EXAMEN 2 EVAL ACCESO A DATOS

Ejercicio1

1. App.java

```
package com.example;

import entity.Ataque;
import entity.Permite;
import entity.Sistema;
import entity.Vulnerabilidad;
import java.time.LocalDate;
import java.util.List;
import java.util.Scanner;
import org.hibernate.Session;
import org.hibernate.Transaction;

public class App {

    static Scanner sc = new Scanner(System.in);
    static int opcion;

    public static void main(String[] args) {
        System.out.println("Test");

        Session session = HibernateUtil.get().openSession();

        do {
            System.out.println("**********MENÚ**********");
            System.out.println("1. Crear vulnerabilidad");
            System.out.println("2. Asignar vulnerabilidad - sistema");
            System.out.println("3. Asignar vulnerabilidad - ataque");
            System.out.println("4. Consulta 1");
            System.out.println("5. Consulta 2");
            System.out.println("6. Consulta 3");
            System.out.println("7. Consulta 4");
            System.out.println("8. Salir");

            System.out.println("Elige una opción: ");
            opcion = Integer.parseInt(sc.nextLine());

            switch (opcion) {
                case 1:
                    crearVulnerabilidad(session);
                    break;

                case 2:
                    asignarVulnerabilidadSistema(session);
                    break;
```

```
                case 3:
                    asignarVunerabilidadAtaque(session);
                    break;

                case 4:
                    obtenerDatosVulnerabilidad(session);
                    break;

                case 5:
                    listarDescripcionVulnerabilidad(session);
                    break;

                case 6:
                    contarVulnerabilidadesRiesgo4(session);
                    break;

                case 7:
                    listarVulnerabilidadesAtaques(session);
                    break;

                case 8:
                    break;

                default:
                    System.out.println("¡¡Debes elegir una opción correcta!!");
            }

        } while (opcion != 8);

        session.close();
        System.out.println("Finalizando la conexion a MySQL");
    }

    /**
     * método que crea una vulnerabilidad
     *
     * @param session
     */
    private static void crearVulnerabilidad(Session session) {
        Transaction trt = session.beginTransaction(); //iniciamos una transación
        //pedimos los datos
        System.out.println("Introduce un nombre de una vulnerabilidad: ");
        String nombre = sc.nextLine();

        System.out.println("Introduce una descripción: ");
        String descripcion = sc.nextLine();

        System.out.println("Introduce el nivel de riego: ");
        Byte nivel = Byte.valueOf(sc.nextLine());

        Vulnerabilidad nuevaVulnera = new Vulnerabilidad(nombre, descripcion,
nivel); //creamos una vulnerabilidad
        session.persist(nuevaVulnera); //guardamos
```

```
        trt.commit();
    }

    /**
     * método que devuelve una vulnerabilidad a partir de un id
     *
     * @param id
     * @param session
     * @return
     */
    private static Vulnerabilidad getVulneraById(int id, Session session) {
        Vulnerabilidad vulnera = session.createQuery("FROM Vulnerabilidad v WHERE
v.id_vulnerabilidad = :idV", Vulnerabilidad.class).setParameter("idV",
id).uniqueResult();
        if (vulnera == null) {
            System.out.println("id no encontrado");
        }
        return vulnera;
    }

    /**
     * método que devuelve un sistema a partir de un id
     *
     * @param id
     * @param session
     * @return
     */
    private static Sistema getSistemaById(int id, Session session) {
        Sistema sistema = session.createQuery("FROM Sistema s WHERE s.id_sistema =
:idS", Sistema.class).setParameter("idS", id).uniqueResult();
        if (sistema == null) {
            System.out.println("Id no encontrado");
        }
        return sistema;
    }

    /**
     * método que asigna una vulnerabilidad a un sistema. La asignación será
     * bidireccional.
     *
     * @param session
     */
    private static void asignarVulnerabilidadSistema(Session session) {
        Transaction trt = session.beginTransaction(); //iniciamos una transación
        System.out.println("Introduce el id de la vulnerabilidad: ");
        int idVulnera = Integer.parseInt(sc.nextLine());
        Vulnerabilidad vulneraEncontrado = getVulneraById(idVulnera, session);
        System.out.println("Introduce el id del sistema: ");
        int idSistema = Integer.parseInt(sc.nextLine());
        Sistema sistemaEncontrado = getSistemaById(idSistema, session);
        if (vulneraEncontrado != null && sistemaEncontrado != null) {
            vulneraEncontrado.setSistemaBidireccional(sistemaEncontrado);
            sistemaEncontrado.setVulnerabilidadBidireccional(vulneraEncontrado);
