0ª Emisión

DIPLOMADO

Desarrollo de Sistemas con Tecnología Java

Módulo 4

Persistencia con Hibernate

Ing. Jorge Alberto Montalvo Olvera

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
Dirección General de Cómputo y de Tecnologías de información y Comunicación
Dirección de Docencia en TIC

DGTIC

50 AÑOS Educación Continua 1971 - 2021

TEMARIO

- Introducción
  - Persistencia de Datos
  - ORM (Object Relational Mapping)

- Hibernate
  - Configuración
  - Mapeo de Clases Persistentes
  - Tipos de Datos
  - Mapeo de Colecciones
  - Transacciones
  - Tipo de relaciones
  - Claves primarias

TEMARIO

- Hibernate Query
- Hibernate Query Language

Persistencia de Datos

- Almacenamiento de la información después de finalizar un programa.
- La información se necesita preservar para su posterior uso.

- La mayoría de sistemas hace uso de bases de datos relacionales.
- Mapeo entre bases de datos y objetos de aplicación.

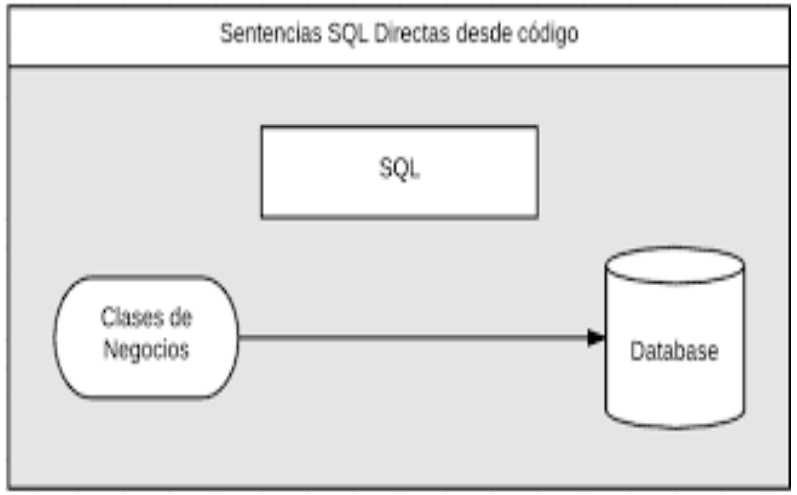
JDBC

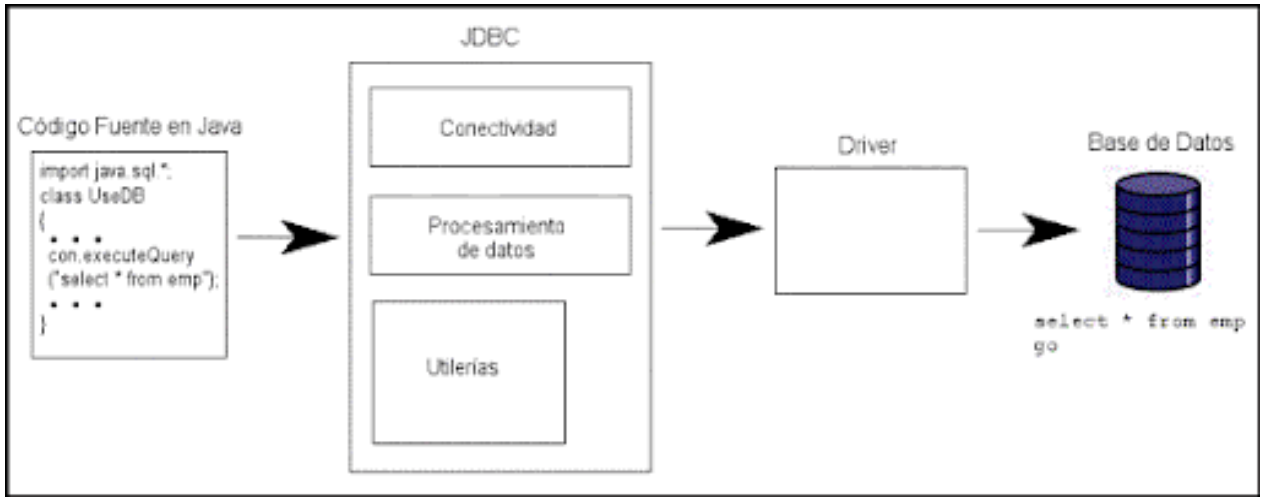
- API del lenguaje JAVA que permite la ejecución de operaciones sobre bases de datos.
- Independiente del sistema operativo y de la base de datos.

- Utiliza el dialecto SQL del modelo de base de datos.

Conexión JDBC

- La manera tradicional de acceder sería a través de JDBC directamente conectando a la BD mediante ejecución de sentencias SQL.





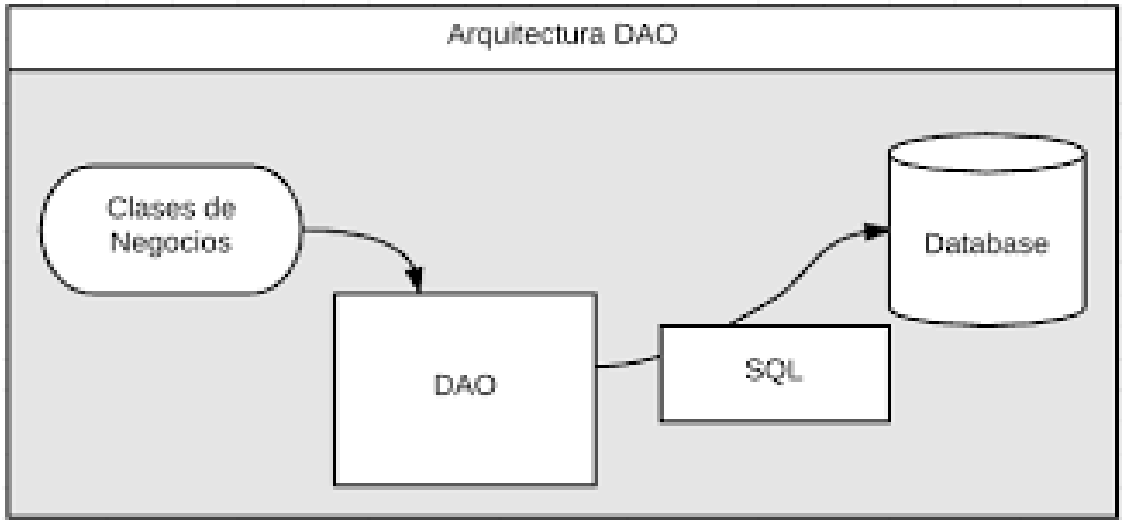
Desventaja

- Esta primer aproximación puede ser útil para proyectos o arquitecturas con pocas clases de negocio, ya que el mantenimiento del código esta altamente ligado a los cambios en el modelo de datos relacional de la BD.

- Un mínimo cambio implica la revisión de casi todo el código así como de su compilación y nueva instalación en el cliente.

Modelo DAO

- Una aproximación más avanzada sería la creación de unas clases de acceso a datos (DAO Data Access Object).



Desventaja

- Los problemas de esta implementación siguen siendo el mantenimiento de la mismo así como su portabilidad. Lo único que podemos decir es que tenemos el código de transacciones encapsulado en las clases DAO.

Problemas Generales

- Mapeo de objetos a base de datos relacional y viceversa.

- Asociaciones, herencia y polimorfismo.

Lo Deseado

- Lo que parece claro es que debemos separar el código de nuestras clases de negocio de la realización de nuestras sentencias SQL contra la Base de Datos.

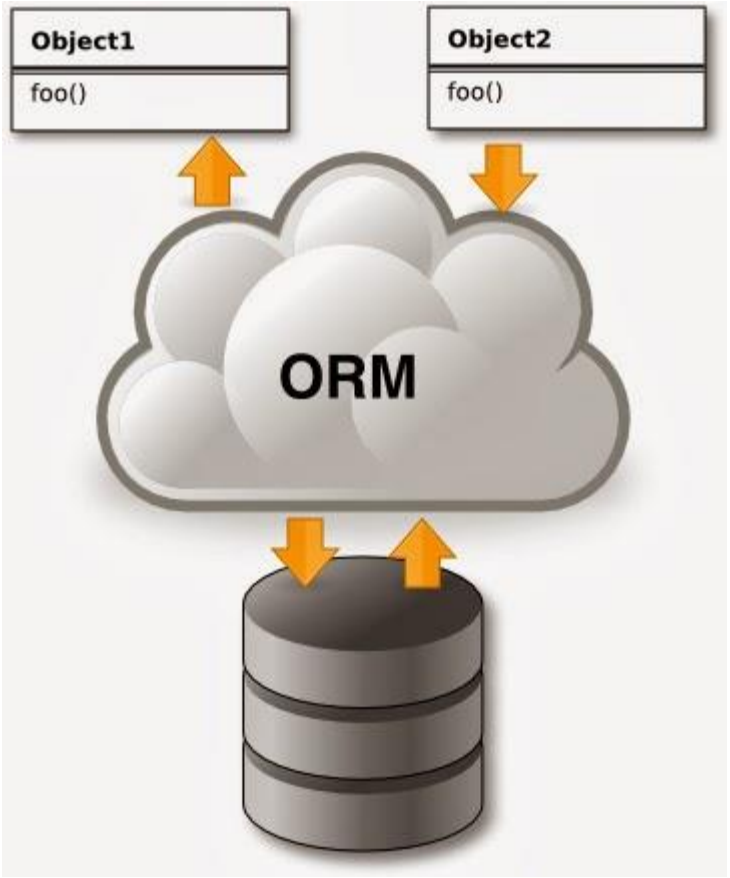
ORM Object Relational Mapping

- Mapeo objeto-relacional (object-relational mapping, o/rm, orm, u o/r mapping).

- Técnica de programación para convertir datos entre un lenguaje de programación OO utilizado para una BD relacional.
- Utiliza un motor de persistencia.
- En la práctica crea una base de datos virtual orientada a objetos.

