DESARROLLO DE APLICACIONES WEB

Tema 5.- Java Entreprise Edition

Modelo - Acceso a Datos: JPA

Contenido

- **Modelos. Acceso a Datos**
  - JDBC
  - Pool Connection
  - JavaBeans
  - JPA

Contenido

- **Modelos. Acceso a Datos**
  - **JDBC**
  - Pool Connection
  - JavaBeans
  - JPA

Acceso a BD - JDBC

– Pasos: Es necesario importar el paquete java.sql.*

Ver javadoc API

1) Cargar el Driver. Posteriormente será manipulado con el *DriverManager*

Class.forName(“Class.forName("com.mysql.cj.jdbc.Driver"); // usamos mySQL

(Es necesario disponer del Driver y agregarlo a las librerías de nuestra aplicación)

2) Establecer la conexión. Crear un objeto *Connection* desde el *DriverManager*.

Connection bd;
bd = DriverManager.getConnection(”jdbc:mysql://lhost:port/baseDatos", “Usu", “Pass");

Donde: *BaseDatos* es la base de datos que deseamos utilizar
Usu es el usuario de la base de dato y *Pass* la clave de acceso

Este método lanza una **SQLException** si se produce un error en la conexión

Acceso a BD - JDBC

3) Crear una sentencia SQL con el objeto Statement desde el objeto *Connection*.

```
Statement st;
st = bd.createStatement();
```

El método lanza una **SQLException** si se produce un error en la base de datos.

4) Preparar la sentencia SQL

Mediante un objeto *PreparedStatement* del objeto *Connection*.

```
PreparedStatement ps;
ps = bd.preparedStatement("SELECT * FROM ? WHERE n= ? AND f< ? ";
ps.setString(1, tabla);
ps.setInt(2, 10);
ps.setFloat(3, 3.14);
```

El método lanza una **SQLException** si se produce un error en la base de datos.

Acceso a BD - JDBC

5) Ejecutar la sentencia

A partir del objeto *PreparedStatement*

Dependiendo de la sentencia a ejecutar se utilizan, básicamente, los métodos:

execute() : Para cualquier sentencia SQL.
Este método lanza una **SQLException** si se produce un error

executeQuery() : Para una consulta SELECT. Devuelve un objeto ResultSet.
Lanza **SQLException** si se produce un error o no se obtiene un objeto Resultset.

executeUpDate(): Para una consulta que no devuelva resultados de la Base Datos
Devuelve un entero que indica las filas insertadas, actualizadas o borradas, ó 0.
Lanza **SQLException** si se produce un error o se obtiene un objeto Resultset.

```
Ej1.    ResultSet rs = st.executeQuery("SELECT * FORM Articulos");
Ej2.    st.execute(sql);  // donde sql es un String con una sentencia SQL
Ej3.    ResultSet rs = ps.executeQuery();
Ej4.    ps.executeUpDate();
```

Acceso a BD - JDBC

6) Para las consultas, acceder a la información obtenida (ResultSet)

Un objeto *ResultSet* mantiene un cursor apuntando a los datos obtenidos.

Ej: ResultSet rs = st.executeQuery("SELECT * FORM Articulos");

Principales métodos:

getString(*pos*) : Devuelve el String almacenado en la columna pos.

También, se puede usar el nombre de la columna.

getInt(pos) : Devuelve el Integer almacenado en la columna pos.

next() : Cambia a la siguiente fila, devuelve verdadero si existe o falso en caso contrario.

