Unidad 3. Herramientas de mapeo objeto-relacional (ORM)

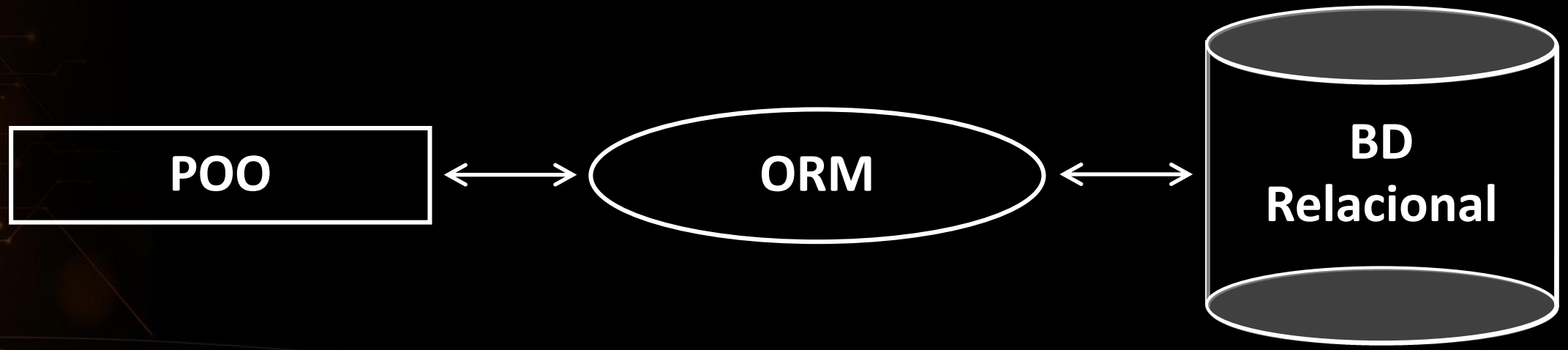
Acceso a Datos. 2do DAM

1. Introducción

ORM (Object Relational Mapping) es una herramienta que permite traducir la lógica de la orientación a objetos a la lógica de una BD relacional, las tablas de nuestra base de datos pasan a ser clases y las filas de las tablas (o registros) objetos que podemos manejar con facilidad.

2. Mapeo objeto-relacional

El mapeo objeto-relacional (Object-Relational Mapping, ORM) es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y la utilización de una base de datos relacional como motor de persistencia. (Wikipedia)



2. Mapeo objeto-relacional

Ventajas:

- Simplifica el desarrollo.
- Abstracción de la BD.
- Independencia de la BD.
- Permiten la portabilidad y escalabilidad del software.

Inconvenientes: Aplicaciones ligeramente más lentas debido a la traducción o mapeo que se realiza con ORM.

3. Herramientas ORM

Diferentes Herramientas ORM son:

- Para Java: Hibérnate, iBatis, Ebean, Torque
- Para .Net: nHibernate, Entity Framework, DataObjects.NET
- Para PHP: Doctrine, Propel , Torpo
- Para Python: SQLObject, Django, Tryton, Storm

4. Hibernate

Hibernate es una herramienta de mapeo objeto-relacional para la plataforma Java (y disponible también para .Net) que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante ficheros declarativos (XML) que permiten establecer estas relaciones.

- Es de software libre bajo licencia GNU LGPL.
- Estable, y con una gran comunidad de desarrolladores.
- Potente respecto a sentencias, APIs, concurrencia, cacheo, etc.
- Soporta la mayoría de BD: Oracle, MySQL, SQL Server, Postgre, etc.

