

# Persistência com JDBC e JPA

## Aula 3

Marcos Alberto Lopes da Silva  
([malopes21@gmail.com](mailto:malopes21@gmail.com))



# Sumário

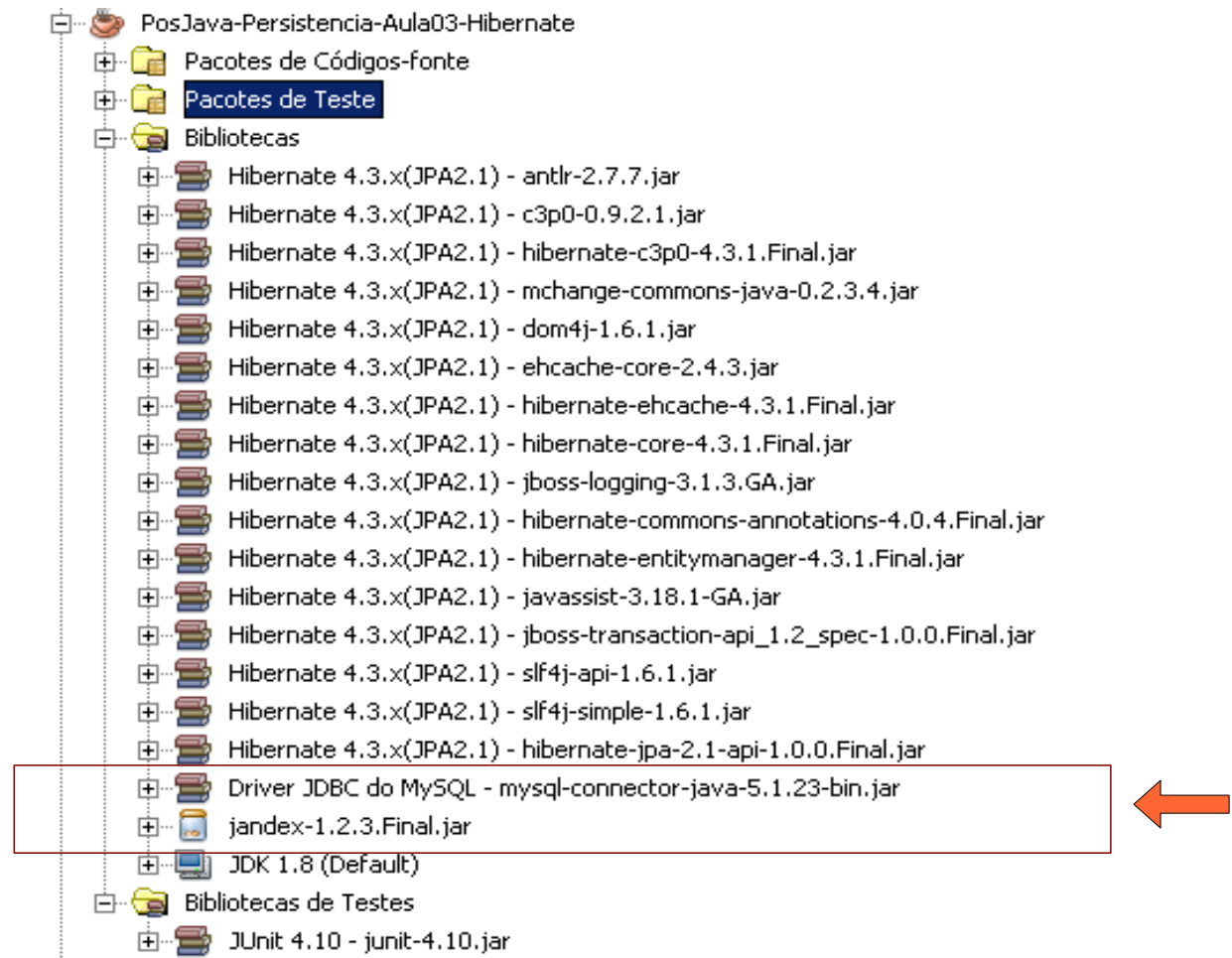
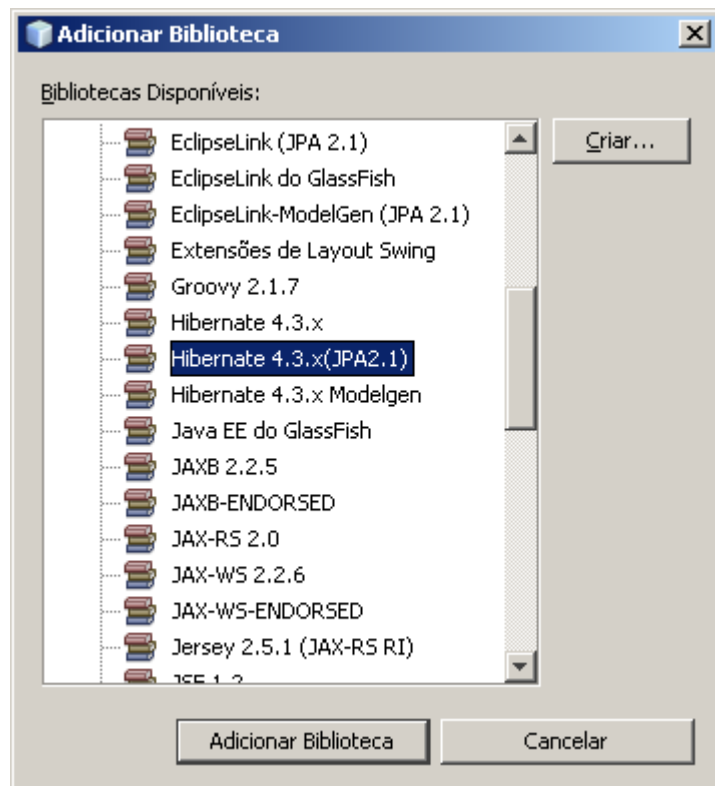
- Projeto JPA/Netbeans;
- Projeto JPA/Eclipse;
- JPA – manipulando entidades;
- JPA – mapeamento de tipos de dados;
- JPA – transações;
- JPA – relacionamentos;
- JPA – eager/lazy;

# Siglas e acrônimos

- POO = Programação Orientada a Objetos;
- SGBDR = Sistema de gerenciamento de bancos de dados relacionais;
- DAO = Data Access Object;
- JDBC = Java Database Connectivity;
- JPA = Java Persistence API;
- ORM = object-relational mapping ou mapeamento objeto-relacional;
- JSR = Java Specification Request;
-

