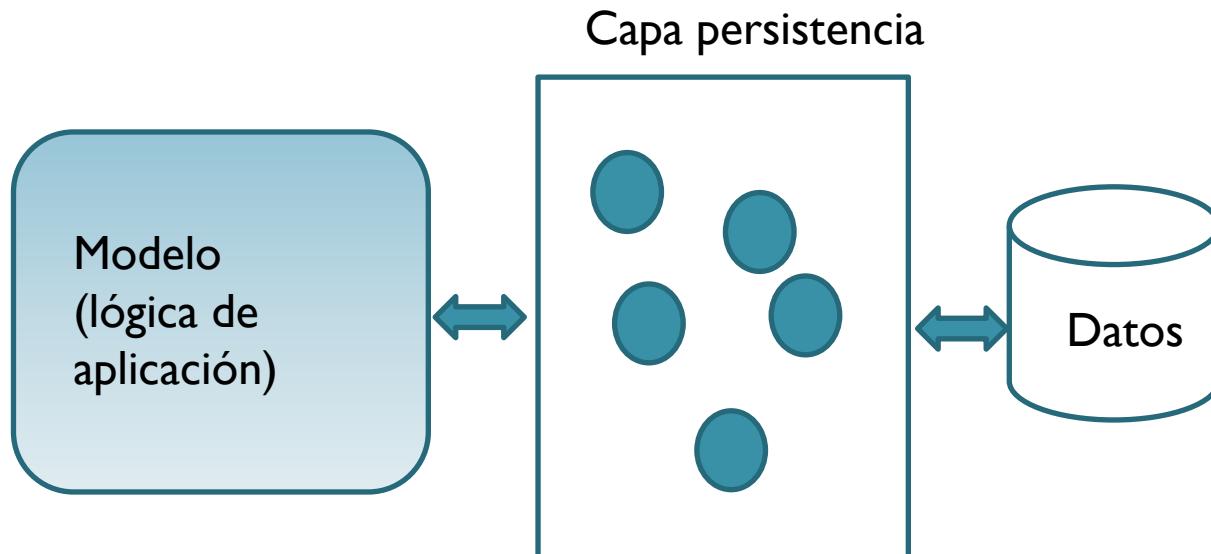




La capa de persistencia en una aplicación

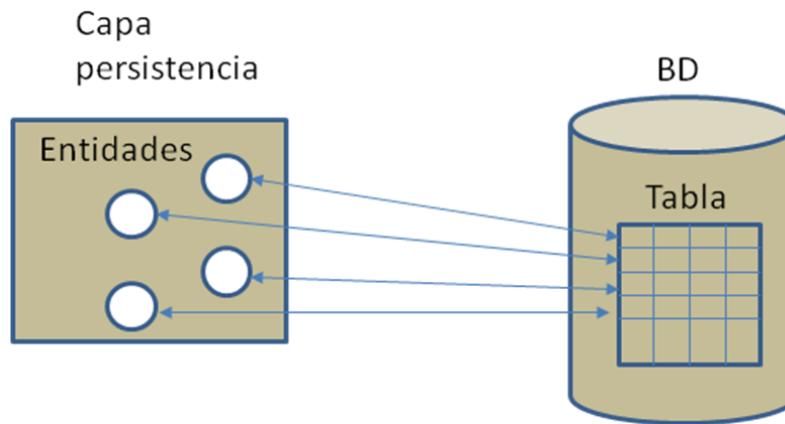
¿Qué es una capa de persistencia?

- Los datos son expuestos a la aplicación en forma de objetos (entidades).
- La lógica de negocio maneja objetos, no tablas



Entidades

- Objetos que representan una fila de una tabla de la base de datos.



- Se definen mediante clases tipo JavaBean
- Las operaciones sobre una capa de persistencia consisten en crear, modificar, eliminar y recuperar entidades.