Ej.

```
while ( rs.next() ) {
        c1 = rs.getString(1);
        c2 = rs.getInt(2);
        c3 = rs.getFloat(3);
}
```

Nota 1: Es muy buena práctica liberar los recursos con close(). [Connection/ Statement / ResultSet]

Nota 2: Para eliminar los espacios en blanco de una cadena se puede usar trim();

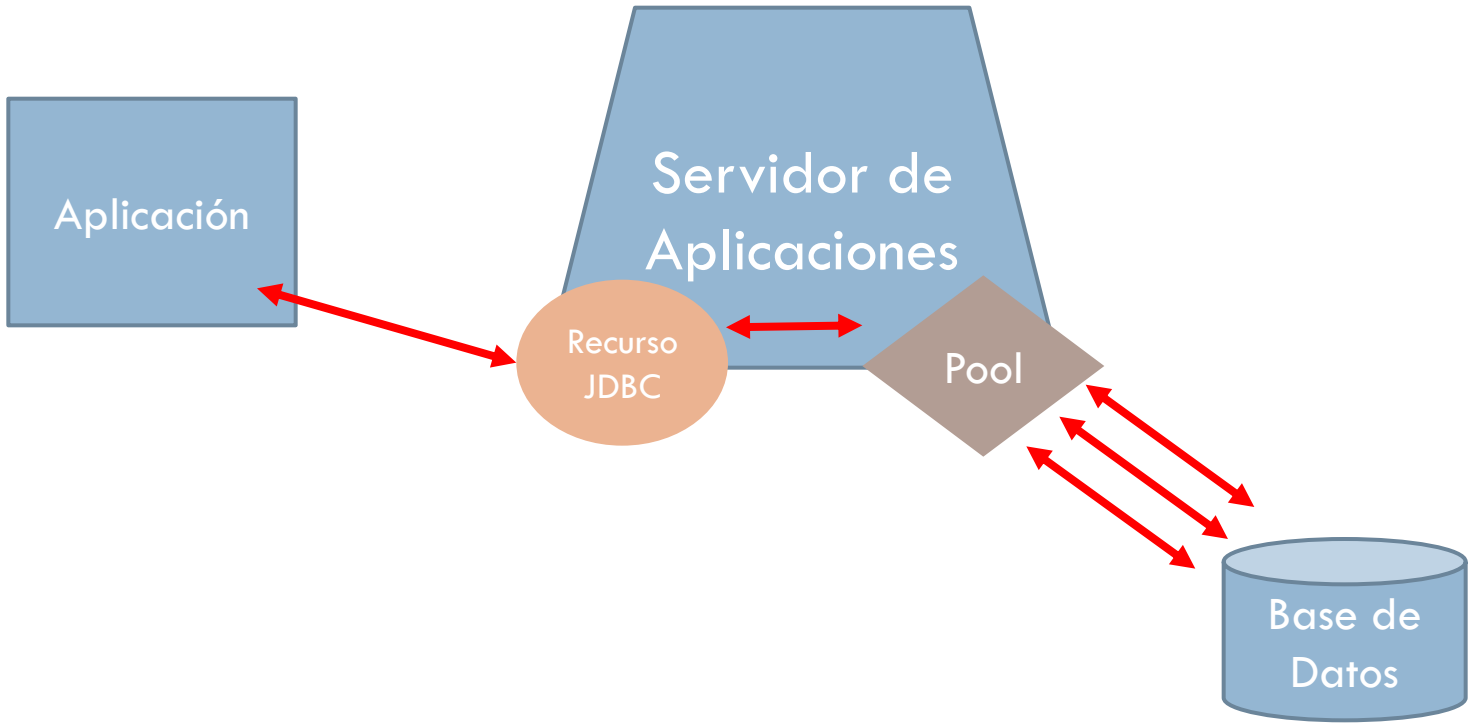
Contenido

- **Modelos. Acceso a Datos**
  - JDBC
  - **Pool Connection**
  - JavaBeans
  - JPA

