Campus: POLO VILA DOS REMÉDIOS – OSASCO – SP
Curso: Desenvolvimento Full Stack
Nome Disciplina: Nível 2: Vamos Manter as Informações?
Matrícula: 2023 0397 9797
Semestre Letivo: 3º semestre
Nome: Anderson Barbosa Almeida
Repositório no GIT:
https://github.com/andydevbarbosa/andydevbarbosa/tree/main/RPG0015%20-%20Vamos%20manter%20as%20informa%C3%A7%C3%B5es!

Título da Prática: 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 nesse roteiro:

```
-- Tabela de pessoa
CREATE TABLE Pessoa (
    id INT PRIMARY KEY,
    nome VARCHAR(100),
    tipo CHAR(1) CHECK (tipo IN ('F', 'J')), -- 'F' para pessoa física, 'J' para
pessoa jurídica
    cpf VARCHAR(14), -- Apenas para pessoas físicas
    cnpj VARCHAR(18), -- Apenas para pessoas jurídicas
    endereco VARCHAR(255),
    telefone VARCHAR(20)
);

-- Tabela de usuários
CREATE TABLE Usuarios (
    id INT PRIMARY KEY IDENTITY(1,1),
    nome VARCHAR(100),
    senha VARCHAR(50)
);

-- Tabela de pessoas físicas
CREATE TABLE PessoasFisicas (
    id INT PRIMARY KEY,
```

```
    nome VARCHAR(100),
    cpf VARCHAR(14),
    endereco VARCHAR(255),
    telefone VARCHAR(20)
);

-- Tabela de pessoas jurídicas
CREATE TABLE PessoasJuridicas (
    id INT PRIMARY KEY,
    nomeFantasia VARCHAR(100),
    razaoSocial VARCHAR(100),
    cnpj VARCHAR(18),
    endereco VARCHAR(255),
    telefone VARCHAR(20)
);

-- Tabela de produtos
CREATE TABLE Produtos (
    id INT PRIMARY KEY IDENTITY(1,1),
    nome VARCHAR(100),
    quantidade INT,
    preco DECIMAL(10, 2)
);

-- Tabela de movimentos de compra
CREATE TABLE Compras (
    id INT PRIMARY KEY IDENTITY(1,1),
    idProduto INT,
    idFornecedor INT, -- Referência para PessoasJuridicas
    quantidade INT,
    precoUnitario DECIMAL(10, 2),
    FOREIGN KEY (idProduto) REFERENCES Produtos(id),
    FOREIGN KEY (idFornecedor) REFERENCES PessoasJuridicas(id)
);

-- Tabela de movimentos de venda
CREATE TABLE Vendas (
    id INT PRIMARY KEY IDENTITY(1,1),
    idProduto INT,
    idCliente INT, -- Referência para PessoasFisicas
    quantidade INT,
    precoUnitario DECIMAL(10, 2),
    FOREIGN KEY (idProduto) REFERENCES Produtos(id),
    FOREIGN KEY (idCliente) REFERENCES PessoasFisicas(id)
);

-- Sequência para geração de identificadores de pessoa
CREATE SEQUENCE SeqPessoa START WITH 1 INCREMENT BY 1;
```

```
-- Inserir movimentações de entrada (compra) na tabela Compras
INSERT INTO Compras (idProduto, quantidade, precoUnitario)
VALUES
    (1, 100, 5.00), -- Exemplo: Compra de 100 unidades do produto com id 1, a um preço unitário de 5.00
    (2, 50, 5.00);  -- Exemplo: Compra de 50 unidades do produto com id 2, a um preço unitário de 5.00

-- Inserir movimentações de saída (venda) na tabela Vendas
INSERT INTO Vendas (idProduto, quantidade, precoUnitario)
VALUES
    (1, 80, 5.00), -- Exemplo: Venda de 80 unidades do produto com id 1, a um preço unitário de 5.00
    (2, 40, 2.00); -- Exemplo: Venda de 40 unidades do produto com id 2, a um preço unitário de 2.00

SELECT *
FROM Pessoa
WHERE tipo = 'F';

SELECT *
FROM Pessoa
WHERE tipo = 'J';

SELECT c.idCompra, p.nome AS Produto, f.nome AS Fornecedor,
c.quantidade, c.precoUnitario, c.quantidade * c.precoUnitario AS ValorTotal
FROM Compras c
JOIN Produtos p ON c.idProduto = p.idProduto
JOIN Fornecedores f ON c.idFornecedor = f.idFornecedor;

SELECT v.idVenda, p.nome AS Produto, c.nome AS Comprador, v.quantidade,
v.precoUnitario, v.quantidade * v.precoUnitario AS ValorTotal
FROM Vendas v
JOIN Produtos p ON v.idProduto = p.idProduto
JOIN Clientes c ON v.idCliente = c.idCliente;

SELECT p.nome AS Produto, SUM(c.quantidade * c.precoUnitario) AS
ValorTotalEntradas
FROM Compras c
JOIN Produtos p ON c.idProduto = p.idProduto
GROUP BY p.nome;

SELECT p.nome AS Produto, SUM(v.quantidade * v.precoUnitario) AS
ValorTotalSaidas
FROM Vendas v
JOIN Produtos p ON v.idProduto = p.idProduto
GROUP BY p.nome;
SELECT u.nome AS Operador
FROM Usuarios u
```

