



Campos : Polo Barbosa II - Marília - SP

Curso : Desenvolvedor Full Stack

Turma : 2023.3

Integrante : Rafael Leal Altero

2º Procedimento :

Criando um banco de dados para um sistema de comércio eletrônico

Objetivo da Prática :

- 1 - Criar um banco de dados para um sistema de gerenciamento de estoque.
- 2 - Inserir dados básicos no banco de dados, como usuários, produtos e movimentações.
- 3 - Efetuar consultas nos dados inseridos para obter informações sobre pessoas físicas, pessoas jurídicas, movimentações de entrada e saída, valor total das entradas e saídas agrupadas por produto, operadores que não efetuaram movimentações de entrada, valor total de entrada e saída agrupado por operador, e valor médio de venda por produto.

segue o **LINK** 🖱️ **Codigos**

Códigos solicitados no roteiro de aula:

Link GitHub :

https://github.com/Rafa1a/DB_sql/tree/main/SQL

Nota : Para adicionar os dados no banco criei PROCEDURES que fazem certas averiguações antes da entrada de dados (principalmente em MOVIMENTAÇÕES), ou seja está além do que foi pedido, para aprender mais preferi fazer assim.

Resultados da execução dos códigos:

Inserir Usuário :

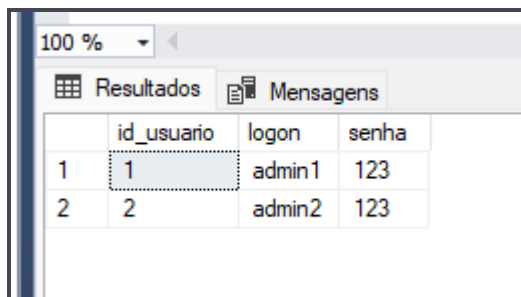
Adicionei um PROCEDURE que faz uma averiguação se o usuário já existe cadastrado no sistema, impedindo dados redundantes e repetidos.

```
--STORED PROCEDURE Usuario insert
CREATE PROCEDURE DBM.insert_usuario
@logon varchar(10) ,
@senha varchar(50)
AS
BEGIN
    IF NOT EXISTS(SELECT 1 FROM DBM.Usuario WHERE logon=@logon)
    BEGIN
        INSERT INTO DBM.Usuario(logon,senha)
        VALUES (@logon,@senha)
    END
    ELSE
    BEGIN
        RAISERROR('Usuario ja cadastrado',16,1)
    END
END
GO
```

Execute :

```
EXEC DBM.insert_usuario @logon = 'admin1', @senha = '123';
EXEC DBM.insert_usuario @logon = 'admin2', @senha = '123';
```

Resultado :



	id_usuario	logon	senha
1	1	admin1	123
2	2	admin2	123

Inserir Produtos :

Adicionei um PROCEDURE que faz uma averiguação se o produto já existe cadastrado no sistema, com base no nome, impedindo dados redundantes e repetidos.

```

--STORED PROCEDURE PRODUTO INSERT
CREATE PROCEDURE DBM.insert_produto
@nome varchar(50) ,
@quantidade int ,
@preco_de_venda decimal(18,2)
AS
BEGIN
    IF NOT EXISTS(SELECT 1 FROM DBM.Produtos WHERE nome=@nome)
    BEGIN
        INSERT INTO DBM.Produtos(nome,quantidade,preco_de_venda)
        VALUES (@nome,@quantidade,@preco_de_venda)
    END
    ELSE
    BEGIN
        RAISERROR('Produto ja cadastrado',16,1)
    END
END
GO

```

Execute :

```

EXEC DBM.insert_produto @nome = 'Maça', @quantidade = 10, @preco_de_venda = 10.00;
EXEC DBM.insert_produto @nome = 'Ovos', @quantidade = 12, @preco_de_venda = 5.00;
EXEC DBM.insert_produto @nome = 'Carne', @quantidade = 1, @preco_de_venda = 20.00;
EXEC DBM.insert_produto @nome = 'Frango', @quantidade = 1, @preco_de_venda = 15.00;
EXEC DBM.insert_produto @nome = 'Peixe', @quantidade = 1, @preco_de_venda = 10.00;
EXEC DBM.insert_produto @nome = 'Salmão', @quantidade = 1, @preco_de_venda = 20.00;
EXEC DBM.insert_produto @nome = 'Atum', @quantidade = 1, @preco_de_venda = 15.00;
EXEC DBM.insert_produto @nome = 'Caranguejo', @quantidade = 1, @preco_de_venda = 10.00;
EXEC DBM.insert_produto @nome = 'Cerveja', @quantidade = 6, @preco_de_venda = 5.00;
EXEC DBM.insert_produto @nome = 'Vinho', @quantidade = 3, @preco_de_venda = 10.00;