            session.merge(vulneraEncontrado);
```

```
        }
        trt.commit();
    }

    /**
     * método que devuelve un ataque a partir de un id
     *
     * @param id
     * @param session
     * @return
     */
    private static Ataque getAtaqueById(int id, Session session) {
        Ataque ataque = session.createQuery("FROM Ataque a WHERE a.id_ataque =
:idA", Ataque.class).setParameter("idA", id).uniqueResult();
        if (ataque == null) {
            System.out.println("Id no encontrado");
        }
        return ataque;
    }

    /**
     * método que asigna una vulnerabilidad a un vector de ataque para un día
     * concreto. La asignación será bidireccional. La fecha puede usarse
     * Localdate.now()
     *
     * @param session
     */
    private static void asignarVunerabilidadAtaque(Session session) {
        Transaction trt = session.beginTransaction(); //iniciamos una transación
        System.out.println("Introduce el id de la vulnerabilidad: ");
        int idVulnera = Integer.parseInt(sc.nextLine());
        Vulnerabilidad vulneraEncontrado = getVulneraById(idVulnera, session);
        System.out.println("Introduce el id del ataque: ");
        int idAtaque = Integer.parseInt(sc.nextLine());
        Ataque ataqueEncontrado = getAtaqueById(idAtaque, session);
        if (vulneraEncontrado != null && ataqueEncontrado != null) {
            System.out.println("Introduce el impacto: ");
            Byte impacto = Byte.valueOf(sc.nextLine());
            Permite nuevoPermite = new Permite(vulneraEncontrado,
ataqueEncontrado, impacto, LocalDate.now());
            vulneraEncontrado.setPermiteBidireccional(nuevoPermite);
            ataqueEncontrado.setPermiteBidireccional(nuevoPermite);
            session.persist(nuevoPermite);
        }
        trt.commit();
    }

    /**
     * método que obtiene el nombre, descripción y nivel de riesgo de una
     * vulnerabilidad a partir de su id.
     *
     * @param session
     */
    private static void obtenerDatosVulnerabilidad(Session session) {
```

```
        Transaction trt = session.beginTransaction(); //iniciamos una transación
        System.out.println("Introduce el id de la vulnerabilidad: ");
        int idVulnera = Integer.parseInt(sc.nextLine());

        // Consulta que selecciona solo los campos necesarios
        Object[] resultados = session.createQuery("SELECT v.nombre, v.descripcion,
v.nivel_riesgo FROM Vulnerabilidad v WHERE v.id_vulnerabilidad = :idV",
Object[].class).setParameter("idV", idVulnera).uniqueResult();
        // imprimimos las tres columnas:
        System.out.println("Solución consulta 1: " + resultados[0] + ", " +
resultados[1] + ", " + resultados[2]);
    }

    /**
     * método que lista la descripción de la solución de la vulnerabilidad
     * “Ransomware” (Solución: Implementar copias de …).
     *
     * @param session
     */
    private static void listarDescripcionVulnerabilidad(Session session) {
        Transaction trt = session.beginTransaction(); //iniciamos una transación

        Vulnerabilidad vulneraConsulta = session.createQuery("FROM Vulnerabilidad
v WHERE v.nombre = 'Ransomware'", Vulnerabilidad.class).uniqueResult();
        System.out.println("vulneraConsulta = " + vulneraConsulta.getNombre());
        // Consulta que selecciona solo los campos necesarios
        Object[] resultado = session.createQuery("SELECT s.descripcion FROM
Solucion s WHERE s.vulnerabilidad = :vulnera",
Object[].class).setParameter("vulnera", vulneraConsulta).uniqueResult();
        // imprimimos el resultado:
        System.out.println("Solución consulta 2: " + resultado[0]);
    }

    /**
     * método que cuenta cuántas vulnerabilidades de nivel_riesgo 4 tiene el
     * sistema “Windows 10”
     *
     * @param session
     */
    private static void contarVulnerabilidadesRiesgo4(Session session) {
        Transaction trt = session.beginTransaction(); //iniciamos una transación

        //count devuelve un Long:
        Long totalVulnera = session.createQuery("SELECT COUNT(v) FROM Sistema s
JOIN s.vulnerabilidades v WHERE s.nombre = 'Windows 10' AND v.nivel_riesgo = 4",
Long.class).uniqueResult();
        System.out.println("Solución consulta 3: " + totalVulnera);
    }

    /**
     * método que lista todas las vulnerabilidades con sus ataques permitidos,
pero sólo si el impacto del ataque es mayor o igual a 3
     * @param session
     */
```

```
    private static void listarVulnerabilidadesAtaques(Session session) {
        Transaction trt = session.beginTransaction(); //iniciamos una transación
        List<Object[]> resultados = session.createQuery("SELECT v.nombre,
a.nombre, p.impacto FROM Permite p JOIN p.vulnerabilidad v JOIN p.ataque a WHERE
p.impacto >= 3", Object[].class).list();

        for (Object[] resultado : resultados) {
            System.out.println("Solución consulta 4: " + (String) resultado[0] +
", " + (String) resultado[1] + ", " + (Byte) resultado[2]);
        }
    }
}
```

