

Bases de datos objeto-relacionales y orientadas a objetos.

Caso práctico

Esta mañana **Juan**, **María** y **Ana** han quedado a primera hora en el despacho de **Ada**. Según les comentó ayer **Ada**, antes de cerrar, urge comenzar a planificar un proyecto nuevo que ha llegado a la empresa. Se trata de desarrollar una aplicación informática para una empresa de la industria de la madera. La empresa se dedica a la fabricación de mobiliario de diseño, así como a su distribución y venta.



La aplicación que desarrollen debe gestionar un gran volumen de piezas diferentes, piezas de chapa y madera, permitiendo diseñarlas desde la forma más simple a la más compleja. También, debe permitir desplegar y gestionar directamente la evolución de la pieza añadiendo pliegues, cortes, deformaciones, etc. Por descontado, debe gestionar también el almacenamiento de todas esas piezas y los productos finales.

Ada y **Juan** tienen claro que en esta aplicación no tiene cabida el uso de bases de datos relacionales, ya que habrá que almacenar un gran volumen de datos diferentes, con propiedades y comportamientos que no son fijos, y con complejas relaciones entre ellos. Será necesario un tipo de bases de datos más avanzado, posiblemente orientadas a objetos u objeto-relacionales, sin las limitaciones que presentan las bases de datos relacionales, entre ellas, que solo son capaces de manejar estructuras muy simples (filas y columnas).

Ana ha estado algo callada durante la reunión, sabe que ella va a echar una mano en este proyecto junto a **Antonio**, y necesita repasar muchísimo sobre bases de datos avanzadas. Lo primero que ha decidido hacer es coger sus apuntes del ciclo de DAM e ir echando un vistazo a las características de las bases de datos orientadas a objetos y objeto-relacionales.

1.- Introducción.

Las Bases de Datos Relacionales (BDR) son ideales para aplicaciones tradicionales que soportan tareas administrativas, y que trabajan con datos de estructuras simples y poco cambiantes, incluso cuando la aplicación pueda estar desarrollada en un lenguaje OO y sea necesario un Mapeo Objeto Relacional (ORM)

Pero cuando la aplicación requiere otras necesidades, como por ejemplo, soporte multimedia, almacenar objetos muy cambiantes y complejos en estructura y relaciones, este tipo de base de datos no son las más adecuadas. Recuerda, que si queremos representar un objeto y sus relaciones en una BDR esto implica que:



- ✓ Los objetos deben ser descompuestos en diferentes tablas.
- ✓ A mayor complejidad, mayor número de tablas, de manera que se requieren muchos enlaces (joins) para recuperar un objeto, lo cual disminuye dramáticamente el rendimiento.

Las **Bases de Datos Orientadas a Objetos** (BDOO) o Bases de Objetos se integran directamente y sin problemas con las aplicaciones desarrolladas en lenguajes orientados a objetos, ya que **soporan un modelo de objetos puro** y son ideales para almacenar y recuperar datos complejos permitiendo a los usuarios su navegación directa (sin un mapeo entre distintas representaciones).

Las Bases de Objetos aparecieron a finales de los años 80 motivadas fundamentalmente por dos razones:

- ✓ Las necesidades de los lenguajes de Bases de Datos Orientadas a Objetos (POO), como la necesidad de persistir objetos.
- ✓ Las limitaciones de las bases de datos relacionales, como el hecho de que sólo manejan estructuras muy simples (tablas) y tienen poca riqueza semántica

Pero como las BDOO no terminaban de asentarse, debido fundamentalmente a la inexistencia de un estándar, y las BDR gozaban y gozan en la actualidad de una gran aceptación, experiencia y difusión, debido fundamentalmente a su gran robustez y al lenguaje SQL, los fabricantes de bases de datos comenzaron a implementar nuevas funcionalidades orientadas a objetos en las BDR existentes. Así surgieron, las bases de datos objeto-relacionales.



Las **Bases de Datos Objeto-Relacionales** (BDOR) son bases de datos relacionales que han evolucionado hacia una base de datos más extensa y compleja, incorporando conceptos del modelo orientado a objetos. Pero en estas bases de datos **aún existe un mapeo de objetos subyacente**, que es costoso y poco flexible, cuando los objetos y sus interacciones son complejos.

Para saber más

Si consultas las páginas 5-9 del documento de este enlace, verás la propuesta de STONEBREAKER sobre la idoneidad de uno u otro sistema de bases de datos, en función de la aplicación que vayamos a desarrollar.

[Bases de datos que almacenan objetos. \(0.82 MB\) \(pdf - 844.01 KB\)](#).

2.- Características de las bases de datos orientadas a objetos.

Caso práctico

Ana se ha llevado a la empresa sus apuntes sobre bases de datos y acceso a datos del ciclo formativo, y ha empezado a repasar con Juan las características de las Bases de Objetos.

—¡Pero claro!, —le dice Juan a Ana— habrá que detenerse en valorar tanto las posibles ventajas como los inconvenientes de estas bases de datos, ya que tenemos que elegir el sistema de almacenamiento óptimo para esta aplicación. Ya sabes que no podemos fallar. ¡Somos los mejores! —y le sonríe.



En una BDOO, los datos se almacenan como objetos. Un **objeto** es, al igual que en POO, una entidad que se puede identificar únicamente y que describe tanto el estado como el comportamiento de una entidad del 'mundo real'. El estado de un objeto se describe mediante atributos y su comportamiento es definido mediante procedimientos o métodos.

Entonces, ¿a qué equivalen las entidades, ocurrencias de entidades y relaciones del modelo relacional? Las entidades son las clases, las ocurrencias de entidad son objetos creados desde las clases, las relaciones se mantienen por medio de inclusión lógica, y no existen claves primarias, los objetos tienen un identificador.

La principal característica de las BDOO es que **soportan un modelo de objetos puro y que el lenguaje de programación y el esquema de la base de datos utilizan las mismas definiciones de tipos**.



Otras características importantes de las BDOO son las siguientes:

- ✓ **Soportan las características propias de la Orientación a Objetos** como agregación, encapsulamiento, polimorfismo y herencia. La herencia se mantiene en la propia base de datos.
- ✓ **Identificador de objeto (OID)**. Cada objeto tiene un identificador, generado por el sistema, que es único para cada objeto, lo que supone que cada vez que se necesite modificar un objeto, habrá que recuperarlo de la base de datos, hacer los cambios y almacenarlo nuevamente. Los OID son independientes del contenido del objeto, esto es, si cambia su información, el objeto sigue teniendo el mismo OID. Dos objetos serán equivalentes si tienen la misma información pero diferentes OID.
- ✓ **Jerarquía y extensión de tipos**. Se pueden definir nuevos tipos basándose en otros tipos predefinidos, cargándolos en una jerarquía de tipos (o jerarquía de clases).
- ✓ **Objetos complejos**. Los objetos pueden tener una estructura de objeto de complejidad arbitraria, a fin de contener toda la información necesaria que describe el objeto.
- ✓ **Acceso navegacional de datos**. Cuando los datos se almacenan en una estructura de red densa y probablemente con una estructura de diferentes niveles de profundidad, el acceso a datos se hace principalmente navegando la estructura de objetos y se expresa de forma natural utilizando las construcciones nativas del lenguaje, sin necesidad de uniones o joins típicas en las BDR.
- ✓ **Gestión de versiones**. El mismo objeto puede estar representado por múltiples versiones. Muchas aplicaciones de bases de datos que usan orientación a objetos requieren la existencia de varias versiones del mismo objeto, ya que si estando la aplicación en funcionamiento es necesario modificar alguno de sus módulos, el diseñador deberá crear una nueva versión de cada uno de ellos para efectuar cambios.

Autoevaluación

Señala las opciones correctas. Las bases de datos orientadas a objetos:

- Soportan conceptos de orientación a objetos como la herencia.
Cuestionario
- Permiten la manipulación navegacional.
Cuestionario
- Tienen el mismo tipo de problemas que las relacionales para gestionar objetos complejos.
Cuestionario
- Cada objeto posee un identificador de objeto.
Cuestionario

[Mostrar retroalimentación](#)

Solución

1. Correcto
2. Correcto
3. Incorrecto
4. Correcto

2.1.- Ventajas e inconvenientes.

El uso de una BDOO puede ser ventajoso frente a una BDR relacional si nuestra aplicación requiere alguno de estos elementos :

- ✓ Un gran número de tipos de datos diferentes.
- ✓ Un gran número de relaciones entre los objetos.
- ✓ Objetos con comportamientos complejos.

Una de las principales ventajas de los sistemas de bases de datos orientados a objetos es la **transparencia**, (manipulación directa de datos utilizando un entorno de programación basado en objetos), por lo que el programador, solo se debe preocupar de los objetos de su aplicación, en lugar de cómo los debe almacenar y recuperar de un medio físico.

Otras ventajas de un sistema de bases de datos orientado a objetos son las siguientes:

- ✓ **Gran capacidad de modelado.** El modelado de datos orientado a objetos permite modelar el 'mundo real' de una manera óptima gracias al encapsulamiento y la herencia.
- ✓ **Flexibilidad.** Permiten una estructura cambiante con solo añadir subclases.
- ✓ **Soporte para el manejo de objetos complejos.** Manipula de forma rápida y ágil objetos complejos, ya que la estructura de la base de datos está dada por referencias (apuntadores lógicos) entre objetos.
- ✓ **Alta velocidad de procesamiento.** Como el resultado de las consultas son objetos, no hay que reensamblar los objetos cada vez que se accede a la base de objetos.
- ✓ **Extensibilidad.** Se pueden construir nuevos tipos de datos a partir de los ya existentes, agrupar propiedades comunes de diversas clases e incluirlas en una superclase, lo que reduce la redundancia.
- ✓ **Mejora los costes de desarrollo,** ya que es posible la reutilización de código, una de las características de los lenguajes de programación orientados a objetos.
- ✓ **Facilitar el control de acceso y concurrencia,** puesto que se puede bloquear a ciertos objetos, incluso en una jerarquía completa de objetos.
- ✓ Funcionan de forma **eficiente en entornos cliente/servidor y arquitecturas distribuidas.**

Pero aunque los sistemas de bases de datos orientados a objetos pueden proporcionar soluciones apropiadas para muchos tipos de aplicaciones avanzadas de bases de datos, también tienen sus **desventajas**. Éstas son las siguientes:

- ✓ **Carencia de un modelo de datos universal.** No hay ningún modelo de datos aceptado universalmente, y la mayor parte de los modelos carecen de una base teórica.
- ✓ **Falta de estándares.** Existe una carencia de estándares general para los sistemas de BDOO.
- ✓ **Complejidad.** La estructura de una BDOO es más compleja y difícil de entender que la de una BDR.
- ✓ **Competencia de otros modelos.** Las bases de datos relacionales y objeto-relacionales están muy asentadas y extendidas, siendo un duro competidor.
- ✓ **Difícil optimización de consultas.** La optimización de consultas requiere una compresión de la implementación de los objetos, para poder acceder a la base de datos de manera eficiente. Sin embargo, esto compromete el concepto de encapsulación.