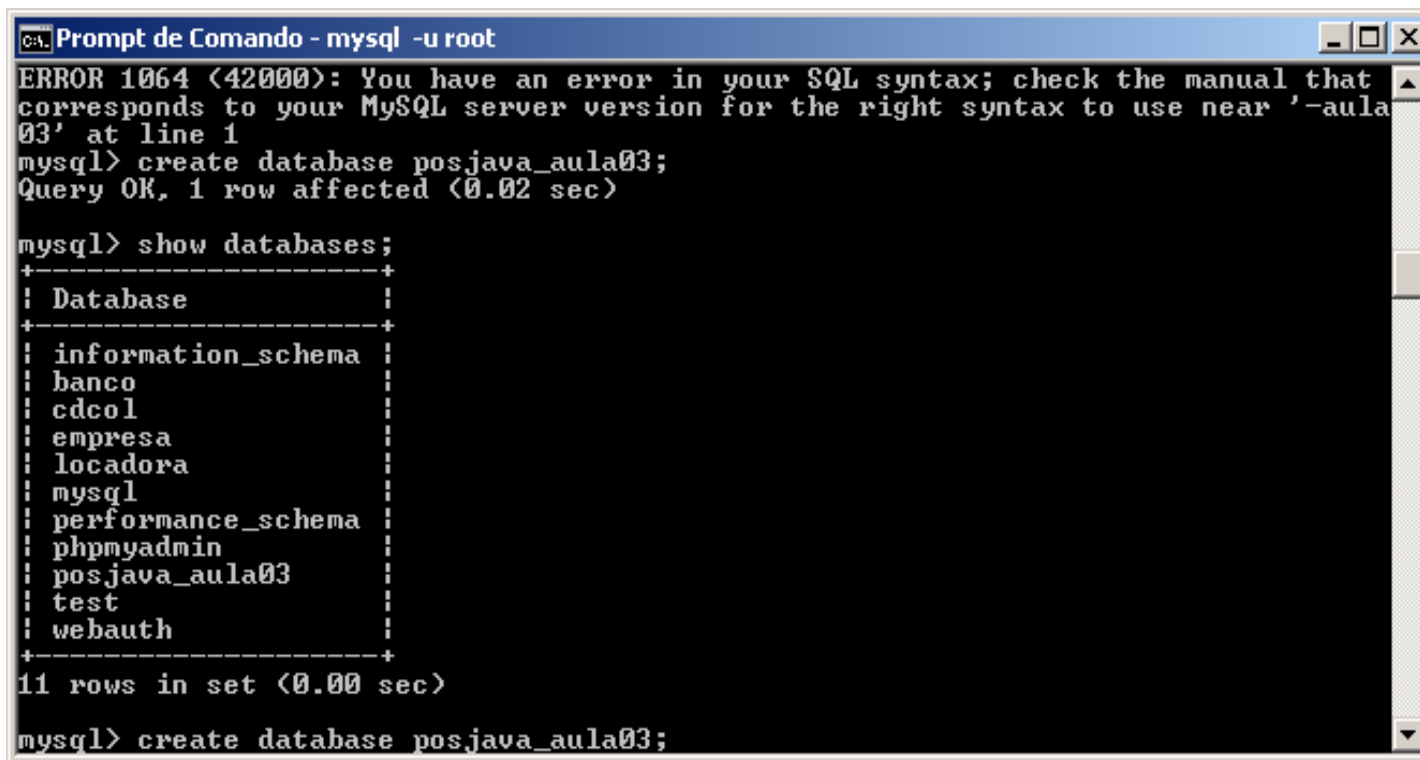
# Projeto Java com JPA - NetBeans

Projeto Java 1.8 "padrão" com bibliotecas do Hibernate/JPA



# Projeto Java com JPA - NetBeans

Criar um BD no MySql por exemplo, com o nome posjava\_aula03:



```
Prompt de Comando - mysql -u root
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that
corresponds to your MySQL server version for the right syntax to use near '-aula
03' at line 1
mysql> create database posjava_aula03;
Query OK, 1 row affected (0.02 sec)

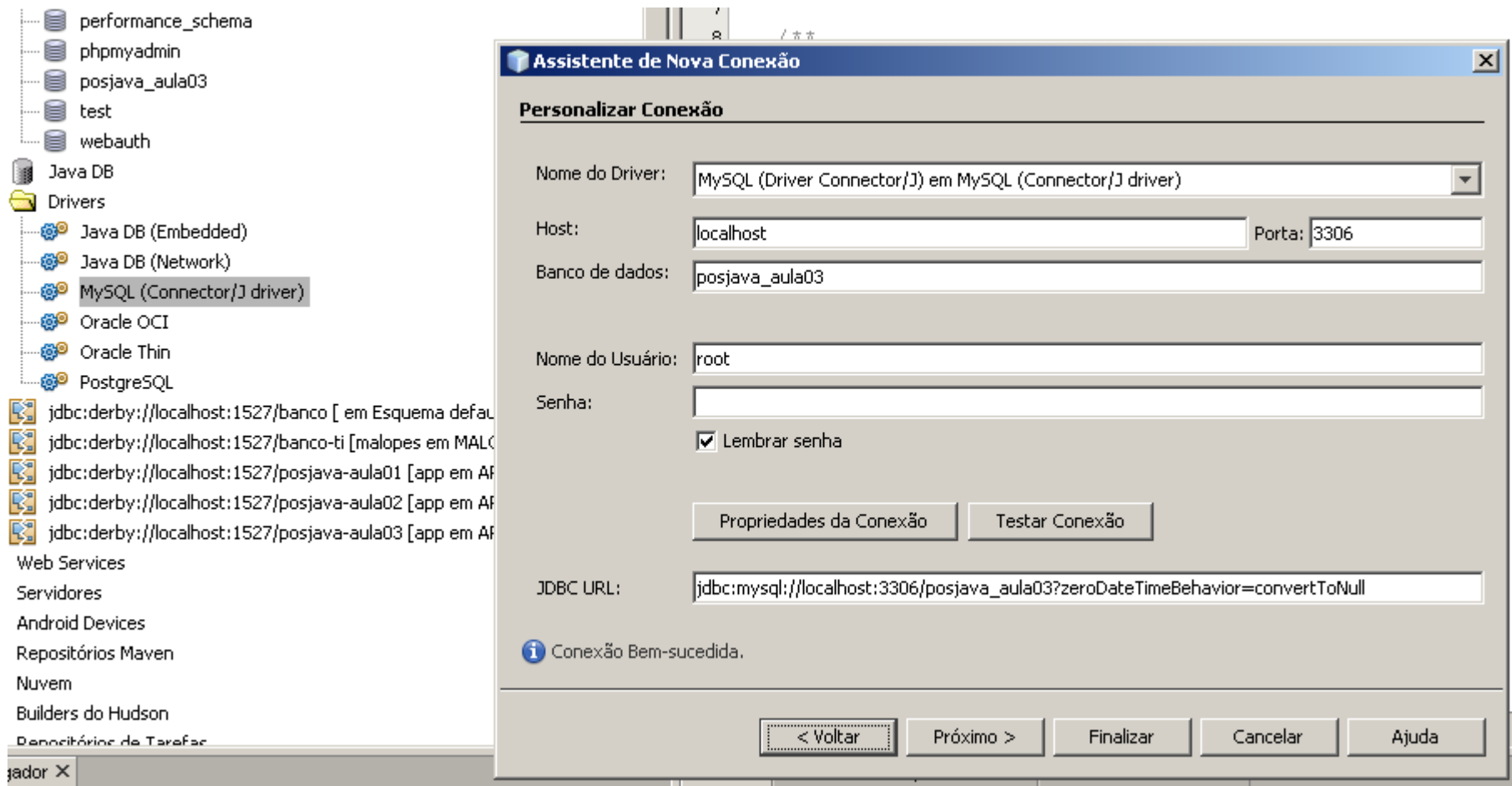
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| banco |
| cdcol |
| empresa |
| locadora |
| mysql |
| performance_schema |
| phpmyadmin |
| posjava_aula03 |
| test |
| webauth |
+-----+
11 rows in set (0.00 sec)

mysql> create database posjava_aula03;
```

mysql>use posjava\_aula03;

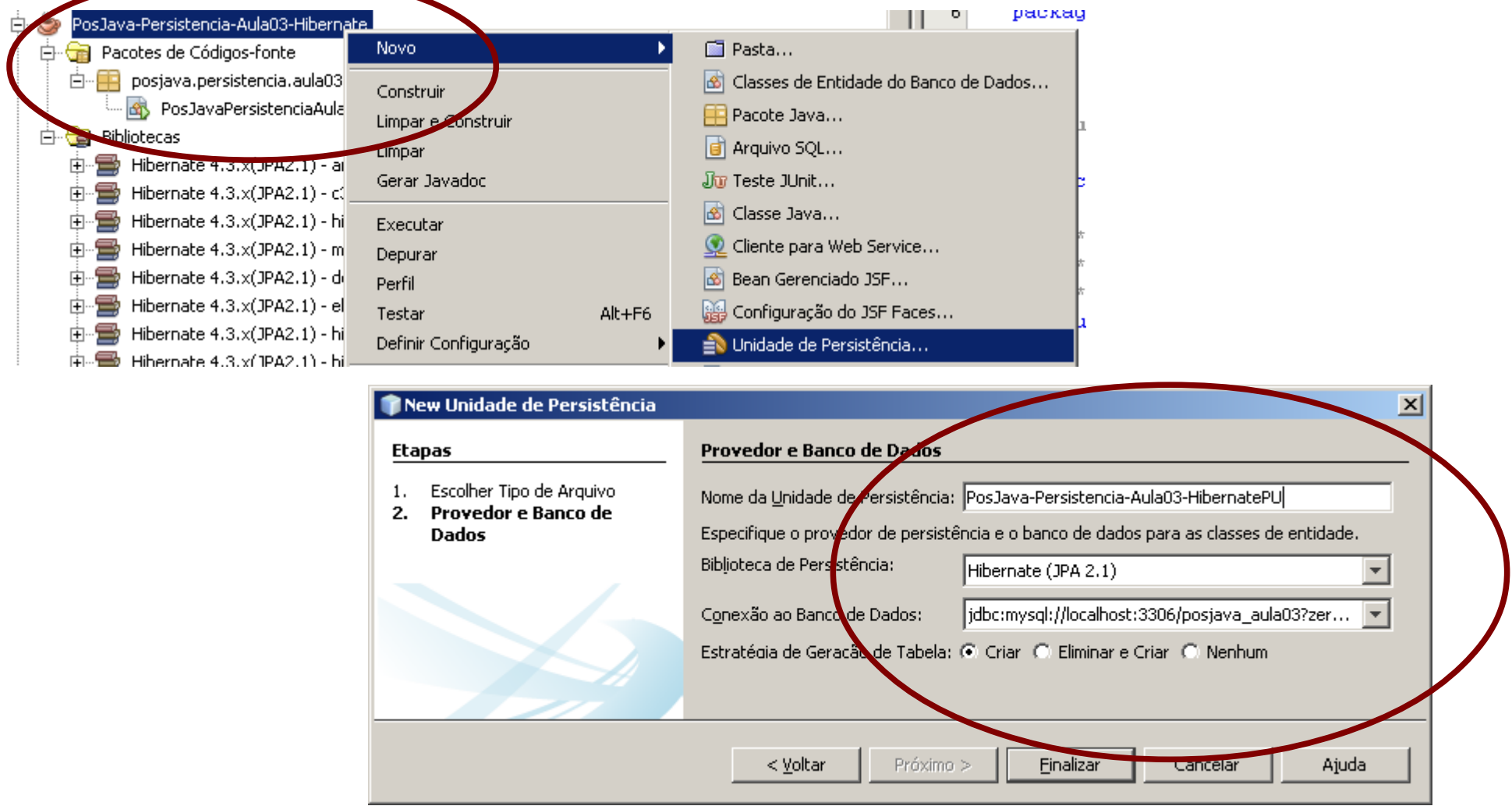
# Projeto Java com JPA - NetBeans

Na aba serviços, pasta Drivers, Conectar no BD (Mysql) posjava\_aula03:



# Projeto Java com JPA - NetBeans

Criar o arquivo de configuração da unidade de persistência: persistence.xml



# Projeto Java com JPA - NetBeans

Com algumas configurações adicionais: persistence.xml

```
<?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="PosJava-Persistencia-Aula03-HibernatePU" transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
    <properties>
      <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/posjava_aula03"/>
      <property name="javax.persistence.jdbc.user" value="root"/>
      <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
      <property name="javax.persistence.jdbc.password" value=""/>
      <property name="hibernate.cache.provider_class" value="org.hibernate.cache.NoCacheProvider"/>
      <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5InnoDBDialect"/>
      <property name="hibernate.show_sql" value="true"/>
      <property name="javax.persistence.schema-generation.database.action" value="drop-and-create"/>
      <property name="hibernate.format_sql" value="true"/>
    </properties>
  </persistence-unit>
</persistence>
```



# Projeto Java com JPA

Criando a entidade domain.Pessoa:

```
@Entity
public class Pessoa implements Serializable {

    /**
     *
     */
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue
    private Long id;
    private String nome;
    private String email;

    //setters e getters
    //contrutores
    //equals e hashCode
    //toString
    //etc
}
```

