HIBERNATE

HIBERNATE — INTRODUCCIÓN

1

HIBERNATE – INTRODUCCIÓN

HIBERNATE

- Hibernate es un *framework* de mapeo objeto-relacional (ORM) para Java. Permite interactuar con bases de datos relacionales de una manera más sencilla y abstracta, sin necesidad de escribir consultas SQL manualmente.
  - Mapeo Objeto-Relacional (ORM): Hibernate permite mapear clases de Java a tablas de una base de datos relacional, lo que significa que puedes trabajar con objetos Java en lugar de escribir consultas SQL directamente.
  - Abstracción de la Base de Datos: Hibernate abstrae las diferencias entre distintos sistemas de gestión de bases de datos (DBMS). Esto te permite cambiar la base de datos sin necesidad de cambiar el código de acceso a datos.
  - Automatización de Operaciones CRUD: Hibernate simplifica las operaciones básicas de bases de datos como crear, leer, actualizar y eliminar (CRUD). No necesitas escribir SQL explícito para estas operaciones, ya que Hibernate lo maneja automáticamente.
  - Gestión de Transacciones y Conexiones: Hibernate gestiona las conexiones a la base de datos y las transacciones, lo que facilita la implementación de transacciones de manera segura y eficiente.
  - Caché de Primer y Segundo Nivel: Hibernate incluye un sistema de caché para mejorar el rendimiento de las consultas, manteniendo los resultados de las operaciones anteriores en memoria, lo que reduce el número de accesos a la base de datos.

HIBERNATE – INTRODUCCIÓN

HIBERNATE

https://hibernate.org/
https://hibernate.org/orm/quickly/
https://hibernate.org/tools/
https://docs.jboss.org/hibernate/
https://docs.jboss.org/hibernate/orm/6.6/userguide/html_single/Hibernate_User_Guide.html

HIBERNATE – INTRODUCCIÓN

HIBERNATE

- Dependencias:

```
<dependencies>
    <!-- https://mvnrepository.com/artifact/org.postgresql/postgresql -->
    <dependency>
      <groupId>org.postgresql</groupId>
      <artifactId>postgresql</artifactId>
      <version>42.7.5</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/jakarta.persistence/jakarta.persistence-api -->
    <dependency>
      <groupId>jakarta.persistence</groupId>
      <artifactId>jakarta.persistence-api</artifactId>
      <version>3.2.0</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.hibernate.orm/hibernate-core -->
    <dependency>
      <groupId>org.hibernate.orm</groupId>
      <artifactId>hibernate-core</artifactId>
      <version>6.6.13.Final</version>
    </dependency>
  </dependencies>
```

HIBERNATE – INTRODUCCIÓN

HIBERNATE

- Fichero de configuración: resources/hibernate.cfg.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!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>

    <!-- Database connection settings -->
    <property name="hibernate.connection.driver_class">org.postgresql.Driver</property>
    <property name="hibernate.connection.url">jdbc:postgresql://localhost:5432/aves</property>
    <property name="hibernate.connection.username">user-aves</property>
    <property name="hibernate.connection.password">user-aves</property>

    <!-- JDBC connection pool (use the built-in) -->
    <property name="hibernate.connection.pool_size">10</property>

    <!-- DIALECTO SQL -->
    <property name="hibernate.dialect">org.hibernate.dialect.PostgreSQLDialect</property>

    <!-- Enable Hibernate's automatic session context management -->
    <property name="hibernate.current_session_context_class">thread</property>

    <!-- Echo all executed SQL to stdout -->
    <property name="hibernate.show_sql">true</property>

    <!-- Update the database schema on startup -->
    <property name="hibernate.hbm2ddl.auto">update</property>

    <!-- List of mapped classes -->
    <mapping class="com.fernandopaniagua.demohibernate.model.Ave"/>

  </session-factory>
</hibernate-configuration>
```

HIBERNATE – INTRODUCCIÓN

HIBERNATE

- Clase HibernateUtil.java

```
package com.fernandopaniagua.demohibernate.util;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {
    private static final SessionFactory sessionFactory = buildSessionFactory();

    private static SessionFactory buildSessionFactory() {
        try {
            // Cargar configuración desde hibernate.cfg.xml
            return new Configuration().configure().buildSessionFactory();
        } catch (Throwable ex) {
            System.err.println("Error al crear SessionFactory: " + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

