Unidad 3. Bases de datos orientadas a objetos

Acceso a Datos. 2º DAM

1. Introducción

- Las bases de datos orientadas a objetos (BDOO) son aquellas cuyo modelo de datos está **orientado a objetos** y almacenan y recuperan objetos en los que se almacena estado y comportamiento.

- Su origen se debe a que en los modelos clásicos de datos existen problemas para representar cierta información, puesto que aunque permiten representar gran cantidad de datos, las operaciones que se pueden realizar con ellos son bastante simples

1. Introducción

- Las clases utilizadas en un determinado lenguaje de programación orientado a objetos son las mismas clases que serán utilizadas en una BDOO; de tal manera, que no es necesaria una transformación del modelo de objetos para ser utilizado por un SGBDOO.

- De forma contraria, el modelo relacional requiere abstraerse lo suficiente como para adaptar los objetos del mundo real a tablas.

2. OODB: Conceptos

- **Base de datos orientada a objetos** (BDOO): una colección persistente y compatible de objetos definida por un modelo de datos orientado a objetos

- **Modelo de datos orientado a objetos**: Un modelo de datos que captura la semántica delos objetos soportados en la programación orientada a objetos.

- **Sistema Gestor de Bases de Datos Orientadas a Objetos** (SGBDOO): El gestor de una base de datos orientada a objetos.

2. OODB: Limitaciones de las bases de datos relacionales

- Pobre representación de las entidades del ‘mundo real’.
- Sobrecarga y poca riqueza semánticas.
- Soporte inadecuado para las restricciones de integridad y empresariales
- Estructura de datos homogénea
- Operaciones limitadas
- Dificultades para gestionar las consultas recursivas
- Desadaptación de impedancias
- Problemas asociados a la concurrencia, cambios en los esquemas y el inadecuado acceso navegacional.
- No ofrecen soporte para tipos definidos por el usuario (sólo dominios)

2. OODB: Escenarios BDOO

- El uso de BDOO es más conveniente si se presenta en alguno de los siguientes escenarios:
  - **Un gran número de tipos de datos diferentes**
  - **Un gran número de relaciones entre los objetos**
  - **Objetos con comportamientos complejos**

2. OODB: BDOO vs BDR

Mientras que en una BDR los datos a almacenar se almacenan representados en tablas, en un BDOO los datos se almacenan como objetos.

Un objeto en BDOO como en POO es una **entidad identificable unívocamente** que describe tanto el **estado** como el **comportamiento** de una entidad del ‘mundo real’.

El estado de un objeto es descrito mediante **atributos** mientras que su comportamiento es definido mediante **métodos**.

2. OODB: Características BDOO

- **Objetos**: cada entidad del mundo real se modela como un objeto.

- La forma de identificar objetos es mediante un identificador de objetos (OID, Object Identifier), único para cada objeto.
  - Los OID son independientes del contenido. Es decir, si un objeto cambia los valores de atributos, sigue siendo el mismo objeto con el mismo OID. Si dos objetos tienen el mismo estado pero diferentes OID, son equivalentes pero tienen identidades diferentes.

- **Encapsulamiento**: cada objeto contiene y define procedimientos (métodos) y la interfaz mediante la cual se puede acceder a él y otros objetos pueden manipularlo.
  - La mayoría delos SGBDOO permite el acceso directo a los atributos incluyendo operaciones definidas por el propio SGBDOO las cuales leen y modifican los atributos para evitar que el usuario tenga que implementar una cantidad considerable de métodos cuyo único propósito sea el de leer y escribir los atributos de un objeto

2. OODB: Características BDOO

- Otros conceptos utilizados de la misma manera que en la POO son:
  - Clases
  - Herencia simple, múltiple y repetida (a través de otra clase que también hereda).
  - Polimorfismo, o capacidad de una función de recibir por parámetro valores de distinto tipo, que podrán ser:
    - de operación (método).
    - de inclusión (redefinición de métodos, su comportamiento).
    - paramétrico (una única definición de la función).
    - ligadura tardía (latebinding): la vinculación método-parámetros se realiza en tiempo de ejecución.
    - sobrecarga (overloading): mismo método y diferentes parámetros.
    - suplantación o anulación (overriding): sobreescritura de un método por una clase hija.
  - Objetos complejos

2. OODB: Manifiesto Malcolm Atkinson

- En 1989 se hizo el Manifiesto de los sistemas de base de datos orientados a objetos el cual propuso trece características obligatorias para un SGBDOO y cuatro opcionales.
- Las trece características obligatorias estaban basadas en dos criterios: debía tratarse de un sistema orientado a objetos y un SGBD.