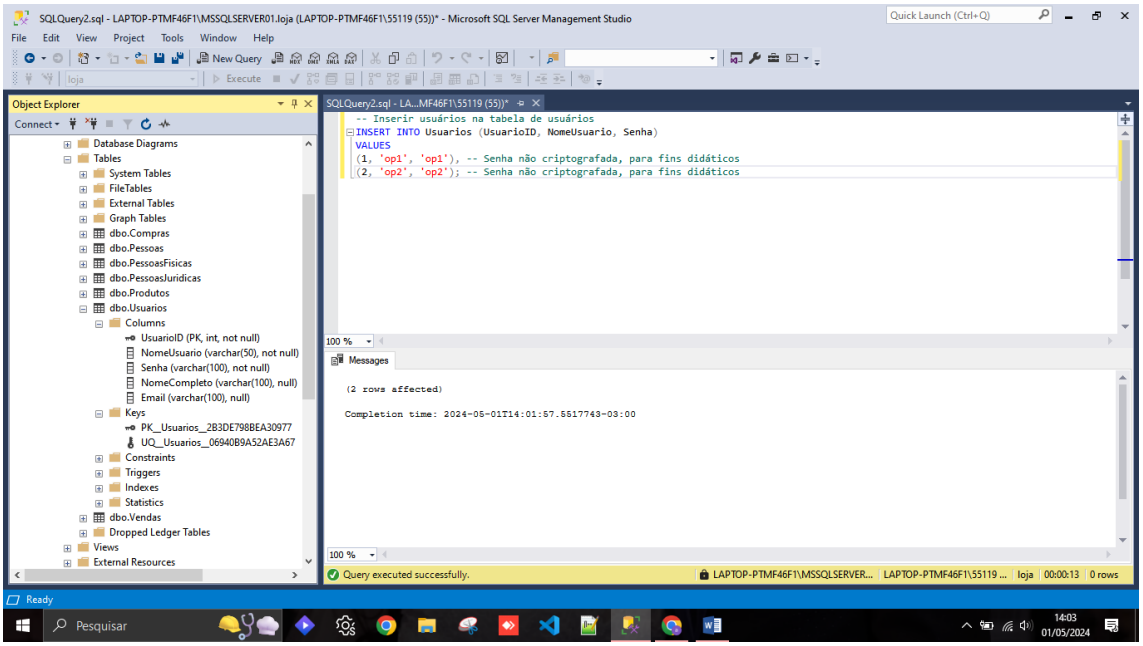
LEFT JOIN Compras c ON u.idUsuario = c.idUsuario
WHERE c.idCompra IS NULL;

SELECT u.nome AS Operador, SUM(c.quantidade * c.precoUnitario) AS ValorTotalEntradas
FROM Usuarios u
JOIN Compras c ON u.idUsuario = c.idUsuario
GROUP BY u.nome;

