



## **RPG0015 - Vamos Manter as Informações?**

**Jéssica Maria de Carvalho Matrícula: 202209187939**

**POLO JARDIM SÃO BERNARDO - SÃO PAULO - SP**

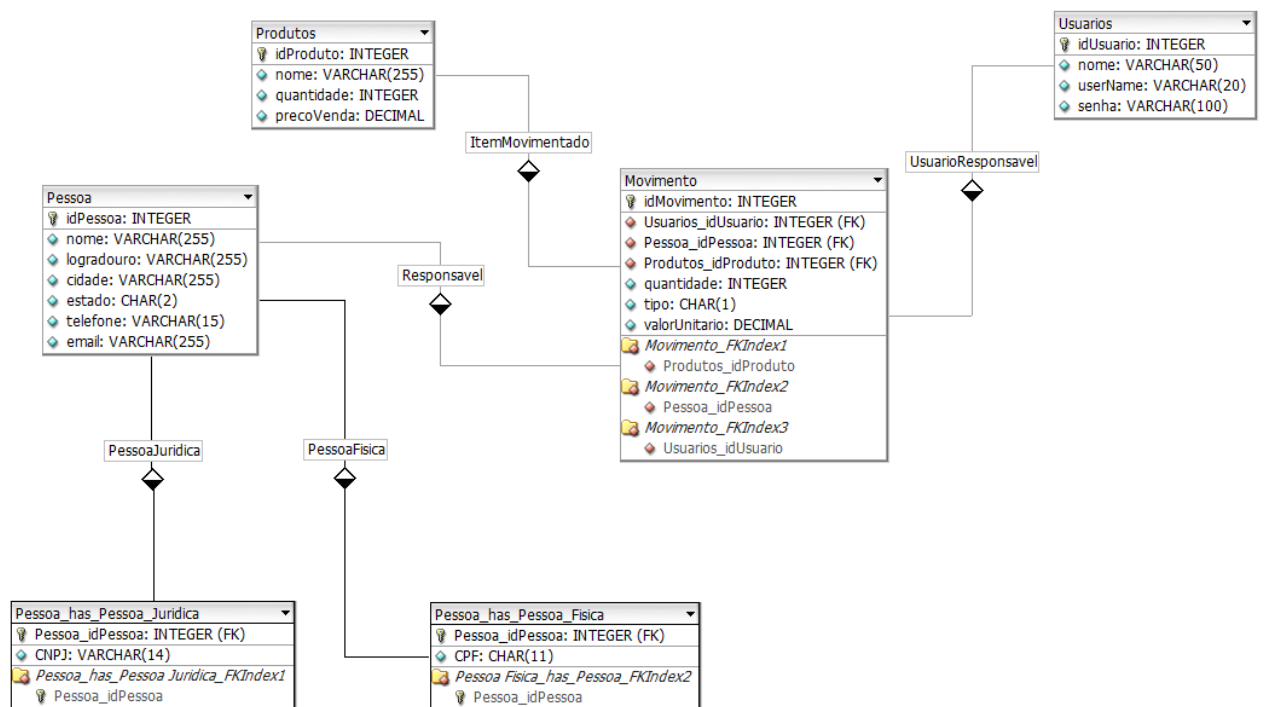
**Nível 2: Vamos Manter as Informações? – 9003 – 3º**

**Endereço do Repositório GIT: <https://github.com/Jessicac30/Criando-Banco-de-Dados>**

### **Objetivo da Prática**

1. Identificar os requisitos de um sistema e transformá-los no modelo adequado.
2. Utilizar ferramentas de modelagem para bases de dados relacionais.
3. Explorar a sintaxe SQL na criação das estruturas do banco (DDL).
4. Explorar a sintaxe SQL na consulta e manipulação de dados (DML)
5. No final do exercício, o aluno terá vivenciado a experiência de modelar a base de dados para um sistema simples, além de implementá-la, através da sintaxe SQL, na plataforma do SQL Server.

### **1º Procedimento – Criando o Banco de Dados**



## 1) Tabela de Usuários (Usuarios):

- a) Armazena informações sobre os usuários do sistema.
- b) Campos:
  - i) **idUsuario**: Chave primária para identificação do usuário.
  - ii) **nome**: Nome do usuário.
  - iii) **userName**: Nome de usuário único para login.
  - iv) **Senha**: Senha do usuário para autenticação.

## 2) Tabela de Produtos (Produtos):

- a) Armazena informações sobre os produtos disponíveis.
- b) Campos:
  - i) **idProduto**: Chave primária para identificação do produto.
  - ii) **nome**: Nome do produto.
  - iii) **quantidade**: Quantidade disponível em estoque.
  - iv) **precoVenda**: Preço de venda do produto.