2. OODB: Manifiesto Malcolm Atkinson

- Características obligatorias de orientación a objetos:
  1. Deben soportarse objetos complejos.
  2. Deben soportarse mecanismos de identidad de los objetos (OID – Object Identifier).
  3. Debe soportarse la encapsulación: solo se tendrá acceso a la interfaz de los métodos.
  4. Deben soportarse los tipos o clases.
  5. Los tipos o clases deben ser capaces de heredar de sus ancestros
  6. Debe soportar sobrecarga.
  7. El DML debe ser computacionalmente complejo
  8. El conjunto de todos los tipos de datos debe ser ampliable

2. OODB: Manifiesto Malcolm Atkinson

- Características obligatorias de SGBD:
  9. Debe proporcionarse persistencia a los datos (la información se mantendrá al cerrar a aplicación).
  10. El SGBD debe ser capaz de gestionar bases de datos de muy gran tamaño
  11. El SGBD debe soportar concurrencia (acceso simultaneo a la BD).
  12. El SGBD debe ser capaz de recuperarse de fallos hardware y software
  13. El SGBD debe proporcionar una forma simple de consultar los datos.

2. OODB: Manifiesto Malcolm Atkinson

- Características opcionales:
  - Herencia múltiple
  - Comprobación de tipos e inferencia (conocimiento) de tipos
  - Sistema de base de datos distribuido
  - Soporte de versiones.

2. OODB: Ventajas de las BDOO

- Mayor capacidad de modelado: La utilización de objetos permite representar de una forma más natural los datos que se necesitan guardar.
- Extensibilidad: Se pueden construir nuevos tipos de datos a partir de los existentes.
- Lenguaje de consulta más expresivo: El lenguaje de consultas es navegacional de un objeto al siguiente.
- Soporte a transacciones largas
- Adecuación a las aplicaciones avanzadas de base de datos: CASE, CAD, multimedia
- Mayores prestaciones

2. OODB: Inconvenientes de las BDOO

- Falta de un modelo de datos universal: La mayoría carecen de base teórica
- Falta de experiencia: El uso es todavía relativamente limitado
- Falta de estándares: No hay estándar como en SQL, aunque hay un estándar de facto (OQL)
- Competencia: SGBDR y SGBDOR
- La optimización de consultas compromete la encapsulación: Para optimizar se requiere conocer la implementación.
- Complejidad
- Falta de soporte a las vistas
- Falta de soporte a la seguridad

2. OODB: ODMG: el estándar de facto para modelos de objetos

- ODMG (Object Database Management Group) es un grupo de representantes de la industria de bases de datos el cual fue concebido en 1991 con el objetivo de definir estándares para los SGBDOO.

- Uno de sus estándares, el cual lleva el mismo nombre del grupo (ODMG), es el del modelo para la semántica de los objetos de una base de datos.

2. OODB: ODMG 3.0

- La última versión del estándar, ODMG 3.0, propone los siguientes componentes principales de la arquitectura ODMG para un SGBDOO:
  - Modelo de objetos
  - Lenguaje de definición de objetos (ODL, Object Definition Language)
  - Lenguaje de consulta de objetos (OQL, Object Query Language)
  - Conexión con los lenguajes C++, Smalltalk y Java (al menos)

2. OODB: ODMG 3.0

- ODMG especifica los siguientes tipos de objetos literales:
  - **Átomicos**: boolean, short, long, unsigned long, unsigned short, float, doublé, char, string, enum, octet.
  - **Estructuras**: date, time, timestamp, interval.
  - Colecciones:
    - **set<tipo>**: colección desordenada sin duplicados.
    - **bag<tipo>**: colección desordenada con duplicados.
    - **list<tipo>**: colección ordenada con duplicados.
    - **array<tipo>**: colección ordenada accesible por posición.
    - **dictionary<clave**, valor>: colección ordenada accesible por clave.
- La declaración de un objeto se define con **class**.
- Todos ellos pueden ser implementados en Java.

2. OODB: Lenguaje ODL

- ODL es un lenguaje para definir la especificación de los tipos de objetos para sistemas compatibles con ODMG. ODL es el equivalente de DDL (Data Definition Languaje) de los SGBD tradicionales. Define los atributos y las relaciones entre tipos y especifica la signatura de las operaciones.