HIBERNATE – INTRODUCCIÓN

HIBERNATE

- Anotaciones importantes en Hibernate:
  - @Entity: Marca una clase como entidad.
  - @Table: Define la tabla a la que está asociada la entidad.
  - @Id: Indica que el campo es la clave primaria.
  - @Column: Especifica las propiedades de la columna de la base de datos para un atributo de entidad
  - @GeneratedValue: Define cómo se generará el valor de la clave primaria (por ejemplo, de manera automática).
    - GenerationType.IDENTITY: El valor es generado por la base de datos.
    - GenerationType.SEQUENCE: Usa una secuencia de base de datos.
    - GenerationType.AUTO: Hibernate selecciona la mejor estrategia dependiendo de la base de datos.
    - GenerationType.TABLE: Usa una tabla especial para generar valores.
  - @OneToMany, @ManyToOne, @ManyToMany, @OneToOne: Para definir relaciones entre entidades.
  - @JoinColumn. Especifica la columna que se usará para unir una entidad con otra
  - @JoinTable. Se utiliza en relaciones @ManyToMany para definir la tabla intermedia que une las dos entidades.
  - @Transient. Indica que un campo no debe ser persistido en la base de datos.
  - @Version. Se utiliza para implementar el control de concurrencia optimista. Marca un campo que se usará para verificar que los datos no hayan sido modificados por otra transacción antes de realizar una actualización.
  - @Fetch. Especifica el tipo de carga que se debe usar en una relación.

HIBERNATE – INTRODUCCIÓN

HIBERNATE

- Definición entidad

```
package com.fernandopaniagua.demohibernate.model;

import jakarta.persistence.*;

@Entity
@Table(name = "aves")
public class Ave {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(name = "nombre")
    private String nombre;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
```

HIBERNATE – INTRODUCCIÓN

HIBERNATE

- Clase **Session**:

Método	Descripción
persist()	Inserta una nueva entidad (NO excepción si ya existe)
save()	Inserta una nueva entidad (excepción si ya existe)
saveOrUpdate()	Inserta o actualiza una entidad (NO excepción si ya existe)
merge()	Combina estado con la sesión, persistiendo el objeto.
load()	Carga una entidad por su ID (proxy si no existe). Utiliza lazy loading.
update()	Actualiza una entidad existente
get()	Obtiene una entidad por su ID. Null si no existe. Más "fiable".
delete()	Elimina una entidad
remove()	Elimina una entidad
flush()	Sincroniza el estado de la sesión con la BD
createQuery()	Crea una consulta HQL
createCriteria()	Crea una consulta Criteria programática.
createSQLQuery()	Ejecuta una consulta SQL nativa.
getTransaction()	Obtiene la transacción asociada con la sesión.
setFlushMode()	Establece el modo de sincronización (flush) para la sesión.
setCacheMode()	Define el comportamiento del caché de la sesión.
clear()	Limpia el contexto de la sesión
getCurrentSession()	Obtiene la sesión actual, usada principalmente en Spring.
isOpen()	Verifica si la sesión está abierta o cerrada.

HIBERNATE – INTRODUCCIÓN

HIBERNATE

- CRUD - Create

```
public static void create(String nombre) {
    // Obtener sesión
    Session session = HibernateUtil.getSessionFactory().openSession();
    session.beginTransaction();

    // Crear nueva instancia
    Ave ave = new Ave();
    ave.setNombre(nombre);

    // Guardar en base de datos
    session.persist(ave);

    session.getTransaction().commit();
    session.close();

    System.out.println("¡Ave guardada con éxito!");
}
```

HIBERNATE – INTRODUCCIÓN

HIBERNATE

- CRUD - Read

```
public static Ave read(int id) {
    Session session = HibernateUtil.getSessionFactory().openSession();
    session.beginTransaction();

    Ave ave = session.get(Ave.class, id);

    session.getTransaction().commit();
    session.close();

    return ave;
  }
```

HIBERNATE – INTRODUCCIÓN

HIBERNATE

- CRUD - Update

```
public static Ave update(Ave aveModificada) {
    Session session = HibernateUtil.getSessionFactory().openSession();
    session.beginTransaction();

    session.merge(aveModificada);
    Ave aveLeida = session.get(Ave.class, aveModificada.getId());

    session.getTransaction().commit();
    session.close();

    return aveLeida;
 }
```

