Missão Prática | Mundo 3 - Nível 2

Polo Centro - Palhoça – SC

Curso: Desenvolvimento Full Stack

Disciplina: Nível 2: Vamos Manter as Informações?

Turma: 9001

Semestre Letivo: 3

Integrantes da Prática: Andrey Haertel Aires

Repositório GIT

https://github.com/AndreyHaires/MissaoPraticaMundo3_N2

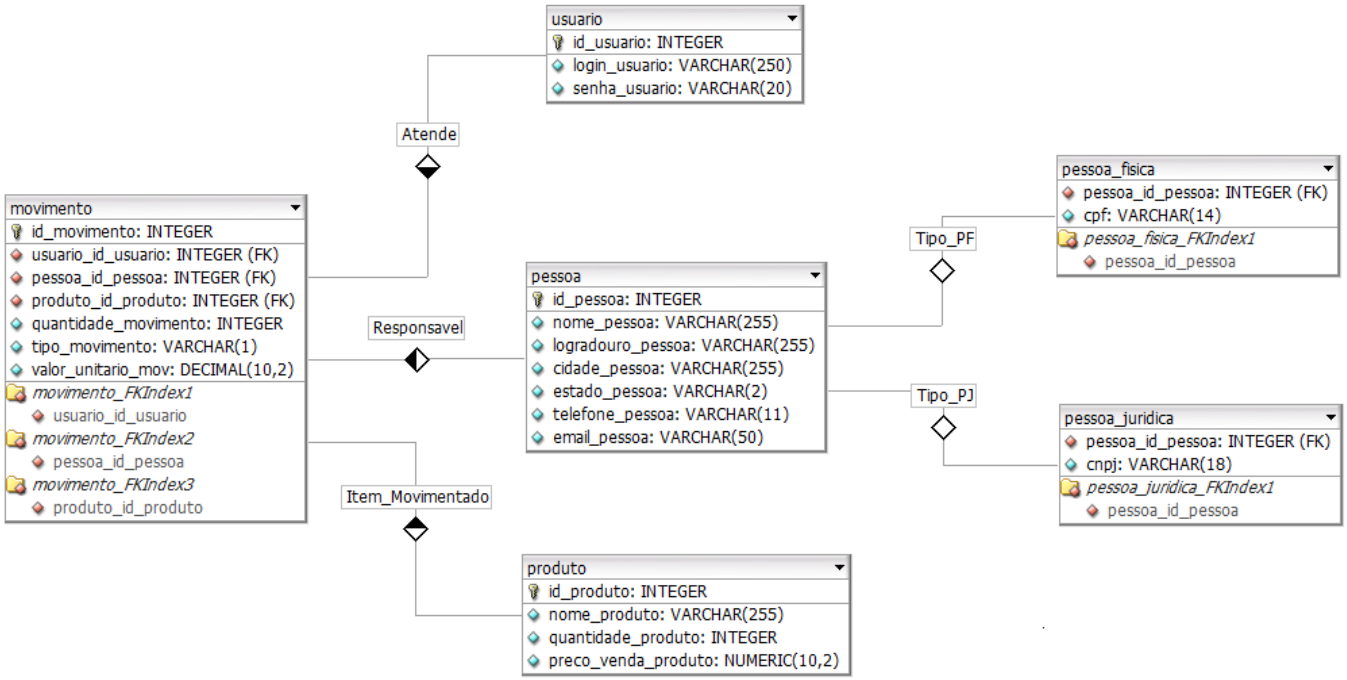
1 - **1º Procedimento | Criando o Banco de Dados**

2 - **Objetivos 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.