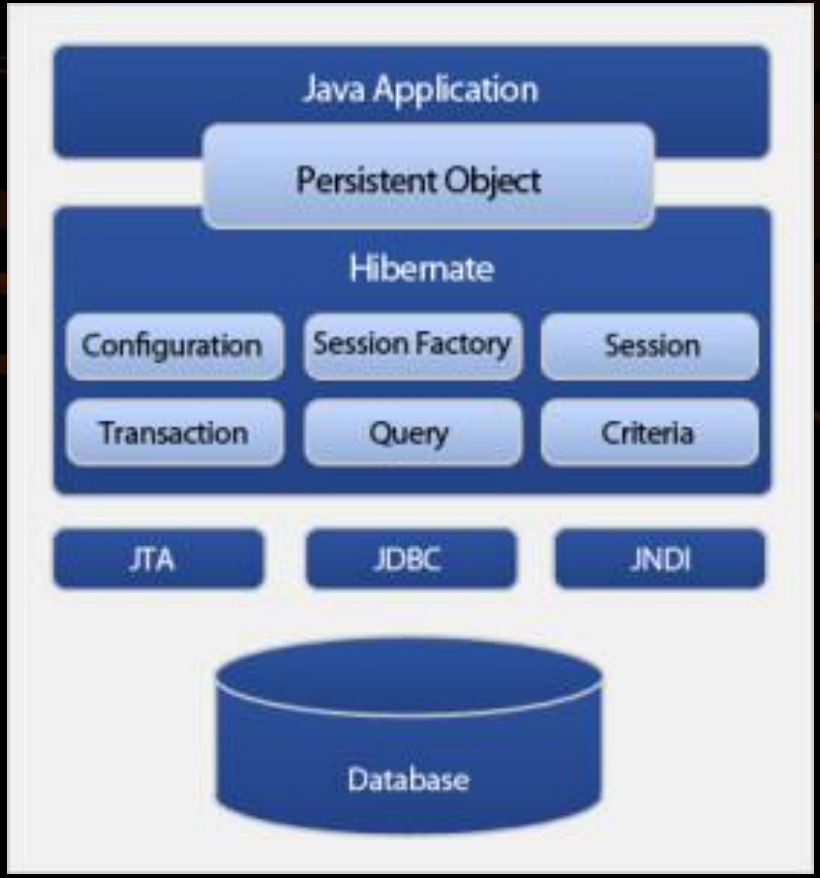
5. Arquitectura Hibernate

Hibernate utiliza varios API de Java existentes:

- JDBC, para bases de datos relacionales.
- Java Transacción API (JTA) y Java Naming and Directory Interface (JNDI), para ser integrado con servidores de aplicaciones J2EE.

Las interfaces de Hibernate son:

- Session Factory:nos permite obtener instancias de Session para realizar operaciones con una BD. (una Session Factory por cada BD)
- Configuration: para crear la configuración de Hibernate que utilizará durante la sesión.
- Query: para realizar consultas con la BD en SQL o HQL (Hibernate Query Languaje).
- Transaction: para controlar las transacciones.
- Criteria: para la creación de filtros con los que hacer las consultas.



6. Instalación y configuración.

Para la utilización de Hibernate:

- Instalaremos el plugin de Hibernate en nuestro IDE.
- Instalaremos en el IDE el driver correspondiente. Esto le permitirá al IDE conectarse como cliente a la BD.

Para su configuración crearemos:

- Un fichero de configuración (cfg.xml) para conectar a la BD a través del driver.
- Un fichero para el mapeo inverso (reveng.xml) encargado de mapear clases a partir de tablas

6. Instalación y configuración.

Con el fichero reveng.xml es posible crear las clases persistentes a partir de las tablas cuyos atributos coinciden con las columnas de las tablas.

Además se crea un fichero xml por cada clase con la etiqueta <hibernate-mapping> que engloba:

- <class name="…" table="…" catalog="…">
- <id...>...</id> para la clave primaria.
- <property...>...</property> para las columans

```
<hibernate-mapping auto-import="true" default-access="property" default-cascade="none" default-lazy="true">
    <class catalog="starwars" dynamic-insert="false" dynamic-update="false" mutable="true"
    name="ad_ud03_p3_package.Films" optimistic-lock="version" polymorphism="implicit" select-before-update="false"
    table="films">
        <id name="id" type="int">
            <column name="id"/>
            <generator class="assigned"/>
        </id>
        <property generated="never" lazy="false" name="episode" optimistic-lock="true" type="string" unique="false">
            <column length="12" name="episode"/>
        </property>
        <property generated="never" lazy="false" name="title" optimistic-lock="true" type="string" unique="false">
            <column length="30" name="title"/>
        </property>
    </class>
</hibernate-mapping>
```

6. Instalación y configuración.

Práctica 01: Instalación y configuración de Hibernate.