HIBERNATE – INTRODUCCIÓN

HIBERNATE

- CRUD - Delete

```
public static void delete(Ave ave) {
    Session session = HibernateUtil.getSessionFactory().openSession();
    session.beginTransaction();

    session.remove(ave);

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

HIBERNATE – INTRODUCCIÓN

HIBERNATE

- HQL
  - ***Hibernate Query Language*** (HQL) es un lenguaje de consulta específico de Hibernate.
  - ***Java Persistence Query Language*** (JPQL) está basado en HQL.
  - Es un lenguaje de consulta orientado a objetos que se utiliza en Hibernate para interactuar con la base de datos a través de las entidades Java en lugar de las tablas de la base de datos directamente.
    - **Objetos en lugar de tablas**: En lugar de referirse a tablas, como en SQL, se hace referencia a las entidades (o clases) Java. Por ejemplo, en lugar de **FROM persona** (en SQL), en HQL sería **FROM Persona**.
    - **Atributos en lugar de columnas**: Las propiedades de la clase representan las columnas de la tabla en la base de datos. Por ejemplo, si tienes una clase Persona con una propiedad nombre, en HQL usarías **SELECT p.nombre FROM Persona p** en lugar de **SELECT nombre FROM persona**.
  - https://docs.jboss.org/hibernate/orm/6.6/userguide/html_single/Hibernate_User_Guide.html#query-language

HIBERNATE – INTRODUCCIÓN

HIBERNATE

- HQL
  - Sintaxis básica:
    - Operadores:
      - HQL admite operadores lógicos, de comparación y aritméticos similares a SQL:
        - Comparación: =, <, >, <=, >=, <>
        - Lógicos: AND, OR, NOT
        - En listados: IN, NOT IN
        - Patrón: LIKE, BETWEEN
    - Agregaciones:
      - HQL soporta funciones agregadas como COUNT, SUM, AVG, MIN, MAX.
        - SELECT AVG(e.salario) FROM Empleado e
    - Alias:
      - SELECT e.nombre, e.salario FROM Empleado e WHERE e.salario > 3000

HIBERNATE – INTRODUCCIÓN

HIBERNATE

- HQL
  - Sintaxis básica:
    - Subconsultas:
      - SELECT e FROM Empleado e WHERE e.salario > (SELECT AVG(e2.salario) FROM Empleado e2)
    - Update:
      - UPDATE Empleado e SET e.salario = 3500 WHERE e.id = 1
    - Delete:
      - DELETE FROM Empleado e WHERE e.salario < 2000
    - Insert:
      - INSERT Empleado (id, nombre, salario) values (101, 'Alberto', 30000)

HIBERNATE – INTRODUCCIÓN

HIBERNATE

- HQL. Sentencias esenciales.

Sentencia	Descripción	Ejemplo
FROM	Recupera todas las instancias de una clase (entidad).	FROM Usuario
SELECT	Selecciona propiedades específicas de una entidad.	SELECT u.nombre FROM Usuario u
WHERE	Filtra los resultados de la consulta según condiciones.	FROM Usuario u WHERE u.edad > 30
ORDER BY	Ordena los resultados de la consulta.	FROM Usuario u ORDER BY u.nombre ASC
JOIN / LEFT JOIN / INNER JOIN	Realiza combinaciones entre tablas relacionadas (usando asociaciones entre entidades).	FROM Pedido p JOIN p.cliente c
FETCH JOIN	Recupera asociaciones en una sola consulta (previene consultas adicionales).	FROM Pedido p LEFT JOIN FETCH p.detalles
GROUP BY	Agrupa los resultados de la consulta para aplicar funciones agregadas.	SELECT u.pais, COUNT(u) FROM Usuario u GROUP BY u.pais
HAVING	Filtra los resultados después de aplicar GROUP BY (similar a WHERE, pero para grupos).	SELECT u.pais, COUNT(u) FROM Usuario u GROUP BY u.pais HAVING COUNT(u) > 5
UPDATE	Actualiza registros de una entidad.	UPDATE Usuario u SET u.estado = 'activo' WHERE u.id = :id
DELETE	Elimina registros de una entidad.	DELETE FROM Usuario u WHERE u.estado = 'inactivo'
INSERT INTO ... SELECT	Inserta registros en una tabla basándose en una selección de otra tabla.	INSERT INTO UsuarioBackup (id, nombre) SELECT u.id, u.nombre FROM Usuario u
DISTINCT	Elimina duplicados de los resultados.	SELECT DISTINCT u.pais FROM Usuario u
LIKE	Filtra por coincidencias parciales en cadenas de texto.	FROM Usuario u WHERE u.nombre LIKE '%Juan%'
IN	Filtra por valores dentro de una lista o conjunto.	FROM Usuario u WHERE u.pais IN ('España', 'México', 'Argentina')
BETWEEN	Filtra valores dentro de un rango determinado.	FROM Usuario u WHERE u.edad BETWEEN 18 AND 30

HIBERNATE – INTRODUCCIÓN

HIBERNATE

- HQL
  - Ejemplos de SELECT:
    - Selección de datos (SELECT):
      - SELECT e FROM Empleado e
    - Condiciones (WHERE):
      - SELECT e FROM Empleado e WHERE e.salario > 3000
    - Ordenación (ORDER BY):
      - SELECT e FROM Empleado e ORDER BY e.salario DESC
    - Proyección (SELECT de campos específicos):
      - SELECT e.nombre, e.salario FROM Empleado e
    - Join:
      - SELECT e.nombre, d.nombre FROM Empleado e JOIN e.departamento d

HIBERNATE – INTRODUCCIÓN

HIBERNATE

- CRUD – Read All – Usando **Query y HQL**
  - Marcado como **deprecated** a partir de Hibernate 6.x
  - Propuesta: utilizar **TypedQuery** (createQuery con tipo explícito) o **createSelectionQuery**.