- Su principal objetivo es el de facilitar la portabilidad de los esquemas entre sistemas compatibles al mismo tiempo que proporciona interoperabilidad entre distintos SGBD.

- La sintaxis de ODL extiende el lenguaje de definición de interfaces (IDL) de la arquitectura CORBA.

En nuestro caso no veremos ODL, ya que manejaremos la BDOO a través código Java.

```
class C1 {
   attribute byte b;
   attribute char c;
   attribute short s;
   attribute int i;
   attribute long l;
   attribute double d;
   attribute oid o;
   constraint<notnull> on s;
   constraint<unique> on  i;
};
class C {
  attribute char str[];
  attribute int i_a[3][4][8];
  attribute C1  l_c1;
  attribute C1 *o_c1_2[];
  attribute set<int>  i_lset;
  attribute bag<C1 *> o_c1_lbag;
  attribute array<C1 *> o_c1_larr;
};
```

2. OODB: Lenguaje OQL

- OQL es un lenguaje declarativo del tipo de SQL que permite realizar consultas sobre bases de datos orientadas a objetos, incluyendo primitivas de alto nivel para conjuntos de objetos y estructuras.

- Está basado en SQL-92, proporcionando un súper conjunto de la sentencia SELECT.

- OQL no posee primitivas para modificar el estado de los objetos, ya que éstas se deben realizar a través de los métodos que dichos objetos poseen. La sintaxis básica de OQL es una estructura SELECT...FROM...WHERE..., como en SQL.

3. Neodatis

- NeoDatis Object Database:
  - Es una base de datos orientada a objetos de código abierto que funciona con JAVA, .net, Groovy y Android. Los objetos se pueden almacenar y recuperar con una sola línea de código, sin tener que mapearlos a tablas.

  - http://neodatis.wikidot.com/

3. Neodatis: Creación, apertura, almacenamiento y borrado.

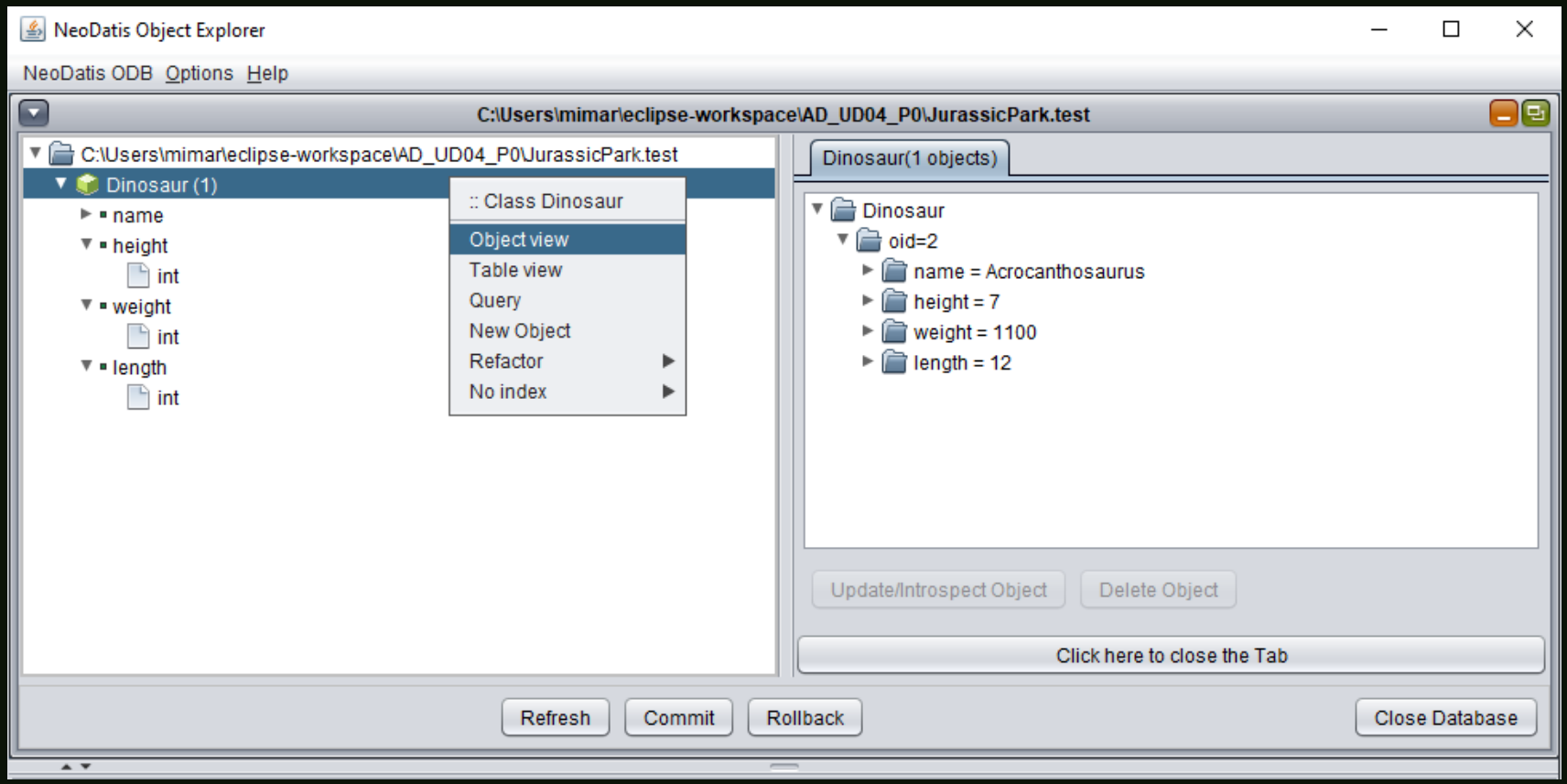
- La apertura de una BDOO Neodatis se hace mediante el método **open**(String) de la clase **ODBFactory**, que devuelve un objeto ODB.
- Después, podemos almacenar objetos el la BD creada con el método .**store**(Object) de la clase ODB.
- Si el objeto has sido cargado y modificado, para que los cambios surjan efecto en la BD se utilizará **.store**() y a continuación **.commit**().
- Si queremos elimiar un objeto se utiliza el método .**delete**()
- Si queremos obtener todos los objetos de un clase de la BD se utiliza el método .**getObjects**(The_Cass.class) de de la clase ODB.
- Objects implementa la interfaz Collection que es un Iterable, y la forma de recorrerlos es como en un iterador con los métodos **hasNext**() y **next**().
- Si queremos obtener el primer objeto que devuelva, utilizamos **.getFirst**().
- El método getObjectFromOID() de la clase ODB devuelve un objeto a partir de su identificador OID.
- Es posible obtener un objeto OID que contiene el identificador de un objeto:
  - Al utilizar el método .store().
  - Mediante el método .getObjectID().
  - Creándolo mediante OIDFactory.buildObjectOID(num_id);
- Para cerrar la BD utilizamos el método .**close**().

