Programação Orientada a Objetos

Persistência de Dados

Rayanne Giló da Silva
rayanne.silva@alunos.ufersa.edu.br

Persistência de Dados

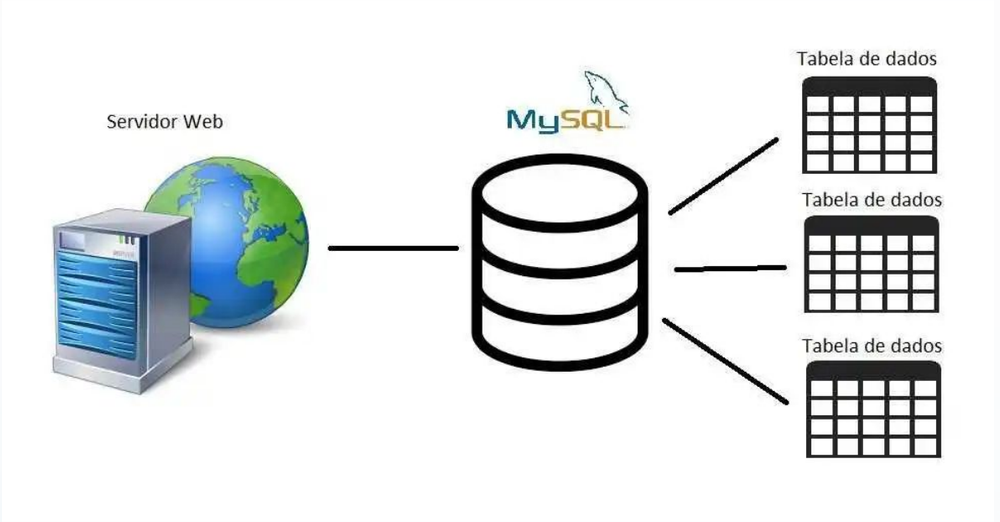
- Muitos sistemas precisam manter as informações com as quais trabalham
- Dados na Memória são voláteis
- Deve-se armazená-los de forma persistente para:
  - consultas futuras, geração de relatórios ou possíveis alterações

Persistência de Dados

- Necessidades:
  - Armazenamento
  - Organização
  - Busca
  - Compartilhamento
  - Integridade
  - Concorrência
- O objetivo principal é minimizar o número de requisições a base de dados

Banco de Dados

- Coleção de dados relacionados
- Sistemas de armazenamento de dados
- Armazenamento de forma independente:
  - Independente de linguagem
  - Independente de aplicação



Banco de Dados

Banco de Dados

- A maioria dos bancos de dados comerciais são os chamados relacionais
- Trata-se de uma forma de trabalhar e pensar diferente do paradigma orientado a objetos.

Alunos

Matricula	Nome	Dt_nasc	NumNat
M01	Maria	15/06/1986	N02
M02	Cláudia	06/07/1984	N01
M03	Simone	23/09/1981	N01
M04	Fernando	01/12/1978	N03

Disciplina

NumDisc	Nome	C.H
D01	Banco de Dados I	72
D02	Programação II	72
D03	Auditoria	80
D04	Calculo I	96

Notas

Matricula	NumDisc	Nota
M01	D01	10
M01	D02	12
M01	D03	15
M02	D01	12
M03	D02	13
M03	D04	14
M04	D02	11
M04	D03	13

Naturalidade

NumNat	Descrição
N01	Belo Horizonte
N02	João Monlevade
N03	Ipatinga
N04	Governador Valadares

Banco de Dados

- Entre os diversos SGBDs, utilizaremos o **MySQL**
- Tem versão gratuita, é robusto e fácil instalar (MySQL Community Server)
- Depois de instalado, para acessá-lo via terminal, fazemos da seguinte forma:
  - mysql -u root -p
  - Em seguida, basta digitar a senha do root

Problema

- Programamos usando o Paradigma OO
- O BD é relacional

Mapeamento Objeto-relacional

- Atividade não trivial!
- Representação dos dados em BDR é incompatível com a representação em uma hierarquia de objetos
- Tipos de dados OO não são os mesmos existentes em um banco de dados
  - Necessidade de mecanismos para mapeamento de objetos em memória para meios persistentes relacionais
  - Deve-se fornecer transparência ao usuário
- Processo não automático