Práctica 02: Mapeo de clases con Hibernate.

7. HQL

HQL (Hibernate Query Language) es el lenguaje incorporado a Hibernate para consultar los objetos de nuestro mapa.

A tener en cuenta:

- HQL no es sensible a mayúsculas/minúsculas en palabras reservadas, pero sí lo es en las clases que tengamos definidas.
- No se puede utilizar el * como símbolo de “Todas las columnas”.

7. HQL

HQL (Hibernate Query Language) es el lenguaje incorporado a Hibernate para consultar los objetos de nuestro mapa.

A tener en cuenta:

- HQL no es sensible a mayúsculas/minúsculas en palabras reservadas, pero sí lo es en las clases que tengamos definidas.
- No se puede utilizar el * como símbolo de “Todas las columnas”.

- Link Hibernate HQL.

7. HQL

La siguiente práctica consta de un manual y podrás practicar un poco con el lenguaje HQL.

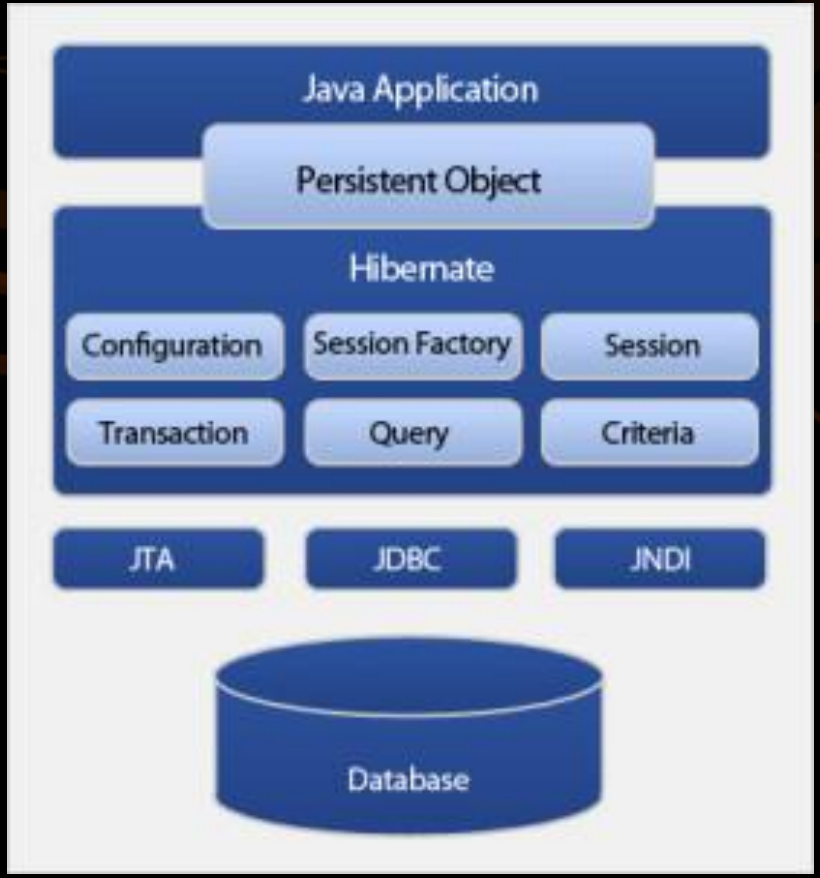
Práctica 03: Consultas HQL.