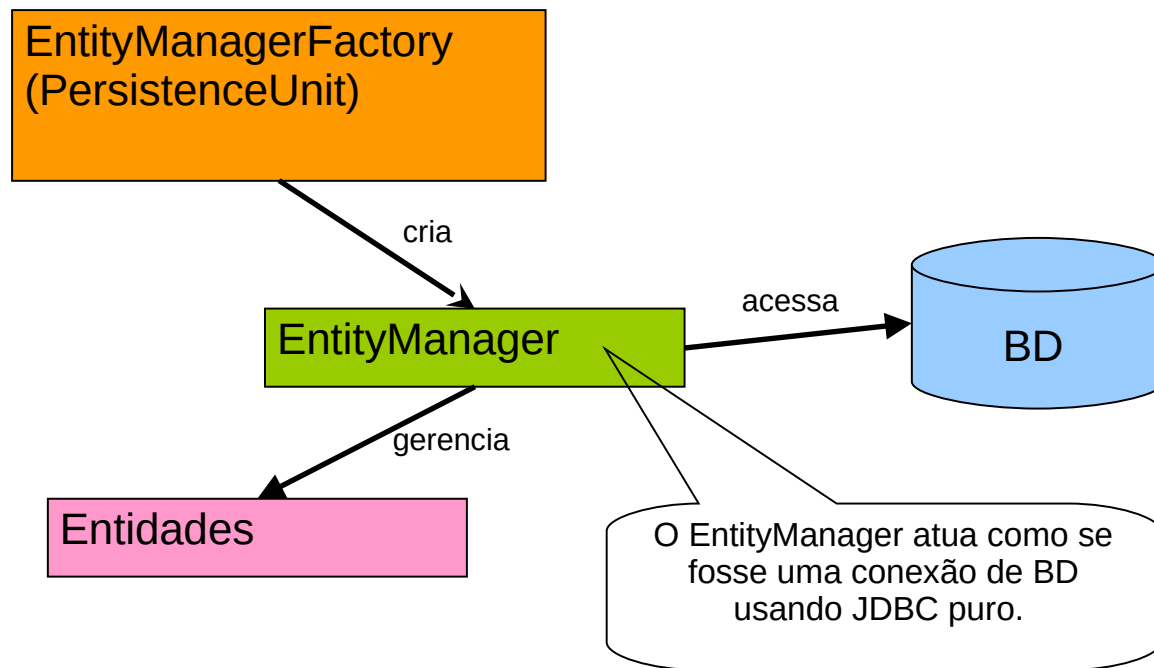
# Projeto Java com JPA

Criando a classe util.JpaUtil (não tão util para testes junit)

```
public class JpaUtil {  
  
    private static EntityManagerFactory emf = null;  
  
    public static EntityManager getManager() {  
        if(emf == null) {  
            emf = Persistence.  
                createEntityManagerFactory("PosJava-Persistencia-Aula03-HibernatePU");  
        }  
        return emf.createEntityManager();  
    }  
  
    public static void closeManager(EntityManager manager) {  
        try {  
            manager.close();  
        } catch (Exception ex) {  
            System.err.println("Erro: "+ex.getMessage());  
        }  
    }  
}
```

# Projeto Java com JPA

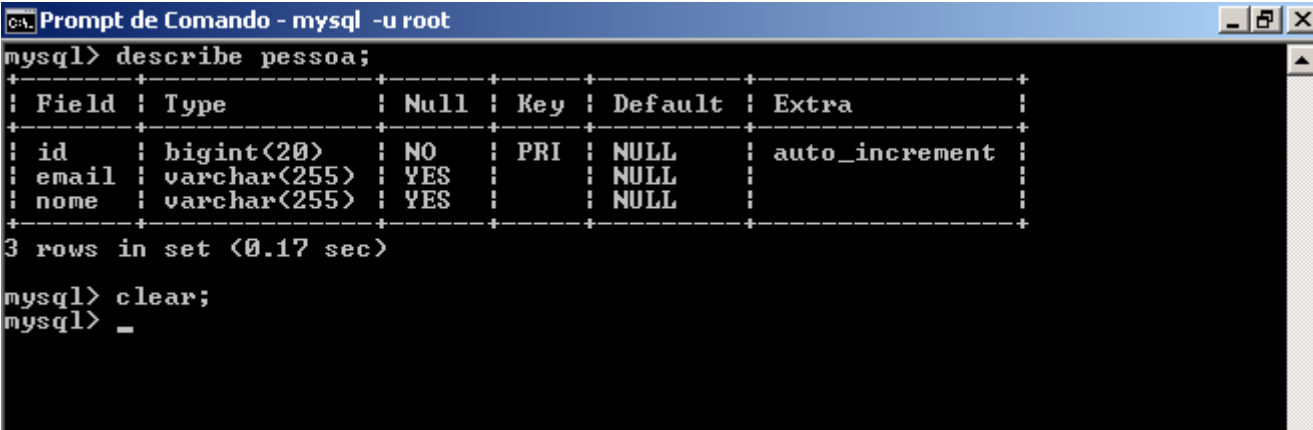
## Arquitetura da JPA:



# Projeto Java com JPA

Criando a tabela PESSOA via caso de teste junit:

```
@Test
public void CreateTablePessoa() {
    EntityManager manager = JpaUtil.getManager();
    assertTrue(manager != null);
    JpaUtil.closeManager(manager);
}
```



```

C:\> Prompt de Comando - mysql -u root
mysql> describe pessoa;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| id     | bigint(20)    | NO   | PRI | NULL    | auto_increment |
| email  | varchar(255)  | YES  |     | NULL    |                |
| nome   | varchar(255)  | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.17 sec)

mysql> clear;
mysql> _

```

# JPA – Manipulando entidades

## Persistindo um objeto da classe PESSOA:

```
private static EntityManagerFactory emf;  
private EntityManager manager;
```

```
@BeforeClass
```

```
public static void setUpClass() {  
    emf = Persistence  
        .createEntityManagerFactory("PosJava-Persistencia-Aula03-HibernatePU"); }  
@AfterClass
```

```
@AfterClass
```

```
public static void tearDownClass() {  
    emf.close();}
```

```
@Before
```

```
public void setUp() {  
    manager = emf.createEntityManager();}
```

```
@After
```

```
public void tearDown() {  
    manager.close();  
}
```

```
@Test
```

```
public void persist() {  
    manager.getTransaction().begin();  
    Pessoa pessoa = new Pessoa(null, "Marcos Lopes", "malopes21@gmail.com");  
    pessoa.setNascimento(Calendar.getInstance());  
    pessoa.getNascimento().set(1970, 5, 15);  
    manager.persist(pessoa);  
    manager.getTransaction().commit();  
}
```



```
CA. Prompt de Comando - mysql -u root  
mysql> select * from Pessoa;  
+----+-----+-----+-----+  
| id | email                | nascimento | nome      |  
+----+-----+-----+-----+  
| 1  | malopes21@gmail.com  | 1970-06-15 | Marcos Lopes |  
+----+-----+-----+-----+  
1 row in set (0.00 sec)  
mysql>
```

# JPA – Manipulando entidades - persist

## Persistindo um objeto da classe PESSOA:

```
private static EntityManagerFactory emf;  
private EntityManager manager;
```

```
@BeforeClass
```

```
public static void setUpClass() {  
    emf = Persistence  
        .createEntityManagerFactory("PosJava-Persistencia-Aula03-HibernatePU"); }  
@AfterClass
```

```
@Before
```

```
public static void tearDownClass() {  
    emf.close();}
```

```
@Before
```

```
public void setUp() {  
    manager = emf.createEntityManager();}
```

```
@After
```

```
public void tearDown() {  
    manager.close();  
}
```

```
@Test
```

```
public void persist() {  
    manager.getTransaction().begin();  
    Pessoa pessoa = new Pessoa(null, "Marcos Lopes", "malopes21@gmail.com");  
    pessoa.setNascimento(Calendar.getInstance());  
    pessoa.getNascimento().set(1970, 5, 15);  
    manager.persist(pessoa);  
    manager.getTransaction().commit();  
}
```



CA. Prompt de Comando - mysql -u root

```
mysql> select * from Pessoa;
```

id	email	nascimento	nome
1	malopes21@gmail.com	1970-06-15	Marcos Lopes

```
1 row in set (0.00 sec)  
  
mysql>
```

# JPA – Manipulando entidades - find

Buscando um objeto da classe PESSOA:

```
private static EntityManagerFactory emf;  
private EntityManager manager;
```

...

```
@Test  
public void find() {  
    Pessoa pessoa = manager.find(Pessoa.class, 1L);  
    System.out.println("Achei: " + pessoa);  
    assertTrue(pessoa != null && pessoa.getNome().equals("Marcos Lopes"));  
}
```

Estratégia de Geração de Tabela:

☐ Criar ☐ Eliminar e Criar ☒ Nenhum

Não é necessário demarcar uma transação para essa operação!

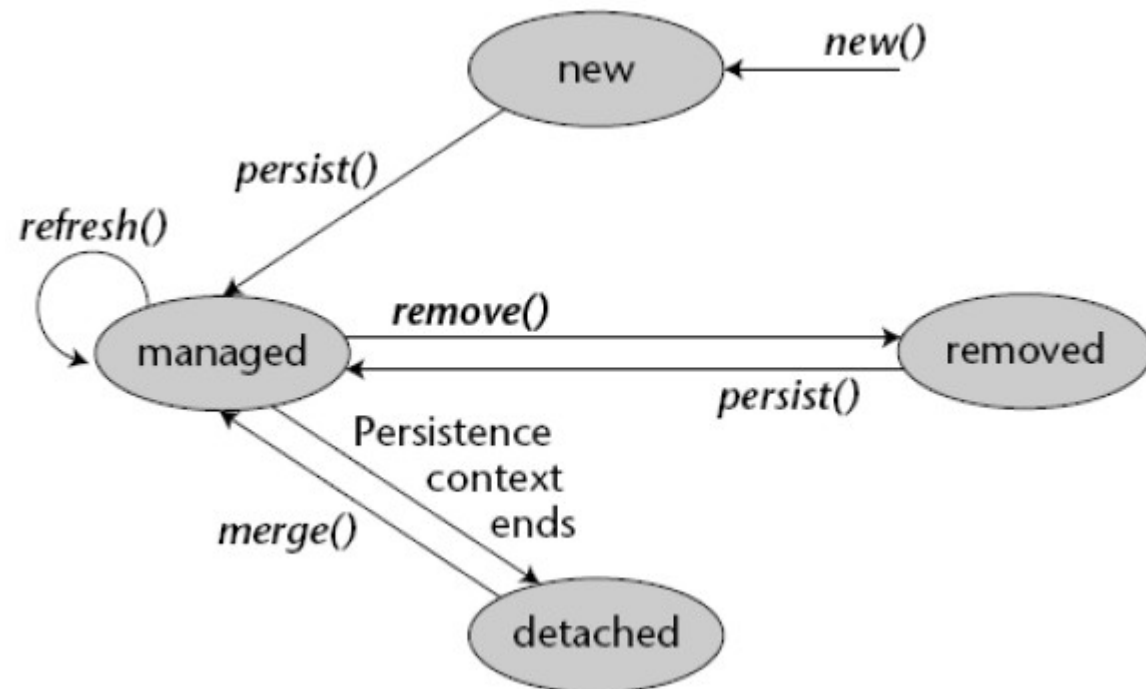
# JPA – Manipulando entidades - remove

Apagando um objeto (registro) da classe PESSOA:

```
private static EntityManagerFactory emf;  
private EntityManager manager;  
  
...  
  
@Test  
public void remove() {  
    manager.getTransaction().begin();  
    Pessoa pessoa = manager.find(Pessoa.class, 1L);  
    manager.remove(pessoa);  
    manager.getTransaction().commit();  
}
```

```
mysql> select * from Pessoa;  
Empty set (0.00 sec)
```

Só podemos "apagar"  
objetos gerenciados!