Mapeamento Objeto-relacional

- Possui 3 componentes:
  - Modelo orientado a objeto: objetos da aplicação
  - Persistência física: entidade relacional em que os dados são armazenados
  - Persistência lógica: tradução do modelo OO para a persistência física e vice-versa

Mapeamento Objeto-relacional

Mapeamento Objeto-relacional

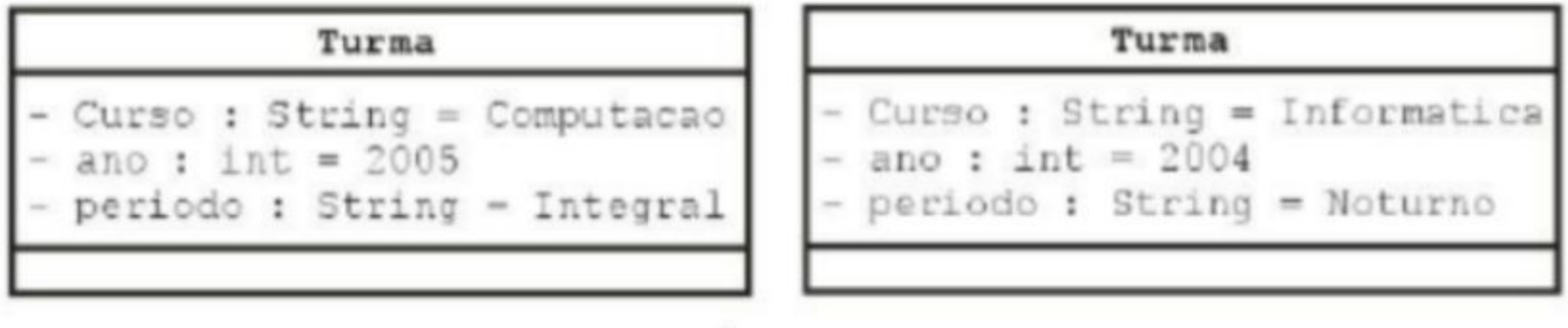
- Dois tipos principais de mapeamento:
  - Objetos compostos por tipos primitivos
  - Objetos que possuem coleções de objetos

Mapeamento Objeto-relacional

- Mapeamento de objetos compostos por tipos primitivos
  - Mapeamento simples
  - Objetos contém apenas tipos primitivos de dados
  - Possuem correspondência em SQL

