



BBDDs

Objeto Relacionales (OR) y  
Orientadas a Objetos (OO)



# Índice

- ◆ Introducción
- ◆ Bases de datos Objeto-Relacionales (BDOR-Oracle 8)
- ◆ Bases de datos Orientadas a Objetos (BDOO- Neodatis, Matisse)
- ◆ Ejemplos de DBOO – NeoDatis, DB4o, Matisse



## Introducción

- ▶ Bases de datos relacionales → Datos primitivos (int, char, bool, float..) y estructuras sencillas (arrays y cadenas)
- ▶ Años 80 y 90 → Sistemas más complejos
- ▶ Evolución de programación estructurada a orientada a Objetos
- ▶ Requisitos de los sistemas más complejos → Trabajar a alto nivel
- ▶ Necesidad de almacenar fotografías, vídeos, audio, diagramas



## Introducción

- ▶ Bases de datos tradicionales → Gran éxito en aplicaciones tradicionales
- ▶ Problemas en aplicaciones de ingeniería, CAD, experimentos científicos, telecomunicaciones, multimedia, documentales...
- ▶ Alternativas en la industria

Bases de datos Objeto-Relacionales

Bases de datos Orientadas a Objetos





## Bases datos Objeto-Relacionales (BDOR)

- ✓ Extensión de las bases de datos relacionales
- ✓ Guarda objetos en sus tablas
- ✓ Idea básica: El usuario puede crear sus propios tipos de datos
- ✓ Modelo ORBDMS

### u Ejemplos:

- Oracle Database – Lenguaje PL/SQL
- PostgreSQL



## Bases de datos Orientas a Objetos (BDOO) - Introducción

- ✓ En las bases de datos relacionales
  - ❑ se tienen que mapear las clases con las tablas para crear persistencia
  - ❑ Almacenar y recuperar objetos
- ✓ Surgen por problemas en las bases de datos tradicionales de representar datos más complejos.
- ✓ Se usarán las mismas clases. No es necesario hacer mapeo.





## Bases de datos Orientas a Objetos (BDOO) - Características

- ✓ Almacenan objetos
- ✓ Cada objeto - identificado un único OID (Object Identifier)
- ✓ Cada objeto - define sus métodos y atributos
- ✓ Todas las características de SGBD + características de Sist OO



## Bases de datos Orientas a Objetos (BD00) - Manifiesto de Malcolm Atkinson

1. Objetos complejos
2. Identidad del objeto
3. Encapsulación
4. Soporte de tipos o clases
5. Herencia
6. Polimorfismo
7. DML complejo (OML)
8. Tipos de datos extensible
9. Persistencia
10. Gestión de gran tamaño de datos
11. Concurrencia
12. Recuperación ante fallos
13. Método de consulta sencilla





## Bases de datos Orientas a Objetos (BDOO) - Ventajas

Se puede decir que un SGBDOO es

un SGBD que almacena objetos incorporando así todas las ventajas de la OO. Para los usuarios tradicionales de BD, esto quiere decir que pueden tratar directamente con objetos, no teniendo que hacer la traducción a tablas o registros. Para los programadores de aplicaciones, esto quiere decir que sus objetos se conservan, pueden ser gestionados aunque su tamaño sea muy grande, pueden ser compartidos entre múltiples usuarios, y se mantienen tanto su integridad como sus relaciones.

- Las BDOO permiten implementar los tres componentes de un modelo de datos:
- 1. Propiedades estáticas (objetos, atributos y relaciones)
- 2. Reglas de integridad de los objetos y operaciones
- 3. Propiedades dinámicas



## Bases de datos Orientas a Objetos (BD00) - Inconvenientes

- ✓ Carencia de modelo de datos universal
- ✓ Uso muy limitado
- ✓ Falta de estándares
- ✓ Competencia con SGBDR y SGBDOR
- ✓ Optimización de consultas compromete la encapsulación
- ✓ Complejidad
- ✓ Falta de soporte a las vistas
- ✓ Falta de soporte de seguridad