## 3) Tabela de Pessoas (Pessoa):

- a) Armazena informações sobre pessoas físicas.
- b) Campos:
  - i) **idPessoa**: Chave primária para identificação da pessoa.
  - ii) **nome**: Nome da pessoa física.
  - iii) **logradouro**: Endereço da pessoa física.
  - iv) **cidade**: Cidade da pessoa física.
  - v) **estado**: Estado da pessoa física.
  - vi) **telefone**: Informações de contato da pessoa física.
  - vii) **email**: Email da pessoa física.

## 4) Tabela Pessoa\_has\_Pessoa\_Fisica (Pessoa Física):

- a) Armazena informações sobre pessoas físicas.
- b) Campos:
  - i) **CPF**: Número de CPF da pessoa física (único).

## 5) Tabela Pessoa\_has\_Pessoa\_Juridica (Pessoa Jurídica):

- a) Armazena informações sobre pessoas jurídicas.
- b) Campos:
  - i) **CNPJ**: Número de CNPJ da empresa (único).

## 6) Tabela de Movimento (Movimento):

- a) Registra as informações dos movimentos realizadas.
- b) Campos:

- i) **idMovimento**: Chave primária para identificação da venda.
- ii) **Usuarios\_idUsuario**: Chave estrangeira para o ID do usuário relacionado ao movimento.
- iii) **Pessoa\_idPessoa**: Chave estrangeira para o ID da pessoa relacionada ao movimento
- iv) **Produtos\_idProduto**: Chave estrangeira para o ID do produto movimentado.
- v) **quantidade**: Quantidade de produtos movimentados.
- vi) **tipo**: Tipo do movimento, onde **E** representa **Entrada** e **S** representa **Saída**.
- vii) **ValorUnitario**: Valor unitário do produto movimentado.

Criando as tabelas para as entidades de acordo com os requisitos mencionados:

-- Script para criação de tabelas no SQL Server

--                                      Tabela                                      de                                      Usuários

```
CREATE TABLE Usuarios (
    idUsuario INT PRIMARY KEY IDENTITY,
    nome VARCHAR(50) NOT NULL,
```

```
    userName VARCHAR(20) UNIQUE NOT NULL,  
    senha VARCHAR(100) NOT NULL  
);
```

-- Tabela de Produtos

```
CREATE TABLE Produtos (  
    idProduto INT PRIMARY KEY IDENTITY,  
    nome VARCHAR(255) NOT NULL,  
    quantidade INT NOT NULL CHECK (quantidade >= 0),  
    precoVenda DECIMAL(10, 2) NOT NULL CHECK (precoVenda >= 0)  
);
```

-- Criando sequência

```
CREATE SEQUENCE Seq_idPessoa  
    START WITH 1  
    INCREMENT BY 1  
    MINVALUE 1  
    NO MAXVALUE  
    CACHE 10;
```

-- Tabela de Pessoas

```
CREATE TABLE Pessoa (  
    idPessoa INT PRIMARY KEY DEFAULT NEXT VALUE FOR Seq_idPessoa,  
    nome VARCHAR(255) NOT NULL,  
    logradouro VARCHAR(255) NOT NULL,  
    cidade VARCHAR(255) NOT NULL,  
    estado CHAR(2) NOT NULL CHECK (LEN(estado) = 2),  
    telefone VARCHAR(15),
```

```
email VARCHAR(255)
);
```

-- Tabela Pessoa Física

```
CREATE TABLE PessoaFisica (
    Pessoa_idPessoa INT PRIMARY KEY,
    CPF CHAR(11) UNIQUE NOT NULL,
    FOREIGN KEY (Pessoa_idPessoa) REFERENCES Pessoa (idPessoa)
);
```

-- Tabela Pessoa Jurídica

```
CREATE TABLE PessoaJuridica (
    Pessoa_idPessoa INT PRIMARY KEY,
    CNPJ VARCHAR(14) UNIQUE NOT NULL,
    FOREIGN KEY (Pessoa_idPessoa) REFERENCES Pessoa (idPessoa)
);
```

