

Contenidos

0. Introducción.....	1
1. Ejemplo de almacenamiento y recuperación de datos. Proyecto UD4_Neodatis_Jugadores.....	2
Actividad 4.10. Crear base de datos Países y Jugadores.....	5
2. Ejemplo de acceso a objetos usando su OID. Proyecto UD4_Neodatis_Jugadores_OID.....	6
3. Ejemplos de consultas con criterios. Proyectos UD4_Neodatis_ConsultasCriterios y UD4_Neodatis_ConsultasExpreLogicas.....	6
Actividad 4.11. Consulta de valores sobre base de datos Países y Jugadores.....	8
4. Ejemplo de actualización y eliminación de datos. Proyecto UD4_Neodatis_ActualizayElimina	8
Actividad 4.12. Consulta de valores sobre base de datos Países y Jugadores.....	10
5. Ejemplo consultas con funciones. Proyecto UD4_Neodatis_ConsultasConFunciones	10
6. Ejemplo consultas con JOINS. Proyecto UD4_Neodatis_VistasDinamicas.....	11
Actividad 4.13. Consulta de valores sobre base de datos Países y Jugadores.....	13
7. Ejemplo de creación de BD NeoDatis desde modelo relacional. Proyecto UD4_Neodatis_Profesores	13
Actividad 4.14. Crear BD NeoDatis VentasArticulos.	22
8. Modelo Cliente-Servidor de la BD NeoDatis	22

0. Introducción.

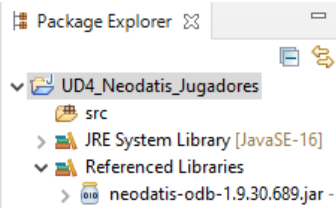
Los ejemplos se van a realizar con la base de datos orientada a objetos NeoDatis y empleando el IDE Eclipse. Descargo el fichero de la última versión 'Neodatis-odb-1.9.30.689.zip' desde el link => <http://Neodatis.wikidot.com/downloads>. Descomprimo el archivo y localizo el archivo **Neodatis-odb-1.9.30.689.jar** que tendré que incluir en el proyecto Java, que puedo implementar usando el IDE Eclipse o Netbeans. En la carpeta doc podemos encontrar documentación sobre esta base de datos.

> UD4 - Bases de datos ORyOO > Parte2_BDOO > SW_BBDD NeoDatis > neodatis-odb-1.9.30.689 > neodatis-odb-1.9.30-689			
Nombre	Fecha de modificación	Tipo	Tamaño
doc	10/11/2010 8:21	Carpeta de archivos	
lib	10/11/2010 8:21	Carpeta de archivos	
maven	10/11/2010 8:21	Carpeta de archivos	
src	10/11/2010 8:21	Carpeta de archivos	
LICENSE.ASM.txt	10/11/2010 8:21	Documento de te...	2 KB
license.txt	10/11/2010 8:21	Documento de te...	27 KB
neodatis-odb-1.9.30.689.jar	10/11/2010 8:21	Executable Jar File	754 KB
odb-explorer.bat	10/11/2010 8:21	Archivo por lotes ...	1 KB
odb-explorer.sh	10/11/2010 8:21	Archivo SH	1 KB
readme.txt	10/11/2010 8:21	Documento de te...	2 KB

Además del fichero jar de NeoDatis, en esta carpeta tenemos el fichero **odb-explorer** que corresponde a la aplicación NeoDatis Object Explorer, nos va a permitir acceder a los datos y otras operaciones de gestión de la base de datos.

1. Ejemplo de almacenamiento y recuperación de datos. Proyecto UD4_Neodatis_Jugadores.

Como se menciona en la wiki de NeoDatis <http://neodatis.wikidot.com/> las operaciones para hacer la información persistente y recuperarla, al hacerse directamente como objetos, se lleva a cabo con pocos pasos. Para probar como se realiza el almacenamiento y recuperación de datos en NeoDatis, creo el proyecto ejemplo de nombre UD4_NeoDatis_Jugadores. Incluyo en el classpath el fichero **NeoDatis-odb-1.9.30.689.jar**



Empiezo por crear en el proyecto la clase Jugador, con 4 atributos, el constructor vacío, el constructor con los 4 atributos, y los correspondientes métodos get y set para acceder a sus atributos. Esta clase define las instancias de los objetos que vamos a almacenar y recuperar en la BD.

```
public class Jugador {

    private String nombre;
    private String deporte;
    private String ciudad;
    private int edad;

    public Jugador() {
    }

    public Jugador(String nombre, String deporte, String ciudad, int edad) {
        this.nombre = nombre;
        this.deporte = deporte;
        this.ciudad = ciudad;
        this.edad = edad;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getNombre() {
        return nombre;
    }

    public void setDeporte(String deporte) {
        this.deporte = deporte;
    }

    public String getDeporte() {
        return deporte;
    }

    public void setCiudad(String ciudad) {
        this.ciudad = ciudad;
    }

    public String getCiudad() {
        return ciudad;
    }

    public void setEdad(int edad) {
        this.edad = edad;
    }

    public int getEdad() {
        return edad;
    }

} // fin clase Jugador
```

Después añado la clase que incluye el método main() desde la que se van a probar las operaciones básicas con la BD:

- Abrir la base de datos => método open(String nombreBD) de la clase ODBFactory que instancia un objeto ODB
- Almacenar objetos => método store(Objeto) de la clase ODB
- Recuperar objetos => método getObjects(nombreClase.class) de la clase ODB, que recupera los objetos como una colección de la clase que se especifique. Este método recibe una clase y devuelve un objeto del tipo Objects que implementa el interfaz Collection.
- Cerrar la base de datos => método close() de la clase ODB.

```
// Operaciones básicas sobre Neodatis
import org.neodatis.odm.ODM;
import org.neodatis.odm.ODMFactory;
import org.neodatis.odm.Objects;

public class UD4_Neodatis_Jugadores {

    public static void main(String[] args) {
        // Crear varias instancias del objeto de clase Jugador
        Jugador j1 = new Jugador("Maria", "voleibol", "Madrid", 14);
        Jugador j2 = new Jugador("Miguel", "tenis", "Madrid", 15);
        Jugador j3 = new Jugador("Mario", "baloncesto", "Guadalajara", 15);
        Jugador j4 = new Jugador("Alicia", "tenis", "Madrid", 14);

        // Abrir BD
        ODM odb = ODMFactory.open("neodatis.test");

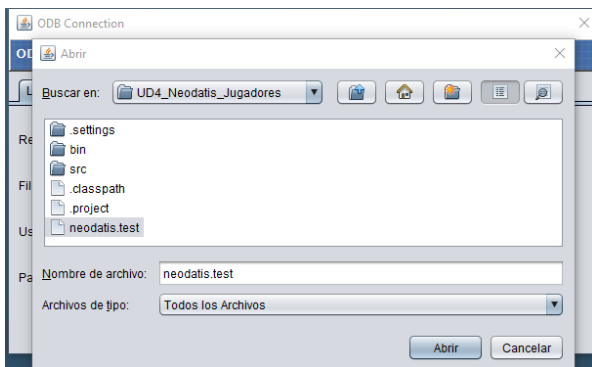
        // Almacenamos los objetos jugador en la base de datos
        odb.store(j1);
        odb.store(j2);
        odb.store(j3);
        odb.store(j4);

        // Recuperamos todos los jugadores
        // OJO. Objects no es la clase Object. Objects implementa la interface Collection
        Objects<Jugador> objetos = odb.getObjects(Jugador.class);
        System.out.printf("%d Jugadores: %n", objetos.size());
        int i = 1;
        // Recorre la colección de objetos
        while(objetos.hasNext()){
            Jugador jug = objetos.next(); // Asigna cada elemento de la colección a una instancia de la clase Jugador
            System.out.printf("%d: %s, %s, %s %n", i++, jug.getNombre(), jug.getDeporte(), jug.getCiudad(), jug.getEdad());
        }
        odb.close(); // Cerrar BD
    } // fin main
} // fin clase
```

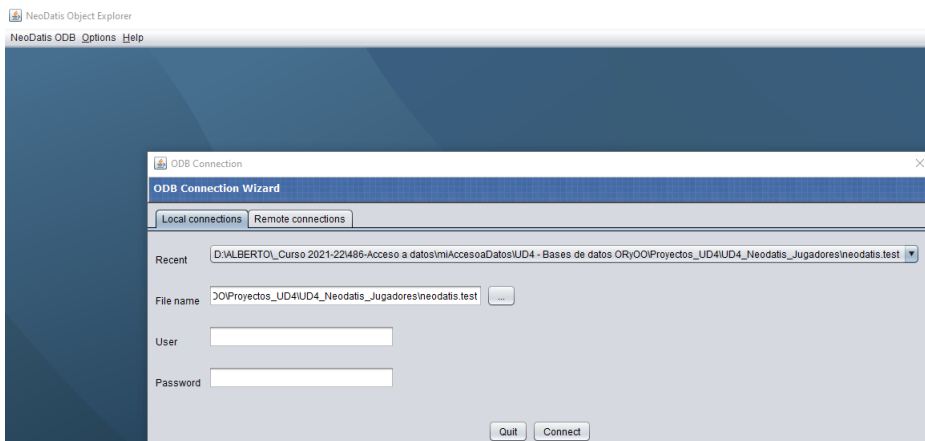
Al no tratarse de una base de datos relacional, en la que en una tabla no puede haber dos filas con la misma clave primaria, en esta base de datos cada vez que se ejecuta el programa ejemplo, se puede comprobar que se vuelven a insertar nuevos objetos, aunque tengan los mismos datos. Al ejecutar el programa por segunda vez, la salida es la siguiente:

```
8 Jugadores:
1: Maria, voleibol, Madrid
2: Miguel, tenis, Madrid
3: Mario, baloncesto, Guadalajara
4: Alicia, tenis, Madrid
5: Maria, voleibol, Madrid
6: Miguel, tenis, Madrid
7: Mario, baloncesto, Guadalajara
8: Alicia, tenis, Madrid
```

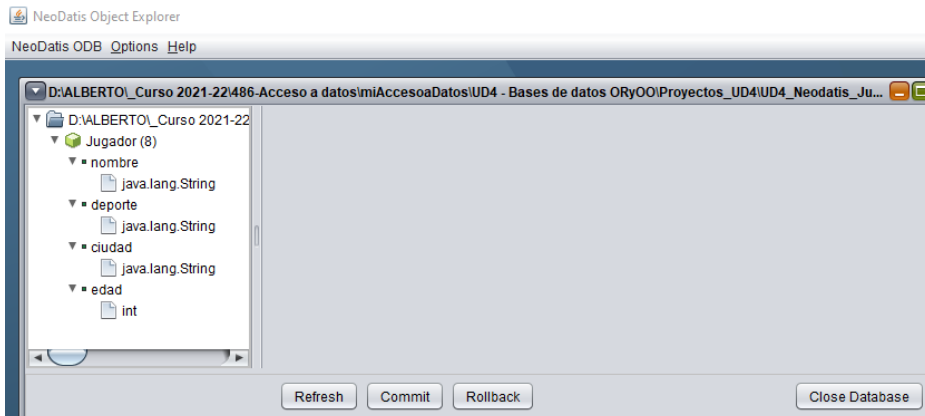
Al abrir la base de datos al fichero le hemos dado el nombre **neodatis.test** y podemos acceder a ella desde la aplicación **NeoDatis Object Explorer**, que en mi caso al estar con sistema operativo Windows lo hare ejecutando el fichero **odb-explorer.bat**. Para abrir la BD desde esta aplicación debo seleccionar el fichero de la base de datos y para ello, debo darme cuenta que se habrá creado en la carpeta donde está ubicado el proyecto que la ha creado. No es necesario especificar Usuario y Contraseña, sino sólo el nombre del fichero.