Acceso a Datos

- Pool de Conexiones

  Esquema de funcionamiento



Acceso a Datos

- Pool de Conexiones: Configuración del Servidor (Glassfish)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE resources PUBLIC "-//GlassFish.org//DTD GlassFish Application Server 3.1 Resource Definitions//EN"
                           "http://glassfish.org/dtds/glassfish-resources_1_5.dtd">
<resources>
  <jdbc-resource enabled="true" jndi-name="jdbc/agenda" object-type="user" pool-name="connectionPoolAgenda">
    <description/>
  </jdbc-resource>
  <jdbc-connection-pool allow-non-component-callers="false" associate-with-thread="false"
                        connection-creation-retry-attempts="0" connection-creation-retry-interval-in-seconds="10"
                        connection-leak-reclaim="false" connection-leak-timeout-in-seconds="0"
                        connection-validation-method="auto-commit"
                        datasource-classname="org.apache.derby.jdbc.ClientDataSource"
                        fail-all-connections="false" idle-timeout-in-seconds="300"
                        is-connection-validation-required="false" is-isolation-level-guaranteed="true"
                        lazy-connection-association="false" lazy-connection-enlistment="false" match-connections="false"
                        max-connection-usage-count="0" max-pool-size="32" max-wait-time-in-millis="60000"
                        name="connectionPoolAgenda" non-transactional-connections="false" pool-resize-quantity="2"
                        res-type="javax.sql.ConnectionPoolDataSource" statement-timeout-in-seconds="-1"
                        steady-pool-size="8" validate-atmost-once-period-in-seconds="0" wrap-jdbc-objects="false">
    <property name="URL" value="jdbc:derby://localhost:1527/aganda_db"/>
    <property name="serverName" value="localhost"/>
    <property name="PortNumber" value="1527"/>
    <property name="DatabaseName" value="aganda_db"/>
    <property name="User" value="app"/>
    <property name="Password" value="app"/>
  </jdbc-connection-pool>
</resources>
```

Acceso a Datos

- Pool de Conexiones: Conexión desde la Aplicación con JDBC

Clases:

```
javax.sql.DataSource
javax.naming.InitialContext
java.sql.Connection
```

  - Inyectar un atributo para acceso al Pool

```
@Resource(name = "PoolArticulosDB")
private DataSource poolArticulosDB;
```

  - Establecer la conexión y el resto de operaciones

```
try {
    // establecer la conexión
    Context c = new InitialContext();
    poolArticulosDB = (DataSource) c.lookup("jdbc/bdArticulos");
    conn = poolArticulosDB.getConnection();

    // Preparar la sentencia SQL a realizar
    ps = conn.prepareStatement("SELECT * FROM PRODUCTOS");
    rs = ps.executeQuery();

