Asignatura

Programación

UNIVERSAE
Instituto Superior de FP

Asignatura

Programación

UNIDAD 13

Mantenimiento de la persistencia de los datos

UNIVERSAE
Instituto Superior de FP

Base de datos de objetos (BDOO)

Acercarse al lenguaje de programación

- Tratan directamente con objetos del lenguaje de programación
- No existen tablas, filas ni columnas como en las base de datos relacionales
- Operan con Java, C#, Visual Basic y C++

Ventajas

- Se adaptan al lenguaje de programación
- Menos tiempo de desarrollo y mantenimiento
- Mejor rendimiento

Inconvenientes

- Pocos sistemas que utilices este tipo de base de datos
- Oferta muy limitada

Ejemplos

- Db4o, Objectivity/DB, Objectdb

Características

Acceso rápido

- Trabajan conjuntamente con los lenguajes de programación

Identidad de objetos

- Identificar un objeto inequívocamente.
- Igualdad vs Identidad

Soporte de objetos complejos, Clases y Tipos

- Tratar tipos primitivos y clases.

Extensibilidad

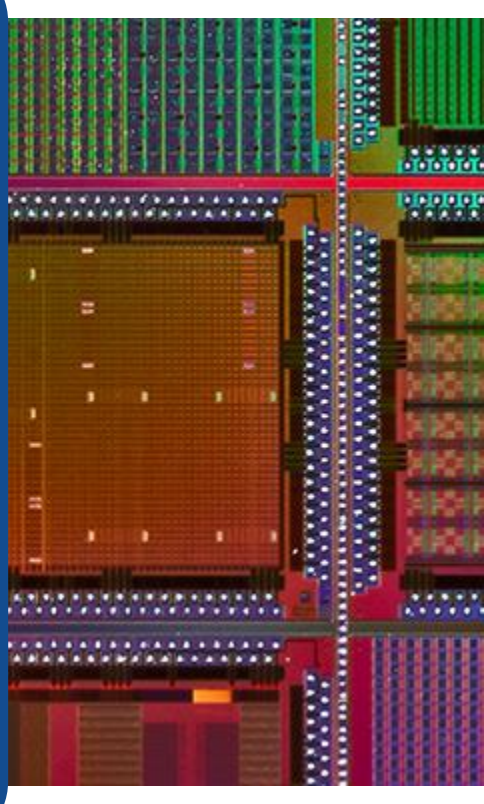
- Definir nuevos tipos

Consultas

- Eficientes e independientes de la base de datos y lenguaje de programación,
- Usan su propio lenguaje de consultas