3. Neodatis: Creación, apertura, almacenamiento y borrado.

```java
//Creación/apertura de la BD.
ODB odb = ODBFactory.open("JurassicPark.test");

Dinosaur d = new Dinosaur("Acrocanthosaurus",7,1100,12);

//Almacenamiento de objetos en la BD.
OID oid_d = odb.store(d);
odb.commit();
//Obtener todos los objetos de una clase
Objects <Dinosaur> objetos = odb.getObjects(Dinosaur.class);
while (objetos.hasNext()) {
    Dinosaur dn = objetos.next();
    System.out.println("Dinorasuio "+odb.getObjectId(dn)+": "+dn.toString());
}

//Obtener un objeto a partir de su OID y modificarlo
OID oid_d2 = OIDFactory.buildObjectOID(2);
Dinosaur d2 = (Dinosaur)odb.getObjectFromId(oid_d2);
d2.setLength(10);
odb.store(d2);
odb.commit();

//Borrado de un objeto
odb.delete(d2);

//Cerrar la base de datos.
odb.close();
```

3. Neodatis: explorador

- Para poder explorar el contenido de la BD creada con Neodatis, éste ofrece un navegador accesible:
  - En Windows: desde el bat odb-explorer.bat o haciendo doble click en el propio .jar.
  - En Linux: desde el fichero obb-explorer.sh.



3. Neodatis:

- **Práctica 1: Instalación de Neodatis.**
- **Práctica 2: Creación de una base de datos con Neodatis**