# JPA – Manipulando entidades - "update"

Atualizando um objeto (registro) da classe PESSOA:

```
private static EntityManagerFactory emf;  
private EntityManager manager;  
  
...  
  
@Test  
public void update() {  
    manager.getTransaction().begin();  
    Pessoa pessoa = manager.find(Pessoa.class, 2L);  
    pessoa.setNome("Marcos Alberto Lopes da Silva");  
    manager.getTransaction().commit();  
}
```

```
mysql> select * from Pessoa;  
+----+-----+-----+-----+  
| id | email                | nascimento | nome                |  
+----+-----+-----+-----+  
| 2  | malopes21@gmail.com | 1970-06-15 | Marcos Alberto Lopes da Silva |  
+----+-----+-----+-----+  
1 row in set (0.00 sec)  
  
mysql> _
```

# JPA – Manipulando entidades - "select"

Listando objetos da classe PESSOA:

```
private static EntityManagerFactory emf;  
private EntityManager manager;
```

...

@Test

```
public void select() {  
    List<Pessoa> pessoas = manager  
        .createQuery("select p from Pessoa p", Pessoa.class)  
        .getResultList();  
    pessoas.forEach((p) -> System.out.println(p));  
}
```

```
mysql> select * from Pessoa;
```

id	email	nascimento	nome
2	malopes21@gmail.com	1970-06-15	Marcos Alberto Lopes da Silva
3	fulano@gmail.com	1980-01-05	Fulano de Tal

```
2 rows in set (0.00 sec)
```

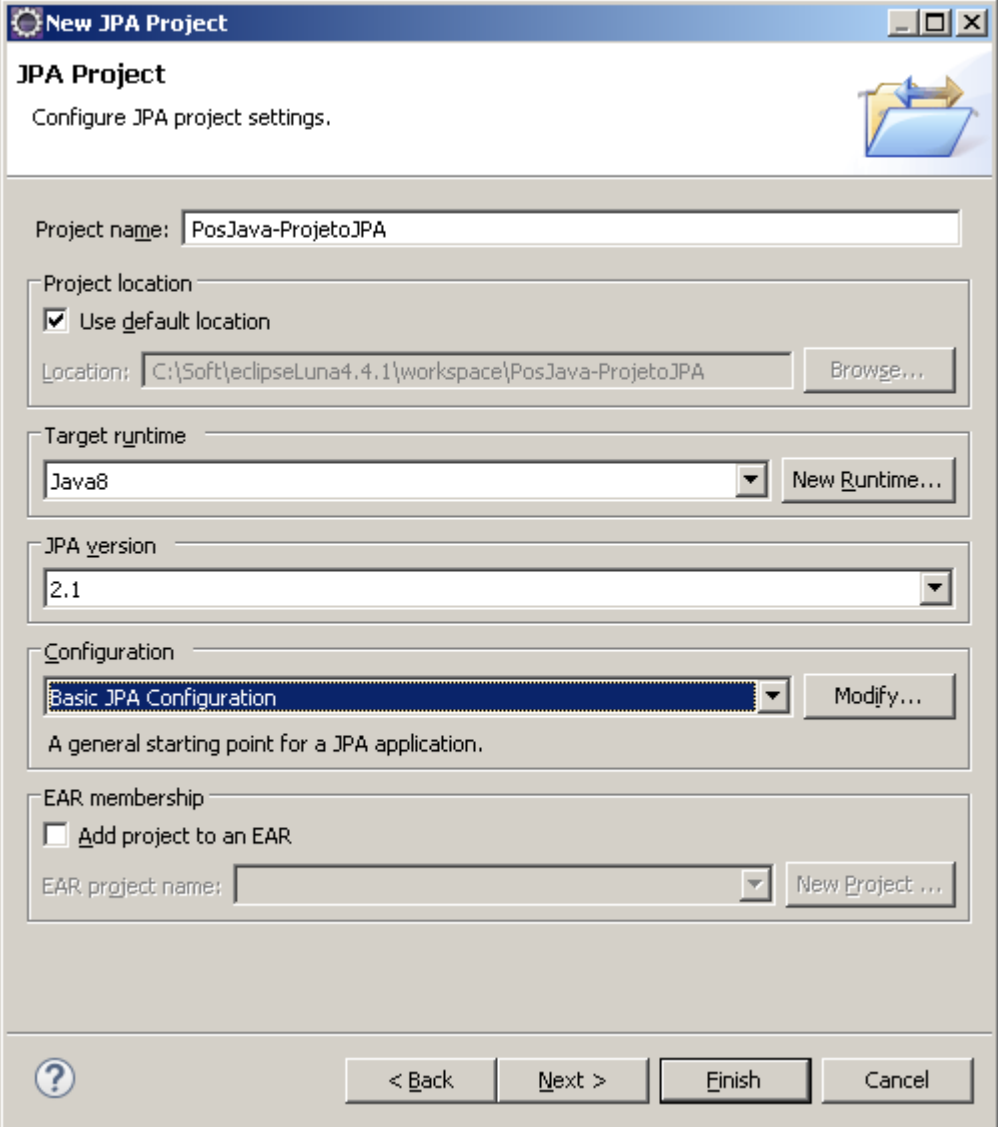
```
mysql>
```

Hibernate:

```
select  
    pessoa0_.id as id1_0_,  
    pessoa0_.email as email2_0_,  
    pessoa0_.nascimento as nascimen3_0_,  
    pessoa0_.nome as nome4_0_  
from  
    Pessoa pessoa0_  
Pessoa{id=2, nome=Marcos Alberto Lopes da Silva, email=malopes21@gmail.com}  
Pessoa{id=3, nome=Fulano de Tal, email=fulano@gmail.com}
```

# Projeto Java com JPA - Eclipse

New project >> JPA Project:



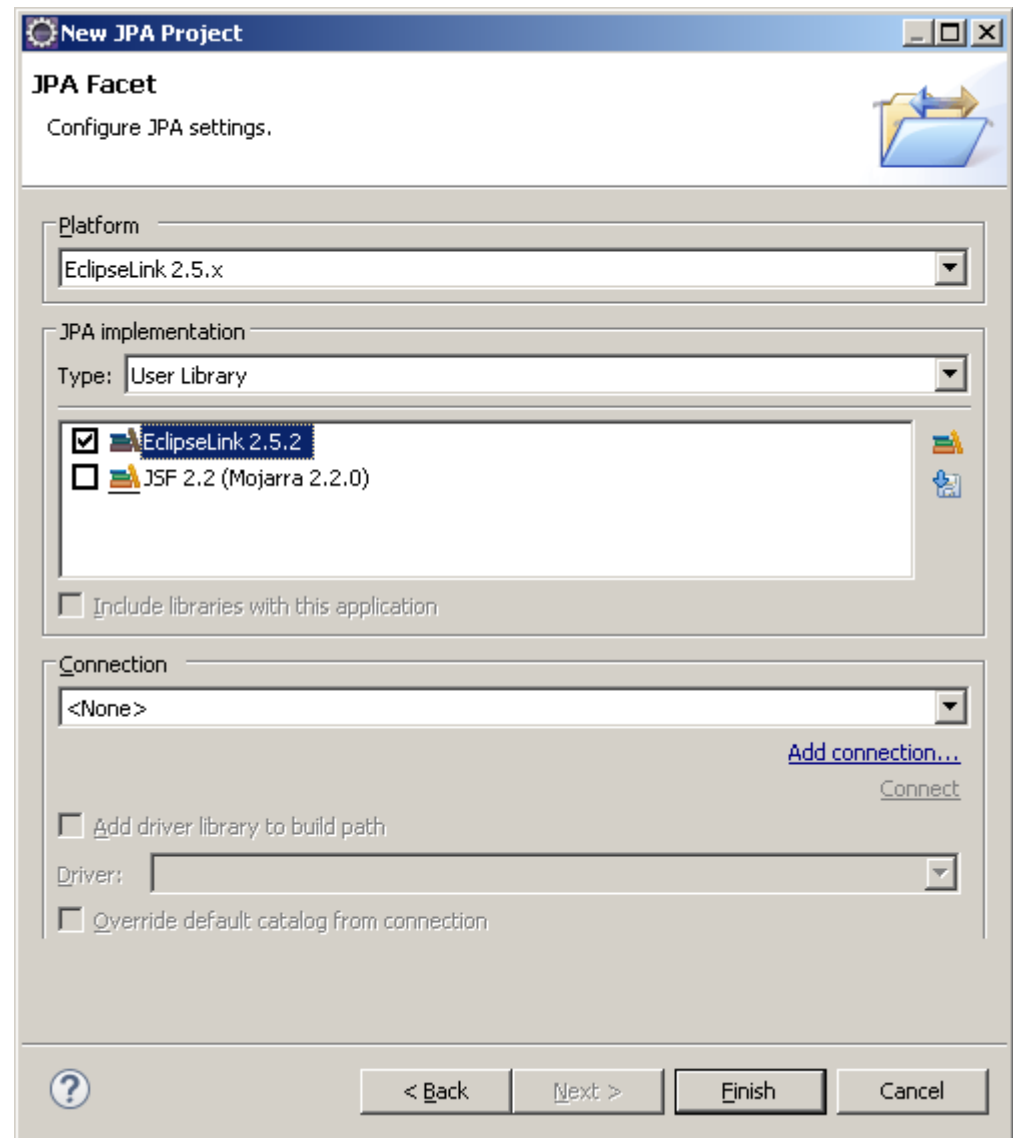
The screenshot shows the 'New JPA Project' dialog box in Eclipse. The dialog is titled 'New JPA Project' and has a subtitle 'JPA Project' with the instruction 'Configure JPA project settings.' Below this, there are several sections for configuration:

- Project name:** A text field containing 'PosJava-ProjetoJPA'.
- Project location:** A section with a checked checkbox 'Use default location' and a text field 'Location:' containing 'C:\Soft\ eclipseLuna4.4.1\workspace\PosJava-ProjetoJPA'. A 'Browse...' button is to the right.
- Target runtime:** A dropdown menu showing 'Java8' and a 'New Runtime...' button.
- JPA version:** A dropdown menu showing '2.1'.
- Configuration:** A dropdown menu showing 'Basic JPA Configuration' and a 'Modify...' button. Below this is the text 'A general starting point for a JPA application.'
- EAR membership:** A section with an unchecked checkbox 'Add project to an EAR' and a text field 'EAR project name:' with a dropdown arrow. A 'New Project ...' button is to the right.