## Bases de datos Orientas a Objetos (BDOO) - ODMG

- ✓ Se trata de un estándar de facto
- ✓ Última versión: 3.0
- ✓ Modelo de objetos
- ✓ Lenguaje de **definición** de objetos (ODL)
- ✓ Lenguaje de **manipulación** de objetos (OML)
- ✓ Lenguaje de **consultas** de objetos (OQL)
- ✓ Conexión lenguajes C++, Smalltalk y Java
- ✓ Especifica las características de los objetos y como se relacionan

## Bases de datos Orientas a Objetos (BDOO) - Ejemplo: NeoDatis

- ✓ Ejemplo de base de datos orientada a objetos sencilla: NeoDatis
- ✓ De código abierto
- ✓ Objetos - CRUD única línea sin mapeo.
- ✓ Funciona en Java, Groovy, .NET y Android.
- ✓ Disponible en MAVEN añadiendo las líneas correspondientes en el pom.xml

```
<dependencies>
  <!-- https://mvnrepository.com/artifact/org.openengsb.wrapped/org.neodatis.odb-all -->
  <dependency>
    <groupId>org.openengsb.wrapped</groupId>
    <artifactId>org.neodatis.odb-all</artifactId>
    <version>1.9.30.687.w1</version>
  </dependency>
</dependencies>
```

## Bases de datos Orientas a Objetos (BDOO) – Ejemplo: NeoDatis

- ✓ Vamos a crear el proyecto NeoDatis v. Snapshot.
- ✓ ¿Qué significa versión Snapshot?
- ✓ Configuración de Maven
  - Ejemplo de Jugadores





# NeoDatis DB en Java





## Índice

1. Crear el conector con la base de datos
2. CRUD – Create
3. CRUD – Read all (Leer todo)
4. CRUD – Read OID (Leer un objeto concreto - sabiendo su OID)
5. Mostrar por pantalla el objeto leído
6. Consultas sencillas (Opción IQuery)
7. Consultas complejas (Pasar opción ICriterion al IQuery)
8. Consultas por valores (values)
9. CRUD - Update (Actualizar)
10. CRUD – Delete (Eliminar)
11. Cerrar la base de datos
12. Cliente/Servidor



## 1- Crear el conector con la base de datos

// Genérico

```
ODB odb = ODBFactory.open("nombreDelArchivo");
```

// Ejemplo de base de datos nueva o existente

// en z:\equipos.db

```
ODB odb = ODBFactory.open("z://equipos.db");
```



## 2- CRUD - Create

// Genérico

```
odb.store(objeto);
```

// Ejemplo de crear el objeto jug1 en la base de datos

```
odb.store(jug1);
```



### 3- CRUD - Read all (Leer todo)

// Genérico. Leer todos los objetos de la base de datos

```
Objects<Clase> objects = odb.getObjects(query);
```

// Ejemplo de leer (read) los objetos de la clase Jugadores de la BD

// y guardarlos en un array de Jugadores (clase Objects)

```
Objects<Jugadores> objects = odb.getObjects(Jugadores.class);
```

#### 4- CRUD – Read OID (Leer objeto concreto -> sabiendo su OID)

// Genérico. Leer un objeto de la base de datos con un OID en concreto.

// Hace la búsqueda

```
OID oid = OIDFactory.buildObjectOID(numeroOID);
```

// Lee de la base de datos

```
Clase objeto = (Clase) odb.getObjectFromId(oid);
```

// Ejemplo de leer (read) el objeto con OID 3

```
OID oid = OIDFactory.buildObjectOID(3);
```

```
Jugadores jug = (Jugadores) odb.getObjectFromId(oid);
```

## 5- Mostrar por pantalla el objeto leído

// Genérico

// Crear un objeto desde el array de objetos

Clase objeto = arrayObjects.next();

// Mostrar getters por pantalla

System.out.println(objeto.getMetodo());

// Ejemplo de leer el primer elemento del array de objetos

Jugadores jug = objects.next();

// Usar un getter: mostrar el nombre del jugador por pantalla

System.out.println(jug.getNombre());



## 6- Consultas sencillas (Opción IQuery)

// Genérico. Se hacen con un objeto de la clase IQuery

```
IQuery query = new CriteriaQuery(Clase.class, criterio);
```

// Leo los objetos que coinciden con la búsqueda

```
Objects<Clase> objects = odb.getObjects(query);
```