```
public static List<Ave> readAll() {
        Session session = HibernateUtil.getSessionFactory().openSession();

        Query query = session.createQuery("FROM Ave");
        List<Ave> aves = query.list();

        session.close();

        return aves;
    }
```

HIBERNATE – INTRODUCCIÓN

HIBERNATE

- CRUD – Read All – Usando **TypedQuery y HQL**

```
public static List<Ave> readAll() {
    Session session = HibernateUtil.getSessionFactory().openSession();

    TypedQuery<Ave> query = session.createQuery("FROM Ave", Ave.class);
    List<Ave> aves = query.getResultList();

    session.close();

    return aves;
}
```

HIBERNATE – INTRODUCCIÓN

HIBERNATE

- CRUD – Find – Usando **NamedQuery y HQL**
  - **Se define a nivel de entidad:**

```
@Entity
@Table(name = "aves")
@NamedQuery(
      name = "Ave.findByNombre",
      query = "FROM Ave e WHERE e.nombre LIKE :cadena"
)
public class Ave {
```

  - **Se usa en el DAO:**

```
public static List<Ave> readByContains(String cadenaBuscada) {
      Session session = HibernateUtil.getSessionFactory().openSession();

      Query query = session.createNamedQuery("Ave.findByNombre", Ave.class);
      query.setParameter("cadena", "%" + cadenaBuscada + "%");
      List<Ave> aves = query.getResultList();

      session.close();

      return aves;
   }
```

HIBERNATE – INTRODUCCIÓN

HIBERNATE

- CRUD – Read All – Usando **createSelectionQuery y HQL**

```
//Utilizando createSelectionQuery
  public static List<Ave> readAll() {
     Session session = HibernateUtil.getSessionFactory().openSession();

     List<Ave> aves = session.createSelectionQuery("FROM Ave", Ave.class).getResultList();

     session.close();

     return aves;
  }
```

HIBERNATE – INTRODUCCIÓN

HIBERNATE

- CRUD – Read All – Usando **JPA Criteria API**
  - JPA Criteria API es una forma de construir consultas dinámicas en Java, sin necesidad de escribir HQL o SQL directamente como cadenas de texto.
    - Cuando necesitas construir consultas de manera dinámica (por ejemplo, filtros opcionales).
    - Cuando quieres evitar *hardcode* con strings de HQL en tu código.
    - Cuando quieres consultas seguras y tipadas que sean fáciles de mantener.