```

Resultado :

Resultados		Mensagens		
	id_produto	nome	quantidade	preco_de_venda
1	1	Maça	10	10.00
2	2	Ovos	12	5.00
3	3	Came	1	20.00
4	4	Frango	1	15.00
5	5	Peixe	1	10.00
6	6	Salmão	1	20.00
7	7	Atum	1	15.00
8	8	Caranguejo	1	10.00
9	9	Cerveja	6	5.00
10	10	Vinho	3	10.00

Inserir Pessoa CPF e CNPJ :

Possui 2 tabelas.

Adicionei o id_pessoa com o SEQUENCE dentro do PROCEDURE recomendado pela missão prática.

Adicionei também uma averiguação se a pessoa já existe cadastrada no sistema, com base no nome, impedindo dados redundantes e repetidos.

```
CREATE PROCEDURE DBM.insert_pessoa
@nome varchar(50),
@logradouro varchar(50),
@cidade varchar(20),
@estado varchar(3),
@telefone varchar(11),
@email varchar(35)
AS
BEGIN
    IF NOT EXISTS(SELECT 1 FROM DBM.Pessoa WHERE nome=@nome)
    BEGIN
        DECLARE @id_pessoa int;
        SET @id_pessoa = NEXT VALUE FOR sequencia;

        INSERT INTO DBM.Pessoa(id_pessoa, nome, logradouro,cidade,estado,telefone,email)
        VALUES (@id_pessoa,@nome,@logradouro,@cidade,@estado,@telefone,@email)
    END
    ELSE
    BEGIN
        RAISERROR('Pessoa já cadastrada',16,1)
    END
END
GO
```

Execute em Pessoa :

```
EXEC DBM.insert_pessoa @nome = 'João da Silva', @logradouro = 'Rua da Paz', @cidade = 'São Paulo',
@email = 'joao.da.silva@email.com';
EXEC DBM.insert_pessoa @nome = 'rafa', @logradouro = 'Rua da raiva', @cidade = 'São Paulo',
@email = 'rafa.da.raiva@email.com';
EXEC DBM.insert_pessoa @nome = 'renan', @logradouro = 'Rua da raiva', @cidade = 'São Paulo',
@email = 'rafa.da.raiva@email.com';
```

Inserir em Pessoa fisica e Juridica :

```
INSERT INTO DBM.Pessoa_fisica(id_cpf,id_pessoa) VALUES ('111122222',1);
INSERT INTO DBM.Pessoa_juridica(id_cnpj,id_pessoa) VALUES ('111111111',2);
```

Pessoa :

	id_pessoa	nome	logradouro	cidade	estado	telefone	email
1	1	João da Silva	Rua da Paz	São Paulo	SP	11999999999	joao.da.silva@email.com
2	2	rafa	Rua da raiva	São Paulo	SP	11999999669	rafa.da.raiva@email.com
3	3	renan	Rua da raiva	São Paulo	SP	11999999669	rafa.da.raiva@email.com

CPF e CNPJ :

	id_cpf	id_pessoa
1	111122222	1

	id_cnpj	id_pessoa
1	111111111	2

Inserir Movimentação “E” entrada e “S” saída :