At the bottom of the dialog, there is a help icon (question mark) and four buttons: '< Back', 'Next >', 'Finish', and 'Cancel'.

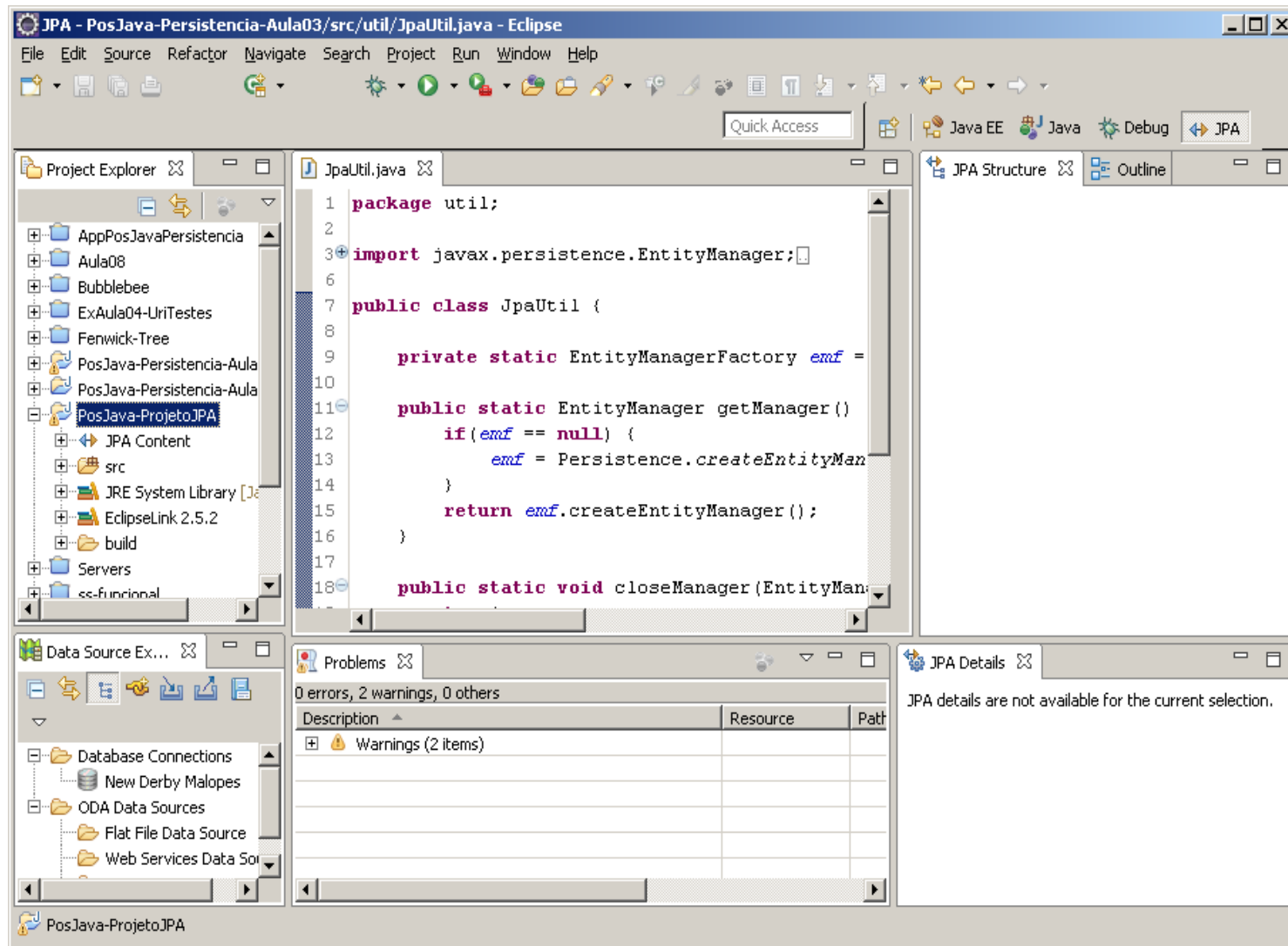
# Projeto Java com JPA - Eclipse

JPA Project: configurando o projeto para EclipseLink



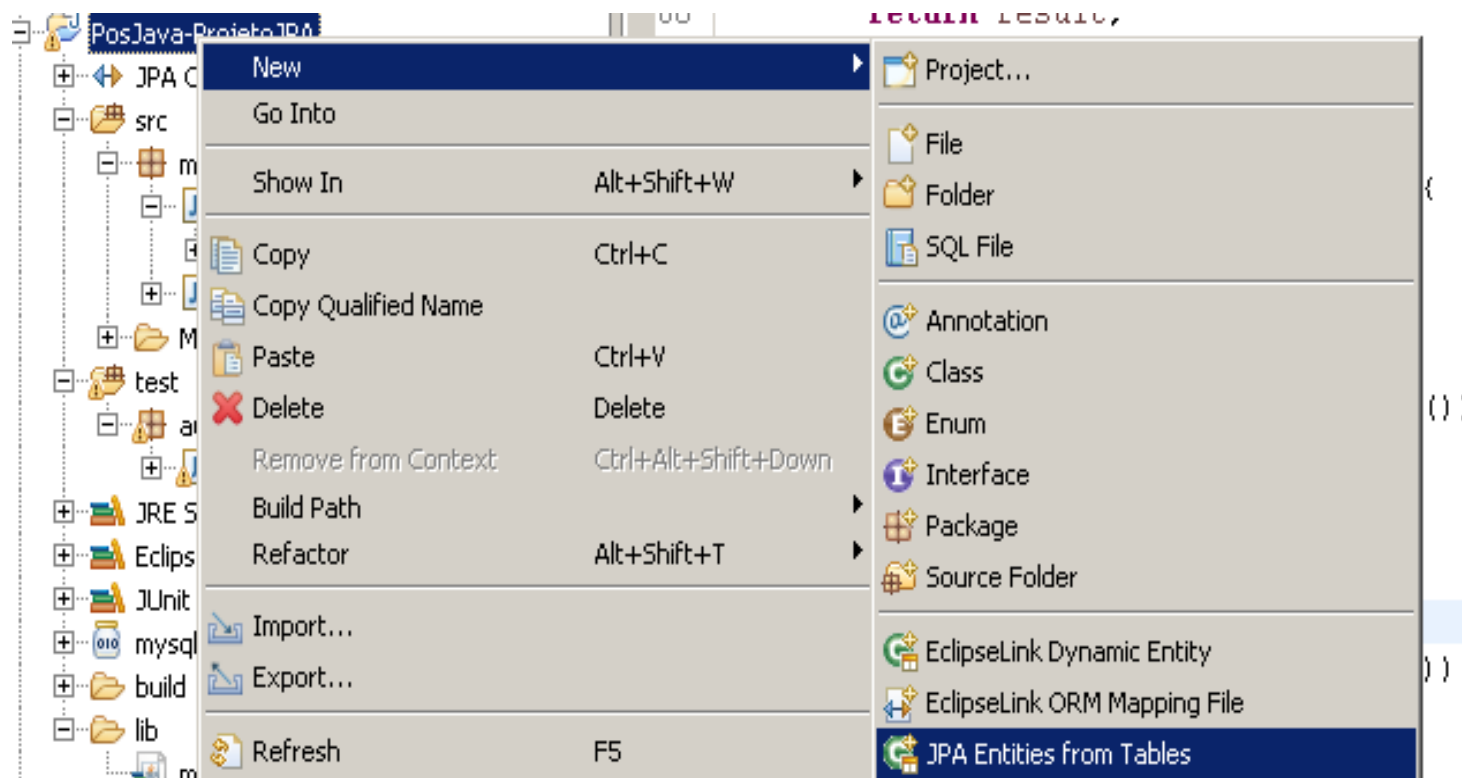
# Projeto Java com JPA - Eclipse

## JPA Project: perspectiva JPA



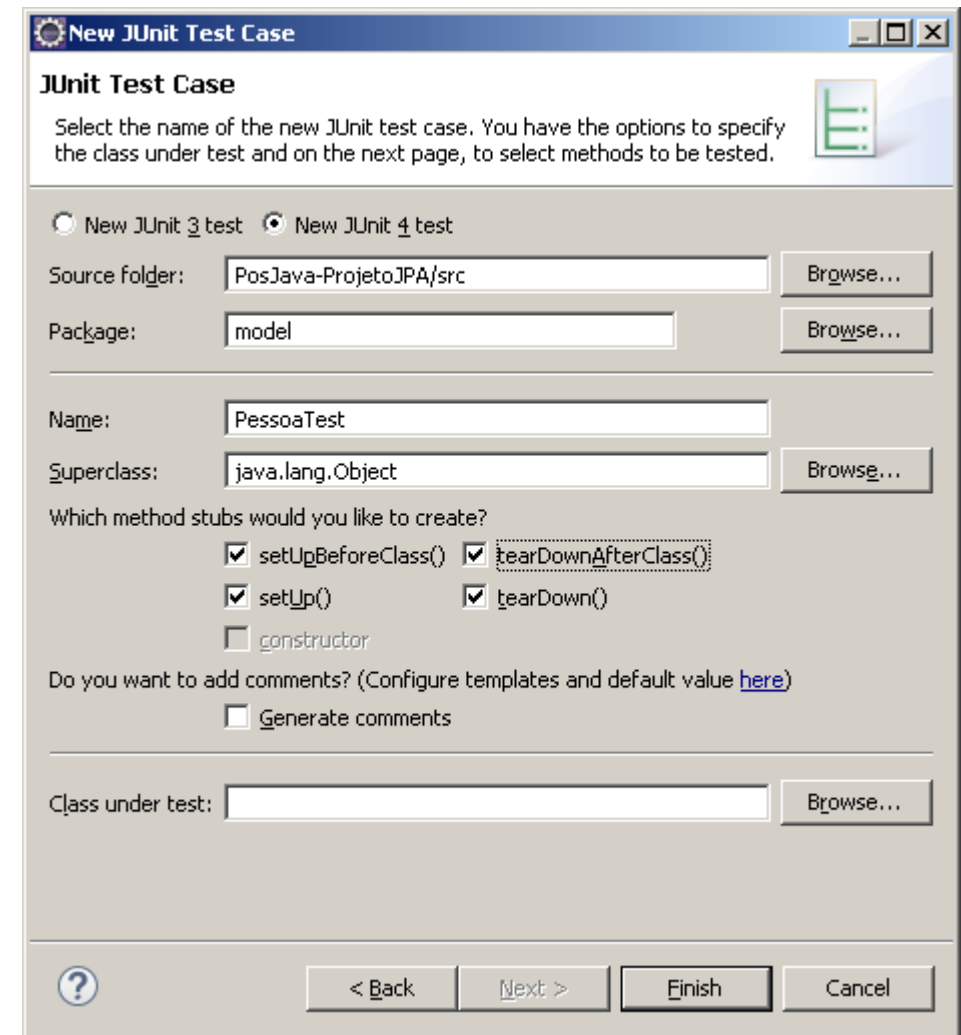
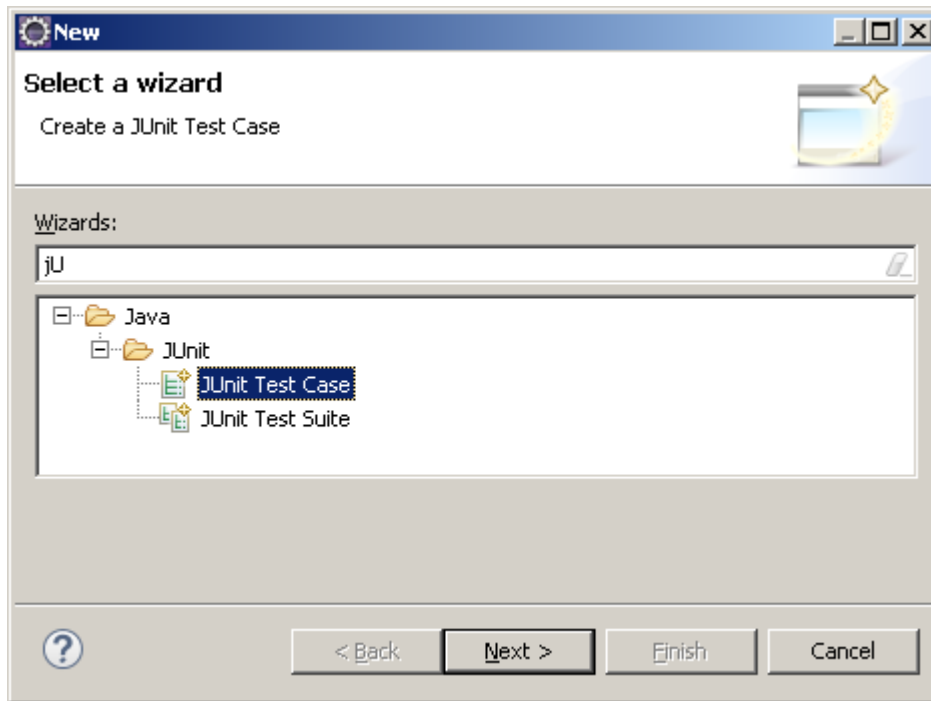
# Projeto Java com JPA - Eclipse

JPA Project: criando entidades a partir do BD



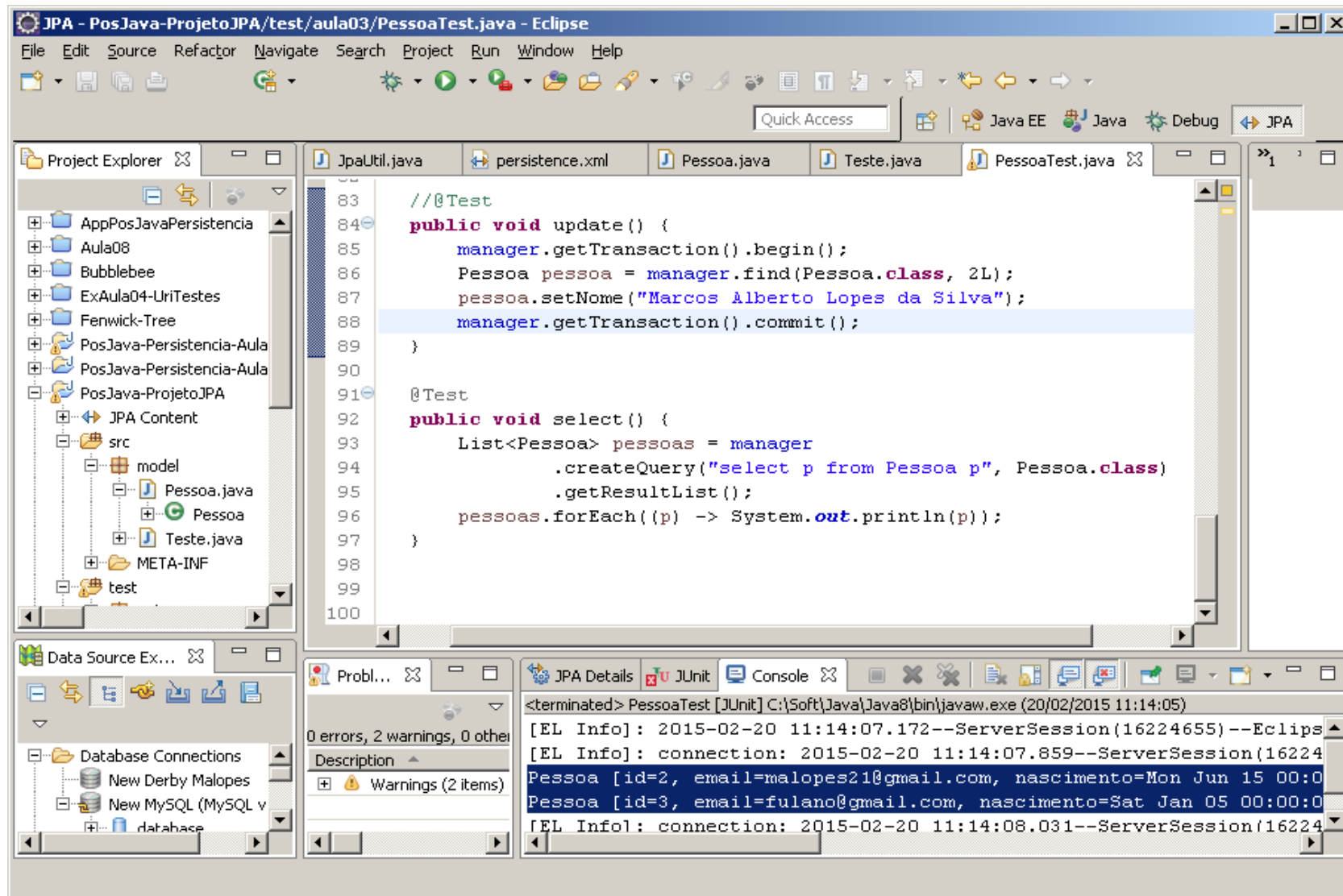
# Projeto Java com JPA - Eclipse

## JPA Project: executando os testes com jUnit



# Projeto Java com JPA - Eclipse

## JPA Project: executando os testes com jUnit





# Projeto Java com JPA - Eclipse

Persistence.xml adequado pra começar:

```
<?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="PosJava-ProjetoJPA">
        <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
        <class>model.Pessoa</class>
        <class>model.Teste</class>
        <properties>
            <property name="javax.persistence.jdbc.url"
value="jdbc:mysql://localhost:3306/posjava_aula03"/>
            <property name="javax.persistence.jdbc.user" value="root"/>
            <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
            <property name="javax.persistence.jdbc.password" value=""/>
            <property name="eclipselink.logging.level" value="FINE"/>
        </properties>
    </persistence-unit>
</persistence>
```

# JPA – mapeamento de tipos

Alguns tipos do Java são mapeados automaticamente para tipos correspondentes do banco de dados. São eles:

- Tipos primitivos (byte, short, char, int, long, float, double e boolean)
- ClassesWrappers (Byte, Short, Character, Integer, Long, Float, Double e Boolean)
- String
- BigInteger e BigDecimal
- java.util.Date e java.util.Calendar