-- Tabela de Movimentos

```
CREATE TABLE Movimento (
    idMovimento INT PRIMARY KEY IDENTITY,
    Usuarios_idUsuario INT NOT NULL,
    Pessoa_idPessoa INT NOT NULL,
    Produtos_idProduto INT NOT NULL,
    quantidade INT NOT NULL CHECK (quantidade > 0),
    tipo CHAR(1) NOT NULL CHECK (tipo = 'E' OR tipo = 'S'),
```

```
valorUnitario DECIMAL(10, 2) NOT NULL CHECK (valorUnitario >= 0),  
FOREIGN KEY (Usuarios_idUsuario) REFERENCES Usuarios (idUsuario),  
FOREIGN KEY (Pessoa_idPessoa) REFERENCES Pessoa (idPessoa),  
FOREIGN KEY (Produtos_idProduto) REFERENCES Produtos (idProduto)  
);
```

## **Análise e Conclusão:**

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

As diferentes cardinalidades (1x1, 1xN e NxN) em um banco de dados relacional são implementadas através do uso de chaves estrangeiras (foreign keys) e da estruturação apropriada das tabelas. Aqui está uma explicação de como cada uma delas é implementada:

### 1. Cardinalidade 1x1:

Na cardinalidade 1x1, cada registro em uma tabela está associado a um único registro em outra tabela e vice-versa.

**Implementação:** Isso é geralmente alcançado por meio de uma chave estrangeira em uma tabela que referencia a chave primária de outra tabela.

**Exemplo:** Uma tabela de Usuários tem uma chave estrangeira que faz referência a uma tabela de DetalhesUsuario. Cada usuário tem um único conjunto de detalhes associados a ele.

### 2. Cardinalidade 1xN (ou 1 para muitos):

Na cardinalidade 1xN, um registro em uma tabela está associado a vários registros em outra tabela, mas cada registro na segunda tabela está associado a apenas um registro na primeira tabela.

**Implementação:** Isso é alcançado quando uma tabela tem uma chave estrangeira que faz referência à chave primária de outra tabela.

**Exemplo:** Uma tabela de Clientes pode ter várias entradas na tabela de Pedidos. Cada pedido está associado a apenas um cliente, mas um cliente pode ter vários pedidos.

### 3. Cardinalidade NxN (ou muitos para muitos):

Na cardinalidade NxN, vários registros em uma tabela estão associados a vários registros em outra tabela.

**Implementação:** Isso é alcançado por meio de uma tabela intermediária (tabela de associação ou tabela de junção) que conecta as duas tabelas principais por meio de suas chaves estrangeiras.

**Exemplo:** Se tivermos tabelas de Estudantes e Cursos, uma tabela de Matrículas pode ser usada para representar a relação entre estudantes e cursos. Cada entrada na tabela de Matrículas conecta um estudante a um curso específico.



**Chaves Estrangeiras:**

As chaves estrangeiras são a base para estabelecer esses relacionamentos. Elas garantem a integridade referencial e são usadas para conectar registros em diferentes tabelas. Ao criar tabelas e definir relacionamentos, é fundamental entender a natureza dos dados e as relações entre eles para escolher o tipo correto de cardinalidade para cada situação.

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

Para representar o uso de herança em bancos de dados relacionais, pode-se usar o conceito de herança de tipo de entidade. Geralmente, isso é implementado usando uma tabela para a classe base (ou entidade base) e tabelas separadas para as classes derivadas (ou entidades derivadas). Existem várias estratégias para implementar herança em um banco de dados relacional, incluindo Tabelas Separadas (Cada tabela representa uma entidade concreta), Tabelas de União (Todas as entidades são armazenadas em uma única tabela) e Tabelas Concretas com Chaves Estrangeiras (Uma tabela para a classe base e uma para cada classe derivada).

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

O SQL Server Management Studio (SSMS) é uma ferramenta robusta e oferece diversas funcionalidades para gerenciar bancos de dados SQL Server de maneira eficiente, melhorando a produtividade. Algumas dessas funcionalidades incluem:

**Interface Gráfica Amigável:** O SSMS fornece uma interface gráfica intuitiva para criar, editar e gerenciar objetos de banco de dados, como tabelas, procedimentos armazenados, visões e muito mais.

**Editor SQL Avançado:** Possui um editor de consultas SQL avançado com realce de sintaxe, assistência automática, sugestões de código e depuração de consultas para facilitar a escrita e execução de scripts SQL.

**Gerenciamento de Segurança:** Permite gerenciar permissões de usuários, funções, logins e outras configurações de segurança para proteger o banco de dados.

**Monitoramento e Otimização de Desempenho:** Oferece ferramentas para monitorar o desempenho do banco de dados, identificar gargalos de desempenho e otimizar consultas para melhorar a eficiência do sistema.

**Backup e Restauração:** Facilita a realização de backups regulares do banco de dados e também permite restaurar backups quando necessário, garantindo a segurança e a disponibilidade dos dados.