Citas para pensar

"Cada día sabemos más y entendemos menos".

Albert Einstein

Autoevaluación

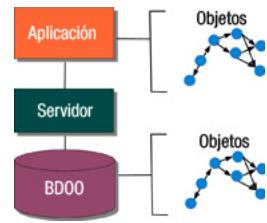
Señala si la siguiente afirmación es verdadera o falsa.

En una BDOO el resultado de una consulta son objetos, por lo que no es necesario reensamblar los objetos cada vez que se accede a la base de datos.

Verdadero Falso

Verdadero

Es verdadero, precisamente ésta es una de las ventajas de estos sistemas de almacenamiento.



3.- Gestores de bases de datos orientadas a objetos.

Caso práctico

Esta mañana, **Juan** ha preparado una serie de sistemas de bases de objetos diferentes, para instalarlos e ir probando algunas de sus características.



Juan le pregunta a **Ana** —¿Sabes qué al contrario de las bases de datos relacionales, los gestores de bases de datos orientados a objetos pueden llegar a ser muy diferentes entre sí?

A lo que **Ana** responde —Sí, en clase estuvimos trabajando con diferentes sistemas, precisamente para dar fe de ello, y..., quiero recordar que había una que tenía el nombre de un pintor francés. ¿Matisse?

—Efectivamente —dice **Juan**—, pero hoy he pensado en ver la base de objetos Db4o, que es la que actualmente utiliza el sistema de trenes español AVE. ¿Lo sabías?

Un **Sistema Gestor de Bases de Datos Orientada a Objetos** (SGBDOO) y en inglés **ODBMS**, Object Databases Management System) es un software específico, dedicado a servir de interfaz entre la base de objetos, el usuario y las aplicaciones que la utilizan. Un SGBDOO incorpora el paradigma de Orientación a Objetos y permite el almacenamiento de objetos en soporte secundario:

- ✓ Por ser SGBD debe incluir mecanismos para optimizar el acceso, gestionar el control de concurrencia, la seguridad y la gestión de usuarios, así como facilitar la consulta y recuperación ante fallos.
- ✓ Por ser OO incorpora características de identidad, encapsulación, herencia, polimorfismo y control de tipos.



Cuando aparecieron las bases de datos orientadas a objetos, un grupo formado por desarrolladores y usuarios de bases de objetos, denominado **ODMG** (Object-Oriented Database Management Group), propuso un estándar que se conoce como estándar **ODMG-93** y que se ha ido revisando con el tiempo, pero que en realidad **no ha tenido mucho éxito**, aunque es un punto de partida.

Debes conocer

Desde el siguiente enlace te puedes descargar un resumen de las características del estándar ODMG.

[Resumen del estándar ODMG.-](#)

¿Qué estrategias o enfoques se siguen para el desarrollo de SGBDOO? Básicamente, las siguientes:

- ✓ Ampliar un lenguaje de programación OO existente con capacidades de BD (Ejemplo: GemStone).
- ✓ Proporcionar bibliotecas de clases con las capacidades tradicionales de las bases de datos, como persistencia, transacciones, concurrencia, etc., (Ejemplo: ObjectStore y Versant).
- ✓ Ampliar un lenguaje de BD con capacidades OO, caso de SQL 2003 y Object SQL (OQL, propuesto por ODMG).

Tal y como estarás pensando, la carencia de un estándar real hace difícil el soporte para la portabilidad de aplicaciones y su interoperabilidad, y es en parte por ello, que a diferencia de las bases de datos relacionales donde hay muchos productos donde elegir, la variedad de sistemas de bases de datos orientadas a objetos es mucho menor. En la actualidad hay diferentes productos de este tipo, tanto con licencia libre como propietaria.

A continuación te indicamos algunos **ejemplos de SGBOO**:

- ✓ **Db4o de Versant.** Es una BDOO Open Source para Java y .NET. Se distribuye bajo licencia GPL. La base de datos DB4o se encuentra actualmente descontinuada. La empresa que la compró en 2008 dejó de soportar sus actualizaciones en 2015. Avisan de que en breve pueden dejar de ofrecer el software en sus repositorios. [Información](#)
- ✓ **Matisse.** Es un SGBOO basado en la especificación ODMG, proporciona lenguajes para definición y manipulación de objetos, así como interfaces de programación para C, C++, Eiffel y Java.
- ✓ **ObjectDB.** Es una BDOO que ofrece soporte para Java, C++, y Python entre otros lenguajes. No es un producto libre, aunque ofrecen versiones de prueba durante un periodo determinado.
- ✓ **EyeDB.** Es un SGBOO basado en la especificación ODMG, proporciona lenguajes para definición y manipulación de objetos, e interfaces de programación para C++ y Java. Se distribuye bajo licencia GNU y es software libre.
- ✓ Neodatis, ObjectStore y GemStone. Son otros SGBDOO.

3.1.- Objetos simples y objetos estructurados.

En las BDOO los objetos se encuentran interrelacionados por referencias entre ellos de manera similar a como los objetos se referencian entre sí en memoria.

Un **objeto de tipo simple** u objeto simple es aquel que no contiene a otros objetos y por tanto posee una estructura de un solo nivel de profundidad en este sentido.

Un **objeto de tipo estructurado** u objeto estructurado incluye entre sus componentes a otros objetos y se define aplicando los constructores de tipos disponibles por el SGBDOO recursivamente a varios niveles de profundidad.

Entre un objeto y sus componentes de cada nivel, existen dos tipos de referencia:

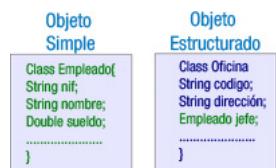
- ✓ **Referencia de propiedad.** Se aplica cuando los componentes de un objeto se encapsulan dentro del propio objeto y se consideran, por tanto, parte de ese objeto. **Relación es-parte-de.** No necesitan tener identificadores de objeto y sólo los métodos de ese objeto pueden acceder a ellos. Desaparecen si el propio objeto se elimina.
- ✓ **Referencia de asociación.** Se aplica cuando entre los componentes del objeto estructurado existen objetos independientes, pero es posible hacer referencia a ellos desde el objeto estructurado. **Relación está-asociado-con.** Cuando un objeto estructurado tiene que acceder a sus componentes referenciados, lo hace invocando los métodos apropiados de los componentes, ya que no están encapsulados dentro del objeto estructurado.

Por ejemplo, si observas la figura superior derecha, en un objeto tipo Oficina los componentes `codigo` y `dirección` son_parte_del objeto, mientras que el componente `jefe`, es un objeto independiente que `está_asociado_con` el objeto oficina.

Entonces, ¿las referencias de asociación son como las relaciones del modelo relacional? Así es:

- ✓ La referencia de asociación representa las relaciones o interrelaciones entre objetos independientes, dando la posibilidad de que los objetos puedan detectarse mutuamente en una o dos direcciones, (lo que en el modelo relacional representamos mediante claves ajenas o foráneas que provienen de relaciones uno a uno, uno a muchos y muchos a muchos).
- ✓ Una relación uno a muchos se representa mediante un objeto tipo colección (List, Set, etc.). En una BDOO la colección se maneja como cualquier otro objeto (aunque potencialmente profundo) y normalmente se podrá recuperar y almacenar el objeto padre junto con la colección asociada y su contenido en una sola llamada.

Además, un objeto miembro referenciado puede ser referenciado por más de un objeto estructurado y, no se elimina automáticamente cuando se elimina el objeto del nivel superior.



Debes conocer

En la siguiente presentación encontrarás información sobre las colecciones o tipo collection de Java:

http://www.slideshare.net/slideshow/embed_code/884346



[Colecciones en Java](#) from [Ronny Parra](#)

[Resumen textual alternativo](#)

Autoevaluación

Señala si la siguiente afirmación es verdadera o falsa.

Un objeto estructurado es aquel que contiene a otros objetos.

Verdadero Falso

Verdadero

Es verdadero, efectivamente es estructurado cuando contiene a otro u otros objetos y en diferentes niveles de profundidad.

3.2.- Instalación del gestor de objetos Db4o.

Java, disponible para Java y .Net, y utilizada en la actualidad por diversas compañías para el desarrollo de aplicaciones de dispositivos móviles, dispositivos médicos y biotecnología, aplicaciones web, software enlatado y aplicaciones en tiempo real.



Tres **características importantes de Db4o** son las siguientes:

- ✓ El modelo de clases es el propio esquema de la base de datos, por lo que se elimina el proceso de diseño, implementación y mantenimiento de la base de datos.
- ✓ Está diseñada bajo la estrategia de proporcionar bibliotecas de clases con las capacidades tradicionales de las bases de datos, y con el objetivo de cero administración.
- ✓ Puede trabajar como **base de datos embebida**, lo que significa que se puede distribuir con la aplicación, y solo la aplicación que lanza la base de datos embebida puede acceder a ella, siendo esta invisible para el usuario final.

Una de las ventajas de db4o es que es un simple fichero .jar distribuible con cualquier aplicación sin necesidad de tener que instalar nada. Tampoco necesita drivers de tipo JDBC o similar.

La instalación de db4o consistirá en:

- ✓ Instalar el motor de base de datos, que son las clases necesarias para hacer que funcione la API en toda su extensión.
- ✓ Instalar alguna aplicación para visualizar los datos con los que se está trabajando. Esto último es necesario, pues en otro caso se trabajaría a ciegas con los datos.

En el siguiente recurso didáctico encontrarás un tutorial con los pasos a seguir para la descarga del software db4o y realizar su instalación e integración con el IDE NetBeans.

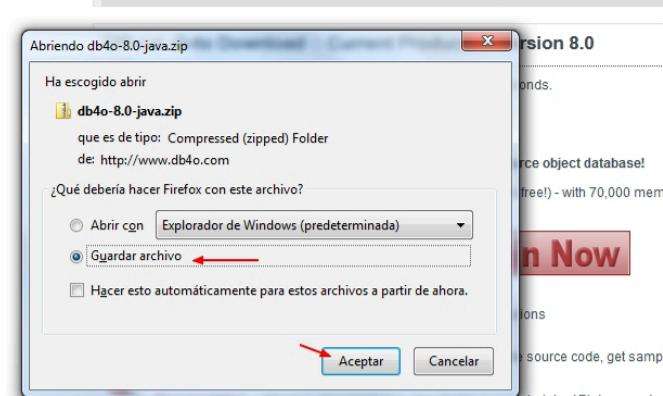
Db4o: Instalación en Windows 7

Db4o: Instalación en Windows 7

- ✓ Accedemos a la web oficial de Db4o <http://www.db4o.com/>
- ✓ Pulsamos el botón **Downloads Now**
- ✓ En la siguiente página seleccionamos el producto a descargar (**db4o.8.0 for Java**) y pulsamos el botón **Download**.

