

TEMA 4. BASES DE DATOS OBJETO RELACIONALES Y ORIENTADAS A OBJETOS.

Objetivos

- Características de las bases de datos objeto-relacionales.
- Gestión de objetos con SQL; ANSI SQL 1999.
- Gestores de bases de datos objeto relacionales; conectores.
- Acceso a las funciones del gestor desde el lenguaje de programación.
- Características de las bases de datos orientadas a objetos.
- Gestores de bases de datos orientadas a objetos.
- Tipos de datos: tipos básicos y tipos estructurados.
- El interfaz de programación de aplicaciones de la base de datos.
- El lenguaje de consultas OQL: sintaxis, expresiones, operadores.
- Gestión de transacciones.

Contenidos

1.- Base de Datos Objeto-Relaciones.

Las Bases de Datos Objetos-Relaciones (BDOR) son una extensión de las bases de datos relaciones tradicionales a las que se les ha añadido conceptos del modelo orientado a objetos. Se trata pues de un modelo relacional que permite almacenar objetos en las tablas.

Las características más importantes de los SGBDOR son las siguientes:

- Soportes para tipos de datos simples, complejos y definidos por el usuario.
- Soporte para crear métodos para estos datos.
- Gestión sencilla de datos complejos.
- Herencia,
- Se pueden almacenar varios valores en una columna de una misma fila.
- Relaciones anidadas.
- Compatibilidad con bases de datos relacionales tradicionales.
- Inconveniente que aumenta la complejidad del Sistema.

2.- Base de Datos Orientada a Objetos.

Las Bases de Datos Orientadas a Objetos (BDOO) simplifican la programación orientada a objetos (POO) almacenando directamente los objetos en la base de datos y empleando las mismas estructuras y relaciones que los lenguajes de POO. Por lo tanto un SGBDOO es un SGBD que almacena objetos.

Características, Ventajas e Inconvenientes.

Características:

- Los datos se almacenan como objetos.
- Cada objeto se identifica mediante un identificador OID que no es modificable por el usuario.

- Cada objeto define sus atributos y métodos.
- Debe cumplir los principios de persistencia, concurrencia, recuperación ante fallos, gestión de almacenamiento secundario, facilidad de consultas (típico del SGBD) y encapsulación, herencia y polimorfismo (típico de OO).

Ventajas del SGBDOO:

- Mayor capacidad de modelado.
- Extensibilidad de tipos de datos.
- Lenguaje de consultas expresivo.
- Soporte para transacciones.

Inconvenientes del SGBDOO:

- Falta de un modelo universal.
- Falta de estándares.
- Competencia de los SGBD y SGBDOR.
- Complejidad.
- Falta de Soporte.

Estándar ODMG.

ODMG es un grupo de fabricantes de SGBDOO que pretender crear un estándar que lleva su nombre. Dispone del componente ODL que es un lenguaje de definición de objetos y OQL que es un lenguaje de consultas de objetos.

3.- SGBDOO NeoDatis Object Database.

NeoDatis ODB es una base de datos orientada a objetos de código abierto que funciona en Java y permite almacenar y recuperar con una sola línea de código sin necesidad de tener que mapear las tablas.

Podemos descargar la última versión desde <http://neodatis.wikidot.com/downloads>.

Una vez descargado el fichero, debemos descomprimir el ZIP y copiamos el fichero neodatis-odb-1.9.30.689.jar en el proyecto para poder añadirlo a la biblioteca.

Insertar.

La Base de datos se abre con el método open() de la clase ODBFactory, devolviendo un ODB. Los objetos se almacenan con el método store() y se recuperan con getObjects(). Para validar los cambios y cerrar la base de datos usamos close().

Ejemplo:

```
package ejemplo1neodatis;
// Importamos las clases del gestor de base de datos
// orientado a objeto de neoDatis
import org.neodatis.odb.*;
//Clase Jugadores
class Jugadores {
    // atributos
```

```

private String nombre;
private String deporte;
private String ciudad;
private int edad;
// constructores
public Jugadores() {}
public Jugadores(String nombre, String deporte,
                  String ciudad, int edad) {
    this.nombre = nombre;
    this.deporte = deporte;
    this.ciudad = ciudad;
    this.edad = edad;
}
// Setters y Getters
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;}
}
//
public class Ejemplo1Neodatis {
    public static void main(String[] args) {

        // Crear instancias para almacenar en BD
        Jugadores j1 = new Jugadores("Maria", "volei bol", "Madrid", 14);
        Jugadores j2 = new Jugadores("Miguel", "tenis", "Madrid", 15);
        Jugadores j3 = new Jugadores
                        ("Mariano", "baloncesto", "Guadalajara", 15);
        Jugadores j4 = new Jugadores("Alicia", "tenis", "Madrid", 14);

        ODB odb = ODBFactory.open("Jugadores.neo");// Abrir BD

        // 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.println("Jugadores: "+objects.size());

        int i = 1;
        // visualizar los objetos mientras haya objeto que mostrar
        while(objects.hasNext()){
            // Recuperamos el objeto de tipo Jugadores
            Jugadores jug = objects.next();
            System.out.println(i+": "+jug.getNombre()+