// Ejemplo. Búsqueda de todos los objetos de la clase

// Jugadores que de atributo 'deporte' tienen puesto 'tenis'

```
IQuery query = new CriteriaQuery(Jugadores.class, Where.equal("deporte", "tenis"));
```

```
Objects<Jugadores> objects = odb.getObjects(query);
```

## 6- Consultas sencillas (Opción IQuery) Ordenar los resultados

// Genérico. Después de crear el objeto de la clase IQuery,

// usar el método .orderBy...

```
query.orderByAsc("nombre,edad");
```

// Ejemplo. Ordenar nuestra búsqueda en el objeto query (de

// la clase IQuery) en orden ascendente por nombre y edad

```
query.orderByAsc("nombre,edad");
```

## 6- Consultas sencillas (Opción IQuery) Obtener los objetos

// Genérico. Es igual que el primer método visto de

// CRUD (Read)

```
Objects<Clase> objects = odb.getObjects(query);
```

// Ejemplo

```
Objects<Jugadores> objects = odb.getObjects(query);
```

## 7- Consultas complejas (Pasar opción ICriterion al IQuery) Where I

```
// Es una consulta SIMPLE que recibe un argumento de
// consulta COMPLEJA (un objeto ICriterion)
// Genérico. Se debe pasar un objeto de la clase ICriterion cuando construimos la IQuery
ICriterion criterio = Where.equal("atributo", valorAtributo);
// Se añade el objeto de consulta compleja creada a la consulta simple
IQuery query = new CriteriaQuery(Clasa.class, criterio);

// Ejemplo. Se quiere buscar objetos de la clase Jugadores
// en el que la edad es 14.
ICriterion criterio = Where.equal("edad", 14);
IQuery query = new CriteriaQuery(Jugadores.class, criterio);
```

## 7- Consultas complejas (Pasar opción ICriterion al IQuery) Where II

// Where.ge: mayor o igual

// Where.lt: menor que

// Where.le: menor o igual

// Where.Not: Distinto

// Where.isNull("atributo"): si un atributo es nulo

// Where.isNotNull("atributo"): si un atributo no es nulo

// Where.like: uso patrones (wildcard)

// %: cualquier número de caracteres

// x%: empieza por x

// \_: encaja con un carácter

// Mari\_: palabras que empiecen por Mari y la última

//letra cualquier carácter(Mario, Maria...)

// Where.equal: es igual

## 7- Consultas complejas (Pasar opción ICriterion al IQuery) And

// Ejemplo. Se quiere buscar objetos de la clase Jugadores

// de un país en concreto (EEUU) que practiquen un deporte

// en concreto (tenis)

ICriterion criterio = new

Crear el objeto

And().add(Where.equal("pais.nombrePais", "EEUU"))

Añado el primer criterio

.add(Where.equal("deporte", "tenis"));

Añado el segundo criterio

Que cumpla todos  
los criterios (AND)



## 7- Consultas complejas (Pasar opción ICriterion al IQuery) And y Or

// Ejemplo. Se quiere buscar objetos de la clase Jugadores

// que tengan:

//        14 años

//    y   España O Italia O Francia

```
ICriterion criterio = new
```

```
And().add(Where.equal("edad", 14))
```

```
.add(new Or().add(Where.equal("pais.nombrePais", "España"))
```

```
.add(Where.equal("pais.nombrePais", "Italia"))
```

```
.add(Where.equal("pais.nombrePais", "Francia")));
```



## 8- Consultas por valores (values)

// Queremos hacer operaciones con los valores directamente

// de los objetos de dentro de la base de datos.

// En este tipo de consultas NO obtengo los objetos enteros, // sinó que obtengo ciertos atributos (fields).