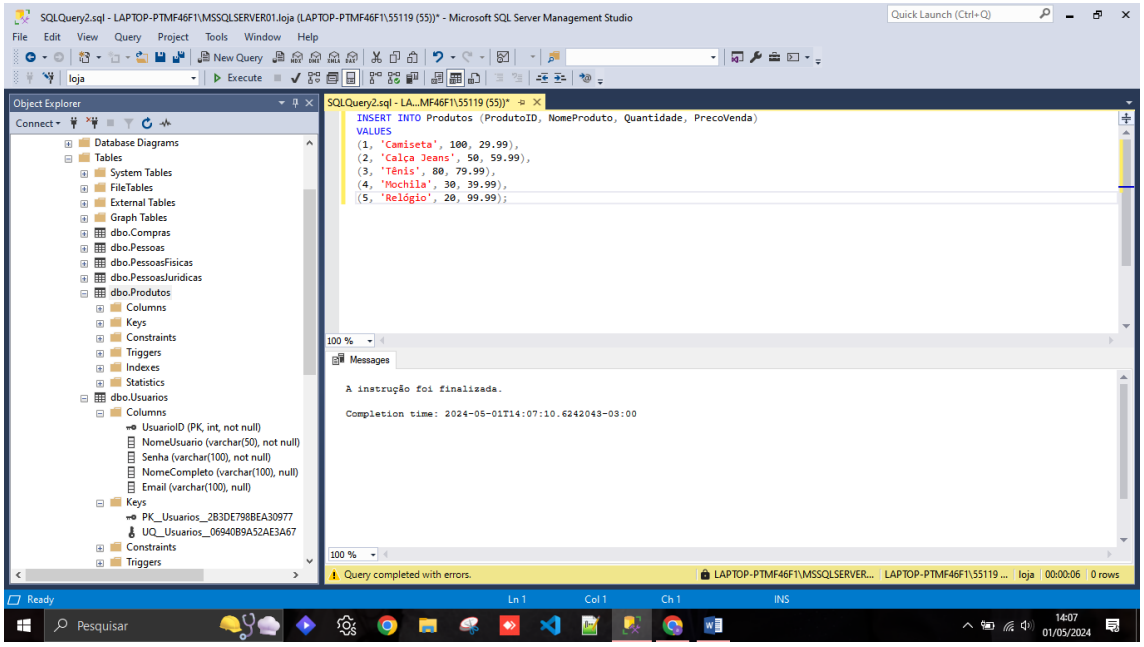
SELECT u.nome AS Operador, SUM(v.quantidade * v.precoUnitario) AS ValorTotalSaidas
FROM Usuarios u
JOIN Vendas v ON u.idUsuario = v.idUsuario
GROUP BY u.nome;

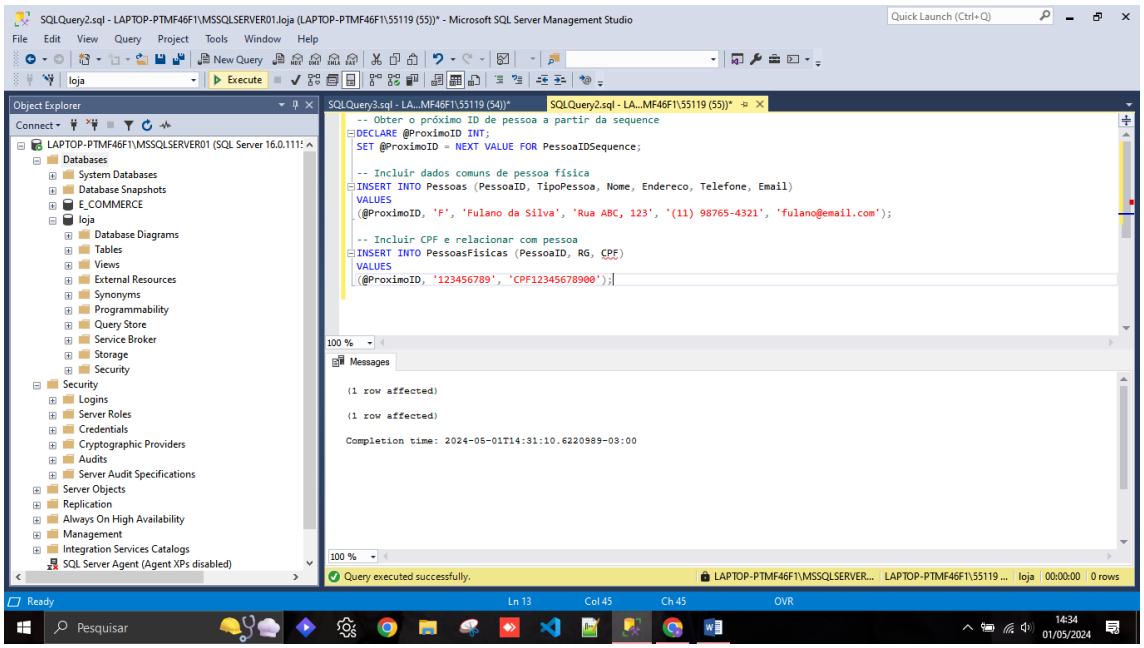
SELECT p.nome AS Produto, SUM(v.quantidade * v.precoUnitario) / SUM(v.quantidade) AS ValorMedioVenda
FROM Vendas v
JOIN Produtos p ON v.idProduto = p.idProduto
GROUP BY p.nome;