```

```

        " "+jug.getDeporte()+" "+jug.getCiudad()+
        " "+jug.getEdad());
        i++;
    }
    odb.close(); // Cerrar BD
}
}
}

```

Salida:

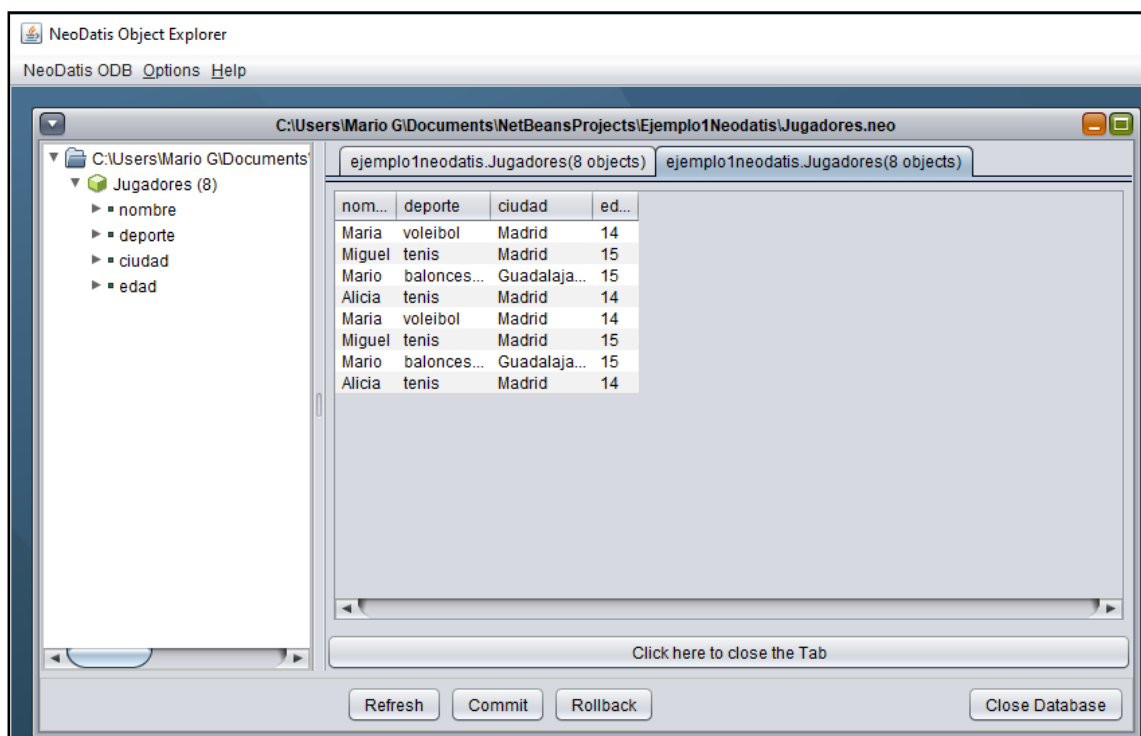
```

run:
4 Jugadores:
1: Maria, voleibol, Madrid
2: Miguel, tenis, Madrid
3: Mario, baloncesto, Guadalajara
4: Alicia, tenis, Madrid
BUILD SUCCESSFUL (total time: 0 seconds)

```

Explorador de BDOO de NeoDatis.

NeoDatis dispone de un explorador que permite navegar por los objetos de la base de datos. Podemos abrir el Explorador pulsando doble clic sobre el fichero **neodatis-odb-1.9.30.689**.



Para abrir una base de datos pulsamos NeoDatis ODB / Open Database. Podemos mostrar los datos pulsando con el botón derecho sobre la clase Jugadores y seleccionando Table view. Podemos consultar los datos pulsando con el botón derecho sobre la clase Jugadores y seleccionado Query. Para Cerrar la base de datos pulsaremos Close Database.

Consulta por OID.