Frameworks de persistencia

- Conjunto de utilidades que facilitan la creación y manipulación de una capa de persistencia.
- Entre los más populares:
 - Hibernate
 - Ibatis
 - JPA (no exactamente un framework)
 - Componentes de un framework de persistencia
 - Motor de persistencia (mapeo ORM)
 - Sistema de configuración
 - API (acceso a la capa de persistencia)

Java Persistence API

- Especificación Java EE para la configuración y gestión de una capa de persistencia
- Universal: compatible con diferentes motores(hibernate, ibatis,..)
- API estandarizado para acceso a entidades desde una aplicación

Ventajas de una capa de persistencia

- Simplifica el acceso a los datos desde la lógica de negocio de una aplicación
- Los datos son tratados como objetos desde aplicaciones orientadas a objetos
- Elimina la continua interacción en el código entre el mundo de los objetos y el mundo relacional

Java Persistence API

Java Persistence API

- Especificación Java EE para la configuración y gestión de una capa de persistencia
- Proporciona un API y sistema de configuración universal
- Permite la utilización de diferentes proveedores de persistencia (hibernate, toplink, etc.)

Componentes JPA

➤ **Archivo de configuración persistence.xml:**

- **Define unidades de persistencia**
- **Cada unidad de persistencia indica el motor utilizado, propiedades de conexión a BD y lista de entidades**

```
<persistence ..>
    <persistence-unit name="unidad_1" >
        <provider>...</provider>
        <properties>
            :
        </properties>
    </persistence-unit>
</persistence>
```

➤ **Conjunto de anotaciones para la configuración de entidades (@Entity, @Table, @Id,...)**

➤ **API JPA. Conjunto de clases e interfaces para acceso a la capa de persistencia**



Creación capa persistencia JPA

Proceso

- **Creación de entidades y configuración a través de anotaciones**
- **Configuración de la unidad de persistencia a través del archivo persistence.xml:**
 - **Proveedor de persistencia**
 - **Lista de entidades**
 - **Propiedades de conexión a base de datos**

Base de datos

➤ Creación de base de datos de ejemplo en MySQL:

▪ Base de datos agenda

▪ Tabla de contactos

MySQL Table Editor

Table Name: contactos Database: agenda Comment: InnoDB free: 10240 kB

Columns and Indices Table Options Advanced Options

Column Name	Datatype	NOT NULL	AUTO	Flags	Default Value	Comment
idContacto	INTEGER	✓	✓	<input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL	NULL	
nombre	VARCHAR(45)	✓		<input type="checkbox"/> BINARY		
email	VARCHAR(45)	✓		<input type="checkbox"/> BINARY		
telefono	INTEGER	✓		<input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL		