- 1) Além de realizar a AVERIGUAÇÃO de cada FOREIGN “ID” se já existe cadastrado.
- 2) Realizar a averiguação para ver se a pessoa está cadastrada no cpf ou cnpj.
- 3) O código realiza uma condição que se a pessoa for física só pode realizar compra “S” saída de produtos e se for uma pessoa jurídica só pode realizar venda “E” entrada de produtos em “ESTOQUE”.
- 4) Caso seja “S” não poderá o preço de venda ser menor que o preço cadastrado (o recomendado para vender) ou seja você só poderá vender “S” se o preço for maior para ter lucro e caso seja “E” não poderá o preço de compra ser maior que o preço de venda recomendado.
- 5) Realiza uma operação em QUANTIDADE de produtos, um aumento na quantidade de produtos em ESTOQUE caso “E” entrada de novos itens e retirada de quantidade caso “S” saída de itens do estoque e se a saída for inferior a quantidade que tem em estoque ele não permite realizar a movimentação, ou seja se no estoque tem 1 e vender 3, não terá estoque para realizar a operação.

```
CREATE PROCEDURE DBM.insert_movimentacao
@id_pessoa int ,
@id_usuario int ,
@id_produto int ,
@tipo varchar(1) ,
@quantidade_E_S int ,
@preco_unitario decimal(18,2)
AS
BEGIN
--Tratamento do ID
IF NOT EXISTS (SELECT 1 FROM DBM.Pessoa WHERE id_pessoa=@id_pessoa)
BEGIN
RAISERROR('ID de PESSOA nao existe no banco de dados',16,1)
END
ELSE IF NOT EXISTS (SELECT 1 FROM DBM.Usuario WHERE id_usuario=@id_usuario)
BEGIN
RAISERROR('ID de USUARIO nao existe no banco de dados',16,1)
END
ELSE IF NOT EXISTS (SELECT 1 FROM DBM.Produtos WHERE id_produto=@id_produto)
BEGIN
RAISERROR('ID de PRODUTO nao existe no banco de dados',16,1)
END
--Tratamento do TIPO
ELSE IF @tipo = 'E'
BEGIN
IF EXISTS (SELECT 1 FROM DBM.Pessoa_fisica WHERE id_pessoa=@id_pessoa AND id_cpf IS not null)
BEGIN
RAISERROR('Uma pessoa fisica nao pode VENDER produtos',16,1)
END
ELSE IF EXISTS (SELECT 1 FROM DBM.Pessoa_juridica WHERE id_pessoa=@id_pessoa AND id_cnpj IS not null)
BEGIN
--Tratamento dos valores
IF NOT EXISTS (SELECT 1 FROM DBM.Produtos WHERE id_produto=@id_produto AND preco_de_venda > @preco_unitario)
BEGIN
RAISERROR('preco de COMPRA maior do que o preco de VENDA nao permitido',16,1)
END
ELSE
BEGIN
INSERT INTO DBM.Movimentacao(id_pessoa,id_usuario,id_produto,tipo,quantidade_E_S,preco_unitario )
VALUES (@id_pessoa,@id_usuario,@id_produto,@tipo,@quantidade_E_S,@preco_unitario)
--Tratamento da quantidade
UPDATE DBM.Produtos SET quantidade = quantidade + @quantidade_E_S WHERE id_produto = @id_produto
END
END
ELSE
BEGIN
RAISERROR('Pessoa nao cadastrada no cnpj, por favor realizar o cadastro',16,1)
END
END
END
```

Continuação.... :

```

ELSE IF @tipo = 'S'
BEGIN
    IF EXISTS (SELECT 1 FROM DBM.Pessoa_fisica WHERE id_pessoa=@id_pessoa AND id_cpf IS not null)
    BEGIN
        --Tratamento dos valores
        IF NOT EXISTS (SELECT 1 FROM DBM.Produtos WHERE id_produto=@id_produto AND preco_de_venda < @preco_unitario)
        BEGIN
            RAISERROR('preco de VENDA menor do que o preco de COMPRA, nao permitido',16,1)
        END
    ELSE
    BEGIN
        --Tratamento da quantidade
        IF EXISTS(SELECT 1 FROM DBM.Produtos WHERE quantidade - @quantidade_E_S < 0 AND id_produto = @id_produto )
        BEGIN
            RAISERROR('Quantidade no Estoque menor do que Quantidade da venda',16,1)
        END
    ELSE
    BEGIN
        INSERT INTO DBM.Movimentacao(id_pessoa,id_usuario,id_produto,tipo,quantidade_E_S,preco_unitario )
        VALUES (@id_pessoa,@id_usuario,@id_produto,@tipo,@quantidade_E_S,@preco_unitario)
        UPDATE DBM.Produtos SET quantidade = quantidade - @quantidade_E_S WHERE id_produto = @id_produto

        END
    END
END
ELSE IF EXISTS (SELECT 1 FROM DBM.Pessoa_juridica WHERE id_pessoa=@id_pessoa AND id_cnpj IS not null)
BEGIN
    RAISERROR('Uma pessoa Juridica nao pode COMPRAR produtos',16,1)
END
ELSE
BEGIN
    RAISERROR('Pessoa nao cadastrada no cpf, por favor realizar o cadastro',16,1)
END
END
END
GO