Objetivos de un ORM

- Garantizar la persistencia de objetos.
  - Conexiones JDBC + consultas SQL
- Eliminar problemas
  - Objetos con muchas propiedades
  - Claves foráneas
  - Restricciones de integridad



¿Qué ofrece un ORM?

- Definir el mapeo en un solo punto.
- Persistencia directa de objectos.
- Carga automática de objetos.
- Buen lenguaje de consultas (HQL, JPQL).

```
Query q = session.createQuery("from Item")
                 .setReadOnly(true);

Criteria criteria = session.createCriteria(Item.class)
                           .setReadOnly(true);

Query q = em.createQuery("select i from Item i")
            .setHint("org.hibernate.readOnly", true);
```

HIBERNATE

- Hibernate comenzó en el año 2001 por Gavin King con un equipo de Cirrus Technologies, como una alternativa al estilo de programación de EJB2, que usa beans como entidades. El objetivo original era ofrecer mejores capacidades de persistencia que las ofrecidas por EJB2, simplificar las complejidades, y complementar algunas características necesarias.

¿Que Ofrece Hibernate?

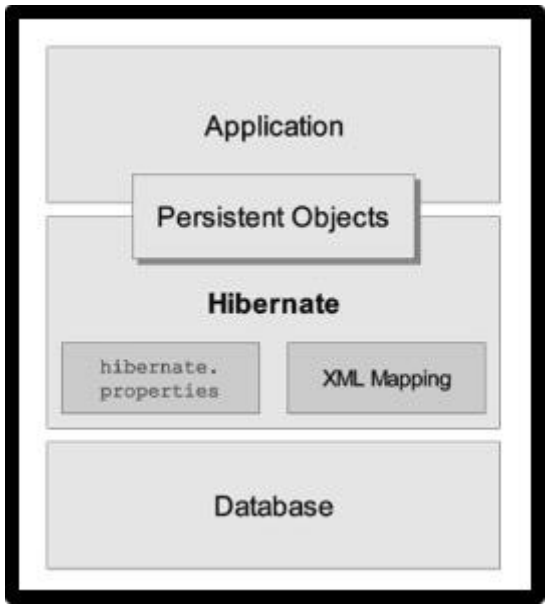
- Es una capa de persistencia objeto-relacional y generador de sentencias SQL.
- Facilita el mapeo entre BD relacional y modelo de objetos.
- Funciona mediante archivos declarativos (xml) y anotaciones.
- Lenguaje de consulta de datos HQL (Hibernate Query Languaje).

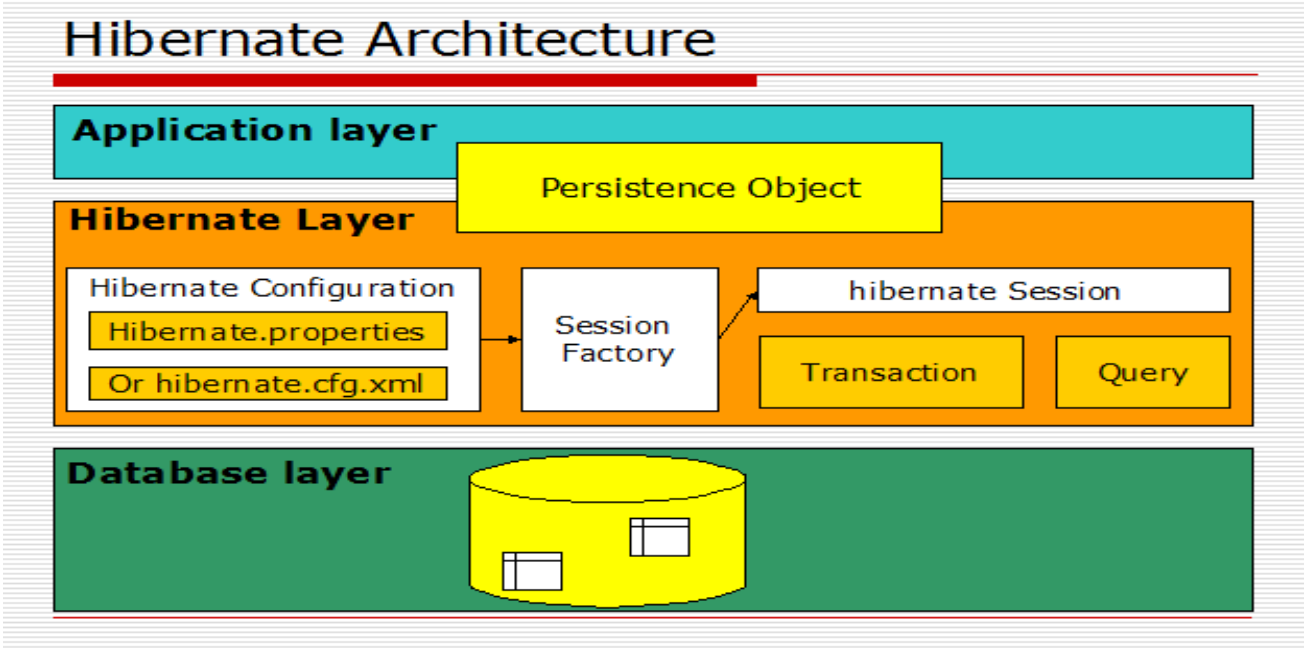
¿Porqué Hibernate?

- Es Free Software.
- Buena documentación y estabilidad.
- Utiliza clases de forma directa.
- No depende de ningún manejador de base de datos.

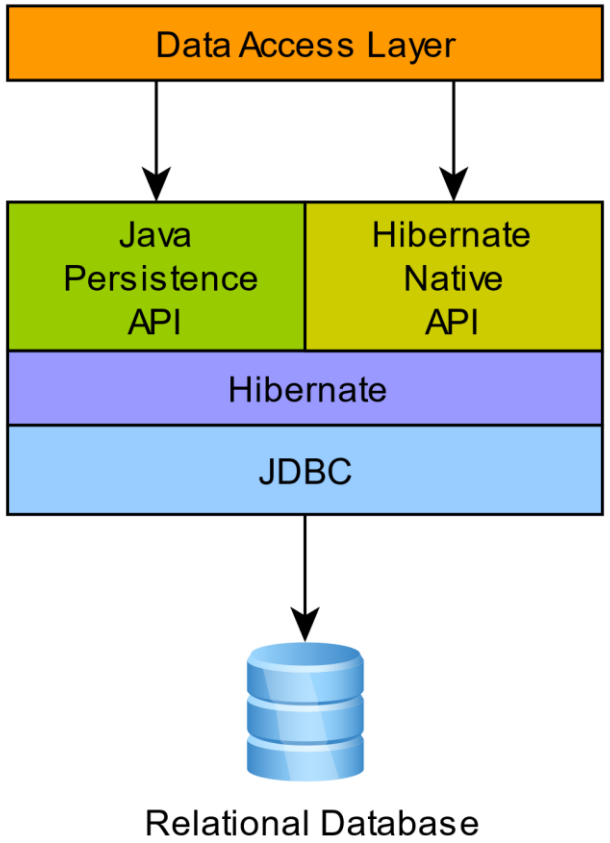
Arquitectura

- Hibernate se integra en cualquier tipo de aplicación justo por encima del contenedor de datos.





Arquitectura



Configuración

- La configuración de hibernate se puede realizar mediante el archivo hibernate.cfg.xml
- Dentro de la configuración se encuentra el mapeo de las tablas de BD y las clases mediante archivos hbm.xml
- La integración con la infraestructura java es con datasource, jndi, jta.

hibernate.cfg-xml

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <!-- JDBC Database connection settings -->
        <property name="connection.driver_class">org.mariadb.jdbc.Driver</property>
        <property name="connection.url">jdbc:mariadb://localhost:3306/db</property>
        <property name="connection.username">root</property>
        <property name="connection.password">root</property>
        <!-- JDBC connection pool settings ... using built-in test pool -->
        <property name="connection.pool_size">1</property>
        <!-- Select our SQL dialect -->
        <property name="dialect">org.hibernate.dialect.MariaDB103Dialect</property>
        <!-- Echo the SQL to stdout -->
        <property name="show_sql">true</property>
        <!-- Set the current session context -->
        <property name="current_session_context_class">thread</property>
        <!-- Drop and re-create the database schema on startup -->
        <property name="hbm2ddl.auto">create-drop</property>
        <!-- dbcp connection pool configuration -->
        <property name="hibernate.dbcp.initialSize">5</property>
        <property name="hibernate.dbcp.maxTotal">20</property>
        <property name="hibernate.dbcp.maxIdle">10</property>
        <property name="hibernate.dbcp.minIdle">5</property>
        <property name="hibernate.dbcp.maxWaitMillis">-1</property>
        <mapping class="mx.unam.dgtic.diplomado.Entidad" />
    </session-factory>
</hibernate-configuration>
```

Configuración JDBC

- Propiedades JDBC
  - connection.driver_class
  - connection.url
  - connection.username
  - connection.password
  - connection.pool_size

Dialecto Hibernate

- Dialect es una clase y un puente entre los tipos de Java JDBC y los tipos de SQL, que contiene el mapeo entre el tipo de datos del lenguaje Java y el tipo de datos de la base de datos. Dialect permite a Hibernate generar SQL optimizado para una base de datos relacional en particular. Hibernate genera consultas para la base de datos específica basada en la clase Dialect.
  - org.hibernate.dialect.MariaDB103Dialect

Mapeo de Archivo hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
"http://hibernate.sourceforge.net/hibernate-mapping.dtd">
<hibernate-mapping>
        <class name="mx.unam.dgtic.diplomado.User" table="users">
                <id name="id" column="id" type="integer">
                        <generator class="assigned" />
                </id>
                <property name="userName" column="Name" type="string" />
                <property name="password" type="string" />
                <property name="emailAddress" type="string" />
                <property name="lastLogon" type="date" />
        </class>
</hibernate-mapping>
```

Mapeo de Archivo hbm.xml

- Declaración de la DTD
  - El documento DTD que se usa en los archivos hbm.xml viene en la distribución de hibernate (jar)
- El elemento raíz es <hibernate-mapping>, se declara el elemento <class>.
- Elemento <class> declara la clase persistente. Una clase persistente equivale a una tabla de base de datos
- Elemento <id> permite definir el identificador del objeto, corresponde a la clave principal de la tabla de base de datos

