Tema ORM: Arquitectura ORM - Hibernate

Herramienta ORM (Object Relational Mepping): Permite convertir datos entre lenguajes orientados a objetos y SGBD relacionales. Convierte de forma automática los datos primitivos a los almacenados en las tablas

Características:

- Permite trabajar con clases en lugar de tablas
- Permite trabajar con diversas BD y cambios sencillos solo modificando hibernate.conf
- Se genera de forma automática el SQL para el acceso a la BD Permite trabajar con objetos persistentes y sus asociaciones

Ventajas	Inconvenientes
Rapidez de desarrollo	Tiempo de aprendizaje
Abstracción de la BD	Aplicaciones algo más lentas
Reutilización	
Seguridad	
Mantenimiento de código	
Lenguaje propio para consultas basado en los objetos	

Hibernate:

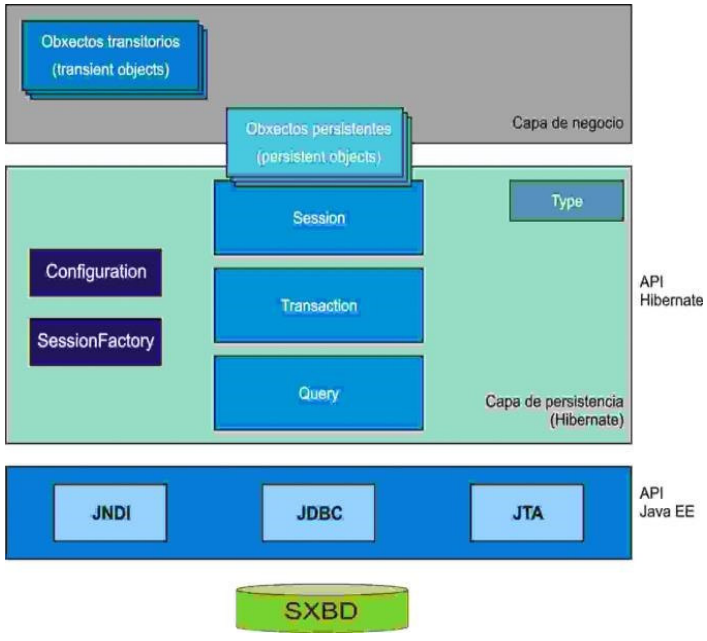
Permite usar objetos Java, conocidos como POJOS, los cuales se almacenan y recuperan de la BD mediante una conexión entre Hibernate y la BD usando objetos Session

Pojo: Objeto serializable, con un constructor sin argumentos y permite acceso a las propiedades mediante getters y setters

Arquitectura de Hibernate:

Sigue una arquitectura de dos capas:

- *Capa de **Persistencia***
- *Capa de **Dominio/Negocio***



Interfaces más importantes de la arquitectura:

- Session: Interfaz primaria utilizada por cualquier aplicación Hibernate para establecer la comunicación entre Hibernate y la BD
- SessionFactory: Permite obtener instancias de Session y configura la conexión a la BD
- Transaction: Permite controlar las transacciones contra la BD
- Query: Permite realizar peticiones contra la BD mediante HQL
- Configuration: Para configurar y arrancar Hibernate, además de indicar los mapeos de las clases

Métodos destacados:

- objSession.save(objeto)
- objSession.update(objeto)
- objSession.delete(objeto)
- objSession.createQuery(HQL)

Estructura de fichero para el mapeo objeto-relacional:

- Clases Java (Pojos): Representan los objetos que tienen una correspondencia con las tablas de la BD relacional
- Fichero de mapeo (.hbm.xml): Indica el mapeo entre los atributos de un pojo y los campos de la tabla a la que hace referencia

Clases persistentes:

Representan objetos de una aplicación que usa Hibernate y se corresponde con la información almacenada en una BD relacional, para poder ser usados en Hibernate deben cumplir las siguientes especificaciones JavaBeans:

- Constructor sin argumentos
- Atributo con funcionalidad ID
- Getters y setters
- Implementar Serializable