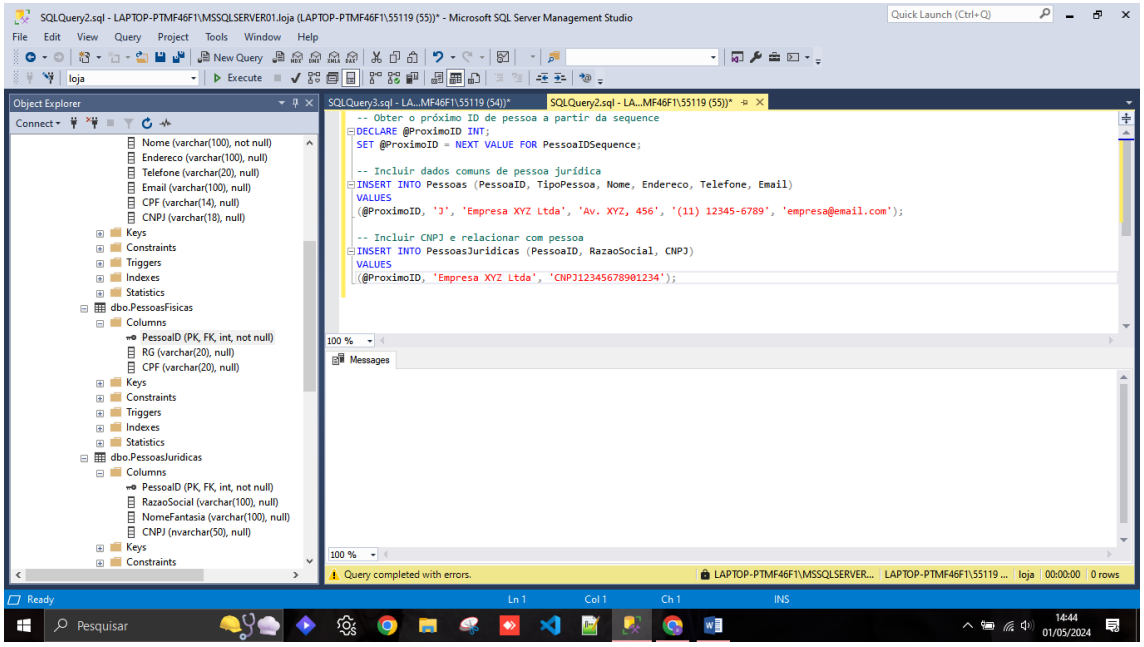
Resultados de execução do Códigos:

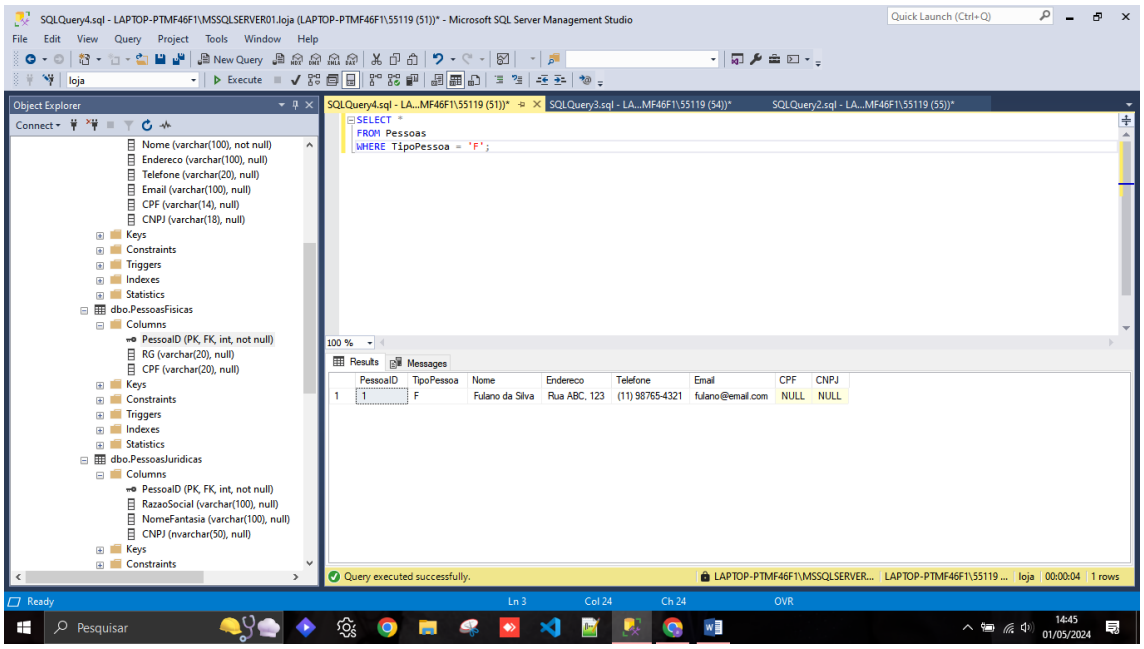


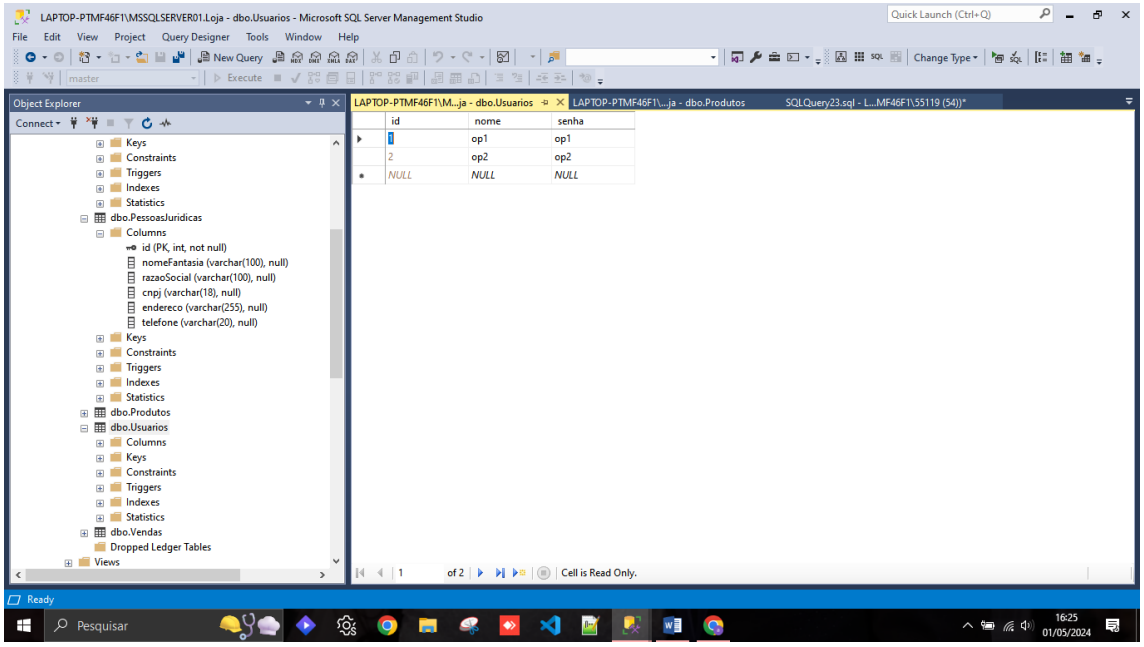
Estácio

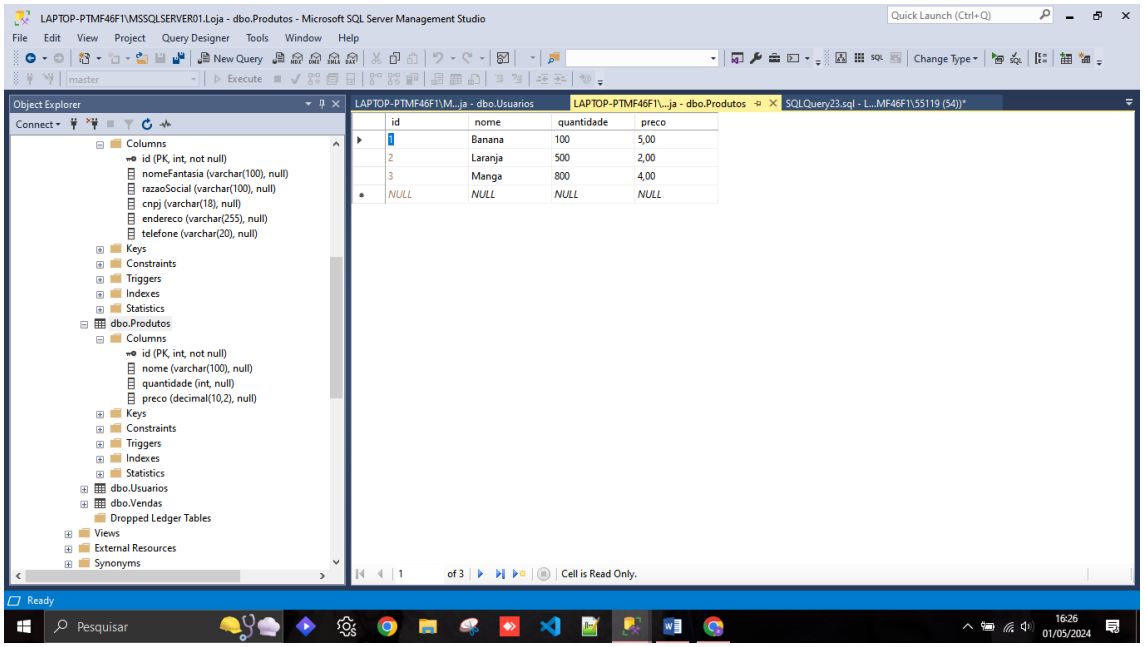

SQLQuery2.sql - LAPTOP-PTMF46F1\MSSQLSERVER01.loja (LAPTOP-PTMF46F1\55119 (55))* - Microsoft SQL Server Management Studio
Object Explorer
INSERT INTO Produtos (ProdutoID, NomeProduto, Quantidade, PrecoVenda)
VALUES
(1, 'Camiseta', 100, 29.99),
(2, 'Calça Jeans', 50, 59.99),
(3, 'Tênis', 80, 79.99),
(4, 'Mochila', 30, 39.99),
(5, 'Relógio', 20, 99.99);
Messages
A instrução foi finalizada.
Completion time: 2024-05-01T14:07:10.6242043-03:00
Query completed with errors.
Ready
Pesquisar
14:07
01/05/2024