Mapeo de Archivo hbm.xml

- Elemento <generator> define que clase se utilizará para generar los identificadores.
- Elemento <property> declara una propiedad persistente de la clase que corresponde a una columna.
- Tipos de relaciones (componentes y relaciones de objetos)
  - 1 – 1, 1 – N, N – M, N – 1
  - N – M y 1 – N son colecciones
  - 1 – 1 y N – 1 son componentes

Opciones de Mapeo de una Clase

<hibernate-mapping>

```
<hibernate-mapping

schema="schemaName"
catalog="catalogName"
default-cascade="cascade_style"
default-
access="field|property|ClassName"
default-lazy="true|false"
auto-import="true|false"
package="package.name"

/>
```

Opciones de Mapeo de una Clase

<class> attributes

```
<class

name="ClassName"
table="tableName"
discriminator-
value="discriminator_value"
mutable="true|false"
schema="owner"
polymorphism="implicit|explicit"
where="arbitrary sql where
condition"
persister="PersisterClass" ...

/>
```

<id> attributes

```
<id

name="propertyName"
type="typename"
column="column_name"
unsaved-
value="null|undefined|id_value"
access="field|property|ClassName"
>
<generator
class="generatorClass"/>
/>
```

Opciones de Mapeo de una Clase

<property> attributes

```
<property

name="propertyName"
column="column_name"
type="typename"
update="true|false"
insert="true|false"
formula="arbitrary SQL
expression"
access="field|property|ClassName"
...

/>
```

<generator> attributes

```
<id name="id" type="long" column="cat_id">
        <generator class="org.hibernate.id.TableHiLoGenerator">
                <param name="table">uid_table</param>
                <param name="column">next_hi_value_column</param>
        </generator>
</id>
```

Opciones de Mapeo de una Clase

<many-to-one>

```
<many-to-one

name="propertyName"
column="column_name"
class="ClassName"
cascade="cascade_style"
fetch="join|select"
update="true|false"
insert="true|false"
...

/>
```

<one-to-one> attributes

```
<one-to-one

name="propertyName"
class="ClassName"
cascade="cascade_style"
constrained="true|false"
fetch="join|select"
access="field|property|ClassName"

/>
```

POJO (Plain Old Java Object)

- POJO son las iniciales de "Plain Old Java Object". Un POJO es una instancia de una clase que no extiende ni implementa nada en especial. Para los programadores Java sirve para enfatizar el uso de clases simples y que no dependen de un framework en especial.

```java
public class ClienteBean implements java.io.Serializable {
   private String nombre;
   private int edad;

   public ClienteBean() {
      // Constructor
   }

   public String getNombre() {
     return nombre;
   }

   public void setNombre(String nombre) {
     this.nombre = nombre;
   }

   public int getEdad() {
     return edad;
   }

   public void setEdad(int edad) {
     this.edad = edad;
   }
}
```

Configuración de API

- SessionFactory
  - Es una fabrica de sesiones. Un objeto configuration es capaz de crear una sessionfactory ya que tiene toda la informacion necesaria. Normalmente, una aplicación solo tiene una sessionfactory.
  - Apartir del hibernate.cfg.xml podemos crear una sessionfactory.

```
if (sessionFactory == null) {
    try {
        // Create registry
        registry = new StandardServiceRegistryBuilder().configure().build();

        // Create MetadataSources
        MetadataSources sources = new MetadataSources(registry);

        // Create Metadata
        Metadata metadata = sources.getMetadataBuilder().build();

        // Create SessionFactory
        sessionFactory = metadata.getSessionFactoryBuilder().build();

    } catch (Exception e) {
        e.printStackTrace();
        if (registry != null) {
            StandardServiceRegistryBuilder.destroy(registry);
        }
    }
}
return sessionFactory;
```

Configuración de API

- Session
  - La principal interfaz entre la aplicación java y hibernate. Es la que mantiene las conversaciones entre la aplicación y la base de datos. Permite añadir, modificar y borrar objetos en la base de datos.

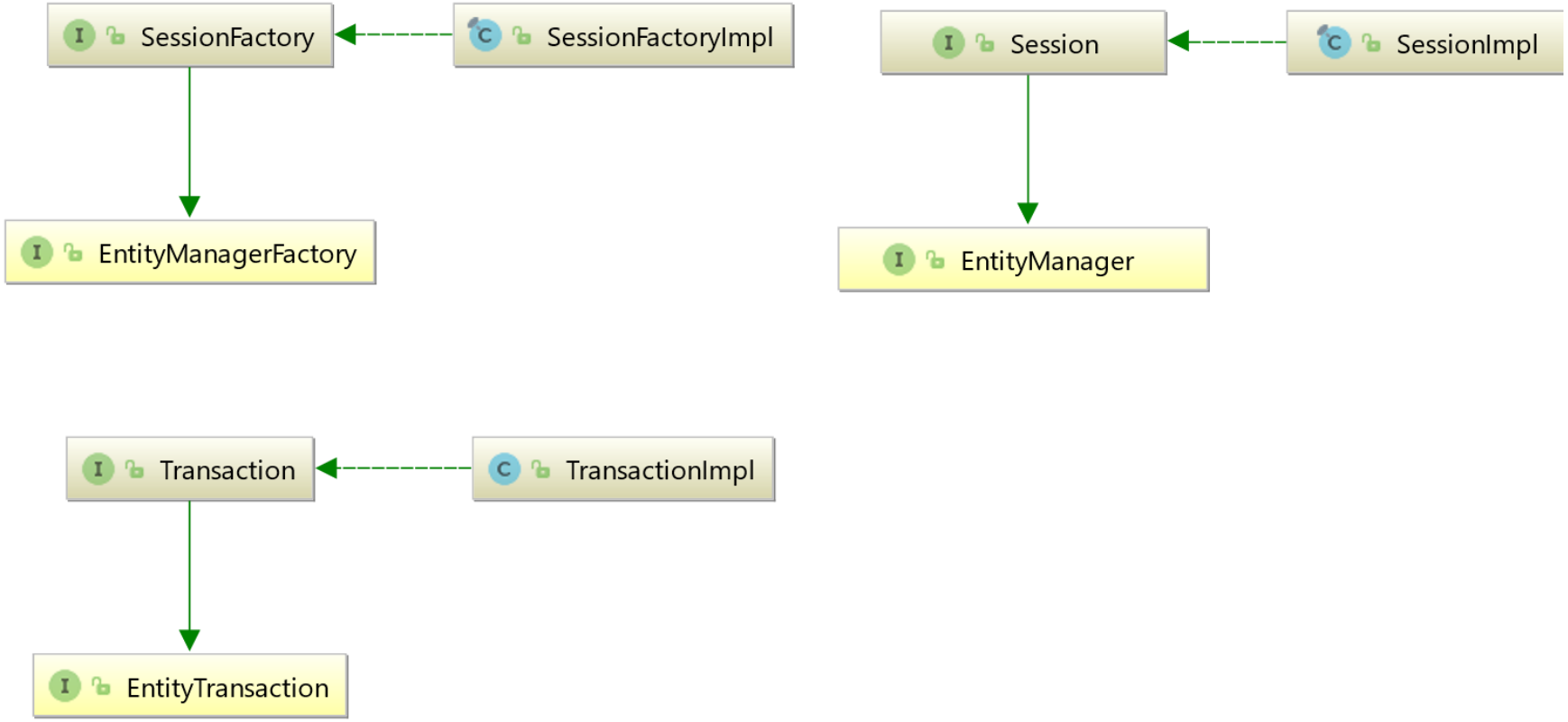
```
try (Session session = HibernateUtil.getSessionFactory().openSession()) {
    // start a transaction
    transaction = session.beginTransaction();
    // save the student objects
    session.save(student);
    session.save(student1);
    // commit transaction
    transaction.commit();
```

Configuración de API

- Transaction
  - Se encarga de las transacciones hacia la base de datos
    - Commit
    - Rollback

```
try (Session session = HibernateUtil.getSessionFactory().openSession()) {
    // start a transaction
    transaction = session.beginTransaction();
    // save the student objects
    session.save(student);
    session.save(student1);
    // commit transaction
    transaction.commit();
```

Configuración de API



API HIBERNATE

- Hibernate provee de diferentes métodos para la pesistencia de datos.
  - save
  - delete
  - get
  - load
- Para las búsquedas en la base de datos se cuenta con HQL (Hibernate Query Language).

Mapeo de Clases

- Dentro del mapeo de clases persistentes con hibernate encontramos la compatibilidad de tipos de datos. Hibernate maneja los siguientes tipos de datos para la correspondencia con el sgdb.

Hibernate type (org.hibernate.type package)	JDBC type	Java type	BasicTypeRegistry key(s)
StringType	VARCHAR	java.lang.String	string, java.lang.String
MaterializedClob	CLOB	java.lang.String	materialized_clob
TextType	LONGVARCHAR	java.lang.String	text
CharacterType	CHAR	char, java.lang.Character	character, char, java.lang.Character
BooleanType	BOOLEAN	boolean, java.lang.Boolean	boolean, java.lang.Boolean
NumericBooleanType	INTEGER, 0 is false, 1 is true	boolean, java.lang.Boolean	numeric_boolean
YesNoType	CHAR, 'N'/'n' is false, 'Y'/'y' is true. The uppercase value is written to the database.	boolean, java.lang.Boolean	yes_no

Mapeo de Clases

ByteType	TINYINT	byte, java.lang.Byte	byte, java.lang.Byte
ShortType	SMALLINT	short, java.lang.Short	short, java.lang.Short
IntegerType	INTEGER	int, java.lang.Integer	integer, int, java.lang.Integer
LongType	BIGINT	long, java.lang.Long	long, java.lang.Long
FloatType	FLOAT	float, java.lang.Float	float, java.lang.Float
DoubleType	DOUBLE	double, java.lang.Double	double, java.lang.Double
BigIntegerType	NUMERIC	java.math.BigInteger	big_integer, java.math.BigInteger
BigDecimalType	NUMERIC	java.math.BigDecimal	big_decimal, java.math.bigDecimal
TimestampType	TIMESTAMP	java.util.Date	timestamp, java.sql.Timestamp, java.util.Date
DbTimestampType	TIMESTAMP	java.util.Date	dbtimestamp
TimeType	TIME	java.util.Date	time, java.sql.Time
DateType	DATE	java.util.Date	date, java.sql.Date
CalendarType	TIMESTAMP	java.util.Calendar	calendar, java.util.Calendar, java.util.GregorianCalendar

Colecciones

- Hibernate también permite persistir colecciones. Estas colecciones persistentes pueden contener casi cualquier tipo de dato Hibernate, incluyendo: tipos básicos, los tipos personalizados, componentes y referencias a otras entidades.
- La diferencia entre el valor y la semántica de referencia es en este contexto muy importante. Un objeto en una colección puede ser manejado con la semántica de "valor" (su vida depende totalmente en el propietario de la colección).