Todos os códigos solicitados neste roteiro de aula

Modelagem



Criando tabelas

```
-- Seleciona o banco de dados "Loja"
USE Loja;

--PROCEDIMENTO 1

CREATE TABLE usuario ( --Cria a tabela 'usuario'
  id_usuario INTEGER IDENTITY(1,1) NOT NULL PRIMARY KEY,
  login_usuario VARCHAR(250) NULL,
  senha_usuario VARCHAR(20) NULL
);

CREATE TABLE pessoa ( --Cria a tabela 'pessoa'
  id_pessoa INTEGER IDENTITY(7,1) NOT NULL PRIMARY KEY,
  nome_pessoa VARCHAR(255) NOT NULL,
  logradouro_pessoa VARCHAR(255) NULL,
  cidade_pessoa VARCHAR(255) NULL,
  estado_pessoa VARCHAR(2) NULL,
  telefone_pessoa VARCHAR(11) NULL,
  email_pessoa VARCHAR(50) NULL
);

CREATE TABLE pessoa_fisica ( --Cria a tabela 'pessoa_fisica'
  id_pessoa INTEGER PRIMARY KEY NOT NULL,
  cpf VARCHAR(14) NULL,
  FOREIGN KEY(id_pessoa)
    REFERENCES pessoa(id_pessoa)
);
```

```sql
-- Cria um índice para otimizar junções relacionadas a chaves estrangeiras na tabela pessoa_fisica.
CREATE INDEX pessoa_fisica_FKIndex1 ON pessoa_fisica (id_pessoa);

-- Cria um índice adicional para aprimorar o desempenho de consultas envolvendo a tabela
pessoa_fisica e a coluna id_pessoa.
CREATE INDEX IFK_Tipo_PF ON pessoa_fisica (id_pessoa);


CREATE TABLE pessoa_juridica ( --Cria a tabela 'pessoa_juridica'
  id_pessoa INTEGER PRIMARY KEY NOT NULL,
  cnpj VARCHAR(18) NULL,
  FOREIGN KEY(id_pessoa)
    REFERENCES pessoa(id_pessoa)
);

-- Cria um índice para otimizar junções relacionadas a chaves estrangeiras na tabela
pessoa_juridica.
CREATE INDEX pessoa_juridica_FKIndex1 ON pessoa_juridica (id_pessoa);

-- Cria um índice adicional para aprimorar o desempenho de consultas envolvendo a tabela
pessoa_juridica e a coluna id_pessoa.
CREATE INDEX IFK_Tipo_PJ ON pessoa_juridica (id_pessoa);

CREATE TABLE produto ( --Cria a tabela 'produto'
  id_produto INTEGER NOT NULL PRIMARY KEY,
  nome_produto VARCHAR(255) NULL,
  quantidade_produto INTEGER NULL,
  preco_venda_produto NUMERIC (10,2) NULL
);

CREATE TABLE movimento ( --Cria a tabela 'movimento'
  id_movimento INTEGER IDENTITY(1,1) NOT NULL PRIMARY KEY,
  id_usuario INTEGER NOT NULL,
  id_pessoa INTEGER NOT NULL,
  id_produto INTEGER NOT NULL,
  quantidade_movimento INTEGER NULL,
  tipo_movimento VARCHAR(1) NULL,
  valor_unitario_mov DECIMAL(10,2) NULL,
  FOREIGN KEY(id_usuario)
    REFERENCES usuario(id_usuario),
  FOREIGN KEY(id_pessoa)
    REFERENCES pessoa(id_pessoa),
  FOREIGN KEY(id_produto)
    REFERENCES produto(id_produto)
);

-- Cria um índice para otimizar junções relacionadas a chaves estrangeiras com a tabela usuário na
tabela movimento.
CREATE INDEX movimento_FKIndex1 ON movimento (id_usuario);

-- Cria um índice para otimizar junções relacionadas a chaves estrangeiras com a tabela pessoa na
tabela movimento.
CREATE INDEX movimento_FKIndex2 ON movimento (id_pessoa);

-- Cria um índice para otimizar junções relacionadas a chaves estrangeiras com a tabela produto na
tabela movimento.
CREATE INDEX movimento_FKIndex3 ON movimento (id_produto);

-- Cria um índice adicional para aprimorar o desempenho de consultas envolvendo a tabela movimento e
a coluna id_usuario.
CREATE INDEX IFK_Atende ON movimento (id_usuario);

-- Cria um índice adicional para aprimorar o desempenho de consultas envolvendo a tabela movimento e
a coluna id_pessoa.
CREATE INDEX IFK_Responsavel ON movimento (id_pessoa);

-- Cria um índice adicional para aprimorar o desempenho de consultas envolvendo a tabela movimento e
a coluna id_produto.
       CREATE INDEX IFK_Item_Movimentado ON movimento (id_produto);
```