- java.sql.Date, java.sql.Time e java.sql.Timestamp
- Array de byte ou char
- Enums
- Serializables

Esses tipos são chamados de tipos básicos.

# JPA – mapeamento de tipos

Eventualmente, dados maiores do que o comum devem ser armazenados no BD. Uma imagem, uma música ou um texto com muitas palavras.

Para esses casos, os BD oferecem tipos de dados específicos.

Do ponto de vista da JPA, basta aplicar a anotação @LOB (Large Objects) em atributos do tipo String, byte[], Byte[], char[] ou Character[]. Exemplo:

```
@Entity
public class Pessoa implements Serializable {
    @Id
    @GeneratedValue
    private Long id;
    private String nome;
    private String email;
    @Lob
    private byte [] avatar ;
}
```

# JPA – mapeamento de tipos

## Data e Hora

Atributos dos tipos `java.util.Date` e `java.util.Calendar` também são mapeadas automaticamente para tipos no BD. Exemplo:

```
@Entity
public class Pessoa implements Serializable {

    @Id
    private Long id;
    private Calendar nascimento ;
}
```

Por padrão, usando o tipo `java.util.Date` ou `java.util.Calendar`, tanto a data quanto a hora serão armazenadas no BD.

Para mudar esse comportamento, devemos aplicar a anotação `@Temporal` escolhendo uma das três formas:

# JPA – mapeamento de tipos

Atributos da anotação @Temporal:

**TemporalType.DATE:** Armazena apenas a data (dia, mês e ano).

**TemporalType.TIME:** Armazena apenas o horário (hora, minuto e segundo).

**TemporalType.TIMESTAMP** (Padrão): Armazena a data e o horário.

Exemplo:

```
@Entity
public class Pessoa implements Serializable {

    @Id
    private Long id;

    @Temporal(TemporalType.DATE)
    private Calendar nascimento ;
}
```

# JPA – mapeamento de tipos

## Dados Transientes:

Eventualmente, não desejamos que alguns atributos de um determinado grupo de objetos sejam persistidos no banco de dados.