// Genérico. Usamos objetos de la clase Values (tipo HashTable) para // obtener SOLAMENTE los atributos A, B y C. de todos los objetos de

// la base de datos

```
Values values = odb.getValues(new ValuesCriteriaQuery(Clase.class)
```

```
.field("atributoA")
```

```
.field("atributoB")
```

```
.field("atributoC")
```

```
);
```



## 8- Consultas por valores (values) Ejemplo I

// Ejemplo. Queremos obtener en la HashTable de la clase  
// VALUES todos los atributos 'nombre', 'edad' y 'nombre del  
// país'. No quiero recuperar los objetos, solamente algunos  
// valores suyos

```
Values values = odb.getValues(new  
ValuesCriteriaQuery(Jugadores.class)  
.field("nombre")  
.field("edad")  
.field("pais.nombrePais"));
```



## 8- Consultas por valores (values) Ejemplo II

// Ejemplo. Se pueden combinar 'Consultas simples',

// 'consultas complejas' y 'consultas por valores'

// Obtengo en una HashTable del tipo values sólo los atributos

// de 'nombre' y 'ciudad' de la consulta de objetos que su país

// es Italia y tienen 15 años de edad

```
Values values = odb.getValues(new ValuesCriteriaQuery( Jugadores.class, new  
And().add(Where.equal("pais.nombrePais", "Italia"))  
.add(Where.equal("edad", 15)) )  
.field("nombre")  
.field("ciudad")  
);
```



## 8- Consultas por valores (values) Ejemplo II(Cont.)

- Opción1

```
while (values.hasNext()) {  
    ObjectValues objectValues = (ObjectValues) values.next();  
    System.out.printf("Nombre : %s, Deporte : %s %n",  
objectValues.getByAlias("nombre"), objectValues.getByIndex(1));  
}
```

- Otras opciones

```
for(ObjectValues valor : values) {  
    System.out.println(valor.toString());  
    System.out.println(valor.getByAlias("nombre"));  
}
```



## 8- Consultas por valores (values) Ejemplo III

// Admiten operadores distintos:

// sum: hace la suma -> // Obtener la suma de las edades

// count: cuenta el número de ocurrencias -> // Obtener el número de jugadores

// avg: promedio -> // Obtener la edad media de los jugadores // Media

// max: valor máximo -> // Obtener la edad máxima y la edad mínima de jugadores

// min: valor mínimo

// groupby: agrupamiento // Obtener Contador, edadMax, edadMin, SumaEdades por país



## - CRUD -

### 9- CRUD - Update (actualizar)

// Genérico. Debemos leer un objeto guardado  
// y modificar algún atributo con el setter  
// y después llamar a los métodos:

```
odb.store(obj);
```

```
odb.commit();
```

```
//(CR)Update(D)
```

```
//actualizarEdadJugadoresPorPais("España");
```

```
//cambiarDeporteJugador();
```

### 10- CRUD - Delete (eliminar)

// Genérico. Debemos leer un objeto guardado y a  
//continuación lo eliminamos con ayuda de métodos:

```
odb.delete(obj);
```

```
//(CRU) Delete
```

```
//borrarPaisdeBD("España");
```

```
//borrarJugador("Maria");
```



## 11- Cerrar la base de datos

// Genérico. Cuando hemos terminado de utilizar la base de datos,

// debemos cerrarla.

```
odb.close();
```



## 12- Cliente/Servidor

- Servidor

```
ODBServer server = null;  
    // Crea el servidor en el puerto 8000  
    server = ODBFactory.openServer(8000);  
    // Abrir BD  
    server.addBase("base1",  
"neodatisServer.test");  
    // Se inicia el servidor ejecutándose en  
segundo plano  
    server.startServer(true);  
    System.out.println("Servidor iniciado....");
```

- Cliente

```
odb = ODBFactory.openClient("localhost", 8000,  
"base1");  
  
Objects<Jugadores> objects =  
odb.getObjects(Jugadores.class);  
  
....  
  
odb.close();
```



## 12- Cliente/Servidor (Cont.)

- Ejercicio en Proyecto Jugadores
  1. Reiniciar la BD con 4 valores a insertar.
  2. Crear el servidor y lanzarlo
  3. Crear cliente y consultar los datos en la BD.
  4. Añadir un nuevo jugador usando cliente
  5. Volver a consultar los datos desde cliente
- Ejercicio enunciado [BDNeodatisLibrosAutores](#)