Indices Foreign Keys Column Details

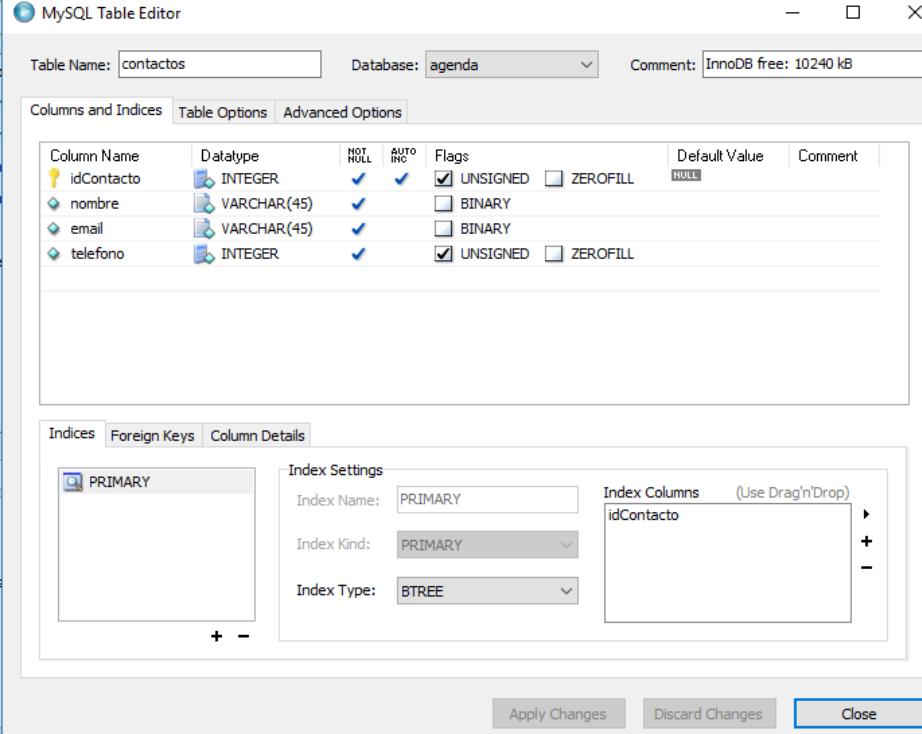
PRIMARY

Index Settings

Index Name: PRIMARY Index Kind: PRIMARY Index Type: BTREE

Index Columns (Use Drag'n'Drop)
idContacto

Apply Changes Discard Changes Close



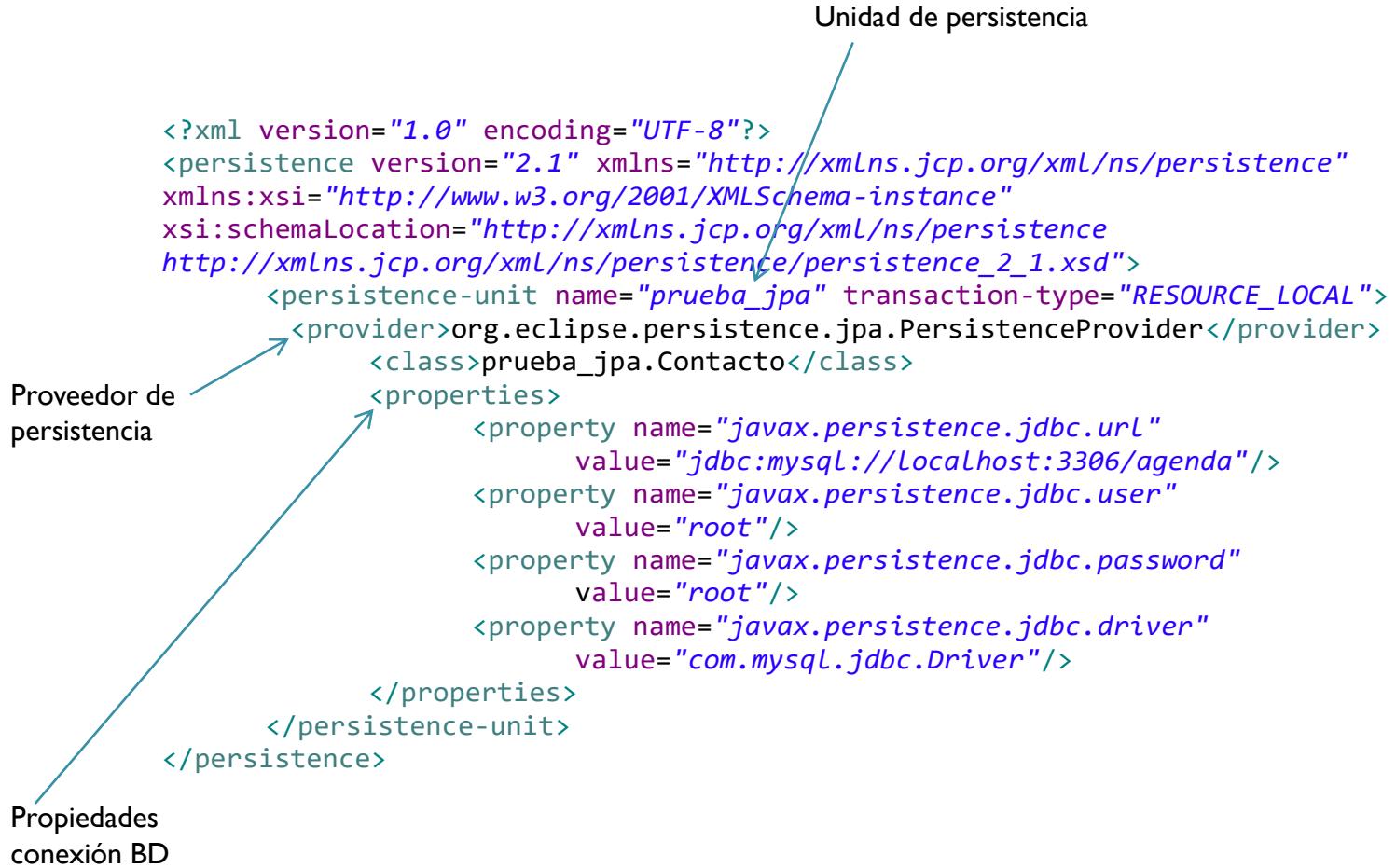
Creación de la entidad Contacto

```
@Entity  
@Table(name="contactos")  
public class Contacto implements Serializable {  
    private static final long serialVersionUID = 1L;  
    @Id  
    @GeneratedValue(strategy=GenerationType.IDENTITY)  
    private int idContacto;  
    private String email;  
    private String nombre;  
    private int telefono;  
    public Contacto() {  
    }  
    public int getIdContacto() {  
        return this.idContacto;  
    }  
    public void setIdContacto(int idContacto) {  
        this.idContacto = idContacto;  
    }  
    public String getEmail() {  
        return this.email;  
    }  
    public void setEmail(String email) {  
        this.email = email;  
    }  
    :  
}
```