    //...
```

Contenido

- **Modelos. Acceso a Datos**
  - JDBC
  - Pool Connection
  - **JavaBeans**
  - JPA

JavaBeans

- JavaBeans es la tecnología de componentes de la plataforma Java. Los componentes (Beans) son clases java reutilizables, que pueden ser compartidos fácilmente por varias aplicaciones Java.

- En general, un Bean es una clase que obedece a ciertas reglas:

  - Un bean debe tener un constructor por defecto (sin argumentos).
  - Un bean tiene que tener persistencia, es decir, implementar la interface Serializable.

  - Un bean debe tener introspección (instrospection), mediante setters y getters (métodos set… y get...) para cada uno de sus atributos.

    public void set*NombrePropiedad***(***TipoPropiedad* valor**);**
    public *TipoPropiedad* **get***NombrePropiedad***( );**

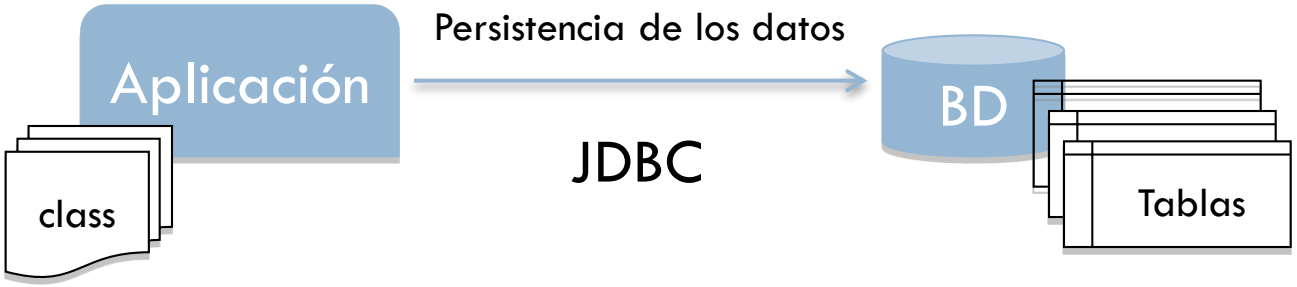
JavaBeans: Ejemplo

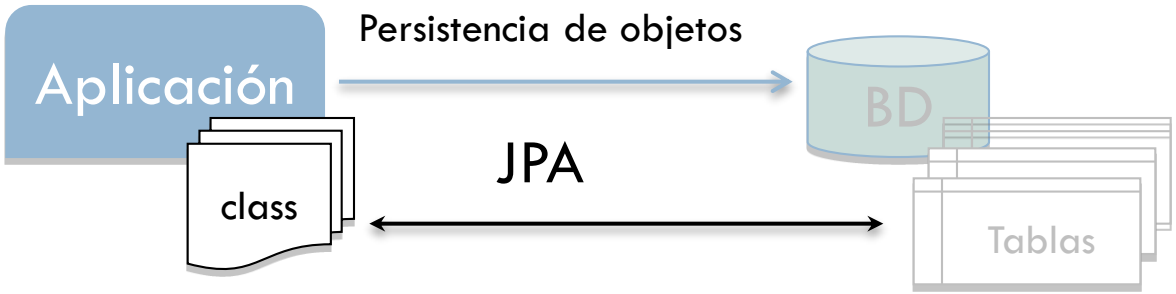
```
package edu.daw.ejBean;
public class miBean implements java.io.Serializable {
    private String nombre;
    private int edad;
    public miBean() {
    }
    public String getNombre() {
       return nombre;
    }
    public void setNombre(String nombre){
       this.nombre = nombre;
    }
    public int getEdad(){
       return edad;
    }
    public void setEdad(int edad){
       this,edad = edad;
    }}
```

Contenido

- **Modelos. Acceso a Datos**
  - JDBC
  - Pool Connection
  - JavaBeans
  - **JPA**

Acceso a Datos - JPA





Acceso a Datos – JPA

- JPA
  - ORM (Object-Relational Mapping) Mapeo Objeto-Relación
  - Unidad de Persistencia
  - Entidades: Entity Manager
  - JTA (Java Transaction API)
  - JPQL (Java Persistence Query Language)

http://en.wikibooks.org/wiki/Java_Persistence

Contenido

- **Modelos. Acceso a Datos**
  - JDBC
  - Pool Connection
  - JavaBeans
  - **JPA**
    - **Entidades**

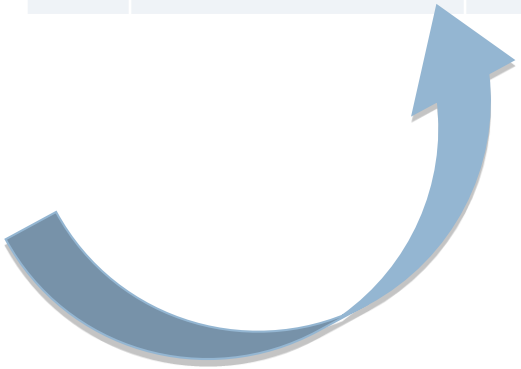
Acceso a Datos – JPA

- Entidades

```
@Entity
public class Persona implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nombre;
    private Date fecha;
    private int edad;

   // Aquí vendrían los getters y setters
}
```

Persona			
Id	nombre	fecha	edad



Acceso a Datos – JPA

- Entidades