8. Programando con Hibernate: SessionFactory

- SessionFactory permite instanciar objetos Session para interactuar con BD.
- Debe compartirse entre muchos hilos de ejecución, por lo que utiliza un patrón singleton (solo puede haber un obeto SessionFactory para toda la aplicación).
- Lo conseguimos creando una clase con un static final SessionFactory. (static: compartido por todas las instancias de la clase y final: impide herencia)

La documentación sobre Hibernate y sus librerías la podemos encontrar en:

https://docs.jboss.org/hibernate/orm/current/javadocs/



8. Programando con Hibernate: SessionFactory

- A la derecha podemos ver la creación del patrón singleton con la clase SessionFactoryUtil, y su utilización para crear y cerrar un sesión.

```java
public class SessionFactoryUtil {
    private static final SessionFactory sessionFactory ;
    static {
        try {
        sessionFactory = new Configuration().configure().buildSessionFactory();
        }
        catch (Throwable ex) {
        System.err.println("La iniciación de la SessionFactory ha fallado. " + ex);
        throw new ExceptionInInitializerError (ex) ;
        }
    }
    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

```java
//Obtener la sesión actual
SessionFactory sesionFactoria = SessionFactoryUtil.getSessionFactory();

//Abrir la sesión
Session sesion = sesionFactoria.openSession();


//Cerrar la sesión
sesion.close();
```

```java
//Inicalizamos el entorno de hibernate
Configuration cfg = new Configuration().configure();

//Crear el ejemplar de SessionFactory
SessionFactory sesionFactoria = cfg.buildSessionFactory(
        new StandardServiceRegistryBuilder().configure().build());

//Abrir la sesión
Session sesion = sesionFactoria.openSession();


//Cerrar la sesión
sesion.close();
```

- A la izquierda la creación de una sesión sin el patrón singleton.

8. Programando con Hibernate: Transacciones

- Un objeto Session de Hibernate representa una única unidad de trabajo para un almacén de datos dado y lo abre un ejemplar de SessionFactory.
- Se deben cerrar las sesiones cuando se haya completado el trabajo de la transacción.
- La clase Transaction proporciona 3 métodos:
  - **beginTransaction**() marca el comienzo de una transacción.
  - **commit**() valida la transacción.
  - **rollback**() deshace la transacción.

```
//Comienza la transaccion
Transaction tx = session.beginTransaction() ;
//... aquí vendría el código para actualizar los datos

//Valida la transacción
tx.commit();
```

8. Programando con Hibernate: consultas

- Las consultas también pueden ser de varios tipos:

  - Consultas de recuperación: permiten obtener colecciones de objetos de una base de datos.
  - Consultas "escalares“: devuelven un único objeto.
  - Consultas de actualización: similares a las de SQL: INSERT INTO, DELETE y UPDATE.

8. Programando con Hibernate: consultas de recuperacion

- Son de la clase Query y se crean a partir del objeto Session.
- A partir de la consulta se pueden utilizar los siguientes métodos para recuperar el resultado de la query:
  - ~~**list**()~~: hace una conexión y devuelve en una colección todos los resultados de la consulta. Requiere memoria suficiente y puede ser lento. Está en desuso.
  - ~~**iterate**()~~: devuelve un iterador Java con todos los id's de los objetos y en cada llamada del método iterador.next() se ejecuta una consulta para obtener el objeto correspondiente. Esto supone un mayor número de accesos a la BD, mayor tiempo de procesamiento y mayor carga del servidor, pero requiere menos memoria. Se puede fijar el numero de filas a recuperar en un acceso a la base de datos con .**setFetchSize** (int size). Está en desuso.
  - **getResultsList**(): devuelve un objeto List sin tipado con lo resultados. Es similar al anterior list().
  - **scroll**(): devuelve un objeto ScrollableResults. Este permite recorrer todos los resultados y tiene métodos como **.next**() que devuelve el siguiente objeto en el resultado y **.get**(int i) que devuelve el contenido de la columna i. Así como otros métodos como .get(), .fist(), .last(), .isFirst(), .IsLast(), entre otros. Ha de cerrarse con **.close()**.

8. Programando con Hibernate: consultas de recuperacion