Podemos tener más pojos que tablas en la BD, dos pojos diferentes podrían mapear una sola tabla de la BD

Fichero de mapeo **(.hbm.xml)**:

- <hibernate-mapping>: Elemento raíz contiene todas las clases de los objetos persistentes default-lazy="true"
- <class>: Clases de los objetos persistentes
  - name="pojo" table="tablaBD"
  - lazy="true" <id>
  - name="atributoPojo" column="atributoBD"
- <property>
  - name="atributoPojo" column="atributoBD"
  - lazy="false"

HibernateUtil, objetos **SessionFactory** y **Session**:

Esta clase basada en el modelo Singleton está especializada en crear objetos Session a esta clase debemos añadirle el siguiente método:

```
public static Session inicioSession () {

    return getSessionFactory().openSession();
}
```

La primera vez que se ejecute este método se creará además del objeto **Session,** el objeto **SessionFactory** (se crea en primera instancia) con el cual Hibernate se comunicará con la BD

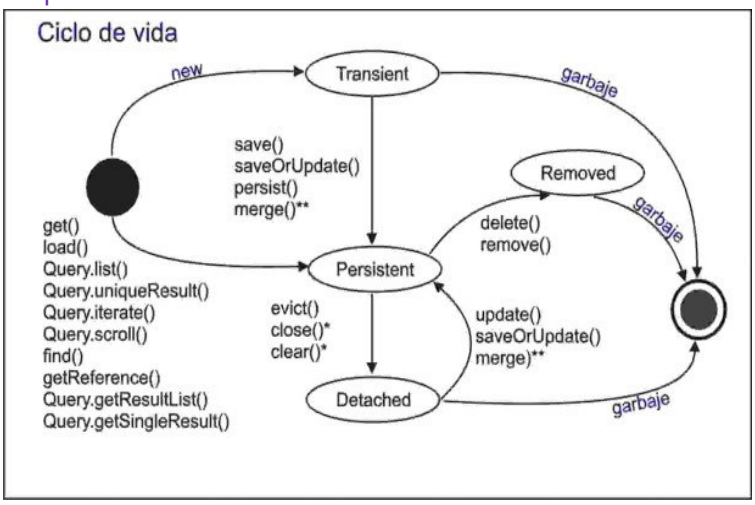
El objeto Session nos permite realizar consultas de las siguientes maneras:

- HQL: Lenguaje de consulta orientado a objetos propio de Hibernate, similar a SQL pero usando clases persistentes
  - objSession.createQuery(HQL)
- QBC (Query By Criteria): Permite crear consultas sobre clases persistentes definiendo restricciones, este tipo de consultas pertenecen a la interfaz Criteria de la API de Hibernate
  - objSession.createCriteria(pojo.class)
- SQL nativo: Solo se debe usar cuando necesitemos una característica propia de SQL y no tengamos posibilidad de usar HQL o QBC
  - objSession.createSQLQuery(SQL)

Todas estas consultas pueden devolver un único resultado o una lista de los mismos:

- Resultado único
  - .uniqueResult()
  - El resultado se debe guardar en un objeto Pojo
- Lista de resultados
  - .list()
    - El resultado se debe guardar en un objeto List <pojo>
  - .iterate()
    - El resultado se debe guardar en un objeto Iterator

Ciclo de vida de un objeto persistente:



- **Transitorio** (temporal): Instanciado pero no asociado a una sesión de Hibernate (aun no pertenecen a la BD)
- **Persistente**: El objeto tiene una representación en la BD, en este estado si el objeto sufre alguna modificación se sincronizará con la BD realizando los cambios en cascada al ejecutar: objSession.getTransaction().commit()
- **Desconectado** (Detached): En este estado están aquellos objetos que fueron instanciados pero no referenciados en la BD, porque la sesión en la que se creó fue cerrada, desde este estado se pueden alcanzar la persistencia (el objeto fue guardado/actualizado) y el borrado (se ejecuta .commit() y el objeto no se guarda en la BD o se borra de la misma)
- **Borrado**: El objeto se borra de la BD al finalizar la transacción, además se descartan todas sus referencias