Completitud computacional

- Disponer de herramientas de manipulación de datos y cálculos



Instalación. ObjectDB

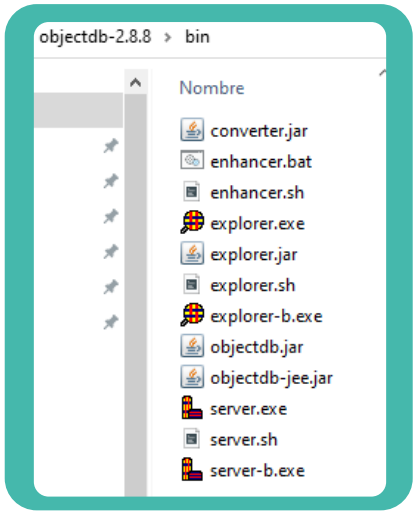
ObjectDB

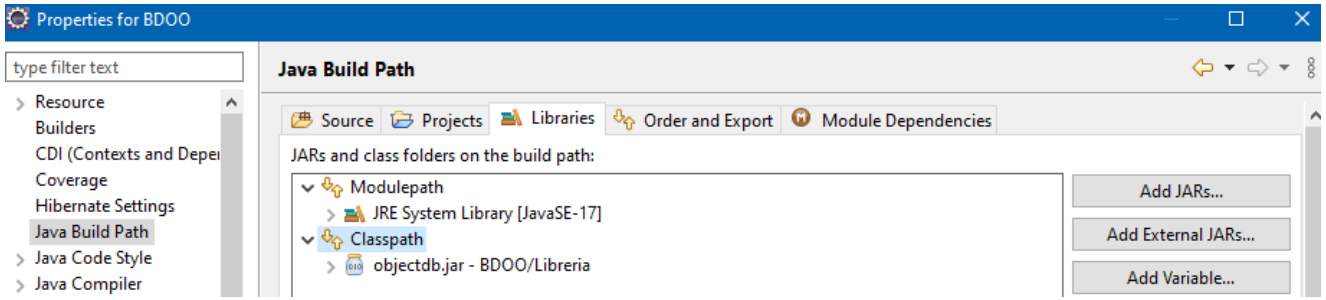
URL: https://www.objectdb.com/

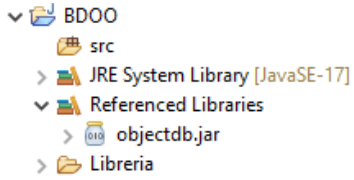
- Gratuito
- Versión actual: 2.8.8

Instalación

1. Descargar Development kit
2. Descomprimir e ir a la carpeta bin
3. Se usará la librería objectdb.jar
4. Crear un proyecto java
5. Añadir la librería al proyecto.
6. Properties sobre el proyecto / Java Build Path





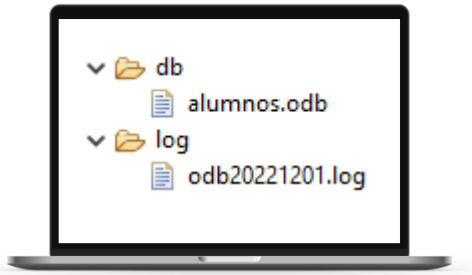


Crear la base de datos

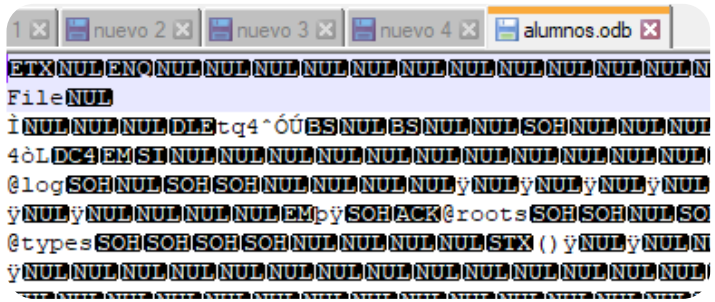
Establecer la conexión

- Uso de la clase *EntityManagerFactory* y *EntityManager*
- Especificar la cadena de conexión
- La cadena de conexión puede ser:
    - Ruta absoluta donde se encuentra la base de datos
    - Uso de la variable $objectdb (referencia donde esta ubicada la librería
    - URL del servidor.

      "objectdb://XXX.odb;user=USUARIO;password=CONTRASEÑA"




```
String rutaBBDD = "$objectdb/db/alumnos.odb";

EntityManagerFactory emf = Persistence.createEntityManagerFactory(rutaBBDD);
EntityManager em = emf.createEntityManager();

// Operar con la base de datos

em.close();
emf.close();
```



Lenguaje de definición de objetos

Java Persistence API (JPA)

- URL: *https://www.w3schools.blog/jpa-tutorial*
- Estándar para manejar y persistir objetos desde Java
- Establece un tipo de etiquetas que actúan sobre clases
- Cada etiqueta va precedido del símbolo @
- Se ubican en el paquete javax.persistence