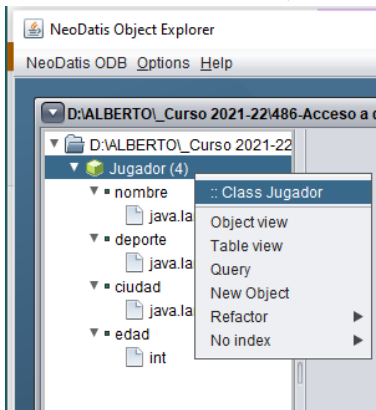
Cuando ya se ha abierto alguna vez la base de datos, el fichero aparece en el campo de Recientes (Recent) pero, en cualquier caso, debe seleccionarse el fichero (File Name) antes de realizar la conexión (Connect)



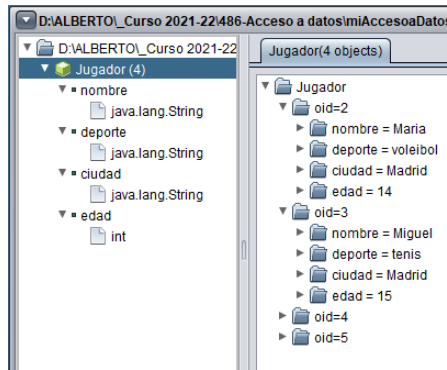
En el explorador de objetos puedo ver el meta-modelo, con las clases con sus atributos y métodos, si los tuviera.



Al seleccionar una clase, con el botón derecho abro el menú contextual y puedo acceder a varias operaciones.



En la vista de Objetos (Object view) veo los objetos con su identificador (oid)



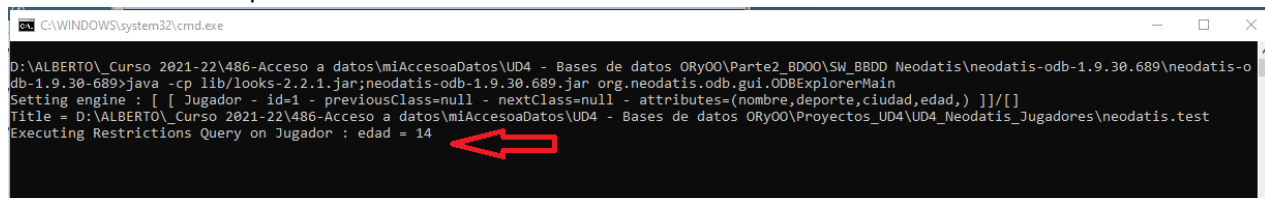
En la vista de Tabla (Table view) puedo ver los valores de los atributos en forma de tabla



Y en Consultas (Query) se abre un asistente gráfico de consultas. Al ejecutar una consulta, por ejemplo los jugadores con edad = 14, el resultado se muestra en una nueva pestaña de acuerdo a la vista que tenga seleccionada.



Como al ejecutar el archivo odb-explorer.bat también se abre una consola de comandos, puede comprobar en la consola la consulta que sea realizado.



Actividad 4.10. Crear base de datos Países y Jugadores.

Enunciado: Crea un proyecto con un paquete de nombre clases. Crea la clase Pais con dos atributos (private int id; private String nombrepais), sus métodos getter y setter y un método que devuelva el nombre del país (public String toString(){return nombrepais;}). Dentro del paquete clases, crea también la clase Jugador, ya vista en los ejemplos, y añade el atributo para indicar el país del jugador (private Pais pais) con sus getter y setter. Después, crea una clase con

que cree una base de datos de nombre EQUIPOS.DB e inserte algunos países y jugadores. Por último, crea una segunda clase para visualizar los países y jugadores que hay en la base de datos.

Solución:

2. Ejemplo de acceso a objetos usando su OID. Proyecto UD4_Neodatis_Jugadores_OID.

Hasta ahora hemos recuperado todos los objetos de un tipo desde la base de datos, pero si conocemos el identificador de un determinado objeto, podemos recuperar un único objeto por su OID (Object Identifier). En el siguiente enlace podemos ver cómo se almacena realmente la información que determina la posición física de cada OID en el fichero de la base de datos <http://Neodatis.wikidot.com/odb-file-format-1-9>.

Para acceder directamente a la información de un OID indicando su número utilizamos el método getObjectFromId(OID id). Si, por el contrario guardamos un determinado objeto y queremos saber su OID, podemos usar los métodos store(Objeto) o getObjectId(Objeto). En el ejemplo se ve cómo se pueden emplear.

```
// Recuperar objetos desde BD Neodatis por su OID
import org.neodatis.odb.ODB;
import org.neodatis.odb.ODBFactory;
import org.neodatis.odb.OID;
import org.neodatis.odb.core.oid.OIDFactory;

public class UD4_Neodatis_Jugadores_OID {

    public static void main(String[] args) {

        ODB odb = ODBFactory.open("neodatis.test"); // Abrir BD

        // Acceso al objeto a partir del identificador
        OID oid = OIDFactory.buildObjectOID(3); // Obtener los datos de un OID específico por su número
        Jugador jug = (Jugador) odb.getObjectFromId(oid); // Recupero el objeto que corresponde a un determinado OID
        System.out.printf("%s, %s, %s, %d %n %n", jug.getNombre(), jug.getDeporte(), jug.getCiudad(), jug.getEdad());

        // Obtener el OID de un determinado objeto e información de ese objeto
        OID oidjug = odb.getObjectId(jug); // En este caso, la instancia de jugador tiene el OID 3
        int tipojug = odb.store(jug).getType(); // Tenemos las clases Pais y Jugador. Jugador es tipo 2
        System.out.printf("OID: %s, Tipo OID: %d %n", oidjug.toString(), tipojug);

        odb.close(); // Cerrar BD
    } // fin main
} // fin clase
```

3. Ejemplos de consultas con criterios. Proyectos UD4_Neodatis_ConsultasCriterios y UD4_Neodatis_ConsultasExpreLogicas

Para realizar consultas se utiliza la clase CriteriaQuery que devuelve un objeto IQuery. Como podemos ver explicado en la wiki de NeoDatis <http://neodatis.wikidot.com/criteria-queries> una consulta por criterios consiste en definir 4 parámetros:

- La clase del objeto que debe ser cargada
- El criterio para filtrar los objetos
- El orden de los resultados
- El rango de los objetos que queremos que se devuelvan

Los criterios se especifican con la interfaz ICriterion, que dispone de varios métodos que se pueden en las siguientes tablas y se aplican con la clase Where. Como resultado se devuelve un objeto ICriterion que se incluye en la definición de la consulta.

Method	Description
equal	Returns an EqualCriterion
like	to execute a like
ilike	To execute a case insensitive like
gt	A greater than criterion
ge	A greater or Equal criterion
lt	A lesser than criterion
le	A lesser or equal criterio
contain	To search in list. Returns true if a list contains a specific element
isNull	Returns true if the object is null
isNotNull	Returns true if Object is not null

Method	Description
sizeEq	Returns true if the size of a list or array is equal to
sizeGt	Returns true if the size of a list or array is greater than
sizeGe	Returns true if the size of a list or array is greater or equal to
sizeLt	Returns true if the size of a list or array is lesser than
sizeLe	Returns true if the size of a list or array is lesser or equal to
or	To combine criterion with or : return criterio1 or criterio2 or criterion3
and	To combine criterion with and : return criterio1 and criterio2 and criterion3
not	To negate criterion

```
// Consultas sobre BD NeoDatis
```

```
import org.neodatis.odbc.Odb;
import org.neodatis.odbc.OdbFactory;
import org.neodatis.odbc.Objects;
import org.neodatis.odbc.core.query.IQuery;
import org.neodatis.odbc.core.query.criteria.ICriterion;
import org.neodatis.odbc.core.query.criteria.Where;
import org.neodatis.odbc.impl.core.query.criteria.CriteriaQuery;

public class UD4_Neodatis_ConsultasCriterios {

    public static void main(String[] args) {
        Odb odb = OdbFactory.open("neodatis.test");// Abrir BD

        // Define el criterio
        ICriterion criterio = Where.equal("deporte", "tenis");

        // Crea la consulta indicando la clase y el criterio
        IQuery query = new CriteriaQuery(Jugador.class, criterio);

        //Ordena de forma ascendente por nombre y edad
        query.orderByAsc("nombre,edad"); // ordena ascendentemente por nombre y edad

        Objects<Jugador> jugadores = odb.getObjects(query); // Recupera la colección de objetos

        System.out.println(jugadores.size() + " Jugadores:");
        while (jugadores.hasNext()) {
            Jugador j = (Jugador) jugadores.next();
            System.out.println("\t" + j.getNombre() + ", " + j.getDeporte() + ", " + j.getEdad());
        }

        /* También se puede definir el criterio directamente en la clase CriteriaQuery
        * Además, si en lugar de la colección de objetos queremos sólo el primer objeto
        * podemos usar el método getFirst que devuelve la excepción IndexOutOfBoundsException
        * si no se encuentra ningún objeto con ese criterio.
        */

        try {
            query = new CriteriaQuery(Jugador.class, Where.equal("deporte", "futbol"));
            Jugador j1 = (Jugador) odb.getObjects(query).getFirst(); // Recupera sólo el primer objeto
        } catch (IndexOutOfBoundsException e) {
            System.out.println("NO SE HA ENCONTRADO NINGUN OBJETO QUE CUMPLA EL CRITERIO");
        }

        odb.close(); // Cerrar BD
    } // fin main
} // fin clase
```

Se pueden realizar consultas con cualquiera de los métodos aplicables a la clase Where y también se pueden combinar varios criterios empleando expresiones lógicas empleando las clases And, Or y Not de NeoDatis, como se puede ver en el siguiente ejemplo. Al tratarse de clases, para poder usarlas tendremos que importarlas.


```

import org.neodatis.odb.core.query.criteria.And;
import org.neodatis.odb.core.query.criteria.Not;
import org.neodatis.odb.core.query.criteria.Or;

// Edad igual a 14
ICriterion criterio = Where.equal("edad", 14);
IQuery query = new CriteriaQuery(Jugador.class, criterio);

// Edad menor que 14
criterio = Where.lt("edad", 14);
query = new CriteriaQuery(Jugador.class, criterio);

// Edad menor o igual que 14
criterio = Where.le("edad", 14);
query = new CriteriaQuery(Jugador.class, criterio);

// Nombre empieza por la letra M
criterio = Where.like("nombre", "M%");
query = new CriteriaQuery(Jugador.class, criterio);

// Ciudad Madrid y edad 15
criterio = new And().add(Where.equal("ciudad", "Madrid"))
    .add(Where.equal("edad", 15));
query = new CriteriaQuery(Jugador.class, criterio);

// Ciudad Madrid o edad mayor o igual que 15
criterio = new Or().add(Where.equal("ciudad", "Madrid"))
    .add(Where.ge("edad", 15));
query = new CriteriaQuery(Jugador.class, criterio);

// Nombre no empieza por letra M
criterio = Where.not(Where.like("nombre", "M%"));
query = new CriteriaQuery(Jugador.class, criterio);

// Negación con método not de Where

// Nombre no empieza por letra A
criterio = new Not(Where.like("nombre", "A%"));
query = new CriteriaQuery(Jugador.class, criterio);

// Negación con la clase Not

```

Actividad 4.11. Consulta de valores sobre base de datos Países y Jugadores.