Un objeto persistente puede ser recuperado de la BD de varias maneras:

- objSession.get(pojo.class, id)
  - Devuelve el objeto o null
- objSession.load(pojo.class, id)
  - Devuelve el objeto o ObjectNotFoundException
- Consulta HQL, QBC o SQL

Arquitectura caché de Hibernate:

Se divide en dos niveles:

- Caché de **primer** nivel (caché de contexto de persistencia):
  Asociada al objeto Session, no se puede desactivar y no necesita configuración. Por cada sesión abierta tendremos una caché de este tipo, funciona de punto de acceso a los objetos

- Caché de **segundo** nivel:
  Asociada al objeto SessionFactory, permite mejorar el rendimiento y el acceso concurrente por varios usuarios a la BD. En esta caché se trabaja con los objetos recuperados y manejados por todas las sesiones. Por defecto esta deshabilitada

Transacciones:

Se expresa el **inicio** de la transacción mediante

objSession.beginTransaction()

y se **cierra** mediante

objSession.getTransaction().commit o objSession.getTransaction().rollback()

(solo afecta a la BD no al objeto en memoria)

Asociaciones entre entidades:

Características:

- Cardinalidad:
  - 1:1
  - 1:N
  - N:M
- Direccionalidad: Establece la forma en que se navega entre las entidades de una relación
  - Unidireccional: Una entidad referencia a otra, solo uno de los objetos conoce el objeto del otro lado de la relación
  - Bidireccional: Cada entidad referencia a otra, ambos objetos conocen el objeto del otro lado de la relación
    - one-to-many
    - many-to-many

Estrategias de carga de objetos:

Estas estrategias permitirán establecer cómo se cargan los objetos en memoria, cargando: solo el objeto o el objeto con sus relaciones y colecciones

- Estrategia temprana (Eager): Indica que en el momento de obtener la entidad maestra se obtienen todas las entidades hijas asociadas
- Estrategia perezosa (Lazy): Solo obtiene la entidad maestra y los datos de las entidades hojas se obtienen al forzar la consulta
- Estrategia de carga de objetos (fetch mode):
  - Recuperación por unión (join fetching): Se recupera la instancia asociada a la colección con un solo SELECT usando join
  - Recuperación por selección (select fetching): Se usa un segundo SELECT para recuperar la entidad/colección asociada
  - Recuperación por subselección (subselect fetching): Se usa un segundo SELECT para recuperar la entidad/colección asociada de un conjunto de entidades ya recuperadas
  - Recuperación por lotes: Las entidades/colecciones asociadas se recuperan en bloques a partir de una lista de IDs

- Estrategia de carga de objetos (fetch type):
  - Recuperación inmediata: La entidad/colección asociada se carga cuando se carga el objeto propietario
  - Recuperación perezosa de colecciones: La colección asociada no se carga hasta que se invoca una operación sobre la colección, posee una versión más perezosa que carga las colecciones por separado cuando sea necesario

- Recuperación por proxy: La entidad asociada no se trae a memoria hasta que no accedemos a una propiedad de la misma
- Recuperación no-proxy: La entidad asociada no se trae a memoria hasta que no se usa la variable de la instancia

Hibernate por defecto trabaja:

- Para referencias a **entidades** mediante recuperación por proxy de forma perezosa
- Para referencias a **colecciones** mediante recuperación perezosa por selección

Mapeos:

Asociación one - to - one:

esquema mapeos.xml

```
<!--MAPEO 1:1-->
<!--NOTA: La etiqueta <one-to-one> se establece en cuanlquier lado de la relacion-->
    <!--POJO A-->
    <hibernate-mapping>
        <class name="pojos.A" table="tablaA">
            <id name="idA" column="idA" type="string" />
            <property name="nombre" column="nombre" />
            <one-to-one class="pojos.B" name="objetoB" cascade="all" />
        </class>
    </hibernate-mapping>
    <!--POJO B-->
    <hibernate-mapping>
        <class name="pojos.B" table="tablaB">
            <id name="idB" column="idB" type="string"/>
            <property name="nombre" column="nombre" />
        </class>
    </hibernate-mapping>

<!--MAPEO 1:1 CLAVE FORANEA / BIDIRECCIONAL-->
<!--NOTA: La etiqueta <one-to-one> se establece en cuanlquier lado de la relacion-->
    <!--POJO A-->
    <hibernate-mapping>
        <class name="pojos.A" table="tablaA">
            <id name="idA" column="idA" type="string" />
            <property name="nombre" column="nombre" />
            <one-to-one class="pojos.B" name="objetoB" cascade="all" />
        </class>
    </hibernate-mapping>
    <!--POJO B-->
    <hibernate-mapping>
        <class name="pojos.B" table="tablaB">
            <id name="idA" column="idA" type="string">
                <generator class="foreing">
                    <param name="property">objA</param>
                </generator>
            </id>
            <property name="nombre" column="nombre" />
            <one-to-one class="pojos.A" name="objetoA" constrained="true" />
        </class>
    </hibernate-mapping>
```

index.php | esquema pojos.java

```
//RELACION 1:1
    //POJO A
    private int idA;
    private String nombre;
    private B objetoB;
    //POJO B
    private idB;
    private String nombre;

//RELACION 1:1 CLAVE FORANEA / BIDIRECCIONAL
    //POJO A
    private int idA;
    private String nombre;
    private B objetoB;
    //POJO B
    private idB;
    private String nombre;
    private A objetoA;
```

Java source file | length : 1950 lines : 92

Asociación one-to-many / many-to-one:

esquema mapeos.xml

```
<!--MAPEO HERENCIA-->
    <!--POJO QUE TIENE HERENCIA-->
    <hibernate-mapping>
        <class name="pojos.Padre" table="tablaPadre">
            <id name="idPadre" column="idPadre" type="string" />
            <property name="nombre" column="nombre" />
            <joined-subclass name="pojos.HijoA" table="tablaHijoA" >
                <key column="idHijoA" />
                <property name="datoA" column="datoA" />
            </joined-subclass>
            <joined-subclass name="pojos.HijoB" table="tablaHijoB" >
                <key column="idHijoB" />
                <property name="datoB" column="datoB" />
            </joined-subclass>
        </class>
    </hibernate-mapping>
```

index.php | esquema pojos.java

```
//RELACION HERENCIA
    //POJO PADRE
    private String idPadre;
    private String nombre;
    //POJO HIJOA
    private String datoA;
    //POJO HIJOB
    private String datoB;
```

Java source file

```
<!--
<hibernate-mapping>
    <class name="pojos.B" table="tablaB">
        <id name="idB" column="idB" type="integer" />
        <property name="nombre" column="nombre" />
        <set name="arrayA" table="tablaA" cascade="save-update" inverse="true" >
            <key>
                <column name="idB" not-null="true" />
                <!--NOTA idB aqui hace referencia al int declarado en la tablaA -->
            </key>
            <one-to-many class="pojos.A" />
        </set>
    </class>
</hibernate-mapping>
```

Asociación many - to - many:

esquema mapeos.xml

```
<!--MAPEOS HIBERNATE-->
<!--MAPEO N:M-->
    <!--POJO A-->
    <hibernate-mapping>
      <class name="pojos.A" table="tablaA" >
        <id name="idA" column="idA" type="integer" />
        <property name="nombre" column="nombre" />
        <set name="arrayB" table="a-has-b" cascade="save-update" inverse="true" >
            <key>
                <column name="a" />
            </key>
            <many-to-many column="b" class="pojos.B" />
        </set>
      </class>
    </hibernate-mapping>
    <!--POJO B-->
    <hibernate-mapping>
      <class name="pojos.B" table="tablaB" >
        <id name="idB" column="idB" type="integer" />
        <property name="nombre" column="nombre" />
        <set name="arrayA" table="a-has-b" cascade="save-update" inverse="false" >
            <key>
                <column name="b" />
            </key>
            <many-to-many column="a" class="pojos.A" />
        </set>
      </class>
    </hibernate-mapping>
```

index.php | esquema pojos.java

```
//RELACIONES POJOS HIBERNATE
//REALACION N:M
    //POJO A
    //ATRIBUTOS
    private int idA;
    private String nombre;
    private Set <B> arrayB = new HashSet();
    //POJO B
    //ATRIBUTOS
    private int idB;
    private String nombre;
    private Set <A> arrayA = new HashSet();
```

