# JPA

## CREAR PROYECTO

**1 -** Creamos un nuevo proyecto **Maven**, con el arquetipo ***'quick-start'.***

**2 -** Cambiamos la versión de java a ***'1.8'*** en el ***'build path'*** del proyecto, y en el ***<properties>*** del ***'pom.xml'.***

**3 -** Añadimos las dependencias necesarias al ***'pom.xml'.***

**4 -** Convertimos el proyecto a ***'Faceted form'*** y añadimos ***'JPA 2.1'***, nos aseguramos que java esté seleccionada en la versión ***'1.8'***.

**5 -** En la pestaña ***'Runtimes'*** seleccionamos ***'jre'***. Seguidamente damos a ***'Further...'*** seleccionamos la versión ***'JPA 2.1'*** y la implementación en ***'Disable Library'.***

**6 -** Ahora vamos a añadir algunas ***'Properties'*** al ***'persistence.xml'.***

## AÑADIR NUEVA CONEXIÓN

**1 -** Añadimos una nueva ***'Database Connection'*** en el ***'Data Source Explorer'***, seleccionamos ***'MySql'*** y ponemos un nombre a la conexión.

**2 -** Completamos los datos necesarios y añadimos el ***'MySqlConnector.jar'***.

**3 -** Ahora añadimos la conexión al ***'persistence.xml'***, en el apartado conexión, especificamos el tipo local y la seleccionamos en ***'Populate from connection'***.

## CLASE APP

**1 -** (Opcional): ***'java.util.logging.Logger.getLogger("org.hibernate").setLevel(Level.OFF);'***

**2 -** Creamos un ***'EntityManagerFactory'*** a partir de la unidad de ***'Persistence'*** con el nombre del proyecto.

**3 -** Generamos un ***'EntityManager'*** al ***'EntityManagerFactory'.***

**4 -** Si vamos a ***insertar, actualizar o eliminar*** datos en la BD necesitamos iniciar una transacción con el ***'em'*** con los métodos ***'.getTransaction()'*** , ***'.begin()'*** y finalizarla persistiendo los objetos con ***'.persist(objeto)'*** y ***.commit()'***.

**5 -** Finalmente siempre hay que liberar los recursos de ***'em'*** y ***'emf'***.

## GENERACIÓN AUTOMÁTICA DE ENTIDADES

**1 -** Creamos el proyecto ***JPA*** igual que siempre a excepción de la propiedad ***'hibernate.hbm2ddl'*** que no la configuraremos. Nos aseguramos de tener habilitada la conexión con la base de datos en cuestión.

**2 -** Botón derecho en el proyecto ***New -> JPA Entities from Tables***, seleccionamos la conexión, y las tablas que queramos.

**3 -** Añadimos alguna asociación más si la necesitamos.

**4 -** Cambiamos el ***Key generator*** a ***'Identity'***, en la siguiente ventana cambiamos los tipos de datos de los campos si lo deseamos y le damos a ***Finish***.

## ANOTACIONES VALORES Y ENTIDADES

| | | | |
|---|---|---|---|
| ***@Entity*** | Convertir una clase Java en una entidad | ***@Column (name="",nullable,lenght)*** | Modificadores campos |
| ***@Table(name= "NEWNAME")*** | Controlar el nombre de la tabla | | |
| ***@Id*** | PK | ***@Embedded = @Embeddable / @EmbeddedId*** | Objetos/ PK compuesta = Clase nueva, Serializable y @Embedded @MapsId("algunaPK") |
| ***@GeneratedValue (strategy= GT.Identity)*** | Generar Id autoNuméricos | ***@Temporal (TemporalType.Date)*** | Tipos temporales |

## MANY-TO-ONE

**Ejemplo ->** Tenemos una clase ***'Persona'*** y una clase ***'Telefono'*** que tiene un atributo ***'Persona'.***

**1 -** Bastaría con indicar la asociación en la clase ***'Telefono'*** , si queremos podemos indicar el nombre de la ***'FK'.***

***@ManyToOne***
***@JoinColumn(name,fk = @fk(name))***

## ONE-TO-MANY Unidireccional

**Ejemplo ->** Tenemos una clase ***'Persona'*** con una lista de ***'Tlf'*** y una clase ***'Tlf'.***

**1 -** Bastaría con indicar la asociación en la clase ***'Persona'.*** Crea una tabla intermedia con los Id.

***@OneToMany(cascade=CT.All,orphanRemoval=v)***

## ONE-TO-MANY Bidireccional

**Ejemplo ->** Tenemos una clase ***'Persona'*** con una lista de ***'Tlf'*** y una clase ***'Tlf'*** que tiene un atributo ***'Persona'***

**1 -** Necesita una asociación 'manyToOne' en el lado hijo. No crea tabla intermedia, sino un campo FK.

***@OneToMany(mappedBy,cascade=All, orphanRemoval=true)***

## CONSULTAS

**1 -** Utilizamos la interfaz ***'javax.persistence.Query'*** que se obtienen directamente desde el ***'EntityManager'***. Creamos la ***'Query'*** con el método ***'createQuery()'.***

**2 -** Si la consulta devuelve un solo resultado usamos ***'getSingleResult()'*** , si por contra devuelve mas de un resultado utilizamos ***'getResultList()'***.

**# -** *Siempre* se añaden alias, se puede navegar con los puntos, como si fueran clases, incluso entre distintas tablas.

**# -** Para asignar parámetros dinámicos a las consultas tenemos dos opciones, ***':nombreParametro'*** y ***'?1'***. Ambos se manejan con el método ***'setParameter("nom/num","valor")'.***

**# -** ***HQL*** no permite el uso de ***LIMIT***, por lo que hay que usar ***'setMaxResults(num)'*** después del ***'createQuery()'.***

**# -** En las consultas de actualización se usa ***'executeUpdate()'***, este método devuelve el número de filas afectadas. ***No se pueden usar Joins.***

**# -** Las consultas pueden retornar múltiples objetos y/o propiedades como un array de tipo ***'Object[]'***, una lista o una clase.

**# -** ***@NamedQueries({ })*** para crear mas de una ***@NamedQuery(name="",query="").***

**# -** Las ***@NamedQuery*** hay que llamarlas con el método ***'.createNamedQuery("nombre")'***, se puede usar el ***'.maxResults()'.***

**# -** ***EXPRESIONES ÚTILES:***

| | | | | | |
|---|---|---|---|---|---|
| ***avg()*** | ***trunc()*** | ***between*** | ***minute()*** | ***size() = F. agregado*** | ***concat(...,...)*** |
| ***sum()*** | ***round()*** | ***is not null*** | ***hour()*** | ***length()*** | ***substring(num,num)*** |
| ***min()*** | ***coalesce()*** | ***is not empty*** | ***year()*** | ***upper()*** | ***left(cadena,num)*** |
| ***max()*** | ***in*** | ***current_date()*** | ***month()*** | ***lower()*** | ***right(cadena,num)*** |
| ***count()*** | ***not in*** | ***current_time()*** | ***day()*** | ***concat_ws(...,...)*** | ***>=ALL(subConsulta)*** |