Missão Prática | Nível 2 | Mundo 3

Bernardo Oliveira Ramos -202307212041

Polo Silva Lobo
Desenvolvedor Full stack – mundo 3 periiodo 2024.3 – Semestre Letivo

Objetivo da Prática

Descreva nessa seção qual o objetivo da sua prática. Todos os Relatórios de Práticas deverão ser confeccionados em arquivo no formato PDF, com a **Logo da Universidade**, **nome do Campus**, **nome do Curso**, **nome da Disciplina**, **número da Turma**, **semestre letivo**. Além disso, o projeto deve ser armazenado em um repositório no GIT e o respectivo endereço deve constar na documentação e essa documentação deve estar no no GIT. O código deve estar versionado no GIT de forma organizada.

Lembre-se que a organização contará pontos.

Esse template é um modelo a ser seguido. O aluno pode optar por seguir outro modelo, **desde que atenda a todas as etapas disponíveis na Missão Prática**. O documento final deve estar em pdf.

1º Procedimento | Criando o Banco de Dados

Inserir neste campo, **de forma organizada**, todos os códigos do roteiro do 1º Procedimento da Atividade Prática, os resultados da execução do código e a Análise e

Criação de Sequence para Pessoa

```
CREATE SEQUENCE seq_pessoa_id
  START WITH 1
  INCREMENT BY 1;
```

Criação das Tabelas

Tabela: pessoas

```
CREATE TABLE pessoas (
   id_pessoa INT PRIMARY KEY DEFAULT NEXT VALUE FOR seq_pessoa_id,
   endereco VARCHAR(200),
   telefone VARCHAR(15),
   email VARCHAR(100)
);
```

Tabela: pessoa_fisica

```
CREATE TABLE pessoa_fisica (
   id_pessoa_fisica INT PRIMARY KEY,
   cpf VARCHAR(11) UNIQUE NOT NULL,
   nome VARCHAR(100) NOT NULL,
   data_nascimento DATE,
   CONSTRAINT FK_pessoa_fisica FOREIGN KEY (id_pessoa_fisica) REFERENCES pessoas(id_pessoa)
);
```

Tabela: pessoa_juridica

```
CREATE TABLE pessoa_juridica (
   id_pessoa_juridica INT PRIMARY KEY,
   cnpj VARCHAR(14) UNIQUE NOT NULL,
   razao_social VARCHAR(150) NOT NULL,
   nome_fantasia VARCHAR(150),
```

```
    CONSTRAINT FK_pessoa_juridica FOREIGN KEY (id_pessoa_juridica) REFERENCES pessoas(id_pessoa)
);
```

Tabela: usuarios

```
CREATE TABLE usuarios (
    id_usuario INT IDENTITY(1,1) PRIMARY KEY,
    nome_usuario VARCHAR(100) NOT NULL,
    login VARCHAR(50) NOT NULL,
    senha VARCHAR(50) NOT NULL
);
```

Tabela: produtos

```
CREATE TABLE produtos (
    id_produto INT IDENTITY(1,1) PRIMARY KEY,
    nome_produto VARCHAR(100) NOT NULL,
    quantidade_estoque INT DEFAULT 0,
    preco_venda DECIMAL(10,2) NOT NULL
);
```

Tabela: movimentos_compra

```
CREATE TABLE movimentos_compra (
    id_compra INT IDENTITY(1,1) PRIMARY KEY,
    id_usuario INT NOT NULL,
    id_produto INT NOT NULL,
    id_fornecedor INT NOT NULL,
    data_compra DATE NOT NULL,
    quantidade INT NOT NULL,
    preco_unitario DECIMAL(10,2) NOT NULL,
```

```
    CONSTRAINT FK_usuario_compra FOREIGN KEY (id_usuario) REFERENCES usuarios(id_usuario),
    CONSTRAINT FK_produto_compra FOREIGN KEY (id_produto) REFERENCES produtos(id_produto),
    CONSTRAINT FK_fornecedor_compra FOREIGN KEY (id_fornecedor) REFERENCES pessoa_juridica(id_pessoa_juridica)
);
```