Tipos	Descripción
Primitivos	Int, float, byte, char, etc..
Estructurados	Array y String. Los arrays deben permitir elementos nulos o instancias de tipo persistente.
Envoltorio	Integer, Float, Double, etc..
Math	BigDecimal, BigInteger
Fecha	Date, Time, Timestamp y Calendar Los tipos del paquete java.sql Los tipos del paquete java.util con el uso de @Temporal
Enumerados	@Enumerated(EnumType.ORDINAL) @Enumerated(EnumType.STRING)
Clases	Definidas por el usuario @Entity
Colecciones o mapas	Que permitan sus elementos nulos o instancias de tipo persistente

Clases

Requisitos

- Campos privados
- Constructor sin argumentos
- Métodos get/set
- Implementar serializable

Sintaxis

Clases

- @Entity
- @Entity(name="Nombre")

Clases compuestas

- @Embedded
- @Embeddable

```java
@Embeddable
public class Direccion implements Serializable {

    private static final long serialVersionUID = -8894324235474321173L;

    private String direccion;
    private String codigoPostal;
    private String poblacion;
    private String provincia;

    public Direccion() {

    }

    public Direccion(String direccion, String codigoPostal,
                     String poblacion, String provincia) {

        this.direccion = direccion;
        this.codigoPostal = codigoPostal;
        this.poblacion = poblacion;
        this.provincia = provincia;
    }

    public String getDireccion() {
        return direccion;
    }

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

    public String getCodigoPostal() {
        return codigoPostal;
    }
```

```java
@Entity
public class Contacto implements Serializable {

    private static final long serialVersionUID = -1226598445347957691L;

    private String nombre;

    @Embedded
    private Direccion direccion;

    private String apellido1;
    private String apellido2;
    private String email;

    public Contacto() {

    }
```

Campos

- Todos los campos de una clase se guardan en la base de datos.
- Se declaran como siempre.
- Deben usar los tipos definidos en JPA
- Se puede excluir campos de ser guardados aquellos que:
  - Se declaren como final, static o transient
  - Los que lleven la etiqueta @Transient

```java
@Entity
public class Campos implements Serializable {

    // Campos que se guardaran en base de datos
    private int campoEntero;
    private String campoCadena;
    private java.sql.Date campoFechaSQL;
    @Temporal(TemporalType.TIME)
    private java.util.Date campoDateUtil;
    @Enumerated(EnumType.STRING)
    private TipoEnumerado campoEnumerado;

    // Campos que no se guardaran en base de datos
    @Transient
    private String campoNoPersistente;
    private final int campoNoPersistente1 = 0;
    private static int campoNoPErsistente2;

}

enum TipoEnumerado {
    VALOR1, VALOR2, VALOR3
}
```

Campos. Clave primaria

Identificador único (PK)

- Cada clase debe disponer un campo que pueda identificar inequívocamente a un objeto
- **@Id**
- Las claves primarias compuestas:
  - Establecer una clase como clave. **@Embeddable**
  - Indicar como campo la clave como clave **@EmbeddedId**

```java
public class Contacto implements Serializable {

    private static final long serialVersionUID

    @Id
    @GeneratedValue
    private long id;
```

```java
@Entity
public class Carrito implements Serializable {

    @EmbeddedId
    private PK identificador;

}
```

```java
@Embeddable
public class PK implements Serializable {

    private int clave1;
    private String clave2;

}
```

¿Como rellenar una clave primaria?

- Manual.
- Automático
  @GeneratedValue
- Estableciendo una estrategia concreta:
  - El proveedor de persistencia elige la estrategia, es igual a @GeneratedValue
    @GeneratedValue(strategy=GenerationType.AUTO)
  - Usa una columna interna de la base de datos
    @GeneratedValue(strategy=GenerationType.IDENTITY)
  - Definir la secuencia.
    @GeneratedValue(strategy=GenerationType.SEQUENCE, generator ="seq")
    @SequenceGenerator(name="seq", initialValue=1, allocationSize=100)