```
@Entity
@Table(name = ”Tabla_Personas")
public class Persona implements Serializable {
    @Id
    @Column(name = "ID_P")
    private Long id;
    ...
}
```

Tabla_Personas			
ID_P	Nombre	fecha	edad

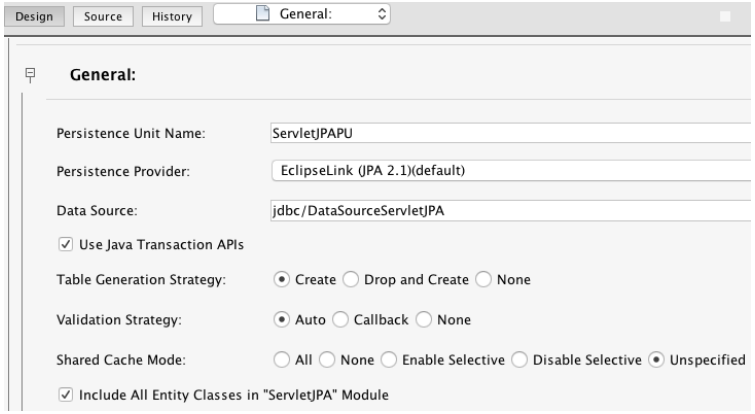
Contenido

- **Modelos. Acceso a Datos**
  - JDBC
  - Pool Connection
  - JavaBeans
  - **JPA**
    - **Unidad de Persistencia** (configuración ORM)

Acceso a Datos – JPA: Persistencia

- Unidad de Persistencia (fichero persitence.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
  <persistence version="3.0" … >
     <persistence-unit name="AgendaPU" transaction-type="JTA">
       <jta-data-source>jdbc/agenda</jta-data-source>
       <exclude-unlisted-classes>false</exclude-unlisted-classes>
       <properties>
         <property name="jakarta.persistence.schema-generation.database.action" value="create"/>
       </properties>
     </persistence-unit>
  </persistence>
```



Contenido

- **Modelos. Acceso a Datos**
  - JDBC
  - Pool Connection
  - JavaBeans
  - **JPA**
    - **Gestor de Entidades** (Entity Manager)

Acceso a Datos – JPA: EntityManager

- Gestor de Entidades: Entity Manager

Ver javadoc API

```
@Entity
public class Persona implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nombre;

    // Getters y Setters
    …
}
```

Acceso a Datos – JPA: EntityManager

- Gestor de Entidades: Entity Manager

Ver javadoc API

```
@PersistenceContext(unitName = “AgendaPU")
private EntityManager em;
@Resource
private javax.transaction.UserTransaction utx;
…

public void save(Object object) {
    try {
       utx.begin();
       em.persist(object);
       utx.commit();
    } catch (Exception e) { … }
}
```

```
Persona p = new Persona();
p.setNombre("Pepe");

try {
    save(p);
} catch (Exception ex) {   …  }
```

Acceso a Datos – JPA: EntityManager

Ver javadoc API

- Principales métodos EntityManager (em)
  - persist(obj): persiste el objeto en la BD

    ```
    Persona p = new Persona(); // creamos objeto con sus datos
    em.persist(p);
    ```

  - merge(obj): Actualiza el objeto en la BD

    ```
    Persona p = … ; // Obtenemos objeto, modificamos datos
    em.merge(p);
    ```

  - find(Nombre.class, id): devuelve el objeto de la BD con clave id

    ```
    Persona p = em.find(Persona.class, 1);
    ```

  - remove(obj): Elimina el objeto de la BD

    ```
    em.remove(p); // una vez tenemos un objeto p lo elimina (trasaccional)
    ```

  - flush(obj): Fuerza el almacenamiento del objeto en la BD
  - refresh(obj): Fuerza la actualización del objeto con los datos de la BD

Contenido

- **Modelos. Acceso a Datos**
  - JDBC
  - Pool Connection
  - JavaBeans
  - **JPA**
    - **Relaciones entre entidades**
      - OneToOne
      - ManyToOne (OneToMany)
      - ManyToMany