Em resumo, o SQL Server Management Studio é uma ferramenta abrangente que simplifica tarefas complexas de administração de banco de dados, ajudando os administradores e desenvolvedores a trabalhar de forma mais eficiente e produtiva.

## **2º Procedimento – Alimentando a Base**

### **-- Inserindo dados na tabela de usuários**

```
USE Loja;
```

```
INSERT INTO Usuarios (Nome, Username, Senha)
```

```
VALUES
```

```
('Operador Um', 'op1', 'op1'), -- Usuário op1, senha op1
```

```
('Operador Dois', 'op2', 'op2'); -- Usuário op2, senha op2
```

### **-- Busca de usuários**

```
SELECT idUsuario, Username, Senha
```

```
FROM Usuarios;
```

The screenshot shows a SQL query execution window. The query is: `SELECT idUsuario, Username, Senha FROM Usuarios;`. Below the query, there are tabs for 'Resultados' (Results) and 'Mensagens' (Messages). The 'Resultados' tab is active, displaying a table with 4 columns: an index, 'idUsuario', 'Username', and 'Senha'. The table contains two rows of data.

	idUsuario	Username	Senha
1	1	op1	op1
2	2	op2	op2

### -- Inserindo produtos na tabela de Produtos

```
SET IDENTITY_INSERT Produtos ON;
```

```
USE Loja;
```

```
INSERT INTO Produtos (idProduto, nome, quantidade, precoVenda)
```

```
VALUES
```

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

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

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

### -- Seleção de Produtos

```
SELECT idProduto, nome, quantidade, precoVenda
```

```
FROM Produtos
```

```
WHERE idProduto IN (1, 3, 4);
```

-- Seleção de Produtos				
<pre> SELECT idProduto, nome, quantidade, precoVenda FROM Produtos WHERE idProduto IN (1, 3, 4); </pre>				
100 %				
<div> <div>Resultados</div> <div>Mensagens</div> </div>				
	idProduto	nome	quantidade	precoVenda
1	1	Banana	100	5.00
2	3	Laranja	500	2.00
3	4	Manga	800	4.00

**-- Obter o próximo ID de pessoa a partir da sequence**

```
DECLARE @PessoaID INT;
```

```
SET @PessoaID = NEXT VALUE FOR Seq_idPessoa;
```

**-- Incluir na tabela Pessoa os dados comuns para Pessoa Física**

```
INSERT INTO Pessoa (idPessoa, nome, logradouro, cidade, estado, telefone, email)
```

```
VALUES (@PessoaID, 'Joao', 'Rua 12, casa 3, Quintanda', 'Riacho do Sul', 'PA', '1111-1111', 'joao@riacho.com');
```

**-- Incluir em PessoaFisica o CPF, relacionando com Pessoa**

```
INSERT INTO PessoaFisica (Pessoa_idPessoa, CPF)
```

```
VALUES (@PessoaID, '11111111111');
```

**-- Obter o próximo ID de pessoa para Pessoa Jurídica**

```
SET @PessoaID = NEXT VALUE FOR Seq_idPessoa;
```

**-- Incluir na tabela Pessoa os dados comuns para Pessoa Jurídica**

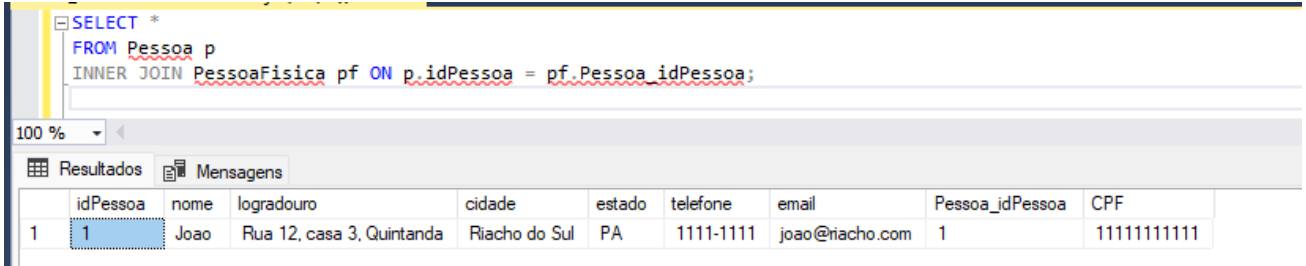
```
INSERT INTO Pessoa (idPessoa, nome, logradouro, cidade, estado, telefone, email)
VALUES (@PessoaID, 'JJC', 'Rua 11, Centro', 'Riacho do Norte', 'PA', '1212-1212',
'jjc@riacho.com');
```