```
Query q = sesion.createQuery("from Dinosaur");
System.out.println("Uso de list()");
List <Dinosaur> dl = q.list();
for(Dinosaur l: dl) System.out.println(l.getName());

System.out.println("Uso de iterate()");
Iterator <Dinosaur> di = q.iterate();
while(di.hasNext()) System.out.println(((Dinosaur) di.next()).getName());

System.out.println("Uso de getResultList()");
dl = sesion.createQuery("from Dinosaur").getResultList();
for(Dinosaur l: dl) System.out.println(l.getName());

System.out.println("Uso de scroll()");
ScrollableResults sc = sesion.createQuery("from Dinosaur").scroll();
while (sc.next()) {
    Dinosaur d = (Dinosaur)sc.get(0);
        System.out.println(d.getName());
}
sc.close();
```

8. Programando con Hibernate: consultas escalares

- Una consulta "escalar" una consulta HQL que devuelve un solo objeto. En este caso podemos simplificar el tratamiento del resultado utilizando UniqueResult():

```
Dinosaur d1 = (Dinosaur)sesion.createQuery("from Dinosaur AS d where d.id = 1").uniqueResult();
System.out.println("Dinosaurio con id=1: "+d1.getName());
```

```
Dinosaurio con id=1: Acrocanthosaurus
```

8. Programando con Hibernate: consultas de recuperación y escalares

- Para acceder a un Set de un objeto se puede hacer de varias formas:
  - Accediendo al objeto y su atributo Set, y recorriéndolo con un iterador:

```
Excavation e = (Excavation)sesion.createQuery("from Excavation AS e where e.name = 'Kaiparowits Formation'").uniqueResult();
Iterator it = e.getDinosaurs().iterator();
System.out.println("Dinosaurios encontardos en: "+e.getName());
while (it.hasNext())
{
    Dinosaur d = (Dinosaur)it.next();
    System.out.println(d.getName());
}
```

  - Extrayendo el Set directamente a un List o a un ScrollableResult para recorrerlo:

```
List<Dinosaur> dl = sesion
        .createQuery("select dinosaurs from Excavation AS e where e.name = 'Kaiparowits Formation'")
        .getResultList();
System.out.println("Dinosaurios encontardos en: Kaiparowits Formation");
for(Dinosaur d: dl)
    System.out.println(d.getName());
```

```
Dinosaurios encontardos en: Kaiparowits Formation
Spinoraptor
Edmontosaurus
Nasutoceratops
```

8. Programando con Hibernate: consultas de actualización

INSERT INTO:

- Su sintaxis es es:
  - INSERT INTO nombreClase (lista propiedades) sentencia-select
- Donde:
  - Solo se soporta la forma INSERT INTO ... SELECT ..., no la forma INSERT INTO ... VALUES. Es decir, solo podemos insertar datos procedentes de otra consulta.
  - La lista de propiedades es análoga a la lista de columnas en la declaración INSERT de SQL.
  - La sentencia-select puede ser cualquier consulta select de HQL válida.
  - La clave primaria id ha de pasarse en ls lista de propiedades, a no ser que la tabla tenga AUTOINCREMENT.

```
Transaction tx= sesion.beginTransaction() ;
String hqlInsert = "INSERT INTO Dinosaur (period, name, height, weight, length) "
                   +"SELECT d.period, d.name, d.height, d.weight, d.length "
                   +"FROM Dinosaur AS d WHERE d.name='Stegosaurus'";

sesion.createQuery(hqlInsert).executeUpdate();
tx.commit();
```

8. Programando con Hibernate: consultas de actualización

UPDATE y DELETE:

- Su sintaxis es es:
  - UPDATE SET {propiedad = nuevovalor} [,...n] + [FROM nombreClase] WHERE condición

```
Transaction tx= sesion.beginTransaction() ;
String hqlInsert = "UPDATE Dinosaur SET height=6, length=8 "
                   +"WHERE name='Stegosaurus'";

sesion.createQuery(hqlInsert).executeUpdate();
tx.commit();
```

  - DELETE nombreClase SET {propiedad = nuevovalor} [,...n] + [FROM nombreClase] WHERE condición
    - {…}: obligatorio al menos una propiedad. [..] opcional más de una
    - [FROM nombreClase]: es opcional, pero solo se puede hacer de una clase.
    - AS: se puede utilizar alias, pero en es e caso todas las propiedades deben utilizarlo.
    - No se pueden utilizar joins. Sí subqueries, pero sólo en WHERE.

