▶️ Procedimento 1 - Mapeamento Objeto-Relacional e DAO

✅ Objetivos

O projeto da Missão Prática foi desenvolvido a título de um exercício completo de integração entre uma aplicação Java padrão (modelo Ant) e um banco de dados SQL Server. Ao longo da implementação, foi possível explorar conceitos fundamentais de JDBC, POO (Programação Orientada a Objetos) e manipulação de dados relacionais com persistência em tabelas normalizadas.

🧱 Estrutura do Projeto

O projeto foi dividido de forma modular, observando-se a separação de responsabilidades:

- **Modelo de domínio** (Pessoa, PessoaFisica, PessoaJuridica)
- **DAO (Data Access Object)** para encapsular a lógica de acesso aos dados
- **Utilitários** (ConectorBD, SequenceManager) para conexão e controle de sequência
- **Classe de testes** (CadastroBDTeste) para validar todas as operações CRUD

⚙️ Funcionalidades

1. **Conexão com o banco**: A comunicação com o banco de dados SQL Server foi estabelecida por intermédio do *driver* **mssql-jdbc-12.10.0.jre11.jar**.
2. **Criação de objetos de negócio**: As classes de entidade foram estruturadas com herança e polimorfismo.
3. **Operações CRUD**:
    - Inclusão de pessoas físicas e jurídicas (tabelas Pessoa, Pessoa_Fisica, Pessoa_Juridica)
    - Alteração e exclusão dos dados persistidos
    - Listagem geral das pessoas cadastradas
4. **Controle de sequência**: O *SequenceManager* foi utilizado para gerenciar as sequências da *chave primária* no banco de dados SQL Server.

📄 Códigos

Os códigos foram desenvolvidos com a IDE NetBeans e se encontram em repositório no GitHub onde podem ser acessados pelo link: https://github.com/CarlosCatao/Mundo_3_Nivel_3_Missao_Pratica/tree/main/Procedimento-1/CadastroBD.

🧪 Testes

Foram explorados testes de inclusão, alteração, exclusão e listagem. Como aprendizado importante há que se destacar o aspecto relativo às permissões necessárias para se efetuar as transações junto ao banco de dados.

- O uso do conn.setAutoCommit(false) com commit() e rollback() permitiu controle transacional, evitando corrupção de dados em falhas intermediárias.
- A listagem no console (via exibir()) confirmou a integridade das operações exploradas nos testes.

📝 Resultados

Os resultados da execução dos códigos se encontram ilustrados no arquivo *Resultados.pdf* que se encontra em repositório no GitHub onde podem ser acessados pelo link: https://github.com/CarlosCatao/Mundo_3_Nivel_3_Missao_Pratica/blob/main/Procedimento-1/RESULTADOS.pdf.

⚠️ Desafios Enfrentados

- **Permissões no banco**: Inicialmente, utilizando o usuário *loja* ocorreram erros de permissão para transações como INSERT, UPDATE e DELETE; que impediram a execução correta dos comandos. A solução encontrada consistiu na aplicação de GRANTs apropriados, utilizando a credencial do administrador do banco de dados.
- **Erros de índice em PreparedStatement**: Em alguns métodos incluir, o número de parâmetros definidos não coincidia com os placeholders da query, gerando erros do tipo "*índice fora do intervalo*".
- **Gerenciamento de sequência**: O *SequenceManager* foi de extrema importância para o tratamento das chaves primárias quando da inclusão de dados no banco.

📌 Conclusão

O projeto **CadastroBD** demonstrou como aplicações Java podem interagir de forma robusta com bancos de dados relacionais, aplicando boas práticas de POO e JDBC. A modularização em pacotes, o uso de DAO, o controle de transações e a organização do código tornam o sistema facilmente escalável e compreensível, o que facilita em muito a depuração do código e sua manutenabilidade.

O exercício permitiu consolidar conhecimentos técnicos importantes, como:

- Conexão com SQL Server via JDBC
- Estruturação de aplicações com camadas separadas
- Tratamento de exceções e recursos (fechamento de ResultSet, Statement, Connection)
- Operações CRUD usando SQL nativo

❓ Questionamentos

Qual a importância dos componentes de middleware, como o JDBC?

O **JDBC** é responsável por intermediar a comunicação (*middleware*) entre aplicações Java e bancos de dados relacionais, garantindo eficiência, segurança e padronização no acesso a dados.

Qual a diferença no uso de *Statement* ou *PreparedStatement* para a manipulação de dados?

Statement executa comandos SQL simples, enquanto *PreparedStatement* permite comandos **parametrizados**, oferecendo maior segurança contra injeções de SQL, melhor desempenho em consultas repetidas.

Como o padrão DAO melhora a manutenibilidade do software?

O padrão DAO isola a lógica de acesso a dados do restante da aplicação, facilitando futuras manutenções, a possível troca do **SGBD** (Sistema Gerenciador do Banco de Dados) e a organização do código.

Como a herança é refletida no banco de dados, quando lidamos com um modelo estritamente relacional?

No modelo relacional, a herança é geralmente implementada usando tabelas separadas para cada classe, utilizando chaves estrangeiras para representar a relação entre as tabelas.