Mapeamento Objeto-relacional

- Exemplo

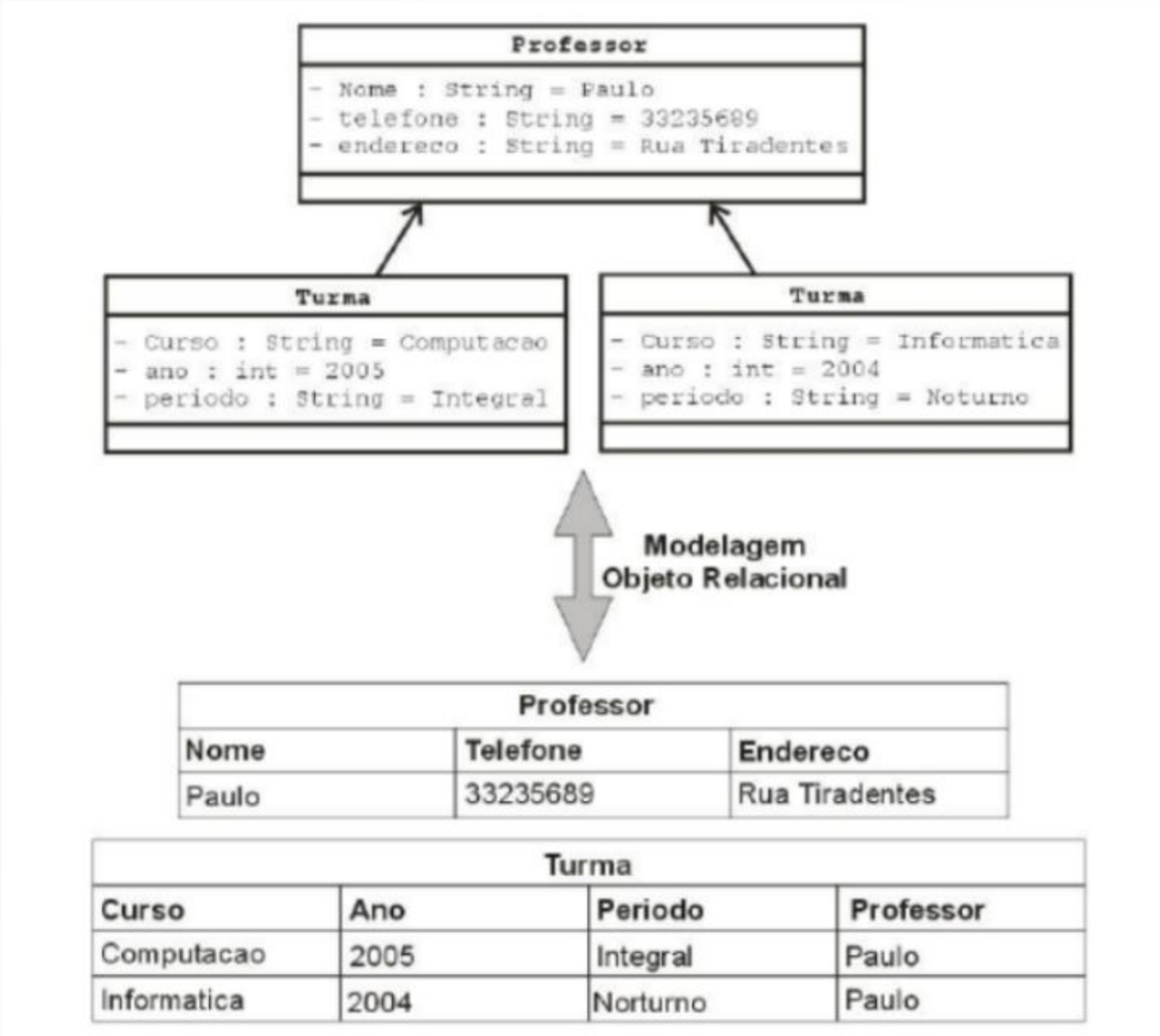


Modelagem
Objeto Relacional

Turma		
Curso	**Ano**	**Periodo**
Computacao	2005	Integral
Informatica	2004	Norturno

Mapeamento Objeto-relacional

- Mapeamento de objetos complexos



Professor		
Nome	**Telefone**	**Endereco**
Paulo	33235689	Rua Tiradentes

Turma			
Curso	**Ano**	**Periodo**	**Professor**
Computacao	2005	Integral	Paulo
Informatica	2004	Norturno	Paulo