Os resultados da execução dos códigos também devem ser apresentados.

```sql
-- Seleciona o banco de dados "Loja"
USE Loja;

SELECT * FROM usuario --Visualizando a tabela 'usuario'

SELECT * FROM pessoa --Visualizando a tabela 'pessoa'

SELECT * FROM pessoa_fisica --Visualizando a tabela 'pessoa fisica'

SELECT * FROM pessoa_juridica --Visualizando a tabela 'pessoa juridica'

SELECT * FROM produto --Visualizando a tabela 'produto'

SELECT * FROM movimento --Visualizando a tabela 'movimento'
```

100 %

Resultados | Mensagens

id_usuario	login_usuario	senha_usuario

id_pessoa	nome_pessoa	logradouro_pessoa	cidade_pessoa	estado_pessoa	telefone_pessoa	email_pessoa

id_pessoa	cpf

id_pessoa	cnpj

id_produto	nome_produto	quantidade_produto	preco_venda_produto

id_movimento	id_usuario	id_pessoa	id_produto	quantidade_movimento	tipo_movimento	valor_unitario_mov

Análise e Conclusão

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

Em um banco de dados relacional é realizada por meio do uso de relacionamentos entre tabelas.

1X1 (Um para Um): Nesse caso, uma tabela está relacionada a outra tabela por meio de uma chave estrangeira, e cada registro na tabela A está associado a exatamente um registro na tabela B, e vice-versa. Por exemplo, uma tabela "Pessoa" pode estar relacionada a uma tabela "Endereço" por meio de uma chave estrangeira que vincula uma pessoa a um único endereço.

1XN (Um para Muitos): Nesse cenário, uma tabela A está relacionada a uma tabela B, onde cada registro na tabela A pode estar associado a vários registros na tabela B. Isso é alcançado por meio de uma chave estrangeira na tabela B que faz referência à chave primária da tabela A. Por exemplo, uma tabela "Cliente" pode estar relacionada a uma tabela "Pedido" com a chave do cliente sendo a chave primária na tabela "Cliente" e uma chave estrangeira na tabela "Pedido" que faz referência ao cliente.

NxN (Muitos para Muitos): Para implementar um relacionamento muitos para muitos, geralmente é necessário criar uma tabela de associação intermediária que relaciona as tabelas A e B. Essa tabela de associação contém chaves estrangeiras que fazem referência às tabelas A e B, permitindo que múltiplos registros de A se relacionem com múltiplos registros de B. Por exemplo, em um sistema de gerenciamento de alunos e cursos, você pode ter uma tabela "Aluno," uma tabela "Curso" e uma tabela de associação "Aluno_Curso" que relaciona alunos a cursos matriculados.

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

Um tipo de relacionamento que pode ser utilizado é o "Tabela por Subclasse" ou "Tabela por Herança." Nesse modelo, cada subclasse ou entidade derivada é representada por uma tabela separada. As tabelas de subclasse contêm as propriedades específicas da subclasse, bem como uma chave estrangeira que faz referência à tabela de superclasse ou à tabela base. A tabela base contém as propriedades comuns a todas as subclasses. Esse modelo permite a representação de hierarquias de classes e herança no banco de dados.