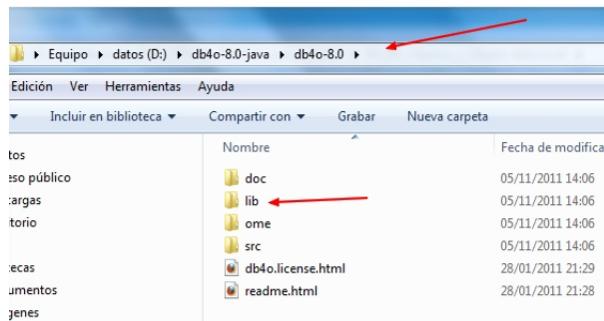
Descarga del software

En breves instantes nos aparece la ventana que nos indica la descarga del **archivo db4o-8.0-java.zip**. Marcamos **Guardar** y Pulsamos **Aceptar**.



Descomprimir archivo .zip

- ✓ Hay que **descomprimir** el archivo **db4o-8.0-java.zip**.
- ✓ La carpeta descomprimida tendrá la siguiente estructura de directorios y archivos:
 - ◆ **doc**: incluye documentación y un tutorial en formato Javadoc y PDF.
 - ◆ **lib**: librerías en formato JAR
 - ◆ **ome**: object manager enterprise. Visor de objetos.
 - ◆ **src**: código fuente de las librerías



El motor de base de datos

Db4o tiene varias configuraciones para su versión 8.0.

- **db4o-8.0-core*.jar**. Archivos core que contienen el núcleo del motor
- **db4o-8.0-cs*.jar**. Archivos cs que contiene la versión cliente/servidor
- **db4o-8.0-optional*.jar**. Archivos que añaden funcionalidad avanzada al motor de base de datos.
- **db4o-8.0-all*.jar**. Contiene todas las características anteriores.
- **Instalación completa**

Db4o también está disponible para diferentes versiones de JDK.

Para la instalación completa existen las siguientes versiones:

- ✓ **db4o-8.0-all-java1.1.jar**. Compatible con los JDK que proporcionen compatibilidad con JDK 1.1.x
- ✓ **db4o-8.0-all-java1.2.jar**. Desarrollado para los JDK entre versiones 1.2 y 1.4
- ✓ **db4o-8.0-all-java5.jar**. Desarrollo para los JDK 5 y 6.

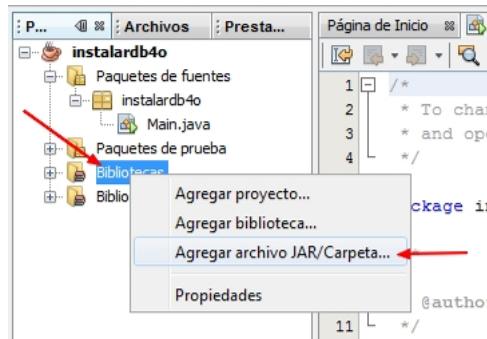
El archivo .jar que interesa

- ✓ Dentro del **directorio lib** se encuentran las librerías en formato . Jar
- ✓ El archivo que nos interesa es **db4o-8.0*-all-java5.jar**

Nombre	Fecha de modifica...	Tipo	Tamaño
3rdpartylibs.html	28/01/2011 21:29	Firefox Document	2 KB
ant.jar	28/01/2011 21:29	Executable Jar File	1.260 KB
ant.license.html	28/01/2011 21:29	Firefox Document	4 KB
bloat.license.html	28/01/2011 21:29	Firefox Document	22 KB
bloat-1.0.jar	28/01/2011 21:29	Executable Jar File	694 KB
db4o-8.0.184.15484-all-java5.jar	28/01/2011 21:29	Executable Jar File	2.559 KB
db4o-8.0.184.15484-bench.jar	28/01/2011 21:29	Executable Jar File	30 KB
db4o-8.0.184.15484-core-java5.jar	28/01/2011 21:29	Executable Jar File	1.423 KB
db4o-8.0.184.15484-cs.optional-java5.jar	28/01/2011 21:29	Executable Jar File	18 KB
db4o-8.0.184.15484-cs.java5.jar	28/01/2011 21:29	Executable Jar File	201 KB
db4o-8.0.184.15484-db4ounit-java5.jar	28/01/2011 21:29	Executable Jar File	242 KB
db4o-8.0.184.15484-instrumentation-java...	28/01/2011 21:29	Executable Jar File	59 KB
db4o-8.0.184.15484-nqopt-java5.jar	28/01/2011 21:29	Executable Jar File	69 KB
db4o-8.0.184.15484-optional-java5.jar	28/01/2011 21:29	Executable Jar File	115 KB
db4o-8.0.184.15484-osgi-java5.jar	28/01/2011 21:29	Executable Jar File	2.265 KB
db4o-8.0.184.15484-osgi-test-java5.jar	28/01/2011 21:29	Executable Jar File	2.949 KB
db4o-8.0.184.15484-taj JAVA5.jar	28/01/2011 21:29	Executable Jar File	23 KB
db4o-8.0.184.15484-tools-java5.jar	28/01/2011 21:29	Executable Jar File	4 KB
readme.html	28/01/2011 21:28	Firefox Document	6 KB

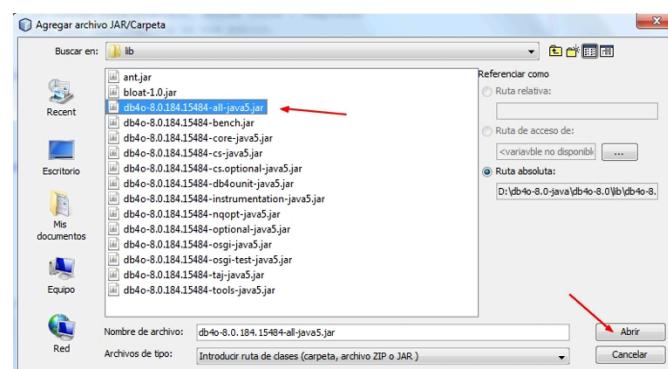
Integrar db4o en el IDE NetBeans (I)

- ✓ Creamos o Abrimos un **proyecto** en el que trabajaremos con db4o
- ✓ Nos situamos sobre **Bibliotecas** y desde su menú contextual (clic derecho)
- ✓ Seleccionamos la opción **Agregar archivo JAR/Carpeta**



Instalación dentro del IDE NetBeans (II)

- ✓ Seleccionamos el .jar que nos interesa, en este caso, **db4o-8.0*-all-java5.jar** y pulsamos **Abrir**



Instalación dentro del IDE NetBeans (III)

- ✓ También puedes copiar el fichero **db4o-8.0*-all-java5.jar** al raíz de tu proyecto y seleccionarlo de esa ubicación indicando una **ruta relativa**. De esta forma, si cambias de equipo, siempre tendrás disponible la librería, aunque esté en otra ubicación.

Comprobamos

- ✓ Podemos comprobar que el archivo .jar se ha incluido en las Bibliotecas del proyecto.

Credenciales

Imagen	Datos licencia
Capturas de pantalla de esta de las diapositivas 2, 3, 4 y 6 de esta presentación.	Autoría: Db4o Licencia: GNU GPL. Procedencia: Captura de pantalla de la instalación de la aplicación Db4o
Capturas de pantalla de esta de las diapositivas 7, 8, 9, y 10 de esta presentación.	Autoría: Isabel M. Cruz Granados Licencia: Uso educativo-no comercial. Procedencia: Captura de pantalla del programa NetBeans, propiedad Sun Microsystems, bajo licencia GNU GPL v2
Capturas de pantalla de esta de la diapositiva 5 de esta presentación	Autoría: Isabel Cruz Granados. Tipo de licencia: Copyright (Cita). Procedencia: Captura de pantalla del sistema operativo Microsoft Windows 7

1 2 3 4 5 6 7 8 9 10

[Resumen textual alternativo](#)

[Software DB4o \(.zip - 39.53 MB\)](#) (.zip - 39.53 MB).

Para saber más

Versant ha dejado de apoyar económicamente al proyecto Db4o y los enlaces de db4o.com no funcionan, pero aún se puede encontrar la aplicación, su API y tutoriales en SourceForge, en el archivo .zip que se encuentra en el siguiente enlace está todo (justo arriba os he dejado el archivo .zip con la última versión).

[Página de SourceForge con los archivos del proyecto Db4o](#)

4.- El API de la base de objetos.

Caso práctico

Ana no puede negar su entusiasmo con el hecho de participar en el desarrollo del nuevo proyecto. Cuando estudió Acceso a Datos en el Ciclo Formativo, nunca pensó que llegaría a trabajar tan de cerca con las bases de datos avanzadas. Ha pasado en el descanso a ver a su amigo **Antonio** y ponerlo al día de los avances que va realizando junto a **Juan**. Hoy le espera un ardua sesión de trabajo, familiarizarse con el API de la base de objetos Db4o.

Todos los SGBDOO, independientemente de su estrategia de diseño, proporcionan un API (Interfaz de Programación de Aplicaciones), más o menos extenso, disponible para ciertos lenguajes OO. En el caso de Db4o, el API está disponible para Java y .Net.

Los principales paquetes del API de Db4o son los siguientes:

- ✓ com.db4o. Paquete principal (core) de la Base de Objetos. Las interfaces y clases más importantes que incluye son:
 - ◆ ObjectContainer. Es el interfaz que permite realizar las principales tareas con la base de objetos. Un ObjectContainer puede representar **una base de datos independiente** (stand-alone) o una **conexión a un servidor** (en línea cliente-servidor). Este interfaz proporciona métodos para almacenar store(), consultar queryByExample() y eliminar delete() objetos de la base de datos, así como cerrar la conexión a ésta close(). También permite confirmar commit y deshacer rollback transacciones.
 - ◆ EmbeddedObjectContainer. Es un interfaz que extiende a ObjectContainer y representa un ObjectContainer local atacando a la base de datos.
 - ◆ Db4oEmbedded. Es una clase que proporciona métodos estáticos para conectar con la base de datos en modo embebido.
 - ◆ ObjectServer. Es el interfaz que permite trabajar con una base de datos db4o en modo cliente-servidor.
 - ◆ ObjectSet. Es un interfaz que representa el conjunto de objetos devueltos por una consulta.
- ✓ com.db4o.query. Paquete con funcionalidades de consulta. Proporciona interfaces que permiten establecer las condiciones y criterios de un consulta y una clase para realizar consultas mediante Native Query (Consultas nativas).
- ✓ com.db4o.config. Paquete con funcionalidades de configuración. Contiene interfaces y clases que nos permiten configurar y/o personalizar la base de objetos según necesidades. La configuración de la base de objetos se hace por norma general antes de abrir la sesión en la misma.
 - ◆ EmbeddedConfiguration. Es la interface de configuración para el uso en modo embebido.

Siempre que trabajemos con bases de objetos Db4o utilizaremos el interface ObjectContainer, puesto que es quien representará a la base de objetos, sea embebida o no.

La documentación del API viene en formato JavaDoc y la puedes encontrar en el directorio /doc/api del fichero .zip descargado y descomprimido

[Db4o](#) (zip - 39.53 MB).