Resumen de etiquetas JPA

Etiqueta	Descripción
@Temporal	Define un campo tipo fecha
@Enumerated(EnumType.ORDINAL)	Especifica un campo enumerado de tipo numérico
@Enumerated(EnumType.STRING)	Especifica un campo enumerado de tipo cadena de caracteres
@Entity	Definir una clase como persistente
@Entity(name="Nombre")	Definir una clase indicando el nombre
@Embedded	Define que es un campo con una clase incrustada
@Embeddable	Define una clase que se incrustará dentro de otra
@Transient	El campo no se guardara en la base de datos
@Id	Define un campo como clave primaria
@EmbeddedId	Define un campo con clave compuesta por otra clase
@GeneratedValue	Especifica que se generara automáticamente la clave primaria
@GeneratedValue(strategy=GenerationType.AUTO)	El proveedor de persistencia escoge la forma de generar la clave primaria. Por defecto es como @GeneratedValue
@GeneratedValue(strategy=GenerationType.IDENTITY)	Usa una columna interna de la base de datos con las mismas características para la clave primaria
@GeneratedValue(strategy=GenerationType.SEQUENCE, generator ="seq")	Usa una secuencia propia para generar la clave primaria
@SequenceGenerator(name="seq", initialValue=1, allocationSize=100)	Define la secuencia que se usa para GenerationType.SEQUENCE