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

Interface Gráfica: O SSMS oferece uma interface gráfica amigável para gerenciar bancos de dados, tabelas, consultas e outros objetos do banco de dados, o que facilita a administração e a navegação.
Editor SQL: O SSMS possui um editor de consultas SQL integrado com recursos como realce de sintaxe, sugestões de código e depuração, que ajuda a escrever e otimizar consultas SQL.
Gerenciamento de Segurança: Permite configurar e gerenciar permissões de segurança, usuários, funções e outros aspectos de segurança do banco de dados.
Monitoramento e Otimização: Oferece ferramentas de monitoramento de desempenho, planejamento de consultas e otimização de índices para melhorar o desempenho do banco de dados.
Integração com Git: Permite integrar projetos de banco de dados com sistemas de controle de versão, facilitando o gerenciamento de mudanças no esquema do banco de dados.
Importação e Exportação de Dados: Facilita a importação e exportação de dados entre diferentes fontes e destinos.
O SSMS é uma ferramenta abrangente que torna mais eficiente a administração e o gerenciamento de bancos de dados SQL Server.

2º Procedimento | Alimentando a Base

Todos os códigos solicitados neste roteiro de aula

Inserindo dados

```
-- Seleciona o banco de dados "Loja"
USE Loja;

--PROCEDIMENTO 2

-- Inserindo dados na tabela usuario
INSERT INTO dbo.usuario (login_usuario, senha_usuario)
VALUES
  ('op1', 'op1'),
  ('op2', 'op2');

-- Inserindo dados na tabela produto
INSERT INTO dbo.produto (id_produto, nome_produto, quantidade_produto, preco_venda_produto)
VALUES
(1, 'Banana', 100, 5.00),
(3, 'Laranja', 500, 2.00),
(4, 'Manga', 800, 4.00);

-- Inserir dados na tabela 'pessoa'
INSERT INTO pessoa (nome_pessoa, logradouro_pessoa, cidade_pessoa, estado_pessoa, telefone_pessoa,
email_pessoa)
VALUES ('Joao', 'Rua 12, casa 3, Quitanda', 'Riacho do Sul', 'PA', '1111-1111', 'joao@riacho.com'),
         ('JJC', 'Rua 11, Centro', 'Riacho do Norte', 'PA', '1212-1212', 'jjc@riacho.com');

-- Inserir dados na tabela 'pessoa_fisica'
INSERT INTO pessoa_fisica (id_pessoa, cpf)
VALUES (7, '12345678900');

-- Inserir dados na tabela 'pessoa_juridica'
INSERT INTO pessoa_juridica(id_pessoa, cnpj)
VALUES (8, '12345678901234');

-- Inserir dados na tabela 'movimento'
INSERT INTO movimento (id_usuario, id_pessoa, id_produto, quantidade_movimento, tipo_movimento,
valor_unitario_mov)
VALUES (1, 7, 1, 20, 'S', 4),
       (1, 7, 3, 15, 'S', 2),
       (2, 7, 3, 10, 'S', 3),
       (1, 8, 3, 15, 'E', 5),
       (1, 8, 4, 20, 'E', 4);
```

