10_BDOO_II

OID

- Igual que en postgreSQL cada objeto dispone de su identificador que se suele llamar OID. Podemos ver los datos del objeto haciendo, por ejemplo:

```
package ejemplos;


import org.neodatis.odb.ODB;
import org.neodatis.odb.ODBFactory;
import org.neodatis.odb.OID;

public class OIDs {

    public static void main (String[] args) {
            Jugadores jug1=new Jugadores();
            ODB odb= ODBFactory.open("neodatis.test");
            Jugadores j1 = new Jugadores("Luis", "voleibol", "Madrid", 14);
            OID oid= odb.store(j1);
            jug1 = (Jugadores)odb.getObjectFromId(oid);
            System.out.printf("%s,%s,%s, %d %n", jug1.getNombre(), jug1.getDeporte(),jug1.getCiudad(),jug1.getEdad());
            odb.close();
    }
}
```

Consultas sencilllas

Para realizar consultas usamos la clase **CriteriaQuery** en donde especificaremos la clase y el criterio para realizar la consulta. La estructura básica es:

```
IQuery query = new CriteriaQuery(Jugadores.class, Where.equal(“deporte”,”tenis”));
query.orderByAsc (“nombre,edad”);
Objects<Jugadores> objects=odb.getObjects(query);
```

Para obtener el primer objeto usamos el método getFirst():

```
IQuery query = new CriteriaQuery(Jugadores.class, Where.equal(“deporte”,”tenis”));
query.orderByAsc (“nombre,edad”);
Jugadores j = odb.getObjects(query).getFirst();
```

Este método lanza la excepción **IndexOutOfBoundsException** si no localiza ningún objeto que cumpla con el criterio.

Sobre OQL

- OQL (Object Query Language) es el lenguaje estándar de consultas de BDOO. Las características son:
  - Es orientado a objetos y está basado en el modelo de objetos de la ODMG
  - Es un lenguaje declarativo del tipo de SQL. Su sintaxis básica es similar a SQL
  - No incluye operaciones de actualización, solo de consulta. Las modificaciones se realizan mediante los métodos que los objetos poseen.
  - Dispone de operadores sobre colecciones (max, min, count, etc.) y cuantificadores (for all, exists).
- La estructura básica es:

```
SELECT <lista de valores>
FROM <lista de colecciones y miembros>
 [WHERE <condición>
```

- En el FROM podemos indicar colecciones o expresiones que evalúan una colección. Se suele utilizar una variable iterador para ir recorriendo todos los elementos de la colección. Así, las siguientes expresiones son equivalentes:

```
FROM Clientes x
FROM x IN Clientes
FROM Clientes AS x
```

- Para acceder a los atributos y objetos relacionados se utilizan expresiones de camino. Veamos algunos ejemplos:

```
SELECT x.nombre.nombreper FROM x IN Clientes WHERE x.sexo=”M”
SELECT x.nombre.nombreper FROM Clientes x WHERE x.sexo=”M”
```

- Los valores pueden ser comparados usando los operadores habituales (=, >, <, >=,<=, etc.)
- Para comparar cadenas de caracteres se pueden utilizar los operadores IN y LIKE de la misma manera que se utilizan en SQL
- Un manual lo podéis encontrar aquí.
- En cuanto a ejemplos prácticos con NeoDatis tenéis esto que seguro que os resulta de gran utilidad.

Interfaz ICriterion

- Con CriteriaQuery se puede usar la interfaz ICriterion para construir el criterio de la consulta. Por ejemplo:

```
ICriterion criterio = Where.equal (“edad”, 14);
CriteriaQuery query = new CriteriaQuery (Jugadores.class, criterio);
Objects<Jugadores> objects = odb.getObjects(query);
```

- La lista de operadores que podemos utilizar para formalizar estos criterios de búsqueda la tenéis aquí

Problemas

1. Realiza un programa que realice una búsqueda por nombre en la base de datos “EQUIPOS.DB”. Pedirá un nombre al usuario y devolverá los datos del jugador que responda a dicho nombre o un mensaje del estilo “no hay ningún jugador que tenga ese nombre en la base de datos”
2. Realiza un programa que devuelva los jugadores cuyas edades estén comprendidas entre 14 y 20 años. Utiliza la interfaz ICriterion.
3. Realiza un programa que añada un año a la edad de todos los jugadores.