Métodos y propiedades

Propiedades/campos

- Todas tienen que ser privadas
- Uso de las recomendaciones de nomenclatura

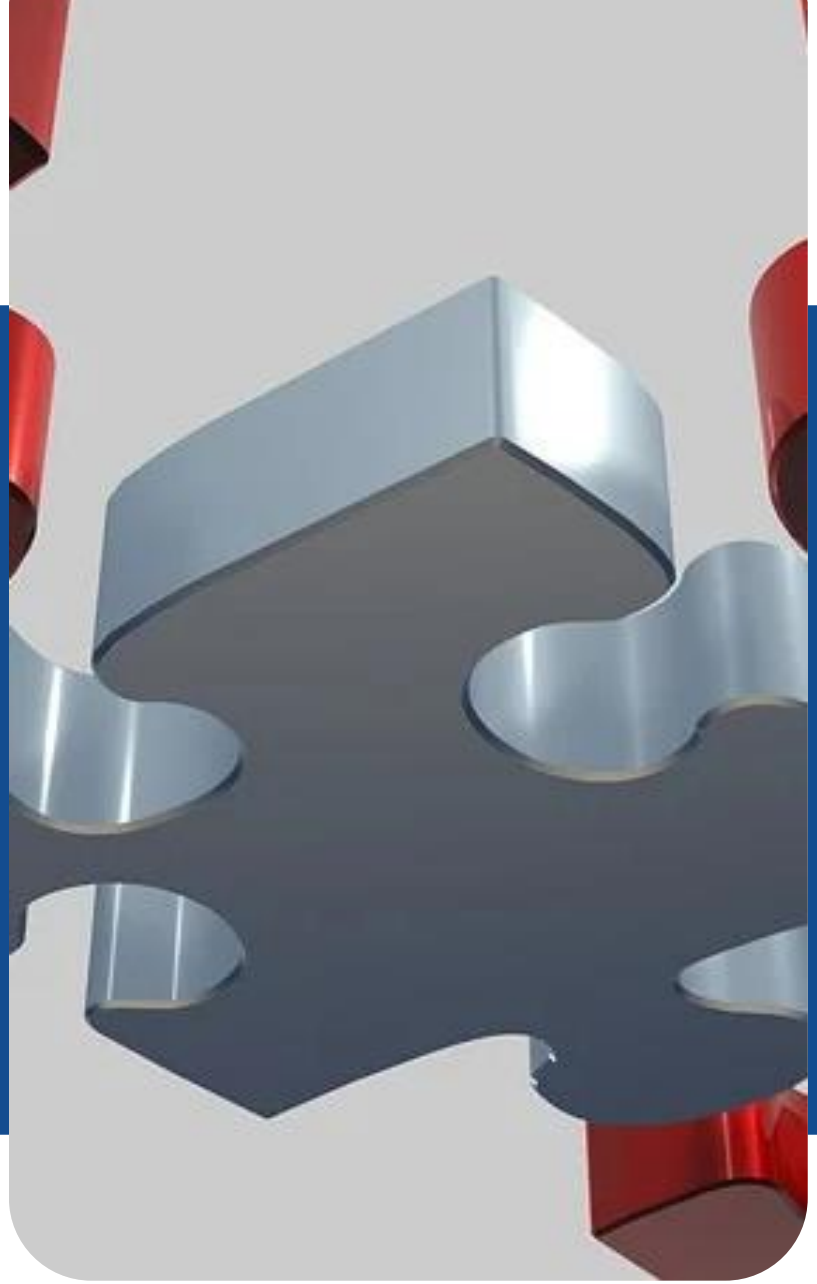
Métodos

- Por cada propiedad debe de haber un método get/set
- Uso nomenclatura:
    - getNombrePropiedad
    - setNombrePropiedad(tipo parámetro)

Constructores

- Debe de haber un constructor vacío
- Su visibilidad debe ser public o protected
- No se puede sobrescribir.
- Se puede añadir nuevos constructores

```java
@Entity
public class Contacto implements Serializable {
    private static final long serialVersionUID = -1226598445347957691L;

    @Id
    @GeneratedValue
    private long id;
    private String nombre;
    @Embedded
    private Direccion direccion;
    private String apellido1;
    private String apellido2;
    private String email;

    public Contacto() {

    }

    public Contacto(String nombre, String apellido1, String apellido2,
                    String email, Direccion direccion) {
        this.nombre = nombre;
        this.apellido1 = apellido1;
        this.apellido2 = apellido2;
        this.email = email;
        this.direccion = direccion;
    }

    public long getId() {

    public void setId(long id) {

    public String getNombre() {

    public void setNombre(String nombre) {

    public String getApellido1() {

    public void setApellido1(String apellido1) {

    public String getApellido2() {

    public void setApellido2(String apellido2) {

    public String getEmail() {

    public void setEmail(String email) {

    @Override
    public String toString() {
        return String.format("(%d, %s, %s, %s, %s, %s)", this.id, this.nombre,
                this.apellido1, this.apellido2, this.email, this.direccion);
    }
}
```



Herencia

- Existe la herencia en las entidades
- Cada clase que interviene en la herencia deben ser persistentes
- No hay etiquetas para definirlas
- Cuando se guarda un objeto de la herencia, también se guarda su correspondencia de la relación

Mecanismos de consulta y operaciones

JPQL (Java Persistence Query Language)

- Lenguaje parecido a SQL
- Trabaja sobre objetos
- Soporta sentencias SELECT, UPDATE y DELETE

```
String rutaBBDD = "$objectdb/db/contactos.odb";

EntityManagerFactory emf = Persistence.createEntityManagerFactory(rutaBBDD);
EntityManager em = emf.createEntityManager();

String consulta = "SELECT c FROM Contacto c";

TypedQuery<Contacto> query = em.createQuery(consulta, Contacto.class);
List<Contacto> results = query.getResultList();
for (Contacto c : results) {
    System.out.println(c);
}

em.close();
emf.close();
```

API Criteria

- Uso del propio lenguaje de programación
- Es mas seguro y no hay que preocuparse por la conversión de tipos

```
String rutaBBDD = "$objectdb/db/contactos.odb";

EntityManagerFactory emf = Persistence.createEntityManagerFactory(rutaBBDD);
EntityManager em = emf.createEntityManager();

CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Contacto> cq = cb.createQuery(Contacto.class);
Root<Contacto> from = cq.from(Contacto.class);
cq.select(from);
TypedQuery<Contacto> q = em.createQuery(cq);
List<Contacto> results = q.getResultList();

for (Contacto c : results) {
    System.out.println(c);
}
```