**-- Incluir em PessoaJuridica o CNPJ, relacionando com Pessoa**

```
INSERT INTO PessoaJuridica (Pessoa_idPessoa, CNPJ)
VALUES (@PessoaID, '22222222222222');
```

**-- Busca de Pessoa física**

```
SELECT *
FROM Pessoa p
INNER JOIN PessoaFisica pf ON p.idPessoa = pf.Pessoa_idPessoa;
```



The screenshot shows a SQL query editor with the following query:

```
SELECT *
FROM Pessoa p
INNER JOIN PessoaFisica pf ON p.idPessoa = pf.Pessoa_idPessoa;
```

Below the query editor, there is a results pane showing a table with 10 columns: idPessoa, nome, logradouro, cidade, estado, telefone, email, Pessoa\_idPessoa, and CPF. The table contains one row of data.

	idPessoa	nome	logradouro	cidade	estado	telefone	email	Pessoa_idPessoa	CPF
1	1	Joao	Rua 12, casa 3, Quintanda	Riacho do Sul	PA	1111-1111	joao@riacho.com	1	11111111111

**-- Busca de Pessoa Juridica**

```
SELECT *
FROM Pessoa p
INNER JOIN PessoaJuridica pf ON p.idPessoa = pf.Pessoa_idPessoa;
```

<pre> SELECT * FROM Pessoa p INNER JOIN PessoaJuridica pf ON p.idPessoa = pf.Pessoa_idPessoa; </pre>									
100 %									
<div>Resultados</div> <div>Mensagens</div>									
	idPessoa	nome	logradouro	cidade	estado	telefone	email	Pessoa_idPessoa	CNPJ
1	2	JJC	Rua 11. Centro	Riacho do Norte	PA	1212-1212	jjc@riacho.com	2	22222222222222

### -- Inserir dados na tabela Movimento

```

INSERT INTO dbo.Movimento (Usuarios_idUsuario, Pessoa_idPessoa,
Produtos_idProduto, quantidade, tipo, valorUnitario)

```

```

VALUES

```

```

(1, 1, 1, 20, 'S', 4.00),

```

```

(1, 1, 3, 15, 'S', 2.00),

```

```

(2, 1, 3, 10, 'S', 3.00),

```

```

(1, 2, 3, 15, 'E', 5.00),

```

```

(1, 2, 4, 20, 'E', 4.00);

```

### -- Consulta Movimento

```

SELECT idMovimento, Usuarios_idUsuario, Pessoa_idPessoa, Produtos_idProduto,
quantidade, tipo, valorUnitario

```

```

FROM dbo.Movimento;

```

<pre>SELECT idMovimento, Usuarios_idUsuario, Pessoa_idPessoa, Produtos_idProduto, quantidade, tipo, valorUnitario FROM dbo.Movimento;</pre>							
100 %							
Resultados Mensagens							
	idMovimento	Usuarios_idUsuario	Pessoa_idPessoa	Produtos_idProduto	quantidade	tipo	valorUnitario
1	2	1	1	1	20	S	4.00
2	3	1	1	3	15	S	2.00
3	4	2	1	3	10	S	3.00
4	5	1	2	3	15	E	5.00
5	6	1	2	4	20	E	4.00

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

SELECT

m.idMovimento,

p.nome AS Produto,

pe.nome AS Fornecedor,

m.quantidade,

m.valorUnitario,

(m.quantidade \* m.valorUnitario) AS ValorTotal

FROM

dbo.Movimento m

JOIN dbo.Produtos p ON m.Produtos\_idProduto = p.idProduto

JOIN dbo.Pessoa pe ON m.Pessoa\_idPessoa = pe.idPessoa

WHERE

m.tipo = 'E';



<pre> SELECT     m.idMovimento,     p.nome AS Produto,     pe.nome AS Fornecedor,     m.quantidade,     m.valorUnitario,     (m.quantidade * m.valorUnitario) AS ValorTotal FROM     dbo.Movimento m JOIN dboProdutos p ON m.Produtos_idProduto = p.idProduto JOIN dbo.Pessoa pe ON m.Pessoa_idPessoa = pe.idPessoa WHERE     m.tipo = 'E'; </pre>						
100 %						
<div> <div>Resultados</div> <div>Mensagens</div> </div>						
	idMovimento	Produto	Fornecedor	quantidade	valorUnitario	ValorTotal
1	5	Laranja	JJC	15	5.00	75.00
2	6	Manga	JJC	20	4.00	80.00