Autoevaluación

Señala si la siguiente afirmación es verdadera o falsa.

Un ObjectContainer es un interfaz que proporciona, entre otros, los métodos store(), queryByExample(), close() y delete().

- Verdadero Falso

Verdadero

Es verdadero, es el interfaz que permite realizar las principales tareas con la base de datos.

4.1.- Apertura y cierre de conexiones.

En general, la conexión de una aplicación Java con una base de objetos se podrá realizar vía:

- ✓ JDBC.
- ✓ El API proporcionado por el propio gestor de objetos.

En el caso de Db4o, el paquete com.db4o proporciona las clases e interfaces que permiten abrir conexiones a una base de objetos db4o, así como el cierre de la misma. Estas son:

- ✓ **Abrir conexión.** Podemos utilizar las siguientes clases:
 - ◆ Db4oEmbedded. Es una clase que hereda de java.lang.Object y proporciona métodos estáticos como openFile() para abrir una instancia de la base de datos en **modo embebido**. En modo embebido tiene la limitación de que solo se puede utilizar en la base de datos una conexión.
 - ◆ ObjectServer. Es una interfaz que permite trabajar con una base de datos db4o en **modo cliente-servidor**. Una vez abierta la base de datos como servidor mediante Db4o.openServer(), el método openClient() de la interfaz ObjectServer permitirá **abrir conexiones cliente directamente en memoria o bien mediante TCP/IP**.
- ✓ **Cerrar conexión.** Para ello utilizaremos el método close() de la interfaz ObjectContainer.

El **esquema de trabajo para operar con la base de objetos** será el siguiente:

- ✓ Declarar un ObjectContainer.
- ✓ Abrir una conexión a la base de objetos (openFile()). Si al abrir la BDOO esta no existe, se creará.
- ✓ Realizar operaciones con la base de objetos como consultas, borrados y modificaciones (en un bloque try{} catch (){}).
- ✓ Cierre o desconexión de la base de objetos (close()).

En el siguiente recurso didáctico tienes un ejemplo de apertura, creación y cierre a una base de objetos en modo embebido. En la BDOO creada se almacenan 4 objetos mediante el método store().

Db4o: Creación de la BDOO Congreso

Db4o: Creación de la BDOO Congreso

Congreso será una **base de objetos** que permitirá almacenar información sobre los ponentes y charlas de un congreso de Informática, esto es, permitirá persistir objetos tipo ponente y objetos tipo charla.

- ✓ Un ponente puede impartir varias charlas
- ✓ Una charla es impartida por un solo ponente

Las clases necesarias son las siguientes:

- ✓ **Clase ponente**
 - ◆ **Atributos:** NIF, nombre, email, cache
 - ◆ **Métodos:** los que permiten asignar y obtener esos valores, y un método para imprimir esos valores.
- ✓ **Clase charla**
 - ◆ **Atributos:** título, duración, objeto ponente
 - ◆ **Métodos:** los que permiten asignar y obtener esos valores, y un método para imprimir esos valores.

Lo que haremos

- ✓ Crearemos el **proyecto Congreso**
- ✓ En el proyecto crearemos la **clase ponente** que implementa a un objeto ponente. Más adelante crearemos la clase charla.
- ✓ Desde el **método main()** realizaremos la **conexión, operaciones y cierre de la base de objetos** siguiendo el esquema visto en la teoría:

- ✓ Físicamente, la base de objetos será un **fichero de nombre congreso.db4o** almacenado **en el directorio raíz del proyecto**.
- ✓ Recuerda, que debes haber descargado el software db4o y haber descomprimido el .zip en tu equipo, tal y como te indicamos en la anterior presentación.

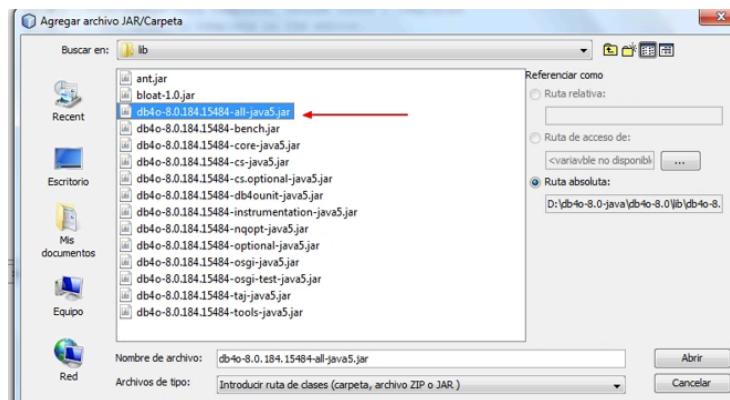


Crear el proyecto y agregar API db4o

- ✓ Desde el IDE NetBeans creamos un nuevo **proyecto de nombre Congreso**
- ✓ Nos situamos sobre el proyecto, en el **nodo Bibliotecas**.
- ✓ Hacemos un clic derecho sobre Bibliotecas
- ✓ Ejecutamos la opción **Agregar archivo JAR/Carpeta...**

Crear el proyecto y agregar API db4o

- ✓ Seleccionamos el JAR de nuestra carpeta local en nuestro caso el archivo **db4o-8.0.184.15484-all-java5.jar**
- ✓ Pulsamos el **botón Abrir**



Recuerda que este archivo .jar también lo podemos copiar al raíz de nuestro proyecto y agregarlo desde esa ubicación

Clase ponente

- ✓ Creamos la **clase ponente** que implementará la entidad ponente, con:
 - ↳ **Atributos:** nif, nombre, e-mail y caché
 - ↳ **Tres constructores**
 - ↳ **Métodos:** que permiten asignar y obtener estos atributos
 - ↳ **Método `toString`** que devuelve los atributos de un objeto ponente

The screenshot shows a Java development environment with the following details:

- Project Explorer:** Shows a project named "Congreso" containing packages "congreso" and "prueba". Inside "congreso", there are files "Main.java" and "ponente.java". Inside "prueba", there is a file "db4o-8.0.184.15484-all-java5.jar".
- Code Editor:** Displays the content of the "ponente.java" file.
- Output:** Shows the Java code for the "ponente" class, which includes constructors, basic assignment methods, and an overridden toString() method.

```

13 public class ponente {
14     private String nif;
15     private String nombre;
16     private String email;
17     private float cache;
18     //constructores
19     public ponente() {...}
20     public ponente(String ni, String n, String e) {...}
21
22     public ponente(String ni, String no, String e, float c) {...}
23     //métodos básicos para asignar y obtener valores de atributos
24     public void setNif(String n) {...}
25     public String getNif() {...}
26     public void setNombre(String n) {...}
27     public String getNombre() {...}
28     public void setEmail(String e) {...}
29     public String getEmail() {...}
30     public void setCache(float c) {...}
31     public float getCache() {...}
32     @Override
33     //comportamiento del método toString heredado de la superclase Obj
34     //Devuelve los atributos de un objeto ponente
35     public String toString() {
36         if (this.cache != -1) {
37             return this.nif+" "+this.nombre+" "+this.email+" Caché:"+this.cache;
38         } else {
39             return this.nif+" "+this.nombre+" "+this.email;
40         }
41     }

```

Clase Main: conexión a la base de objetos

- ✓ En la **clase Main** realizamos la **conexión o apertura de la base de datos orientada a objetos**, y la asociamos al fichero de nombre **congreso.db4o**
- ✓ La ejecución de la siguiente **sentencia** creará la base de objetos si no existe, o bien abrirá la base de datos existente con ese nombre.

```
ObjectContainer db =
Db4oEmbedded.openFile(Db4oEmbedded.newConfiguration(),
"congreso.db4o");
```

- ✓ Los paquetes del API de db4o que se necesitan son:
 - com.db4o.Db4oEmbedded
 - com.db4o.ObjectContainer

Clase Main: almacenar objetos con store()

- ✓ El método **almacenarPonentes(ObjectContainer db)** crea 4 objetos ponente
- ✓ Con el metodo **store()** del interfaz **ObjectContainer** almacenamos objetos.
- ✓ **Invocamos almacenarPonentes()** desde un bloque **try ... finally** y con **close()** desconectanos o cerramos la base de objetos, una vez finalizadas las tareas.

Ejecución del proyecto

- ✓ La ejecución del proyecto, en nuestro caso, creará un fichero de nombre **congreso.db4o** en el raíz de la carpeta de nuestro proyecto **Congreso**, que es la BDOO creada y que contiene los datos de 4 objetos tipo ponente.

- ✓ Si ejecutamos otra vez el proyecto, se añadirán 4 objetos ponente con los mismos datos a nuestra base de objetos, tendríamos por tanto 8 objetos ponente, pero con datos repetidos.

Credenciales

Imagen	Datos de licencia
Todas las capturas de pantalla de esta presentación tienen como datos de licencia: Autoría: Isabel M. Cruz Granados Licencia: Uso educativo-no comercial. Procedencia: Captura de pantalla del programa NetBeans, propiedad Sun Microsystems, bajo licencia GNU GPL v2.	

1 2 3 4 5 6 7 8 9

[Resumen textual alternativo](#)

[Código del ejemplo de creación de una base de objetos db4o.](#) (17 kB)

Autoevaluación

Señala la opción correcta. Para cerrar una conexión a una base de datos db4o se utiliza el método:

- closedb().
- close() de la clase Db4oEmbedded.
- disconnect() del interfaz ObjectContainer.
- close() del interfaz ObjectContainer.

No es correcto, prueba de nuevo.

No es la respuesta correcta, deberías haber leído mejor.

No, ese método no existe.

Muy bien, vamos por buen camino.

Solución

1. Incorrecto
2. Incorrecto
3. Incorrecto
4. Opción correcta

4.2.- Consultas a la base de objetos.

A una BDOO se podrán realizar consultas mediante:

- ✓ Un lenguaje de consultas como OQL, si el gestor está basado en el estándar ODMG e incluye sentencias del tipo SQL.
- ✓ El API proporcionado por el propio sistema gestor de bases de datos orientadas a objetos.

Los tres sistemas de consulta que proporciona Db4o basados en el API del propio gestor, son los siguientes:

- ✓ **Consultas por ejemplo. Query By Example (QBE).** Es la forma más sencilla y básica de realizar consultas, pero tienen bastantes limitaciones.
- ✓ **Consultas nativas. Native Queries (NQ).** Es la interfaz principal de consultas de la base de objetos. Permiten realizar un filtro contra todas las instancias de la base de objetos.
- ✓ **Consultas SODA. Simple Object Data Access (SODA).** Permite generar consultas dinámicas. Es más potente que las anteriores y más rápida, puesto que las anteriores (QBE y NQ) tienen que ser traducidas a SODA para ejecutarse.

¿En qué consiste cada uno de estos sistemas de consulta?

En el siguiente recurso didáctico encontrarás un resumen y ejemplos de estos sistemas de consulta. Los ejemplos se realizan sobre la base de objetos Congreso.