Colecciones

- Las colecciones persistentes inyectadas por Hibernate se comportan como HashMap, HashSet, TreeMap, TreeSet o ArrayList, dependiendo del tipo de interfaz.

Colecciones

- Las instancias de colecciones tienen el comportamiento usual de los tipos de valor. Son automáticamente persistidas al ser referenciadas por un objeto persistente y se borran automáticamente al perder la referencia. Si una colección se pasa de un objeto persistente a otro, puede que sus elementos se muevan de una tabla a otra. Dos entidades no pueden compartir una referencia a la misma instancia de colección. Debido al modelo relacional subyacente, las propiedades valuadas en colección no soportan la semántica de valor nulo.
- Hibernate no distingue entre una referencia de colección nula y una colección vacía.

Colecciones

- Las colecciones son declaradas utilizando <set>, <list>, <map>, <bag>, <array>.

Transacción

- Una transacción es un conjunto de operaciones de datos que se realizan como una única unidad lógica de trabajo, que deben tener éxito o fallar por completo para garantizar la consistencia e integridad de los datos.

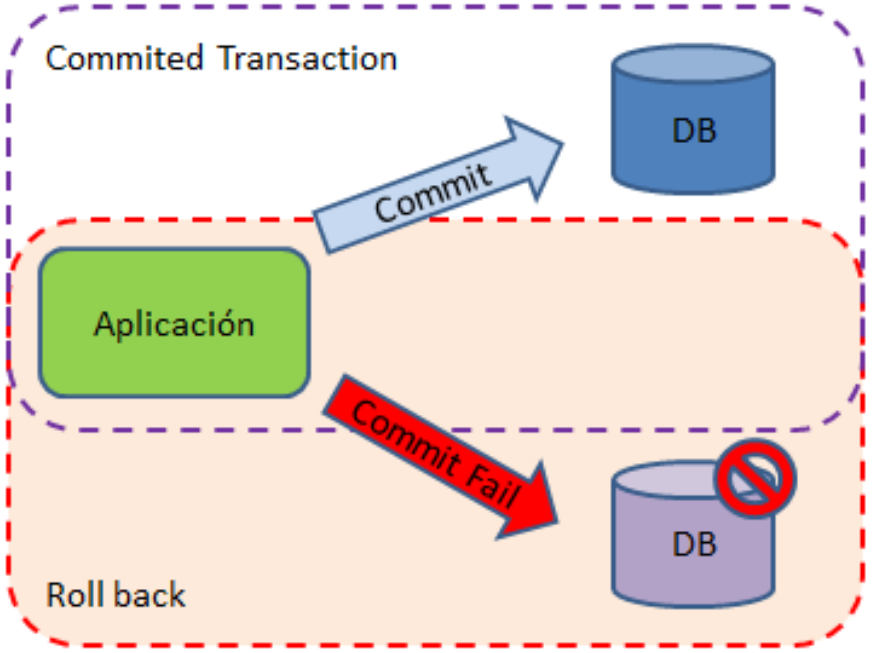
Transacción

- La transacción tiene propiedades ACID:
  - Atómica: una transacción consiste en una o más acciones unidas como si fuera una sola unidad de trabajo. Atomicity asegura que todas las operaciones en una transacción ocurran o no ocurran.
  - Consistencia (consistente): una vez que se completa una transacción (independientemente del éxito o el fracaso), el estado del sistema y sus reglas comerciales son consistentes. Es decir, los datos no deben destruirse.

Transacción

– Aislado: la transacción debe permitir que varios usuarios operen con los mismos datos, y las operaciones de un usuario no se confundirán con las operaciones de otros usuarios.
– Durable: una vez que se completa la transacción, el resultado de la transacción debe ser persistente.

Transacciones



Transacciones JDBC

- En la API JDBC, la clase java.sql.Connection representa una conexión de base de datos. Proporciona los siguientes métodos para controlar las transacciones:
  - setAutoCommit (Boolean autoCommit): establece si se deben confirmar transacciones automáticamente.
  - commit (): confirma la transacción.
  - rollback (): deshace la transacción.

Transacciones JDBC

```
Connection = null;
PreparedStatement pstmt = null;
try{
con = DriverManager.getConnection(dbUrl, username, password);
// Establecer el modo de enviar transacciones manualmente
con.setAutoCommit(false);
pstmt = ……;
pstmt.executeUpdate();
// transacción de confirmación
con.commit();
}catch(Exception e){
// Deshacer transacciones
con.rollback();
…..
} finally{
    ……
}
```

Transacciones en Hibernate

- Hibernate implementa una ligera encapsulación de objetos de JDBC. Hibernate en sí no tiene funciones de procesamiento de transacciones en el momento del diseño. Las transacciones habituales de Hibernate solo encapsulan JDBCTransaction o JTATransaction subyacentes, y colocan Transaction and Session en el exterior. De hecho, la capa inferior es implementar la función de programación de la transacción delegando el JDBC o JTA subyacente.

Transacciones en Hibernate

- Hibernate usará transacciones JDBC por defecto.
- Utiliza el procesamiento de transacciones JDBC de la siguiente manera:

```
Transaction tx = null;
try {
    tx = sess.beginTransaction();

    // do some work
    ...

    tx.commit();
}
catch (RuntimeException e) {
    if (tx != null) tx.rollback();
    throw e; // or display error message
}
finally {
    sess.close();
}
```

Tipo de Relaciones

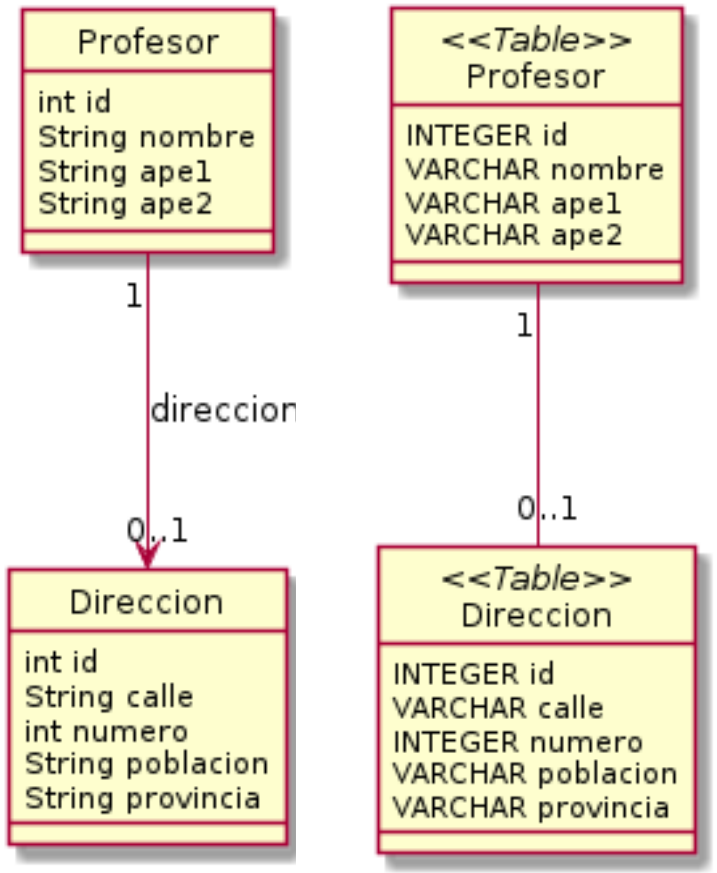
- Uno a uno (unidireccional)
- Uno a uno (bidireccional)
- Uno a muchos (desordenada)
- Uno a muchos (ordenada)
- Muchos a muchos
- Cascade.

Uno a Uno (Unidireccional)

- La relación uno a uno en Hibernate consiste simplemente en que un objeto tenga una referencia a otro objeto de forma que al persistirse el primer objeto también se persista el segundo.

- La relación va a ser unidireccional es decir que la relación uno a uno va a ser en un único sentido.
- Las dos tablas deben tener la misma clave primaria y de esa forma se establece la relación.