Principales anotaciones

- **@Entity.** Indica que la clase es una entidad
- **@Table.** Mapea la entidad a una tabla de la base de datos
- **@Id.** Indica el atributo que contiene la primary key
- **@Column.** Asocia el atributo con una columna de la tabla. No es necesario si el nombre del atributo coincide con el de la columna.
- **@GeneratedValue.** Indica la estrategia de generación de claves en columnas primary key autogeneradas

persistence.xml



The diagram illustrates the structure of the persistence.xml file with the following annotations:

- Unidad de persistencia**: Points to the root element <persistence>.
- Proveedor de persistencia**: Points to the provider element within the persistence-unit.
- Propiedades conexión BD**: Points to the properties element within the persistence-unit.

```
<?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="prueba_jpa" transaction-type="RESOURCE_LOCAL">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <class>prueba_jpa.Contacto</class>
    <properties>
      <property name="javax.persistence.jdbc.url"
        value="jdbc:mysql://localhost:3306/agenda"/>
      <property name="javax.persistence.jdbc.user"
        value="root"/>
      <property name="javax.persistence.jdbc.password"
        value="root"/>
      <property name="javax.persistence.jdbc.driver"
        value="com.mysql.jdbc.Driver"/>
    </properties>
  </persistence-unit>
</persistence>
```

Utilización motor Hibernate

- Incluir librería **hibernate-core** en pom.xml
- Indicar motor hibernate en persistence.xml:

```
<provider>
    org.hibernate.jpa.HibernatePersistenceProvider
</provider>
```