Tabela: movimentos_venda

```
CREATE TABLE movimentos_venda (
    id_venda INT IDENTITY(1,1) PRIMARY KEY,
    id_usuario INT NOT NULL,
    id_produto INT NOT NULL,
    id_cliente INT NOT NULL,
    data_venda DATE NOT NULL,
    quantidade INT NOT NULL,
    preco_venda DECIMAL(10,2) NOT NULL,
    CONSTRAINT FK_usuario_venda FOREIGN KEY (id_usuario) REFERENCES usuarios(id_usuario),
    CONSTRAINT FK_produto_venda FOREIGN KEY (id_produto) REFERENCES produtos(id_produto),
    CONSTRAINT FK_cliente_venda FOREIGN KEY (id_cliente) REFERENCES pessoa_fisica(id_pessoa_fisica)
);
```

Conclusão:

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

 1x1 (Um para Um):

- Uma linha em uma tabela corresponde exatamente a uma linha em outra.
- Implementação: A **chave primária (PK)** de uma tabela é também usada como **chave estrangeira (FK)** na outra.
- Exemplo: Tabelas pessoas e pessoa_fisica. A chave primária id_pessoa em pessoas é a mesma usada como chave estrangeira na tabela pessoa_fisica.

 1xN (Um para Muitos):

- Uma linha na tabela de origem pode estar relacionada a várias linhas na tabela de destino.
- Implementação: A tabela de destino possui uma **chave estrangeira (FK)** que referencia a **chave primária (PK)** da tabela de origem.
- Exemplo: A tabela usuarios pode estar relacionada com várias entradas na tabela movimentos_compra via a FK id_usuario.

 N

(Muitos para Muitos):

- Implementado por meio de uma **tabela associativa** que contém duas ou mais **chaves estrangeiras**.
- Exemplo: Em alguns sistemas, vendas e produtos poderiam ter um relacionamento N

, mas aqui foi simplificado como 1

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

O relacionamento que representa a **herança** em bancos de dados relacionais é geralmente implementado por meio de:

- **Tabelas com Relacionamento 1x1 (Especialização):**
    - Cada classe "filha" (especialização) é representada por uma tabela separada, e a **chave primária** da tabela pai é utilizada como **chave estrangeira** nas tabelas filhas.
    - **Exemplo:**
        - A tabela pessoas é a "tabela pai".
        - As tabelas pessoa_fisica e pessoa_juridica são especializações que usam o id_pessoa como FK, representando a herança.
- **Single Table Inheritance (STI):**
    - Outra abordagem é utilizar **uma única tabela** com uma coluna indicativa para o tipo de entidade (Ex.: tipo_pessoa).

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

Observe que os tópicos acima seguem exatamente o que está na Atividade Prática exigida.

O **SQL Server Management Studio (SSMS)** melhora a produtividade de várias maneiras:

1. **Interface Gráfica Intuitiva:**
    - Permite criar, editar e gerenciar bancos de dados, tabelas e relacionamentos através de uma interface amigável, sem a necessidade de escrever comandos SQL o tempo todo.
2. **Editor de SQL Avançado:**
    - Facilita a escrita e execução de scripts SQL, com destaque de sintaxe, auto-completar e sugestões de código.
3. **Ferramentas de Verificação e Diagnóstico:**

- Possui ferramentas como **Query Execution Plan** e **Query Profiler**, que ajudam a identificar gargalos e melhorar a performance das consultas.

4. **Automação de Tarefas:**
   - Tarefas rotineiras, como **backups automáticos** e **restauração de bancos**, podem ser agendadas via **SQL Server Agent**.
5. **Gestão de Permissões e Segurança:**
   - Facilita a criação e controle de logins e usuários, com gerenciamento detalhado de permissões para diferentes funções.
6. **Exportação e Importação Simples:**
   - Possui assistentes para **exportação** e **importação de dados** e scripts SQL, otimizando a portabilidade do banco de dados.