Enunciado: Empleando la base de datos EQUIPOS.DB creada en la actividad anterior, crea una clase que realice la consulta de los jugadores que practican tenis, pero en este caso que salga ordenada de forma descendente por nombre y edad, y que en el listado se vea también el país del jugador.

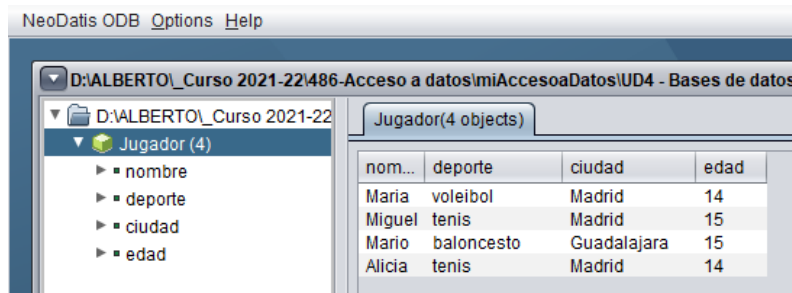
Solución:

4. Ejemplo de actualización y eliminación de datos. Proyecto UD4_Neodatis_ActualizayElimina

Para modificar un objeto tendremos que tenerlo en memoria. Para ello, lo cargo desde la BD, modifico los atributos necesarios con el correspondiente método set, después volvemos a almacenarlo en la BD con store() y, por último, valido los cambios con commit(). Para recuperar el objeto a modificar, en lugar de recorrer toda la colección que devuelve la consulta, utilizo el método getFirst() para obtener el objeto. Ahora bien, al utilizar el método getFirst() debo asegurarme de gestionar las excepción que saltará en caso de que trate de recuperar un objeto que no existe. Si se quieren modificar varios objetos a la vez, entonces tendré que recorrer la colección que devuelva la consulta.

Para eliminar un objeto, una vez recuperado de la BD, utilizo el método delete(Objeto) y confirmo con commit().

Compruebo lo que tengo en la BD antes de ejecutar el ejemplo



nom...	deporte	ciudad	edad
Maria	voleibol	Madrid	14
Miguel	tenis	Madrid	15
Mario	baloncesto	Guadalajara	15
Alicia	tenis	Madrid	14

Para ejecutar el programa debo desconectar odb-explorer porque no tengo acceso compartido.

```
// Actualiza y elimina objetos sobre BD Neodatis
// Gestiona también las posibles excepciones

import org.neodatis.odb.ODB;
import org.neodatis.odb.ODBFactory;
import org.neodatis.odb.core.query.IQuery;
import org.neodatis.odb.core.query.criteria.Where;
import org.neodatis.odb.impl.core.query.criteria.CriteriaQuery;

public class UD4_Neodatis_ActualizayElimina {

    public static void main(String[] args) {


        ODB odb = null; // Declaro la variable ODB fuera del bloque try para que se vea en todo el método
        try {
            odb = ODBFactory.open("neodatis.test"); // Abrir BD

            // -- Actualizar el deporte en el jugador de nombre Maria --
            IQuery query = new CriteriaQuery(Jugador.class, Where.equal("nombre", "Maria"));
            // Recupera el primer objeto que cumple la condición
            Jugador j = (Jugador) odb.getObjects(query).getFirst();
            // Actualizo el atributo
            j.setDeporte("voley-playa");
            odb.store(j); // Almaceno de nuevo el objeto en la BD

            // -- Eliminar el jugador de nombre Miguel --
            query = new CriteriaQuery(Jugador.class, Where.equal("nombre", "Miguel"));
            // Recupera el primer objeto que cumple la condición
            j = (Jugador) odb.getObjects(query).getFirst();
            // Actualizo el atributo
            j.setDeporte("voley-playa");
            odb.delete(j); // Elimino el objeto en a BD
            odb.commit(); // Valido los cambios

        } catch (IndexOutOfBoundsException e) {
            System.out.println("No se ha encontrado ningún jugador con ese nombre");
        } catch (Exception e) {
            System.out.println("Error" + e.getMessage());
        } finally {
            if (odb != null) {
                odb.close();
            }
        } // fin try-catch
    } // fin main
} // fin clase
```

Después de ejecutar el ejemplo, en la BD tengo ya sólo 3 objetos y se ha realizado el cambio de deporte para Maria



nom...	deporte	ciudad	edad
Maria	voley-playa	Madrid	14
Mario	baloncesto	Guadalajara	15
Alicia	tenis	Madrid	14

Actividad 4.12. Consulta de valores sobre base de datos Países y Jugadores.

Enunciado Parte A. Sobre la BD EQUIPOS.DB, crear un método que reciba el nombre de un país y un deporte, y visualice los jugadores que son de ese país y practican ese deporte. Si no hay ningún jugador el programa debe indicarlo.

Enunciado Parte B. Crear un método que reciba el nombre de un país y lo borre de la BD. Antes de borrar el país se debe comprobar que no lo tiene asignado ningún jugador y, en caso afirmativo, al jugador se le debe poner el atributo país como null.

Solución:**5. Ejemplo consultas con funciones. Proyecto UD4_Neodatis_ConsultasConFunciones**

En este ejemplo se trata de emplear la **API Object Values** de Neodatis que permite el uso de agrupamientos (GROUP BY) y funciones de grupo (SUM, MAX, MIN, AVG), así como recuperar **los valores de atributos** de los objetos almacenados en la DB.

Uso el método `getValues` de la clase `ODB` para recuperar una colección con los atributos de todos los objetos de la base de datos de una clase determinada. Los atributos a recuperar se detallan con el método `field` ("nombre").

Después para obtener el valor de un atributo concreto de un objeto, de entre todos los atributos recuperados, primero se deben recuperar los atributos de ese objeto recorriendo la colección y después indicar el atributo usando `getByAlias("alias")` o `getByIndex(índice)`, sabiendo que el índice comienza en la posición 0. Si quiero usar el alias, tendré que indicarlo en el método `field("nombreAtributo", "alias")`.

El ejemplo muestra cómo acceder a atributos o campos concretos de la BD (método `field(atributo)`) y después, en lugar de recuperar un campo, nos demuestra como obtener el resultado de una función de agrupamiento sobre un atributo concreto (métodos `sum(atributo)`, `count(atributo)`, `avg(atributo)`, `max(atributo)`, `min(atributo)`). Los valores que devuelven estas funciones son de tipo `BigDecimal`, excepto en el caso de `count` que es `BigInteger`.

```
// Consultas con funciones sobre BD NeoDatis
// API Object Values
import java.math.BigDecimal;
import java.math.BigInteger;

import org.neodatis.odb.ODB;
import org.neodatis.odb.ODBFactory;
import org.neodatis.odb.ObjectValues;
import org.neodatis.odb.Values;
import org.neodatis.odb.impl.core.query.values.ValuesCriteriaQuery;

public class UD4_Neodatis_ConsultasConFunciones {

    public static void main(String[] args) {
        ODB odb = ODBFactory.open("neodatis.test"); // Abro BD

        // Indica que quiere recuperar los atributos nombre y ciudad para todos los objetos de la clase Jugador
        Values valores = odb.getValues(new ValuesCriteriaQuery(Jugador.class).field("nombre").field("ciudad"));
        while (valores.hasNext()) {
            ObjectValues valoresObjeto = (ObjectValues) valores.next();
            // Accedo a los valores de los atributos de cada uno de los objetos por su Alias o su Índice
            System.out.printf("%s, Ciudad : %s %n", valoresObjeto.getByAlias("nombre"), valoresObjeto.getByIndex(1));
        }
    }
}
```

```
// -- OPERACIONES CON FUNCIONES DE GRUPO
// SUMA - Obtiene la suma de las edades, equivalente a SQL: SELECT SUM(edad) FROM Jugadores
Values val1 = odb.getValues(new ValuesCriteriaQuery(Jugador.class).sum("edad"));
ObjectValues ov = val1.nextValues();
BigDecimal value = (BigDecimal) ov.getByAlias("edad");
System.out.printf("Suma de edad : %d %n", value.longValue());
System.out.println("-----");
System.out.printf("Suma de edad : %.2f %n", ov.getByAlias("edad"));

// CUENTA - Obtiene el número de jugadores, equivalente a SQL: SELECT COUNT(nombre) FROM Jugadores
Values val2 = odb.getValues(new ValuesCriteriaQuery(Jugador.class).count("nombre"));
ObjectValues ov2 = val2.nextValues();
BigInteger value2 = (BigInteger) ov2.getByAlias("nombre");
System.out.printf("Numero de jugadores : %d %n", value2.intValue());
System.out.println("-----");

// MEDIA - Obtiene la edad media de los jugadores, equivalente a SQL: SELECT AVG(edad) FROM Jugadores
try {
    Values val3 = odb.getValues(new ValuesCriteriaQuery(Jugador.class).avg("edad"));
    ObjectValues ov3 = val3.nextValues();
    BigDecimal value3 = (BigDecimal) ov3.getByAlias("edad");
    System.out.printf("Edad media : %.2f %n", value3.floatValue());
    System.out.println("-----");
} catch (ArithmeticException e) { // Si la media (AVG) debe redondearse salta ArithmeticException
    System.out.println("Excepción => " + e.getMessage());
    Values val3 = odb.getValues(new ValuesCriteriaQuery(Jugador.class).sum("edad").count("edad"));
    ObjectValues ov3 = val3.nextValues();
    BigDecimal sumaedad = (BigDecimal) ov3.getByIndex(0);
    BigInteger cuenta = (BigInteger) ov3.getByIndex(1);
    float media = sumaedad.floatValue() / cuenta.floatValue();
    System.out.printf("La media de edad es: %.2f Contador = %d " + "suma = %.2f %n", media, cuenta, sumaedad);
}

// MAXIMO Y MINIMO - Obtiene la edad máxima y la edad mínima,
// equivalente a SQL: SELECT MAX(edad) edad_max , MIN(edad) edad_min FROM Jugadores
Values val4 = odb.getValues(new ValuesCriteriaQuery(Jugador.class).max("edad", "edad_max"));
ObjectValues ov4 = val4.nextValues();
BigDecimal maxima = (BigDecimal) ov4.getByAlias("edad_max");
Values val5 = odb.getValues(new ValuesCriteriaQuery(Jugador.class).min("edad", "edad_min"));
ObjectValues ov5 = val5.nextValues();
BigDecimal minima = (BigDecimal) ov5.getByAlias("edad_min");
System.out.printf("Edad máxima: %d, Edad mínima: %d %n", maxima.intValue(), minima.intValue());

// Uso del GROUP BY - Se cuenta el número de jugadores para cada ciudad
// equivalente a SQL: SELECT ciudad, COUNT(nombre) FROM Jugadores GROUP BY ciudad
Values groupby = odb.getValues(new ValuesCriteriaQuery(Jugador.class).field("ciudad").count("nombre").groupBy("ciudad"));
while (groupby.hasNext()) {
    ObjectValues objetos = (ObjectValues) groupby.next();
    System.out.printf("%s, %d%n", objetos.getByAlias("ciudad"), objetos.getByIndex(1));
}
odb.close();
} // fin main
```

IMPORTANTE: Hay que matizar que, en el caso de la función media o promedio (AVG), si el resultado del cálculo de la función no tiene un número de decimales finito para el tipo de dato y necesita redondeo, entonces salta la excepción `ArithmeticException` con el mensaje 'Rounding necessary' y se debería obtener el resultado haciendo la operación entre la suma de los valores (`sum()`) y el número de valores (`count()`).