Podemos acceder a un objeto concreto conociendo su OID. Por ejemplo, podemos consultar el Jugador con OID 3:

```
package ejemplooid;

import org.neodatis.odb.*;
import org.neodatis.odb.core.oid.*;

public class EjemploOID {

    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);

        ODB odb = ODBFactory.open("Jugadores.neo"); // Abrir BD

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

        // Obtenemos el OID del objeto 3
        OID oid = OIDFactory.buildObjectOID(3);
        // Obtenemos el Objeto con el OID del objeto 3
        Jugadores jug = (Jugadores) odb.getObjectFromId(oid);
        // Mostramos datos del objeto recuperado
        System.out.println(i+": "+jug.getNombre()+
        " "+jug.getDeporte()+" "+jug.getCiudad()+
        " "+jug.getEdad());
        // Cerrar BD
        odb.close();
    }
}

//Clase Jugadores
class Jugadores {

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

    // constructores
```

```

public Jugadores() {
}

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

// Setters y Getters
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;
}
}
//

```

Salida:

```

run:
Miguel, tenis, Madrid, 15
BUILD SUCCESSFUL (total time: 0 seconds)

```

Consultas con Condición.

Para realizar consultas usaremos la clase CriteriaQuery donde especificaremos la clase y el criterio para realizar la consulta.

Ejemplo: Listado de Jugadores que juegan al tenis ordenado por el nombre ascendente.

```
package consultaneodatis;
import org.neodatis.odt.*;
import org.neodatis.odt.core.query.criteria.*;
import org.neodatis.odt.impl.core.query.criteria.*;

public class ConsultaNeoDatis {

    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);

        ODB odb = ODBFactory.open("Jugadores.neo"); // Abrir BD

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

        // Creamos el criterio y la consulta
        ICriterion criterio = Where.equal("deporte", "tenis"); Condición
        CriteriaQuery query = new CriteriaQuery(Jugadores.class, criterio); La clase y la
                                                                    condicion que
                                                                    hemos creado

        // Ordenamos el resultado
        query.orderByAsc("nombre");

        //recuperamos los objetos que cumplen la condición
        Objects<Jugadores> objects = odb.getObjects(query);
        System.out.println("Jugadores: "+objects.size());

        // visualizar los objetos mientras haya objeto que mostrar
        while (objects.hasNext()) {
            // Recuperamos el objeto de tipo Jugadores
            try {
                Jugadores jug = objects.next();
                System.out.println(i+": "+jug.getNombre()+
                    " "+jug.getDeporte()+" "+jug.getCiudad()+
                    " "+jug.getEdad());
            } catch (IndexOutOfBoundsException e) {
                System.out.println("Objeto no localizado.");
            }
        }

        odb.close(); // Cerrar BD
    }
}
```

```
//Clase Jugadores
class Jugadores {
    private String nombre;
    private String deporte;
    private String ciudad;
    private int edad;
    public Jugadores() {
    }
    public Jugadores(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;
    }
}
```

Salida:

```
run:
2 Jugadores:
Alicia, tenis, Madrid 14
Miguel, tenis, Madrid 15
BUILD SUCCESSFUL (total time: 0 seconds)
```

Tabla de Operadores para Criterios.

Operador	Criterio
Igual	Where. equal ()
Mayor	Where. gt ()
Menor	Where. lt ()
Mayor o igual	Where. ge ()
Menor o igual	Where. le ()
Distinto	Where. not (Where. equal ())
Nulo	Where. isNull ()
No nulo	Where. isNotNull ()
Comienza, Termina o Contiene	Where. like () y el comodín %
Y	new And ()
O	new Or ()
Añadir Condición	. add ()

Esta tabla la tendremos el día del examen

Ejemplos:

```
Where. equal ("nombre", "Mario")
Where. gt ("edad", 18)
Where. lt ("edad", 65)
Where. ge ("edad", 45)
Where. le ("edad", 10)
Where. not (Where. equal ("deporte", "tenis"))
Where. isNull ("nombre")
Where. isNotNull ("deporte")
Where. like ("nombre", "%a%")
new And(). add (Where. like ("nombre", "%r%")). add (Where. equal ("edad", 14));
new Or(). add (Where. like ("nombre", "%r%")). add (Where. equal ("edad", 14));
```

Actualización.

Para modificar un objeto, primero es necesario cargarlo, después lo modificamos usando los métodos set del objeto y a continuación lo actualizamos con el método store(). Los cambios realizados en la base de datos se validarán utilizando el método commit() o cerrando la base de datos con el método close().