Consultando dados

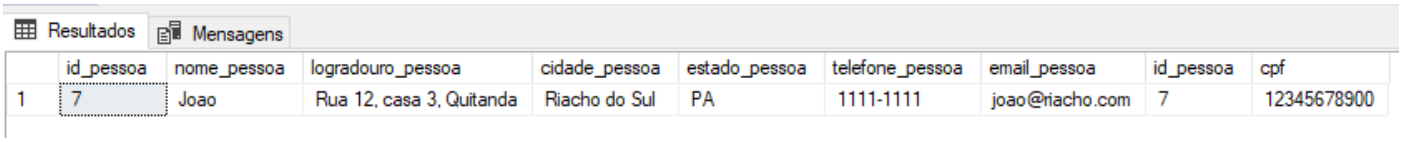
```
-- Seleciona o banco de dados "Loja"
USE Loja;

SELECT -- Este SELECT combina informações de pessoas e pessoas físicas
  p.id_pessoa, p.nome_pessoa, p.logradouro_pessoa,
  p.cidade_pessoa, p.estado_pessoa, p.telefone_pessoa,
  p.email_pessoa, pf.id_pessoa, pf.cpf
FROM
  pessoa p
FULL OUTER JOIN pessoa_fisica pf -- O FULL OUTER JOIN garante que todas as linhas de ambas as
tabelas (pessoa e pessoa_fisica) sejam incluídas no resultado.
ON p.id_pessoa = pf.id_pessoa
WHERE pf.cpf IS NOT NULL; -- Verifica se o CPF na tabela pessoa_fisica não é nulo
```

```sql
SELECT -- Este SELECT combina informações de pessoas e pessoas juridicas
  p.id_pessoa, p.nome_pessoa, p.logradouro_pessoa,
  p.cidade_pessoa, p.estado_pessoa, p.telefone_pessoa,
  p.email_pessoa, pj.id_pessoa, pj.cnpj
FROM
  pessoa p
FULL OUTER JOIN pessoa_juridica pj -- O FULL OUTER JOIN garante que todas as linhas de ambas as
tabelas (pessoa e pessoa_juridica) sejam incluídas no resultado.
ON p.id_pessoa = pj.id_pessoa
WHERE pj.cnpj IS NOT NULL; -- Verifica se o CNPJ na tabela pessoa_juridica não é nulo


-- Movimentações de entrada, com produto, fornecedor, quantidade, preço unitário e valor total.
SELECT
  prod.nome_produto AS PRODUTO,
  p.nome_pessoa AS FORNECEDOR,
  m.quantidade_movimento AS QUANTIDADE,
  m.valor_unitario_mov AS PRECO_UNITARIO,
 (m.quantidade_movimento * m.valor_unitario_mov) AS VALOR_TOTAL
FROM
 movimento m
  JOIN pessoa p ON m.id_pessoa = p.id_pessoa -- Relaciona a tabela movimento com a tabela pessoa
  JOIN produto prod ON m.id_produto = prod.id_produto -- Relaciona a tabela movimento com a tabela
produto
WHERE m.tipo_movimento = 'E'; -- Filtra as movimentações de entrada


--Movimentações de saída, com produto, comprador, quantidade, preço unitário e valor total.
SELECT
  prod.nome_produto AS PRODUTO,
  p.nome_pessoa AS COMPRADOR,
  m.quantidade_movimento AS QUANTIDADE,
  m.valor_unitario_mov AS PRECO_UNITARIO,
 (m.quantidade_movimento * m.valor_unitario_mov) AS VALOR_TOTAL
FROM
 movimento m
  JOIN pessoa p ON m.id_pessoa = p.id_pessoa -- Relaciona a tabela movimento com a tabela pessoa
  JOIN produto prod ON m.id_produto = prod.id_produto -- Relaciona a tabela movimento com a tabela
produto
WHERE m.tipo_movimento = 'S'; -- Filtra as movimentações de entrada


--Valor total das entradas agrupadas por produto.
SELECT
  prod.nome_produto PRODUTO,
  SUM(m.quantidade_movimento * m.valor_unitario_mov) AS VALOR_TOTAL_ENTRADAS
FROM
  pessoa p
INNER JOIN movimento m ON p.id_pessoa = m.id_pessoa
INNER JOIN produto prod ON m.id_produto = prod.id_produto
WHERE
  m.tipo_movimento = 'E'
GROUP BY
  prod.nome_produto;


--Valor total das saídas agrupadas por produto.
SELECT
prod.nome_produto AS PRODUTO,
SUM(m.quantidade_movimento * m.valor_unitario_mov) AS VALOR_TOTAL_SAIDAS
FROM
  pessoa p
INNER JOIN movimento m ON p.id_pessoa = m.id_pessoa
INNER JOIN produto prod ON m.id_produto = prod.id_produto
WHERE
  m.tipo_movimento = 'S'
GROUP BY
```

```
  prod.nome_produto;


--Operadores que não efetuaram movimentações de entrada (compra)
SELECT u.id_usuario, u.login_usuario
FROM usuario u
WHERE u.id_usuario NOT IN (
  SELECT DISTINCT m.id_usuario
  FROM movimento m
  WHERE m.tipo_movimento = 'E'
);

--Valor total de entrada, agrupado por operador.
SELECT
  u.login_usuario OPERADOR,
  SUM(m.quantidade_movimento * m.valor_unitario_mov) AS VALOR_TOTAL_ENTRADAS
FROM
  usuario u
INNER JOIN movimento m ON u.id_usuario = m.id_usuario
WHERE
  m.tipo_movimento = 'E'
GROUP BY
  u.login_usuario;

--Valor total de saída, agrupado por operador.
SELECT
  u.login_usuario OPERADOR,
  SUM(m.quantidade_movimento * m.valor_unitario_mov) AS VALOR_TOTAL_SAIDA
FROM
  usuario u
INNER JOIN movimento m ON u.id_usuario = m.id_usuario
WHERE
  m.tipo_movimento = 'S'
GROUP BY
  u.login_usuario

--Valor médio de venda por produto, utilizando média ponderada.
SELECT
  prod.nome_produto AS PRODUTO,
  CAST(SUM(m.quantidade_movimento * m.valor_unitario_mov) / SUM(m.quantidade_movimento) AS
NUMERIC(18,2)) AS VENDA_MEDIA_PONDERADA
FROM produto prod
LEFT JOIN movimento m ON prod.id_produto = m.id_produto
WHERE m.tipo_movimento = 'S'
GROUP BY prod.nome_produto;
```