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

SELECT

```

    m.idMovimento,

    p.nome AS Produto,

    pe.nome AS Fornecedor,

    m.quantidade,

    m.valorUnitario,

    (m.quantidade * m.valorUnitario) AS ValorTotal

```

FROM

```

    dbo.Movimento m

```

```

JOIN dbo.Produtos p ON m.Produtos_idProduto = p.idProduto

```

```

JOIN dbo.Pessoa pe ON m.Pessoa_idPessoa = pe.idPessoa

```

WHERE

```

    m.tipo = 'S';

```

```

SELECT
    m.idMovimento,
    p.nome AS Produto,
    pe.nome AS Fornecedor,
    m.quantidade,
    m.valorUnitario,
    (m.quantidade * m.valorUnitario) AS ValorTotal
FROM
    dbo.Movimento m
JOIN dboProdutos p ON m.Produtos_idProduto = p.idProduto
JOIN dbo.Pessoa pe ON m.Pessoa_idPessoa = pe.idPessoa
WHERE
    m.tipo = 'S';

```

100 %

Resultados Mensagens

	idMovimento	Produto	Fornecedor	quantidade	valorUnitario	ValorTotal
1	2	Banana	Joao	20	4.00	80.00
2	3	Laranja	Joao	15	2.00	30.00
3	4	Laranja	Joao	10	3.00	30.00

-- Valor total das entradas agrupadas por produto:

SELECT

p.nome AS Produto,

SUM(m.quantidade \* m.valorUnitario) AS ValorTotalEntradas

FROM

dbo.Movimento m

JOIN dboProdutos p ON m.Produtos\_idProduto = p.idProduto

WHERE

m.tipo = 'E'

GROUP BY

p.nome;

<pre> SELECT     p.nome AS Produto,     SUM(m.quantidade * m.valorUnitario) AS ValorTotalEntradas FROM     dbo.Movimento m JOIN dbo.Produtos p ON m.Produtos_idProduto = p.idProduto WHERE     m.tipo = 'E' GROUP BY     p.nome; </pre>		
100 %		
<div> <div>Resultados</div> <div>Mensagens</div> </div>		
	Produto	ValorTotalEntradas
1	Laranja	75.00
2	Manga	80.00

**-- Valor total das saídas agrupadas por produto:**

SELECT

p.nome AS Produto,

SUM(m.quantidade \* m.valorUnitario) AS ValorTotalSaidas

FROM

dbo.Movimento m

JOIN dbo.Produtos p ON m.Produtos\_idProduto = p.idProduto

WHERE

m.tipo = 'S'

GROUP BY

p.nome;

<pre> SELECT     p.nome AS Produto,     SUM(m.quantidade * m.valorUnitario) AS ValorTotalSaidas FROM     dbo.Movimento m JOIN dbo.Produtos p ON m.Produtos_idProduto = p.idProduto WHERE     m.tipo = 'S' GROUP BY     p.nome; </pre>		
100 %		
<div> <div>Resultados</div> <div>Mensagens</div> </div>		
	Produto	ValorTotalSaidas
1	Banana	80.00
2	Laranja	60.00

-- Operadores que não efetuaram movimentações de entrada (compra):

SELECT

u.idUsuario,

u.nome

FROM

dbo.Usuarios u

WHERE NOT EXISTS (

SELECT 1

FROM dbo.Movimento m

WHERE m.Usuarios\_idUsuario = u.idUsuario AND m.tipo = 'E'

);

<pre> SELECT     u.idUsuario,     u.nome FROM     dbo.Usuarios u WHERE NOT EXISTS (     SELECT 1     FROM dbo.Movimento m     WHERE m.Usuarios_idUsuario = u.idUsuario AND m.tipo = 'E' ); </pre>		
100 %		
<div> <div>Resultados</div> <div>Mensagens</div> </div>		
	idUsuario	nome
1	2	Operador Dois