```

Execute :

```

EXEC DBM.insert_movimentacao @id_pessoa = 2, @id_usuario = 1,
@id_produto = 4, @tipo = 'E', @quantidade_E_S = 1, @preco_unitario = 12.00;
EXEC DBM.insert_movimentacao @id_pessoa = 1, @id_usuario = 2,
@id_produto = 2, @tipo = 'E', @quantidade_E_S = 1, @preco_unitario = 12.00;
EXEC DBM.insert_movimentacao @id_pessoa = 2, @id_usuario = 1,
@id_produto = 4, @tipo = 'S', @quantidade_E_S = 1, @preco_unitario = 12.00;
EXEC DBM.insert_movimentacao @id_pessoa = 1, @id_usuario = 2,
@id_produto = 3, @tipo = 'S', @quantidade_E_S = 1, @preco_unitario = 25.00;
EXEC DBM.insert_movimentacao @id_pessoa = 3, @id_usuario = 1,
@id_produto = 1, @tipo = 'S', @quantidade_E_S = 1, @preco_unitario = 12.00;

```

Resultado :

O primeiro dá certo e adiciona

O SEGUNDO dá errado pois a pessoa 1 é uma pessoa física e não pode “E” fazer uma entrada.

O TERCEIRO dá errado pois a pessoa 2 é uma pessoa jurídica e não pode “S” fazer uma saída.

O quarto dá certo e adiciona.

O QUINTO dá errado pois o id da pessoa 3 não existe cadastro nem no cpf ou cnpj.

	id_movimentacao	id_pessoa	id_usuario	id_produto	tipo	quantidade_E_S	preco_unitario
1	1	2	1	4	E	1	12.00
2	2	1	2	3	S	1	25.00

Consultas Realizadas na tabela :

Dados completos de pessoas físicas :

Execute :

```
SELECT * FROM DBM.Pessoa JOIN DBM.Pessoa_fisica  
ON Pessoa.id_pessoa = Pessoa_fisica.id_pessoa
```

Resultado:

	id_pessoa	nome	logradouro	cidade	estado	telefone	email	id_cpf	id_pessoa
1	1	João da Silva	Rua da Paz	São Paulo	SP	11999999999	joao.da.silva@email.com	1111222222	1

Dados completos de pessoas jurídicas:

Execute :

```
SELECT * FROM DBM.Pessoa JOIN DBM.Pessoa_juridica  
ON Pessoa.id_pessoa = Pessoa_juridica.id_pessoa
```

Resultado:

	id_pessoa	nome	logradouro	cidade	estado	telefone	email	id_cnpj	id_pessoa
1	2	rafa	Rua da raiva	São Paulo	SP	11999999669	rafa.da.raiva@email.com	1111111111	2

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

Execute :

```
SELECT * FROM DBM.Movimentacao WHERE tipo = 'E'
```

Resultado:

	id_movimentacao	id_pessoa	id_usuario	id_produto	tipo	quantidade_E_S	preco_unitario
1	1	2	1	4	E	1	12.00

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

Execute :

```
SELECT * FROM DBM.Movimentacao WHERE tipo = 'S'
```

Resultado:

	id_movimentacao	id_pessoa	id_usuario	id_produto	tipo	quantidade_E_S	preco_unitario
1	2	1	2	3	S	1	25.00

Valor total das entradas agrupadas por produto.

Execute :

```
--valor total de "E"
SELECT id_produto, SUM(preco_unitario * quantidade_E_S) AS valor_total
FROM DBM.Movimentacao
WHERE tipo = 'E'
GROUP BY id_produto
ORDER BY id_produto;
```

Da tabela :

Adicionei alguns valores para teste !

	id_movimentacao	id_pessoa	id_usuario	id_produto	tipo	quantidade_E_S	preco_unitario
1	1	2	1	4	E	1	12.00
2	3	2	1	4	E	1	12.00
3	4	2	1	4	E	1	12.00
4	5	2	1	5	E	1	2.00
5	6	2	1	5	E	1	2.00

Resultado:

	id_produto	valor_total
1	4	36.00
2	5	4.00

Valor total das saídas agrupadas por produto.

Execute :

```
SELECT id_produto, SUM(preco_unitario * quantidade_E_S) AS valor_total
FROM DBM.Movimentacao
WHERE tipo = 'S'
GROUP BY id_produto
ORDER BY id_produto;
```

Da tabela :

Adicionei alguns valores para teste !

	id_movimentacao	id_pessoa	id_usuario	id_produto	tipo	quantidade_E_S	preco_unitario
1	1	1	2	4	S	1	25.00
2	2	1	2	1	S	1	25.00
3	3	1	2	1	S	1	25.00
4	4	1	2	6	S	1	25.00

Resultado:

	id_produto	valor_total
1	1	50.00
2	4	25.00
3	6	25.00

Operadores que não efetuaram movimentações de entrada (compra).

Execute :

```
SELECT logon
FROM DBM.Usuario
WHERE id_usuario NOT IN (
    SELECT id_usuario
    FROM DBM.Movimentacao
    WHERE tipo = 'E'
);
```

Resultado:

	logon
1	admin2

Valor total de entrada, agrupado por operador.