Os resultados da execução dos códigos também devem ser apresentados

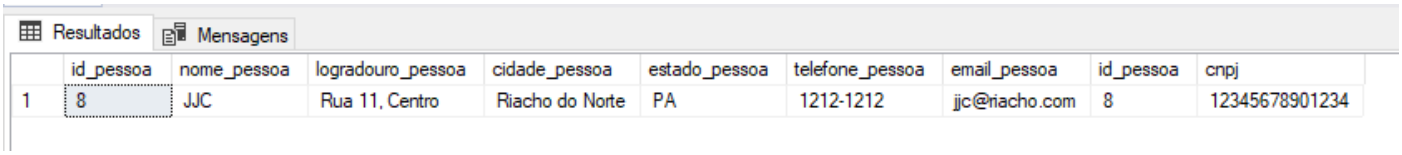
Este SELECT combina informações de pessoas e pessoas físicas



Resultados | Mensagens

	id_pessoa	nome_pessoa	logradouro_pessoa	cidade_pessoa	estado_pessoa	telefone_pessoa	email_pessoa	id_pessoa	cpf
1	7	Joao	Rua 12, casa 3, Quitanda	Riacho do Sul	PA	1111-1111	joao@riacho.com	7	12345678900

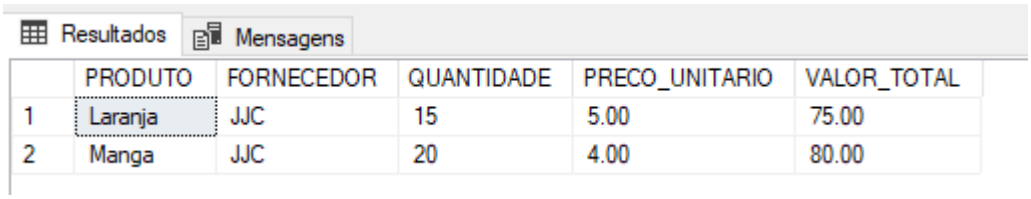
Este SELECT combina informações de pessoas e pessoas jurídicas



Resultados | Mensagens

	id_pessoa	nome_pessoa	logradouro_pessoa	cidade_pessoa	estado_pessoa	telefone_pessoa	email_pessoa	id_pessoa	cnpj
1	8	JJC	Rua 11, Centro	Riacho do Norte	PA	1212-1212	jjc@riacho.com	8	12345678901234

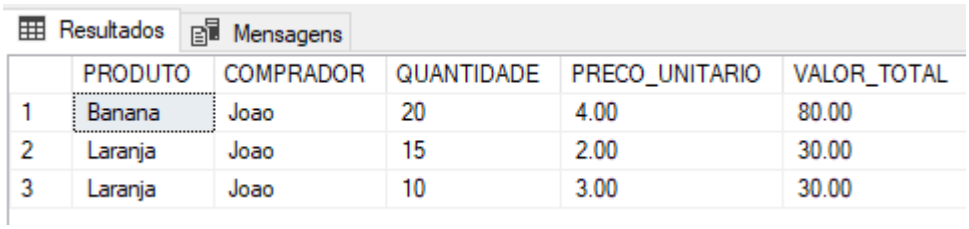
Movimentações de entrada, com produto, fornecedor, quantidade, preço unitário e valor total.