Uno a Uno (Unidireccional)




```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/H
<hibernate-mapping>
  <class name="ejemplo01.Profesor" >
    <id column="Id" name="id" type="integer"/>
    <property name="nombre" />
    <property name="ape1" />
    <property name="ape2" />

    <one-to-one name="direccion" cascade="all" />

  </class>
</hibernate-mapping>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hiberna
<hibernate-mapping>
  <class name="ejemplo01.Direccion" >
    <id column="Id" name="id" type="integer"/>
    <property name="calle"/>
    <property name="numero"/>
    <property name="poblacion"/>
    <property name="provincia"/>

  </class>
</hibernate-mapping>
```

Uno a Uno (Unidireccional)

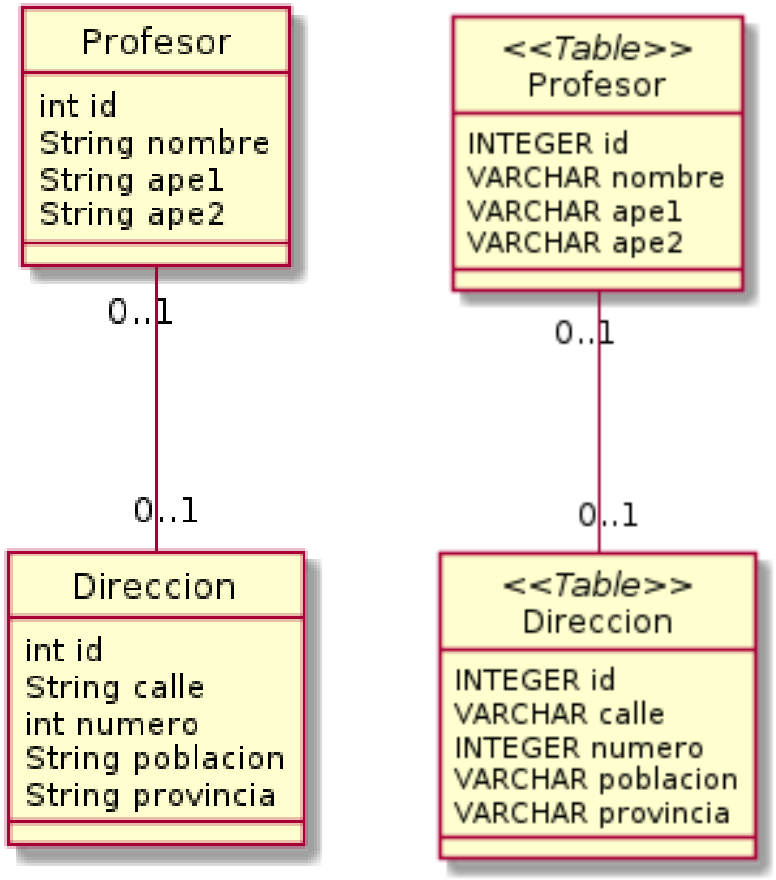
- El tag <one-to-one> se utiliza para definir una relación uno a uno entre las dos clases Java. En su forma más sencilla contiene sólamente dos atributos:
  - name:Este atributo contiene el nombre de la propiedad Java con la referencia al otro objeto con el que forma la relación uno a uno.
  - cascade: Este atributo indicará a hibernate cómo debe actuar cuando realicemos las operaciones de persistencia de guardar, borrar, leer, etc.

Uno a Uno (Bidireccional)

- Esta relación es muy similar a la relación Uno a Uno unidireccional, pero en éste caso la relación va a ser bidireccional.

- Las dos tablas deben tener la misma clave primaria y de esa forma se establece la relación.

Uno a Uno (Bidireccional)



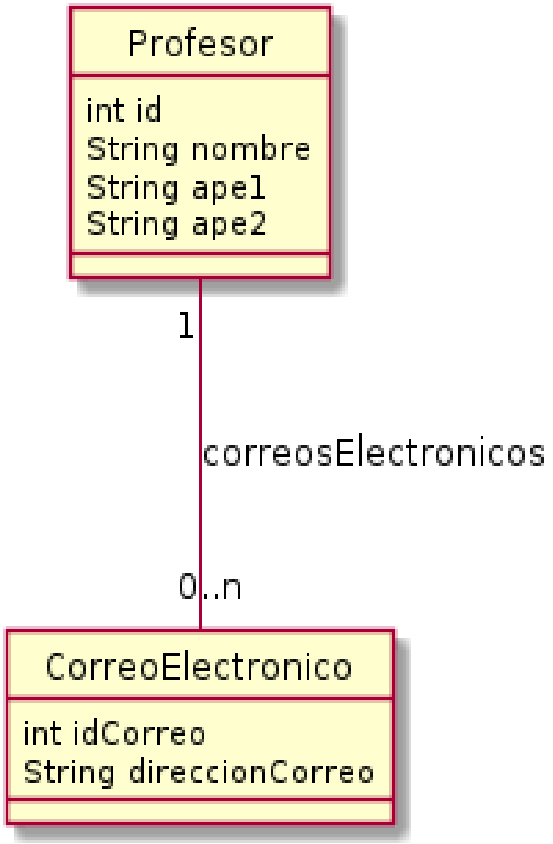

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/H
<hibernate-mapping>
  <class name="ejemplo01.Profesor" >
    <id column="Id" name="id" type="integer"/>
    <property name="nombre" />
    <property name="ape1" />
    <property name="ape2" />

    <one-to-one name="direccion" cascade="all" />

  </class>
</hibernate-mapping>
```

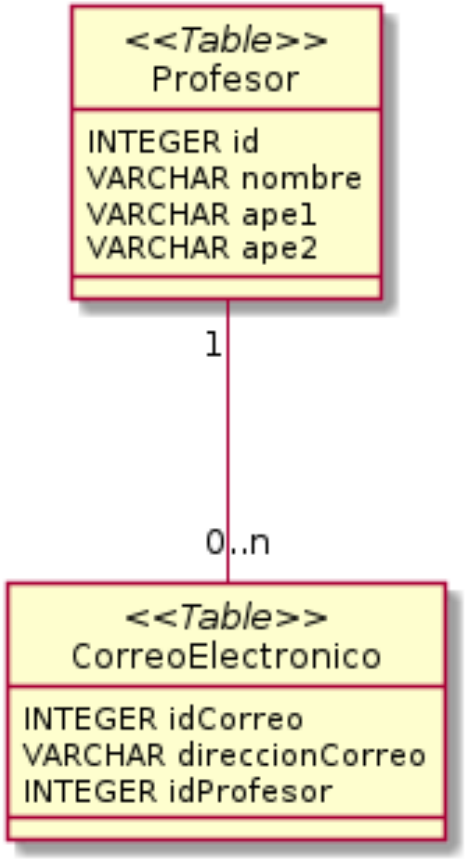
```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/H
<hibernate-mapping>
  <class name="ejemplo01.Direccion" >
    <id column="Id" name="id" type="integer"/>
    <property name="calle"/>
    <property name="numero"/>
    <property name="poblacion"/>
    <property name="provincia"/>

    <one-to-one name="profesor" cascade="all" />

  </class>
</hibernate-mapping>
```

Uno a muchos (desordenada)

- La relación uno a muchos consiste simplemente en que un objeto padre tenga una lista sin ordenar de otros objetos hijo de forma que al persistirse el objeto principal también se persista la lista de objetos hijo. Esta relación también suele llamarse maestro-detalle o padre-hijo.
- En la tabla de origen contiene como clave ajena la clave primaria de la tabla destino y de esa forma se establece la relación uno a muchos.

Uno a muchos (desordenada)





Uno a muchos (desordenada)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTI
<hibernate-mapping>
  <class name="ejemplo05.Profesor" >
    <id column="Id" name="id" type="integer"/>
    <property name="nombre" />
    <property name="ape1" />
    <property name="ape2" />

    <set name="correosElectronicos"  cascade="all" inverse="true"  >
        <key>
            <column name="idProfesor"  />
        </key>
        <one-to-many class="ejemplo05.CorreoElectronico" />
    </set>
  </class>
</hibernate-mapping>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate
<hibernate-mapping>
  <class name="ejemplo05.CorreoElectronico" >
    <id column="IdCorreo" name="idCorreo" type="integer"/>
    <property name="direccionCorreo" />


    <many-to-one name="profesor">
            <column name="idProfesor"  />
    </many-to-one>


  </class>
</hibernate-mapping>
```

Uno a muchos (desordenada)

- El tag <set> se utiliza para definir una relación uno a muchos desordenada entre las dos clases Java.
  - name: Es el nombre de la propiedad Java del tipo Set en la cual se almacenan todos los objetos hijos.
  - cascade: Este atributo indica que se realizan las mismas operaciones con el objeto padre que con los objetos hijos, es decir si uno se borra los otros también, etc. Su valor habitual es all.
  - inverse: Este atributo se utiliza para minimizar las SQLs que lanza Hibernate contra la base de datos. En este caso concreto debe establecerse a true para evitar una sentencia SQL de UPDATE por cada hijo.

Uno a muchos (desordenada)

- Tags anidados
  - key Este tag contiene otro anidado llamado column con el atributo name que indica el nombre de una columna de la base de datos. Esta columna debe ser de la tabla hijo y ser el nombre de la clave ajena a la tabla padre.
  - one-to-many Este tag contiene el atributo class con el FQCN de la clase Java hija.

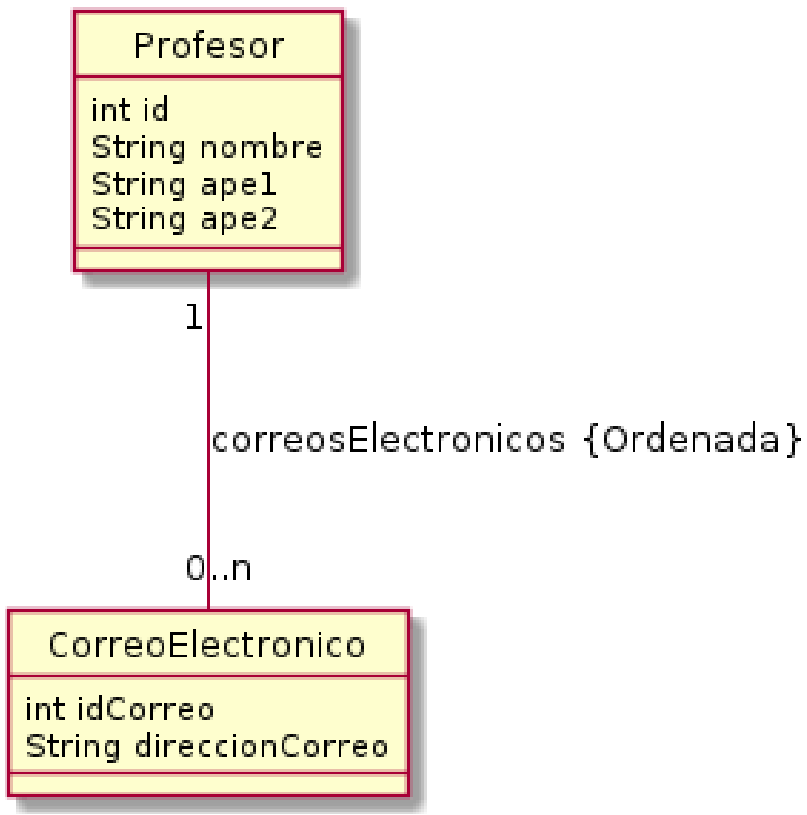
Uno a muchos (desordenada)