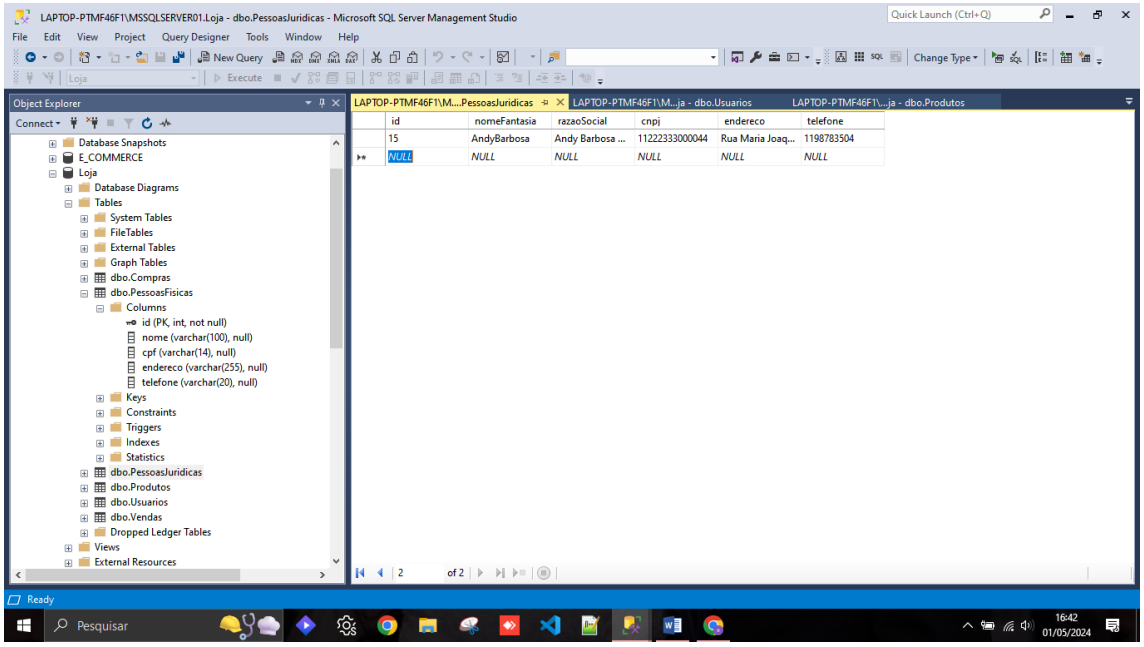

SQLQuery2.sql - LAPTOP-PTMF46F1\MSSQLSERVER01.loja (LAPTOP-PTMF46F1\55119 (55))* - Microsoft SQL Server Management Studio
Object Explorer
-- Obter o próximo ID de pessoa a partir da sequence
DECLARE @ProximoID INT;
SET @ProximoID = NEXT VALUE FOR PessoaIDSequence;
-- Incluir dados comuns de pessoa física
INSERT INTO Pessoas (PessoaID, TipoPessoa, Nome, Endereco, Telefone, Email)
VALUES
(@ProximoID, 'F', 'Fulano da Silva', 'Rua ABC, 123', '(11) 98765-4321', 'fulano@email.com');
-- Incluir CPF e relacionar com pessoa
INSERT INTO PessoasFisicas (PessoaID, RG, CPF)
VALUES
(@ProximoID, '123456789', 'CPF12345678900');
Messages
(1 row affected)
(1 row affected)
Completion time: 2024-05-01T14:31:10.6220989-03:00
Query executed successfully.
Ready
Pesquisar
14:34
01/05/2024