2. **Entities**

- Ataque.java

```
package entity;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.OneToMany;
import jakarta.persistence.Table;
import java.util.List;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

/**
 *
 * @author alba_
 */

@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
@Table(name = "ataques")
public class Ataque {
    @Id
    private int id_ataque;
    private String nombre;
    private String tipo;

    @OneToMany(mappedBy = "ataque")
    private List<Permite> permites;

    public void setPermiteBidireccional(Permite perm) {
        this.permites.add(perm);
        perm.setAtaqueBidireccional(this); //asignamos el permite al ataque
    }
```

```
}
```

- Permite.java

```
package entity;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.ManyToOne;
import java.time.LocalDate;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

/**
 *
 * @author alba_
 */

@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Permite {
    @Id
// private int id_vulnerabilidad;
    @ManyToOne
    @JoinColumn(name="id_vulnerabilidad")
    private Vulnerabilidad vulnerabilidad;
    @Id
    //private int id_ataque;
    @ManyToOne
    @JoinColumn(name="id_ataque")
    private Ataque ataque;
    private Byte impacto;
    private LocalDate fecha_detectada;

    public void setVulnerabilidadBidireccional(Vulnerabilidad vuln) {
        this.vulnerabilidad = vuln;
    }

    public void setAtaqueBidireccional(Ataque ataque) {
        this.ataque = ataque;
    }
}
```

- Sistema.java

```java
package entity;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.ManyToMany;
import jakarta.persistence.Table;
import java.util.List;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

/**
 *
 * @author alba_
 */

@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
@Table(name = "sistemas")
public class Sistema {
    @Id
    private int id_sistema;
    private String nombre;
    private String version;

    @ManyToMany(mappedBy = "sistemas")
    private List<Vulnerabilidad> vulnerabilidades;

    //creamos un método para asignar una solución a una vulnerabilidad y
viceversa
    public void setVulnerabilidadBidireccional(Vulnerabilidad vuln) {
        this.vulnerabilidades.add(vuln);
    }
}
```

- Solucion.java

```java
package entity;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.OneToOne;
import jakarta.persistence.Table;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
```

```
/**
 *
 * @author alba_
 */
@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
@Table(name = "soluciones")
public class Solucion {
    @Id
    private int id_solucion;

@OneToOne
@JoinColumn(name="id_vulnerabilidad") //ponemos en name como se llame en la
base de datos
// private int id_vulnerabilidad;
private Vulnerabilidad vulnerabilidad; //usamos la clase
    private String descripcion;

    //creamos un método para asignar una vulnerabilidad a una solución y
viceversa
    public void setVulnerabilidadBidireccional(Vulnerabilidad vulnera){
        this.vulnerabilidad = vulnera;
        vulnera.setSolucion(this); //asignamos la vulnerabilidad a la
solución
    }
}
```

