



Ingeniería del Software II

Tema 4: Implementación: Frameworks

4.3. Hibernate

A. Goñi, J. Iturrioz

Índice

- 1) Motivación: Por qué utilizar un Framework como Hibernate
- 2) Hibernate
 - 2.1) Introducción
 - 2.2) Arquitectura de Hibernate
 - 2.3) Clases persistentes (Ficheros de configuración: .hbm.xml)
 - 2.4) Asociaciones entre clases (Ficheros de configuración: .hbm.xml)
 - 2.5) Session Factory (Fichero de configuración: hibernate.cfg.xml)
 - 2.6) Session y Transaction
 - 2.7) Lenguajes de interrogación: HQL (y Criteria)
 - 2.8) Aspectos de interés: ciclo de vida, asociaciones 1:N, 1:1, N:M, null values, borrado, actualización, recuperación de datos,...

1) Motivación

- En las vistas se necesita tener disponibles datos (almacenados en el nivel de datos) a los que se accede por medio de la lógica de negocio (objetos bean)
 - Programación: con un lenguaje OO
 - Persistencia de datos: con BDOO o con BD relacionales

Las BD relacionales están más extendidas que las BDOO: buena opción para la persistencia



- Si se usan BD relacionales, entonces hay que “mapear” tuplas relacionales en objetos
- En ese caso, una herramienta de mapeo objeto-relacional como Hibernate es una buena elección

1) Motivación

SET AVAILABILITY

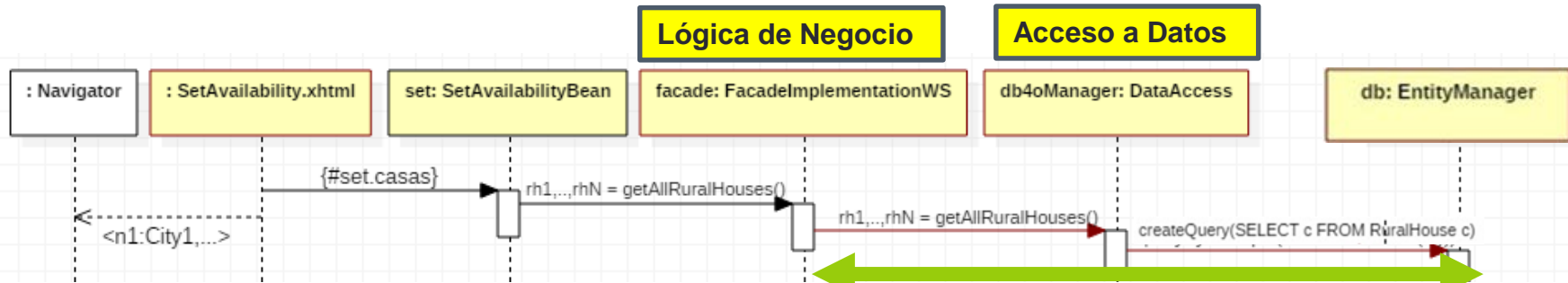
List of houses:

First day:

Price:

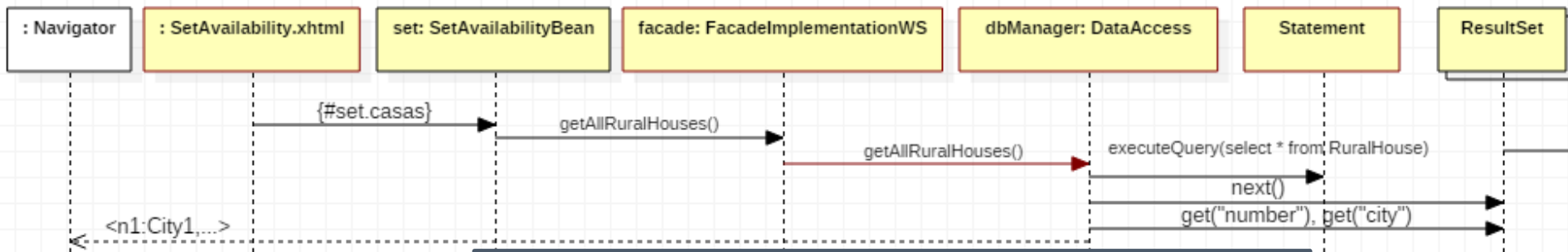
Last day:

¿Cómo se cargan estos objetos en el componente h:selectOneMenu?



Si DataAccess usa un SGBDOO como objectDB

¿Y si se usara un Sistema de Gestión de Bases de Datos Relacional (SGBDR)?



Se puede implementar usando Java JDBC

1) Motivación

SET AVAILABILITY

List of houses:

First day:

Price:

Last day:

HOUSE DESCRIPTION: [Gaztetxea](#)

Sin embargo, JSF permite hacer cosas más potentes...

Expresión escrita en EL (Expression Language) de JSF. Permite más posibilidades de acceso a propiedades.

```
HOUSE DESCRIPTION: <h:outputLabel value="#{setAvailabilityBean.casa.description}" id="description" />
```

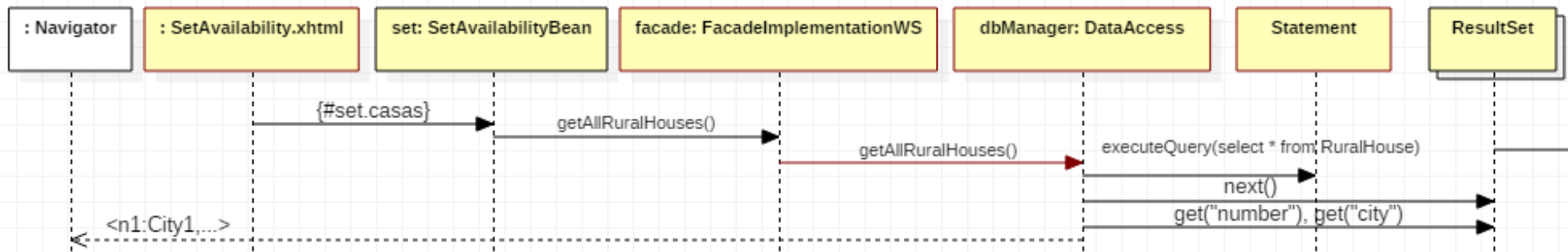
En este caso, invocar al método getDescription()...

```
public class RuralHouse :  
  
    public String getDescription() {  
        return description;  
    }
```

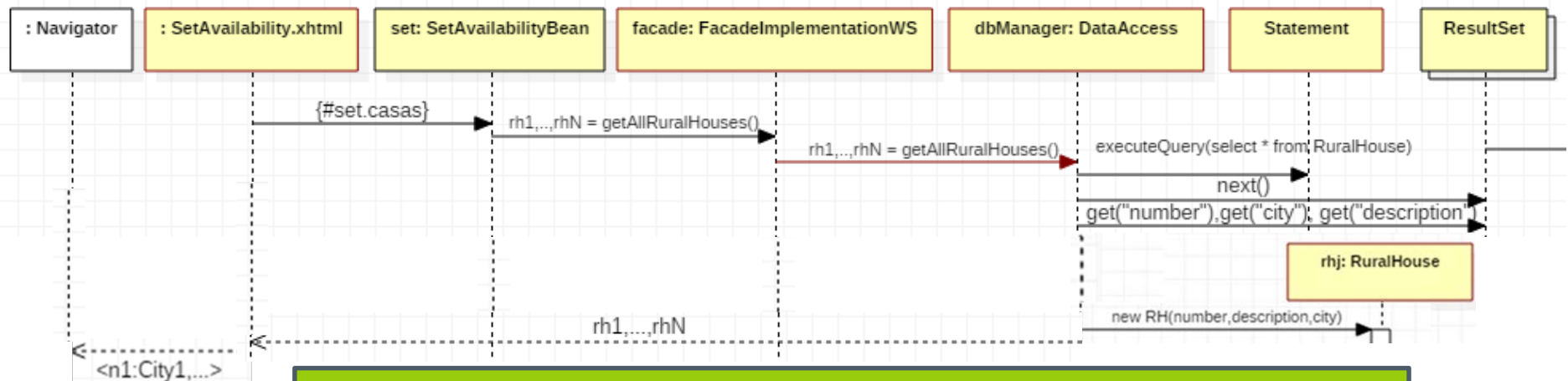
JSF (basado en Java, que es OO) permite trabajar con objetos. Las casas rurales de la lista desplegable son OBJETOS, a los cuales se les puede pedir ejecutar métodos.

1) Motivación

La solución con BD relacional anterior no sería suficiente:



Se necesitaría mapear las tuplas en objetos

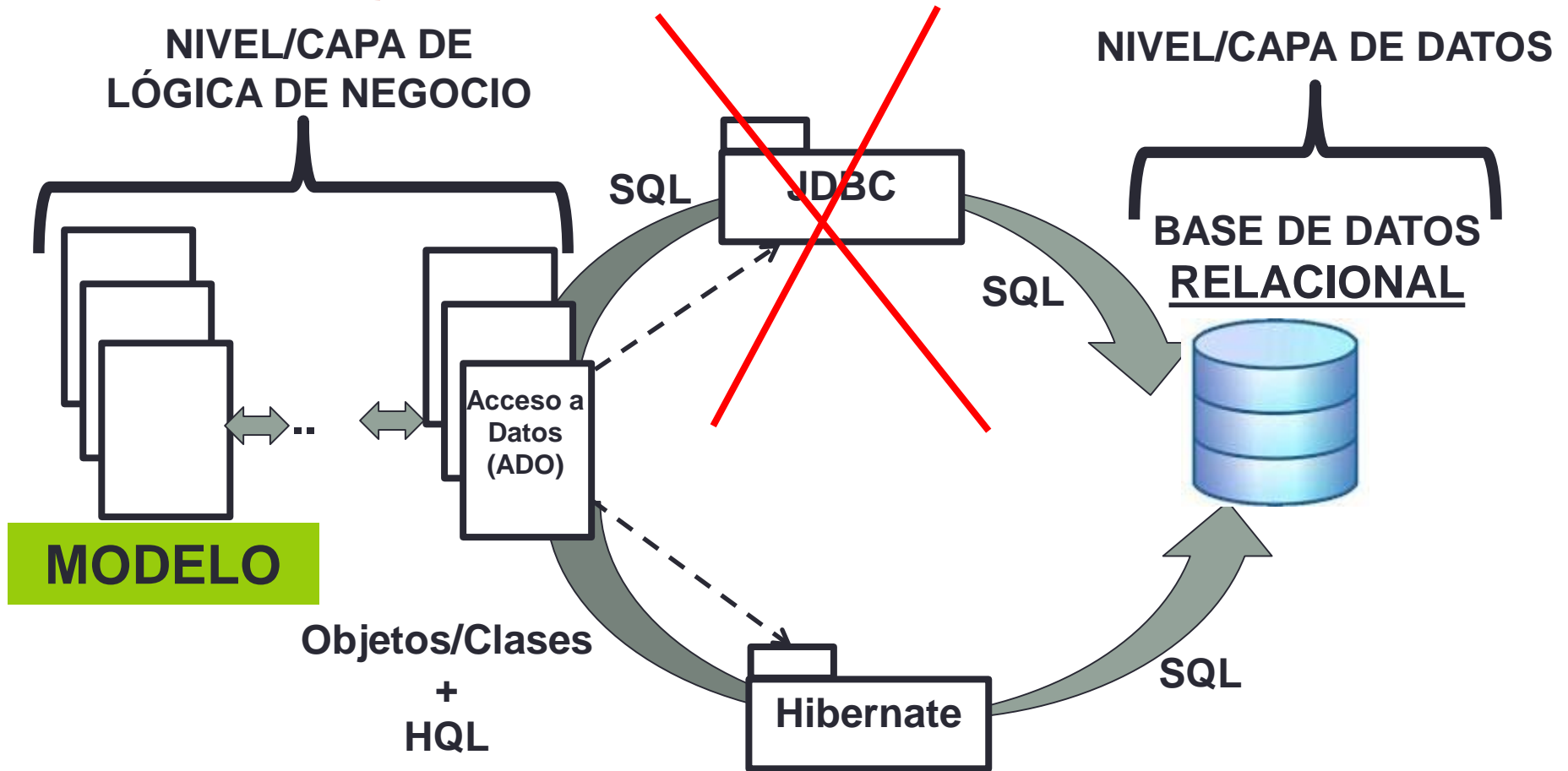


Y hacerlo a mano es muy costoso: mejor usar una herramienta de mapeo O/R como HIBERNATE

2.1) Introducción a Hibernate

- Es una herramienta de software libre para realizar mapeo objeto-relacional (ORM)
 - Para la plataforma Java
 - Disponible para .NET (NHibernate)
 - JBoss Inc. (RedHat) contrató a sus desarrolladores
- Inicialmente no se basó en estándares para ORM como EJB (Enterprise Java Beans) o JDO (Java Data Objects)
- Pero ahora sí es compatible con el estándar ORM llamado JPA (Java Persistence API)
 - Añadiendo la posibilidad de definir los mappings usando anotaciones JPA en vez de ficheros XML

2.1) Introducción a Hibernate



Hibernate permite gestionar la persistencia de objetos en BASES DE DATOS RELACIONALES sin utilizar JDBC ni sentencias SQL (SELECT para consultas e INSERT, DELETE y UPDATE para modificaciones) sino usando la API de Hibernate (save, saveOrUpdate, delete para modificaciones y load/get para consultas) y HQL (para consultas)

2.2) Arquitectura Hibernate

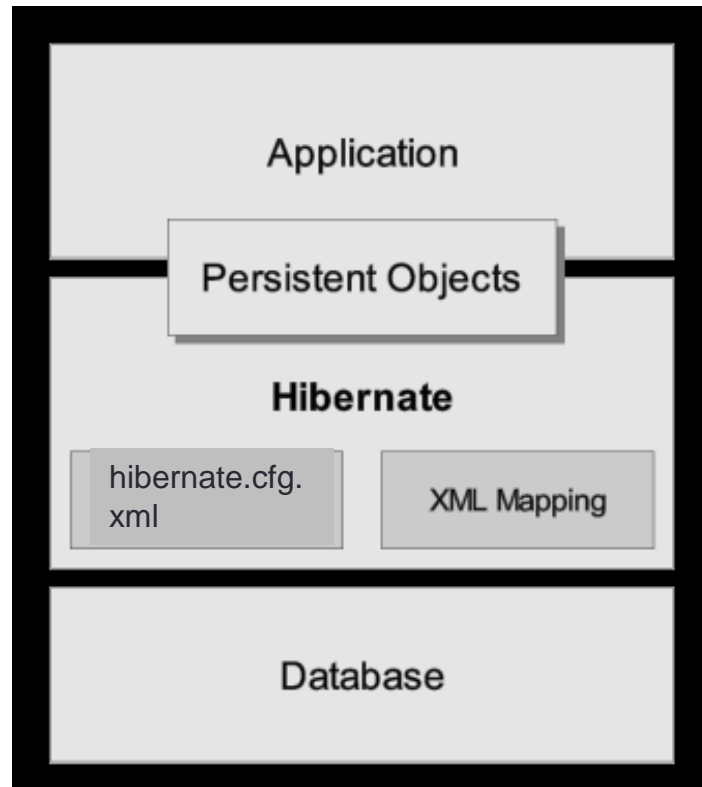


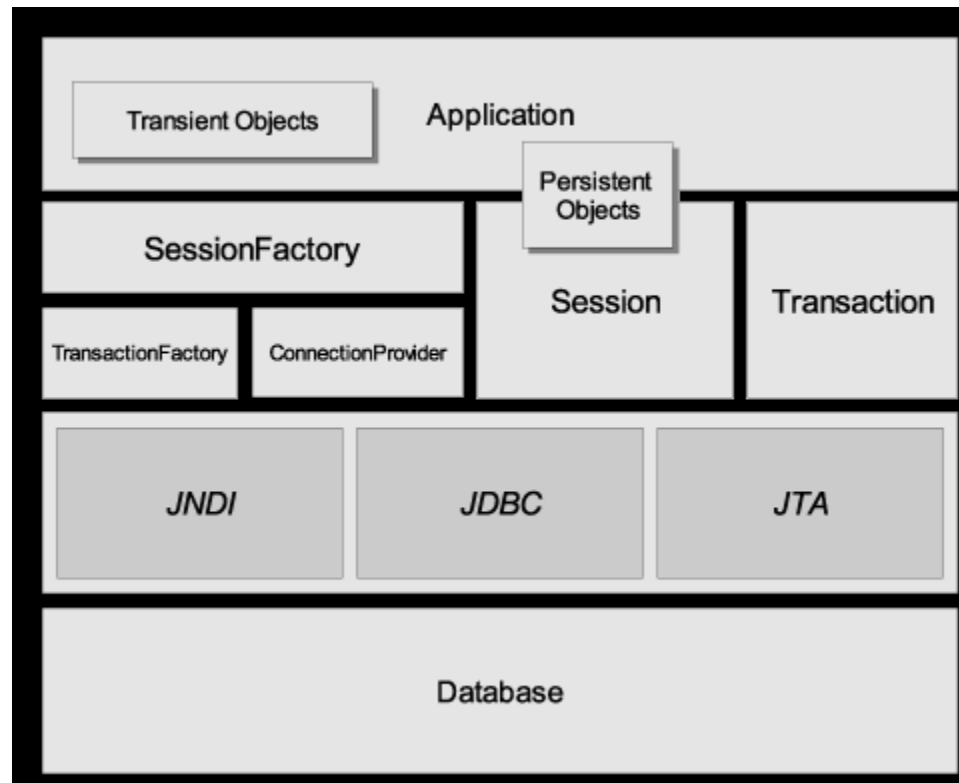
Imagen extraída de: <https://docs.jboss.org/hibernate/orm/3.5/reference/es-ES/html/architecture.html>

La aplicación trabaja con objetos persistentes. De la persistencia de dichos objetos en la BD se encarga Hibernate, utilizando para ello unos ficheros de configuración: propiedades y enlaces (mappings) a la BD

2.2) Arquitectura Hibernate (más detallada)

La aplicación trabaja con “Transient Objects” y con “Persistent Objects”

Y utiliza la funcionalidad ofrecida por “Session Factory”, “Session” y “Transaction”



2.3) Objetos/Clases persistentes...

... y ficheros de configuración .hbm.xml

- Son objetos de clases Java
 - Deben ser Java Beans
 - Se llaman POJO = Plain Old Java Objects
- A los que se les da persistencia en una BD relacional
- La funcionalidad que realiza la persistencia NO SE ENCUENTRA dentro de dichas clases.
 - Se encarga Hibernate utilizando los ficheros de configuración .hbm.xml

Ej: Clase persistente y su mapping

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- Generated 30-nov-2016 23:22:53 by Hibernate Tools 3.4.0.CR1 -->
<hibernate-mapping>
  <class name="modelo.dominio.EventoLogin" table="EVENTOLOGIN">
    <id name="id" type="java.lang.Long">
      <column name="ID" />
      <generator class="assigned" />
    </id>
    <property name="descripcion" type="java.lang.String">
      <column name="DESCRIPCION" />
    </property>
    <property name="fecha" type="java.util.Date">
      <column name="FECHA" />
    </property>
  </class>
</hibernate-mapping>
```

El fichero XML de mapeo o enlace (mapping)

La clase JAVA

```
public class EventoLogin {
  private Long id;
  private String descripcion;
  private Date fecha;
}
```

MySQL 5.7 Command Line Client

```
mysql>
mysql> describe eventologin;
```

Field	Type	Null	Key	Default	Extra
ID	bigint(20)	NO	PRI	NULL	
DESCRIPCION	varchar(255)	YES		NULL	
FECHA	datetime	YES		NULL	

La tabla de la BD relacional

2.4) Asociaciones entre clases ... y ficheros de configuración .hbm.xml

- Entre clases existen asociaciones
 - Uno a uno, Uno a muchos, Muchos a muchos
 - Que pueden ser unidireccionales o bidireccionales
- En UML se representan mediante líneas con cardinalidades y flechas
- En Java se implementan con atributos cuyos tipos son clases o colecciones (set, list,...)

Ej: Asociación 1:N

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- Generated 30-nov-2016 23:55:10 by Hibernate Tools 3.4.0.CR1 -->
<hibernate-mapping>
  <class name="modelo.dominio.Usuario" table="USUARIO">
    <id name="nombre" type="java.Lang.String">
      <column name="NOMBRE" />
      <generator class="assigned" />
    </id>
    <property name="password" type="java.Lang.String">
      <column name="PASSWORD" />
    </property>
    <property name="tipo" type="java.Lang.String">
      <column name="TIPO" />
    </property>
  </class>
</hibernate-mapping>
```

La clase Usuario con su tabla y su mapping

```
mysql> describe usuario;
```

Field	Type	Null	Key	Default	Extra
NOMBRE	varchar(255)	NO	PRI	NULL	
PASSWORD	varchar(255)	YES		NULL	
TIPO	varchar(255)	YES		NULL	

```
public class Usuario {
  String nombre;
  String password;
  String tipo;
```

Ej: Asociación 1:N

¿ Esta asociación es unidireccional o bidireccional?