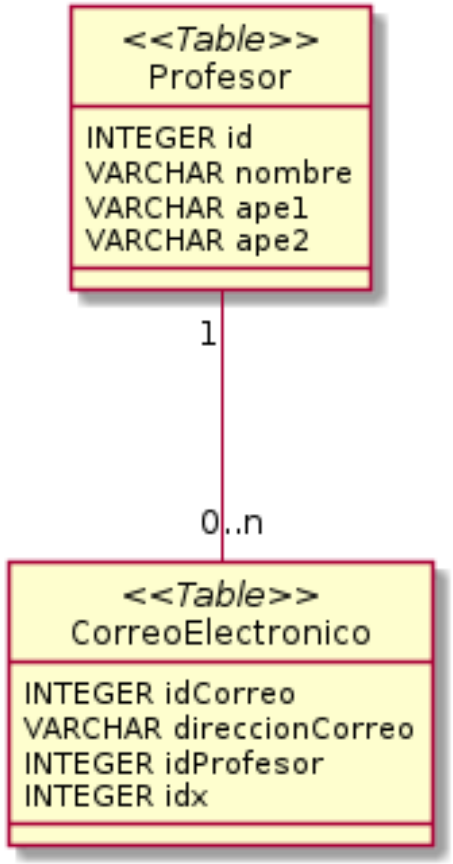
- El tag <many-to-one> se utiliza para definir una relación muchos a uno entre las dos clases Java.
  - name: Es el nombre de la propiedad Java que enlaza con el objeto padre.
  - column Este tag contiene el atributo name que indica el nombre de una columna de la base de datos. Esta columna debe ser de la tabla hijo y ser el nombre de la clave ajena a la tabla padre.

Uno a muchos (ordenada)

- La relación uno a muchos consiste simplemente en que un objeto padre tenga una lista ordenada de otros objetos hijo de forma que al persistirse el objeto principal también se persista la lista de objetos hijo. Esta relación también suele llamarse maestro-detalle o padre-hijo.

Uno a muchos (ordenada)





Uno a muchos (ordenada)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
<hibernate-mapping>
  <class name="ejemplo05.Profesor" >
    <id column="Id" name="id" type="integer"/>
    <property name="nombre" />
    <property name="ape1" />
    <property name="ape2" />

    <list name="correosElectronicos"  cascade="all" inverse="false"  >
        <key>
            <column name="idProfesor"  />
        </key>
        <list-index>
            <column name="Idx"  />
        </list-index>
        <one-to-many class="ejemplo07.CorreoElectronico" />
    </list>
  </class>
</hibernate-mapping>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate
<hibernate-mapping>
  <class name="ejemplo05.CorreoElectronico" >
    <id column="IdCorreo" name="idCorreo" type="integer"/>
    <property name="direccionCorreo" />


    <many-to-one name="profesor">
            <column name="idProfesor"  />
    </many-to-one>


  </class>
</hibernate-mapping>
```

Uno a muchos (ordenada)

- El tag <list> se utiliza para definir una relación uno a muchos entre las dos clases Java en las cuales hay un orden.
  - name: Es el nombre de la propiedad Java del tipo List en la cual se almacenan todos los objetos hijos.
  - cascade:Este atributo indica que se realizan las mismas operaciones con el objeto padre que con los objetos hijos, es decir si un se borra los otros también, etc. Su valor habitual es all.
  - inverse: En este caso el atributo inverse debe tener el valor false ya que de esa forma se guardará en valor del orden de cada objeto. Si se estableciera a true no se guardaría el valor.

Uno a muchos (ordenada)

- Tags anidados
  - key Este tag contiene otro anidado llamado column con el atributo name que indica el nombre de una columna de la base de datos. Esta columna debe ser de la tabla hijo y ser el nombre de la clave ajena a la tabla padre.
  - list-index Este tag contiene otro anidado llamado column con el atributo name que indica el nombre de una columna de la base de datos. Esta columna debe ser de la tabla hijo y ser la columna donde se guarda el orden que ocupa dentro de la lista.
  - one-to-many Este tag contiene el atributo class con el FQCN de la clase Java hija.

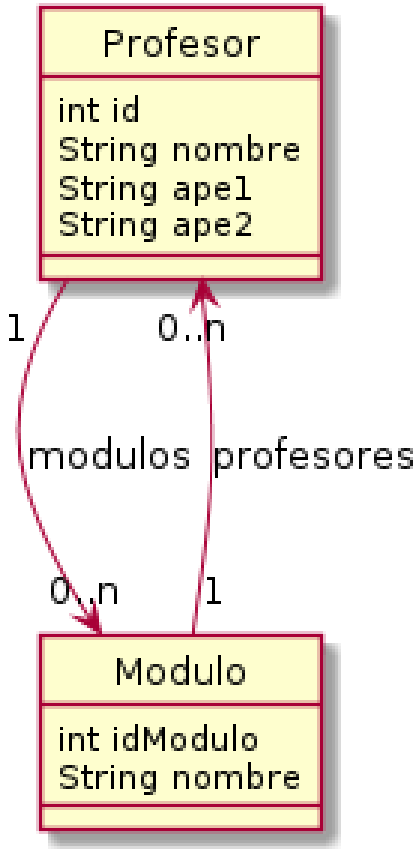
Uno a muchos (ordenada)

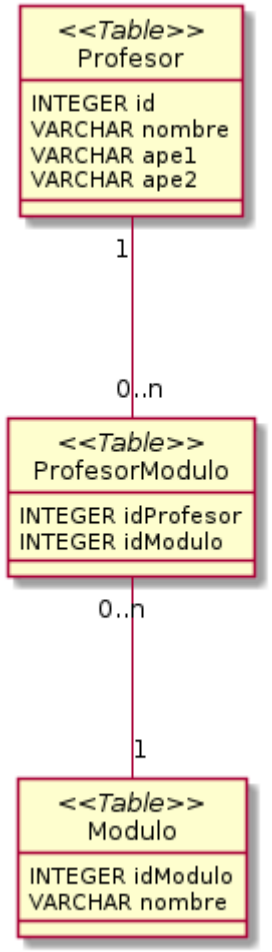
- El tag <many-to-one> se utiliza para definir una relación muchos a uno entre las dos clases Java.
  - name: Es el nombre de la propiedad Java que enlaza con el objeto padre.
  - column Este tag contiene el atributo name que indica el nombre de una columna de la base de datos. Esta columna debe ser de la tabla hijo y ser el nombre de la clave ajena a la tabla padre.

Muchos a muchos

- La relación muchos a muchos consiste en que un objeto A tenga una lista de otros objetos B y también que el objeto B a su vez tenga la lista de objetos A. De forma que al persistirse cualquier objeto también se persista la lista de objetos que posee.

Muchos a muchos





Muchos a muchos

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
<hibernate-mapping>
  <class name="ejemplo09.Profesor" >
    <id column="Id" name="id" type="integer"/>
    <property name="nombre" />
    <property name="ape1" />
    <property name="ape2" />

    <set name="modulos" table="ProfesorModulo"  cascade="all" inverse="true"  >
        <key>
            <column name="idProfesor"  />
        </key>
        <many-to-many column="IdModulo" class="ejemplo09.Modulo" />
    </set>
  </class>
</hibernate-mapping>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "htt
<hibernate-mapping>
  <class name="ejemplo09.Modulo" >
    <id column="IdModulo" name="idModulo" type="integer"/>
    <property name="nombre" />

    <set name="profesores" table="ProfesorModulo"  cascade="all" inverse="false"  >
        <key>
            <column name="idModulo"  />
        </key>
        <many-to-many column="IdProfesor" class="ejemplo09.Profesor" />
    </set>
  </class>
</hibernate-mapping>
```

Muchos a muchos

- El tag <set> se utiliza para definir una lista desordenada entre las dos clases Java.
  - name: Es el nombre de la propiedad Java del tipo Set en la cual se almacenan todos los objetos relacionados.
  - table: Es el nombre de la tabla de la base de datos que contiene la relación muchos a muchos.
  - cascade: Como ya hemos explicado en anteriores lecciones, este atributo indica que se realizan las mismas operaciones con el objeto principal que con los objetos relacionados, es decir si uno se borra los otros también, etc. Su valor habitual es all.
  - inverse: En el caso de las relaciones muchos a muchos es necesario poner un lado de la relación con el valor true y el otro con el valor false.

Muchos a muchos

- Tags anidados
  - key Este tag contiene otro anidado llamado column con el atributo name, el cual contiene el nombre de una columna de la base de datos. Esta columna debe ser una de la tabla de la relación muchos a muchos y ser el nombre de la columna que contiene la clave ajena de tabla que estamos persistiendo.
  - many-to-many Este tag contiene el atributo class con el FQCN de la clase Java con la que se establece la relación.

Muchos a muchos

- El tag <set> se utiliza para definir una lista desordenada entre las dos clases Java.
  - inverse: En el caso de las relaciones muchos a muchos es necesario poner un lado de la relación con el valor true y el otro con el valor false.
- Si ambos valores son true no se realizará ninguna inserción en la tabla intermedia y si ambos valores son false se realizará dos veces la inserción en dicha tabla dando un error de clave primaria duplicada.
- Es decir, que en este caso inverse controla cuándo se realiza la inserción en la tabla intermedia si bien al guardarla en la tabla de izquierda o en la tabla de derecha.

Cascade

- ¿Cuál es el significado del atributo cascade? El significado es indicar qué debe hacer hibernate con las clases relacionadas cuando realizamos alguna acción con la clase principal.
- Es decir, si borramos la clase principal, ¿debería borrarse la clase relacionada?. La respuesta a ésta y otras preguntas depende de nuestro modelo de clases, por ello existen 11 valores distintos y deberemos elegir entre todos ellos.