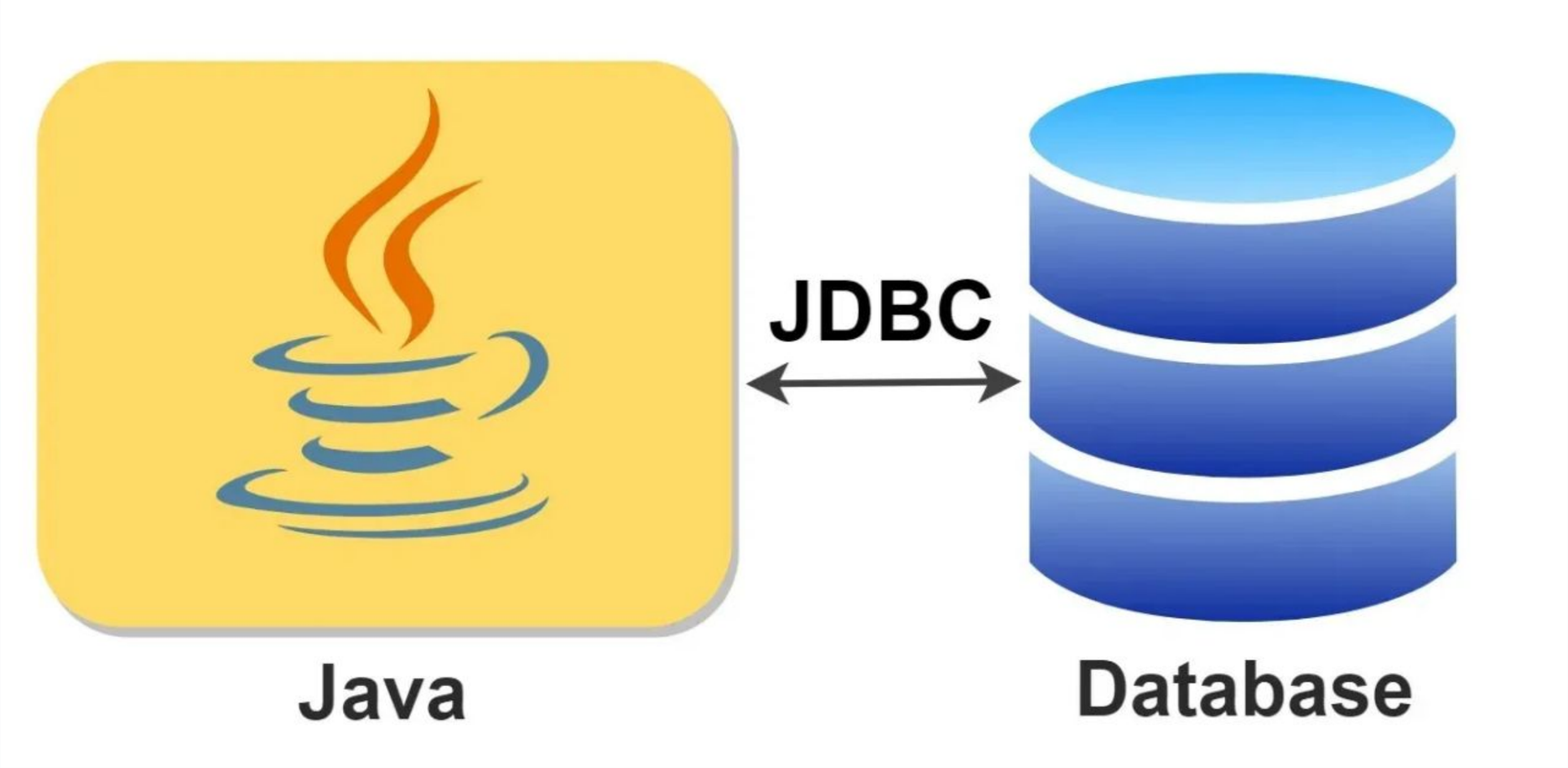
Estratégias de Persistência

- JDBC
  - API nativa;
  - Vantagens: controle total do SQL e da performance;
  - Desvantagens: muito boilerplate (abrir/fechar recursos), mapeamento manual, transações 100% manuais
- JPA
  - Especificação elaborada pelo Java Community Process (JCP) para persistência em Java
  - Define interfaces e anotações
  - Abstrai detalhes de implementação para permitir portar entre frameworks
- Hibernate
  - ORM (Object-Relational Mapping) que implementa a especificação JPA
  - Gera SQL automaticamente a partir de suas entidades Java