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- Generated 30-nov-2016 23:55:10 by Hibernate Tools 3.4.0.CR1 -->
<hibernate-mapping>
  <class name="modelo.dominio.EventoLogin" table="EVENTOLOGIN">
    <id name="id" type="java.lang.Long">
      <column name="ID" />
      <generator class="increment" />
    </id>
    <property name="descripcion" type="java.lang.String">
      <column name="DESCRIPCION" />
    </property>
    <property name="fecha" type="java.util.Date">
      <column name="FECHA" />
    </property>
    <many-to-one name="usuario" class="modelo.dominio.Usuario" fetch="join">
      <column name="USUARIO" />
    </many-to-one>
    <property name="login" type="org.hibernate.type.NumericBooleanType">
      <column name="LOGIN" />
    </property>
  </class>
</hibernate-mapping>
```

1 Usuario –
Many EventoLogin

La asociación 1:N se implementa añadiendo a EventoLogin su usuario

```
mysql> describe eventologin;
+----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra |
+----+-----+-----+-----+-----+-----+
| ID         | bigint(20)    | NO   | PRI | NULL    |       |
| DESCRIPCION | varchar(255)  | YES  |     | NULL    |       |
| FECHA      | datetime      | YES  |     | NULL    |       |
| USUARIO    | varchar(255)  | YES  | MUL | NULL    |       |
| LOGIN      | int(11)       | YES  |     | NULL    |       |
+----+-----+-----+-----+-----+-----+
```

```
public class EventoLogin {
    private Long id;
    private String descripcion;
    private Date fecha;
    private Usuario usuario;
    private boolean login;
```

Creación automática de los ficheros de mapping .hbm.xml usando Eclipse

- 2 posibilidades, dependiendo de qué tenemos inicialmente: los POJOs o la BD
- A partir de una clase POJO se genera automáticamente su fichero de mapping .hbm.xml. Y después, en tiempo de ejecución, se crea la tabla en la BD (configurando bien la propiedad hibernate.cfg.xml)
 - Opción “update” en propiedad *“hibernate.hbm2ddl.auto”*
- A partir de las tablas de una BD, se pueden generar automáticamente los ficheros .hbm.xml y las clases POJO

2.5) Session Factory...

... y su configuración en hibernate.cfg.xml

- Es una factoría para poder obtener sesiones con BDs concretas.
 - Si se trabaja con diferentes BDs hay que definir diferentes “Session Factory”
- Sirve para configurar conexiones JDBC con la BD
- Se configura en el documento XML siguiente:
hibernate.cfg.xml
- Y se necesita crear una instancia para utilizarla desde un programa.

Session Factory: se configura en el fichero hibernate.cfg.xml

Fichero hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory name="">
  <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
  <property name="hibernate.connection.password">admin</property>
  <property name="hibernate.connection.url">jdbc:mysql://158.227.115.135:3306/eventos</property>
  <property name="hibernate.connection.username">root</property>
  <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
  <property name="hibernate.connection.pool_size">1</property>
  <property name="hibernate.hbm2ddl.auto">update</property>
  <property name="hibernate.current_session_context_class">thread</property>
  <property name="hibernate.connection.autocommit">>false</property>
  <mapping resource="org/hibernate/tutorial/domain/Event.hbm.xml"/>
</session-factory>
</hibernate-configuration>
```

Configuración del Driver y SGBD, y datos necesarios para conexiones JDBC

Declarar los ficheros de mapping entre las clases POJO y las tablas de BD

Opción habilitada para modificar el esquema de la BD ya existente (o crearlo si no existe)

Activa la gestión de contextos automática de Hibernate (ver punto 2.6)

Session Factory: obtener la instancia desde un programa

- Se necesita crear una instancia (Singleton) de Session Factory en tiempo de ejecución desde un programa y dejarla accesible cómodamente a toda la aplicación
 - `HibernateUtil.getSessionFactory()`

```
public class HibernateUtil {
private static final SessionFactory sessionFactory;
static {
try {
// Create the SessionFactory from hibernate.cfg.xml
    sessionFactory = new Configuration().configure(new File("hibernate.cfg.xml")).buildSessionFactory();
} catch (Throwable ex) {
// Make sure you log the exception, as it might be swallowed
throw new ExceptionInInitializerError(ex);
}
}
public static SessionFactory getSessionFactory() {
return sessionFactory;
}
}
```

El nombre del fichero de configuración podría ser diferente a `hibernate.cfg.xml`. si se modifica ahí.

Session Factory: obtener la instancia desde un programa

En Hibernate 4 se obtiene el “SessionFactory” de otra manera, ya que `buildSessionFactory()` ha sido desaprobado

```
private static SessionFactory sessionFactory;
private static ServiceRegistry serviceRegistry;

public static SessionFactory createSessionFactory() {
    Configuration configuration = new Configuration();
    configuration.configure();
    serviceRegistry = new ServiceRegistryBuilder().applySettings(
        configuration.getProperties()).build();
    sessionFactory = configuration.buildSessionFactory(serviceRegistry);
    return sessionFactory;
}
```

Pero, a partir de Hibernate 4.3, `ServiceRegistryBuilder` también está desaprobado

```
Configuration configuration = new Configuration().configure();
StandardServiceRegistryBuilder builder = new StandardServiceRegistryBuilder().
    applySettings(configuration.getProperties());
SessionFactory factory = configuration.buildSessionFactory(builder.build());
```

CONCLUSIÓN: consultar en internet según la versión de Hibernate

2.6) Session / Transaction

- Las transacciones con la BD se realizan esta manera:

```
Session session =  
    HibernateUtil.getSessionFactory().getCurrentSession();  
session.beginTransaction();  
// OPERACIONES con la BD (que componen la transacción)  
session.getTransaction().commit();
```

Una sesión `org.hibernate.Session` permite una unidad de trabajo con la BD. La sesión comienza con la llamada a `getCurrentSession()` hecha al “Session Factory”. Hibernate asocia dicha sesión al thread actual si se ha definido así en `hibernate.cfg.xml`: `<property name="hibernate.current_session_context_class">thread</property>`

Al terminar la transacción (con `commit` o con `rollback`), Hibernate desasocia automáticamente la sesión y la cierra. Otra llamada a `getCurrentSession()` obtendrá una nueva sesión.

También se puede usar `HibernateUtil.getSessionFactory().openSession()` para obtener una sesión, pero después hay que cerrarla: `session.close()`

Operar con la BD objeto a objeto

- Se pueden recuperar, salvar y borrar objetos pidiéndoselo a session
 - `session.get(clase, id)`
 - `session.load(clase, id)`
 - `session.save(objeto)`
 - `session.saveOrUpdate(objeto)`
 - `session.delete(objeto)`
- En realidad, ni siquiera se necesita definir una transacción para ello:
 - `HibernateUtil.getSessionFactory().getCurrentSession().save(objX)`

Nota: `get` devuelve null si el objeto con el id a recuperar no existe en la BD y `load` devuelve una excepción en ese caso; `get` accede siempre a BD y `load` no necesariamente

Operar con la BD objeto a objeto: Ejemplos

```
private static Copia createAndStorePrestamo(Socio soc, Copia copia) {  
    session = HibernateUtil.getSessionFactory().getCurrentSession();  
    session.beginTransaction();  
    copia.setSocio(soc);  
    copia.setEstado("ocupada");  
    copia.setFechaPrestamo(new Date());  
    session.update(copia); ← Operación con la BD  
    session.getTransaction().commit();  
    log.info("Modificada: "+copia);  
    return copia;  
}
```

Transacción

Operación con la BD

Las transacciones siguen el esquema:

```
Session session =  
HibernateUtil.getSessionFactory().getCurrentSession();  
session.beginTransaction();  
// OPERACIONES con la BD (que componen la transacción)  
session.getTransaction().commit();
```

```
private static Reserva createAndStoreReserva(Socio soc, Libro lib) {  
    session = HibernateUtil.getSessionFactory().getCurrentSession();  
    session.beginTransaction();  
    ReservaId resId = new ReservaId(lib.getSignatura(), soc.getNumSocio());  
    Reserva res = new Reserva();  
    res.setId(resId);  
    res.setLibro(lib);  
    res.setSocio(soc);  
    res.setFechaReserva(new Date());  
    res.setDisponible("N");  
    lib.getReservas().add(res);  
    session.save(res); ← Operación con la BD  
    session.getTransaction().commit();  
    log.info("Insertada: "+res);  
    return res;  
}
```

Transacción

Operación con la BD

2.7) Operar con la BD sobre varios objetos: otras alternativas

- Utilizar HQL (Hibernate Query Language)
 - Un lenguaje de interrogación sobre objetos Hibernate parecido a SQL
 - <http://docs.jboss.org/hibernate/core/3.3/reference/en/html/queryhql.html>
- Utilizar Criteria Queries
 - Una alternativa a HQL más orientada a objetos
 - <http://docs.jboss.org/hibernate/core/3.3/reference/en/html/querycriteria.html>

Ejemplo de consulta con HQL

from Libro where signatura = '3'

```
public static Libro getLibro(String sign){  
  
    session = HibernateUtil.getSessionFactory().getCurrentSession();  
    session.beginTransaction();  
    List<Libro> result = (List<Libro>)session.createQuery("from Libro where signatura='"+sign+"'").list();  
    session.getTransaction().commit();  
    if (result.isEmpty()) return null;  
    else return (Libro) result.get(0);  
}
```

NOTA: el nombre de clase es sensible a mayúsculas y minúsculas. La clase se llama "Libro"

O bien: select signatura
from Libro where signatura = '3'

Igual que SQL en este caso

Más ejemplos de preguntas HQL

Extraídas de <http://www.cursohibernate.es/>

Operadores de selección y proyección

```
SELECT p FROM Profesor p WHERE nombre='ISABEL' AND ape1<>'ORELLANA'
```