Valores Cascade

Valor	Descripción
`none`	No se realiza ninguna acción en los objetos relacionados al hacerlo sobre el principal
`save-update`	Si se inserta o actualiza el objeto principal también se realizará la inserción o actualización en los objetos relacionados.
`delete`	Si se borra el objeto principal también se realizará el borrado en los objetos relacionados.
`evict`	Si se llama al método `Session.evict(Object objeto)` para el objeto principal también se llamará para los objetos relacionados.
`lock`	Si se llama al método `LockRequest.lock(Object objeto)` [10] para el objeto principal también se llamará para los objetos relacionados.
`merge`	Si se llama al método `Session.merge(Object objeto)` con el objeto principal también se llamará para los objetos relacionados.
`refresh`	Si se llama al método `Session.refresh(Object objeto)` para el objeto principal también se llamará para los objetos relacionados.
`replicate`	Si se llama al método `Session.replicate(Object objeto,ReplicationMode replicationMode)` para el objeto principal también se llamará para los objetos relacionados.
`all`	Si se realiza cualquiera de las **anteriores** acciones sobre el objeto principal también se realizará sobre los objetos relacionados.
`delete-orphan`	Este atributo sólo se usa si el objeto relacionado es una colección. Indica que si en la colección del objeto principal eliminamos un elemento , al persistir el objeto principal deberemos borrar de la base de datos el elemento de la colección que habíamos eliminado.
`all-delete-orphan`	Es la unión de los atributos `all` y `delete-orphan` [11]

Claves Primarias

- ¿Qué propiedades debe tener una buena clave primaria?
  - Debe ser única
  - No puede ser null
  - Nunca debe cambiar
  - Debe ser una única columna
  - Debe ser rápida de generar

Claves Primarias

- El tag <generator> se utiliza para indicar que la clave primaria será generada por el propio Hibernate en vez de asignarla directamente el usuario.
  - class:Este atributo indica el método que usará Hibernate para calcular la clave primaria.

Claves Primarias

Valor	Descripción
native	Hibernate usará alguno de los siguientes métodos dependiendo de la base de datos. De esta forma si cambiamos de base de datos se seguirá usando la mejor forma de generar la clave primaria
identity	Hibernate usará el valor de la columna de tipo autoincremento. Es decir, que al insertar la fila, la base de datos le asignará el valor. La columna de base de datos debe ser de tipo autonumérico
sequence	Se utiliza una secuencia como las que existen en Oracle o PostgreSQL , no es compatible con MySQL. La columna de base de datos debe ser de tipo numérico
increment	Se lanza una consulta `SELECT MAX()` contra la columna de la base de datos y se obtiene el valor de la última clave primaria, incrementando el nº en 1.La columna de base de datos debe ser de tipo numérico
uuid.hex	Hibernate genera un identificador único como un String. Se usa para generar claves primarias únicas entre distintas bases de datos.La columna de base de datos debe ser de tipo alfanumérico.
guid	Hibernate genera un identificador único como un String pero usando las funciones que provee SQL Server y MySQL. Se usa para generar claves primarias únicas entre distintas bases de datos.La columna de base de datos debe ser de tipo alfanumérico.
foreign	Se usará el valor de otro objeto como la clave primaria. Un uso de ello es en relaciones *uno a uno* donde el segundo objeto debe tener la misma clave primaria que el primer objeto a guardar.

Claves Primarias

- Al usar el método sequence es necesario indicar el nombre de la secuencia que va a usar Hibernate para obtener el valor.

```
<id column="Id" name="id" type="integer">
    <generator class="sequence" >
        <param name="sequence">secuencia_idProfesor</param>
    </generator>
</id>
```

Claves Primarias

- Al usar el método foreign es necesario indicar la propiedad del otro objeto del que se obtendrá su clave primaria.
- Suponiendo que hay una relación uno a uno entre la tabla izquierda y tabla derecha como una relación Uno a uno (bidireccional) la clave primaria de tabla derecha sería la misma que la de tabla izquierda.

```
<id column="Id" name="id" type="integer">
    <generator class="foreign" >
        <param name="property">profesor</param>
    </generator>
</id>
```

Hibernate Query

- Hibernate tiene el objeto Query que nos da acceso a todas las funcionalidades para poder leer objetos desde la base de datos.
- Lanzar una consulta con Hibernate es bastante simple. Usando la session llamamos al método createQuery(String queryString) con la consulta en formato HQL y nos retorna un objeto.
  - Después, sobre el objeto Query llamamos al método list() que nos retorna una lista de los objetos que ha retornado.

```
Query query = session.createQuery("SELECT p FROM Profesor p");
List<Profesor> profesores = query.list();
for (Profesor profesor : profesores) {
    System.out.println(profesor.toString());
}
```

Hibernate Query

- Lista de Objetos
  - El método list() nos retorna una lista con todos los objetos que ha retornado la consulta. En caso de que no se encuentre ningún resultado se retornará una lista sin ningún elemento.

```
Query query = session.createQuery("SELECT p FROM Profesor p");
List<Profesor> profesores = query.list();
for (Profesor profesor : profesores) {
    System.out.println(profesor.toString());
}
```

Hibernate Query

- Lista de Array de objetos
  - HQL también permite que las consultas retornen datos escalares en vez de clases completas.

```
SELECT p.id,p.nombre FROM Profesor p
```

  - En estos casos el método list() retorna una lista con una Array de objetos con tantos elementos como propiedades hayamos puesto en la SELECT.

```
Query query = session.createQuery("SELECT p.id,p.nombre FROM Profesor p");
List<Object[]> listDatos = query.list();
for (Object[] datos : listDatos) {
    System.out.println(datos[0] + "--" + datos[1]);
}
```

Hibernate Query

- Hay otro caso cuando hay una única columna en el SELECT de datos escalares. Es ese caso, como el array a retornar dentro de la lista solo tendría un elemento , no se retorna una lista de arrays List<Object[]> sino únicamente una lista de elementos List<Object>.

```
Query query = session.createQuery("SELECT p.nombre FROM Profesor p");
List<Object> listDatos = query.list();
for (Object datos : listDatos) {
    System.out.println(datos);
}
```

Hibernate Query

- uniqueResult
  - En muchas ocasiones una consulta únicamente retornará cero o un resultado. En ese caso es poco práctico que nos retorne una lista con un único elemento. Para facilitarnos dicha tarea Hibernate dispone del método uniqueResult().
  - Este método retornará directamente el único objeto que ha obtenido la consulta. En caso de que no encuentre ninguno se retornará null.

```
Profesor profesor = (Profesor) session.createQuery("SELECT p FROM Profesor p WHERE id=1001").uniqueResult();
System.out.println("Profesor con Id 1001=" + profesor.getNombre());
```

Consultas con nombre

- En cualquier libro sobre arquitectura del software siempre se indica que las consultas a la base de datos no deberían escribirse directamente en el código sino que deberían estar en un fichero externo para que puedan modificarse fácilmente.
- Hibernate provee una funcionalidad para hacer ésto mismo de una forma sencilla. En cualquier fichero de mapeo de Hibernate se puede incluir el tag <query> con la consulta HQL que deseamos lanzar.

Consultas con nombre

- Tag <query>
  - name:Este atributo define el nombre de la consulta.Es el nombre que posteriormente usaremos desde el código Java para acceder a la consulta.
  - contenido:El contenido del tag <query> es la consulta en formato HQL que ejecutará Hibernate.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hib
<hibernate-mapping>
    <class name="ejemplo01.Profesor">
        <id column="Id" name="id" type="integer"/>
        <property name="nombre"/>
        <property name="ape1"/>
        <property name="ape2"/>
    </class>

    <query name="findAllProfesores"><![CDATA[
        SELECT p FROM Profesor p
    ]]></query>
</hibernate-mapping>
```

Hibernate Query Languague (HQL)

- HQL es el lenguaje de consultas que usa Hibernate para obtener los objetos desde la base de datos. Su principal particularidad es que las consultas se realizan sobre los objetos java que forman nuestro modelo de negocio, es decir, las entidades que se persisten en Hibernate.

Hibernate Query Languague (HQL)

- Ésto hace que HQL tenga las siguientes características:
  - Los tipos de datos son los de Java.
  - Las consultas son independientes del lenguaje de SQL específico de la base de datos.
  - Las consultas son independientes del modelo de tablas de la base de datos.
  - Es posible tratar con las colecciones de Java.
  - Es posible navegar entre los distintos objetos en la propia consulta.

Hibernate Query Languague (HQL)

- En Hibernate las consultas HQL se lanzan (o se ejecutan) sobre el modelo de entidades que hemos definido en Hibernate, esto es, sobre nuestras clases de negocio.
- Nuestro modelo de tablas en HQL son las clases Java y NO las tablas de la base de datos. Es decir que cuando hagamos ”SELECT columna FROM nombreTabla” , el ”nombreTabla” será una clase Java y ”columna” será una propiedad Java de dicha clase y nunca una tabla de la base de datos ni una columna de una tabla.

Hibernate Query Languague (HQL)