**-- Valor total de entrada, agrupado por operador:**

```

SELECT
    u.idUsuario,
    u.nome AS Operador,
    SUM(m.quantidade * m.valorUnitario) AS ValorTotalEntrada
FROM
    dbo.Movimento m
JOIN dbo.Usuarios u ON m.Usuarios_idUsuario = u.idUsuario
WHERE
    m.tipo = 'E'
GROUP BY
    u.idUsuario,
    u.nome;

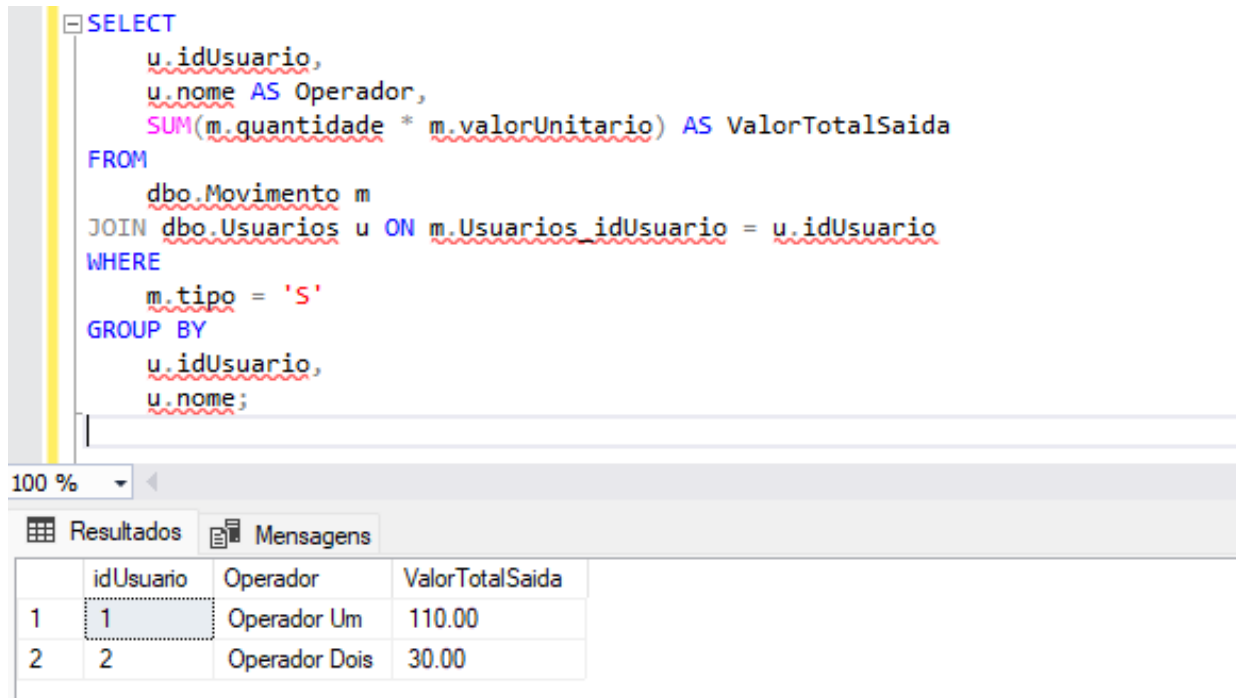
```



GROUP BY

u.idUsuario,

u.nome;



```
SELECT
    u.idUsuario,
    u.nome AS Operador,
    SUM(m.quantidade * m.valorUnitario) AS ValorTotalSaida
FROM
    dbo.Movimento m
JOIN dbo.Usuarios u ON m.Usuarios_idUsuario = u.idUsuario
WHERE
    m.tipo = 'S'
GROUP BY
    u.idUsuario,
    u.nome;
```

	idUsuario	Operador	ValorTotalSaida
1	1	Operador Um	110.00
2	2	Operador Dois	30.00

-- Valor médio de venda por produto, utilizando média ponderada:

SELECT

p.idProduto,

p.nome AS Produto,

SUM(m.quantidade \* m.valorUnitario) / SUM(m.quantidade) AS  
ValorMedioPonderado