- Ningún cambio en las instrucciones JPA de la lógica de negocio

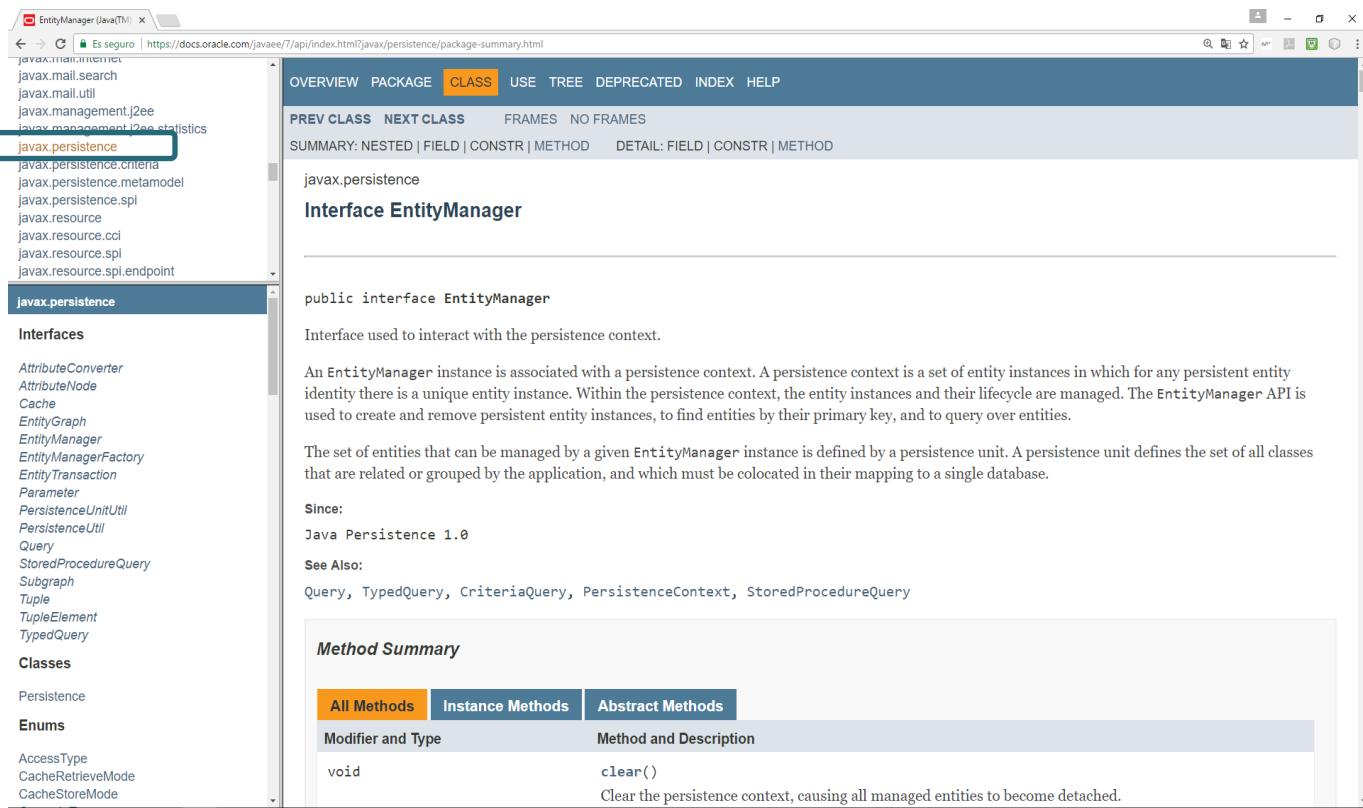
El API JPA

Javadoc

➤ Ayuda oficial del API:

<https://docs.oracle.com/javaee/7/api/index.html?javax/persistence/package-summary.html>

➤ Paquete principal: javax.persistence



The screenshot shows a JavaDoc interface for the `EntityManager` class. The URL in the browser is <https://docs.oracle.com/javaee/7/api/index.html?javax/persistence/package-summary.html>. The page title is "EntityManager (Java™) API". The navigation bar includes links for OVERVIEW, PACKAGE, CLASS (which is highlighted), USE, TREE, DEPRECATED, INDEX, and HELP. Below the navigation bar are links for PREV CLASS, NEXT CLASS, FRAMES, and NO FRAMES. The SUMMARY section includes links for NESTED, FIELD, CONSTR, and METHOD. The DETAIL section includes links for FIELD, CONSTR, and METHOD.

javax.persistence

Interface EntityManager

public interface EntityManager

Interface used to interact with the persistence context.

An `EntityManager` instance is associated with a persistence context. A persistence context is a set of entity instances in which for any persistent entity identity there is a unique entity instance. Within the persistence context, the entity instances and their lifecycle are managed. The `EntityManager` API is used to create and remove persistent entity instances, to find entities by their primary key, and to query over entities.

The set of entities that can be managed by a given `EntityManager` instance is defined by a persistence unit. A persistence unit defines the set of all classes that are related or grouped by the application, and which must be colocated in their mapping to a single database.

Since:
Java Persistence 1.0

See Also:
`Query`, `TypedQuery`, `CriteriaQuery`, `PersistenceContext`, `StoredQuery`

Method Summary

All Methods **Instance Methods** **Abstract Methods**

Modifier and Type	Method and Description
void	<code>clear()</code> Clear the persistence context, causing all managed entities to become detached.