Ejemplo: Actualizar el deporte de Mario a futbol.

```
package actualizaneodatis;

import org.neodatis.odb.*;
import org.neodatis.odb.core.query.criterio.*;
import org.neodatis.odb.impl.core.query.criterio.*;

public class ActualizaNeoDatis {

    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);
```

```

        ODB odb = ODBFactory.open("Jugadores.neo"); // Abrir BD

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

        // Creamos el criterio y la consulta
        ICriterion criterio = Where.equal("nombre", "Mario");
        CriteriaQuery query = new CriteriaQuery(Jugadores.class, criterio);

        //recuperamos los objetos que cumplen la condición
        Objects<Jugadores> objects = odb.getObjects(query);

        // Obtenemos el primer objeto encontrado
        Jugadores jug = (Jugadores) objects.getFirst();

        // Mostramos los datos del Jugador
        System.out.println(i+": "+jug.getNombre()+
            " "+jug.getDeporte()+" "+jug.getCiudad()+
            " "+jug.getEdad());

        // Cambiamos el deporte con el método set
        jug.setDeporte("futbol");

        // Insertamos el objeto modificado en la base de datos
        odb.store(jug);

        // Validamos la modificación
        odb.commit();

        // Mostramos los datos modificados del Jugador
        System.out.println(i+": "+jug.getNombre()+
            " "+jug.getDeporte()+" "+jug.getCiudad()+
            " "+jug.getEdad());

        odb.close(); // Cerrar BD
    }
}

//Clase Jugadores
class Jugadores {

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

    public Jugadores() {
    }

    public Jugadores(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;
    }
}
//

```

Salida:

```

run:
Mario, baloncesto, Guadalajara 15
Mario, futbol, Guadalajara 15
BUILD SUCCESSFUL (total time: 0 seconds)

```

Eliminación.

Para eliminar un objeto primero lo localizamos y después lo eliminamos con el método delete().

Ejemplo: Eliminar el objeto de nombre Mario.

```

package elimineodatis;

```

```

import org.neodatis.odt.*;
import org.neodatis.odt.core.query.criteria.*;
import org.neodatis.odt.impl.core.query.criteria.*;

public class EliminaNeoDatis {

    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);

        ODB odb = ODBFactory.open("Jugadores.neo"); // Abrir BD

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

        // Creamos el criterio y la consulta
        ICriterion criterio = Where.equal("nombre", "Mario");
        CriteriaQuery query = new CriteriaQuery(Jugadores.class, criterio);

        //recuperamos los objetos que cumplen la condición
        Objects<Jugadores> objects = odb.getObjects(query);

        // Obtenemos el primer objeto encontrado
        Jugadores jug = (Jugadores) objects.getFirst();

        // Eliminamos el objeto de la base de datos
        odb.delete(jug);

        // Validamos la modificación
        odb.commit();

        System.out.println("Objeto Eliminado.");

        odb.close(); // Cerrar BD
    }
}

//Clase Jugadores
class Jugadores {

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

    public Jugadores() {
    }
}

```

```

public Jugadores(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;
}

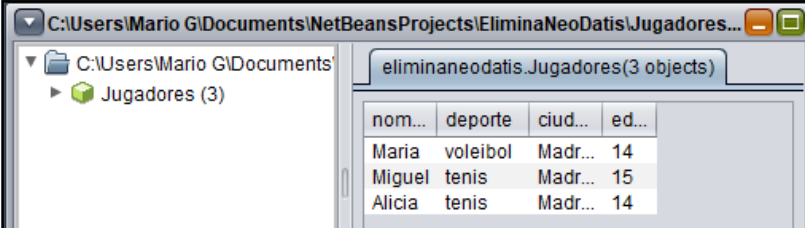
public void setEdad(int edad) {
    this.edad = edad;
}

public int getEdad() {
    return edad;
}
}
//

```

Salida:

run:
Objeto Eliminado.
BUILD SUCCESSFUL (total time: 0 seconds)



The screenshot shows the NetBeans IDE interface. On the left, the 'Project Explorer' pane displays the project structure: 'C:\Users\Mario G\Documents\NetBeansProjects\EliminaNeoDatis\Jugadores...' and a sub-package 'Jugadores (3)'. On the right, the 'Run' console shows the output: 'eliminaneodatis.Jugadores(3 objects)'. Below this, a table displays the data for the three objects.

nom...	deporte	ciud...	ed...
Maria	voleibol	Madr...	14
Miguel	tenis	Madr...	15
Alicia	tenis	Madr...	14