```
Transaction tx= sesion.beginTransaction() ;
String hqlInsert = "DELETE Dinosaur "
                   +"WHERE name='Stegosaurus'";

sesion.createQuery(hqlInsert).executeUpdate();
tx.commit();
```

8. Programando con Hibernate: parámetros en las consultas

- Es posible fijar parámetros en las consultas:
  - Con '?' tal y como se hacía en las PreparedStatement. Es necesario indicar un índice ?1, ?2, etc.
  - Con el nombre de un parámetro de la forma :nombre_parámetro. Con las ventajas:
    - son insensibles al orden en que aparecen en la cadena de consulta,
    - pueden aparecer múltiples veces en la misma petición,
    - son auto-documentados.
- No se pueden mezclar en una misma query parámetros de posición y por nombre. Para fijar los parámetros se utilizan los métodos de la interfaz Query (hay algunos más):
  - **SetXXX(**int posición, XXX valor), y **SetXXX(**String nombre, XXX valor) donde XXX puede ser los tipos Character, Date, Double, Integer, String. Se utiliza para fijar el valor de '?' en el primer caso, indicando la posición, o el nombre del parámetro.
  - **SetParameter(**int posición, Object valor), **SetParameter**(String nombre, Object valor), similar al anterior, pero es un método genérico el tipo de parámetros e especifica según el objeto pasado.
  - **SetParameterList**(String nombre, Collection values): fija el parámetro de la consulta como una lista (List) de parámetros.

8. Programando con Hibernate: parámetros en las consultas

```
List<Dinosaur> dl = sesion.createQuery("from Dinosaur AS d where d.id BETWEEN ?1 and ?2")
        .setParameter(1,10).setParameter(2,12).getResultList();
for(Dinosaur d: dl)
    System.out.println("Dinosaurio con id="+d.getId()+": "+d.getName());
for (int i=13;i<16;i++)
{
    Dinosaur d = (Dinosaur) sesion.createQuery("from Dinosaur AS d where d.id = :idDinosaurio")
            .setParameter("idDinosaurio",i).uniqueResult();
    System.out.println("Dinosaurio con id="+i+": "+d.getName());
}
List <Integer> ids = new ArrayList <Integer>(Arrays.asList(16,17,18));
dl = sesion.createQuery("from Dinosaur AS d where d.id in (:listaId)")
        .setParameterList("listaId",ids).getResultList();
for(Dinosaur d: dl)
    System.out.println("Dinosaurio con id="+d.getId()+": "+d.getName());
```

```
Dinosaurio con id=10: Camarasaurus
Dinosaurio con id=11: Carcharodontosaurus
Dinosaurio con id=12: Carnotaurus
Dinosaurio con id=13: Ceratosaurus
Dinosaurio con id=14: Chasmosaurus
Dinosaurio con id=15: Chungkingosaurus
Dinosaurio con id=16: Compsognathus
Dinosaurio con id=17: Corythosaurus
Dinosaurio con id=18: Crichtonsaurus
```

8. Programando con Hibernate: consultas no asociadas

- Si queremos recuperar los datos de una consulta en la que intervienen varias tablas y no tenemos asociada a ninguna clase los atributos que devuelve esa consulta, debemos utilizar el método scroll() que nos devuelve la clase ScrollableResults .