3. Neodatis: consultas

- Las consultas se realizan mediante la clase **CriteriaQuery**(TheClass.class, Where.equal(atributo,valor))
- Que devuelve un objeto Iquery al cual se puede aplicar un order by **(.orderByAsc**(), **.orderByDesc**()**),** y que se pasa al método getObjects del la clase ODB para obtener los resultados.
- La consulta también se puede realizar mediante odb.criteriaQuery().
- Si no se encuentran resultados se lanzará una excepción **IndexOutOfBoundsException**.

```
IQuery consulta = new CriteriaQuery(Dinosaur.class, Where.equal("height",10));
IQuery mismaConsulta = odb.criteriaQuery(Dinosaur.class, Where.equal("height",10));

consulta.orderByAsc("name");

try {
    Objects <Dinosaur> objetos = odb.getObjects(consulta);
    while (objetos.hasNext()) {
        Dinosaur dn = objetos.next();
        System.out.println("Dinosaurio "+odb.getObjectId(dn)+": "+dn.toString());
    }
}catch(IndexOutOfBoundsException e)
{System.out.println("Ningun dinosaurio encontrado");}
```

```
Dinosaurio 7: Apatosaurus;10;4100;6;
Dinosaurio 9: Baryonyx;10;3400;11;
Dinosaurio 31: Herrerasaurus;10;2100;5;
Dinosaurio 32: Homalocephale;10;2000;10;
Dinosaurio 44: Nigersaurus;10;2500;4;
Dinosaurio 53: Pteranodon;10;2000;3;
```

3. Neodatis: operadores de comparacion.

- CriteriaQuery admite un objeto **Icriterion** que se define de la misma manera mediante Where. Y que luego puede pasarse a CriteriaQuery:

  CriteriaQuery(TheClass.class, Icriterion)

- Icriterion admite varios tipos de filtrado:

```
ICriterion criterio = Where.equal("length",15);

IQuery consulta = new CriteriaQuery(Dinosaur.class,criterio);

//Filtrados mediante Where.
criterio = Where.like("name","T%");      //like:String similar a un patrón
criterio = Where.gt("weight",1000);      //gt: mayor que     (>)
criterio = Where.ge("height",5);         //ge: mayor o igual (>=)
criterio = Where.lt("weight",1000);      //lt: menor que     (<)
criterio = Where.le("height",5);         //le: menor o igual (<=)
criterio = Where.isNull("period");       //isNull: atributo nulo
criterio = Where.isNotNull("period");    //isNotNull: atributo no nulo

//contain: comprueba si un array o na colección contiene un elemento
//OJO: no funciona con la instancia new CriteriaQuery()
criterio =  Where.equal("name","Candeleros Formation");
consulta = odb.criteriaQuery(Excavation.class,criterio);
Excavation ex = (Excavation) odb.getObjects(consulta).getFirst();
criterio = Where.contain("excavation",ex);

//sizeEq, sizeGt, sizeGe, sizeLt, sizeLe: comprueba el tamaño de un array o lista
```

3. Neodatis: expresiones lógicas.

- Icriterion también admite expresiones lógicas mediante **And**(), **Or**(), o N**ot**().

```
ICriterion criterio = new And().add(Where.gt("weight",1000))
                               .add(Where.ge("height",5));

criterio = new Or().add(Where.lt("weight",1000))
                   .add(Where.le("height",5));

criterio =  new Not(Where.like("name","T%"));
```

3. Neodatis:

- **Práctica 3: Inserción en una base de datos con Neodatis y consultas**

3. Neodatis: acceder a los atributos.

- Para acceder a los atributos de un objeto en lugar de al objeto en sí se utiliza el método **getValues()** al que se le pasa la clase y los campos **.field()** a extraer, y que devuelve un objeto Values.

- Values es iterable. Al objetener un objeto con next() será del tipo **ObjectValues** y podremos acceder a los campos de la consulta:
  - **Por nombre con getByAlias()**
  - **Por índice con getByIndex().**

```
Values valores = odb.getValues(new ValuesCriteriaQuery(Dinosaur.class)
                .field("name").field("weight"));

try {
    while (valores.hasNext()) {
        ObjectValues valorObjetos = (ObjectValues)valores.next();
        System.out.println("Dinosaurio "+valorObjetos.getByAlias("name")
                           +". Peso: "+valorObjetos.getByIndex(1));
    }

}
catch(IndexOutOfBoundsException e)
{System.out.println("Ningun dinosaurio encontrado");}
```