FROM

dbo.Movimento m

JOIN dbo.Produtos p ON m.Produtos\_idProduto = p.idProduto

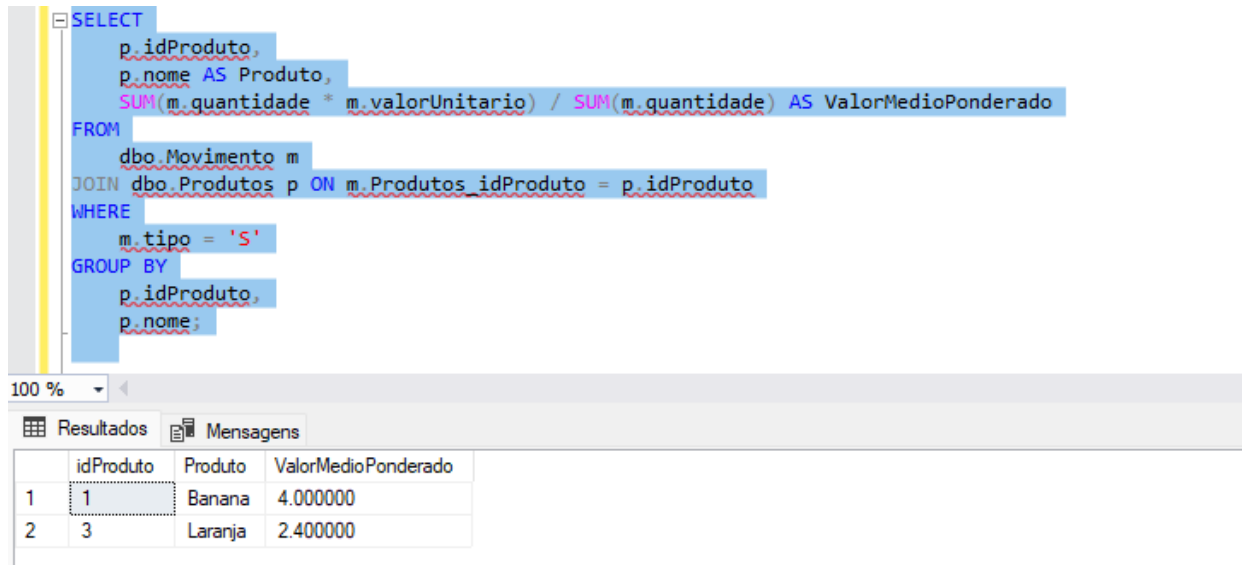
WHERE

m.tipo = 'S'

## GROUP BY

p.idProduto,

p.nome;



The screenshot shows a SQL query in the SQL Server Enterprise Manager interface. The query is as follows:

```
SELECT
    p.idProduto,
    p.nome AS Produto,
    SUM(m.quantidade * m.valorUnitario) / SUM(m.quantidade) AS ValorMedioPonderado
FROM
    dbo.Movimento m
JOIN dbo.Produtos p ON m.Produtos_idProduto = p.idProduto
WHERE
    m.tipo = 'S'
GROUP BY
    p.idProduto,
    p.nome;
```

Below the query, the results are displayed in a table with the following data:

	idProduto	Produto	ValorMedioPonderado
1	1	Banana	4.000000
2	3	Laranja	2.400000

## Análise e Conclusão:

### Quais as diferenças no uso de sequence e identity?

#### Sequence:

- É um objeto que gera uma sequência numérica não vinculada a tabelas específicas, o que permite o seu uso em várias tabelas e situações.
- Tem mais controle sobre a sequence, como definir o número inicial, incremento, mínimo, máximo e até mesmo se ela deve reciclar ou não os valores.
- Pode ser usada quando precisa de um número sequencial que não está diretamente ligado a uma coluna de identidade em uma tabela.

#### Identity:

- É uma propriedade de uma coluna específica que gera automaticamente valores numéricos sequenciais.



- É limitada a uma coluna em uma tabela e, geralmente, é usada para gerar chaves primárias.
- Não oferece tanto controle quanto uma sequence, pois é atrelada à tabela e à coluna, com opções limitadas de configuração.

### **Qual a importância das chaves estrangeiras para a consistência do banco?**

Chaves estrangeiras são cruciais para manter a integridade referencial em um banco de dados relacional, garantindo que as relações entre tabelas permaneçam consistentes.

Elas impedem que dados inconsistentes ou órfãos sejam inseridos no banco de dados, pois só permitem valores que já existem na tabela referenciada.

Facilitam operações de join e outras consultas que dependem de relacionamentos claros e definidos entre tabelas.

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

#### **Álgebra Relacional:**

- Inclui operadores como SELECT (restrição), PROJECT (projeção), JOIN, UNION, INTERSECT, DIFFERENCE, e PRODUCT (produto cartesiano).
- São operações definidas para manipular e recuperar dados de tabelas relacionais.

#### **Cálculo Relacional:**

- Baseia-se em uma formalização lógica, usando variáveis, predicados e quantificadores (FOR ALL, EXISTS).
- No SQL, corresponde a expressões como subconsultas correlacionadas, expressões CASE, e a cláusula WHERE, que podem incorporar lógica de cálculo.

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

- O agrupamento em consultas SQL é feito com o comando GROUP BY, que reúne as linhas que têm os mesmos valores em colunas especificadas.
- O requisito obrigatório para usar GROUP BY é que todas as colunas selecionadas, que não estão incluídas na cláusula GROUP BY, devem ser usadas com funções de agregação (como COUNT, SUM, AVG, MAX, MIN).
- Sem GROUP BY, funções agregadas operam em todo o conjunto de resultados; com GROUP BY, operam em conjuntos particionados de dados.