Nesse caso, devemos aplicar o modificador transient ou a anotação @Transient. Exemplo:

```
@Entity
public class Pessoa implements Serializable {

    @Id
    private Long id;

    @Temporal(TemporalType.DATE)
    private Calendar nascimento;

    @Transient
    private int idade;
}
```

Faça um teste para ver o resultado!!

# JPA – transações

As modificações realizadas nos objetos administrados por um EntityManager são mantidas em memória.

A sincronização com o BD deve ser realizada através de uma transação JPA.

Para abrir uma transação utilizamos o método begin().

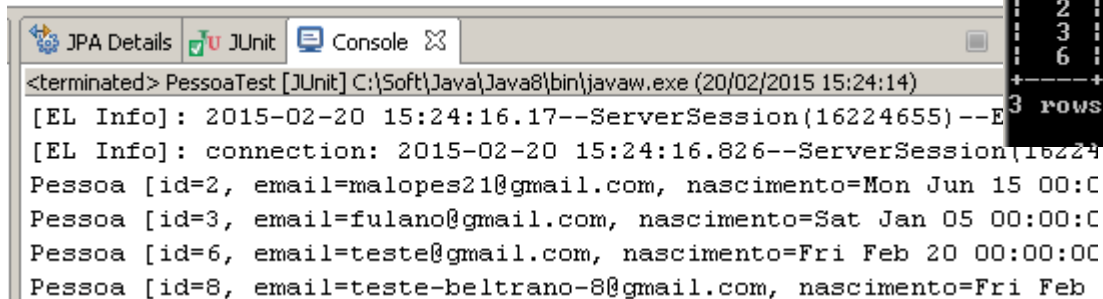
Com a transação aberta, podemos sincronizar os dados como banco através dos métodos:

- flush() (parcialmente)
- commit() (definitivamente).

# JPA – transações

## Exemplo de flush():

```
@Test
public void persist() {
    manager.getTransaction().begin();
    Pessoa pessoa = new Pessoa();
    pessoa.setEmail("teste-beltrano-8@gmail.com");
    pessoa.setNome("Beltrano o Tal Nro 8");
    pessoa.setNascimento(new Date());
    manager.persist(pessoa);
    manager.flush();
    List<Pessoa> pessoas = manager
        .createQuery("select p from Pessoa p", Pessoa.class)
        .getResultList();
    pessoas.forEach((p) -> System.out.println(p));
    //manager.getTransaction().commit();
}
```



JPA Details JUnit Console

<terminated> PessoaTest [JUnit] C:\Soft\Java\Java8\bin\javaw.exe (20/02/2015 15:24:14)

[EL Info]: 2015-02-20 15:24:16.17--ServerSession(16224655) --E

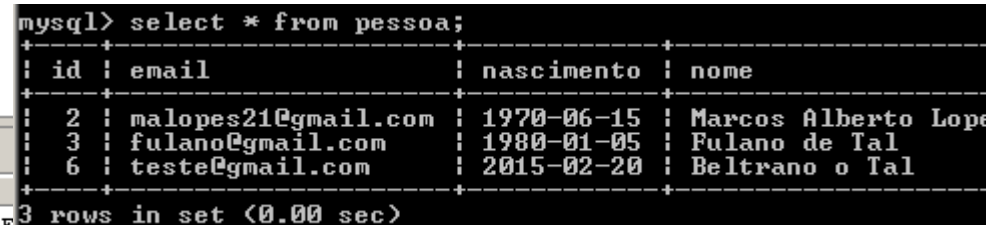
[EL Info]: connection: 2015-02-20 15:24:16.826--ServerSession(16224

Pessoa [id=2, email=malopes21@gmail.com, nascimento=Mon Jun 15 00:00:00

Pessoa [id=3, email=fulano@gmail.com, nascimento=Sat Jan 05 00:00:00

Pessoa [id=6, email=teste@gmail.com, nascimento=Fri Feb 20 00:00:00

Pessoa [id=8, email=teste-beltrano-8@gmail.com, nascimento=Fri Feb



```
mysql> select * from pessoa;
+----+-----+-----+-----+
| id | email                | nascimento | nome                |
+----+-----+-----+-----+
| 2  | malopes21@gmail.com  | 1970-06-15 | Marcos Alberto Lope |
| 3  | fulano@gmail.com     | 1980-01-05 | Fulano de Tal       |
| 6  | teste@gmail.com      | 2015-02-20 | Beltrano o Tal      |
+----+-----+-----+-----+
3 rows in set (0.00 sec)
```



# JPA – mapeamento dos relacionamentos

O mapeamento dos relacionamentos determina como a ferramenta de ORM fará consultas envolvendo mais do que uma tabela do BD.

Os relacionamentos entre as entidades de um domínio devem ser expressos na modelagem através de vínculos entre classes.

De acordo com a JPA, podemos definir quatro tipos de relacionamentos de acordo com a cardinalidade.

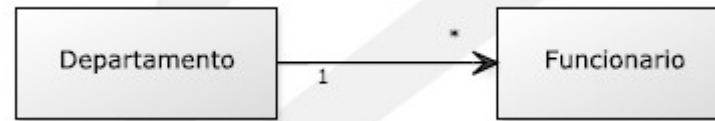
- @OneToOne (default eager)
- @OneToMany (default eager)
- @ManyToOne (default lazy)
- @ManyToMany (default lazy)

# Hibernate/Jpa - relations

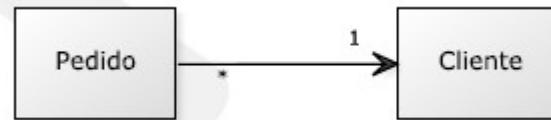
- @OneToOne



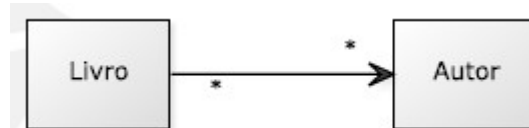
- @OneToMany



- @ManyToOne



- @ManyToMany



Relacionamentos xxToOne – default EAGER

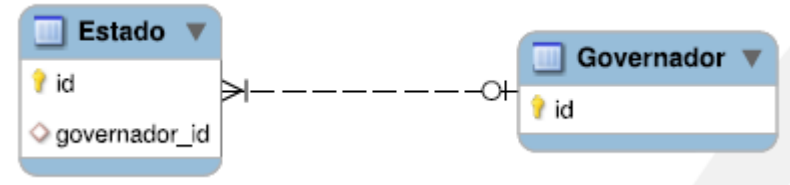
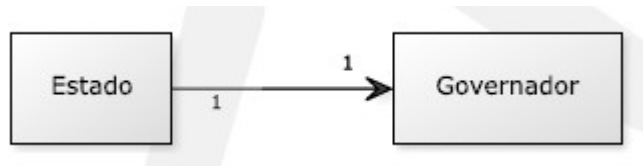
Relacionamentos xxToMany – default LAZY

# O problema do eager/lazy

- Por padrão todo objeto correlacionado é obtido antecipadamente (eager)
- E toda coleção correlacionada é obtida com retardo ou atrasada (lazy)
- E se eu tenho uma tabela pessoa com referência à endereço, cidade, estado, cliente, fornecedor e preciso apenas buscar o nome da pessoa?

```
@Test
public void selectNomes() {
    List<String> pessoas = manager
        .createQuery("select p.nome from Pessoa p", String.class)
        .getResultList();
    pessoas.forEach((p) -> System.out.println("Ops2: "+p));
}
```

# OneToOne



@Entity

```
public class Estado implements Serializable {
```

```
    private static final long serialVersionUID = 1L;
```

```
    @Id
```

```
    @GeneratedValue(strategy=GenerationType.IDENTITY)
```

```
    private Long id;
```

```
    @OneToOne
```

```
    private Governador governador;
```

```
}
```

@Entity

```
public class Governador {
```

```
    @Id
```

```
    @GeneratedValue(strategy=GenerationType.IDENTITY)
```

```
    private Long id;
```

```
}
```

Podemos alterar o nome padrão das join columns aplicando a anotação @JoinColumn, conforme apresentado no exemplo abaixo.

```
@OneToOne
```

```
@JoinColumn (name="gov_id")
```

```
private Governador governador;
```

# OneToOne

## Demonstração OneToOne:

```
@Test
public void adicionaGovernadorEstado(){
    manager.getTransaction().begin();

    Governador g = new Governador();
    g.setNome("Geraldo Alckmin");

    Estado e = new Estado();
    e.setNome("São Paulo");
    e.setGovernador(g);

    manager.persist(g);
    manager.persist(e);

    manager.getTransaction().commit();
    //manager.close();
}
```

```
mysql> use posjava_aula03;
Database changed
mysql> select * from estado;
+----+-----+-----+
| ID | NOME      | gov_id |
+----+-----+-----+
| 1  | São Paulo | 1      |
+----+-----+-----+
1 row in set (0.00 sec)

mysql> select * from governador;
+----+-----+
| ID | NOME      |
+----+-----+
| 1  | Geraldo Alckmin |
+----+-----+
1 row in set (0.00 sec)

mysql> _
```

# OneToMany e ManyToOne

@Entity

```
public class Departamento implements Serializable {
```

```
    private static final long serialVersionUID = 1L;
```

```
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Long id;
```

```
    @OneToMany(mappedBy="departamento", cascade={CascadeType.ALL})
```

```
    private Collection<Funcionario> funcionarios;
```

```
    private String nome;
```

```
    private String descricao;
```

```
}
```

@Entity

```
class Funcionario implements Serializable {
```

```
    private static final long serialVersionUID = 1L;
```

```
    @Id
```

```
    @GeneratedValue(strategy=GenerationType.IDENTITY)
```

```
    private Long id;
```

```
    @ManyToOne
```

```
    //@JoinColumn(name="dep_id")
```

```
    private Departamento departamento;
```

```
    private String nome;
```

```
}
```



```
mysql> describe funcionario;
+-----+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| ID             | bigint(20)    | NO   | PRI | NULL    | auto_increment |
| NOME           | varchar(255)  | YES  |     | NULL    |                |
| DEPARTAMENTO_ID | bigint(20)    | YES  | MUL | NULL    |                |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.01 sec)

mysql> describe departamento;
+-----+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| ID             | bigint(20)    | NO   | PRI | NULL    | auto_increment |
| DESCRICAO      | varchar(255)  | YES  |     | NULL    |                |
| NOME           | varchar(255)  | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.01 sec)
```