Objeto EntityManager

- **Implementa la interfaz javax.persistence.EntityManager**
- **Asociado a una unidad de persistencia**
- **Proporciona métodos para realizar operaciones CRUD sobre la capa de persistencia**
- **Se obtiene a través de un EntityManagerFactory asociado a una unidad de persistencia**

Obtención de un EntityManager

➤ Proceso:

- **Creación de un objeto EntityManagerFactory a partir de clase Persistence.**
- **Creación EntityManager desde EntityManagerFactory**

EntityManagerFactory

```
factory=Persistence.createEntityManagerFactory("unidad_persistencia");
EntityManager em=factory.createEntityManager();
```

Métodos básicos de EntityManager

➤ Realización de operaciones CRUD:

- **void persist(Object entidad).** Persiste una entidad, añadiendo sus datos en una nueva fila de la BD.
- **void merge(Object entidad).** Actualiza la BD con los datos de la entidad, que debe formar parte de la unidad de persistencia
- **T find(Class<T> clase_entidad, Object key).** Recupera una entidad a partir de su clave primaria
- **void remove (Object entidad).** Elimina la entidad

➤ **void refresh(Object entidad).** Actualiza las propiedades de la entidad con los valores de la BD

Transacciones

- **Las operaciones de acción se deben englobar en una transacción**
- **Gestionadas mediante EntityTransaction , que se obtiene desde EntityManager**

```
EntityTransaction tx=em.getTransaction();
```

- **Métodos para gestionar la Transacción:**
 - **begin(). Inicia transacción**
 - **rollback(). Rechaza transacción**
 - **commit(). Confirma transacción**

Consultas JPA

Fundamentos

- Permiten recuperar, actualizar y eliminar entidades en base a diferentes criterios
- Se definen mediante un lenguaje especial llamado JPQL, similar a SQL
- Las consultas son gestionadas a través de un objeto de la interfaz Query o TypedQuery

El lenguaje JPQL

- Adaptación de **SQL** para tratar con entidades
- Formato Select:
 - **Select alias From Entidad alias Where condicion**
- La cláusula **Where** emplea los mismos operadores que **SQL**
- Ejemplos:

Select c From Contacto c

Select e From Empleado e Where e.dni='5555K'

Objeto Query

- **Implementación de la interfaz javax.persistence.Query que permite lanzar consultas JPQL a la capa de persistencia**
- **Se obtiene a partir del método createQuery() de EntityManager:**

```
String jpql="Select c From Contacto c";
Query qr=em.createQuery(jpql);
```

Métodos de Query

- **List getResultList(). Devuelve una colección List con las entidades recuperadas por la consulta:**

```
String jpql="Select c From Contacto c";
Query qr=em.createQuery(jpql);
//casting al tipo de colección específica
List<Contacto> contacts=(List<Contacto>)qr.getResultList();
```

- **Object getSingleResult(). Devuelve la única entidad resultante de la consulta. Si la consulta devolviera más de un resultado, se producirá una excepción**
- **void executeUpdate(). Ejecuta la consulta cuando se trata de una instrucción de acción (Delete o Update)**

TypedQuery

- Subinterfaz de Query que permite especificar el tipo de entidad de la respuesta:
 - **List<T> getResultList(). Devuelve una colección del tipo de las entidades:**

```
String jpql="Select c From Contacto c";
TypedQuery<Contacto> qr=em.createQuery(jpql,Contacto.class);
//No hay que hacer casting
List<Contacto> contacts=qr.getResultList();
```

- **T getSingleResult(). Devuelve la única entidad resultante de la consulta en el tipo de la misma**

- Disponible desde JPA 2.



Consultas parametrizadas

Definición

- **Las consultas JPQL pueden incluir parámetros**
- **Se pueden definir con un nombre:**
 - Select e From Empleado e where e.dni=:dni
- **O con una posición:**
 - Select e From Empleado e where e.dni=?1

Asignación de valores a parámetros

- Los valores se establecen antes de la ejecución de la consulta
- Se emplean los siguientes métodos de Query:
 - **setParameter(String nombre, Object value).** Establece el valor al parámetro cuyo nombre se indica
 - **setParameter (int pos, Object value).** Establece el valor al parámetro cuya posición se indica.

```
String jpql="Select e From Empleado e where e.dni=:dni";
Query qr=em.createQuery(jpql);
qr.setParameter("dni","334355R");
List<Empleado> emps=(List<Empleado>)qr.getResultList();
```