Estratégias de Persistência

JDBC

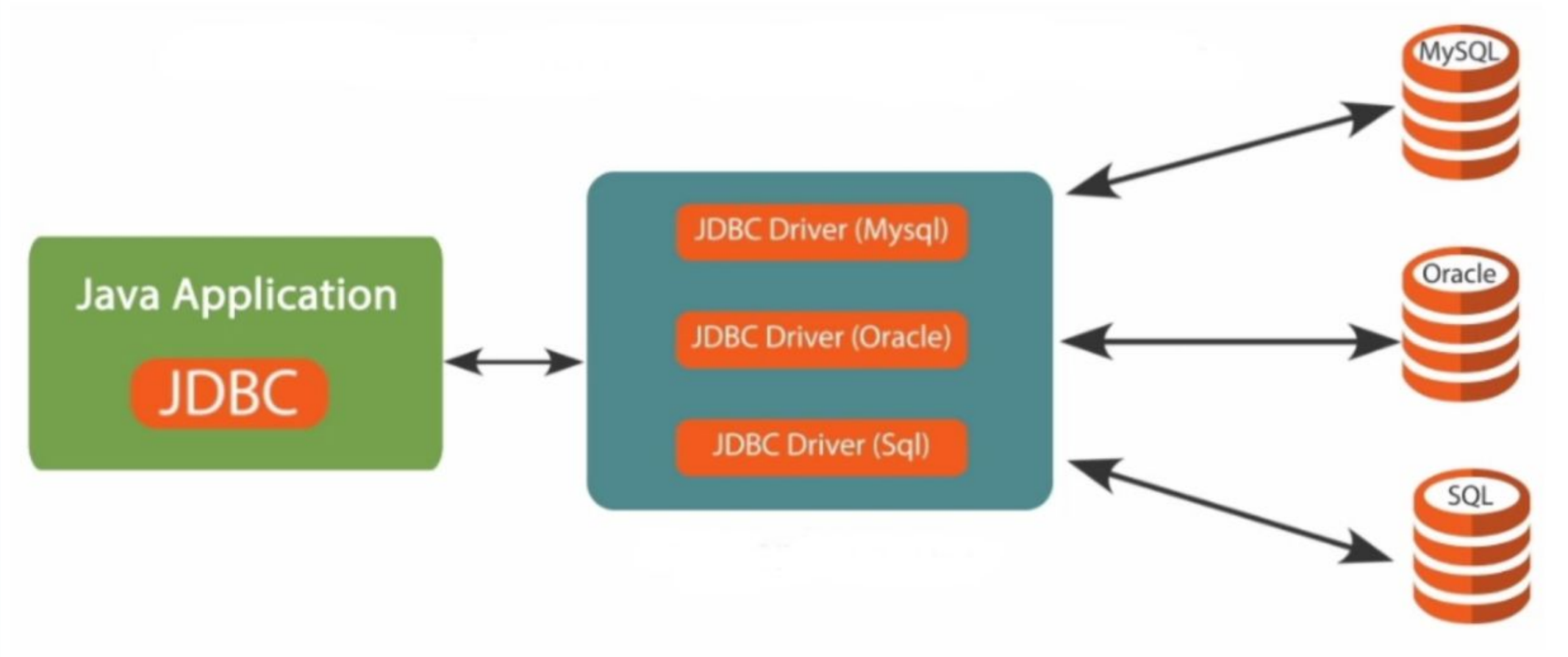
- A API de persistência em banco de dados relacionais do Java



JDBC

- Para evitar que cada BD tenha a sua API e conjunto de classes e métodos, temos um único conjunto de interfaces
- muito bem definidas: A JDBC
- A JDBC fica no pacote **java.sql**

JDBC


Java Application
JDBC
JDBC Driver (Mysql)
JDBC Driver (Oracle)
JDBC Driver (Sql)
MySQL
Oracle
SQL

Por que não usar JDBC puro?

- Boilerplate excessivo : abrir e fechar Connection, Statement e ResultSet em toda operação
- Transações 100% manuais: conn.setAutoCommit(false), commit() / rollback() explicitamente em código
- Mapeamento manual: extrair valores de ResultSet e popular objetos Java “na mão”
- Baixa produtividade e manutenção difícil: código repetido e propenso a erros.
- Sem recursos extras: não há cache automático, fetch strategies nem geração de SQL dinâmico