```
ScrollableResults sc = sesion.createQuery("FROM Dinosaur AS d, Paleontologists as p "
        + "where p.excavation.id in (select id from d.excavations)").scroll();
while (sc.next()) {
    Dinosaur d = (Dinosaur)sc.get(0);
    Paleontologists p = (Paleontologists)sc.get(1);
    System.out.println(p.getName()+" estudia el "+d.getName());
}
sc.close();
```

```
Dr. Henry Wu estudia el Ankylosaurus
Dr. Kajal Dua estudia el Pentaceratops
Dr. Kajal Dua estudia el Sinoceratops
Owen Grady estudia el Pentaceratops
Owen Grady estudia el Sinoceratops
Vic Hoskins estudia el Nodosaurus
Billy Brennan estudia el Baryonyx
Katashi Hamada estudia el Baryonyx
Dr. Ellie Sattler estudia el Brachiosaurus
Dr. Ian Malcolm estudia el Majungasaurus
Karen Mitchell estudia el Majungasaurus
Mr. DNA estudia el Camarasaurus
Mr. DNA estudia el Stegoceratops
```

8. Programando con Hibernate: consultas SQL

- Hibernate permite la ejecución de sentencias SQL con **createSQLQuery**():
  - SELECT que deben ser recogidas con un ScrollableResult.
  - Sentencias DML o DDL para manejar la BD. Para ejecutarlas se utiliza **executeUpdate**()

```
tx = sesion.beginTransaction() ;
String sql = "CREATE TABLE paleontologists(\r\n"
        + "  id INTEGER NOT NULL AUTO_INCREMENT,\r\n"
        + "     name varchar(50) not null,\r\n"
        + "    team INTEGER,\r\n"
        + "    id_excavation INTEGER,\r\n"
        + "    PRIMARY KEY (id),\r\n"
        + "    FOREIGN KEY (id_excavation) REFERENCES excavation(id)\r\n"
        + ");";
sesion.createSQLQuery(sql).executeUpdate();
tx.commit();

ScrollableResults sc = sesion.createSQLQuery("SELECT name FROM dinosaur").scroll();
while (sc.next()) {
        System.out.println(sc.get(0));
}
```

```
Acrocanthosaurus
Albertosaurus
Allosaurus
Ankylodocus
Ankylosaurus
Apatosaurus
Archaeornithomimus
Baryonyx
Brachiosaurus
Camarasaurus
Carcharodontosaurus
Carnotaurus
Ceratosaurus
Chasmosaurus
Chungkingosaurus
```

8. Programando con Hibernate: Carga de un objeto Hibernate

Práctica 04: Consultas desde código con Hibernate.

8. Programando con Hibernate: Estados de un objeto Hibernate

- Hibernate define y soporta los siguientes estados de objeto:
- **Transitorio (Transient)**: objeto recién instanciado con el operador new que no está asociado a una Session de Hibernate. Por tanto, no tiene una representación persistente en la base de datos y no se le ha asignado un valor identificador (primary key).
- **Persistente (Persistent)**: el objeto tiene una representación en la base de datos y un valor identificador, bien creado y guardado en la BD, bien cargado de la BD, pero en el ámbito de la sesión. Hibernate detectará cualquier cambio realizado a un objeto en estado persistente y sincronizará el estado con la base de datos cuando se complete la unidad de trabajo.
- **Separado (Detached)**: es un objeto que se ha hecho persistente, pero cuya sesión ha sido cerrada. La referencia al objeto todavía es válida y se puede trabajar con ella, y la instancia separada puede ser reunida a una nueva sesión más tarde (sincronizándola con la BD) haciéndola persistente de nuevo (con todas las modificaciones).

```
//Comienza la transaccion
Transaction tx = sesion.beginTransaction() ;
//... aquí vendría el código para actualizar los datos

Dinosaur d = new Dinosaur();
d.setName("Brontosaurus");
Period p =new Period();
p.setId(2);
d.setPeriod(p);
d.setLength(22);
d.setHeight(15);     // <- Objeto transitorio
sesion.save(d);      // <- Objeto peristente

//Valida la transacción
//tx.commit();

//Cerrar la session
sesion.close();      // <- Objeto separado
```

8. Programando con Hibernate: Carga de un objeto Hibernate

- Para obtener un objeto persistente de la BD utilizaremos los siguientes métodos de la clase Session:
- **Object load(Class theClass, Serializable id)**: devuelve un objeto persistente dada la clase y el identificador.
- **Object load(String entityName, Serializable id):** devuelve un objeto persistente dado el nombre de la clase y el identificador.
- **Object get(Class theClass, Serializable id):** devuelve un objeto persistente dada la clase y el identificador.
- **Object get(String entityName, Serializable id):** devuelve un objeto persistente dado el nombre de la clase y el identificador.