Consultas nominadas

Definición

- Se crean a partir de instrucciones JPQL definidas en la propia entidad
- A la instrucción se le asigna un nombre
- Permite que sean reutilizadas desde diferentes partes del código
- Pueden incluir parámetros

Creación

➤ Se definen mediante la anotación **@NamedQuery**:

```
@Entity  
@NamedQuery(name = "Coche.findAll", query = "SELECT c FROM Coche c")  
public class Coche{  
:  
}
```

➤ Si se quieren definir más de una instrucción se debe utilizar también **@NamedQueries**:

```
@NamedQueries({  
    @NamedQuery(name = "Coche.findAll", query = "SELECT c FROM Coche c"),  
    @NamedQuery(name = "Coche.deleteByColor", query = "Delete From Coche c  
        where c.color=?1")  
})  
public class Coche{  
:  
}
```

Utilización

- Para crear un objeto **Query** a partir de una **NamedQuery** utilizamos el método **createNamedQuery()** de **EntityManager**:

```
Query qr=em.createNamedQuery("Coche.findAll");
```

- A partir de ahí, el tratamiento es el de una consulta normal



Consultas de acción

Instrucciones JPQL de acción

- **Update. Para la actualización de un conjunto de entidades:**

Update Empleado e Set e.salario=e.salario*1.05

- **Delete. Eliminación de un conjunto de entidades:**

Delete From Curso c where c.nombre like 'P%'

- **Pueden incluir parámetros y definirse también como NamedQueries.**

Ejecución

- Se ejecutan a través del método **executeUpdate()** de **Query**

```
String jpql=Update Empleado e Set e.salario=e.salario*1.05;  
Query qr=em.createQuery(jpql);  
qr.executeUpdate()
```

- El método **executeUpdate()** devuelve el número de entidades afectadas

Relación entre entidades

Concepto

- Si las tablas tienen campos comunes, las entidades se pueden relacionar
- Al relacionar entidades, un objeto de una entidad contendrá el objeto u objetos de la entidad relacionada
- Las relaciones simplifican el acceso a la capa de persistencia, pues al recuperar un objeto recuperamos con él el objeto u objetos relacionados

Tipos de relaciones

- **Uno a uno. Un objeto entidad tiene un objeto de otra entidad asociado. Son poco frecuentes**
- **Uno a muchos y muchos a uno. Son las más frecuentes,. En el lado uno, una entidad dispone de muchos objetos relacionados del lado muchos.**
- **Muchos a muchos. Cada entidad de ambos lados tiene muchos objetos asociados en el otro lado. Requieren una tabla de unión**

Definición de entidades

- Las entidades deberán incluir atributos para recoger el objeto u objetos de la entidad con la que se relacionan:

```
public class Categoria
{
    private int codigoCategoria;
    :
    private List<Producto>
    productos;
    :
}

public class Producto
{
    private int idProducto;
    private String nombre;
    :
    private Categoria categoria;
    :
}
```

```
graph LR
    C["public class Categoria\n{\n    private int codigoCategoria;\n    :\n    private List<Producto>\n    productos;\n    :\n}\n"] --> P["public class Producto\n{\n    private int idProducto;\n    private String nombre;\n    :\n    private Categoria categoria;\n    :\n}\n"]
    C --> C2["public class Categoria\n{\n    private int codigoCategoria;\n    :\n    private List<Producto>\n    productos;\n    :\n}\n"]
    P --> C2
```

Configuración de relaciones

- **Las relaciones se configuran a través de las siguientes anotaciones:**
 - **@OneToOne. Relación uno a uno**
 - **@OneToMany. Relación uno a muchos**
 - **@ManyToOne. Relación muchos a uno**
 - **@ManyToMany. Relación muchos a muchos**
- **Se colocan delante del atributo que contiene el objeto/objetos de la entidad relacionada**

Configuración de relaciones II

- En las relaciones uno-muchos y muchos-uno, la entidad del lado muchos es la propietaria de la relación.
- El atributo del lado muchos incluye la anotación `@JoinColumn` con la información de la relación
- En las relaciones muchos a muchos, cualquier entidad puede incluir la información de relación con `@JoinTable`