```
//Utilizando criteriaAPI
  public static List<Ave> readAll() {
     Session session = HibernateUtil.getSessionFactory().openSession();

     CriteriaBuilder builder = session.getCriteriaBuilder();
     CriteriaQuery<Ave> criteria = builder.createQuery(Ave.class);
     criteria.from(Ave.class);
     List<Ave> aves = session.createQuery(criteria).getResultList();

     session.close();

     return aves;
  }
```

HIBERNATE – INTRODUCCIÓN

HIBERNATE

- Tipos de relaciones:

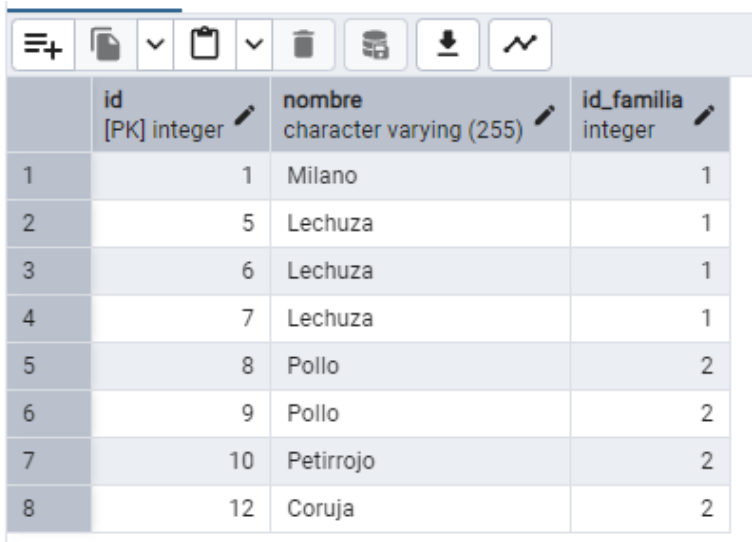
  - Hibernate soporta relaciones entre entidades de varios tipos, como:
    - @OneToOne: Relación uno a uno.
    - @OneToMany: Relación uno a muchos.
    - @ManyToOne: Relación muchos a uno.
    - @ManyToMany: Relación muchos a muchos.

HIBERNATE – INTRODUCCIÓN

HIBERNATE

- Relaciones MANY-TO-ONE
  - Al leer la entidad se lee referencia la entidad relacionada.
  - Al modificar la referencia a la entidad relacionada , se actualiza la entidad principal.

```
@ManyToOne
@JoinColumn(name = "id_familia") // Esta es la clave foránea en la tabla 'aves'
private Familia familia;
```



	id [PK] integer	nombre character varying (255)	id_familia integer
1	1	Milano	1
2	5	Lechuza	1
3	6	Lechuza	1
4	7	Lechuza	1
5	8	Pollo	2
6	9	Pollo	2
7	10	Petirrojo	2
8	12	Coruja	2

HIBERNATE – INTRODUCCIÓN

HIBERNATE

- Relaciones ONE-TO-MANY
  - Ojo a las referencias circulares.
  - Ojo al tipo de carga

```
@OneToMany(mappedBy = "familia", cascade = CascadeType.ALL, orphanRemoval = true, fetch =
FetchType.EAGER)
 private List<Ave> aveList = new ArrayList<>();
```

  - mappedBy →Atributo fk de la otra entidad.
  - cascade → Como se propagan los cambios en la entidad.
  - orphanRemoval → Al eliminar una entidad de la lista, se elimina en la base de datos.
  - fech → Como se realiza la carga de la lista:
    - FetchType.EAGER → Inmediatamente (puede producir consumo excesivo de memoria).
    - FetchType.LAZY → Bajo demanda. Más eficiente con muchos datos.

HIBERNATE – INTRODUCCIÓN

HIBERNATE

- Claves compuestas:
  - Se declara una clase anotada como @Embeddable que:
    - Contiene los atributos de la PK.
    - Implementa la interfaz Serializable.

```
@Embeddable
public class RegionPK implements Serializable {
    private String comunidad;
    private String provincia;
```

HIBERNATE – INTRODUCCIÓN

HIBERNATE

- Claves compuestas:
  - En la definición de la entidad:
    - Se declara un atributo del tipo de la clase que contiene la estructura de la pk y se anota como @EmbeddedId.

```
@Entity
@Table(name = "regiones")
public class Region {
    @EmbeddedId
    private RegionPK pk;
    private String ave;
```

  - Uso:

```
 //LECTURA DE REGION
RegionPK regionPK = new RegionPK("Extremadura", "Cáceres");
Region region = RegionDAO.read(regionPK);
System.out.println(region);
```