```
SELECT c FROM Ciclo c ORDER BY nombre
```

- ¿Qué diferencias podemos ver entre HQL y SQL?
  - Ciclo hace referencia a la clase Ciclo y NO a la tabla CicloFormativo. Nótese que la clase Java y la tabla tienen distinto nombre .
  - Es necesario definir el alias c de la clase Java Ciclo.
  - Tras la palabra SELECT se usa el alias en vez del “*”.
  - Al ordenar los objetos se usa la propiedad nombre de la clase Ciclo en vez de la columna nombreCiclo de la tabla CicloFormativo.
  - Hibernate soporta no incluir la parte del SELECT en la consulta HQL, quedando entonces la consulta de la siguiente forma:
    - FROM Ciclo

Hibernate Query Languague (HQL)

- Respecto a la sensibilidad de las mayúsculas y minúsculas
  - Las palabras clave del lenguaje NO son sensibles a las mayúsculas o minúsculas.
    - select count(*) from Ciclo
    - SELECT COUNT(*) FROM Ciclo
  - El nombre de las clases Java y sus propiedades SI son sensibles a las mayúsculas o minúsculas.
    - SELECT c.nombre FROM Ciclo c WHERE nombre='Desarrollo de aplicaciones Web'
  - Al realizar comparaciones con los valores de las propiedades, éstas NO son sensibles a las mayúsculas o minúsculas.

Hibernate Query Languague (HQL)

- Filtrado
  - Al igual que en SQL en HQL también podemos filtrar los resultados mediante la cláusula WHERE. La forma de usarla es muy parecida a SQL.
    - SELECT p FROM Profesor p WHERE nombre='ISABEL' AND ape1<>'ORELLANA'
  - Al igual que con el nombre de la clase, el nombre de los campos del WHERE siempre hace referencia a las propiedades Java y nunca a los nombres de las columnas. De esa forma seguimos independizando nuestro código Java de la estructura de la base de datos.

Hibernate Query Languague (HQL)

- Para comparar los datos en una expresión se pueden usar las siguientes Operadores:
  - Signo igual ”=”: La expresión será verdadera si los dos datos son iguales. En caso de comparar texto, la comparación no es sensible a mayúsculas o minúsculas.
  - Signo mayor que ”>”: La expresión será verdadera si el dato de la izquierda es mayor que el de la derecha.
  - Signo mayor que ”>=”: La expresión será verdadera si el dato de la izquierda es mayor o igual que el de la derecha.
  - Signo mayor que ”<”: La expresión será verdadera si el dato de la izquierda es menor que el de la derecha.

Hibernate Query Languague (HQL)

- Signo mayor que ”<=”: La expresión será verdadera si el dato de la izquierda es menor o igual que el de la derecha.
- Signo desigual ”<>”: La expresión será verdadera si el dato de la izquierda es distinto al de la derecha.
- Signo desigual ”!=”: La expresión será verdadera si el dato de la izquierda es distinto al de la derecha.
- Operador ”between”: La expresión será verdadera si el dato de la izquierda está dentro del rango de la derecha.
  - SELECT tb FROM TiposBasicos tb WHERE inte BETWEEN 1 AND 10

Hibernate Query Languague (HQL)

- Operador ”in”: La expresión será verdadera si el dato de la izquierda está dentro de la lista de valores de la derecha.
  - SELECT tb FROM TiposBasicos tb WHERE inte IN (1,3,5,7)
- Operador ”like”: La expresión será verdadera si el dato de la izquierda coincide con el patrón de la derecha. Se utilizan los mismos signos que en SQL ”%” y ”_”.
  - SELECT tb FROM TiposBasicos tb WHERE stri LIKE 'H_la%’
- Operador ”not”: Niega el resultado de una expresión.

Hibernate Query Languague (HQL)

- Expresión ”is null”: Comprueba si el dato de la izquierda es null.
  – SELECT tb FROM TiposBasicos tb WHERE dataDate IS NULL

Hibernate Query Languague (HQL)

- Operadores Lógicos
  - Se puede hacer uso de los típicos operadores lógicos como en SQL:
    - AND
    - OR
    - NOT
  - SELECT p FROM Profesor p WHERE nombre='ANTONIO' AND (ape1='LARA' OR ape2='RUBIO')

Hibernate Query Languague (HQL)

- Operadores Aritméticos
  - Se puede hacer uso de los típicos operadores aritméticos:
    - Suma +
    - Resta –
    - Multiplicación *
    - División /
  - SELECT tb FROM TiposBasicos tb WHERE (((inte+1)*4)-10)/2=1

Hibernate Query Languague (HQL)

- Funciones de agregación
  - Las funciones de agregación que soporta HQL son:
    - AVG(): Calcula el valor medio de todos los datos.
    - SUM(): Calcula la suma de todos los datos.
    - MIN(): Calcula el valor mínimo de todos los datos.
    - MAX(): Calcula el valor máximo de todos los datos.
    - COUNT(): Cuanta el nº de datos.
  - SELECT AVG(c.horas), SUM(c.horas), MIN(c.horas), MAX(c.horas), COUNT(*) FROM Ciclo c

Hibernate Query Languague (HQL)

- Funciones sobre escalares
  - Algunas de las funciones que soporta HQL sobre datos escalares son:
    - UPPER(s): Transforma un texto a mayúsculas.
    - LOWER(s): Transforma un texto a minúsculas.
    - CONCAT(s1, s2): Concatena dos textos
    - TRIM(s):Elimina los espacio iniciales y finales de un texto.
    - SUBSTRING(s, offset, length): Retorna un substring de un texto.El offset empieza a contar desde 1 y no desde 0.
    - LENGTH(s): Calcula la longitud de un texto.
    - ABS(n): Calcula el valor absoluto de un número.
    - SQRT(n): Calcula la raíz cuadrada del número
    - Operador ”||” : Permite concatenar texto.
  - SELECT p.nombre || ' ' || p.ape1 || ' ' || p.ape2 FROM Profesor p WHERE id=1001

Hibernate Query Languague (HQL)

- Ordenación
  - Como en SQL también es posible ordenar los resultados usando ORDER BY. Su funcionamiento es como en SQL.
    - SELECT p FROM Profesor p ORDER BY nombre ASC,ape1 DESC
  - Las palabras ASC y DESC son opcionales al igual que en SQL.
  - El uso de funciones escalares y funciones de agrupamiento en la cláusula ORDER BY sólo es soportado por Hibernate si es soportado por el lenguaje de SQL de la base de datos sobre la que se está ejecutando.
  - No se permite el uso de expresiones aritméticas en la cláusula ORDER BY.

Hibernate Query Languague (HQL)

- Agrupaciones
  - Al igual que en SQL se pueden realizar agrupaciones mediante las palabras claves GROUP BY y HAVING
    - SELECT nombre, count(nombre) FROM Profesor p GROUP BY nombre HAVING count(nombre)>1 ORDER BY count(nombre)
  - El uso de funciones escalares y funciones de agrupamiento en la cláusula HAVING sólo es soportado por Hibernate si es soportado por el lenguaje de SQL de la base de datos sobre la que se está ejecutando.
  - No se permite el uso de expresiones aritméticas en la cláusula GROUP BY.

Hibernate Query Languague (HQL)

- Subconsultas
  - HQL también soporta subconsultas como en SQL.
    - SELECT c.nombre,c.horas FROM Ciclo c WHERE c.horas > (SELECT AVG(c2.horas) FROM Ciclo c2)

Hibernate Query Languague (HQL)

- Parámetros
  - Posicional Esta forma de definir parámetros es muy similar a la que usa SQL.Se basa en definir cada parámetro como un interrogante ”?”. Posteriormente estableceremos a la clase Query el valor de cada uno de los parámetros.

```
String nombre="ISIDRO";
String ape1="CORTINA";
String ape2="GARCIA";

Query query = session.createQuery("SELECT p FROM Profesor p where nombre=? AND ape1=? AND ape2=?");
query.setString(0,nombre);
query.setString(1,ape1);
query.setString(2,ape2);

List<Profesor> profesores = query.list();
for (Profesor profesor : profesores) {
    System.out.println(profesor.toString());
}
```

Hibernate Query Languague (HQL)

- Por nombre
  - En este caso los parámetros se definen como nombre precedidos de dos puntos ":". Esto hace que el código sea más legible y menos propenso a error.

```
String nombre="ISIDRO";
String ape1="CORTINA";
String ape2="GARCIA";

Query query = session.createQuery("SELECT p FROM Profesor p where nombre=:nombre AND ape1=:ape1 AND ape2=:ape2");
query.setString("nombre",nombre);
query.setString("ape1",ape1);
query.setString("ape2",ape2);

List<Profesor> profesores = query.list();
for (Profesor profesor : profesores) {
    System.out.println(profesor.toString());
}
```

Hibernate Query Languague (HQL)

- setParameter
  - Existe otra forma de asignar valores a los parámetros que es independiente del tipo, según si el parámetro es por índice o por nombre. Los métodos son:
    - setParameter(int position,Object val)
    - setParameter(String name,Object val)

```
String nombre="ISIDRO";
String ape1="CORTINA";
String ape2="GARCIA";

Query query = session.createQuery("SELECT p FROM Profesor p where nombre=:nombre AND ape1=:ape1 AND ape2=:ape2");
query.setParameter("nombre",nombre);
query.setParameter("ape1",ape1);
query.setParameter("ape2",ape2);

List<Profesor> profesores = query.list();
for (Profesor profesor : profesores) {
    System.out.println(profesor.toString());
}
```

Hibernate Query Languague (HQL)

- Navegación por propiedades
  - Una característica de HQL es la posibilidad de navegar por las propiedades.
    - En HQL si referenciamos una propiedad de un clase Java, esta propiedad puede no ser un valor escalar sino una referencia a otro objeto Java, que a su vez tiene más propiedades, lo que a su vez nos puede llevas hasta otro objeto Java y así sucesivamente.
    - SELECT p.nombre.ape1 FROM Profesor p

Hibernate Query Languague (HQL)

- SELECT p.nombre.ape1, p.direccion.municipio.nombre FROM Profesor p

```
Profesor profesor=.....;

String nombreMunicipio=profesor.getDireccion().getMunicipio().getNombre();
```

```
SELECT
    profesor0_.ape1 AS col_0_0_,
    municipio2_.NombreMunicipio AS col_1_0_
FROM
    Profesor profesor0_,
    Direccion direccion1_,
    Municipios municipio2_
WHERE
    profesor0_.Id=direccion1_.Id AND
    direccion1_.idMunicipio=municipio2_.idMunicipio
```

Hibernate Query Languague (HQL)

- Colecciones
  - Otra característica de HQL es la posibilidad de tratar con colecciones dentro de la propia consulta.
    - SIZE(c): Obtiene el nº de elementos de una colección.
      - SELECT p.nombre.ape1, SIZE(p.correosElectronicos) FROM Profesor p GROUP BY p.nombre.ape1
    - c IS EMPTY: Comprueba si una colección no tiene elementos.
      - SELECT p.nombre FROM Profesor p WHERE p.correosElectronicos IS EMPTY
    - IS NOT EMPTY c: Comprueba si una colección sí tiene elementos.
      - SELECT p.nombre FROM Profesor p WHERE p.correosElectronicos IS NOT EMPTY

Hibernate Query Languague (HQL)

- MINELEMENT(c): Retorna el menor de los elementos de la colección. Para averiguar cual es el menor siempre usa el campo que hace de id.

- MAXELEMENT(c): Retorna el mayor de los elementos de la colección. Para averiguar cual es el mayor siempre usa el campo que hace de id.

Contacto

Jorge Alberto Montalvo Olvera

Ingeniero en Computación

jorge.amontalvoo@gmail.com