6. Ejemplo consultas con JOINS. Proyecto UD4_Neodatis_VistasDinamicas

En la BD NeoDatis se llama Vistas Dinámicas a aquellas consultas que unen información de dos o más objetos que están relacionados a partir de alguno de sus atributos, lo que en el modelo relacional empleando SQL llamaríamos una consulta de tipo JOIN. Para realizar este tipo de consultas también se emplea la API Object Values de forma que al definir el atributo que queremos recuperar, en el método `field(Atributo)` para identificar el atributo de un objeto emplearemos la notación `AtributoClase1.AtributoClase2`, donde los nombres de los atributos están separados por un punto (.). De esta forma podemos navegar entre relaciones de objetos. En el ejemplo, que incluye varios métodos con consultas, se aplica esta notación en varios casos para acceder al `nombrepais` de la clase Pais desde el atributo país de la clase Jugador.

IMPORTANTE: En el caso de que llamemos a un valor NULL nos puede saltar error, por lo que para llamar a un atributo que puede ser nulo o es de otra clase conviene poner la condición `Where.isNotNull`. Es nuestro caso para el atributo `nombrepais` de la clase Pais llamada desde el país de Jugador, sería: `Where.isNotNull("pais.nombrepais")`

```

// Consultas sobre BD NeoDatis con varios objetos que están relacionados por algún atributo
// Consultas equivalentes a JOINS SQL, que en NeoDatis se realizan con la API Object values

import java.math.BigDecimal;
import java.math.BigInteger;

import org.neodatis.odb.ODB;
import org.neodatis.odb.ODBFactory;
import org.neodatis.odb.ObjectValues;
import org.neodatis.odb.Values;
import org.neodatis.odb.core.query.criteria.And;
import org.neodatis.odb.core.query.criteria.Where;
import org.neodatis.odb.impl.core.query.values.ValuesCriteriaQuery;

import clases.Jugador;

public class UD4_Neodatis_VistasDinamicas {

    public static void main(String[] args) {
        // Se llama a varios métodos, cada uno de los cuáles realiza una consulta con varios objetos (Vistas Dinámicas)
        JugadoresPaíses();
        System.out.println("-----");
        JugadoresPaísesSpain();
        System.out.println("-----");

        contadorymediaporpais();
        System.out.println("-----");
        visualizarmediadeedad();
        System.out.println("-----");
    } // fin método main

    // Consulta equivalente SQL: SELECT nombre, edad, nombrepais FROM Jugadores j, Países p WHERE j.id = p.id
    private static void JugadoresPaíses() {
        ODB odb = ODBFactory.open("EQUIPOS.DB");
        Values valores = odb.getValues(new ValuesCriteriaQuery(Jugador.class, Where.isNotNull("pais.nombrepais"))
            .field("nombre").field("edad").field("pais"));
        while (valores.hasNext()) {
            ObjectValues objectValues = (ObjectValues) valores.next();
            System.out.printf("Nombre: %s, Edad: %d, Pais: %s %n",
                objectValues.getByAlias("nombre"), objectValues.getByIndex(1), objectValues.getByIndex(2));
        }
        odb.close();
    } // Fin método JugadoresPaíses()

    // Consulta equivalente SQL: SELECT nombre, ciudad FROM Jugadores j, Países p WHERE (j.id = p.id AND nombrepais= 'ESPAÑA' AND edad= 15)
    private static void JugadoresPaísesSpain() {
        ODB odb = ODBFactory.open("EQUIPOS.DB");
        Values valores = odb.getValues(new ValuesCriteriaQuery(Jugador.class,
            new And().add(Where.equal("pais.nombrepais", "ESPAÑA"))
                .add(Where.equal("edad", 15)))
            .field("nombre")
            .field("ciudad"));

        while (valores.hasNext()) {
            ObjectValues objectValues = (ObjectValues) valores.next();
            System.out.printf("Nombre: %s, Ciudad: %s %n", objectValues.getByAlias("nombre"), objectValues.getByIndex(1));
        }
        odb.close();
    } // Fin método JugadoresPaísesSpain()
}

```

```

// Consulta equivalente SQL: SELECT nombrepais, MAX(edad), AVG(edad) FROM Jugadores j, Paises p WHERE j.id = p.id GROUP BY nombrepais
private static void contadorymediaporpais() {
    ODB odb = ODBFactory.open("EQUIPOS.DB");
    System.out.println("Numero de jugadores por país, " + " max de edad y media de edad: ");
    Values groupby = odb.getValues(new ValuesCriteriaQuery(Jugador.class, Where.IsNotNull("pais.nombrepais"))
        .field("pais.nombrepais").count("nombre")
        .max("edad").sum("edad").groupBy("pais.nombrepais"));

    if (groupby.size() == 0)
        System.out.println("La consulta no devuelve datos. ");
    else {
        while(groupby.hasNext()) { // Obtiene la media directamente dividiendo la suma de edades entre la cantidad de jugadores
            ObjectValues objetos= (ObjectValues) groupby.next();
            float media = ((BigDecimal) objetos.getByIndex(3)).floatValue() / ((BigInteger) objetos.getByIndex(1)).floatValue();

            System.out.printf("País: %-8s Num jugadores: %d, Edad Máxima: %.0f, Suma de Edad: %.0f, Edad media: %.2f %n",
                objetos.getByAlias("pais.nombrepais"), objetos.getByIndex(1), objetos.getByIndex(2), objetos.getByIndex(3), media );
        }
        odb.close();
    } // fin método contadorymediaporpais()

// Consulta equivalente SQL: SELECT AVG(edad) FROM Jugadores
private static void visualizarmediadeedad() {
    ODB odb = ODBFactory.open("EQUIPOS.DB");
    Values val;
    ObjectValues ov;
    try {
        // Al llamar a la función AVG debe gestionar la excepción ArithmeticException
        val = odb.getValues(new ValuesCriteriaQuery(Jugador.class).avg("edad"));
        ov = val.nextValues();
        System.out.printf("AVG-La media de edad es: %f %n", ov.getByIndex(0));

    } catch (ArithmeticException e) { // Si la media (AVG) debe redondearse salta ArithmeticException
        System.out.println("Excepción => " + e.getMessage());
        Values val2 = odb.getValues(new ValuesCriteriaQuery(Jugador.class).sum("edad").count("edad"));
        ObjectValues ov2 = val2.nextValues();
        BigDecimal sumaedad = (BigDecimal) ov2.getByIndex(0);
        BigInteger cuenta = (BigInteger) ov2.getByIndex(1);
        float media = sumaedad.floatValue() / cuenta.floatValue();
        System.out.printf("La media de edad es: %.4f Contador = %d " + "suma = %.2f %n", media, cuenta, sumaedad);
    }
    odb.close();
} // fin método visualizarmediadeedad()

```

Actividad 4.13. Consulta de valores sobre base de datos Países y Jugadores.

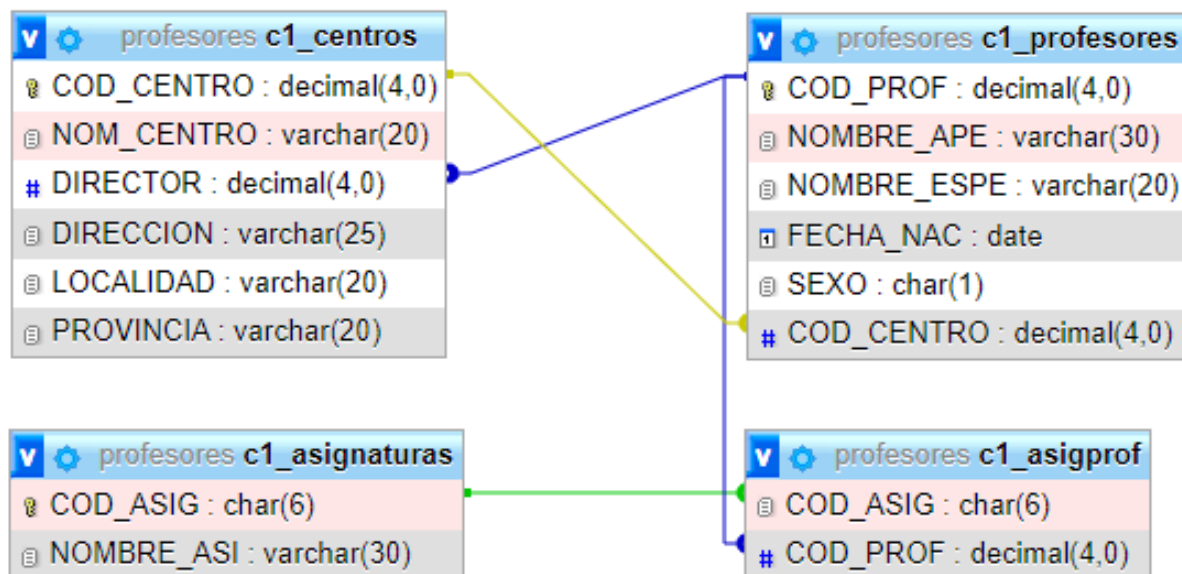
Enunciado. Sobre la BD EQUIPOS.DB realiza un método que reciba el nombre de un país y visualice el número de jugadores que tiene cada ciudad de ese país y la media de edad. Si no tiene jugadores que visualice el país que no tiene jugadores y, si el país no existe, que visualice el país que no existe. Para el cálculo de la media hay que evitar el error de redondeado de la media capturando la excepción que puede aparecer.

Solución:

7. Ejemplo de creación de BD NeoDatis desde modelo relacional. Proyecto UD4_Neodatis_Profesores

Partimos de que tenemos una base de datos MySQL con varias tablas de acuerdo al modelo relacional. La base de datos la creo en Xampp a partir del script de nombre TABLAS- PROFESORES-CENTROS_MYSQL.sql

Creo la BD de nombre 'profesores' y ejecuto el script TABLAS- PROFESORES-CENTROS_MYSQL.sql. Como resultado se crean 4 tablas con la estructura y relaciones que se pueden ver en la siguiente imagen:



Conviene matizar que las entre las tablas se establecen las siguientes relaciones:

- c1_centros: Incluye la información de los centros educativos de una red de centros. Un centro tiene varios profesores y cada centro tiene un profesor que es el director del mismo.
- c1_profesores: Tiene la información de los profesores y el centro al que pertenecen. Un profesor pertenece a un único centro e imparte asignaturas.
- c1_asigprof: Especifica las asignaturas que imparte los profesores. Un profesor imparte varias asignaturas y una asignatura puede ser impartida por varios profesores.
- c1_asignaturas: Incluye la información de las asignaturas y su nombre.