JPA

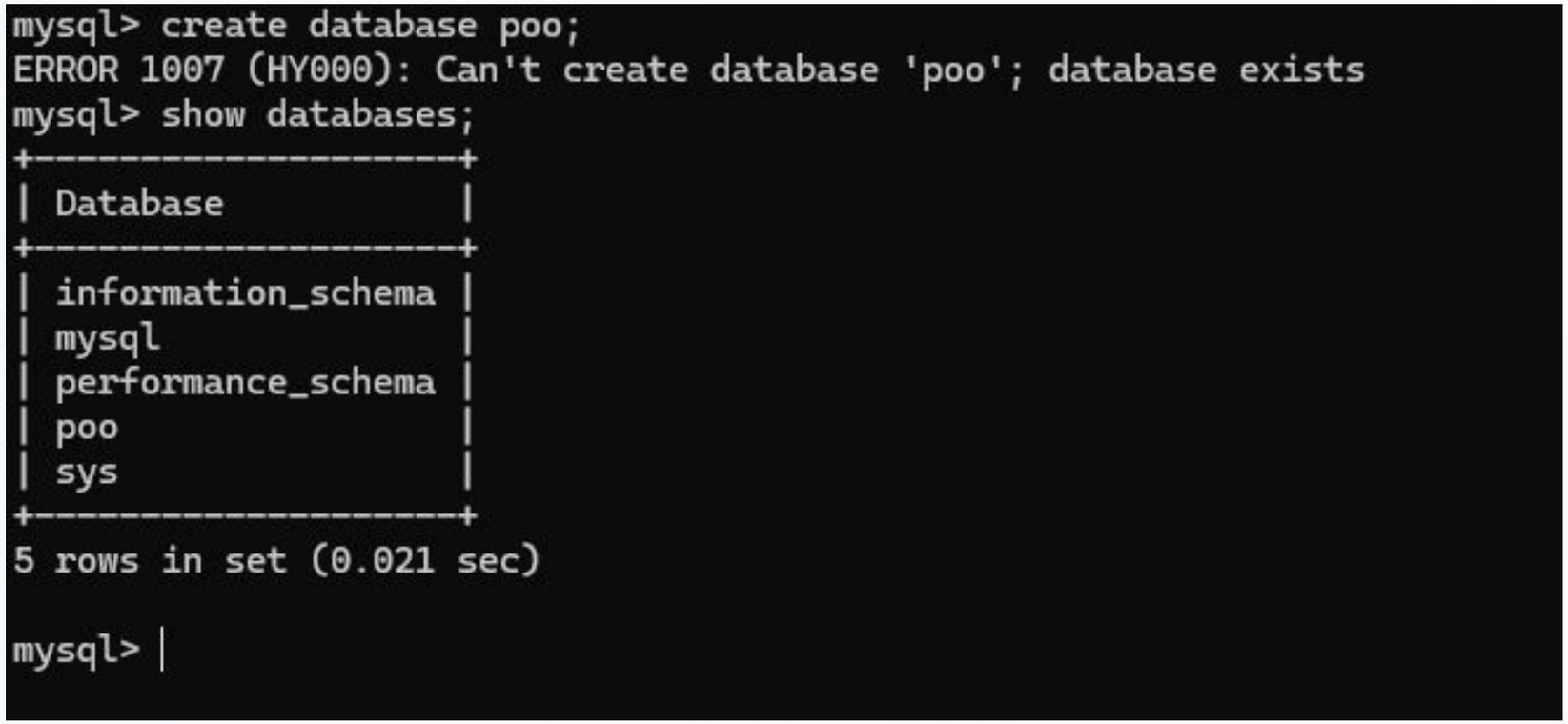
- Java Persistence API (JPA) é uma especificação padrão da plataforma Java EE (pacote javax.persistence) para mapeamento objeto-relacional e persistência de Dados.
- Para trabalhar com JPA é preciso incluir no projeto uma implementação da API (por exemplo, Hibernate).

Hibernate

- Framework ORM (Object-Relational Mapping) para Java
- Implementação JPA mais usada, mas também pode ser usado via API nativa (Session)
- Mapeamento automático de entidades Java ⇄ tabelas SQL
- Gerenciamento de ciclo de vida de entidades: transient, persistent, detached
- Controle de transações simplificado pela API (beginTransaction / commit / rollback)
- Open-source, parte do ecossistema ***Hibernate.org***

Java e MySQL

- No terminal MySQL digite: create database poo; e em seguida digite show databases; para verificar se o bd foi criado.



Configurando o JPA e o Hibernate

- Para configurar o JPA e o Hibernate é preciso adicionar algumas dependências no projeto.
- No arquivo ***pom.xml*** adicione as linhas de codigo:

```
<dependencies>
    <!-- JPA Core -->
    <dependency>
        <groupId>jakarta.persistence</groupId>
        <artifactId>jakarta.persistence-api</artifactId>
        <version>3.2.0</version>
    </dependency>
    <!-- Hibernate Core -->
    <dependency>
        <groupId>org.hibernate.orm</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>7.0.0.Final</version>
    </dependency>

    <dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <version>9.3.0</version>
    </dependency>
</dependencies>
```

Criando o arquivo de configuração

- O JPA precisa de um arquivo chamado: ***persistence.xml***
- Nesse arquivo ficarão as configurações do banco de dados.
- Ele deve ser criado dentro da pasta resources, e por sua vez, dentro da pasta META-INF.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1"
             xmlns="http://xmlns.jcp.org/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">

    <persistence-unit name="aulabd" transaction-type="RESOURCE_LOCAL">
        <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>