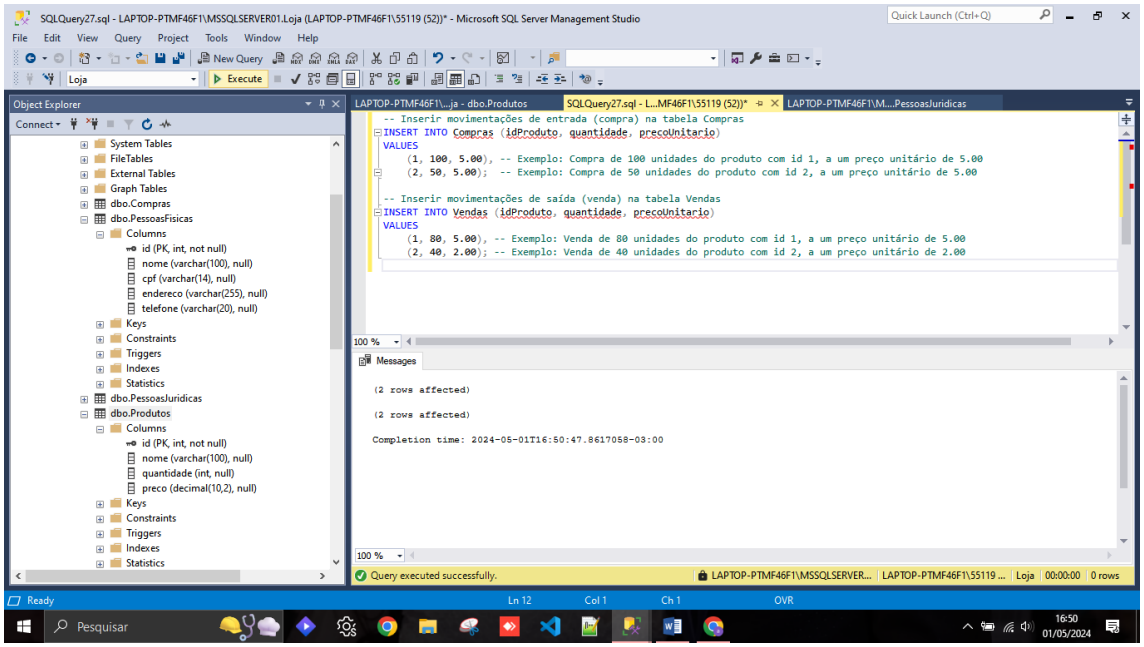

SQLQuery2.sql - LAPTOP-PTMF46F1\MSSQLSERVER01.loja (LAPTOP-PTMF46F1\55119 (55))* - Microsoft SQL Server Management Studio
-- Obter o próximo ID de pessoa a partir da sequence
DECLARE @ProximoID INT;
SET @ProximoID = NEXT VALUE FOR PessoaIDSequence;
-- Incluir dados comuns de pessoa jurídica
INSERT INTO Pessoas (PessoaID, TipoPessoa, Nome, Endereco, Telefone, Email)
VALUES
(@ProximoID, 'J', 'Empresa XYZ Ltda', 'Av. XYZ, 456', '(11) 12345-6789', 'empresa@email.com');
-- Incluir CNPJ e relacionar com pessoa
INSERT INTO PessoasJuridicas (PessoaID, RazaoSocial, CNPJ)
VALUES
(@ProximoID, 'Empresa XYZ Ltda', 'CNPJ12345678901234');
Query completed with errors.


SQLQuery4.sql - LAPTOP-PTMF46F1\MSSQLSERVER01.loja (LAPTOP-PTMF46F1\55119 (51))* - Microsoft SQL Server Management Studio
SELECT *
FROM Pessoas
WHERE TipoPessoa = 'F';
PessoaID TipoPessoa Nome Endereco Telefone Email CPF CNPJ
1 1 F Fulano da Silva Rua ABC, 123 (11) 98765-4321 fulano@email.com NULL NULL
Query executed successfully.


LAPTOP-PTMF46F1\MSSQLSERVER01.Loja - dbo.Usuarios - Microsoft SQL Server Management Studio
Object Explorer
dbo.PessoasJuridicas
dbo.Produtos
dbo.Usuarios
dbo.Vendas
id nome senha
1 op1 op1
2 op2 op2
Cell is Read Only.
Ready


LAPTOP-PTMF46F1\MSSQLSERVER01.Loja - dbo.Produtos - Microsoft SQL Server Management Studio
Object Explorer
dbo.Produtos
id nome quantidade preco
1 Banana 100 5,00
2 Laranja 500 2,00
3 Manga 800 4,00
Cell is Read Only.
Ready


LAPTOP-PTMF46F1\MSSQLSERVER01.Loja - dbo.PessoasJuridicas - Microsoft SQL Server Management Studio
Object Explorer
Database Snapshots
E_COMMERCE
Loja
Database Diagrams
Tables
System Tables
FileTables
External Tables
Graph Tables
dbo.Compras
dbo.PessoasFisicas
Columns
id (PK, int, not null)
nome (varchar(100), null)
cpf (varchar(14), null)
endereco (varchar(255), null)
telefone (varchar(20), null)
Keys
Constraints
Triggers
Indexes
Statistics
dbo.PessoasJuridicas
dbo.Produtos
dbo.Usuarios
dbo.Vendas
Dropped Ledger Tables
Views
External Resources
id nomeFantasia razaoSocial cnpj endereco telefone
15 AndyBarbosa Andy Barbosa ... 11222333000044 Rua Maria Joaq... 1198783504
NULL NULL NULL NULL NULL NULL
Ready
16:42
01/05/2024