Acceso a Datos – JPA: Relaciones

- OneToOne

```
@Entity
public class Persona {
    @Id
    private Long id;
     …
    @OneToOne
    @JoinColumn(name = "DIR") // Opcional
    private Direccion direccion;

    // Getters y setters
}
```

```
@Entity
public class Direccion {
    @Id
    private Long id;
    private String calle;
    private String ciudad;

    // Getters y setters
}
```

Persona		
Id	…	DIR

Direccion		
id	calle	ciudad

Acceso a Datos – JPA: Relaciones

- OneToOne Bidireccional

```
@Entity
public class Persona {
   @Id
   private Long id;
    …
   @OneToOne
   @JoinColumn(name = "DIR") // Opcional
   private Direccion direccion;

   // Getters y setters
}
```

```
@Entity
public class Direccion {
   @Id
   private Long id;
   private String calle;
   private String ciudad;

   @OneToOne(mappedBy=”direccion")
   private Persona owner;

   // Getters y setters
```

Persona		
Id	…	DIR

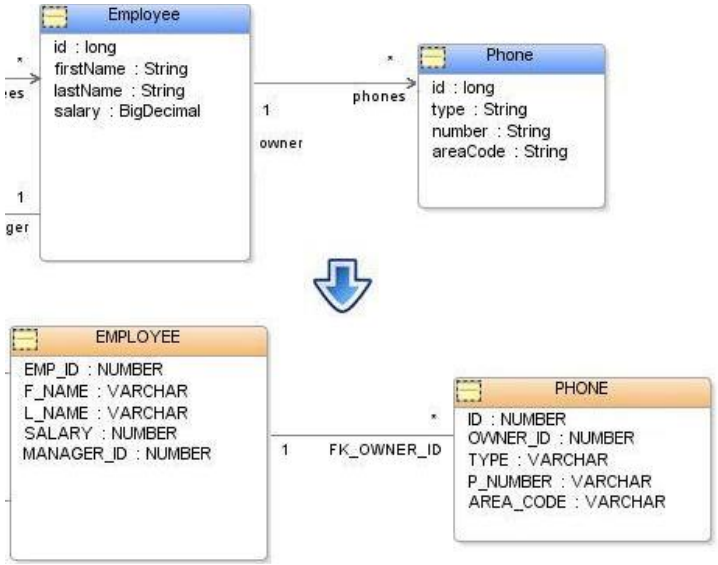
Direccion		
id	calle	ciudad

Acceso a Datos – JPA: Relaciones

- ManyToOne

```
@Entity
public class Employee {
…
}


@Entity
public class Phone {
  @Id
  private long id;
  …
  @ManyToOne(fetch=FetchType.LAZY)
  @JoinColumn(name="OWNER_ID") // Opcional
  private Employee owner;
  …
}
```