        <class>com.rayannegsilva.model.Pessoa</class>

        <properties>
            <property name="javax.persistence.jdbc.driver"
value="com.mysql.cj.jdbc.Driver" />
            <property name="javax.persistence.jdbc.url"
value="jdbc:mysql://localhost:3306/poo" />
            <property name="javax.persistence.jdbc.user" value="USUARIO" />
            <property name="javax.persistence.jdbc.password" value="SENHA
USUARIO" />

            <property name="hibernate.dialect"
value="org.hibernate.dialect.MySQLDialect" />

            <property name="hibernate.hbm2ddl.auto" value="update" />

            <property name="hibernate.show_sql" value="true" />
            <property name="hibernate.format_sql" value="true" />
        </properties>
    </persistence-unit>
</persistence>
```

Criando um helper para o Hibernate

- É uma classe de ajuda, comumente chamada de "helper" ou "utility class", projetada para inicializar e gerenciar a conexão do seu aplicativo com o banco de dados através do Hibernate

```
public class HibernateUtil {
    private static final SessionFactory SF = buildSessionFactory();

    private static SessionFactory buildSessionFactory() {
        try {
            return new Configuration()
                    .configure()
                    .buildSessionFactory();
        } catch (Throwable ex) {
            System.err.println("Erro ao criar SessionFactory: " + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }
    public static SessionFactory getSessionFactory() {
        return SF;
    }
    public static void shutdown() {
        SF.close();
    }
}
```

JPA na entidade

- Exemplo de uma entidade com JPA.

```
@Entity
@Table(name = "pessoas")
public class Pessoa {
@Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private Long id;

        @Column(nullable = false, length = 100)
        private String nome;

public Pessoa() {}

(getters, setters...)
}
```

Utilizando na main

- É uma classe de ajuda, comumente chamada de "helper" ou "utility class", projetada para inicializar e gerenciar a conexão do seu aplicativo com o banco de dados através do Hibernate

```
EntityManagerFactory emf = JPAUtil.getEntityManagerFactory();

(...restante do codigo...)

JPAUtil.shutdown();
```

Adicionando uma Pessoa no BD

- Exemplo para adicionar um dado na tabela.

```
try (EntityManager em = emf.createEntityManager()) {
        EntityTransaction tx = em.getTransaction();
        tx.begin();

        Pessoa p = new Pessoa();
        p.setNome("Rayanne");
        em.persist(p); // O método é o mesmo, mas agora no EntityManager

        tx.commit(); // Finaliza e grava a transação
        System.out.println("ID gerado = " + p.getId());
    }
```

Listando todas as pessoas

- Exemplo para Listar todos os dados de uma tabela.

```
try (EntityManager em = emf.createEntityManager()) {

    TypedQuery<Pessoa> query = em.createQuery("SELECT p FROM Pessoa p", Pessoa.class);
    List<Pessoa> todas = query.getResultList(); // O método padrão é getResultList()

    System.out.println("Todas as pessoas:");
    todas.forEach(p -> System.out.println(" - " + p.getNome()));
}
```

Busca por ID

- Exemplo para buscar por ID.

```
try (EntityManager em = emf.createEntityManager()) {
     Pessoa p = em.find(Pessoa.class, 1L);
       if (p != null) {
           System.out.println("Buscada por ID: " + p.getNome());
       } else {
           System.out.println("Pessoa com ID 1 não encontrada.");
         }
    }
```

Alterando o dado de uma pessoa

- Exemplo alterar um dado.

```
try (EntityManager em = emf.createEntityManager()) {
        EntityTransaction tx = em.getTransaction();
        tx.begin();

         Pessoa p = em.find(Pessoa.class, 1L);
         if (p != null) {
            p.setNome("Rayanne G Silva");
             System.out.println("Atualizada: " + p.getNome());
          }

         tx.commit();
    }
```

Deletando uma pessoa

- Exemplo exclusão de um dado.

```
try (EntityManager em = emf.createEntityManager()) {
      EntityTransaction tx = em.getTransaction();
      tx.begin();

      Pessoa p = em.find(Pessoa.class, 4L);
      if (p != null) {
         em.remove(p);
           System.out.println("Removida: " + p.getNome());
        }

        tx.commit();
 }
```

Atividade

- Agora tente implementar utilizando como base o template disponível no link:
  - https://github.com/rayannegsilva/template_poo

Dúvida?