Funciones agregadas

```
SELECT AVG(c.horas), SUM(c.horas), MIN(c.horas), MAX(c.horas), COUNT(*) FROM Ciclo c
```

Agrupaciones

```
SELECT nombre, count(nombre) FROM Profesor p GROUP BY nombre HAVING count(nombre)>1 ORDER BY count(nombre)
```

Subconsultas anidadas

```
SELECT c.nombre, c.horas FROM Ciclo c WHERE c.horas > (SELECT AVG(c2.horas) FROM Ciclo c2)
```

Joins (inner, left outer, right outer)

```
from Cat as cat left join cat.kittens as kitten with kitten.bodyWeight > 10.0
```

<https://docs.jboss.org/hibernate/orm/3.5/reference/es-ES/html/queryhql.html#queryhql-joins>

Y otros para navegar entre objetos (p.nombre.ape1) y FETCH para cargar en propiedad (colección)

```
SELECT p.nombre.ape1, SIZE(p.correosElectronicos) FROM Profesor p GROUP BY p.nombre.ape1
```

```
SELECT p FROM Profesor p LEFT JOIN FETCH p.correosElectronicos
```

Ejemplo de consulta con Criteria

```
List cats = sess.createCriteria(Cat.class)
.add( Restrictions.like("name", "Fritz%") )
.add( Restrictions.between("weight",
                           minWeight, maxWeight) )
.list();
```

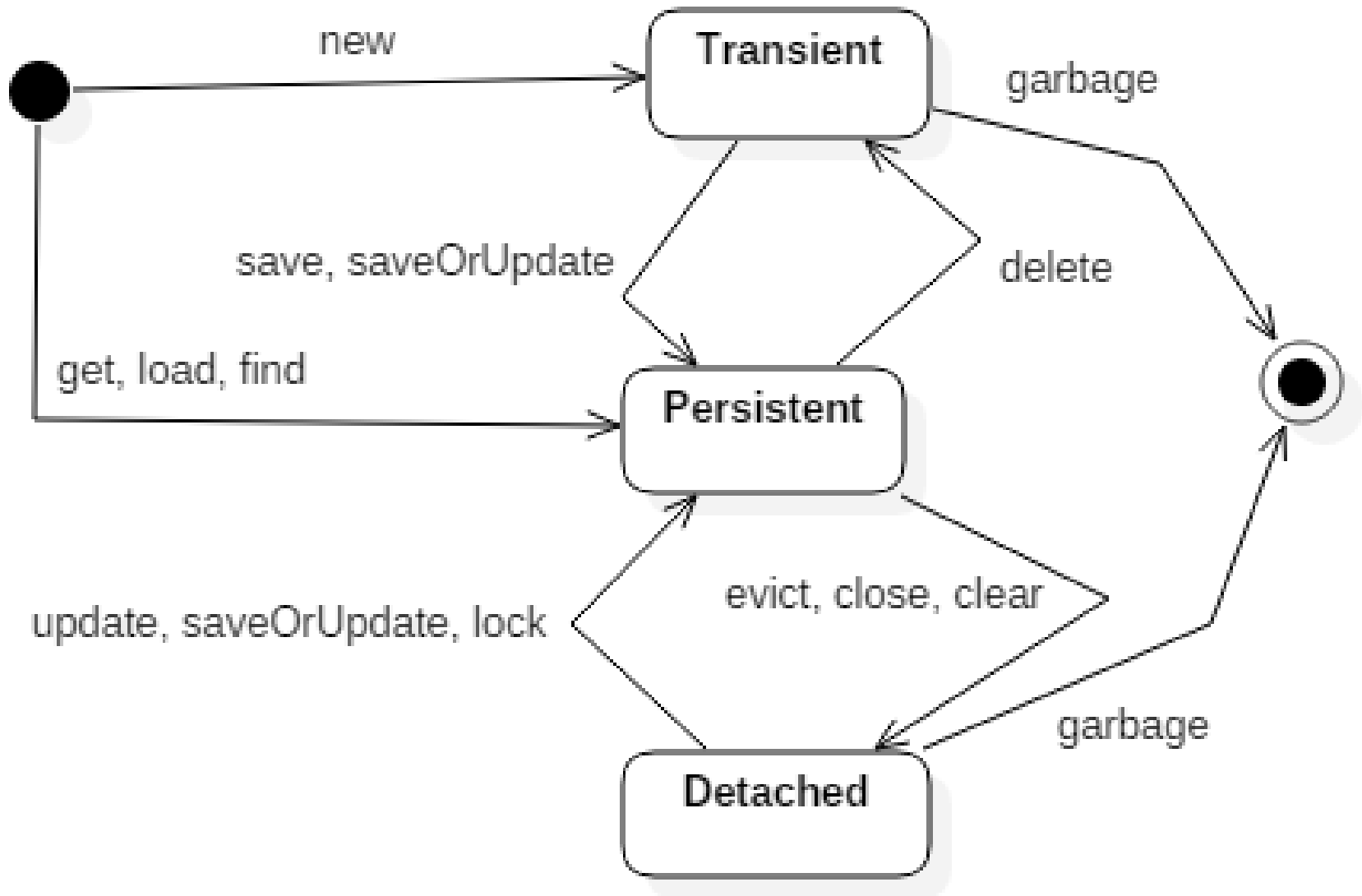
Pregunta extraída de:

<https://docs.jboss.org/hibernate/orm/3.5/reference/es-ES/html/querycriteria.html>

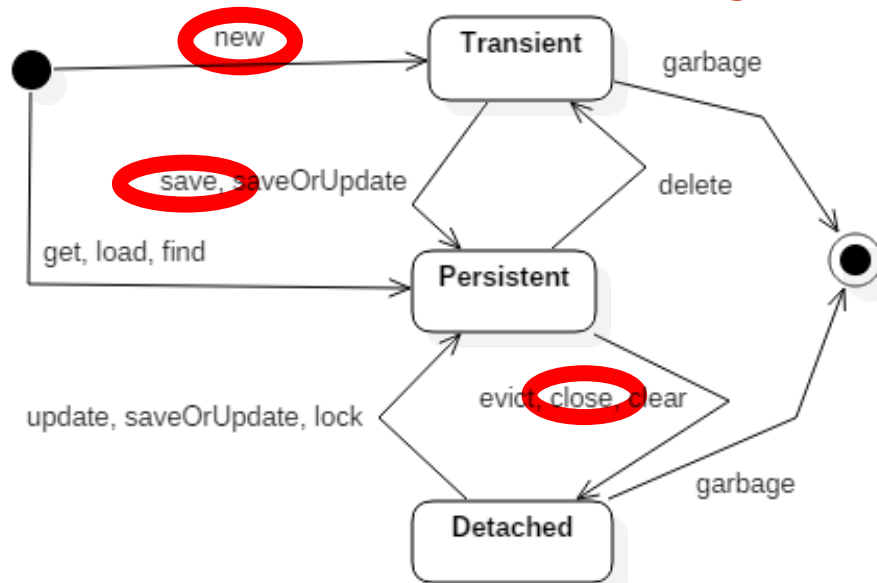
2.8.- Aspectos de interés

- Ciclo de vida de los objetos
- Asociaciones
 - 1:N, 1:1 y N:M
 - Unidireccionales / Bidireccionales
- Valores null en propiedades
- Recuperación de datos:
 - Cómo => fetch Cuándo=> lazy
- Borrado / Actualización => cascade
- Responsable actualizar la asociación
 - Inverse="true"

Ciclo de vida de los objetos



Ciclo de vida de los objetos



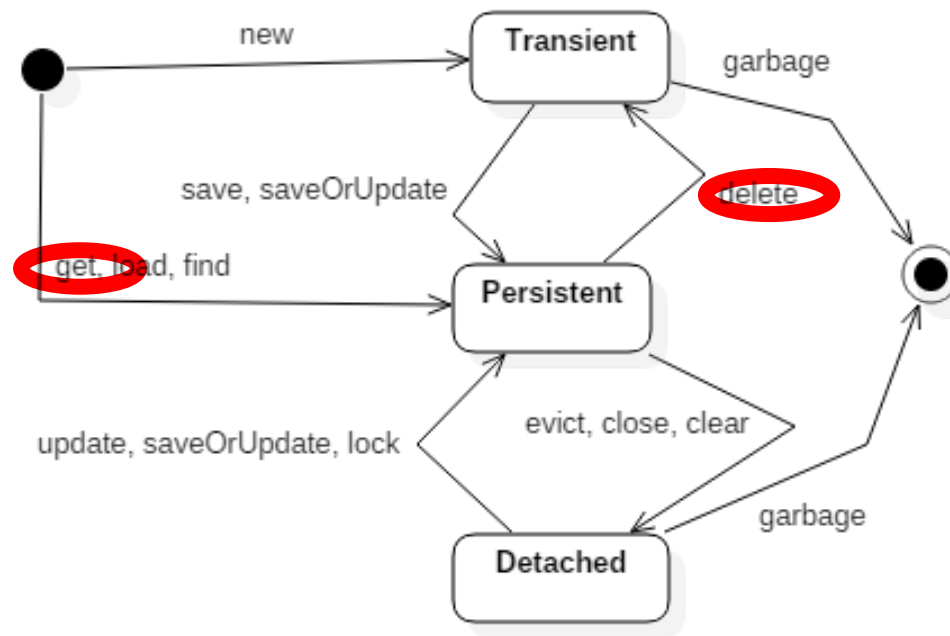
```
private void createAndStoreEventoLogin(Long id, String descripcion, Date fecha) {  
    Session session = HibernateUtil.getSessionFactory().getCurrentSession();  
    session.beginTransaction();  
  
    EventoLogin e = new EventoLogin();  
    e.setId(id);  
    e.setDescripcion(descripcion);  
    e.setFecha(fecha);  
    session.save(e);  
    session.getTransaction().commit();  
}
```

Objeto e es transient (está en memoria principal, pero no en BD)

Ahora e se hace persistente en la BD (con el commit, realmente)

Tras cerrar la sesión, e queda "detached" en Hibernate (borrado de su cache interna)

Ciclo de vida de los objetos



```
private void deleteEventoLogin(Long id) {  
    Session session = HibernateUtil.getSessionFactory().getCurrentSession();  
    session.beginTransaction();  
    EventoLogin e = (EventoLogin) session.get(EventoLogin.class, id);  
    session.delete(e);  
    session.getTransaction().commit();  
}
```