Db4o: Sistemas de consulta



Db4o: Sistemas de consulta



Db4o admite 3 sistemas de consulta:

- ✓ **Consultas por ejemplo o consultas QBE.**
 - ↳ Es la forma más sencilla y básica de realizar consultas.
 - ↳ Tienen bastantes restricciones.
- ✓ **Consultas Nativas**
 - ↳ Permiten realizar un filtro contra todas las instancias de la base de objetos.
 - ↳ Son la interfaz principal de consultas de Db4o.
- ✓ **Consultas SODA.**
 - ↳ Permiten generar consultas en tiempo de ejecución.
 - ↳ Todas las consultas se traducen a consultas SODA para ejecutarse.

Consultas QBE

¿En qué consiste una consulta QBE?

Las **consultas QBE** (Query By Example) se basan en suministrar a db4o un objeto que sirva de plantilla o prototipo de consulta.

- ✓ Mediante el método `queryByExample()` de la interface `ObjectContainer` Db4o retornará todos los objetos que coincidan con la plantilla.
- ✓ El resultado será una instancia de un `ObjectSet`.

Limitaciones:

- ✓ Hay que **proporcionar un ejemplo**, esto es, un objeto prototipo.
- ✓ **No se pueden realizar expresiones avanzadas**, como por ejemplo usar los operadores AND, OR y NOT.
- ✓ **No se puede preguntar por ciertos objetos**, en concreto aquellos cuyo valor de un campo numérico sea 0, strings vacíos o algún campo que sea null. (Estos son los valores que se utilizan en el prototipo para seleccionar cualquier objeto)
- ✓ **Se necesita un constructor** para crear los objetos con valores no inicializados

Ejemplo 1 de consultas QBE

EJEMPLO 1

Para recuperar todos los objetos, tipo ponente, de la base de objetos Congreso, mediante QBE, habrá que pasar un objeto prototipo vacío al método `queryByExample()`, lo que indica al sistema que se deben recuperar todos los objetos.

Donde **mostrarConsulta()** es un método el cual tomará como parámetro un **ObjectSet** (conjunto de objetos) que irá recorriéndolo y mostrando uno a uno los objetos recuperados. El código del método es el siguiente

```
//Método para mostrar objetos recuperados de la Base de Objetos
public static void mostrarConsulta(ObjectSet resul) {
    System.out.println("Recuperados " + resul.size() + " Objetos");
    while (resul.hasNext()) {
        System.out.println(resul.next());
    }
}
```

Ejemplo 2 y 3 de consultas QBE

EJEMPLO 2

Para consultar un ponente o ponentes en concreto, por ejemplo el o los de cierto caché, por ejemplo caché 200, se le pasa a **queryByExample()** el objeto prototipo con valor 200 en el campo caché.

EJEMPLO 3

Para consultar un ponente o ponentes en concreto, por nombre, se le pasa a **queryByExample()** el objeto prototipo con parámetro **nomb** que le indicará el nombre.

Consultas SODA

¿En qué consiste una consulta SODA?

Las **consultas SODA** utilizan la API SODA (Simple Object Data Access) de db4o para realizar consultas, permitiendo redactar consultas en tiempo de ejecución.

- ✓ La consulta se representa como un grafo con nodos.
- ✓ La clase **Query** es la interfaz del **query-graph**. Se necesitará un objeto tipo **Query**
- ✓ **constrain()** agrega una condición de restricción a un nodo
- ✓ **descend()** mueve de un nodo a otro (navega por los nodos)

- ✓ Se necesita el paquete **com.db4o.query**
- ✓ Al objeto **Query** creado se le añaden las **Constraints** que modelan la consulta que se desea obtener.
- ✓ Una vez construida la consulta, esta se ejecuta mediante el método **execute()** del objeto tipo **Query**.
- ✓ Son las consultas más rápidas y potentes, ya que db4o transforma cualquier consulta en una consulta SODA para ser ejecutada.

Ejemplo 1 de consultas SODA

EJEMPLO 1

Para consultar todos los ponentes, la única restricción es seleccionar la **clase ponente**, mediante **query.constrain(ponente.class)**.

Donde **mostrarConsulta()** es un método el cual tomará como parámetro un **ObjectSet** (conjunto de objetos) que irá recorriéndolo y mostrando uno a uno los objetos recuperados. El código del método es el siguiente

Ejemplo 2 de consultas SODA

EJEMPLO 2

Para consultar ponentes de caché 200, se indicará la restricción **constrain(200)**.

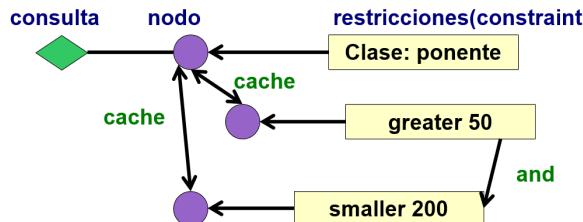
Se construirá una consulta con una serie de nodos en los que se irá descendiendo para comprobar las restricciones o **constraints** establecidos y así incluir o excluir los posibles candidatos de ese nodo.

El código será el siguiente:

Ejemplo 3 de consultas SODA

EJEMPLO 3

Para consultar los ponentes con caché entre 50 y 200, habrá que enlazar dos restricciones. Para ello se necesitará declarar un objeto Constraint que haga una de las restricciones y después enlazar con la otra restricción.



Consultas Nativas

¿En qué consiste una consulta Nativa o NQ?

Las **consultas NQ** (Native Query) son la interfaz principal de consultas y se basan en el **uso de la semántica del lenguaje de programación**.

- ✓ Permiten realizar un filtro contra todas las instancias de una clase.
- ✓ Deberán retornar 'verdadero' para incluir determinadas instancias dentro del conjunto de resultados.
- ✓ Db4o realiza una optimización de las expresiones utilizadas mediante un procesador de consultas que intenta resolverlas usando índices, sin necesidad de instanciar los objetos reales donde sea posible.

Ejemplo 1 de consultas NQ

EJEMPLO 1

Para consultar los ponentes con un caché igual a 200, mediante una consulta Nativa **NQ**, escribiríamos el siguiente código.

Donde **mostrarConsulta()** es un método el cual tomará como parámetro un **ObjectSet** (conjunto de objetos) que irá recorriéndolo y mostrando uno a uno los objetos recuperados. El código del método es el siguiente

Credenciales

Todas las capturas de pantalla de esta presentación tienen como datos de licencia:
Autoría: Isabel M. Cruz Granados
Licencia: Uso educativo-no comercial.
Procedencia: Captura de pantalla del editor de código NetBeans, propiedad Sun Microsystems, bajo licencia GNU GPL v2.

1 2 3 4 5 6 7 8 9 10 11
[Resumen textual alternativo](#)

Desde el siguiente enlace puedes descargar el proyecto completo en el que se realizan consultas por los diferentes sistemas vistos anteriormente a la base de objetos congreso.db4o.

[Código del ejemplo de diferentes sistemas de consulta a la base de objetos.](#) (20,4 KB)

Autoevaluación

Señala la opción correcta. Las consultas SODA en db4o:

- Son muy limitadas y no permiten el uso de restricciones.
- Necesitan de objetos Query para formularlas y en ellas se pueden indicar varias restricciones o constraints.
- No necesitan ningún API especial.
- Son más lentas que las consultas nativas.

No es correcto, prueba de nuevo.

Muy bien, vamos por buen camino.

No es cierto, precisamente utilizan la API SODA.

No, al contrario, son las más rápidas.

Solución

1. Incorrecto
2. Opción correcta
3. Incorrecto
4. Incorrecto

Para saber más

En el siguiente enlace, dispones de información sobre los sistemas de consulta del sistema gestor de objetos Neodatis.

[Texto Consultas con el gestor Neodatis.](#)

4.3.- Actualización de objetos simples.

Recuerda que un **objeto simple** es un objeto que no contiene a otros objetos, como por ejemplo los objetos de la clase ponente. Un segmento de la definición de esta clase, es la siguiente:



Para **consultar objetos simples** se pueden utilizar cualquiera de los tres sistemas de consulta proporcionados por db4o, tal y como has podido ver en el apartado anterior.

Para **modificar objetos** almacenados debes seguir los siguientes pasos:

- ✓ Cambiar los valores del objeto con los nuevos valores.
- ✓ Almacenar de nuevo el objeto con el método `store()` de la interfaz `ObjectContainer`.

Por ejemplo, el siguiente método permitirá modificar objetos ponente, en concreto actualizar el e-mail de ponentes por nif de ponente:

Db4o necesita conocer previamente un objeto para poder actualizarlo. Esto significa que para poder ser actualizados los objetos, éstos deben de haber sido insertados o recuperados en la misma sesión; en otro caso se añadirá otro objeto en vez de actualizarse.

Para eliminar objetos almacenados utilizaremos el método `delete()` de la interface `ObjectContainer`.

Por ejemplo, el siguiente método elimina un objeto ponente por su nif:

Desde el siguiente enlace puedes descargar el proyecto completo. Comprueba los resultados de su ejecución.

[Código del ejemplo de actualización de objetos simples. \(18,1 KB\)](#)

Autoevaluación

Señala si la siguiente afirmación es verdadera o falsa.

Para actualizar un objeto almacenado, db4o necesita conocerlo previamente.

- Verdadero Falso

Verdadero

Es verdadero, pues si no es un objeto insertado o recuperado en la misma sesión, db4o creará un nuevo objeto.

4.4.- Actualización de objetos estructurados.

Los **objetos estructurados** son objetos que contienen a su vez a otros objetos (objetos hijo u objetos miembro).

En el caso de objetos estructurados se habla de diferentes **niveles de profundidad del objeto**. El nivel más alto, nivel 1, será el que corresponde a la definición del objeto estructurado (objeto padre), el siguiente nivel, nivel 2, corresponderá a la definición del objeto hijo y así sucesivamente podrá haber un nivel 3, 4... dependiendo de que los objetos hijos a su vez incluyan en su definición a otro u otros objetos miembro.

En el siguiente ejemplo, definimos la clase charla (objeto estructurado padre) que incorpora a un objeto ponente (objeto miembro). El nivel más alto de profundidad o nivel 1 es el que corresponde a la definición de charla y el nivel 2 corresponderá a la definición del objeto ponente.

¿Cómo se almacenan, consultan y actualizan los objetos estructurados en Db4o?

- ✓ Los objetos estructurados se almacenan asignando valores con set() y después persistiendo el objeto con store(). Al almacenar un objeto estructurado del nivel más alto, se almacenarán de forma implícita todos los objetos hijo.
- ✓ Las consultas se realizan por cualquiera de los sistemas soportados por el gestor y se podrá ir descendiendo por los diferentes niveles de profundidad.
- ✓ La eliminación o borrado de un objeto estructurado se realiza mediante el método delete(). Por defecto, no se eliminarán los objetos miembro. Para eliminar objetos estructurados en cascada o de forma recursiva, eliminando los objetos miembro, habrá que configurar de modo apropiado la base de objetos antes de abrirla, mediante el paquete com.db4o.config. En el caso de modo embebido, se hará mediante la interface EmbeddedConfiguration. En la nueva configuración se debe indicar cascadeOnDelete(true).
- ✓ La modificación se realizará actualizando los nuevos valores mediante el método set(). Por defecto las modificaciones solo afectan al nivel más alto. Para actualizar de forma recursiva todos los objeto miembro habrá que indicar en la configuración cascadeOnUpdate(true).