Tabela referência teste :

	id_movimentacao	id_pessoa	id_usuario	id_produto	tipo	quantidade_E_S	preco_unitario
1	1	1	2	4	S	1	25.00
2	2	1	2	1	S	1	25.00
3	3	1	2	1	S	1	25.00
4	4	1	2	6	S	1	25.00
5	5	2	1	5	E	1	2.00
6	6	2	1	5	E	1	2.00
7	7	2	1	5	E	1	2.00

Execute :

```

SELECT Movimentacao.id_usuario, Usuario.logon, SUM(preco_unitario * quantidade_E_S) AS valor_total
FROM DBM.Movimentacao
JOIN DBM.Usuario ON Movimentacao.id_usuario = Usuario.id_usuario
WHERE tipo = 'E'
GROUP BY Movimentacao.id_usuario, Usuario.logon;

```

Resultado:

	id_usuario	logon	valor_total
1	1	admin1	6.00

Valor total de saída, agrupado por operador.

Execute :

```

SELECT Movimentacao.id_usuario, Usuario.logon, SUM(preco_unitario * quantidade_E_S) AS valor_total
FROM DBM.Movimentacao
JOIN DBM.Usuario ON Movimentacao.id_usuario = Usuario.id_usuario
WHERE tipo = 'S'
GROUP BY Movimentacao.id_usuario, Usuario.logon;

```

Resultado:

	id_usuario	logon	valor_total
1	2	admin2	100.00

Valor médio de venda por produto, utilizando média ponderada.

Tabela referência :

	id_movimentacao	id_pessoa	id_usuario	id_produto	tipo	quantidade_E_S	preco_unitario
1	1	1	2	4	S	1	25.00
2	2	1	2	1	S	1	25.00
3	3	1	2	1	S	1	25.00
4	4	1	2	6	S	1	25.00
5	5	2	1	5	E	1	2.00
6	6	2	1	5	E	1	2.00
7	7	2	1	5	E	1	2.00
8	8	1	2	7	S	1	40.00
9	9	2	1	5	E	1	7.00

Execute :

```

SELECT id_produto, AVG(preco_unitario * quantidade_E_S) AS valor_medio
FROM DBM.Movimentacao
WHERE tipo = 'S'
GROUP BY id_produto;

```

Resultado:

	id_produto	valor_medio
1	1	25.000000
2	4	25.000000
3	6	25.000000
4	7	40.000000

Análise e Conclusão:

a) Quais as diferenças no uso de sequence e identity?

Uma sequência é um objeto do banco de dados que gera números. Uma identidade é uma coluna de tabela que gera números. As sequências podem ser usadas por várias tabelas, enquanto as identidades só podem ser usadas pela tabela em que são criadas.

b) Qual a importância das chaves estrangeiras para a consistência do banco?

As chaves estrangeiras são responsáveis pelo relacionamento entre tabelas e herança sendo essencial para a criação e manutenção das tabelas.

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

Os seguintes operadores do SQL pertencem à álgebra relacional:

- **SELECT:** Seleciona linhas de uma tabela com base em certos critérios.
- **JOIN:** Combina duas ou mais tabelas de acordo com valores comuns em certas colunas.
- **UNION:** Combina linhas de duas ou mais tabelas em uma nova tabela.
- **INTERSECT:** Combina linhas que estão presentes em duas ou mais tabelas em uma nova tabela.
- **EXCEPT:** Combina linhas que estão presentes em uma tabela, mas não em outra, em uma nova tabela.
- **DISTINCT:** Remove duplicatas de uma tabela.
- **GROUP BY:** Agrupa linhas de uma tabela com base em certos critérios e calcula uma função de agregação para cada grupo.

Os seguintes operadores do SQL são definidos no cálculo relacional:

- **COUNT:** Conta o número de linhas em uma tabela.
- **SUM:** Calcula a soma dos valores em uma coluna.
- **AVG:** Calcula a média dos valores em uma coluna.
- **MIN:** Calcula o valor mínimo em uma coluna.
- **MAX:** Calcula o valor máximo em uma coluna.

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

O agrupamento em consultas é feito usando a cláusula **GROUP BY**.

O requisito obrigatório para o agrupamento é que as colunas especificadas na cláusula **GROUP BY** devem ser únicas para cada grupo. Isso significa que nenhuma linha pode ser agrupada em mais de um grupo.

Basicamente você escolhe qual coluna vai ser agrupada normalmente com os cálculos como sum, count, min,max e avg.