Objeto e se borra de la BD y es transient (en MP)

Objeto e se carga de la BD y es persistente (conectado a la BD)

Asociación 1:N. Acceso unidireccional: (EventoLogin=>Usuario)

```
mysql> select * from usuario;
```

NOMBRE	PASSWORD	TIPO
Nerea	125	estudiante
Pepe	125	estudiante

```
public class Usuario {  
    String nombre;  
    String password;  
    String tipo;
```

```
mysql> select * from eventologin;
```

ID	DESCRIPCION	FECHA	USUARIO	LOGIN
1	NULL	2016-12-01 11:04:29	Pepe	1
2	NULL	2016-12-01 11:04:29	Pepe	0
3	NULL	2016-12-01 11:04:29	Nerea	0

```
public class EventoLogin {  
    private Long id;  
    private String descripcion;  
    private Date fecha;  
    private Usuario usuario;
```

EventoLogin.hbm.xml

```
<many-to-one name="usuario" class="modelo.dominio.Usuario" fetch="join">  
    <column name="USUARIO" />  
</many-to-one>
```

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();  
session.beginTransaction();
```

```
EventoLogin ev=(EventoLogin)session.get(EventoLogin.class, 1L);  
System.out.print("Tipo de usuario de "+ev.getUsuario().getNombre());  
System.out.println(" es "+ev.getUsuario().getTipo());
```

¿Accedería correctamente a los datos de la BD?

Sí



Tipo de usuario de Pepe es estudiante

Asociación 1:N. ¿Acceso unidirecc. Usuario=>EventoLogin ?

```
mysql> select * from usuario;
```

NOMBRE	PASSWORD	TIPO
Nerea	125	estudiante
Pepe	125	estudiante

```
mysql> select * from eventologin;
```

ID	DESCRIPCION	FECHA	USUARIO	LOGIN
1	NULL	2016-12-01 11:04:29	Pepe	1
2	NULL	2016-12-01 11:04:29	Pepe	0
3	NULL	2016-12-01 11:04:29	Nerea	0

```
public class Usuario {  
    String nombre;  
    String password;  
    String tipo;
```

```
public class EventoLogin {  
    private Long id;  
    private String descripcion;  
    private Date fecha;  
    private Usuario usuario;  
    private boolean login;
```

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();  
session.beginTransaction();
```

```
Usuario u1=(Usuario)session.get(Usuario.class, "Pepe");  
Set<EventoLogin> leventos = u1.getEventos();
```

```
for (EventoLogin ev: leventos){  
    System.out.print("Evento de "+u1.getNombre()+" =>");  
    System.out.println("Id: " + ev.getId() + " Fecha: " + ev.getFecha());  
}
```

¿Accedería correctamente a los datos de la BD?

No, no hay atributo "eventos" en Usuario => ~~getEventos()~~

Asociación 1:N. ¿Acceso unidirecc. Usuario=>EventoLogin ?

```
mysql> select * from usuario;
```

NOMBRE	PASSWORD	TIPO
Nerea	125	estudiante
Pepe	125	estudiante

```
mysql> select * from eventologin;
```

ID	DESCRIPCION	FECHA	USUARIO	LOGIN
1	NULL	2016-12-01 11:04:29	Pepe	1
2	NULL	2016-12-01 11:04:29	Pepe	0
3	NULL	2016-12-01 11:04:29	Nerea	0

```
public class Usuario {  
    String nombre;  
    String password;  
    String tipo;  
    Set<EventoLogin> eventos;
```

Único cambio

```
    Session session = HibernateUtil.getSessionFactory().getCurrentSession();  
    session.beginTransaction();
```

```
    Usuario u1=(Usuario)session.get(Usuario.class, "Pepe");  
    Set<EventoLogin> leventos = u1.getEventos();
```

```
    for (EventoLogin ev: leventos){  
        System.out.print("Evento de "+u1.getNombre()+" =>");  
        System.out.println("Id: " + ev.getId() + " Fecha: " + ev.getFecha());  
    }
```

¿Accedería correctamente a los datos de la BD?

No, si no está definido el MAPPING. Error “nada útil” de ejecución (no de compilación)

```
Exception in thread "main" java.lang.NullPointerException  
at principal.CrearEventos.main(CrearEventos.java:143)
```



Asociación 1:N con Bidireccionalidad (Usuario<=>EventoLogin)

```
mysql> select * from usuario;
```

NOMBRE	PASSWORD	TIPO
Nerea	125	estudiante
Pepe	125	estudiante

```
mysql> select * from eventologin;
```

ID	DESCRIPCION	FECHA	USUARIO	LOGIN
1	NULL	2016-12-01 11:04:29	Pepe	1
2	NULL	2016-12-01 11:04:29	Pepe	0
3	NULL	2016-12-01 11:04:29	Nerea	0

 Usuario.hbm.xml 

```
<set name="eventos" table="EVENTOLOGIN" inverse="true" lazy="true">
  <key>
    <column name="USUARIO" />
  </key>
  <one-to-many class="modelo.dominio.EventoLogin" />
</set>
```

Nota: cambiado a mano NOMBRE por USUARIO ya que Hibernate Tools asigna el mismo nombre que ID de USUARIO (que es NOMBRE)

¿Accedería correctamente a los datos de la BD?

Sí



```
Evento de Pepe =>Id: 2 Fecha: 2016-12-01 11:41:42.0
Evento de Pepe =>Id: 1 Fecha: 2016-12-01 11:41:42.0
```

Asociación 1:1 (basada en claves prim) (One Usuario \Leftrightarrow One Persona)

```
public class Persona {  
    String nombrePersona;  
    String tfno;  
    Usuario usuario;
```

```
public class Usuario {  
    String nombre;  
    String password;  
    String tipo;  
    Set<EventoLogin> eventos;  
    Persona persona;
```

← ID →

Persona.hbm.xml

```
<one-to-one name="usuario" class="modelo.dominio.Usuario"></one-to-one>
```

Usuario.hbm.xml

```
<one-to-one name="persona" class="modelo.dominio.Persona"></one-to-one>
```

NO HAY QUE AÑADIR NADA MÁS EN EL MAPPING

En las tablas de la BD NO se añade ningún atributo/columna nueva: existe una asociación 1:1 entre una persona y un usuario cuando tienen el mismo valor para la clave primaria (nombrePersona de Persona y nombre de Usuario)

```
mysql> select * from usuario;  
+-----+-----+-----+  
| NOMBRE | PASSWORD | TIPO |  
+-----+-----+-----+  
| Pepe   | 125      | estudiante |  
+-----+-----+-----+
```

```
mysql> select * from persona;  
+-----+-----+  
| NOMBREPERSONA | TFNO |  
+-----+-----+  
| Pepe           | 943010102 |  
+-----+-----+
```

**ASOCIACIÓN
N 1:1**

Asociación 1:1 (basada en claves prim) (One Usuario \Leftrightarrow One Persona)

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();  
session.beginTransaction();
```

```
Usuario u1 = new Usuario();  
u1.setNombre("Pepe");  
u1.setPassword("125");  
u1.setTipo("estudiante");  
Persona p1 = new Persona();  
p1.setNombrePersona("Pepe");  
p1.setTfno("943010102");  
p1.setUsuario(u1);  
u1.setPersona(p1);  
session.save(u1);  
session.save(p1);  
session.getTransaction().commit();
```

```
mysql> select * from usuario;  
+-----+-----+-----+  
| NOMBRE | PASSWORD | TIPO |  
+-----+-----+-----+  
| Pepe   | 125      | estudiante |  
+-----+-----+-----+
```

```
mysql> select * from persona;  
+-----+-----+  
| NOMBREPERSONA | TFNO |  
+-----+-----+  
| Pepe          | 943010102 |  
+-----+-----+
```

ASOCIACIÓN
1:1

Acceso Usuario => Persona

```
Usuario u2=(Usuario)session.get(Usuario.class, "Pepe");  
System.out.print(u2.getPersona().getNombrePersona()+"/");  
System.out.println(u2.getPersona().getTfno());
```

→ Pepe/943010102

Acceso Persona => Usuario


```
Persona p2=(Persona)session.get(Persona.class, "Pepe");  
System.out.print(p2.getUsuario().getNombre()+"/");  
System.out.println(p2.getUsuario().getTipo());
```

→ Pepe/estudiante

Asociación 1:1 (no con claves primarias)

Acceso unidirecc. Usuario=>Persona

Si se quiere que la asociación (join) se haga entre una clave primaria y otra extranjera: Usuario (c. prim) => Persona (c. ext)

```
Usuario.hbm.xml   
<one-to-one name="persona" property-ref="nombrePersona" class="modelo.dominio.Persona"></one-to-one>
```

La propiedad clave de Persona es ahora código, y queremos hacer el join de `usuario.nombre=persona.nombrePersona`

```
public class Persona {  
    int codigo;  
    String nombrePersona;  
    String tfno;  
    Usuario usuario;  
}  
  
public class Usuario {  
    String nombre;  
    String password;  
    String tipo;  
    Set<EventoLogin> eventos;  
    Persona persona;  
}
```

Acceso Usuario => Persona

```
Usuario u2=(Usuario)session.get(Usuario.class, "Pepe");  
System.out.print(u2.getPersona().getNombrePersona()+"/");  
System.out.println(u2.getPersona().getTfno());
```

Los valores de la propiedad "persona" de Usuario son los de la clave de Usuario (columna NOMBRE)

Pepe/943010102

```
mysql> select * from usuario;  
+-----+-----+-----+  
| NOMBRE | PASSWORD | TIPO |  
+-----+-----+-----+  
| Pepe   | 125      | estudiante |  
+-----+-----+-----+
```

```
mysql> select * from persona;  
+-----+-----+-----+  
| CODIGO | NOMBREPERSONA | TFNO |  
+-----+-----+-----+  
| 1      | Pepe          | 943010102 |  
+-----+-----+-----+
```

Asociación 1:1 (no con claves primarias) (Acceso Persona => Usuario) => bidirec.

Si se quiere que la asociación (join) se haga entre una clave primaria y otra extranjera: Persona (c. ext) => Usuario (c. prim)

```
Persona.hbm.xml ✖  
<many-to-one name="usuario" class="modelo.dominio.Usuario" column="NOMBREPERSONA"  
  unique="true" not-null="true" insert="false" update="false"/>
```