Resultados | Mensagens

	PRODUTO	FORNECEDOR	QUANTIDADE	PRECO_UNITARIO	VALOR_TOTAL
1	Laranja	JJC	15	5.00	75.00
2	Manga	JJC	20	4.00	80.00

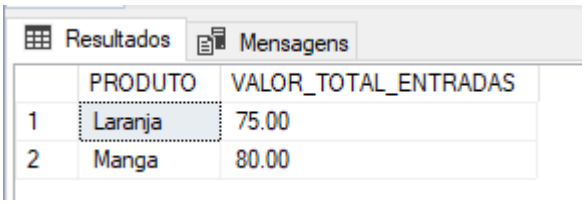
Movimentações de saída, com produto, comprador, quantidade, preço unitário e valor total.



Resultados | Mensagens

	PRODUTO	COMPRADOR	QUANTIDADE	PRECO_UNITARIO	VALOR_TOTAL
1	Banana	Joao	20	4.00	80.00
2	Laranja	Joao	15	2.00	30.00
3	Laranja	Joao	10	3.00	30.00

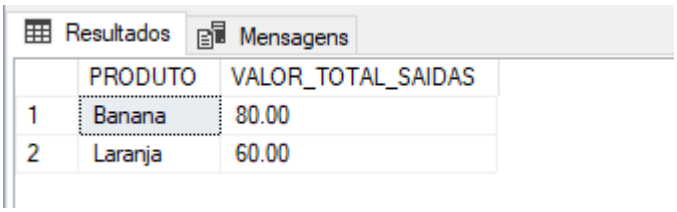
Valor total das entradas agrupadas por produto.



Resultados | Mensagens

	PRODUTO	VALOR_TOTAL_ENTRADAS
1	Laranja	75.00
2	Manga	80.00

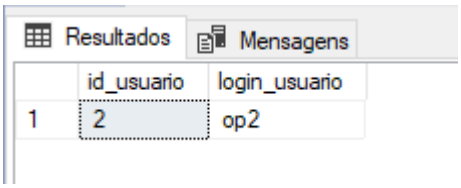
Valor total das saídas agrupadas por produto.



Resultados | Mensagens

	PRODUTO	VALOR_TOTAL_SAIDAS
1	Banana	80.00
2	Laranja	60.00

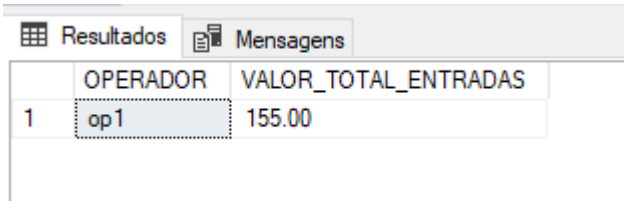
Operadores que não efetuaram movimentações de entrada (compra)



Resultados | Mensagens

	id_usuario	login_usuario
1	2	op2

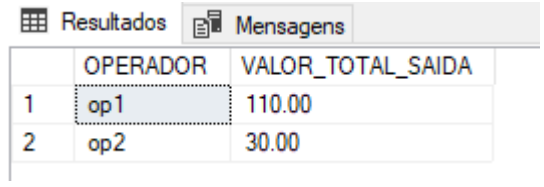
Valor total de entrada, agrupado por operador.