En este ejemplo vamos a crear una base de datos NeoDatis de nombre **profesores.neo** que permita realizar la gestión de la asignación de profesores a los centros indicando qué asignaturas imparten, de forma similar a la BD relacional MySQL, pero almacenando objetos. Además, la aplicación va a insertar en la base de datos profesores.neo, los valores de los atributos correspondientes de la BD MySQL profesores.

Empiezo creando una clase para cada una de las entidades a gestionar (**Asignatura, Centro, Profesor**). A la hora de definir estas clases, para establecer las relaciones entre los diferentes datos (profesores que imparten una asignatura, profesores que pertenecen a un centro) vamos a aprovechar que **podemos tener colecciones de objetos** y utilizaremos la **interfaz set**, que es una colección de objetos que no permite valores duplicados, para definir los Profesores que imparten una Asignatura y los Profesores que pertenecen a un Centro. De igual modo, para definir el centro al que pertenece un profesor tendremos que tener en cuenta que ese centro será una instancia de la clase Centro.

Con estas premisas, las clases tendrían los siguientes atributos y métodos:


```
*Asignatura.java
package clases;

import java.util.Set;

public class Asignatura {

    private String codAsig;
    private String nombreAsig;
    private Set<Profesor> setprofesores;

    public Asignatura(String codAsig, String nombreAsig, Set<Profesor> setprofesores) {
        super();
        this.codAsig = codAsig;
        this.nombreAsig = nombreAsig;
        this.setprofesores = setprofesores;
    }

    public Asignatura(){}

    public String getCodAsig() {
        return codAsig;
    }

    public void setCodAsig(String codAsig) {
        this.codAsig = codAsig;
    }

    public String getNombreAsig() {
        return nombreAsig;
    }

    public void setNombreAsig(String nombreAsig) {
        this.nombreAsig = nombreAsig;
    }

    public Set<Profesor> getSetprofesores() {
        return setprofesores;
    }

    public void setSetprofesores(Set<Profesor> setprofesores) {
        this.setprofesores = setprofesores;
    }

} // fin class Asignatura
```

```

Centro.java
package clases;

import java.util.Set;

public class Centro {
    private Integer codCentro;
    private String nomCentro;
    private Profesor director;
    private String direccion;
    private String localidad;
    private String provincia;
    private Set<Profesor> setprofesores;

    public Centro(Integer codCentro, String nomCentro, Profesor director, String direccion, String localidad,
        String provincia, Set<Profesor> setprofesores) {
        super();
        this.codCentro = codCentro;
        this.nomCentro = nomCentro;
        this.director = director;
        this.direccion = direccion;
        this.localidad = localidad;
        this.provincia = provincia;
        this.setprofesores = setprofesores;
    }

    public Centro(){}

    public Integer getCodCentro() { return codCentro; }
    public void setCodCentro(Integer codCentro) { this.codCentro = codCentro; }

    public String getNomCentro() { return nomCentro; }
    public void setNomCentro(String nomCentro) { this.nomCentro = nomCentro; }

    public Profesor getDirector() { return director; }
    public void setDirector(Profesor director) { this.director = director; }

    public String getDireccion() { return direccion; }
    public void setDireccion(String direccion) { this.direccion = direccion; }

    public String getLocalidad() { return localidad; }
    public void setLocalidad(String localidad) { this.localidad = localidad; }

    public String getProvincia() { return provincia; }
    public void setProvincia(String provincia) { this.provincia = provincia; }

    public Set<Profesor> getSetprofesores() { return setprofesores; }
    public void setSetprofesores(Set<Profesor> setprofesores) { this.setprofesores = setprofesores; }

} // fin clase Centro

```

```

Profesor.java
package clases;

import java.sql.Date;

public class Profesor {
    private Integer codProf;
    private String nombreApe;
    private String nombreEsp;
    private Date fechaNac;
    private String sexo;
    private Centro centros;

    public Profesor(Integer codProf, String nombreApe, String nombreEsp, Date fechaNac, String sexo, Centro centros) {
        super();
        this.codProf = codProf;
        this.nombreApe = nombreApe;
        this.nombreEsp = nombreEsp;
        this.fechaNac = fechaNac;
        this.sexo = sexo;
        this.centros = centros;
    }

    public Profesor(){}

    public Integer getCodProf() { return codProf; }
    public void setCodProf(Integer codProf) { this.codProf = codProf; }

    public String getNombreApe() { return nombreApe; }
    public void setNombreApe(String nombreApe) { this.nombreApe = nombreApe; }

    public String getNombreEsp() { return nombreEsp; }
    public void setNombreEsp(String nombreEsp) { this.nombreEsp = nombreEsp; }

    public Date getFechaNac() { return fechaNac; }
    public void setFechaNac(Date fechaNac) { this.fechaNac = fechaNac; }

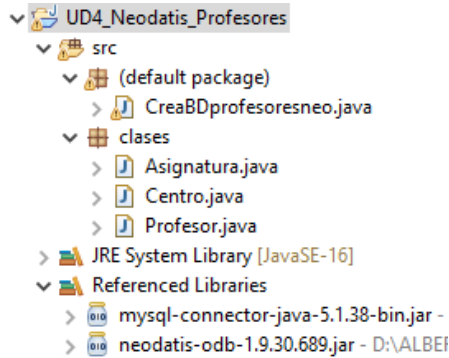
    public String getSexo() { return sexo; }
    public void setSexo(String sexo) { this.sexo = sexo; }

    public Centro getCentros() { return centros; }
    public void seCentros(Centro centros) { this.centros = centros; }

} // fin clase Profesor

```

Una vez definidas las clases, para crear la base de datos NeoDatis y llenarla con los valores de las tablas de la BD MySQL, tendremos que establecer conexión con la BD MySQL y crear la BD Neodatis. Para ello necesitaremos incluir en el classpath del proyecto los drivers de las dos BBDD. Una vez abierta la BD NeoDatis, iremos leyendo las tablas de la BD relacional con las consultas adecuadas y asignando valores a instancias de objetos de cada una de las clases, para después hacerlos persistentes en la BD NeoDatis. Hay que tener en cuenta que en el caso de las colecciones de tipo Profesor Set<Profesor>, que hemos creado en las clases Asignatura y Centro, tendremos también que llenar también estos conjuntos. Para cada una de estas acciones creamos un método y así, la estructura del proyecto y del método main será la siguiente:



```

CreaBDprofesoresneo.java
// Crea una base de datos NeoDatis a partir de una base de datos relacional MySQL

import java.sql.*;

public class CreaBDprofesoresneo {
    static ODB bd;
    public static void main(String[] args) {

        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection conexion = DriverManager.getConnection("jdbc:mysql://localhost/profesores", "root", "");
            bd = ODBFactory.open("profesores.neo");

            // Recorrer c1_asignaturas y guardar en NeoDatis
            InsertarAsignaturas(conexion);

            // Recorrer c1_Centros y guardar en NeoDatis
            InsertarCentros(conexion);

            // Recorrer c1_Profesores y guardar en NeoDatis
            InsertarProfesores(conexion);

            // llenar el set de profesores de asignaturas, por cada objeto asignatura hacemos la select
            LlenarSetProfesAsignaturas(conexion);

            // llenar el set de profesores de Centros y el director
            LlenarSetProfesEnCentrosYDirector(conexion);

            conexion.close();
            bd.close();

        } catch (ClassNotFoundException cn) {
            cn.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }

    } // fin main

```

Ahora, analizo en detalle cada uno de los métodos.

En primer lugar, al estar trabajando con una BD de objetos y no tener las restricciones del modelo relacional (integridad de identidad e integridad referencial), el evitar que los objetos no se dupliquen al ejecutar varias veces el programa y el garantizar que entre los objetos que tienen atributos relacionados la manipulación de los datos (inserciones, actualizaciones y borrado de valores) se realice de forma coherente y manteniendo la integridad de los datos, será algo que tengamos que gestionar desde nuestra aplicación. Por ello, por un lado vamos a crear métodos para comprobar la existencia de objetos de cada una de las clases (entidades) y, por otro lado, en los métodos para insertar valores, nos aseguraremos de asignar valores adecuados a aquellos atributos que son colecciones del tipo set que se relacionan con otras entidades.

Los métodos para comprobar, sobre la BD NeoDatis, la existencia o no de objetos son los siguientes:

```
private static boolean comprobarasig(String cod) {
    try {
        IQuery consulta = new CriteriaQuery(Asignatura.class, Where.equal("codAsig", cod));
        Asignatura obj = (Asignatura) bd.getObjects(consulta).getFirst();
        return true; // "EXISTE DEVUELVE true.";
    } catch (IndexOutOfBoundsException e) { // Salta excepción => "NO EXISTE DEVUELVE false."
        return false;
    }
} // fin comprobarasig

private static boolean comprobarcentro(int cod) {
    try {
        IQuery consulta = new CriteriaQuery(Centro.class, Where.equal("codCentro", cod));
        Centro obj = (Centro) bd.getObjects(consulta).getFirst();
        return true; // "EXISTE DEVUELVE true.";
    } catch (IndexOutOfBoundsException e) { // Salta excepción => "NO EXISTE DEVUELVE false."
        return false;
    }
} // fin comprobarcentro

private static boolean comprobarprofe(int cod) {
    try {
        IQuery consulta = new CriteriaQuery(Profesor.class, Where.equal("codProf", cod));
        Profesor obj = (Profesor) bd.getObjects(consulta).getFirst();
        return true; // "EXISTE DEVUELVE true.";
    } catch (IndexOutOfBoundsException e) { // Salta excepción => "NO EXISTE DEVUELVE false."
        return false;
    }
} // fin comprobarprofe
```

El método para insertar asignaturas, comprobará si ya existe llamando al método comprobarasig() y almacena el objeto de la clase Asignatura, creando el atributo setprofesores, pero sin valores.

```
private static void InsertarAsignaturas(Connection conexion) {
    try {
        Statement sentencia = (Statement) conexion.createStatement();
        ResultSet resul = sentencia.executeQuery("SELECT * FROM cl_asignaturas"); // Obtiene los valores de la tabla relacional
        while (resul.next()) {
            if (comprobarasig(resul.getString(1)) == false) {
                HashSet<Profesor> setprofesores = new HashSet<Profesor>(); // Nueva instancia de conjunto tipo Profesor
                Asignatura ass = new Asignatura(resul.getString(1), resul.getString(2), setprofesores);
                bd.store(ass); // Almacena objeto clase Asignatura
                System.out.println("Asignatura grabada " + resul.getString(1));
            } else {
                System.out.println("Asignatura: " + resul.getString(1) + ", EXISTE.");
            }
            bd.commit(); // Confirma todos los cambios
            resul.close();
            sentencia.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    } // fin InsertarAsignaturas
```

Por su parte para insertar Centros, comprueba si ya existe en la BD llamando al método comprobarcentro() y almacena el objeto de la clase Centro, también creando el atributo setprofesores sin valores y dejando a null el atributo Director, que es un objeto de la clase Profesor. Estos valores los rellenamos después en otro método.

```
private static void InsertarCentros(Connection conexion) {
    try {
        Statement sentencia = (Statement) conexion.createStatement();
        ResultSet resul = sentencia.executeQuery("SELECT * FROM c1_centros"); // Obtiene los valores de la tabla relacional
        while (resul.next()) {
            if (comprobarcentro(resul.getInt(1)) == false) {
                HashSet<Profesor> setprofesores = new HashSet<Profesor>(); // Nueva instancia de conjunto tipo Profesor
                // El atributo que ocupa la tercera posición es el Director, que es de la clase Profesor, y lo dejamos a null
                Centro cen = new Centro(resul.getInt(1), resul.getString(2), null, resul.getString(4),
                    resul.getString(5), resul.getString(6), setprofesores);
                bd.store(cen); // Almacena objeto clase Centro
                System.out.println("Centro grabado " + resul.getInt(1));
            } else {
                System.out.println("Centro: " + resul.getInt(1) + ", EXISTE.");
            }
        }
        bd.commit(); // Confirma todos los cambios
        resul.close();
        sentencia.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
} // fin InsertarCentros
```