2º Procedimento | Alimentando a Base

Inserir neste campo, **de forma organizada**, todos os códigos do roteiro do 2º Procedimento da Atividade Prática, os resultados da execução do código e a Análise e

```
INSERT INTO usuarios (login, senha)
VALUES
  ('op1', 'op1'),
  ('op2', 'op2');


INSERT INTO produtos (nome_produto, quantidade_estoque, preco_venda)
VALUES
  ('Banana', 100, 5.00),
  ('Laranja', 500, 2.00),
  ('Manga', 800, 4.00);


DECLARE @idPessoa INT;
SET @idPessoa = NEXT VALUE FOR seq_pessoa_id;
```

```
INSERT INTO pessoas (id_pessoa, endereco, telefone, email)
VALUES (@idPessoa, 'Rua 12, casa 3, Quitanda', '1111-1111', 'joao@riacho.com');

INSERT INTO pessoa_fisica (id_pessoa_fisica, cpf, nome, data_nascimento)
VALUES (@idPessoa, '11111111111', 'Joao', '1985-01-01');

DECLARE @idPessoaJur INT;
SET @idPessoaJur = NEXT VALUE FOR seq_pessoa_id;

INSERT INTO pessoas (id_pessoa, endereco, telefone, email)
VALUES (@idPessoaJur, 'Avenida Central, 100', '2222-2222', 'empresa@exemplo.com');

INSERT INTO pessoa_juridica (id_pessoa_juridica, cnpj, razao_social, nome_fantasia)
VALUES (@idPessoaJur, '22222222222222', 'Empresa ABC Ltda', 'ABC');

INSERT INTO movimentos_compra (id_usuario, id_produto, id_fornecedor, data_compra, quantidade, preco_unitario)
VALUES
    (1, 1, @idPessoaJur, '2024-10-10', 50, 4.50),
    (2, 2, @idPessoaJur, '2024-10-12', 100, 1.80);
```

```
INSERT INTO movimentos_venda (id_usuario, id_produto, id_cliente, data_venda, quantidade, preco_venda)
VALUES
    (1, 1, @idPessoa, '2024-10-13', 10, 5.00),
    (2, 3, @idPessoa, '2024-10-14', 50, 4.00);
```

Consultas

Consulta 1: Dados Completos de Pessoas Físicas

```
SELECT pf.*, p.endereco, p.telefone, p.email
FROM pessoa_fisica pf
JOIN pessoas p ON pf.id_pessoa_fisica = p.id_pessoa;
```

Consulta 2: Dados Completos de Pessoas Jurídicas

```
SELECT pj.*, p.endereco, p.telefone, p.email
FROM pessoa_juridica pj
JOIN pessoas p ON pj.id_pessoa_juridica = p.id_pessoa;
```

Consulta 3: Movimentações de Entrada (Compra)

```
SELECT mc.id_compra, pr.nome_produto, pj.razao_social AS fornecedor, mc.quantidade,
    mc.preco_unitario, (mc.quantidade * mc.preco_unitario) AS valor_total
FROM movimentos_compra mc
JOIN produtos pr ON mc.id_produto = pr.id_produto
JOIN pessoa_juridica pj ON mc.id_fornecedor = pj.id_pessoa_juridica;
```

Consulta 4: Movimentações de Saída (Venda)

```
SELECT mv.id_venda, pr.nome_produto, pf.nome AS comprador, mv.quantidade,
    mv.preco_venda, (mv.quantidade * mv.preco_venda) AS valor_total
FROM movimentos_venda mv
JOIN produtos pr ON mv.id_produto = pr.id_produto
JOIN pessoa_fisica pf ON mv.id_cliente = pf.id_pessoa_fisica;
```