Inserción

Las transacciones

- Son bloques de sentencia
- Se tienes que ejecutar por completo
- Afecta a :
  - Inserción
  - Actualización
  - Borrado

Sintaxis

- getTransaction().begin()
- Si todo ha ido bien
  - getTransaction().commit()
- Si ha ocurrido algún fallo
  - getTransaction().rollback()

```
String rutaBBDD = "$objectdb/db/contactos.odb";

// Abrir conexión
EntityManagerFactory emf = Persistence.createEntityManagerFactory(rutaBBDD);
EntityManager em = emf.createEntityManager();

em.getTransaction().begin();

Direccion direccion = new Direccion("C/ Calle, 1","00822","Poblacion1","Provincia1");
Contacto contacto = new Contacto("Laura", "Carrillo", "Gonzalez", "laura@email.com",direccion);
Direccion direccion1 = new Direccion("Avd. avenida, 2","30123","Poblacion2","Provincia2");
Contacto contacto1 = new Contacto("Antonio","Baeza","Miró","antonio@email.com",direccion1);

em.persist(contacto);
em.persist(contacto1);

em.getTransaction().commit();

// Cerrar conexión
em.close();
emf.close();
```

Consultas

Directo

```
String rutaBBDD = "$objectdb/db/contactos.odb";

// Abrir conexión
EntityManagerFactory emf = Persistence.createEntityManagerFactory(rutaBBDD);
EntityManager em = emf.createEntityManager();

int identificador = 1;

Contacto contacto = em.find(Contacto.class, identificador);
System.out.println(contacto);

// Cerrar conexión
em.close();
emf.close();
```

Con query dinámica

```
String rutaBBDD = "$objectdb/db/contactos.odb";

// Abrir conexión
EntityManagerFactory emf = Persistence.createEntityManagerFactory(rutaBBDD);
EntityManager em = emf.createEntityManager();

int identificador = 7;
String consulta = "SELECT c FROM Contacto c WHERE id ="+identificador;

TypedQuery<Contacto> query = em.createQuery(consulta, Contacto.class);
List<Contacto> results = query.getResultList();
for (Contacto c : results) {
    System.out.println(c);
}

// Cerrar conexión
em.close();
emf.close();
```

Con query estática

```
@Entity
@NamedQueries({
    @NamedQuery(name="Contacto.consultaTodos", query="SELECT c FROM Contacto c"),
    @NamedQuery(name="Contacto.consultaId", query="select c FROM Contacto c WHERE c.id=:identificador")
})
public class Contacto implements Serializable {

    private static final long serialVersionUID = -1226598445347957691L;
```

```
String rutaBBDD = "$objectdb/db/contactos.odb";

EntityManagerFactory emf = Persistence.createEntityManagerFactory(rutaBBDD);
EntityManager em = emf.createEntityManager();

int identificador = 7;
TypedQuery<Contacto> consultaAlumnos= em.createNamedQuery("Contacto.consultaId", Contacto.class);
consultaAlumnos.setParameter("identificador", identificador);

List<Contacto> results = consultaAlumnos.getResultList();

for (Contacto c : results) {
    System.out.println(c);
}
```

Actualización

Directo

```
String rutaBBDD = "$objectdb/db/contactos.odb";

// Abrir conexión
EntityManagerFactory emf = Persistence.createEntityManagerFactory(rutaBBDD);
EntityManager em = emf.createEntityManager();

int identificador = 7;

Contacto contacto = em.find(Contacto.class, identificador);
em.getTransaction().begin();
contacto.setNombre("Carla");
em.getTransaction().commit();

// Cerrar conexión
em.close();
emf.close();
```

