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)***

JDBC

ESTABLECER CONEXIÓN

1 - Importamos la libreria ***'mysql-connector'*** en ***'modules-dependencies'.***

2 - Cargamos el driver con el método ***'forName()'*** pasandole por parámetro ***'com.mysql.cj.jdbc.Driver'***.

3 - Creamos una nueva ***Connection*** con el método ***'getConnection()'*** y le pasámos por parámetro la ruta local ***'jdbc:mysql://localhost/nombreBD'*** , el user y la psw.

4 - ***IMPORTANTE:*** Siempre debemos ***liberar los recursos*** de la conexión.

SENTENCIAS SIMPLES

1 - Creamos un ***'Statement'*** a partir de la conexión con el método ***'createStatement()'.***

2 - Declaramos un ***'String'*** con la sentencia *SQL*.

3 - Almacenamos el resultado en un ***'ResultSet'*** (iterador), llamando al método ***'executeQuery(sql)'*** o ***'executeUpdate(sql)'*** del ***'Statement'*** creado anteriormente.

4 - Cada iteracción del resultado es un registro, para acceder a sus columnas usamos los métodos ***'.getInt()...'***, pasando el ***numColumna*** o el ***nomColumna*** de la tabla.

5 - Cuando hacemos uso de ***'executeUpdate(sql)'*** no nos devuelve un ***ResultSet*** sino un ***int*** con el numero de filas afectadas.

6 - ***IMPORTANTE:*** Siempre debemos ***liberar los recursos*** de la sentencia y resultado.

SENTENCIAS PREPARADAS

1 - Declaramos un ***'String'*** con la sentencia *SQL*, añadiendo ***'?'*** en los parámetros que introduciremos nosotros.

2 - Creamos una ***'PreparedStatement'*** a partir de la conexión con el método ***'prepareStatement(sql)'.***

3 - Introducimos los parámetros con los métodos ***'.setInt()...'***, pasando el ***numParametro*** y el ***valor***.

4 - Almacenamos los resultados de igual manera que en las sentencias simples.

5 - ***IMPORTANTE:*** Siempre debemos ***liberar los recursos*** de la sentencia y resultado.

CLASE RESULTSET

1 - Si no sabemos la estructura de la BD podemos obtener datos de ella con la clase ***ResultSet*** con los métodos ***'getTables()','getColumns()','getPrimaryKeys()'***

2 - ***EJEMPLO:*** para extraer datos de las tablas, creamos un '***DataBaseMetaData***' a partir de la conexión con el método ***'.getMetaData()'*** ,después creamos un String[] de los tipos que queremos obtener ***'TABLE', 'VIEW' ...***

3 - Después almacenamos el resultado en un ***ResultSet*** usando el método ***'.getTables(null,null,null, tipos)'*** para la BD.

4 - De cada registro, podremos obtener con el ***'.getString()'*** el ***'TABLE_CAT', 'TABLE_SCHEM', 'TABLE_NAME', 'TABLE_TYPE' ...***

5 - ***IMPORTANTE:*** Siempre debemos ***liberar los recursos*** del resultado.

EJECUCIÓN DE PROCEDIMIENTOS

1 - En primer lugar creamos una ***'Statement'*** como hemos visto anteriormente y ejecutamos la creación del procedimiento con el método ***'.execute(sql)'.***

2 - Creamos un ***String*** con el ***SQL*** introduciendo la sentencia entre corchetes ***'{call...}'.***

3 - Creamos un ***'CallableStatement'*** a la conexión usando el método ***'.prepareCall(sql)'.***

4 - Introducimos los parámetros de igual manera que con las ***PreparedStatement.***

5 - Usamos los métodos ***'.executeUpdate()'*** o ***'.executeQuery()'*** respectivamente de la ***'CallableStatement'***, sabemos que este último necesita un ***ResultSet.***

PARÁMETROS DE SALIDA EN PROCEDIMIENTOS

1 - En primer lugar debemos especificar el parametro ***OUT*** en el ***SQL*** de creación del procedimiento.

2 - Cuando registramos los argumentos del procedimiento, en el parámetro ***OUT*** debemos usar el método ***'.registerOutParameter(numP, Types.VARCHAR)'.***

3 - Una vez ejecutado el procedimiento de igual manera que antes, simplemente debemos hacer un sout ***'.getString(numPout)'*** para mostrar el resultado.

RESÚMEN SQL

DDL

```
AUTO_INCREMENT
ALTER tabla MODIFY columna
CHECK (campo >= ...)
PRIMARY KEY (...,...)
DEFAULT valor
FOREIGN KEY nom_fk (campo) REFERENCES tabla (campo) ON UPDATE CASCADE ON DELETE CASCADE
```

CONSULTAS

```
SELECT DISTINCT
BETWEEN v AND v
LIKE 'cadena%'
IN ('valor',...)
WHERE NOT
WHERE EXISTS ()
```

PROCEDIMIENTOS

```
DECLARE var INT;
SELECT INTO var

CASE
    WHEN condition1 THEN result1
    ELSE result
END;

SET var = valor;
CREATE OR REPLACE

IF condition THEN
        SELECT result;
    ELSE
        SELECT resultElse;
    END IF;
```