```
<property name="nombrePersona" type="java.lang.String" >  
  <column name="NOMBREPERSONA" />  
  ..  
  ..  
  public class Persona {  
    int codigo;  
    String nombrePersona;  
    String tfno;  
    Usuario usuario;  
  }  
  public class Usuario {  
    String nombre;  
    String password;  
    String tipo;  
    Set<EventoLogin> eventos;  
    Persona persona;  
  }
```

La propiedad usuario se define como "many-to-one" con unique="true" (=>"one-to-one") Los atributos insert y update a false indican que la propiedad usuario es derivada: su valor se cambia con otra propiedad mapeada en la misma columna (nombrePersona)

Acceso Persona => Usuario

```
Persona p2=(Persona)session.get(Persona.class, 1);  
System.out.print(p2.getUsuario().getNombre()+"/");  
System.out.println(p2.getUsuario().getTipo());
```

Pepe/estudiante

```
mysql> select * from usuario;  
+-----+-----+-----+  
| NOMBRE | PASSWORD | TIPO |  
+-----+-----+-----+  
| Pepe  | 125     | estudiante |  
+-----+-----+-----+
```

```
mysql> select * from persona;  
+-----+-----+-----+  
| CODIGO | NOMBREPERSONA | TFNO |  
+-----+-----+-----+  
| 1     | Pepe         | 943010102 |  
+-----+-----+-----+
```

Valores null en propiedades

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();  
session.beginTransaction();
```

```
EventoLogin ev = new EventoLogin();  
ev.setLogin(true);  
ev.setFecha(new Date());  
ev.setDescripcion("Nerea haciendo login");  
session.save(ev);
```

```
session.getTransaction().commit();
```

```
public class EventoLogin {  
    private Long id;  
    private String descripcion;  
    private Date fecha;  
    private Usuario usuario;  
    private boolean login;
```

```
mysql> describe eventologin;  
+-----+-----+-----+-----+-----+-----+  
| Field      | Type          | Null | Key | Default | Extra |  
+-----+-----+-----+-----+-----+-----+  
| ID         | bigint(20)    | NO   | PRI | NULL     |       |  
| DESCRIPCION | varchar(255)  | YES  |     | NULL     |       |  
| FECHA      | datetime      | YES  |     | NULL     |       |  
| USUARIO    | varchar(255)  | YES  | MUL | NULL     |       |  
| LOGIN      | int(11)       | YES  |     | NULL     |       |  
+-----+-----+-----+-----+-----+-----+
```

¿Dará error porque a EventoLogin no se le ha asignado usuario?

No, porque no se dice que usuario NO puede ser NULL

```
mysql> select * from eventologin;  
+----+-----+-----+-----+-----+  
| ID | DESCRIPCION | FECHA          | USUARIO | LOGIN |  
+----+-----+-----+-----+-----+  
| 1  | Nerea haciendo login | 2016-12-01 00:36:38 | NULL    | 1    |  
+----+-----+-----+-----+-----+
```

```
<many-to-one name="usuario" class="modelo.dominio.Usuario" fetch="join">  
    <column name="USUARIO" />  
</many-to-one>
```


Valores null en propiedades

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();  
session.beginTransaction();
```

```
EventoLogin ev = new EventoLogin();  
ev.setLogin(true);  
ev.setFecha(new Date());  
ev.setDescripcion("Nerea haciendo login");  
session.save(ev);
```

```
session.getTransaction().commit();
```

```
public class EventoLogin {  
    private Long id;  
    private String descripcion;  
    private Date fecha;  
    private Usuario usuario;  
    private boolean login;
```

```
mysql> describe eventologin;
```

Field	Type	Null	Key	Default	Extra
ID	bigint(20)	NO	PRI	NULL	
DESCRIPCION	varchar(255)	YES		NULL	
FECHA	datetime	YES		NULL	
USUARIO	varchar(255)	NO	MUL	NULL	
LOGIN	int(11)	YES		NULL	

¿Daría error si el mapping fuera así?

```
<many-to-one name="usuario" class="modelo.dominio.Usuario" fetch="join">  
    <column name="USUARIO" not-null="true" />  
</many-to-one>
```

Sí, porque usuario NO puede ser NULL

```
Exception in thread "main" org.hibernate.PropertyValueException: not-null property references a null or transient value: modelo.dominio.EventoLogin.usuario  
at org.hibernate.engine.Nullability.checkNotNull(Nullability.java:100)  
at org.hibernate.event.def.AbstractSaveEventListener.performSaveOrReplicate(AbstractSaveEventListener.java:312)
```

¿Cómo recupera datos de la BD?

```
mysql> select * from eventologin;
```

ID	DESCRIPCION	FECHA	USUARIO	LOGIN
1	NULL	2016-12-01 01:29:53	Pepe	1
2	NULL	2016-12-01 01:29:53	Pepe	0

```
mysql> select * from usuario;
```

NOMBRE	PASSWORD	TIPO
Pepe	125	estudiante

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();  
session.beginTransaction();
```

```
EventoLogin ev=(EventoLogin)session.get(EventoLogin.class, 1L);  
System.out.println(ev.getUsuario().getTipo());
```

```
Hibernate:  
select  
  eventologi0_.ID as ID0_1_,  
  eventologi0_.DESCRIPCION as DESCRIPC2_0_1_,  
  eventologi0_.FECHA as FECHA0_1_,  
  eventologi0_.USUARIO as USUARIO0_1_,  
  eventologi0_.LOGIN as LOGIN0_1_,  
  usuario1_.NOMBRE as NOMBRE1_0_,  
  usuario1_.PASSWORD as PASSWORD1_0_,  
  usuario1_.TIPO as TIPO1_0_  
from  
  EVENTOLOGIN eventologi0_  
inner join  
  USUARIO usuario1_  
  on eventologi0_.USUARIO=usuario1_.NOMBRE  
where  
  eventologi0_.ID=?  
estudiante
```

En la traza de la consola puede verse que ha ejecutado el join de eventologin y usuario en el session.get. Por eso ya tenía todos los datos.

Se trae con un join la info de "todos" ("uno") los usuarios del evento de id 1

```
<many-to-one name="usuario" class="modelo.dominio.Usuario" fetch="join">  
  <column name="USUARIO" not-null="true" />  
</many-to-one>
```

¿Cómo recupera datos de la BD?

```
mysql> select * from eventologin;
+----+-----+-----+-----+-----+
| ID | DESCRIPCION | FECHA | USUARIO | LOGIN |
+----+-----+-----+-----+-----+
| 1  | NULL        | 2016-12-01 01:29:53 | Pepe   | 1     |
| 2  | NULL        | 2016-12-01 01:29:53 | Pepe   | 0     |
+----+-----+-----+-----+-----+
```

```
mysql> select * from usuario;
+-----+-----+-----+
| NOMBRE | PASSWORD | TIPO |
+-----+-----+-----+
| Pepe   | 125      | estudiante |
+-----+-----+-----+
```

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();
session.beginTransaction();
```

```
EventoLogin ev=(EventoLogin)session.get(EventoLogin.class, 1L);
System.out.println(ev.getUsuario().getTipo());
```

```
-----
eventologi0_.DESCRIPCION as DESCRIPC2_0_0_,
eventologi0_.FECHA as FECHA0_0_,
eventologi0_.USUARIO as USUARIO0_0_,
eventologi0_.LOGIN as LOGIN0_0_
from
EVENTOLOGIN eventologi0_
where
eventologi0_.ID=?
Hibernate:
select
usuario0_.NOMBRE as NOMBRE1_0_,
usuario0_.PASSWORD as PASSWORD1_0_,
usuario0_.TIPO as TIPO1_0_
from
USUARIO usuario0_
where
usuario0_.NOMBRE=?
estudiante
```

En la traza de la consola puede verse que:

Ejecuta esa pregunta select para el get de ev. 1

Y después ejecuta esta select para el getUsuario().getTipo() del ev. 1

```
<many-to-one name="usuario" class="modelo.dominio.Usuario" fetch="select">
  <column name="USUARIO" not-null="true" />
</many-to-one>
```

Para cada objeto, se trae la info con "select"

¿Cuándo recupera datos de la BD?

```
mysql> select * from eventologin;
+----+-----+-----+-----+-----+
| ID | DESCRIPCION | FECHA          | USUARIO | LOGIN |
+----+-----+-----+-----+-----+
| 1  | NULL        | 2016-12-01 01:29:53 | Pepe   | 1    |
| 2  | NULL        | 2016-12-01 01:29:53 | Pepe   | 0    |
+----+-----+-----+-----+-----+
```

```
mysql> select * from usuario;
+-----+-----+-----+
| NOMBRE | PASSWORD | TIPO |
+-----+-----+-----+
| Pepe   | 125     | estudiante |
+-----+-----+-----+
```

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();
session.beginTransaction();
```

```
Usuario u1=(Usuario)session.get(Usuario.class, "Pepe");
Set<EventoLogin> leventos = u1.getEventos();
```

Se traen los eventos cuando se necesitan, esto es, cuando se hace `getEventos` => estrategia "perezosa"

```
<set name="eventos" table="EVENTOLOGIN" inverse="true" lazy="true">
  <key>
    <column name="USUARIO" />
  </key>
  <one-to-many class="modelo.dominio.EventoLogin" />
</set>
```

```
Hibernate:
select
  usuario0_.NOMBRE as NOMBRE1_0_,
  usuario0_.PASSWORD as PASSWORD1_0_,
  usuario0_.TIPO as TIPO1_0_
from
  USUARIO usuario0_
where
  usuario0_.NOMBRE=?
Hibernate:
select
  eventos0_.USUARIO as USUARIO1_1_,
  eventos0_.ID as ID1_,
  eventos0_.ID as ID0_0_,
  eventos0_.DESCRIPCION as DESCRIPC2_0_0_,
  eventos0_.FECHA as FECHA0_0_,
  eventos0_.USUARIO as USUARIO0_0_,
  eventos0_.LOGIN as LOGIN0_0_
from
  EVENTOLOGIN eventos0_
where
  eventos0_.USUARIO=?
```