Secciones

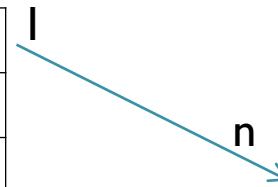
idSeccion
seccion
responsable

Productos

idProducto
nombre
idSeccion
precio
descripcion

1

n



Cuentas

numeroCuenta
saldo
tipoCuenta

Titulares

idCuenta
idCliente

Clientes

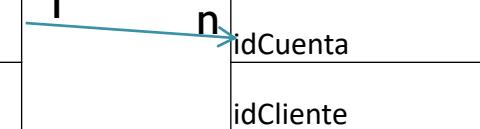
dni
nombre
direccion
telefono

|

n

|

n



Joins

Definición

- Un join permite definir uniones en una consulta JPQL entre entidades relacionadas
- Gracias a los join, podemos operar sobre una entidad en base a condiciones que afectan a la entidad relacionada.
- Dos tipos:
 - Join implícitos.
 - Join explícitos.

Join implícito

- **No necesita la utilización de la palabra join**
- **Se utiliza en relaciones muchos a uno cuando se opera sobre la entidad del lado muchos y la condición afecta a la entidad del lado uno**
- **Ejemplos:**
 - Select p From Producto p where p.seccion.responsable='María Salgado'
 - Select e From Empleado e where e.departamento.nombre='informatica'

Join explícito

- Se emplea la cláusula **join** dentro de **from**.
- Se utiliza cuando la asociación se da a través de un conjunto, es decir, en relaciones muchos a muchos y uno a muchos en el lado uno.
- Ejemplos:
 - Select distinct(s) From Seccion s **join** s.productos p where p.nombre like “%cable%”
 - Select c From Cliente c **join** c.cuentas b where b.saldo>1000

Claves primarias compuestas

Fundamentos

- **Tablas cuya clave primaria es la combinación de dos o mas columnas**
- **A nivel de capa de persistencia, se define una clase tipo JavaBean que encapsula la clave primaria**
- **La clave primaria de la entidad se define como un objeto de dicha clase**

Clase Primary Key

- JavaBean que encapsula los datos de la clave primaria
- Debe sobrescribir `equals()` y `hashCode()`
- Anotada con `@Embeddable`

Table Name:		sucursales	Database:	
Columns and Indices		Table Options	Advanced Options	
Column Name	Datatype	NOT NULL	AUTO INC	
nombre	VARCHAR(45)	✓		
calle	VARCHAR(45)	✓		
presupuesto	DOUBLE	✓		
inauguracion	INTEGER	✓		



```
@Embeddable  
public class SucursalPK {  
    private String nombre;  
    private String calle;  
    //setter y getter  
  
    public boolean equals(Object other) {  
        :  
    }  
    public int hashCode() {  
        :  
    }  
}
```

Definición de Primary Key en entidad

- La clave primaria de la entidad se define como un atributo de la clase Primary Key.
- El atributo se registra con **@EmbeddedId**

```
@Entity  
public class Sucursal{  
    @EmbeddedId  
    private SucursalPK id;  
    private int inauguracion;  
    private double presupuesto;  
    //definición de setter y getter  
  
}
```

Clase Primary Key II

- JavaBean que encapsula los datos de la clave primaria
- Debe sobrescribir `equals()` y `hashCode()`

Table Name: Database:

Columns and Indices Table Options Advanced Options

Column Name	Datatype	NOT NULL	AUTO INC
nombre	VARCHAR(45)	✓	
calle	VARCHAR(45)	✓	
presupuesto	DOUBLE	✓	
inauguracion	INTEGER	✓	



```
public class SucursalPK {  
    private String nombre;  
    private String calle;  
    //setter y getter  
  
    public boolean equals(Object other) {  
        :  
    }  
    public int hashCode() {  
        :  
    }  
}
```

Definición de Primary Key en entidad II

- Cada uno de los campos de la clave primaria se define en la propia entidad.
- En la entidad se debe indicar la clase que encapsula ambos campos mediante la anotación `@IdClass`

```
@IdClass(SucursalPK.class)
@Entity
public class Sucursal{
    @Id
    private String nombre;
    @Id
    private String calle;

    private int inauguracion;
    private double presupuesto;
    //definición de setter y getter

}
```