Podemos alterar o nome padrão das join columns e etc.

```
@ManyToOne
```

```
@JoinColumn(nullable=false)
```

```
private Departamento departamento;
```

# OneToMany eManyToOne



@Test

```
public void adicionaDepartamentoFuncionarios() {
    manager.getTransaction().begin();

    Departamento departamento = new Departamento();
    departamento.setNome("Financeiro");
    departamento.setDescricao("Lida com a grana");

    Funcionario funcionario = new Funcionario();
    funcionario.setNome("Fulano de Tal");
    funcionario.setDepartamento(departamento);

    departamento.setFuncionarios(new ArrayList<Funcionario>());
    departamento.getFuncionarios().add(funcionario);

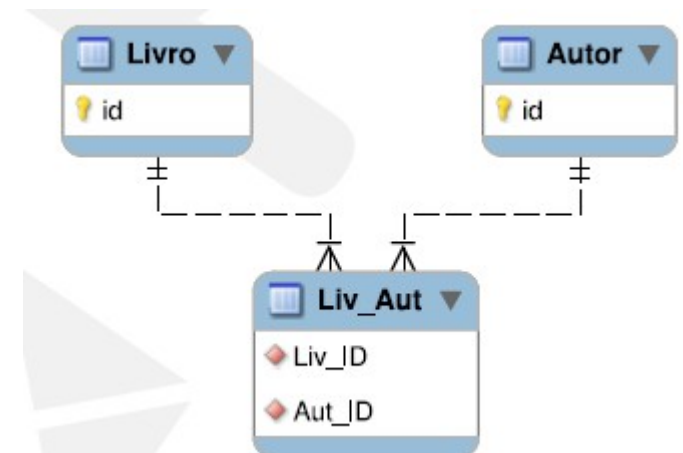
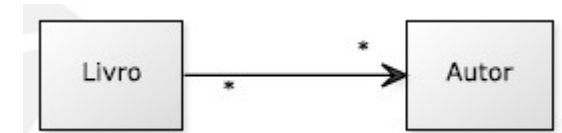
    manager.persist(departamento);
    //manager.persist(funcionario);

    manager.getTransaction().commit();
    // manager.close();
}
```

# ManyToMany

```
@Entity
public class Livro implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
    private String editora;
    @ManyToMany
    @JoinTable(name = "Liv_Aut",
        joinColumns = @JoinColumn(name = "Liv_ID"),
        inverseJoinColumns = @JoinColumn(name = "Aut_ID"))
    private Collection<Autor> autores;
}

@Entity
public class Autor implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
    @ManyToMany
    @JoinTable(name = "Liv_Aut",
        joinColumns = @JoinColumn(name = "Aut_ID"),
        inverseJoinColumns = @JoinColumn(name = "Liv_ID"))
    private Collection<Livro> livros;
}
```





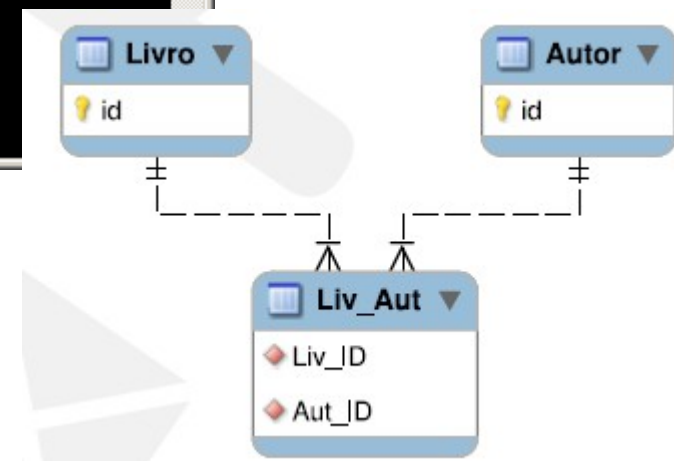
# ManyToMany

```
mysql> describe autor;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| ID     | bigint(20)    | NO   | PRI | NULL    |       |
| NOME   | varchar(255)  | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.01 sec)

mysql> describe livro;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| ID         | bigint(20)    | NO   | PRI | NULL    |       |
| EDITORA    | varchar(255)  | YES  |     | NULL    |       |
| NOME       | varchar(255)  | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.01 sec)

mysql> describe liv_aut;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| Aut_ID     | bigint(20)    | NO   | PRI | NULL    |       |
| Liv_ID     | bigint(20)    | NO   | PRI | NULL    |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.02 sec)

mysql>
```



# ManyToMany

```
@Test
public void adicionaAutorLivros() {

    manager.getTransaction().begin();

    Autor autor = new Autor();
    autor.setNome("Malopes");

    Livro livro = new Livro();
    livro.setNome("Testes de JPA2");
    livro.setAutores(new ArrayList<Autor>());
    livro.getAutores().add(autor);

    manager.persist(autor);
    manager.persist(livro);

    manager.getTransaction().commit();

}
```

```
mysql> select * from autor;
+----+-----+
| ID | NOME  |
+----+-----+
| 1  | Malopes |
+----+-----+
1 row in set (0.00 sec)

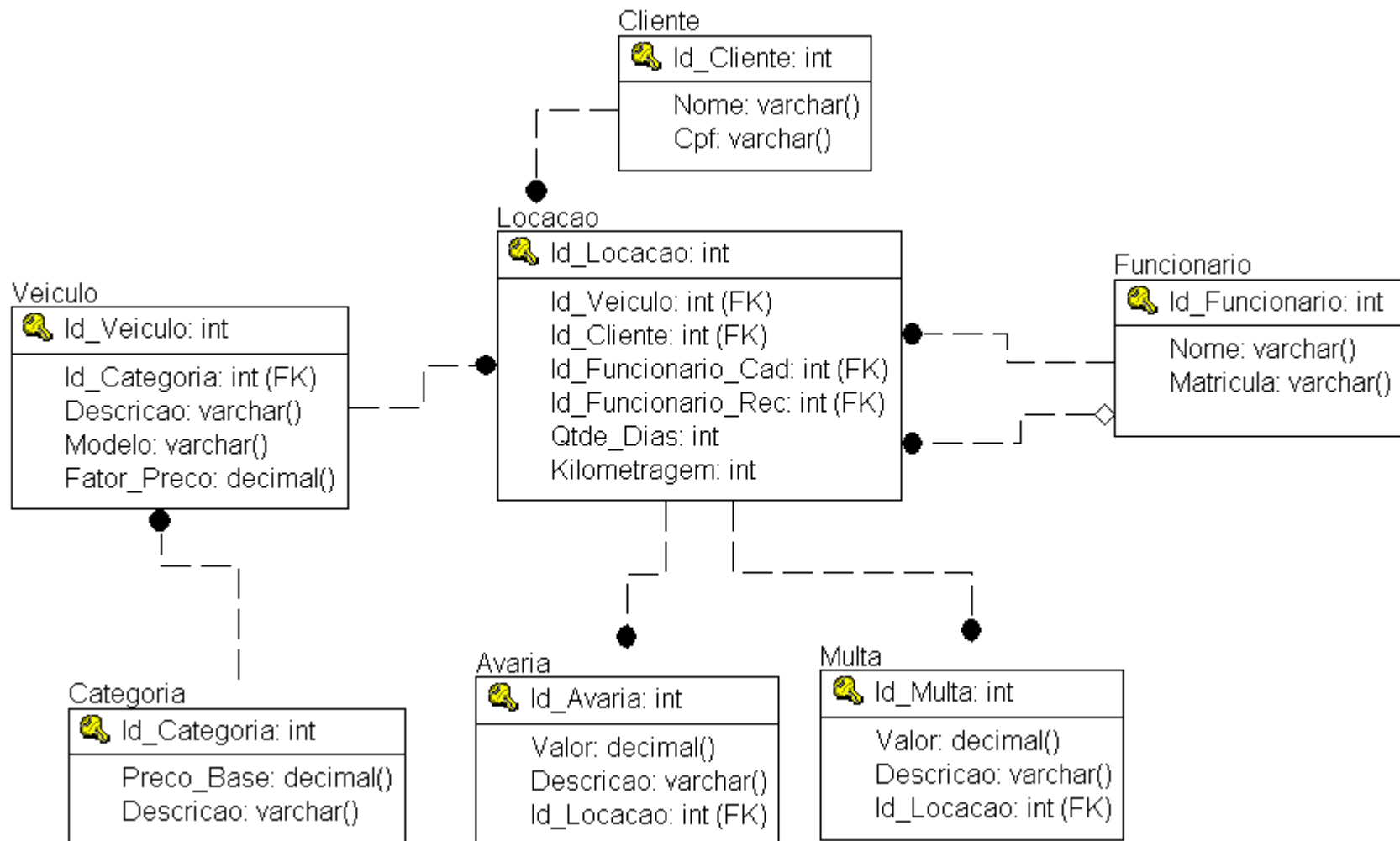
mysql> select * from livro;
+----+-----+-----+
| ID | EDITORA | NOME          |
+----+-----+-----+
| 1  | NULL    | Testes de JPA2 |
+----+-----+-----+
1 row in set (0.00 sec)

mysql> select * from liv_aut;
+-----+-----+
| Aut_ID | Liv_ID |
+-----+-----+
| 1      | 1      |
+-----+-----+
1 row in set (0.00 sec)
```

# Exercícios

- Construir casos de teste para executar operações de find, remove, update sobre as entidades trabalhadas durante a aula;
- Construir o modelo OO (outro projeto) referente ao modelo ER no próximo slide;
- Construir um aplicativo Java standalone console com operações básicas, via JPA, como: (cad. Cliente, Funcionario, Categoria, Veiculo, Locacao, Multa, Avaria), (ed. Cliente, Funcionario, Categoria, Veiculo, Locacao, Multa, Avaria), (del. Cliente, Funcionario, Categoria, Veiculo, Locacao, Multa, Avaria), (buscar Cliente, Funcionario, Categoria, Veiculo, Locacao, Multa, Avaria), (selec. Cliente, Funcionario, Categoria, Veiculo, Locacao, Multa, Avaria)

# Exercícios



# Referências bibliográficas

- [1] Bauer, Christian e King, Gavin – Java persistence com Hibernate. Rio de Janeiro, Ed. Ciência Moderna, 2007;
- [2] Apostila "Persistência com JPA2 e Hibernate", K19 online.