Borrado en cascada

```
mysql> select * from usuario;
+-----+-----+-----+
| NOMBRE | PASSWORD | TIPO |
+-----+-----+-----+
| Nerea  | 125      | estudiante |
| Pepe   | 125      | estudiante |
+-----+-----+-----+
```

```
mysql> select * from eventologin;
+-----+-----+-----+-----+-----+-----+
| ID | DESCRIPCION | FECHA | USUARIO | LOGIN |
+-----+-----+-----+-----+-----+-----+
| 1 | NULL | 2016-12-01 02:58:35 | Pepe | 1 |
| 2 | NULL | 2016-12-01 02:58:35 | Pepe | 0 |
| 3 | NULL | 2016-12-01 02:58:35 | Nerea | 0 |
+-----+-----+-----+-----+-----+-----+
```

```
<many-to-one name="usuario" cascade="delete" class="modelo.dominio.Usuario" fetch="select">
  <column name="USUARIO" not-null="true" />
</many-to-one>
```

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();
session.beginTransaction();
```

```
EventoLogin ev3=(EventoLogin)session.get(EventoLogin.class, 3L);
session.delete(ev3);
```

```
session.getTransaction().commit();
HibernateUtil.getSessionFactory().close();
```

¿Qué hace el delete de ev3 en cascada?

Borra el evento 3 y su usuario (Nerea)

```
mysql> select * from usuario;
+-----+-----+-----+
| NOMBRE | PASSWORD | TIPO |
+-----+-----+-----+
| Pepe   | 125      | estudiante |
+-----+-----+-----+
```

```
mysql> select * from eventologin;
+-----+-----+-----+-----+-----+-----+
| ID | DESCRIPCION | FECHA | USUARIO | LOGIN |
+-----+-----+-----+-----+-----+-----+
| 1 | NULL | 2016-12-01 03:11:40 | Pepe | 1 |
| 2 | NULL | 2016-12-01 03:11:40 | Pepe | 0 |
+-----+-----+-----+-----+-----+-----+
```

Borrado en cascada (viendo la traza)

```
<many-to-one name="usuario" cascade="delete" class="modelo.dominio.Usuario" fetch="select">
  <column name="USUARIO" not-null="true" />
</many-to-one>
```

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();
session.beginTransaction();
```

```
EventoLogin ev3=(EventoLogin)session.get(EventoLogin.class, 3L);
session.delete(ev3);
```

```
session.getTransaction().commit();
HibernateUtil.getSessionFactory().close();
```

```
Hibernate:
select
  eventologi0_.ID as ID0_0_,
  eventologi0_.DESCRIPCION as DESCRIPC2_0_0_,
  eventologi0_.FECHA as FECHA0_0_,
  eventologi0_.USUARIO as USUARIO0_0_,
  eventologi0_.LOGIN as LOGIN0_0_
from
  EVENTOLOGIN eventologi0_
where
  eventologi0_.ID=?
```

```
Hibernate:
select
  usuario0_.NOMBRE as NOMBRE1_0_,
  usuario0_.PASSWORD as PASSWORD1_0_,
  usuario0_.TIPO as TIPO1_0_
from
  USUARIO usuario0_
where
  usuario0_.NOMBRE=?
```

```
Hibernate:
delete
from
  EVENTOLOGIN
where
  ID=?
```

```
Hibernate:
delete
from
  USUARIO
where
  NOMBRE=?
```

Get evento id=3

Traer su usuario por
fetch="select"

Delete evento id=3

Delete su usuario por
cascade="true"

Borrado en cascada

```
mysql> select * from usuario;
+-----+-----+-----+
| NOMBRE | PASSWORD | TIPO |
+-----+-----+-----+
| Nerea  | 125      | estudiante |
| Pepe   | 125      | estudiante |
+-----+-----+-----+
```

```
mysql> select * from eventologin;
+-----+-----+-----+-----+-----+-----+
| ID | DESCRIPCION | FECHA | USUARIO | LOGIN |
+-----+-----+-----+-----+-----+-----+
| 1  | NULL        | 2016-12-01 02:58:35 | Pepe   | 1 |
| 2  | NULL        | 2016-12-01 02:58:35 | Pepe   | 0 |
| 3  | NULL        | 2016-12-01 02:58:35 | Nerea  | 0 |
+-----+-----+-----+-----+-----+-----+
```

```
<many-to-one name="usuario" cascade="delete" class="modelo.dominio.Usuario" fetch="select">
  <column name="USUARIO" not-null="true" />
</many-to-one>
```

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();
session.beginTransaction();
```

```
EventoLogin ev1=(EventoLogin)session.get(EventoLogin.class, 1L);
session.delete(ev1);
```

```
session.getTransaction().commit();
HibernateUtil.getSessionFactory().close();
```

Porque al borrar Pepe, quedan eventos 1 y 2 de Pepe
=> Viola restric. integridad

Sin embargo, el delete de ev1 da un error. ¿Por qué?

Could not execute JDBC batch update

"Cannot delete or update a parent row: a foreign key constraint fails
ement.java:1469)

Actualización en cascada

```
<set name="eventos" table="EVENTOLOGIN" cascade="save-update" inverse="true" lazy="false" fetch="join">  
  <key>  
    <column name="USUARIO" />  
  </key>  
  <one-to-many class="modelo.dominio.EventoLogin" />  
</set>
```



Usuario.hbm.xml

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();  
session.beginTransaction();
```

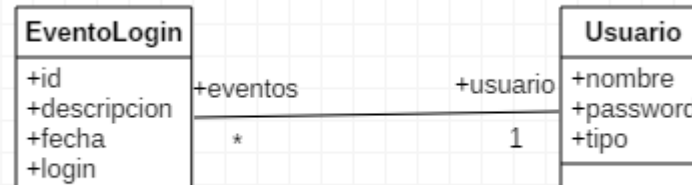
```
Usuario u1=(Usuario)session.get(Usuario.class, "Nerea");
```

```
EventoLogin ev=new EventoLogin();  
ev.setLogin(true);  
ev.setFecha(new Date());  
ev.setDescripcion("Nerea haciendo login");  
ev.setUsuario(u1);  
Set hs=u1.getEventos();  
hs.add(ev);  
u1.setEventos(hs);  
//session.save(ev);  
session.update(u1);  
session.getTransaction().commit();
```

En este caso se actualiza el usuario, y de la actualización del evento se encarga también usuario porque se ha definido actualización en cascada. No es necesario salvarlo explícitamente.

```
Hibernate:  
insert  
into  
  EVENTOLOGIN  
(DESCRIPCION, FECHA, USUARIO, LOGIN, ID)  
values  
(?, ?, ?, ?, ?)
```


¿Quién es responsable de actualizar la asociación? (atributo inverse)



La asociación UML se implementa añadiendo 2 atributos (usuario y eventos) en las clases Java. Ambos atributos permiten acceso bidireccional, pero añaden redundancia: hay que añadir el usuario a “usuario” de EventoLogin y el evento a “eventos” de Usuario

```
public class EventoLogin {
    private Long id;
    private String descripcion;
    private Date fecha;
    private Usuario usuario;
    private boolean login;
```

```
public class Usuario {
    String nombre;
    String password;
    String tipo;
    Set<EventoLogin> eventos;
```

Sin embargo, en la BD Relacional, es eventologin quien almacena el usuario, y no al revés

```
mysql> select * from usuario;
+-----+-----+-----+
| NOMBRE | PASSWORD | TIPO |
+-----+-----+-----+
| Nerea  | 125      | estudiante |
| Pepe   | 125      | estudiante |
+-----+-----+-----+
```

```
mysql> select * from eventologin;
+----+-----+-----+-----+-----+
| ID | DESCRIPCION | FECHA | USUARIO | LOGIN |
+----+-----+-----+-----+-----+
| 1  | NULL        | 2016-12-01 11:04:29 | Pepe   | 1     |
| 2  | NULL        | 2016-12-01 11:04:29 | Pepe   | 0     |
| 3  | NULL        | 2016-12-01 11:04:29 | Nerea  | 0     |
+----+-----+-----+-----+-----+
```

En el mapping de Hibernate, con `inverse="true"`, indicamos el atributo responsable de actualizar la asociación en la BD, que debería ser el atributo “eventos” de la clase Usuario

¿Quién es responsable de actualizar la asociación? (atributo inverse)



```
mysql> select * from eventologin;
```

ID	DESCRIPCION	FECHA	USUARIO	LOGIN
1	NULL	2016-12-01 16:37:00	Pepe	1
2	NULL	2016-12-01 16:37:00	Pepe	0
3	NULL	2016-12-01 16:37:00	Nerea	0

```
mysql> select * from usuario;
```

NOMBRE	PASSWORD	TIPO
Nerea	125	estudiante
Pepe	125	estudiante

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();  
session.beginTransaction();
```

```
Usuario u1=(Usuario)session.get(Usuario.class, "Nerea");  
EventoLogin ev=new EventoLogin();  
ev.setLogin(true);  
ev.setFecha(new Date());  
ev.setDescripcion("Nerea haciendo login");  
ev.setUsuario(u1);  
Set hs=u1.getEventos();  
hs.add(ev);  
u1.setEventos(hs);  
session.save(ev);  
session.update(u1);  
session.getTransaction().commit();
```

Es necesario dar persistencia en la BD al nuevo evento creado ev

Al actualizar usuario u1 en la BD, se actualiza en la BD el evento ev de la asociación, ya que (inverse="false") indica que es la clase usuario la responsable de la actualización

```
mysql> select * from eventologin;
```

ID	DESCRIPCION	FECHA	USUARIO	LOGIN
1	NULL	2016-12-01 16:38:46	Pepe	1
2	NULL	2016-12-01 16:38:46	Pepe	0
3	NULL	2016-12-01 16:38:46	Nerea	0
4	Nerea haciendo login	2016-12-01 16:38:46	Nerea	1

Usuario.hbm.xml

```
<set name="eventos" table="EVENTOLOGIN" inverse="false" lazy="false" fetch="join">  
  <key>  
    <column name="USUARIO" />  
  </key>  
  <one-to-many class="modelo.dominio.EventoLogin" />  
</set>
```



inverse="false" => EventoLogin NO actualiza la asociación (sino Usuario)




¿Quién es responsable de actualizar la asociación? (atributo inverse)

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();  
session.beginTransaction();
```

```
Usuario u1=(Usuario)session.get(Usuario.class, "Nerea");  
EventoLogin ev=new EventoLogin();  
ev.setLogin(true);  
ev.setFecha(new Date());  
ev.setDescripcion("Nerea haciendo login");  
ev.setUsuario(u1);  
Set hs=u1.getEventos();  
hs.add(ev);  
u1.setEventos(hs);  
session.save(ev);  
session.update(u1);  
session.getTransaction().commit();
```

```
Hibernate: -  
insert  
into  
    EVENTOLOGIN  
    (DESCRIPCION, FECHA, USUARIO, LOGIN, ID)  
values  
    (?, ?, ?, ?, ?)  
Hibernate:  
update  
    EVENTOLOGIN  
set  
    USUARIO=?  
where  
    ID=?
```