```
sesion = sesionFactoria.openSession();
Dinosaur d68 = (Dinosaur) sesion.load(Dinosaur.class, 68);
System.out.println("Dinosaurio "+d68.getId()+": "+d68.getName());
sesion.close();
```

```
Dinosaurio 68: Velociraptor
```

```
sesion = sesionFactoria.openSession();
ArrayList <Dinosaur> dinosaurios = new ArrayList<Dinosaur>();
int id=1;
Dinosaur dinosaurio;
do{
    dinosaurio = (Dinosaur) sesion.get(Dinosaur.class, id++);
    if(dinosaurio!=null)
        dinosaurios.add(dinosaurio);;
}while (dinosaurio!=null);

for (Dinosaur d: dinosaurios)
    System.out.println(d.getName());
sesion.close();
```

- El método load() lanza una excepción ObjectNotFoundException si la fila no existe. El método get() devuelve null si la fila no existe.

8. Programando con Hibernate: Carga de un objeto Hibernate

- Ejemplo.

```
sesion = sesionFactoria.openSession();
Dinosaur d68 = (Dinosaur) sesion.load(Dinosaur.class, 68);
System.out.println("Dinosaurio "+d68.getId()+": "+d68.getName());
sesion.close();
```

```
Dinosaurio 68: Velociraptor
```

```
sesion = sesionFactoria.openSession();
ArrayList <Dinosaur> dinosaurios = new ArrayList<Dinosaur>();
int id=1;
Dinosaur dinosaurio;
do{
    dinosaurio = (Dinosaur) sesion.get(Dinosaur.class, id++);
    if(dinosaurio!=null)
        dinosaurios.add(dinosaurio);;
}while (dinosaurio!=null);

for (Dinosaur d: dinosaurios)
    System.out.println(d.getName());
sesion.close();
```

```
Acrocanthosaurus
Albertosaurus
Allosaurus
Ankylodocus
Ankylosaurus
Apatosaurus
Archaeornithomimus
Baryonyx
Brachiosaurus
Camarasaurus
Carcharodontosaurus
Carnotaurus
Ceratosaurus
Chasmosaurus
Chungkingosaurus
Compsognathus
Corythosaurus
Crichtonsaurus
Deinonychus
```

8. Programando con Hibernate: almacenamiento, inserción y borrado

- Para almacenamiento, inserción y borrado de objetos en la BD se utilizan los métodos de la clase Session:
  - **save**(): guarda el objeto en la base de datos, convirtiendo un objeto transitorio en persistente. Se creará una nueva entrada en la tabla correspondiente en la BD.
  - **update**(): Actualiza en la BD el objeto pasado como argumento, que previamente había sido cargado con load() o get().
  - **delete**(): Borra de la BD el objeto pasado como argumento, que previamente había sido cargado con load() o get().

```
Transaction tx;
//Almacenamiento de Brontosaurio : 22m, 15m, 15000kg, Jurasico
tx= sesion.beginTransaction() ;
Dinosaur d = new Dinosaur();
d.setName("Brontosaurus");
d.setPeriod((Period) sesion.get(Period.class, 2));
d.setLength(22);
d.setHeight(15);
d.setWeight(15000);
sesion.save(d);
tx.commit();

//Actualización
tx = sesion.beginTransaction() ;
d = (Dinosaur) sesion.get(Dinosaur.class, 10);
d.setHeight(6);
sesion.update(d);
tx.commit();

//Borrado
tx = sesion.beginTransaction() ;
d = (Dinosaur) sesion.get(Dinosaur.class, 69);
sesion.delete(d);
tx.commit();
```