- Vulnerabilidad.java

```
package entity;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.JoinTable;
import jakarta.persistence.ManyToMany;
import jakarta.persistence.OneToMany;
import jakarta.persistence.OneToOne;
import jakarta.persistence.Table;
import java.util.List;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.NonNull;
import lombok.RequiredArgsConstructor;

/**
 *
 * @author alba_
 */
```

```java
@Entity
@Data //pone getter, setter y toString
@AllArgsConstructor
@NoArgsConstructor
@Table(name = "vulnerabilidades") //si no coincide el nombre de la Clase con el de la tabla
public class Vulnerabilidad {

    @Id
    //  @GeneratedValue(strategy = GenerationType.AUTO) //autoincrement (poner si da error)
    private int id_vulnerabilidad;
    @NonNull
    private String nombre;
    @NonNull
    private String descripcion;
    @NonNull
    private Byte nivel_riesgo;

    public Vulnerabilidad(String nombre, String descripcion, byte nivel_riesgo) {
        this.nombre = nombre;
        this.descripcion = descripcion;
        this.nivel_riesgo = nivel_riesgo;
    }

    @OneToOne(mappedBy = "vulnerabilidad") //ponemos el nombre de la clase de OneToOne
    private Solucion solucion;

    @OneToMany(mappedBy = "vulnerabilidad")
    private List<Permite> permites;

    @ManyToMany
    @JoinTable(name = "afecta", joinColumns = @JoinColumn(name = "id_vulnerabilidad"), inverseJoinColumns = @JoinColumn(name = "id_sistema")) //nombre de la tabla intermedia
    private List<Sistema> sistemas;

    //creamos un método para asignar una solución a una vulnerabilidad y viceversa
    public void setSolucionBidireccional(Solucion solu) {
        this.solucion = solu;
        solu.setVulnerabilidad(this); //asignamos la solución a la vulnerabilida
    }

    //creamos un método para asignar un sistema a una vulnerabilidad y viceversa
    public void setSistemaBidireccional(Sistema sistem) {
        this.sistemas.add(sistem);
        sistem.setVulnerabilidadBidireccional(this); //asignamos el sistema a la vulnerabilidad
    }
```

```
    //hacemos un método para asignar un permite a una vulnerabilidad y
viceversa
    public void setPermiteBidireccional(Permite perm) {
        this.permites.add(perm);
        perm.setVulnerabilidadBidireccional(this); //asignamos el permite a
la vulnerabilidad
    }
}
```

