****

**Banco de dados usando JBDC e JPA**

**- Repositórios**

github.com/danielkv7/digital-innovation-one/tree/master/jdbc-basico

github.com/danielkv7/digital-innovation-one/tree/master/Aula\_JPA\_basico

**JDBC (Java Database Connectivity)**

É uma API de diversas classes e interfaces escritas em Java que estão no pacote java.sql e javax.sql, que têm com finalidade fazer com que programas escritos em Java possam ter acesso a banco de dados.

**- Classe DriverManager**

Responsável pela comunicação com os drivers disponíveis. Para isso ela cria uma Connection com o banco de dados através de uma URL, esta tem a função de especificar o driver, localização e o nome do BD.

**- Interface Connection**

Representa a conexão com o banco de dados, permitindo criar Statements que constroem consultas SQL.

**- Conectando um banco de dados**

1 – Baixar o driver específico do banco de dados que será utilizado.

2 – Criar a URL de conexão com os parâmetros: driver, endereço do BD e nome do BD.

3 – Criar uma conexão com o DriverManager utilizando o método getConnection, passando os parâmetros: string de conexão, usuário e senha.

**- Interfaces para montar comandos SQL**

Statement - Executar comandos SQL estáticos.

PreparedStatement – Executar comandos SQL parametrizáveis.

CallableStatement – Executar stored procedures

**- Métodos para executar comandos SQL**

execute – para executar qualquer tipo de SQL

executeQuery – usados para executar comandos Selects

executeUpdate – usados para comandos que fazer alteração no bando de dados (INSERT, UPDATE, DELETE, CREATE, ALTER)

**- Obter dados de uma consulta**

ResultSet - é o objeto que obtém os dados de uma determinada consulta de um BD.

Ele tem métodos getters para buscar os dados: getInt, getString, getFloat.

Para percorrer os registros do ResultSet usa o método next(), normalmente fazendo-se uso do while.

**JPA (Java Persistence API)**

Foi motivada a sua criação pela percepção do tempo desprendido para se fazer consultas SQL nos processo com JDBC, além de que se ter de conciliar a programação orientada a objeto ao modo relacional utilizados pelos SGBDs.

**- ORM (Mapeamento Objeto Relacional)**

É um modelo que propõe uma relação entre as entidades da programação a objeto com as entidades de um banco de dados relacional.

Tabela < --- > Classes

Coluna < --- > Atributo

Registro < --- > Objeto

Para padronizar a implementação do modelo ORM foi criada a interface JPA. Ela descreve como deve ser o comportamento dos frameworks Java ORM que desejam implementar a sua especificação.

As classes, interfaces e anotações JPA estão presentes no pacote javax.persistence.

A JPA precisa de uma implementação para que se possa comunicar com o banco de dados.

**- Passos para utilizar o JPA**

**1- Ter a biblioteca do JPA e o driver JDBC do banco que será utilizado.**

**2 – Criar o arquivo persistence.xml para configurar os parâmetros da conexão:**

- URL da string de conexão.

- Driver do banco de dados utilizado.

- Endereço do banco de dados.

- Nome, usuário e senha do banco de dados.

- Driver e classes que serão mapeadas.

**3 – Fazer o mapeamento das classes, ou seja, dizer quais as classes que representam quais tabelas nos bancos de dados.**

Esses mapeamentos são feitos através das anotações.

@Entity - indica que os objetos da classe, onde ela está declarada, serão persistidos no banco de dados. Deve ser colocada antes da linha de declaração da classe.

@Table – Anotação opcional. esta anotação só é necessária caso o nome da entidade seja diferente do nome da tabela no banco de dados.

@Id – é obrigatório determinar ao menos um id para a entidade. Deve mapear a coluna da tabela que tem id único.

@Column – faz mapeamento de uma coluna da tabela.

@ManyToMany - @ManyToOne - @OneToMany - @OneToOne - são anotações de relacionamento entre tabelas (que possuem chaves estrangeiras)

Em aplicações utilizando JPA os relacionamentos podem ser unidirecionais ou bidirecionais.

Propriedades das anotações de relacionamento:

fetch – esta propriedade possui dois valores:

- Fetchtype.EAGER - a entidade mapeada com este atributo será carregada na aplicação, quando a entidade que está mapeando for consultada, mesmo que não esteja sendo usada.

- Fetchtype.LAZY - a entidade mapeada com este atributo só será carregada na aplicação quando for explicitamente consultada pela entidade que a está mapeando.

**4 – Manipulando as transações.**

entityManager – é o método que deve ser importado na classe que irá manipular as transações e o ciclo de vida das transações.

Dentro da classe segue-se a sequência:

1 - Passos iniciais para criar um gerenciador de entidades com o banco de dados especificado no arquivo "persistence.xml"

EntityManagerFactory entityManagerFactory = Persistence.createEntityManagerFactory("part1-DIO");

EntityManager entityManager = entityManagerFactory.createEntityManager();

2.1 - Criar instancias para serem adicionadas no banco de dados

Estado estadoParaAdicionar = new Estado("Rio de Janeiro", "RJ");

Aluno alunoParaAdicionar = new Aluno("Daniel", 29, estadoParaAdicionar);

2.2 - Iniciar uma trasacao para adiconar as instancias no banco de dados

entityManager.getTransaction().begin();

entityManager.persist(estadoParaAdicionar);

entityManager.persist(alunoParaAdicionar);

entityManager.getTransaction().commit(); (se quiséssemos que a transação não fosse implementada usaríamos ao invés de commit() o rollback()

3 - Encerrar o gerenciador de entidades e encerrar a fábrica de gerenciadores de entidade.

entityManager.close();

entityManagerFactory.close();

**5 – Implementar o JPA (Hibernate e EclipseLink)**

Para utilizar a implementação com as especificações JPA deve-se seguir os seguintes passos:

1 – Realizar o download da API da implementação utilizada.

2 – Modificar o arquivo persistence.xml configurando a tag <provider> indicando a classe da implementação que será utilizada.

3 – Verificar os parâmetros específico de cada implementação, tais como dialeto do DB, log dos SQLs criados e automatização dos comandos DDL.

**JPQL (Java Persistente Query Language)**

JPQL é uma linguagem de consulta independente orientada a objetos definida pelo JPA.

É usada para realizar consultas no banco de dados.

Ele não interage diretamente com o banco de dados, mas faz uso das entidades JPA para interagir com o BD.

Com ele é possível utilizar as propriedades de orientação a objeto nas consultas realizadas no banco de dados, através das entidades mapeadas, tal como herança.

O JPQL se torna mais eficiente que o EntityManager pois com este só é possível trabalhar com uma entidade de cada vez, enquanto o que com o JPQL consegue trabalhar com um conjunto de dados através de consultas.