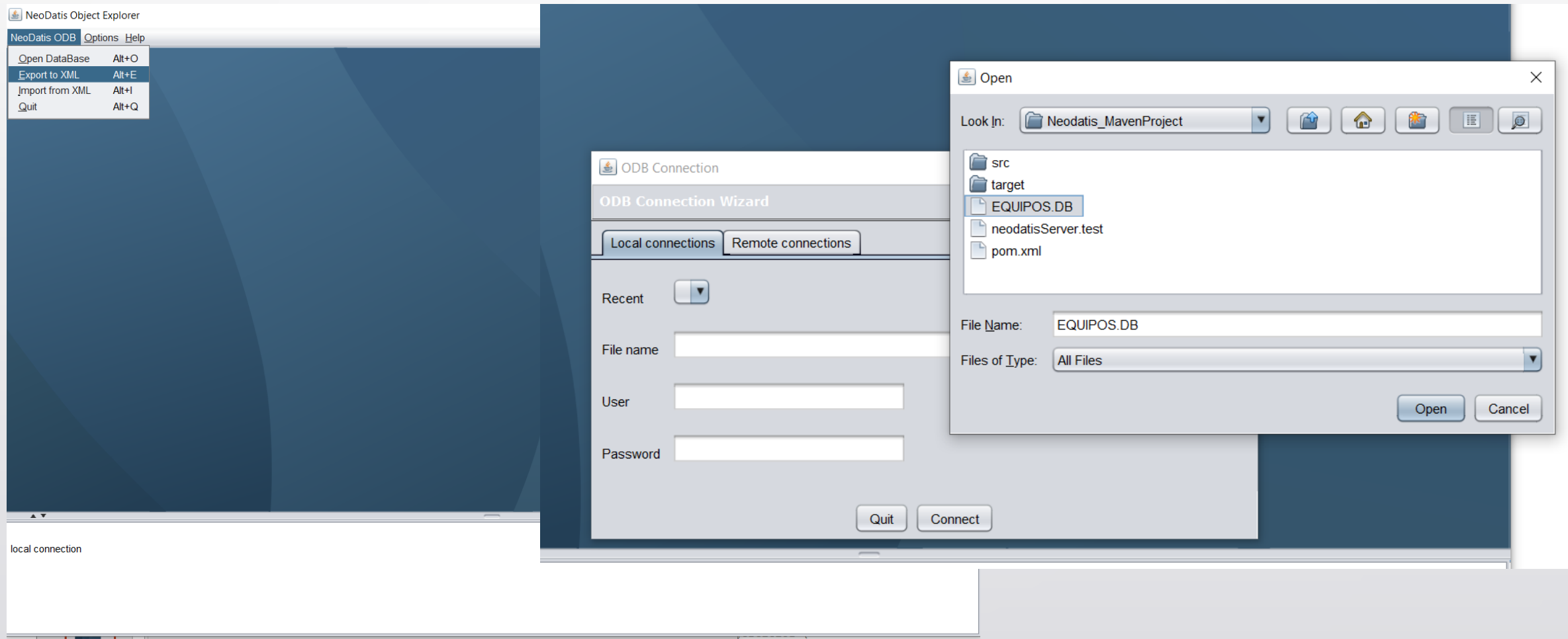


## GUI-Entorno Visual

neodatis-odb-1.9.30.689.jar (doble click)