Para insertar Profesores sigue la misma lógica y comprueba si ya existe en la BD llamando al método comprobarprofe() y almacena el objeto de la clase Profesor, pero obteniendo antes el objeto Centro que se corresponde con el código de centro que ha leído de la base de datos relacional.

```
private static void InsertarProfesores(Connection conexion) {
    try {
        Statement sentencia = conexion.createStatement();
        ResultSet resul = sentencia.executeQuery("SELECT * FROM c1_Profesores"); // Obtiene los valores de la tabla relacional
        while (resul.next()) {
            if (comprobarprofe(resul.getInt(1)) == false) {
                // Obtiene el objeto Centro que se corresponde al código de centro del profesor que ha leído de la tabla relacional
                IQuery consulta = new CriteriaQuery(Centro.class, Where.equal("codCentro", resul.getInt(6)));
                Centro cen = (Centro) bd.getObjects(consulta).getFirst(); //
                // Crea la instancia de Profesor, teniendo en cuenta que el atributo centros es el objeto que ha recuperado en la consulta
                Profesor nueprof = new Profesor(resul.getInt(1), resul.getString(2), resul.getString(3),
                    resul.getDate(4), resul.getString(5), cen);
                bd.store(nueprof); // Almacena objeto clase Profesor
                System.out.println("Profe grabado " + resul.getInt(1));
            } else {
                System.out.println("Profe: " + resul.getInt(1) + ", EXISTE.");
            }
        }
        bd.commit(); // Confirma todos los cambios
        resul.close();
        sentencia.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
} // fin InsertarProfesores
```

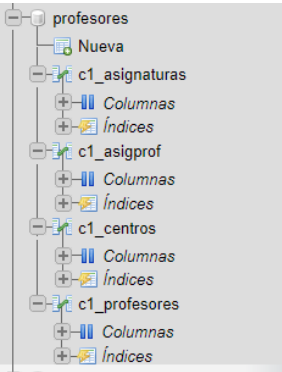
Por último, faltan los métodos para rellenar las dos colecciones de tipo set (profesores que imparten una asignatura, profesores que pertenecen a un centro), que hemos instanciado para insertar los objetos Asignatura y Centro, pero que estaban sin valores. Estos métodos son llenarSetProfesAsignaturas() y llenarSetProfesEnCentrosYDirector().

```
private static void llenarSetProfesAsignaturas(Connection conexion) throws SQLException {
    Objects<Asignatura> objects = bd.getObjects(Asignatura.class);
    while (objects.hasNext()) {
        Asignatura asi = objects.next();
        HashSet<Profesor> setprofesores = new HashSet<Profesor>();
        Statement sentencia = conexion.createStatement();
        ResultSet resul = sentencia
            .executeQuery("SELECT * FROM c1_asigprof where cod_asig = '" + asi.getCodAsig() + "'");
        while (resul.next()) {
            IQuery consulta = new CriteriaQuery(Profesor.class, Where.equal("codProf", resul.getInt(2)));
            Profesor obj = (Profesor) bd.getObjects(consulta).getFirst();
            setprofesores.add(obj);
        }
        asi.setSetprofesores(setprofesores);
        bd.store(asi);
        resul.close();
        sentencia.close();
    }
    bd.commit();
} // llenarSetProfesAsignaturas
```

En el método `llenarSetProfesEnCentrosYDirector()`, además de llenar el set que indica los profesores por centro, se asigna valor al atributo `Director` del Centro, que en esta BD NeoDatis es un objeto de la clase `Profesor`.

```
private static void llenarSetProfesEnCentrosYDirector(Connection conexion) throws SQLException {
    Objects<Centro> objectscen = bd.getObjects(Centro.class);
    while (objectscen.hasNext()) {
        Centro cee = objectscen.next();
        HashSet<Profesor> setprofesores = new HashSet<Profesor>();
        Statement sentencia = conexion.createStatement();
        ResultSet resul = sentencia
            .executeQuery("SELECT * FROM c1_profesores where cod_centro=" + cee.getCodCentro());
        while (resul.next()) {
            IQuery consulta = new CriteriaQuery(Profesor.class, Where.equal("codProf", resul.getInt(1)));
            Profesor obj = (Profesor) bd.getObjects(consulta).getFirst();
            setprofesores.add(obj);
        }
        cee.setSetprofesores(setprofesores);
        // Inserto el director.
        sentencia = conexion.createStatement();
        resul = sentencia.executeQuery("SELECT director FROM c1_centros where cod_centro=" + cee.getCodCentro());
        if (resul.next()) {
            IQuery consulta = new CriteriaQuery(Profesor.class, Where.equal("codProf", resul.getInt(1)));
            try {
                Profesor obj = (Profesor) bd.getObjects(consulta).getFirst();
                cee.setDirector(obj);
            } catch (IndexOutOfBoundsException ee) {
                System.out.println("Centro " + cee.getCodCentro() + ", Sin Director, es null.");
            }
        }
        bd.store(cee);
        resul.close();
        sentencia.close();
    }
    bd.commit();
} // llenarSetProfesEnCentrosYDirector
```

Para probar esta aplicación, creo la base de datos relacional MySQL sobre Xampp y compruebo los valores que hay en cada una de las tablas.



COD_ASIG	NOMBRE_ASI	COD_ASIG	COD_PROF
DB0001	Plástica	IF0002	1001
DB0002	Taller cerámica	IF0003	1001
DB0003	Dibujo Técnico	IF0001	1000
IF0001	DAHC	LG0001	2000
IF0002	RAL	LG0002	2000
IF0003	IMSI	LG0003	2003
IF0004	DPEG	LG0004	2003
IF0006	PLE	DB0001	2002
IF0007	FPE	DB0002	2002
LG0001	Lengua 1 ESO	DB0003	3000
LG0002	Lengua 2 ESO	MT0001	1010
LG0003	Lengua 3 ESO	MT0001	1011
LG0004	Lengua 4 ESO	MT0001	1022
MT0001	Matemáticas 1 BAC	MT0002	1010
MT0002	Matemáticas 2 BAC		

COD_CENTRO	NOM_CENTRO	DIRECTOR	DIRECCION	LOCALIDAD	PROVINCIA
1000	IES El Quijote	1000	Avda. Los Molinos 25	GUADALAJARA	GUADALAJARA
1015	CP Los Danzantes	1010	C/Las Musas s/n	PASTRANA	GUADALAJARA
1022	IES Planeta Tierra	2000	C/Mina 45	AZUQUECA	GUADALAJARA
1045	CP Manuel Hidalgo	NULL	C/Granada 5	GUADALAJARA	GUADALAJARA
1050	IES Antofiete	NULL	C/Los Toreros 21	SIGUENZA	GUADALAJARA

COD_PROF	NOMBRE_APE	NOMBRE_ESPE	FECHA_NAC	SEXO	COD_CENTRO
1000	Martínez Salas, Fernando	INFORMÁTICA	1961-09-07	H	1000
1001	Bueno Zarco, Elisa	INFORMÁTICA	1960-02-17	M	1000
1010	Montes García, M.Pilar	MATEMÁTICAS	1970-10-10	M	1015
1011	Arroba Conde, Manuel	MATEMÁTICAS	1970-10-12	H	1015
1022	Ruiz Lafuente, Manuel	MATEMÁTICAS	1966-11-11	H	1015
1045	Serrano Laguía, María	INFORMÁTICA	1976-01-02	M	1022
2000	Ramos Ruiz, Luis	LENGUA	1963-08-08	H	1022
2002	Rivera Silvestre, Ana	DIBUJO	1950-10-10	M	1000
2003	Segura Molina, Irene	LENGUA	1963-07-08	M	1022
3000	De Lucas Fdez, M.Angel	DIBUJO	1980-09-09	M	1000

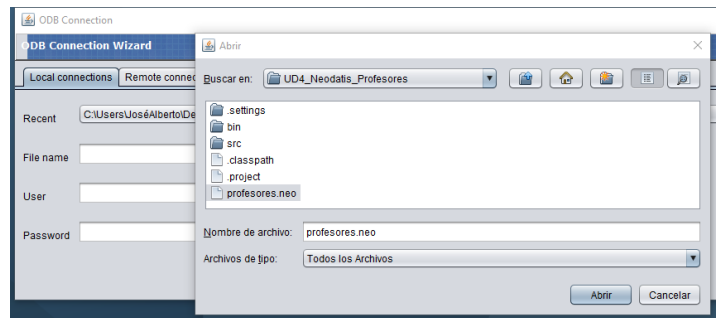
A continuación, ejecuto la aplicación, compruebo que se ha creado en la carpeta del proyecto la nueva base de datos de tipo NeoDatis y nombre 'profesores.neo', y veo los objetos que incluye utilizando odb-explorer.


```

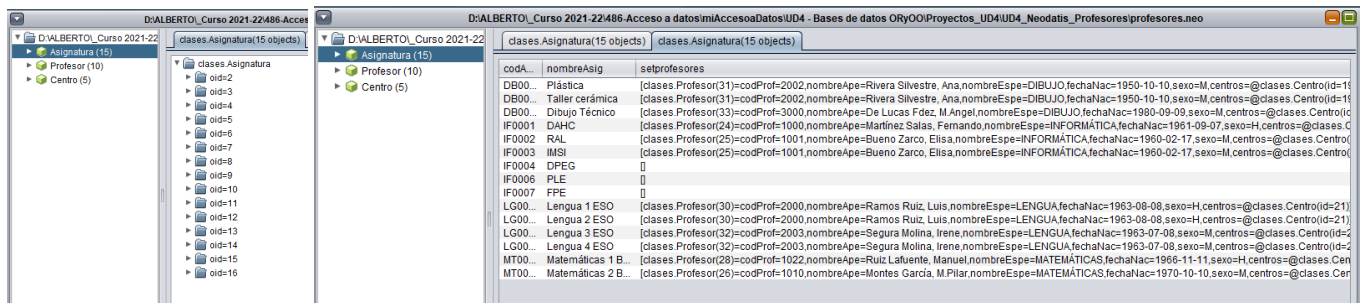
Console
<terminated> CreaBDprofesoresneo [Java Applicati
Asignatura grabada DB0001
Asignatura grabada DB0002
Asignatura grabada DB0003
Asignatura grabada IF0001
Asignatura grabada IF0002
Asignatura grabada IF0003
Asignatura grabada IF0004
Asignatura grabada IF0006
Asignatura grabada IF0007
Asignatura grabada LG0001
Asignatura grabada LG0002
Asignatura grabada LG0003
Asignatura grabada LG0004
Asignatura grabada MT0001
Asignatura grabada MT0002
Centro grabado 1000
Centro grabado 1015
Centro grabado 1022
Centro grabado 1045
Centro grabado 1050
Profe grabado 1000
Profe grabado 1001
Profe grabado 1010
Profe grabado 1011
Profe grabado 1022
Profe grabado 1045
Profe grabado 2000
Profe grabado 2002
Profe grabado 2003
Profe grabado 3000
Centro 1045, Sin Director, es null.
Centro 1050, Sin Director, es null.