Con query

```
String rutaBBDD = "$objectdb/db/contactos.odb";

// Abrir conexión
EntityManagerFactory emf = Persistence.createEntityManagerFactory(rutaBBDD);
EntityManager em = emf.createEntityManager();

em.getTransaction().begin();
int identificador = 7;
String query = "UPDATE Contacto SET nombre = \"Susana\" WHERE id = "+identificador;
Query consulta = em.createQuery(query);
int resultado = consulta.executeUpdate();
em.getTransaction().commit();

// Cerrar conexión
em.close();
emf.close();
```

Borrado

Directo

```
String rutaBBDD = "$objectdb/db/contactos.odb";

// Abrir conexión
EntityManagerFactory emf = Persistence.createEntityManagerFactory(rutaBBDD);
EntityManager em = emf.createEntityManager();

Contacto contacto = em.find(Contacto.class, 1);
em.getTransaction().begin();
em.remove(contacto);
em.getTransaction().commit();

// Cerrar conexión
em.close();
emf.close();
```

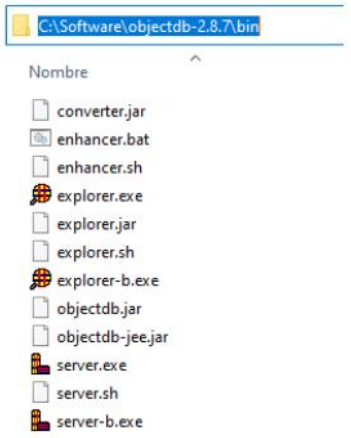
Con query

```
String rutaBBDD = "$objectdb/db/contactos.odb";

// Abrir conexión
EntityManagerFactory emf = Persistence.createEntityManagerFactory(rutaBBDD);
EntityManager em = emf.createEntityManager();

em.getTransaction().begin();
int identificador = 7;
String query = "DELETE FROM Contacto WHERE id = "+identificador;
Query consulta = em.createQuery(query);
int resultado = consulta.executeUpdate();
em.getTransaction().commit();

// Cerrar conexión
em.close();
emf.close();
```