Java source file | length : 1950 lines : 92

Herencia:

Clave compuesta:

esquema mapeos.xml

```
<!--MAPEO CLAVE COMPUESTA SIMPLE-->
    <!--POJO QUE TIENE LA CLAVE COMPUESTA-->
    <hibernate-mapping>
        <class name="pojos.Usos" table="usos" >
            <composite-id>
              <key-many-to-one name="idA" column="idA" class="pojos.A"/>
              <key-property name="fecha" column="fecha"/>
            </composite-id>
            <property name="tipoUso" column="tipoUso"/>
        </class>
    </hibernate-mapping>

<!--MAPEO CLAVE COMPUESTA COMPUESTO-->
    <!--POJO QUE TIENE LA CLAVE COMPUESTA-->
    <!--VERSION A-->
    <hibernate-mapping>
        <class name="pojos.Usos" table="usos" >
            <composite-id>
              <key-many-to-one name="idA" column="idA" class="pojos.A"/>
              <key-many-to-one name="idB" column="idB" class="pojos.B"/>
              <key-property name="fecha" column="fecha"/>
            </composite-id>
            <property name="tipoUso" column="tipoUso"/>
        </class>
    </hibernate-mapping>
    <!--VERSION B-->
    <hibernate-mapping>
        <class name="pojos.Usos" table="usos" >
            <composite-id>
              <key-many-to-one name="idA" column="idA" class="pojos.A"/>
              <key-property name="fecha" column="fecha"/>
            </composite-id>
            <property name="tipoUso" column="tipoUso"/>
            <many-to-one class="pojo.B" name="idB" column="idB" />
        </class>
    </hibernate-mapping>
```

index.php | esquema pojos.java

```
//RELACION CLAVE COMPUESTA SIMPLE
    //POJO QUE TIENE LA CLAVE COMPUESTA
    private A objetoA;
    private Date fecha; //import java.sql.Date;

//RELACION CLAVE COMPUESTA COMPUESTO
    //POJO QUE TIENE LA CLAVE COMPUESTA => VALIDO PARA LA VERSION A y B
    private A objetoA;
    private B objetoB;
    private Date fecha; //import java.sql.Date;
```

Java source file | length : 1950 lines : 92 | Ln : 1 Col : 1 Sel : 0

Clave primaria autoincrementable:

esquema mapeos.xml

```
<!--CLAVE AUTOINCREMENTABLE-->
    <!--POJO CON CLAVE AUTOINCREMENTABLE-->
    <hibernate-mapping>
        <class name="pojos.A" table="tablaA" >
            <id name="idA" column="idA" type="integer">
                <generator class="increment"/>
            </id>
          <property name="nombre" column="nombre"/>
        </class>
    </hibernate-mapping>
```

index.php | esquema pojos.java

```
//CLAVE AUTOINCREMENTABLE
    //POJO CON CLAVE AUTOINCREMENTABLE
    private int idA;        //no se añade al constructor
    private String nombre;
```

Java source file | length : 1950 lines : 92 | Ln : 1

Otros datos de repaso para los pojos:

```
68  //Getters y Setters
69      //Set <A> arrayA = new HashSet();
70      public void setArrayA (Set<A> arrayA) {
71          this.arrayA = arrayA;
72      }
73
74      public Set<A> getArrayA () {
75          return arrayA;
76      }
77
78      //Variables primitivas
79      public void nombre (String nombre) {
80          this.nombre = nombre;
81      }
82
83      public String getArrayA () {
84          return nombre;
85      }
86
87  //CONSTRUCTOR HERENCIA HIJOS
88      public Hijo (int idPadre, String nombrePadre, String datoHijo) {
89
90          super(idPadre, nombrePadre);
91          this.datoHijo = datoHijo;
92      }
```