3. Neodatis: consultas de agrupación.

- Neodatis permite las típicas consultas de agrupación con las que en SQL se utiliza la clausula GROUP BY como son: SUM, MAX, MIN y AVG.
- Para ello basta con utilizar las funciones **sum**(), **max**(), **min**() o **avg**() con la consulta deseada. Deben utilizarse con un atributo.
- El resultado de la consulta se recoge en un objeto Values, y accedemos al único valor mediante los métodos next() o nextValues() (sin diferencias).
- **Nota**: avg() devuelve ArithmeticException si el resultado no está redondeado, lo que obliga a dividir el resultado de sum() entre count() para obtener el resultado.

```
ICriterion criterio = Where.equal("length",7);
Values valores = odb.getValues(new ValuesCriteriaQuery(Dinosaur.class,criterio)
                .sum("weight"));

try {
    ObjectValues valorObjetos = valores.nextValues();
    BigDecimal valor = (BigDecimal)valorObjetos.getByAlias("weight");
    System.out.println("Peso de los dinosaurios con 7m de largo: "+valor);
}
catch(IndexOutOfBoundsException e)
{System.out.println("Ningun dinosaurio encontrado");}
```

```
Peso de los dinosaurios con 7m de largo: 5000
```

3. Neodatis: consultas de agrupación.

- Además están los siguientes métodos, que también se utilizan agrupando u atributo:
  - **sublist**() **que** devuelve una sub-lista de la colección.
  - **size**(): que devuelve el tamaño de la colección.
  - **groupBy**(): que agrupa la consulta según un atributo.

```
Values valores = odb.getValues(new ValuesCriteriaQuery(Dinosaur.class).field("length")
                .sum("weight").groupBy("length"));
try {
    while (valores.hasNext()) {
    ObjectValues valorObjetos = valores.nextValues();
    BigDecimal peso = (BigDecimal)valorObjetos.getByAlias("weight");
    Integer largo = (Integer)valorObjetos.getByAlias("length");
    System.out.println("Peso de los dinosaurios con "+largo+"m de largo: "+peso);
    }
}
catch(IndexOutOfBoundsException e)
{System.out.println("Ningun dinosaurio encontrado");}
```

```
Peso de los dinosaurios con 1m de largo: 7300
Peso de los dinosaurios con 2m de largo: 7200
Peso de los dinosaurios con 3m de largo: 9900
Peso de los dinosaurios con 4m de largo: 6700
Peso de los dinosaurios con 5m de largo: 11900
Peso de los dinosaurios con 6m de largo: 15500
Peso de los dinosaurios con 7m de largo: 5000
Peso de los dinosaurios con 8m de largo: 7000
Peso de los dinosaurios con 9m de largo: 9200
Peso de los dinosaurios con 10m de largo: 26400
Peso de los dinosaurios con 11m de largo: 23800
Peso de los dinosaurios con 12m de largo: 14300
Peso de los dinosaurios con 13m de largo: 10900
Peso de los dinosaurios con 14m de largo: 3700
Peso de los dinosaurios con 15m de largo: 4900
```

3. Neodatis: restricciones de clave.

- Neodatis no permite la creación de restricciones de clave primaria o alternativa.
- Lo que sí permite es la creación de índices únicos o no, para acelerar loas búsquedas para atributos de objeto de una colecció.
- Si se utiliza un índice único con **addUniqueIndexOn**(), se puede utilizar con el mismo propósito que para la creación de claves primarias o alternativa.

```
//Los índices pueden crearse antes de hacer una búsqueda,
//o para respetar claves, antes de hacer un guardado.
if (!odb.getClassRepresentation(Dinosaur.class).existIndex("name-index"))
{   String [] fieldIndex = {"name"};
    ClassRepresentation represntacionClase = odb.getClassRepresentation(Dinosaur.class);
    represntacionClase.addUniqueIndexOn("name-index", fieldIndex, true);
}

IQuery consulta = new CriteriaQuery(Dinosaur.class, Where.equal("name","Tyrannosaurus Rex"));

Dinosaur d = (Dinosaur) odb.getObjects(consulta).getFirst();
System.out.println("Dinosaurio "+odb.getObjectId(d)+": "+d.toString());

Dinosaur d2 = new Dinosaur("Tyrannosaurus Rex",8,3900,10);
try {
    odb.store(d2);
}
catch(ODBRuntimeException e)
{System.out.println(e.getMessage());
System.out.println(e.getCause());}
```

```
Creating index name-index on class Dinosaur - Class has already 67 Objects. Updating index
name-index : loading 67 objects from database
name-index : 67 objects loaded
name-index created!
Dinosaurio 67: Tyrannosaurus Rex;8;3800;10;

Version=1.9.30 , Build=689, Date=10-11-2010-08-21-21, Thread=main
NeoDatisError:1067:Error while managing index name-index
StackTrace:
org.neodatis.btree.exception.DuplicatedKeyException: Tyrannosaurus Rex
```

3. Neodatis:

- **Práctica 4: Creación de restricciones y consultas**