SQLQuery27.sql - LAPTOP-PTMF46F1\MSSQLSERVER01.Loja (LAPTOP-PTMF46F1\55119 (52))* - Microsoft SQL Server Management Studio
-- Inserir movimentações de entrada (compra) na tabela Compras
INSERT INTO Compras (idProduto, quantidade, precoUnitario)
VALUES
(1, 100, 5.00), -- Exemplo: Compra de 100 unidades do produto com id 1, a um preço unitário de 5.00
(2, 50, 5.00); -- Exemplo: Compra de 50 unidades do produto com id 2, a um preço unitário de 5.00
-- Inserir movimentações de saída (venda) na tabela Vendas
INSERT INTO Vendas (idProduto, quantidade, precoUnitario)
VALUES
(1, 80, 5.00), -- Exemplo: Venda de 80 unidades do produto com id 1, a um preço unitário de 5.00
(2, 40, 2.00); -- Exemplo: Venda de 40 unidades do produto com id 2, a um preço unitário de 2.00
Messages
(2 rows affected)
(2 rows affected)
Completion time: 2024-05-01T16:50:47.8617058-03:00
Query executed successfully.
Ready
16:50
01/05/2024

Análise e Conclusão:

Quais as diferenças no uso de sequence e identity?

IDENTITY é mais simples e específico para colunas autoincrementadas, enquanto SEQUENCE é mais flexível e pode ser usado em várias situações onde é necessário gerar valores sequenciais únicos em diferentes tabelas ou colunas. A escolha entre eles depende das necessidades específicas do banco de dados e das funcionalidades disponíveis no sistema de gerenciamento de banco de dados utilizado.

Qual a importância das chaves estrangerias para a consistência do banco?

As chaves estrangeiras desempenham um papel crucial na garantia da integridade, consistência e eficiência dos dados em um banco de dados relacional, mantendo a integridade referencial e promovendo uma estrutura de dados organizada e coerente.

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

Álgebra Relacional:

- Seleção (σ): Retorna as tuplas que satisfazem uma condição específica.
- Projeção (π): Seleciona colunas específicas de uma tabela.
- Uniãp (∪): Combina duas relações em uma, removendo duplicatas.
- Interseção (∩): Retorna as tuplas comuns a duas relações.
- Diferença (-): Retorna as tuplas presentes em uma relação, mas não na outra.
- Produto Cartesiano (×): Combina cada tupla de uma relação com cada tupla da outra relação.
- Junção (⋈): Combina as tuplas de duas relações com base em uma condição de junção.

Cálculo Relacional:

- Cálculo de Tupla (Tupla): Define consultas em termos de variáveis de tupla.
- Cálculo de Domínio (Domínio): Define consultas em termos de variáveis de domínio.
- Operadores Específicos: Incluem operadores como EXISTS, FOR ALL, UNIQUE, e outras funções específicas do cálculo relacional.

Os operadores da álgebra relacional são mais comuns e amplamente utilizados em sistemas de gerenciamento de banco de dados SQL, enquanto o cálculo

relacional é mais abstrato e é usado principalmente em teoria de banco de dados e linguagens de consulta baseadas em lógica. Embora o SQL tenha sido influenciado pela álgebra relacional, ele também inclui elementos do cálculo relacional para fornecer uma linguagem de consulta mais abrangente.

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

O agrupamento em consultas SQL é feito usando a cláusula GROUP BY.
Esta cláusula agrupa as linhas de uma tabela com base nos valores de uma ou mais colunas.
As funções de agregação, como SUM, COUNT, AVG, MIN e MAX, são frequentemente usadas em conjunto com a cláusula GROUP BY para calcular valores agregados para cada grupo.
As colunas que não estão nas funções de agregação devem estar listadas na cláusula GROUP BY.

Requisito Obrigatório: O requisito obrigatório ao usar a cláusula GROUP BY é que todas as colunas selecionadas na consulta que não são funções de agregação devem estar presentes na cláusula GROUP BY.
Isso garante que cada linha resultante da consulta esteja associada a um grupo específico, evitando ambiguidades nos resultados da consulta.
Em resumo, o agrupamento em consultas SQL é realizado usando a cláusula GROUP BY, que agrupa as linhas da tabela com base nos valores das colunas especificadas. O requisito obrigatório é que todas as colunas selecionadas na consulta que não são funções de agregação devem estar presentes na cláusula GROUP BY para evitar ambiguidades nos resultados da consulta.