Ejercicio2.java

```
package examen2eval2025_parte2;
import java.io.IOException;
import java.util.Scanner;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.basex.examples.api.BaseXClient;

public class Ejercicio2 {

    static Scanner sc;
    static String documento = "db:open('universidad')";

    public static void main(String[] args) {
        // Conexión a base X --> ("localhost", 1984, "admin", "abc123")
        try ( BaseXClient conexionBX = new BaseXClient("localhost", 1984, "admin",
"admin");) {

            sc = new Scanner(System.in).useDelimiter("\n");
            String menu = "1. Consulta 1.\n2. Consulta 2\n3. Consulta 3\n4.
Salir\nIntroduzca una opción";
            int opcion;
            do {
                opcion = pedirInt(menu);
                switch (opcion) {
                    case 1:
                        contarProfesores(conexionBX);
                        break;
                    case 2:
                        sacarMediaProfesores(conexionBX);
                        break;
                    case 3:
                        sacarMediaCursos(conexionBX);
                        break;
                }
            } while (opcion != 4);

        } catch (IOException ex) {
            Logger.getLogger(Ejercicio2.class.getName()).log(Level.SEVERE, null,
ex);
```

```
        }
    }

    /**
     * método que imprime todas las consultas
     * @param cliente
     * @param consulta
     */
    private static void imprimirConsultas(BaseXClient cliente, String consulta) {
        try {
            BaseXClient.Query sql = cliente.query(consulta);
            while (sql.more()) { //mientras haya consultas...
                System.out.println(sql.next());
            }
        } catch (IOException ex) {
            Logger.getLogger(Ejercicio2.class.getName()).log(Level.SEVERE, null,
ex);
        }
    }

    private static String pedirString(String mensaje) {
        while (true) {
            try {
                System.out.println(mensaje);
                sc.nextLine();
                return sc.nextLine();
            } catch (Exception ignored) {
            }
        }
    }

    private static int pedirInt(String mensaje) {
        while (true) {
            try {
                System.out.println(mensaje);
                return sc.nextInt();
            } catch (Exception ignored) {
            }
        }
    }

    /**
     * método que cuenta el número de profesores que hay en un departamento dado.
     * Solución: para el departamento Psicología: 2 profesores
     * @param cliente
     */
    private static void contarProfesores(BaseXClient cliente) {
        String departamento = pedirString("Introduce el departamento: ");
        String consulta = "for $x in " + documento + "//departamento where
$x[@nombre=\"" + departamento + "\"] return count($x//profesor)";
        System.out.println("El número de profesores del departamento " +
departamento + " es: ");
        imprimirConsultas(cliente, consulta);
        System.out.println("");
```

```
    }


    /**
     * método que saca la media de antigüedad de los profesores de la facultad de
Ingeniería que trabajan algún Lunes.
     * Solución: 7.666…
     * @param cliente
     */
    private static void sacarMediaProfesores(BaseXClient cliente) {
        String consulta = "avg(for $p in " + documento +
"//facultad[@nombre=\"Ingeniería\"]//profesor \n"
                + "let $h := for $pr in $p\n"
                + "              where $pr/horario[@dia=\"Lunes\"]\n"
                + "              return xs:decimal($pr/@antigüedad)\n"
                + "return $h)";

        System.out.println("La media de antigúedad de los profesores de la
facultad de Ingeniería que trabaja algún lunes es: ");
        imprimirConsultas(cliente, consulta);
        System.out.println("");
    }

    /**
     * método que saca el número medio de cursos ofrecidos por cada facultad.
     * Solución: 3.66…
     * @param cliente
     */
    private static void sacarMediaCursos(BaseXClient cliente) {
        String consulta = "avg(for $x in " + documento +"//facultad\n"
                + "return count($x//curso))";
        System.out.println("El número medio de cursos ofrecidos por cada facultad
es: ");
        imprimirConsultas(cliente, consulta);
    }
}
```