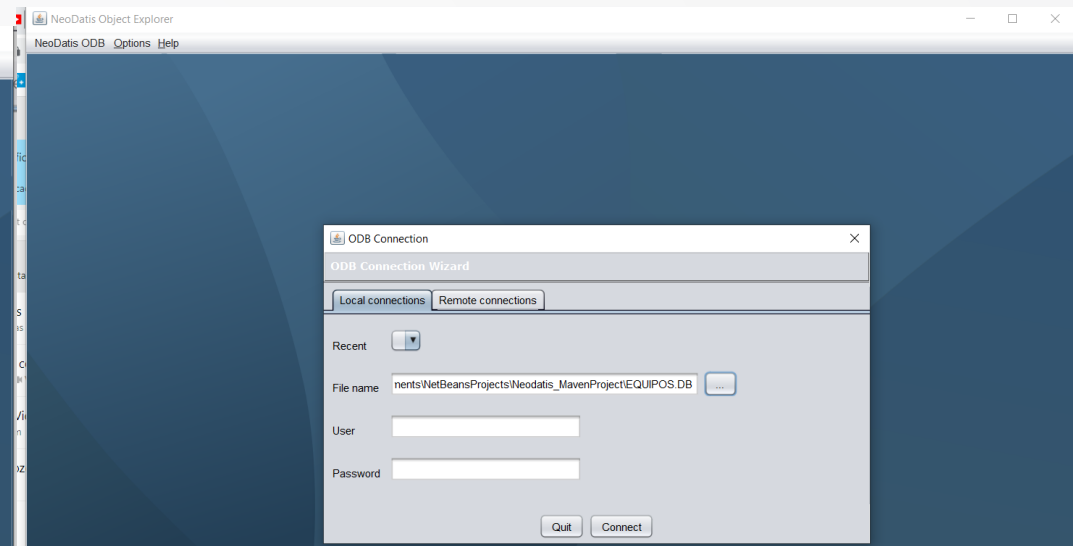
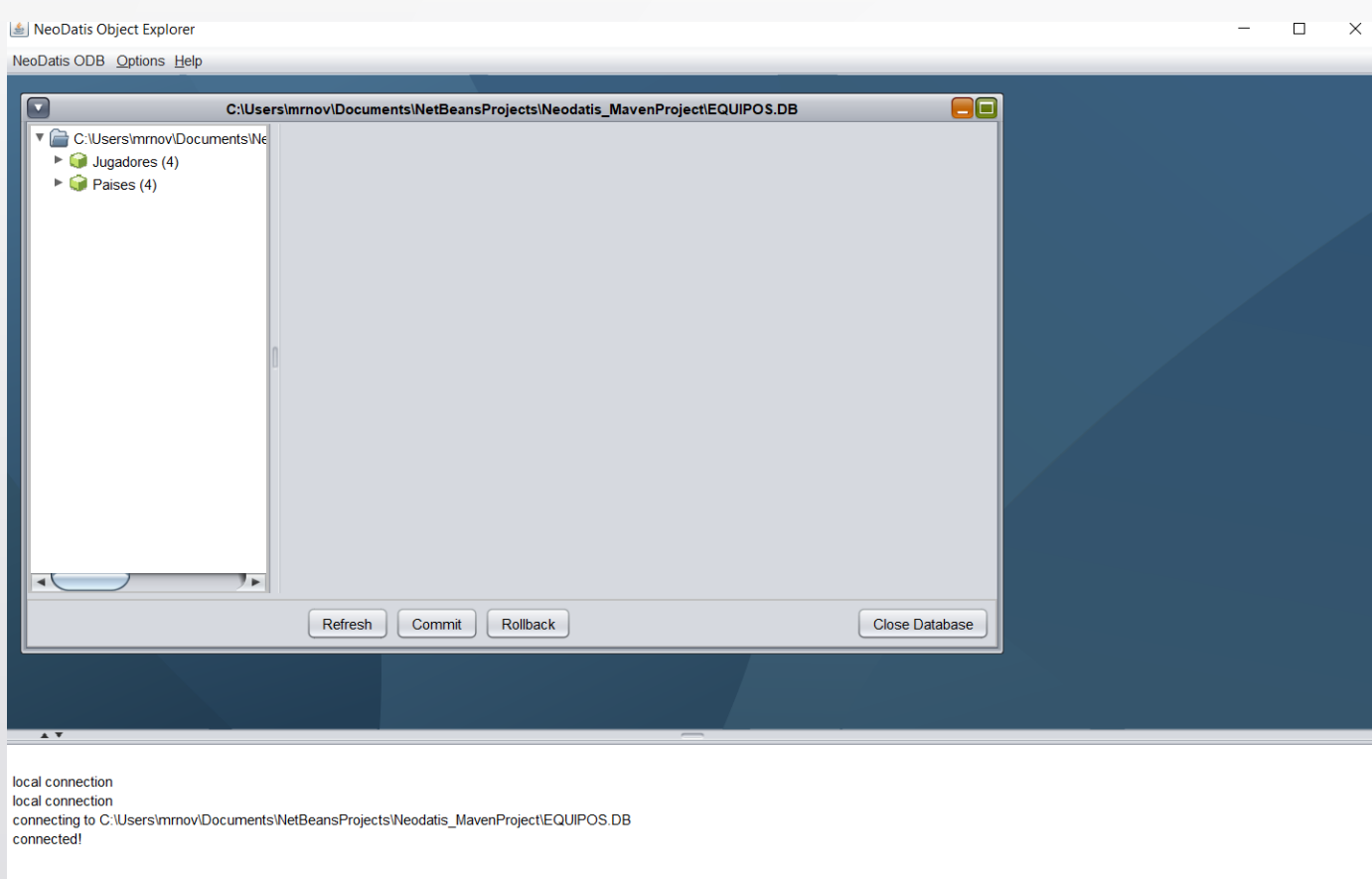