Explorar de objetos

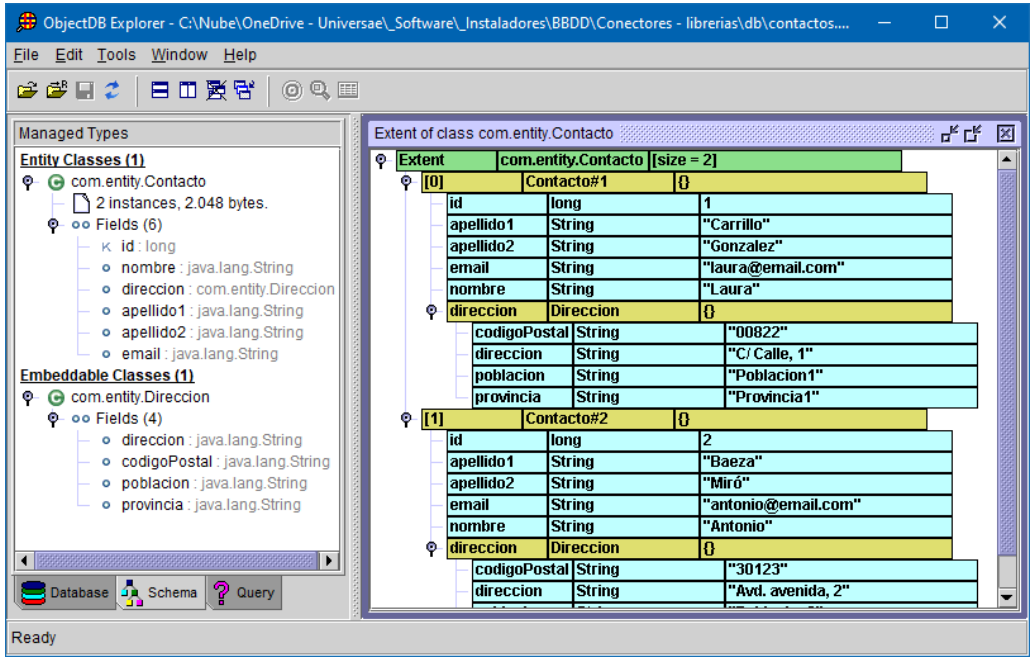
Herramienta externa

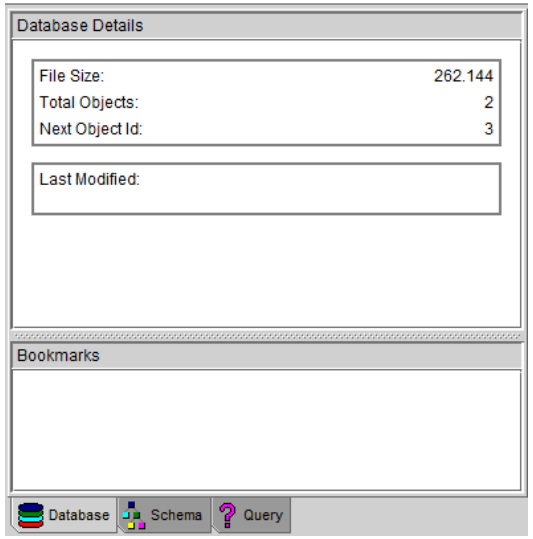
- Permite operar en la base de datos
- Modo gráfico
- \bin\explorer-b.exe

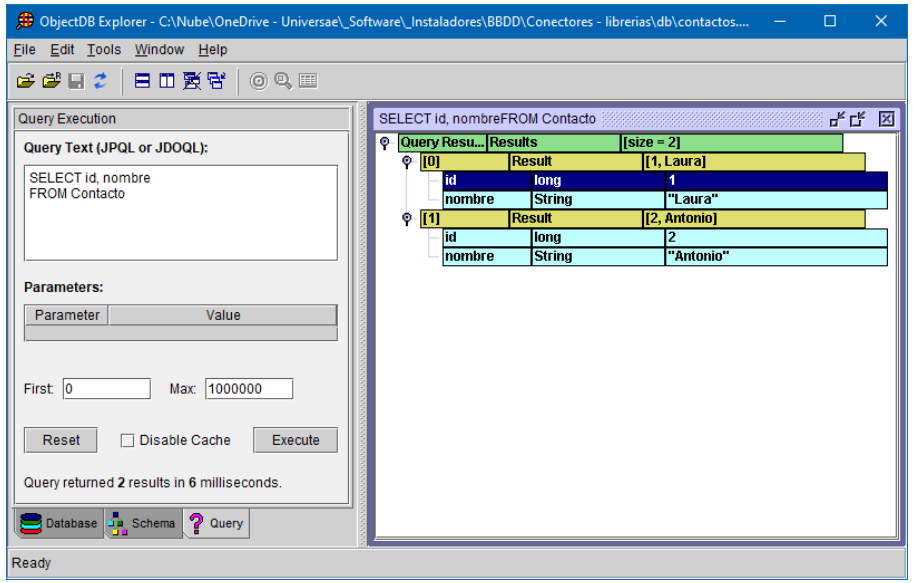


Pestañas

- Database
- Schema
- Query







Resumen

1. Base de datos de objetos (BDOO)
2. Características
3. Instalación *ObjectDB*
4. Creación de la base de datos
5. Lenguaje de definición de objetos
6. Clases
7. Campos
8. Campos. Clave primaria
9. Resumen de etiquetas
10. Métodos y propiedades
11. Herencia
12. Mecanismos de consulta
13. Inserción
14. Consultas
15. Modificación
16. Borrado
17. Explorador de objetos

UNIVERSAE

CHANGE YOUR WAY