Consulta 5: Valor Total das Entradas Agrupadas por Produto

```
SELECT pr.nome_produto, SUM(mc.quantidade * mc.preco_unitario) AS total_entrada
FROM movimentos_compra mc
JOIN produtos pr ON mc.id_produto = pr.id_produto
GROUP BY pr.nome_produto;
```

Consulta 6: Valor Total das Saídas Agrupadas por Produto

```
SELECT pr.nome_produto, SUM(mv.quantidade * mv.preco_venda) AS total_saida
FROM movimentos_venda mv
JOIN produtos pr ON mv.id_produto = pr.id_produto
GROUP BY pr.nome_produto;
```

Consulta 7: Operadores que Não Efetuaram Movimentações de Entrada

```
SELECT u.nome_usuario
FROM usuarios u
LEFT JOIN movimentos_compra mc ON u.id_usuario = mc.id_usuario
WHERE mc.id_usuario IS NULL;
```

Consulta 8: Valor Total de Entrada Agrupado por Operador

```
SELECT u.nome_usuario, SUM(mc.quantidade * mc.preco_unitario) AS total_entrada
FROM movimentos_compra mc
JOIN usuarios u ON mc.id_usuario = u.id_usuario
GROUP BY u.nome_usuario;
```

Consulta 9: Valor Total de Saída Agrupado por Operador

```
SELECT u.nome_usuario, SUM(mv.quantidade * mv.preco_venda) AS total_saida
FROM movimentos_venda mv
JOIN usuarios u ON mv.id_usuario = u.id_usuario
GROUP BY u.nome_usuario;
```

Consulta 10: Valor Médio de Venda por Produto (Média Ponderada)

SELECT pr.nome_produto, SUM(mv.quantidade * mv.preco_venda) / SUM(mv.quantidade) AS media_ponderada FROM movimentos_venda mv JOIN produtos pr ON mv.id_produto = pr.id_produto GROUP BY pr.nome_produto;

Conclusão:

a. Quais as diferenças no uso de *sequence* e *identity*?

SEQUENCE: Objeto independente, flexível e reutilizável.

IDENTITY: Atributo específico para uma coluna em uma tabela, com menos controle.

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

 Chaves Estrangeiras (Foreign Keys) garantem **integridade referencial** no banco de dados.

 Elas fazem com que um valor em uma coluna de uma tabela (chave estrangeira) **precise existir na tabela relacionada** como uma chave primária.

 Importância para a consistência:

1. **Evita inserções inválidas:** Não permite inserir um valor que não exista na tabela de referência.
2. **Preserva relações:** Garante que as tabelas mantenham conexões corretas entre os dados.
3. **Evita exclusões inconsistentes:** Impede que uma linha referenciada por outra seja excluída, a menos que seja tratada (como com **ON DELETE CASCADE**).
4. **Manutenção da integridade:** Evita dados órfãos ou inconsistentes entre tabelas relacionadas.

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

Seleção (SELECT): Filtra linhas com base em condições.

SELECT * FROM produtos WHERE preco_venda > 100;

Projeção: Seleciona colunas específicas

SELECT nome_produto, preco_venda FROM produtos;

União (UNION): Combina resultados de duas consultas

SELECT nome FROM pessoa_fisica

UNION

SELECT nome_fantasia FROM pessoa_juridica;

Junção (JOIN): Combina tabelas com base em uma condição comum

SELECT u.nome_usuario, m.data_venda

FROM usuarios u

JOIN movimentos_venda m ON u.id_usuario = m.id_usuario;

Diferença: Retorna a diferença entre duas consultas.

SELECT id_produto FROM produtos

EXCEPT

SELECT id_produto FROM movimentos_venda;

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

 Agrupamento em SQL é feito com a cláusula **GROUP BY**, que permite agrupar linhas com valores comuns em uma ou mais colunas.

 Uso de Agrupamento:

- O **GROUP BY** agrupa os dados e permite que funções de agregação (como SUM, COUNT, AVG, etc.) sejam aplicadas a cada grupo.

Observe que os tópicos acima seguem exatamente o que está na Atividade Prática exigida.