```

Nombre	Tamaño
.settings	
bin	
src	
.classpath	1 KB
.project	1 KB
profesores.neo	30 KB



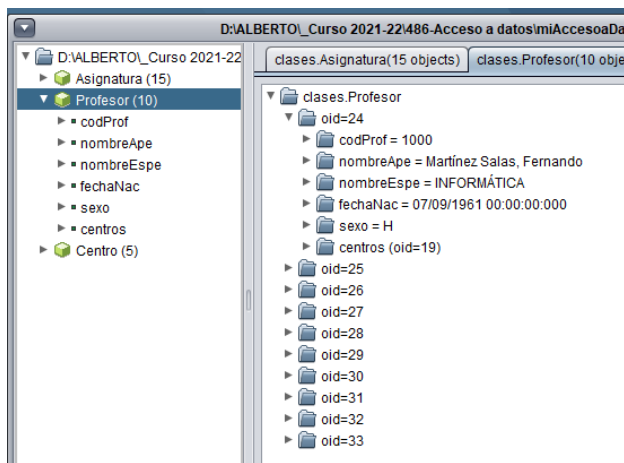
Objetos Asignatura



Objetos Centro (no se permite ver en modo Tabla porque tiene referencias cíclicas)



Objetos Profesor (no se permite ver en modo Tabla porque tiene referencias cíclicas)



Actividad 4.14. Crear BD NeoDatis VentasArticulos.

Enunciado: Crear una BD NeoDatis de nombre ARTICVENTAS.DAT empleando el proyecto UD4_Actividad14_CreaBD. Esta BD incluye objetos de tres clases (Clientes, Artículos y Ventas). Respecto a la definición de estas clases hay que destacar varios aspectos:

- En la clase Ventas el atributo numcli es un objeto de tipo Clientes que identifica al cliente al que se realiza la venta.
- En la que la clase Artículos el atributo Compras contiene una colección de tipo Set (definido como Set<Ventas>) y que incluye las Ventas de ese artículo (lo que se ha comprado) y dentro de esas Ventas también el Cliente que compró el Artículo.

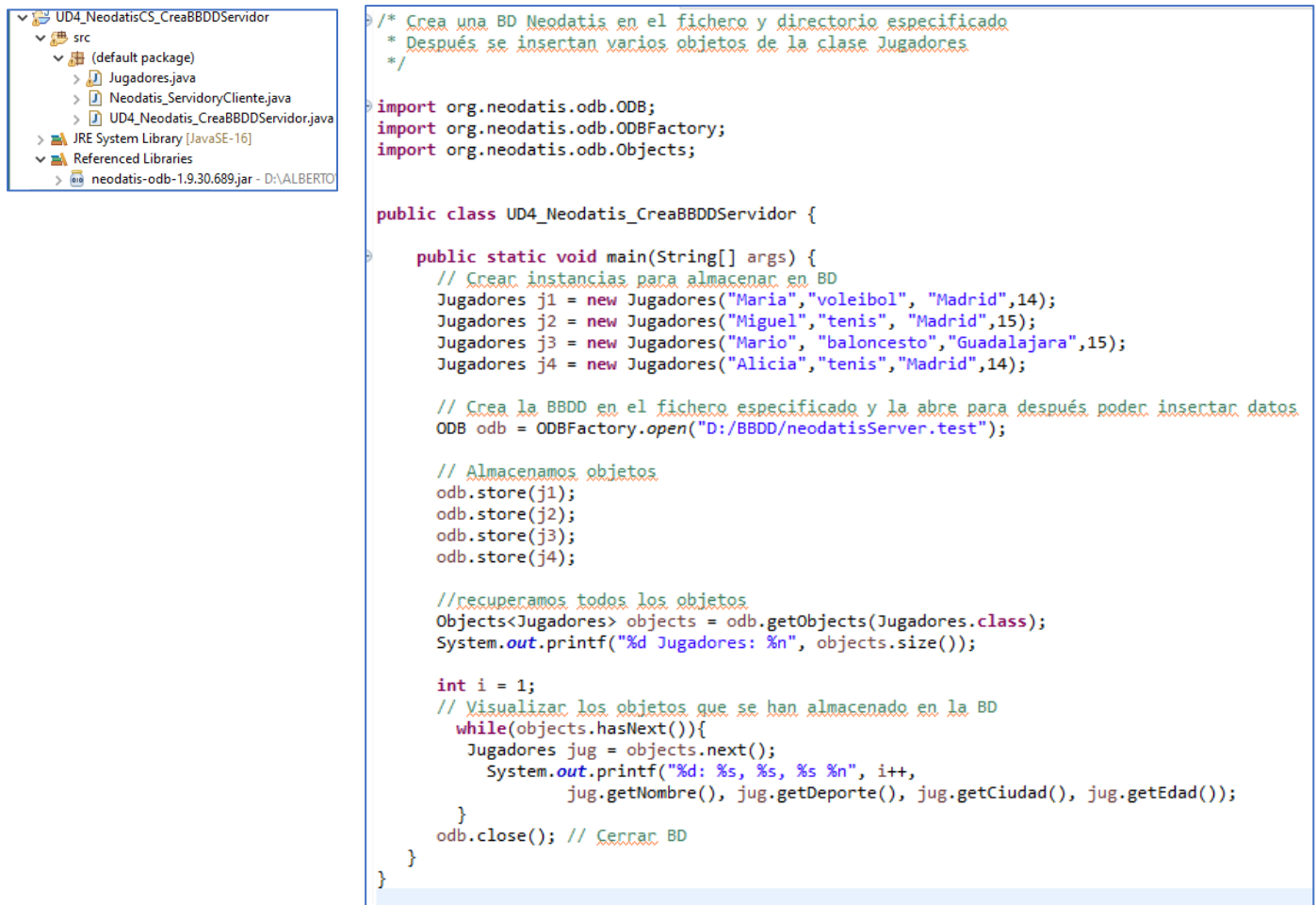
La actividad consiste en que a partir de esta base de datos hay que obtener:

- a) Un listado con las ventas de cada artículo, de forma que para cada artículo previamente se calcule:
 - Suma de unidades vendidas (SUMA_UNIVEN), que es la suma de las unidades vendidas del artículo que se encuentran en el set de Ventas de cada artículo.
 - Suma del importe (SUMA_IMPORTE), que es el resultado de multiplicar la suma de las unidades vendidas por el PVP del artículo.
 - Número de ventas (NUM_VENTAS), que es el contador de ventas del artículo que se corresponde con el número de elementos que aparecen en el set.
- b) Una línea de totales con la suma de todas las columnas que aparecen en el listado anterior.

Solución:**8. Modelo Cliente-Servidor de la BD NeoDatis**

Para probar el funcionamiento de Neodatis en modo cliente Servidor tendremos que tener dos equipos físicos o virtuales, o bien ejecutar el Servidor y la aplicación cliente en el mismo equipo. Por facilidad, opto por ejecutar tanto el Servidor como la aplicación Cliente que se conecta al servidor, dentro del mismo equipo.

Empiezo por crear una base de datos. Para ello empleo el proyecto de nombre UD4_NeodatisCS_CreaBBDDServidor



```

/* Crea una BD Neodatis en el fichero y directorio especificado
 * Después se insertan varios objetos de la clase Jugadores
 */

import org.neodatis.oddb.Odb;
import org.neodatis.oddb.OdbFactory;
import org.neodatis.oddb.Objects;

public class UD4_Neodatis_CreaBBDDServidor {

    public static void main(String[] args) {
        // Crear instancias para almacenar en BD
        Jugadores j1 = new Jugadores("Maria", "voleibol", "Madrid", 14);
        Jugadores j2 = new Jugadores("Miguel", "tenis", "Madrid", 15);
        Jugadores j3 = new Jugadores("Mario", "baloncesto", "Guadalajara", 15);
        Jugadores j4 = new Jugadores("Alicia", "tenis", "Madrid", 14);

        // Crea la BBDD en el fichero especificado y la abre para después poder insertar datos
        Odb odb = OdbFactory.open("D:/BBDD/neodatisServer.test");

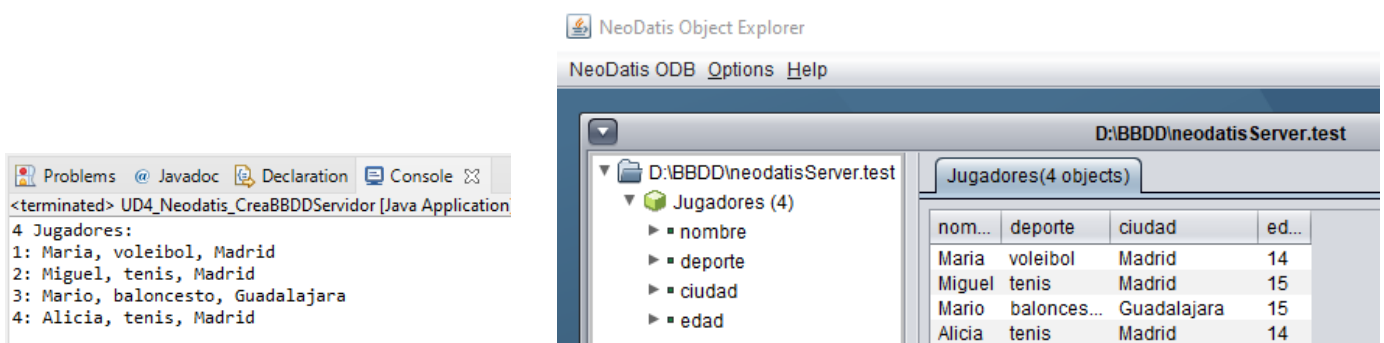
        // Almacenamos objetos
        odb.store(j1);
        odb.store(j2);
        odb.store(j3);
        odb.store(j4);

        //recuperamos todos los objetos
        Objects<Jugadores> objects = odb.getObjects(Jugadores.class);
        System.out.printf("%d Jugadores: %n", objects.size());

        int i = 1;
        // Visualizar los objetos que se han almacenado en la BD
        while(objects.hasNext()){
            Jugadores jug = objects.next();
            System.out.printf("%d: %s, %s, %s %n", i++,
                jug.getNombre(), jug.getDeporte(), jug.getCiudad(), jug.getEdad());
        }
        odb.close(); // Cerrar BD
    }
}

```

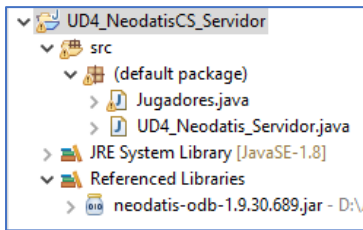
El propio programa muestra en la consola los 4 objetos insertados, pero también puedo comprobar su contenido con odb-explorer.bat



nom...	deporte	ciudad	ed...
Maria	voleibol	Madrid	14
Miguel	tenis	Madrid	15
Mario	balonces...	Guadalajara	15
Alicia	tenis	Madrid	14

Una vez creada la BBDD, ya puedo arrancar el Servidor, pero antes debo desconectarme de la BD si todavía estoy conectado con odb-explorer.