Desde el siguiente enlace puedes descargar el proyecto completo que realiza diferentes consultas y actualización de objetos estructurados.

[Código con ejemplo de actualización y consultas de objetos estructurados.](#) (23 KB)

En este otro enlace dispones de un ejemplo con eliminación y modificación de charlas en cascada.

[Código con ejemplo de eliminación de objetos estructurados en cascada.](#) (20,5 KB)

Citas para pensar

"No basta saber, se debe también aplicar. No es suficiente querer, se debe también hacer".

Johann W. Goethe

Autoevaluación

Señala si la siguiente afirmación es verdadera o falsa.

La actualización de un objeto estructurado supone la actualización de todos sus objetos hijo.

- Verdadero Falso

Falso

Es falso, para actualizar los objetos hijo al actualizar el objeto padre habrá que indicarlo expresamente, por ejemplo en db4o sería mediante cascadeOnUpdate(true).

5.- El lenguaje de consulta de objetos OQL.

Caso práctico

Esta tarde, **Ana** ha decidido darse un respiro y dar una vuelta por el parque. Necesita descansar. Esta última semana ha sido muy intensa, —que si pruebo esta funcionalidad de Db4o, que si pruebo esta otra, ¿qué pasa con las eliminaciones en cascada? ¿Por qué no me funciona este ejemplo? —y además, ha tenido que repasar el tipo colección de Java, que lo tenía algo olvidado.

Realmente está muy entusiasmada y así se lo ha dicho a sus compañeros de clase. Mañana, empieza a repasar con **Juan** las posibilidades del lenguaje de consultas OQL. Eso ya no le preocupa tanto, **Ana** sabe que es muy parecido al SQL, y en este campo se defiende muy bien. De hecho, esta última semana, ha tenido que programar diferentes consultas para una aplicación de bases de datos de otro cliente de la empresa BK.

OQL (Object Query Language) es el lenguaje de consulta de objetos propuesto en el estándar ODMG.

Las siguientes son algunas de las **características más relevantes de OQL**:

- ✓ Es un lenguaje declarativo del tipo de SQL que permite realizar consultas de modo eficiente sobre bases de datos orientadas a objetos.
- ✓ Su **sintaxis es similar a la de SQL**, proporcionando un superconjunto de la sintaxis de la sentencia SELECT, con algunas características añadidas para los conceptos ODMG, como la identidad del objeto, los objetos complejos, las operaciones, la herencia, el polimorfismo y las relaciones.
- ✓ **No posee primitivas para modificar el estado de los objetos** ya que las modificaciones se pueden realizar mediante los métodos que estos poseen.
- ✓ Puede ser usado como un **lenguaje autónomo o incrustado** dentro de otros lenguajes como C++, Smalltalk y Java.
- ✓ Una **consulta OQL incrustada** en uno de estos lenguajes de programación puede devolver objetos que coincidan con el sistema de tipos de ese lenguaje.
- ✓ Desde OQL se pueden invocar operaciones escritas en estos lenguajes.
- ✓ Permite **acceso tanto asociativo como navegacional**:
 - ◆ Una **consulta asociativa** devuelve una colección de objetos.
 - ◆ Una **consulta navegacional** accede a objetos individuales y las interrelaciones entre objetos sirven para navegar entre objetos.

Autoevaluación

Señala si la siguiente afirmación es verdadera o falsa.

El lenguaje OQL incluye un rico repertorio de sentencias para modificar el estado de los objetos.

Verdadero Falso

Falso

Es falso, ya que lo normal es utilizar para ello los propios métodos del objeto.

5.1.- Sintaxis, expresiones y operadores.

La sintaxis básica y resumida de una sentencia SELECT del OQL estándar es la siguiente:

```
SELECT [DISTINCT] <expresión, ...>
FROM <lista from>
[WHERE <condición> ]
[ORDER BY <expresión>]
```

Por ejemplo, suponiendo el esquema de base de objetos que puedes ver en la figura ampliable de la derecha, la siguiente sentencia select recupera de la base de objetos los atributos nombre y el correo de objetos tipo profesor cuyo año de ingreso es anterior al 1990, y ordenados alfabéticamente por nombre:

```
SELECT p.nombre, p.email FROM p in Profesor WHERE p.ingreso <= 1990 ORDER BY p.nombre;
```

Las siguientes, son algunas **consideraciones a tener en cuenta**:

- ✓ En las consultas se necesita un punto de entrada, que suele ser el nombre de una clase.
- ✓ El resultado de una consulta es una colección que puede ser tipo bag (si hay valores repetidos) o tipo set (no hay valores repetidos). En este último caso habrá que especificar SELECT DISTINCT.
- ✓ En general, una consulta OQL puede devolver un resultado con una estructura compleja especificada en la misma consulta utilizando struct.
- ✓ Una vez que se establece un punto de entrada, se pueden utilizar expresiones de caminos para especificar un camino a atributos y objetos relacionados. Una expresión de camino empieza normalmente con un nombre de objeto persistente o una variable iterador, seguida de ninguno o varios nombres de relaciones o de atributos conectados mediante un punto.
- ✓ Es posible crear objetos mutables (no literales) formados por el resultado de una consulta.

Además, en una consulta OQL se pueden utilizar, entre otros, los siguientes **operadores y expresiones**:

- ✓ Operadores de acceso: "." / "->" aplicados a un atributo, una expresión o una relación.
FIRST / LAST (primero / último elemento de una lista o un vector).
- ✓ Operadores aritméticos: +, -, *, /, -(unario), MOD, ABS para formar expresiones aritméticas.
- ✓ Operadores relacionales: >, <, >=, <=, <>, = que permiten comparaciones y construir expresiones lógicas.
- ✓ Operadores lógicos: NOT, AND, OR que permiten enlazar otras expresiones.

A continuación te indicamos de manera resumida, **algunas otras características**, del estándar OQL:

- ✓ Definición de vistas, es decir, es posible dar nombre a una consulta y utilizarlo en otras consultas.
- ✓ Extracción de elementos sencillos de colecciones set, bag, o list.
- ✓ Operadores de colecciones como funciones de agregaciones (MAX(), MIN(), COUNT(), SUM() y AVG()) y cuantificadores (FOR ALL, EXISTS).
- ✓ Realización de agrupaciones mediante GROUP BY y filtro de los grupos mediante HAVING.
- ✓ Combinación de consultas mediante JOINs
- ✓ Unión, intersección y resta de colecciones mediante los operadores UNION, INTERSEC y EXCEPT.

En los siguientes apartados nos centraremos en el OQL concreto de un SGBDOO y que por supuesto, incorpora sus propias particularidades y diferencias respecto a la propuesta OQL de ODMG.

Para saber más

En el siguiente enlace puedes consultar una extensa guía de referencia y con ejemplos, en inglés, del lenguaje OQL propuesto por el ODMG.

[Guía de referencia del lenguaje OQL.](#)

5.2.- Matisse, un gestor de objetos que incorpora OQL.

Para practicar y trabajar con OQL utilizaremos Matisse, un gestor orientado a objetos que incorpora características del estándar ODMG, como los lenguajes ODL y OQL, y que tiene soporte para Java. Aunque Matisse llama SQL a su lenguaje de consultas, nosotros nos referiremos a él como OQL.



El propio gestor proporciona el driver matisse.jar para interactuar con aplicaciones escritas en Java.

Dentro de los **paquetes del API** destacamos:

- ✓ com.matisse. Proporciona las clases e interfaces básicos para trabajar con Java y una base de datos de objetos Matisse.
- ✓ MtDatabase. Clase que proporciona todos los métodos para realizar las conexiones y transacciones en la base de objetos.
- ✓ com.matisse.sql. Proporciona clases que permiten interactuar con la base de objetos vía JDBC.

Todas las interacciones entre una aplicación Java y la base de objetos Matisse se realizan en el contexto de una transacción (implícita o explícita).

En el siguiente recurso didáctico encontrarás un tutorial con los pasos a seguir para la descarga del software Matisse y realizar su instalación e integración en NetBeans a partir de del driver matisse.jar.

Podrás descargarlo desde la tarea.

Matisse: Instalación en Windows 7



Matisse: Instalación en Windows 7



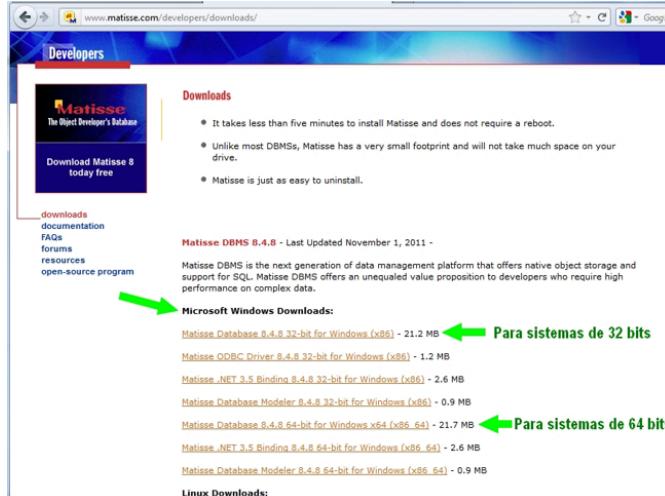
- ✓ Accedemos a la web oficial de Matisse <http://www.matisse.com>
- ✓ Pulsamos el botón **free developer's version downloads now >>**
- ✓ En la siguiente página habrá que registrarse para poder bajarse el producto.

Registro para descarga del software

- ✓ Si no estás registrado debes registrarte. En la siguiente página debes acceder como **usuario registrado para poder descargar el software**.

Descarga del archivo de instalación

- ✓ Descarga el archivo de instalación, según tu JDK sea de 32 o 64 bits.

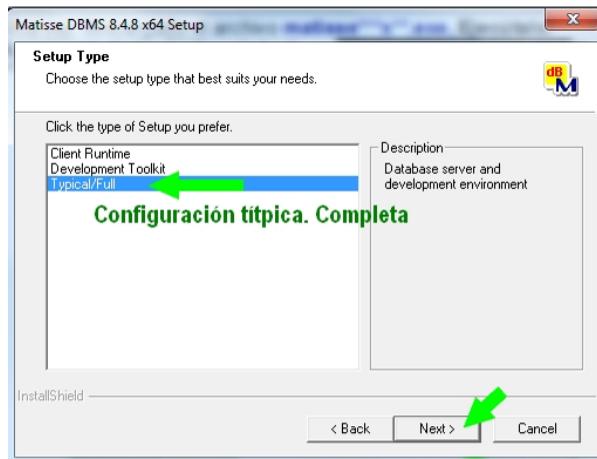


Asistente de instalación Matisse

- ✓ Una vez descargado el archivo **matisse***x**.exe**, Ejecútalo.
- ✓ Aparecerá el siguiente asistente, y bastará con pulsando en **Next**

Seleccionar configuración típica

- ✓ Después de aceptar los términos de la licencia e indicar el nombre de usuario y compañía, seleccionamos la **instalación Típica/Completa (Typical/Full)**
- ✓ Pulsamos **Next**



Directorio de instalación

- ✓ Dejamos el directorio de instalación por defecto y pulsamos **Next**.
- ✓ Aparecerá una pantalla que nos indicará que se realizará la copia de ficheros.
- ✓ Pulsaremos **Next**.