# Abrir BD

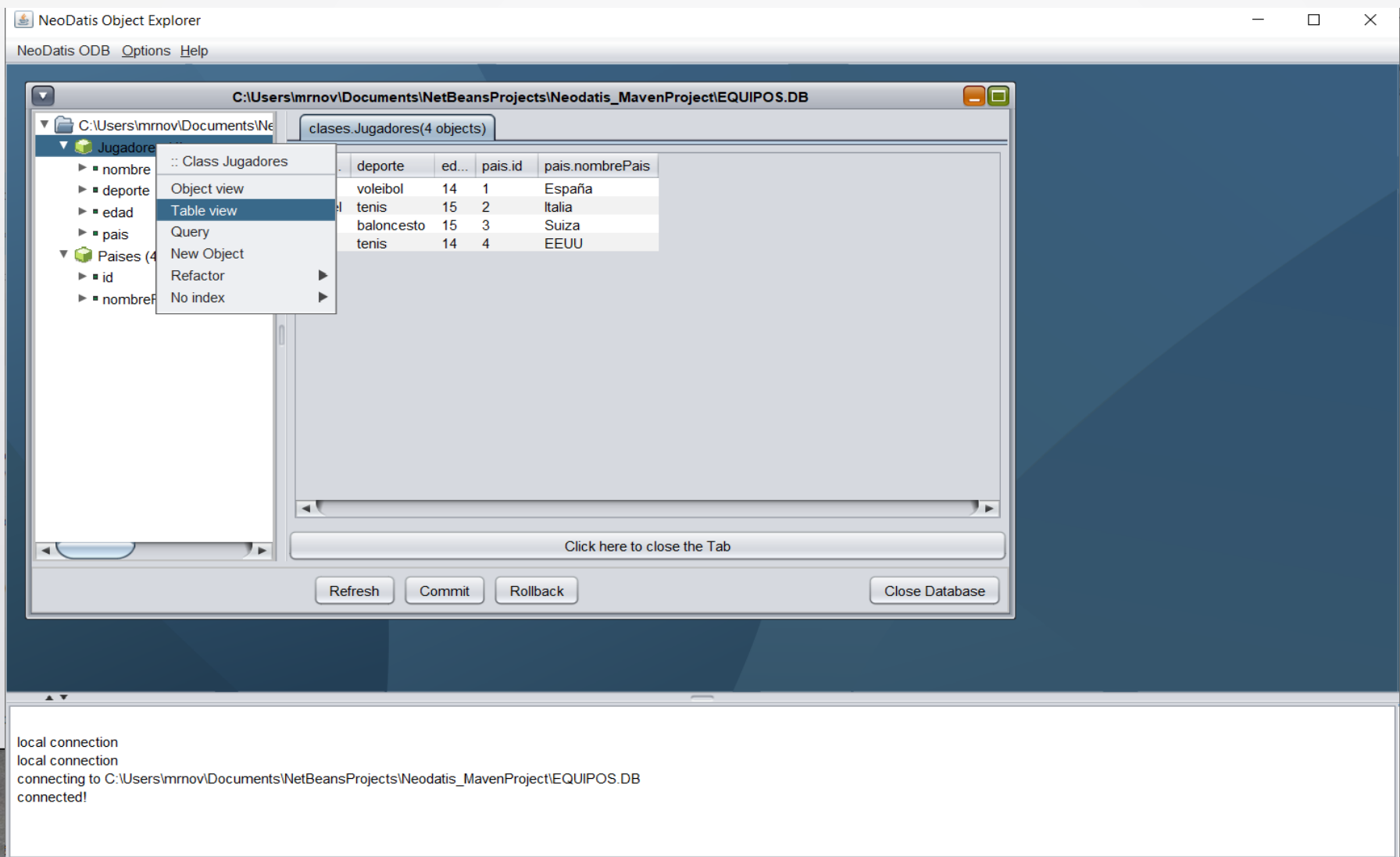




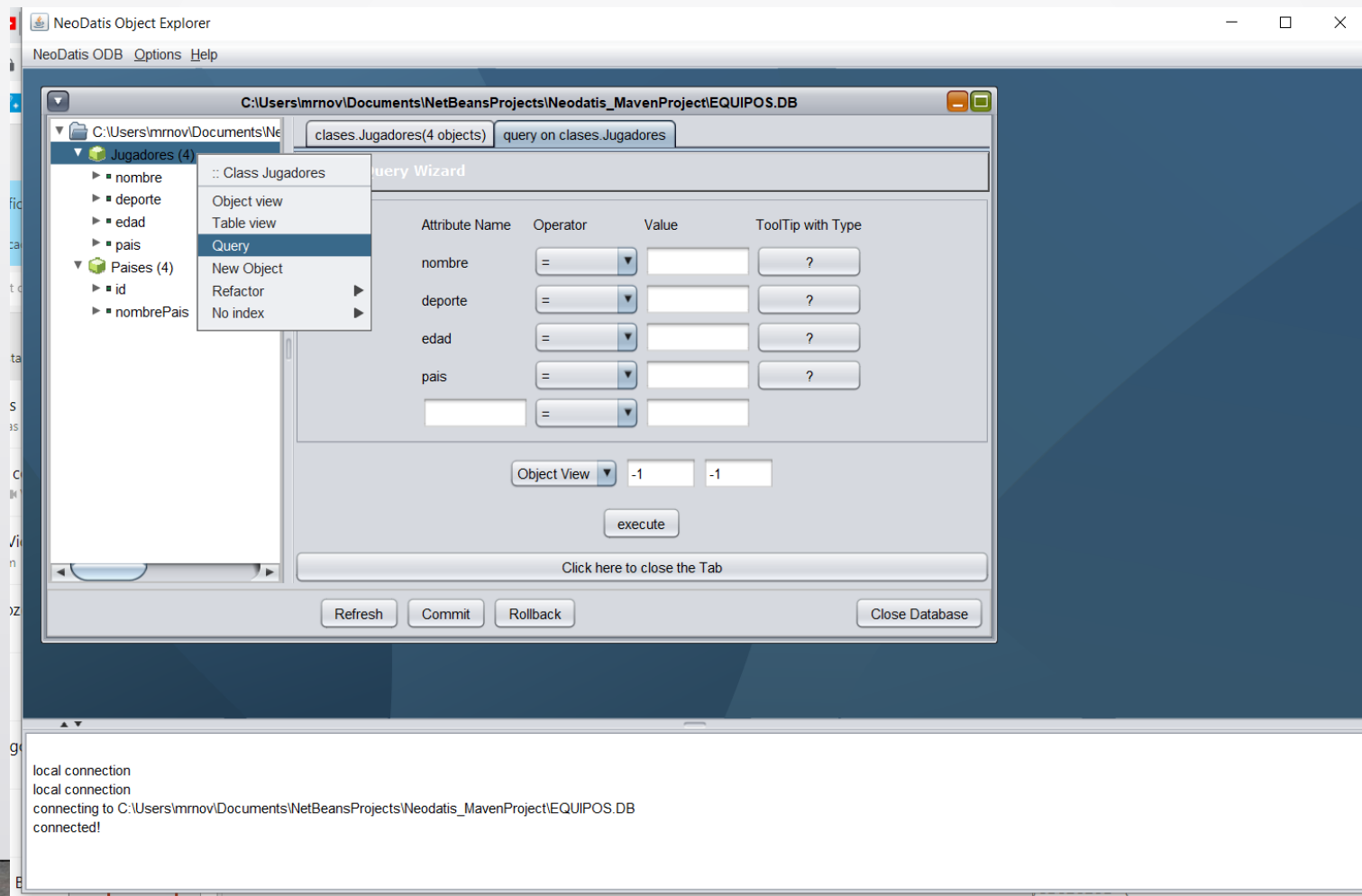
# Cargar BD



# Visualizar Tabla Objetos



# Realizar Consultas



## Consultar Datos Objetos