Resultados | Mensagens

	OPERADOR	VALOR_TOTAL_ENTRADAS
1	op1	155.00

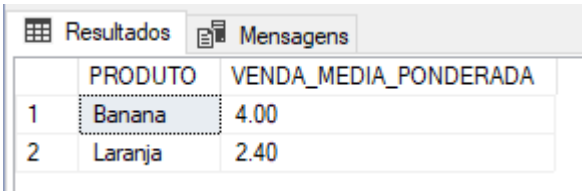
Valor total de saída, agrupado por operador.



Resultados | Mensagens

	OPERADOR	VALOR_TOTAL_SAIDA
1	op1	110.00
2	op2	30.00

```
Valor médio de venda por produto, utilizando média ponderada.
```



Resultados | Mensagens

	PRODUTO	VENDA_MEDIA_PONDERADA
1	Banana	4.00
2	Laranja	2.40

a) Diferenças entre SEQUENCE e IDENTITY:

SEQUENCE: É um objeto de banco de dados que gera valores sequenciais em um banco de dados SQL. A principal diferença é que as sequências não estão vinculadas a uma tabela específica e podem ser usadas em várias tabelas. O valor da sequência é gerado independentemente de qualquer inserção em uma tabela. Pode ser compartilhado entre várias tabelas e até mesmo entre bancos de dados.

IDENTITY: É uma propriedade de coluna usada em algumas bases de dados (incluindo o SQL Server) para gerar valores sequenciais exclusivos automaticamente quando linhas são inseridas em uma tabela. A diferença fundamental é que a propriedade IDENTITY está associada a uma coluna específica em uma tabela, e os valores são gerados automaticamente apenas quando novas linhas são inseridas nessa tabela.

b) Importância das Chaves Estrangeiras para a Consistência do Banco:

As chaves estrangeiras (foreign keys) são essenciais para manter a integridade referencial e a consistência do banco de dados. A importância das chaves estrangeiras inclui:

Integridade Referencial: As chaves estrangeiras garantem que os relacionamentos entre tabelas sejam consistentes. Elas garantem que os valores em uma tabela filha (referenciada) correspondam aos valores na tabela pai (referência). Isso evita que dados inconsistentes ou órfãos sejam inseridos no banco de dados.

Manutenção da Consistência dos Dados: As chaves estrangeiras ajudam a manter a consistência dos dados, garantindo que os relacionamentos entre as tabelas sejam mantidos de forma apropriada. Isso é fundamental para evitar erros e inconsistências nos dados.

Evita Deleções Indesejadas: As chaves estrangeiras podem ser configuradas para impor a regra de cascata, o que significa que, se um registro na tabela pai for excluído, os registros correspondentes na tabela filha também serão excluídos. Isso evita a exclusão acidental de dados relacionados.

c) Operadores do SQL relacionados à Álgebra Relacional e Cálculo Relacional:

Álgebra Relacional: Alguns dos operadores da álgebra relacional incluem SELEÇÃO (σ), PROJEÇÃO (π), UNIÃO (∪), INTERSEÇÃO (∩), DIFERENÇA (-), PRODUTO CARTESIANO (×), JUNÇÃO (JOIN) e DIVISÃO (÷).

Cálculo Relacional: O cálculo relacional não é uma linguagem baseada em operadores como a álgebra relacional, mas em predicados e lógica de primeira ordem. Não há uma correspondência direta entre os operadores da álgebra relacional e o cálculo relacional, mas os conceitos de consulta são semelhantes.

d) Agrupamento em Consultas e Requisito Obrigatório:

O agrupamento em consultas SQL é realizado usando a cláusula "GROUP BY." O requisito obrigatório ao usar "GROUP BY" é que todas as colunas na lista de projeção (SELECT) que não fazem parte de uma função de agregação devem estar presentes na cláusula "GROUP BY."

Em outras palavras, quando você deseja agrupar os resultados de uma consulta com base em determinadas colunas, todas as colunas que não são alvo de funções de agregação (como SUM, COUNT, AVG, etc.) devem ser listadas na cláusula "GROUP BY." Isso garante que a consulta seja semântica e logicamente correta, agrupando os resultados conforme desejado.