Fin de instalación

- ✓ Tras finalizar la copia de ficheros ya habrá terminado la instalación.
- ✓ Pulsamos **Finish** para finalizar.
- ✓ Es recomendable dejar marcado el **Readme de Matisse**, pues nos mostrará cómo empezar a trabajar con este sistema de bases de objetos.

Fichero Readme.html de Matisse

- ✓ Si has dejado marcado el **Readme de Matisse**, aparecerá la siguiente pantalla.
- ✓ En cualquier momento podrás acceder a esta información ejecutando el fichero **readme.html** de Matisse.

Directorio de instalación

- ✓ En el **directorio de instalación de Matisse C:\Products\Matisse**, en la carpeta **lib**, encontraremos el archivo **matisse.jar**, biblioteca necesaria para integrar Matisse con nuestro Entorno de Desarrollo NetBeans.

Integración dentro del IDE NetBeans (I)

- ✓ Creamos o Abrimos un **proyecto en el que trabajaremos con matisse**
- ✓ Nos situamos sobre **Bibliotecas** y desde su menú contextual (clic derecho)
- ✓ Seleccionamos la opción **Agregar archivo JAR/Carpeta**

Instalación dentro del IDE NetBeans (II)

- ✓ Seleccionamos del directorio lib el .jar que nos interesa, **matisse.jar**, y pulsamos **Abrir**

Instalación dentro del IDE NetBeans (III)

- ✓ También puedes copiar el fichero **matisse.jar** al raíz de tu proyecto y seleccionarlo de esa ubicación indicando una **ruta relativa**. De esta forma, si cambias de equipo, siempre tendrás disponible la librería, aunque esté en otra ubicación.

Comprobamos

- ✓ Podemos comprobar que el archivo **matisse.jar** se ha incluido en las **Bibliotecas del proyecto**.

- ✓ Ya **podemos utilizar** en nuestro proyecto:
 - ↳ una base de objetos Matisse
 - ↳ toda su Interfaz de Programación para Aplicaciones (API).

Credenciales

Imagen	Datos licencia
Todas las capturas de pantalla de esta presentación	Autoría:Matisse Software Inc. Tipo de licencia: Copyright (Cita). Procedencia: Instalación del gestor de objetos Matisse.

1 2 3 4 5 6 7 8 9 10 11 12 13 14
[Resumen textual alternativo](#)

En este otro recurso didáctico verás cómo crear la base de objetos para practicar con OQL.

Matisse: Creación de la BDOO Doctorado

Doctorado será una **base de objetos** que permitirá almacenar información sobre profesores, su departamento y las tesis que dirigen.

- ✓ Un profesor puede dirigir varias tesis y pertenece a un solo departamento.
- ✓ Una tesis es dirigida por un único profesor.
- ✓ Un departamento está formado por varios profesores.

Las clases necesarias son las siguientes: **Profesor, Tesis y Departamento**.

Matisse: Creación de la BDOO Doctorado

Lo que haremos

Seguiremos los siguientes pasos:

- ✓ En Matisse, crearemos una nueva **base de objetos** de nombre **doctorado**.
- ✓ En Matisse, **importaremos el esquema ODL** (fichero doctorado.odl) con las clases Profesor, Tesis y Departamento a la base de objetos doctorado. Este fichero te lo descargarás junto al proyecto y lo ubicarás en una carpeta de tu equipo.
- ✓ En NetBeans, crearemos el **proyecto DoctoradoMatisse**.
- ✓ Agregaremos el **driver de matisse**, matisse.jar, a las bibliotecas del proyecto.
- ✓ En el método main() indicaremos la **conexión a la base de objetos** doctorado.
- ✓ Desde Matisse **generaremos el código java que implementa las clases** Profesor, Tesis y Departamento en nuestro proyecto.
- ✓ En el método main() incluiremos llamadas a los métodos que permiten **insertar objetos en la base de objetos** y almacenarlos al ejecutar la aplicación.

Desde Matisse crear base de objetos

- ✓ Desde **Inicio/Programas** seleccionamos **Matisse/Enterprise Manager**.
- ✓ Nos situamos sobre el servidor en marcha (será nuestro equipo).
- ✓ Pulsamos el botón derecho del ratón y seleccionamos **New Database**.

Damos nombre a la base de objetos

- ✓ Introducimos el nombre de la base de objetos **doctorado**.
- ✓ Pulsamos el **botón OK**.

Iniciar la base de objetos en el servidor

- ✓ Desde el Enterprise Manager nos situamos sobre la base de objeto recién creada, doctorado.
- ✓ Pulsamos el **botón derecho** del ratón y seleccionamos **start**. Esto iniciará o pondrá en línea la base de objetos doctorado.

Importar esquema ODL

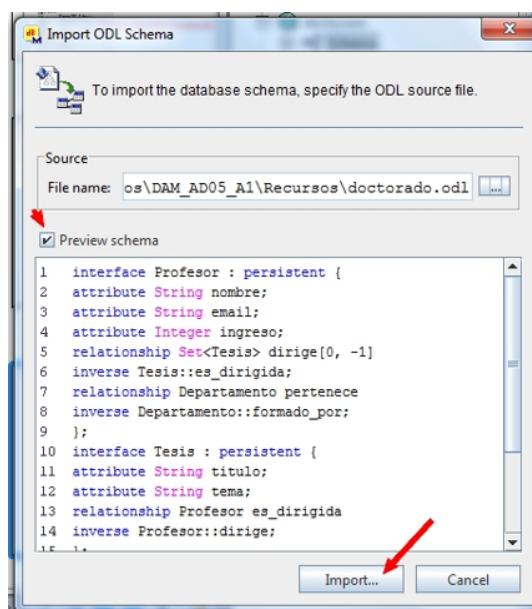
- ✓ Nos situamos sobre la base de objetos **doctorado** y hacemos **doble clic**.
- ✓ Nos situamos sobre **Schema**.
- ✓ Pulsamos el **botón derecho** del ratón y seleccionamos **Import ODL Schema**.

Importando esquema ODL, archivo .odl

- ✓ Buscamos el fichero **doctorado.odl** en nuestro equipo y pulsamos **Open**.

Confirmar importación esquema ODL

- ✓ En la siguiente pantalla, pulsamos **Import...** para confirmar la importación.



Importación ODL realizada

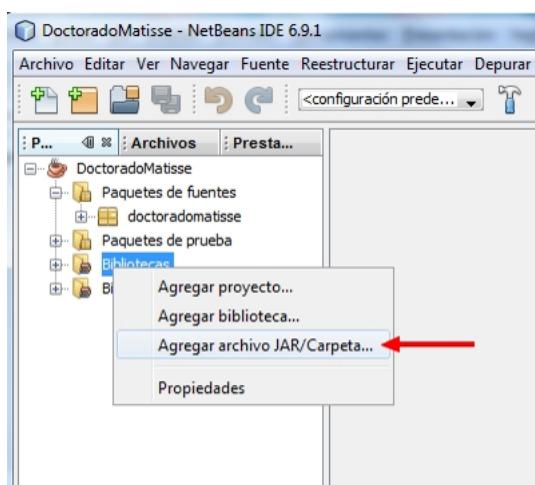
- ✓ La siguiente pantalla nos informa del estado de la importación. Pulsamos **Done**.

Navegando por las clases

- ✓ Desde Matisse (Enterprise Manager) podemos navegar por las clases haciendo doble clic sobre cada una de ellas y observar los atributos e interrelaciones.

Crear el proyecto y agregar API matisse

- ✓ En NetBeans creamos el **proyecto de nombre DoctoradoMatisse**.
- ✓ Nos situamos sobre el proyecto, en el nodo **Bibliotecas**.
- ✓ Hacemos un clic derecho sobre Bibliotecas.
- ✓ Ejecutamos la opción **'Agregar archivo JAR/Carpeta...**



Agregando API de Matisse

- ✓ Seleccionamos el JAR de nuestra carpeta local, archivo **matisse.jar**
- ✓ Si **matisse.jar** lo hemos ubicado en el raíz de nuestro proyecto, seleccionamos Ruta relativa.
- ✓ Pulsamos el botón **Abrir**.

Indicar cadena de conexión en main()

- ✓ En nuestra clase principal, la de main(), importamos la biblioteca de matisse **com.matisse.MtDatabase**
- ✓ En el método main() creamos un objeto **MtBasedata** indicando la **cadena de conexión** ("nombrehost", "nombrebbaseobjetos")

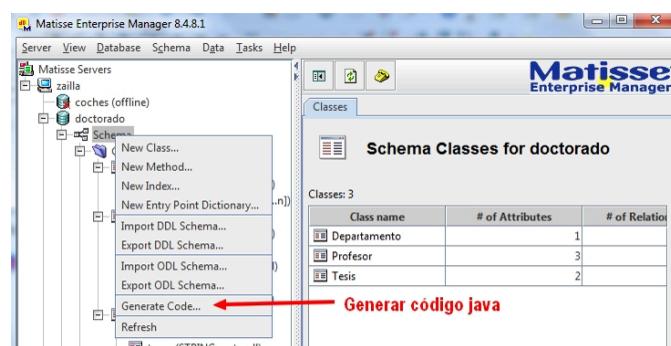
```

7
8 //API de matisse
9 import com.matisse.MtDatabase; ←
10
11 public class Main {
12
13     public static void main(String[] args) {
14         //crea el objeto base de datos MtDatabase indicando la cadena de conexión
15         //nombre del host "zailla" y base de datos "doctorado"
16         //no se necesita usuario porque no se ha definido un control de acceso
17         MtDatabase db = new MtDatabase("zailla", "doctorado");
18     }
19 }
20
21 }
```

Cadena de conexión
o bien:
Cadena conexión (localhost, "doctorado")

Desde Matisse generamos código java

- ✓ En **Matisse**, nos situamos sobre **Schema** de la base de objetos **doctorado** y en su menú contextual seleccionamos **Generate Code...**



Indicamos carpeta destino y lenguaje java

- ✓ Buscamos el directorio de nuestro proyecto DoctoradoMatisse, carpeta **src\doctoradomatisse**
- ✓ Indicamos que el lenguaje es **Java**
- ✓ Indicamos el nombre del **package** que es **doctoradomatisse**
- ✓ Una vez generado el código, pulsamos **Done**

Código generado en proyecto DoctoradoMatisse

- ✓ Desde NetBeans, podemos ver que se han generado las clases Profesor, Tesis y Departamento.

