Lista de Exercícios: Aplicações JDBC com CRUD Completo

Objetivo: Praticar o ciclo completo de persistência de dados (CRUD) em Java. Cada exercício exige a criação de uma aplicação de console interativa que permite ao usuário gerenciar uma entidade específica no banco de dados H2.

Requisitos Chave para TODOS os Exercícios:

1. **Interação com o Usuário:** A aplicação deve ser controlada por um menu no console (ex: um loop while com um switch ou if-else) que usa a classe Scanner para receber as escolhas e os dados do usuário.
2. **Segurança (Obrigatório):** Todo e qualquer comando SQL que envolva dados fornecidos pelo usuário (INSERT, UPDATE, DELETE, SELECT...WHERE) **DEVE** utilizar PreparedStatement para prevenir SQL Injection.
3. **Gerenciamento de Recursos:** Utilize a estrutura try-with-resources para gerenciar seus objetos Connection, PreparedStatement e ResultSet, garantindo que sejam fechados automaticamente.
4. **Banco de Dados:** Todos os exercícios devem rodar sobre o banco de dados H2 que configuramos (jdbc:h2:./...).

Exercício 1: Controle de Estoque da Loja

Contexto: Você precisa criar um sistema simples de controle de estoque para uma pequena loja. A entidade principal é o Produto.

- **Entidade:** Crie a classe Produto com id (int), nome (String), preco (double) e quantidade (int).
- **Tabela SQL (execute no início):**

```
CREATE TABLE IF NOT EXISTS produto (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nome VARCHAR(255) NOT NULL,
  preco DOUBLE,
  quantidade INT
);
```

- **Aplicação (Menu):** Crie uma classe Main que ofereça as seguintes opções ao usuário:
    1. **Adicionar Produto (Create):** Pede ao usuário o nome, preço e quantidade. Salva um novo produto no banco.
    2. **Listar Todos os Produtos (Read):** Executa um SELECT * e imprime todos os produtos cadastrados.
    3. **Atualizar Preço (Update):** Pede ao usuário o id de um produto e o novoPreco. Executa um UPDATE para alterar o preço daquele produto.
    4. **Deletar Produto (Delete):** Pede ao usuário o id de um produto e o remove do banco.
    5. **Sair.**

Exercício 2: Mini-CRM de Clientes

Contexto: Um escritório de advocacia precisa de um sistema básico para gerenciar sua lista de clientes (um mini-CRM).

- **Entidade:** Crie a classe Cliente com id (int), nome (String), email (String) e telefone (String).

- **Tabela SQL (execute no início):**

```
CREATE TABLE IF NOT EXISTS cliente (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nome VARCHAR(255) NOT NULL,
  email VARCHAR(255) UNIQUE,
  telefone VARCHAR(20)
);
```

- **Aplicação (Menu):** Crie uma classe Main com as opções:
    1. **Cadastrar Cliente (Create):** Pede nome, email e telefone e salva no banco.
    2. **Listar Todos (Read):** Exibe todos os clientes.
    3. **Buscar Cliente por Email (Read):** Pede um email, executa um SELECT...WHERE email = ? e exibe os dados do cliente encontrado (ou uma mensagem se não encontrar).
    4. **Atualizar Telefone (Update):** Pede o id do cliente e um novoTelefone, e atualiza o registro.
    5. **Remover Cliente (Delete):** Pede o id do cliente e o remove.
    6. **Sair.**

Exercício 3: Gerenciador de Tarefas (To-Do List)

Contexto: Crie uma aplicação de linha de comando para gerenciar uma lista de tarefas pessoais.

- **Entidade:** Crie a classe Tarefa com id (int), descricao (String) e status (String: "PENDENTE" ou "CONCLUIDA").
- **Tabela SQL (execute no início):**

```
CREATE TABLE IF NOT EXISTS tarefa (
  id INT AUTO_INCREMENT PRIMARY KEY,
  descricao VARCHAR(255) NOT NULL,
  status VARCHAR(20) DEFAULT 'PENDENTE'
);
```

- **Aplicação (Menu):** Crie uma classe Main com as opções:
    1. **Adicionar Nova Tarefa (Create):** Pede apenas a descricao. O status deve ser "PENDENTE" por padrão.
    2. **Listar Todas as Tarefas (Read):** Exibe todas, mostrando ID, descrição e status.
    3. **Listar Apenas Pendentes (Read):** Executa um SELECT...WHERE status = 'PENDENTE'.
    4. **Marcar Tarefa como Concluída (Update):** Pede o id da tarefa e executa um UPDATE para mudar o status para "CONCLUIDA".
    5. **Excluir Tarefa (Delete):** Pede o id e remove a tarefa.
    6. **Sair.**

Exercício 4: Videoteca Pessoal

Contexto: Um colecionador de filmes quer um sistema simples para catalogar sua coleção de Blu-rays.

- **Entidade:** Crie a classe Filme com id (int), titulo (String), diretor (String) e anoLancamento (int).
- **Tabela SQL (execute no início):**

```
CREATE TABLE IF NOT EXISTS filme (
```

```
id INT AUTO_INCREMENT PRIMARY KEY,
titulo VARCHAR(255) NOT NULL,
diretor VARCHAR(255),
ano_lancamento INT
);
```

- **Aplicação (Menu):** Crie uma classe Main com as opções:
    1. **Catalogar Filme (Create):** Pede título, diretor e ano e salva no banco.
    2. **Listar Coleção (Read):** Exibe todos os filmes.
    3. **Buscar por Título (Read):** Pede uma parte do título e executa um SELECT...WHERE titulo LIKE ?. (Dica: pstmt.setString(1, "%" + termoBusca + "%");).
    4. **Atualizar Diretor (Update):** Pede o id do filme e um novoDiretor, e atualiza o registro.
    5. **Remover Filme (Delete):** Pede o id e remove o filme.
    6. **Sair.**

Exercício 5: Pátio de Veículos (Desafio com Chave Primária Diferente)

Contexto: Uma pequena concessionária de carros usados precisa gerenciar seu pátio. O identificador único de um veículo não é um ID numérico, mas sim sua placa.

- **Entidade:** Crie a classe Veiculo com placa (String), marca (String), modelo (String) e ano (int).
- **Tabela SQL (execute no início):**

```
CREATE TABLE IF NOT EXISTS veiculo (
placa VARCHAR(10) PRIMARY KEY, -- A Placa é a Chave Primária!
marca VARCHAR(100),
modelo VARCHAR(100),
ano INT
);
```

- **Aplicação (Menu):** Crie uma classe Main com as opções:
    1. **Cadastrar Veículo (Create):** Pede placa, marca, modelo e ano. Salva no banco.
    2. **Listar Pátio (Read):** Exibe todos os veículos.
    3. **Buscar por Placa (Read):** Pede a placa e exibe os dados do veículo.
    4. **Atualizar Ano (Update):** Pede a placa e o novoAno, e atualiza o registro.
    5. **Vender Veículo (Delete):** Pede a placa e remove o veículo do banco.
    6. **Sair.**