NOTA: el UPDATE es INNECESARIO. Actualizar la propiedad "eventos" de "Usuario" consiste en actualizar la columna "USUARIO" de "EVENTOLOGIN", pero es inútil ya que es exactamente lo que se hace al guardar la propiedad "usuario" de "EventoLogin"

 Usuario.hbm.xml

```
<set name="eventos" table="EVENTOLOGIN" inverse="false"  
  <key  
    <column name="USUARIO" />  
  </key>  
  <one-to-many class="modelo.dominio.EventoLogin" />  
</set>
```

 EventoLogin.hbm.xml ;

```
<class name="modelo.dominio.EventoLogin" table="EVENTOLOGIN">  
  <many-to-one name="usuario" class="modelo.dominio.Usuario" f  
    <column name="USUARIO" />  
</many-to-one>
```

¿Quién es responsable de actualizar la asociación? (atributo inverse)



```
mysql> select * from eventologin;
```

ID	DESCRIPCION	FECHA	USUARIO	LOGIN
1	NULL	2016-12-01 16:37:00	Pepe	1
2	NULL	2016-12-01 16:37:00	Pepe	0
3	NULL	2016-12-01 16:37:00	Nerea	0

```
mysql> select * from usuario;
```

NOMBRE	PASSWORD	TIPO
Nerea	125	estudiante
Pepe	125	estudiante

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();  
session.beginTransaction();
```

```
Usuario u1=(Usuario)session.get(Usuario.class, "Nerea");  
EventoLogin ev=new EventoLogin();  
ev.setLogin(true);  
ev.setFecha(new Date());  
ev.setDescripcion("Nerea haciendo login");  
ev.setUsuario(u1);  
Set hs=u1.getEventos();  
hs.add(ev);  
u1.setEventos(hs);  
session.save(ev);  
session.update(u1);  
session.getTransaction().commit();
```

El código es el mismo: se guarda el evento y se actualiza el usuario

Pero en la traza se ve que para update(u1) no se lanza sentencia SQL alguna (ya que no es responsable de actualizar el evento ev)

Hibernate:

```
insert  
into  
EVENTOLOGIN  
(DESCRIPCION, FECHA, USUARIO, LOGIN, ID)  
values  
(?, ?, ?, ?, ?)
```



Usuario.hbm.xml

```
<set name="eventos" table="EVENTOLOGIN" inverse="true" lazy="false" fetch="join">  
  <key>  
    <column name="USUARIO" />  
  </key>  
  <one-to-many class="modelo.dominio.EventoLogin" />  
</set>
```

↓ Sí

inverse="true" => EventoLogin sí actualiza la asociación (no Usuario)

¿Quién es responsable de actualizar la asociación? (inverse="true" en MANY)

CONSEJO: en una asociación ONE-TO-MANY o MANY-TO-ONE hay que poner inverse="true" en la parte MANY

```
<set name="eventos" table="EVENTOLOGIN" inverse="true" lazy="false" fetch="join">
  <key>
    <column name="USUARIO" />
  </key>
  <one-to-many class="modelo.dominio.EventoLogin" />
</set>
```



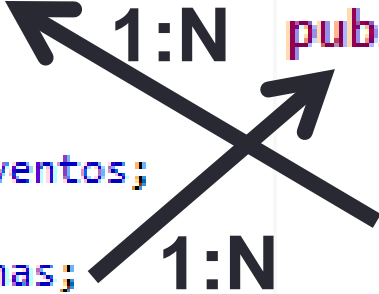
```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();
session.beginTransaction();
```

```
Usuario u1=(Usuario)session.get(Usuario.class, "Nerea");
EventoLogin ev=new EventoLogin();
ev.setLogin(true);
ev.setFecha(new Date());
ev.setDescripcion("Nerea haciendo login");
ev.setUsuario(u1);
//Set hs=u1.getEventos();
//hs.add(ev);
//u1.setEventos(hs);
session.save(ev);
//session.update(u1);
session.getTransaction().commit();
```

De esta manera, se puede ver que **NI SIQUIERA ES NECESARIO** salvar el usuario, ni añadirle el nuevo evento, ya que no va a ser responsable de su actualización: es suficiente con **SALVAR** el evento, que será responsable de su actualización

Asociación N:M (Many Usuario – Many Persona)

```
public class Usuario {  
    String nombre;  
    String password;  
    String tipo;  
    Set<EventoLogin> eventos;  
    Persona persona;  
    Set<Maquina> maquinas;  
}  
  
public class Maquina {  
    int codigo;  
    String nombreMaquina;  
    Set<Usuario> usuarios;  
}
```



El mapping automático con Hibernate Tools ha dado lo siguiente: (porque no puede suponer que es M:N, sino dos 1:N)

 Maquina.hbm.xml

```
<set name="usuarios" table="USUARIO" inverse="false" lazy="true">  
    <key>  
        <column name="CODIGO" />  
    </key>  
    <one-to-many class="modelo.dominio.Usuario" />  
</set>
```

 Usuario.hbm.xml

```
<set name="maquinas" table="MAQUINA" inverse="false" lazy="true">  
    <key>  
        <column name="NOMBRE" />  
    </key>  
    <one-to-many class="modelo.dominio.Maquina" />  
</set>
```

Asociación N:M (Many Usuario – Many Persona)

```
public class Usuario {
    String nombre;
    String password;
    String tipo;
    Set<EventoLogin> eventos;
    Persona persona;
    Set<Maquina> maquinas;
}

public class Maquina {
    int codigo;
    String nombreMaquina;
    Set<Usuario> usuarios;
}
```

N:M

Hay que poner `inverse="true"` en uno y solo uno de los dos atributos

```
mysql> describe maquina;
+-----+-----+-----+-----+
| Field | Type   | Null | Key |
+-----+-----+-----+-----+
| CODIGO | int(11) | NO   | PRI |
| NOMBREMAQUINA | varchar(255) | YES |   |
+-----+-----+-----+-----+
```

```
mysql> describe maquinausuario;
+-----+-----+-----+-----+
| Field | Type   | Null | Key |
+-----+-----+-----+-----+
| NOMBRE | varchar(255) | NO   | PRI |
| CODIGO | int(11) | NO   | PRI |
+-----+-----+-----+-----+
```

```
mysql> describe usuario;
+-----+-----+-----+-----+
| Field | Type   | Null | Key |
+-----+-----+-----+-----+
| NOMBRE | varchar(255) | NO   | PRI |
| PASSWORD | varchar(255) | YES |   |
| TIPO | varchar(255) | YES |   |
+-----+-----+-----+-----+
```

```
Maquina.hbm.xml
<set name="usuarios" table="MAQUINAUSUARIO" inverse="false" lazy="true">
  <key>
    <column name="CODIGO" not-null="true"/>
  </key>
  <many-to-many class="modelo.dominio.Usuario">
    <column name="NOMBRE" not-null="true" />
  </many-to-many>
</set>

Usuario.hbm.xml
<set name="maquinas" table="MAQUINAUSUARIO" inverse="true" lazy="true">
  <key>
    <column name="NOMBRE" not-null="true" />
  </key>
  <many-to-many class="modelo.dominio.Maquina">
    <column name="CODIGO" not-null="true" />
  </many-to-many>
</set>
```

Asociación N:M (Many Usuario – Many Persona)

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();  
session.beginTransaction();
```

```
Usuario u1 = new Usuario();  
u1.setNombre("Pepe");  
u1.setPassword("125");  
u1.setTipo("estudiante");  
Usuario u2 = new Usuario();  
u2.setNombre("Nerea");  
u2.setPassword("125");  
u2.setTipo("estudiante");
```

```
mysql> select * from usuario;  
+-----+-----+-----+  
| NOMBRE | PASSWORD | TIPO |  
+-----+-----+-----+  
| Nerea  | 125      | estudiante |  
| Pepe   | 125      | estudiante |  
+-----+-----+-----+
```

```
Maquina m1 = new Maquina();  
m1.setCodigo(1);  
m1.setNombreMaquina("Casa");  
Maquina m2 = new Maquina();  
m2.setCodigo(2);  
m2.setNombreMaquina("Trabajo");
```

```
mysql> select * from maquina;  
+-----+-----+  
| CODIGO | NOMBREMAQUINA |  
+-----+-----+  
| 1      | Casa           |  
| 2      | Trabajo        |  
+-----+-----+
```

```
Set<Maquina> sm=new HashSet<Maquina>();  
sm.add(m1);  
sm.add(m2);
```

```
Set<Usuario> su=new HashSet<Usuario>();  
su.add(u1);  
su.add(u2);
```

```
mysql> select * from maquinausuario;  
+-----+-----+  
| NOMBRE | CODIGO |  
+-----+-----+  
| Nerea  | 1      |  
| Pepe   | 1      |  
| Nerea  | 2      |  
| Pepe   | 2      |  
+-----+-----+
```

```
//u1.setMaquinas(sm); // No necesario cuando <set name="maquinas" table="MAQUINAUSUARIO" inverse="true" EN Usuario.hbm.xml  
//u2.setMaquinas(sm); // No necesario cuando <set name="maquinas" table="MAQUINAUSUARIO" inverse="true" EN Usuario.hbm.xml
```

```
m1.setUsuarios(su);  
m2.setUsuarios(su);  
session.save(u1);  
session.save(u2);  
session.save(m1);  
session.save(m2);  
session.getTransaction().commit();
```

Hay que hacer `maquina.setUsuarios(usuarios)` y no `usuario.setMaquinas(maquinas)`. Con `inverse="true"` en el atributo "maquinas" de usuario, es la clase "Maquina" quien actualiza la asociación

Referencias

- <http://www.hibernate.org/>
- <http://courses.coreservlets.com/Course-Materials/hibernate.html>
- Las siguientes en castellano:
- <http://www.cursohibernate.es/>
- https://docs.jboss.org/hibernate/orm/3.6/reference/es-ES/pdf/hibernate_reference.pdf
- https://www.slideshare.net/Emmerson_Miranda/hibernate-32-short-manual-9367150