemplos de funções de agregação incluem: SUM, AVG, COUNT, MIN, MAX.

❖ Exemplo Prático

Considere uma tabela sales com as colunas product_id, quantity, e price. Para calcular a quantidade total vendida e a receita total por produto, você pode usar a seguinte consulta:

```
sql Copiar código  
  
SELECT product_id, SUM(quantity) AS total_quantity, SUM(price * quantity) AS total_revenue  
FROM sales  
GROUP BY product_id;
```

❖ Requisito Obrigatório

✓ Colunas no SELECT:

- ✓ Todas as colunas listadas na cláusula SELECT que não são argumentos de funções de agregação devem estar presentes na cláusula GROUP BY.
- ✓ Isso é necessário porque a cláusula GROUP BY agrupa linhas com base nas colunas especificadas, e todas as outras colunas devem ser agregadas ou agrupadas.

❖ Exemplos de Requisitos

1. Correto:

```
sql Copiar código  
  
SELECT department, AVG(salary)  
FROM employees  
GROUP BY department;
```

2. Incorreto (causaria um erro):

```
sql Copiar código  
  
SELECT department, salary  
FROM employees  
GROUP BY department;
```

⬇

- ✓ Aqui, a coluna salary não está agregada nem incluída no GROUP BY, resultando em um erro.

❖ Importância do GROUP BY

- ✓ **Sumarização de Dados:** Permite calcular totais, médias e outras estatísticas para grupos de dados.
- ✓ **Organização:** Facilita a análise de dados agrupando informações relacionadas.
- ✓ **Eficiência:** Permite que grandes conjuntos de dados sejam reduzidos a informações sumarizadas úteis para relatórios e análises.

Em resumo, o agrupamento em consultas SQL é realizado usando a cláusula GROUP BY junto com funções de agregação. O requisito obrigatório é que todas as colunas no SELECT que não são usadas em funções de agregação devem estar incluídas no GROUP BY.