Ejercicio3.java

```
package examen2eval2025_parte2;

import com.mongodb.MongoClient;
import com.mongodb.MongoClientURI;
import com.mongodb.client.AggregateIterable;
import com.mongodb.client.FindIterable;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import static com.mongodb.client.model.Filters.and;
import static com.mongodb.client.model.Filters.eq;
import static com.mongodb.client.model.Filters.gt;
import static com.mongodb.client.model.Filters.size;
```

```
import java.util.Arrays;
import java.util.Scanner;
import org.bson.BsonNull;
import org.bson.Document;
import org.bson.conversions.Bson;

public class Ejercicio3 {

    static Scanner sc;
    static MongoClient mongoClient;
    static MongoCollection<Document> collection;
    static MongoDatabase database;

    public static void main(String[] args) {

        //creamos la conexión
        crearConexion();

        sc = new Scanner(System.in).useDelimiter("\n");
        String menu = "1. Consulta 1.\n2. Consulta 2\n3. Consulta 3\n4. Eliminar
autores\n5. Salir\nIntroduzca una opción";
        int opcion;
        do {
            opcion = pedirInt(menu);
            switch (opcion) {
                case 1:
                    crearConsulta1();
                    break;
                case 2:
                    crearConsulta2();
                    break;
                case 3:
                    crearConsulta3();
                    break;
                case 4:
                    eliminarAutores();
                    break;
            }
        } while (opcion != 5);
    }

    //para seleccionar la colección
    private static void seleccionarColeccion(String coleccion) {
        database = mongoClient.getDatabase("biblioteca");
        collection = database.getCollection(coleccion);
    }

    private static String pedirString(String mensaje) {
        while (true) {
            try {
                System.out.println(mensaje);
                return sc.nextLine();
            } catch (Exception ignored) {
            }
```

```
        }
    }

    private static int pedirInt(String mensaje) {
        while (true) {
            try {
                System.out.println(mensaje);
                return sc.nextInt();
            } catch (Exception ignored) {
            }
        }
    }

    private static void crearConexion() {
        //creamos la conexión MongoDB
        mongoClient = new MongoClient(
                new MongoClientURI(
                        "mongodb://localhost:27017/"
                )
        );
    }

    /**
     * método que Lista los autores que han escrito libros del género Ficción
     * histórica y devolver el nombre, la edad y los premios pero no el ID.
     * Solución: Isabel Allende
     */
    private static void crearConsulta1() {
        seleccionarColeccion("autores");

        Bson filter = eq("genero", "Ficción histórica");
        Bson project = and(eq("nombre", 1L), eq("edad", 1L), eq("premios", 1L),
eq("_id", 0L));

        FindIterable<Document> resultados =
collection.find(filter).projection(project);
        for (Document resultado : resultados) {
            resultado.toString();
            System.out.println(resultado.toString()); //imprimimos los resultados
        }
    }

    /**
     * método que Lista el nombre de los autores mayores de 60 años que han
     * recibido el "Premio Nobel de Literatura". Solución: Gabriel García
     * Márquez
     */
    private static void crearConsulta2() {
        seleccionarColeccion("autores");

        Bson filter = and(Arrays.asList(eq("premios.nombre", "Premio Nobel de
Literatura"), gt("edad", 60L)));
        Bson project = and(eq("nombre", 1L), eq("_id", 0L));
```

```
        FindIterable<Document> resultados2 =
collection.find(filter).projection(project);
        for (Document resultado : resultados2) {
            resultado.toString();
            System.out.println(resultado.toString()); //imprimimos los resultados
        }
    }

    /**
     * método que Crea una agregación que permita obtener la cantidad total de
     * ventas de libros de todos los autores. Solución: {"total_ventas":
     * 324000000}
     */
    private static void crearConsulta3() {
        seleccionarColeccion("autores");

        AggregateIterable<Document> resultados3 =
collection.aggregate(Arrays.asList(new Document("$unwind",
                new Document("path", "$libros")
                        .append("preserveNullAndEmptyArrays", true)),
                new Document("$group",
                        new Document("_id",
                                new BsonNull())
                                .append("sumaTotal",
                                        new Document("$sum", "$libros.ventas"))),
                new Document("$project",
                        new Document("_id", 0L)
                                .append("sumaTotal", 1L))));

        for (Document resultado : resultados3) {
            resultado.toString();
            System.out.println(resultado.toString()); //imprimimos los resultados
        }
    }

    /**
     * método que Elimina los autores cuya nacionalidad sea "Japonesa" y que no
hayan recibido ningún premio.
     * Solución: un elemento eliminado
     */
    private static void eliminarAutores() {
        seleccionarColeccion("autores");

         Bson filter = and(Arrays.asList(eq("nacionalidad", "Japonesa"),
size("premios", 0)));
        collection.deleteMany(filter); //eliminamos todos con las características
especificadas
    }
}
```

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>
    <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
    <property
name="connection.url">jdbc:mysql://localhost:3306/vulnerabilidades_db?
createDatabaseIfNotExist=true&amp;serverTimezone=UTC</property>
    <property name="connection.username">root</property>
    <property name="connection.password"></property>
    <property name="hbm2ddl.auto">validate</property>
    <property name="dialect">org.hibernate.dialect.MySQL8Dialect</property>
    <property name="hibernate.dialect.storage_engine">innodb</property>
    <property name="hibernate.show_sql">true</property>

    <!-- mapping de las clases -->
    <mapping class="entity.Solucion"></mapping>
     <mapping class="entity.Vulnerabilidad"></mapping>
     <mapping class="entity.Ataque"></mapping>
     <mapping class="entity.Permite"></mapping>
     <mapping class="entity.Sistema"></mapping>
  </session-factory>
</hibernate-configuration>
```