Para iniciar el servidor que atienda las peticiones hechas desde las aplicaciones clientes, creamos un proyecto de nombre UD4_NeodatisCS_Servidor que establece el puerto donde escucha el servidor y nombre que se da va a dar a de datos que se corresponde con el fichero donde se ha creado previamente la base de datos. Es necesario dar un nombre, porque el servidor puede atender peticiones para varias bases de datos almacenadas en diferentes ficheros.



En el proyecto tendré que tener también la clase que define los objetos que almacena la BD, en este caso Jugadores.java

```
/* Crea un servidor Neodatis en el puerto 8000
 * asignando el nombre "base1" al fichero de la BBDD
 * Las aplicaciones clientes tendrán que usar ese nombre y puerto
 * para después conectarse a la BBDD Neodatis.
 * Para que se pueda iniciar el servidor, el fichero de la BBDD tendrá
 * que existir en la ubicación especificada y el puerto tendrá que estar libre.
 */

import org.neodatis.odb.ODBFactory;
import org.neodatis.odb.ODBServer;

public class UD4_Neodatis_Servidor {

    public static void main(String[] args) {

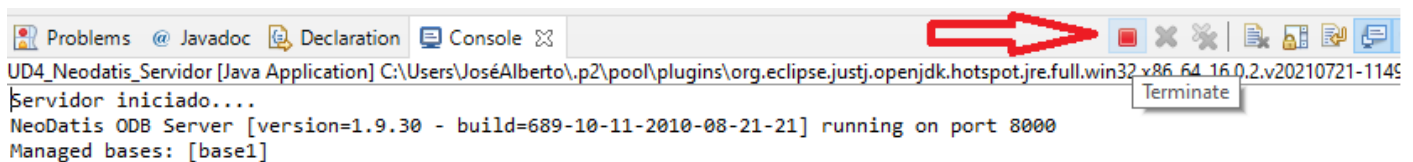
        ODBServer server = null;
        // Crea el servidor en el puerto 8000
        server = ODBFactory.openServer(8000);
        // Abre BD
        server.addBase("base1", "D:/BBDD/neodatisServer.test");
        // Se inicia el servidor ejecutándose en segundo plano
        server.startServer(true);

        System.out.println("Servidor iniciado...");

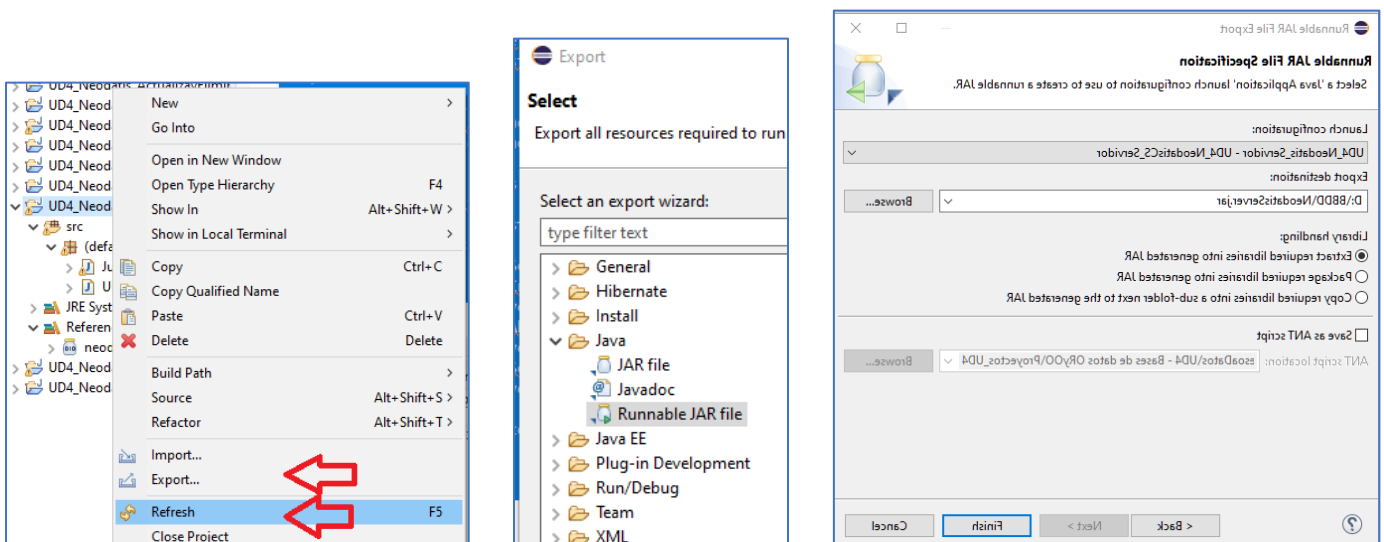
    }

}
```

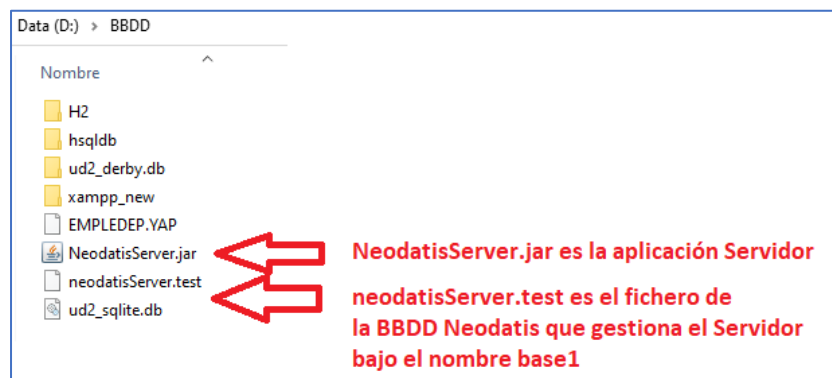
Puedo iniciar y después parar el servidor desde la consola de Eclipse



Pero en este caso voy a crear un archivo .jar que ejecutaré directamente. Para crear el archivo jar, selecciono el proyecto, pulso en Refresh (F5) y después en Export, indicando Runnable Jar File.



Para iniciar el servidor me sitúo en el directorio donde he dejado el archivo jar y lo inicio.



```
D:\>cd bbdd

D:\BBDD>dir
El volumen de la unidad D es Data
El número de serie del volumen es: 08DD-0115

Directorio de D:\BBDD

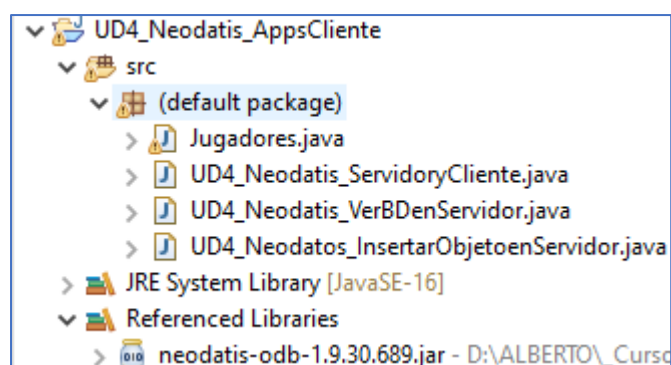
14/01/2022  21:55    <DIR>          .
14/01/2022  21:55    <DIR>          ..
26/10/2021  10:59                3.095 EMPLEDEP.YAP
30/10/2021  12:31    <DIR>          H2
31/10/2021  12:20    <DIR>          hsqldb
15/01/2022  12:05       780.222 NeodatisServer.jar
15/01/2022  11:53       20.333 neodatisServer.test
30/10/2021  11:46    <DIR>          ud2_derby.db
23/10/2021  11:12       20.480 ud2_sqlite.db
21/11/2021  22:46    <DIR>          xampp_new
                4 archivos          824.130 bytes
                6 dirs  382.905.671.680 bytes libres

D:\BBDD>java -jar NeodatisServer.jar
```

Una vez iniciado el servidor nos muestra en la consola del sistema lo mismo que aparece en la consola de Eclipse.

```
D:\BBDD>java -jar NeodatisServer.jar
Servidor iniciado....
NeoDatis ODB Server [version=1.9.30 - build=689-10-11-2010-08-21-21] running on port 8000
Managed bases: [base1]
```

Teniendo ya el Servidor escuchando en el puerto 8000, puede hacer peticiones de información desde aplicaciones cliente. Pruebo desde Eclipse a visualizar la información de la BBDD y después pruebo a insertar un objeto en la base de datos. Estas aplicaciones están como clases independientes en el proyecto UD4_NeodatisCS_AppsCliente.



Primero pruebo a visualizar la información de la BBDD (clase UD4_Neodatis_VerBDenServidor)

```
// Aplicación que visualiza desde una aplicación Cliente los datos en una BBDD Neodatis
// que se está ejecutando en modo Servidor (puerto = 8000, nombre asignado al fichero de BBDD = base1)

import org.neodatis.odbc.ODB;
import org.neodatis.odbc.ODBCFactory;
import org.neodatis.odbc.ODBCObjects;

public class UD4_Neodatis_VerBDenServidor {
    public static void main(String[] args) {
        ODB odb = null;
        try {
            odb = ODBCFactory.openClient("localhost", 8000, "base1");
            ODBCObjects<Jugadores> objects = odb.getObjects(Jugadores.class);
            System.out.printf("%d Jugadores: %n", objects.size());
            int i = 1;
            // visualizar los objetos
            while (objects.hasNext()) {
                Jugadores jug = objects.next();
                System.out.printf("%d: %s, %s, %s %n", i++,
                    jug.getNombre(), jug.getDeporte(), jug.getCiudad(), jug.getEdad());
            }
        } finally {
            if (odb != null)
                odb.close();
        } // fin finally
    } // fin main
} // fin clase UD4_Neodatis_VerBDenServidor
```

Problems @ Javadoc Declaration Console

<terminated> UD4_Neodatis_VerBDenServidor (1) [Java Applicac

```
4 Jugadores:
1: Maria, voleibol, Madrid
2: Miguel, tenis, Madrid
3: Mario, baloncesto, Guadalajara
4: Alicia, tenis, Madrid
```

Ahora pruebo a insertar un objeto Jugadores en la BBDD (clase UD4_Neodatis_InsertarObjetoenServidor)

```
// Aplicación que inserta un objeto en en una BBDD Neodatis desde una aplicación Cliente
// La BD se está ejecutando en modo Servidor (puerto = 8000, nombre asignado al fichero de BBDD = base1)

import org.neodatis.odbc.ODB;

public class UD4_Neodatos_InsertarObjetoenServidor {
    public static void main(String[] args) {
        ODB odb = null;
        try{
            odb = ODBCFactory.openClient("localhost", 8000, "base1");
            Jugadores j4 = new Jugadores("Andrea", "padel", "Guadalajara", 14);
            odb.store(j4);
            odb.commit();
            System.out.println("Jugador Insertado...");
        } finally {
            if (odb != null) {
                odb.close();
            }
        }
    } // --main
}
```

Problems @ Javadoc Declaration Console

<terminated> UD4_Neodatos_InsertarObjetoenServidor (1) [Java Appli

```
Jugador Insertado...
```


Por último, paro el Servidor desde la consola del sistema (Ctrl+C)

```
D:\BBDD>java -jar NeodatisServer.jar
Servidor iniciado....
NeoDatis ODB Server [version=1.9.30 - build=689-10-11-2010-08-21-21] running on port 8000
Managed bases: [base1]
D:\BBDD>
```