Estrategia

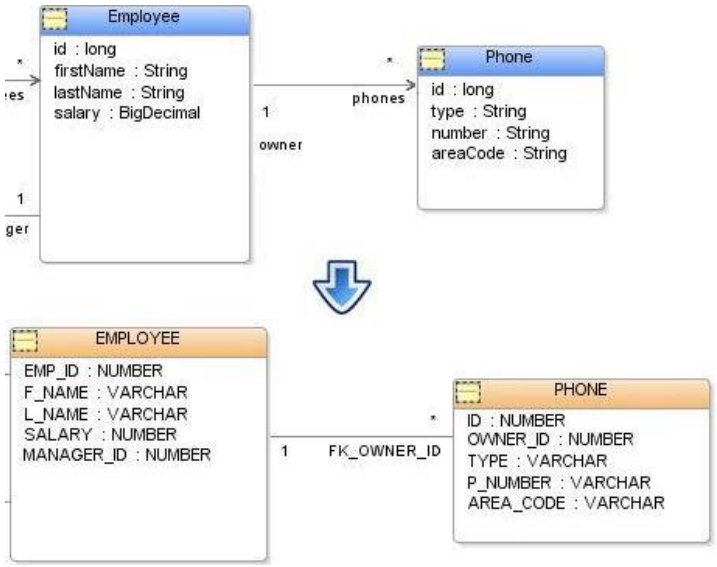
fetch=FetchType.LAZY

fetch=FetchType.EAGER

Acceso a Datos – JPA: Relaciones

- OneToMany (Bidireccional ManyToOne)

```
@Entity
public class Employee {
@Id
@Column(name="EMP_ID") // Opcional
private long id;
...
@OneToMany(mappedBy="owner“,
                    cascade = CascadeType.PERSIST)
private List<Phone> phones;
...
}
```



Acceso a Datos – JPA: Relaciones

- ManyToMany

```
@Entity
public class Employee {
  @Id
  @Column(name="ID")
  private long id;
  ...
  @ManyToMany
  @JoinTable(  // Opcional
      name="EMP_PROJ",
      joinColumns={@JoinColumn(name="EMP_ID", referencedColumnName="ID")},
      inverseJoinColumns={@JoinColumn(name="PROJ_ID", referencedColumnName="ID")})
  private List<Project> projects;
  ...
}
```

EMPLOYEE (table)

ID	FIRSTNAME	LASTNAME
1	Bob	Way
2	Sarah	Smith

EMP_PROJ (table)

EMP_ID	PROJ_ID
1	1
1	2
2	1

PROJECT (table)

ID	NAME
1	GIS
2	SIG

Acceso a Datos – JPA: Relaciones

- ManyToMany

```
@Entity
public class Project {
@Id
@Column(name="ID")
private long id;
...

@ManyToMany(mappedBy="projects")
private List<Employee> employees;
...
}
```

EMPLOYEE (table)

ID	FIRSTNAME	LASTNAME
1	Bob	Way
2	Sarah	Smith

EMP_PROJ (table)

EMP_ID	PROJ_ID
1	1
1	2
2	1

PROJECT (table)

ID	NAME
1	GIS
2	SIG

Contenido

- **Modelos. Acceso a Datos**
  - JDBC
  - Pool Connection
  - JavaBeans
  - **JPA**
    - **Consultas**
      - Estáticas (NamedQueries) y Dinámicas
      - Genéricas y Tipadas

Acceso a Datos – JPA: Consultas JPQL

- Estáticas
  - Named Query (mediante anotaciones)

```
@Entity
@NamedQuery(name="Persona.findByName",
        query ="SELECT p FROM Persona p WHERE p.name LIKE :pnombre";
public class Persona … {
        …
}
```

```
public List<Persona> findPersonasByName(String nombre) {
        List<Persona> lp;
        Query q = em. createNamedQuery("Persona.findByName");
        q.setParameter("pnombre", nombre);
        lp = q.getResultList();
         return lp;
}
```

Acceso a Datos – JPA: Consultas JPQL

- Dinámicas
  - Genéricas (sin tipo específico)

```
…
Query q = em.createQuery(“SELECT p FROM Persona p”;
List resultados = q.getResultList();
Personas p = (Persona) resultados.get(0)
…
```

  - Tipadas (con tipo concreto)

```
…
TypedQuery<Persona> q =  em.createQuery("SELECT p FROM Persona p", Persona.class);
List<Persona> results = query.getResultList();
…
```

Acceso a Datos – JPA: Consultas JPQL

- Aclaración
  - Tanto las Query (genéricas) como las TypeQuery (tipadas) pueden usar 'variables'
    - Variables: Creación con :var / asignación con setParameter()

      Por ejemplo:

```
Query query = em.createQuery("SELECT e FROM Employee e
                                        WHERE e.firstName = :first and e.lastName = :last");
query.setParameter("first", "Bob");
query.setParameter("last", "Smith");
List<Employee> list = query.getResultList();
```

- Más información sobre JPQL

  https://en.wikibooks.org/wiki/Java_Persistence/Querying

  https://en.wikibooks.org/wiki/Java_Persistence/JPQL