Operaciones desde main()

- ✓ En el método **main()**, tras la conexión a la base de objetos, en un **bloque try-catch**:
- ✓ Abrimos conexión db.open()
- ✓ Invocamos al método para insertar objetos insertarObjetos()
- ✓ Cerramos conexión db.close()

Creación de objetos en clase Main

- ✓ En la clase Main, creamos el método **insertarObjetos()**, donde crearemos diversos objetos tipo Profesor, Tesis y Departamento.
- ✓ Relacionaremos los objetos mediante los **métodos asociados a las interrelaciones establecidas**.
- ✓ Todos estos métodos se han generado en la generación de código.
- ✓ **EJECUTAMOS** y los objects se almacenarán en la base de objetos.

```

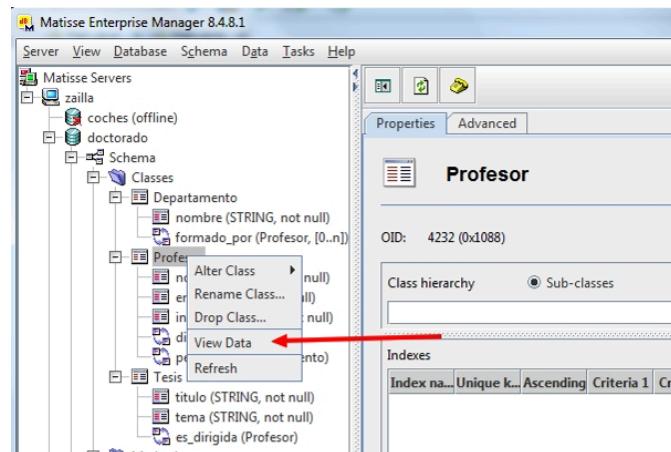
private static void insertarObjetos(MtDatabase db) {
    //crea objetos Departamento
    Departamento d1 = new Departamento(db);
    d1.setNombre("Bases de Datos");
    Departamento d2 = new Departamento(db);
    d2.setNombre("Lenguajes");
    // Crea objetos Tesis
    Tesis t1 = new Tesis(db);
    t1.setTitulo("Persistencia de objetos");
    t1.setTema("Bases de Objetos");
    Tesis t2 = new Tesis(db);
    t2.setTitulo("Bases de Datos Nativas XML");
    t2.setTema("Bases de Datos XML");
    Tesis t3 = new Tesis(db);
    t3.setTitulo("Mapeo Objeto-Relacional");
    t3.setTema("Bases de Datos");
    Tesis t4 = new Tesis(db);
    t4.setTitulo("Multiproceso en Java");
    t4.setTema("Lenguajes de Programación");
}

Profesor p1 = new Profesor(db);
p1.setNombre("Ana Martos Gil");
p1.setEmail("ana.martos@universi.es");
p1.setIngreso(1990);
Profesor p2 = new Profesor(db);
p2.setNombre("Isabel Ruiz Granados");
p2.setEmail("isabel.ruz@universi.es");
p2.setIngreso(1986);
Profesor p3 = new Profesor(db);
p3.setNombre("Antonio Barcia Navarro");
p3.setEmail("antonio.barea@universi.es");
p3.setIngreso(1995);
//establece relaciones entre Profesor y Depto
//Al establecer una relación no hace falta
p1.setPertenece(d1);
p1.appendDirige(t1);
p2.setPertenece(d1);
p2.appendDirige(t2);
p2.appendDirige(t3);

```

Visualizando objetos en Matisse

- ✓ Desde Matisse, nos situamos en una de las clases, por ejemplo **Profesor** y en su menú contextual seleccionamos **View Data**.



Ejecutando consulta en Matisse

- ✓ En la siguiente pantalla, ejecutamos la consulta mediante **F5** o bien botón Execute Query.

Visualizando consulta en Matisse

- ✓ En la siguiente pantalla se visualiza el resultado de la consulta

Credenciales

Imagen	Datos licencia
Imagen de la diapositiva 2	Autoría: Isabel M. Cruz Granados Licencia: Uso educativo-no comercial. Procedencia: Dibujo realizado por la autora.
Capturas de pantalla de esta presentación EXCEPTO diapositivas , 2, 14, 18, 19 y 20.	Autoría:Matisse Software Inc. Tipo de licencia: Copyright (Cita). Procedencia: Instalación del gestor de objetos Matisse.
Capturas de pantalla de esta de las diapositivas 12, 18 de esta presentación	Autoría: Isabel M. Cruz Granados Licencia: Uso educativo-no comercial. Procedencia: Captura de pantalla del programa NetBeans, propiedad Sun Microsystems, bajo licencia GNU GPL v2
Capturas de pantalla de esta de las diapositivas 14,19, 20 de esta presentación	Autoría: Isabel M. Cruz Granados Licencia: Uso educativo-no comercial. Procedencia: Captura de pantalla del editor de código NetBeans, propiedad Sun Microsystems, bajo licencia GNU GPL v2

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22

[Resumen textual alternativo](#)

Desde el siguiente enlace puedes descargar el proyecto completo de creación de una base de objetos en Matisse, así como el esquema ODL de dicha base de objetos.

[Código con ejemplo de creación de base de objetos en Matisse. \(0.17 MB\)](#)

Para saber más

Desde el siguiente enlace podrás descargar numerosos manuales y abundante documentación sobre Matisse.

[Documentación, manuales y guías de Matisse.](#)

5.3.- Ejecución de Sentencias OQL.

La sintaxis básica del OQL de Matisse es una sentencia SELECT de la forma: SELECT... FROM... WHERE...;

Por ejemplo, para recuperar el valor de todos los atributos de los objetos tipo Profesor, escribiríamos la siguiente sentencia OQL:

```
SELECT * FROM Profesor;
```

Y para recuperar del atributo nombre de los objetos tipo Profesor cuyo año de ingreso es anterior al 1990, escribiríamos la siguiente sentencia

```
SELECT nombre FROM Profesor WHERE ingreso <= 1990;
```

Las siguientes son algunas consideraciones y ejemplos sobre las consultas con SELECT:

- ✓ Toda sentencia SELECT finaliza en punto y coma.
- ✓ Además de la cláusula WHERE, que permite establecer un criterio de selección se pueden utilizar las cláusulas de agrupamiento GROUP BY, y de ordenación ORDER BY, entre otras. También se pueden asignar alias mediante AS, realizar búsquedas por patrones de caracteres con LIKE.

Ejemplo: título y tema de las tesis cuyo tema contiene la palabra Objeto, ordenadas por tema.

```
SELECT titulo AS "Tesis", tema FROM Tesis
WHERE tema LIKE '%Objeto%' ORDER BY tema;
```

- ✓ Al recuperar todos los atributos de una clase mediante SELECT *, la consulta retornará el OID de cada objeto recuperado, así como las interrelaciones o relaciones definidas para esa clase. El OID y la interrelación son del tipo string y se representan mediante un número hexadecimal. Realmente el identificador recuperado para la interrelación, hace referencia al primer objeto relacionado, incluso aunque la interrelación incluya a más de un objeto.
- ✓ Se pueden hacer JOIN de clases, y por ejemplo esto puede permitir obtener todos los objetos relacionados con otro objeto en una consulta asociativa.

Ejemplo: nombre de cada profesor y título y tema de las tesis que dirige con JOIN

```
SELECT p.nombre, t.titulo, t.tema
FROM Profesor p JOIN Tesis t ON t.es_dirigida = p.OID;
```

- ✓ Mediante SELECT se pueden realizar consultas navegacionales haciendo referencia a las interrelaciones entre objetos.

Ejemplo: nombre de cada profesor y título y tema de las tesis que dirige, navegacional

```
SELECT t.titulo AS "Tesis", tema, t.es_dirigida.nombre
AS "Profesor " FROM Tesis t;
```

- ✓ También se pueden realizar consultas navegacionales a través de la referencia de los objetos (REF()). **Ejemplo:** tesis que dirigen los profesores con ingreso el 1990 o posterior

```
SELECT REF(p.dirige) FROM Profesor p
1) WHERE p.ingreso >= 1990;
```

En el siguiente recurso didáctico encontrarás diferentes ejemplos de consultas utilizando de forma autónoma o conversacional el lenguaje OQL de Matisse.

Matisse: Ejecución de consultas OQL

Doctorado es una **base de objetos** que almacena objetos tipo Profesor, Departamento y Tesis.

- ✓ Un profesor puede dirigir varias tesis y pertenece a un solo departamento.
- ✓ Una tesis es dirigida por un único profesor.
- ✓ Un departamento está formado por varios profesores.

Iniciar la base de objetos en el servidor

- ✓ Desde el Enterprise Manager de Matisse nos situamos sobre la base de objetos **doctorado**.
- ✓ Pulsamos el **botón derecho** del ratón y seleccionamos **start**. Esto iniciará o pondrá en línea la base de objetos doctorado.

Acceder a SQL Query Analyzer

- ✓ Desde Matisse, nos situamos en **Data** y hacemos doble clic.
- ✓ Seleccionamos **SQL Query Analyzer**

Ejecutando consultas en Matisse

- ✓ En la siguiente pantalla, **redactamos una consulta OQL**
- ✓ **Ejecutamos** la consulta mediante **F5** o bien botón **Execute Query**.
- ✓ **Consulta 1:** nombre y email de los profesores.

↳ SELECT nombre, email FROM Profesor;

Consulta 2

- ✓ **Consulta 2:** nombre e ingreso de los profesores cuyo ingreso es 1990 o anterior

```
↳ SELECT nombre FROM Profesor WHERE ingreso<=1990;
```

Consulta 3

- ✓ **Consulta 3:** Título y tema de las Tesis cuyo tema incluya la palabra Objeto.

```
↳ SELECT titulo AS "Tesis", tema FROM Tesis  
↳ WHERE tema LIKE '%Objeto%';
```

Consulta 4

- ✓ **Consulta 4:** Todos los datos de los profesores.

```
↳ SELECT * FROM Profesor;
```

Consulta 5 con JOIN

- ✓ **Consulta 5:** Título y tema de las tesis que dirige cada profesor.

```
↳ SELECT p.nombre, t.titulo, t.tema FROM Profesor p  
    JOIN Tesis t ON t.es_dirigida = p_OID;
```

Consulta 5 en modo Navegacional

- ✓ **Consulta 5 b:** Título y tema de las tesis que dirige cada profesor.

```
↳ SELECT t.titulo AS "Tesis", tema, t.es_dirigida.nombre AS "Profesor"  
    FROM Tesis t;
```

Consulta 6 Navegacional mediante REF()

✓ Consulta 6: Tesis que dirigen los profesores con ingreso el 1990 o posterior.

↳ SELECT REF(p.dirige) FROM Profesor p WHERE p.ingreso >= 1990;

Credenciales

Imagen	Datos licencia
Todas las capturas de pantalla de esta presentación, salvo la diapositiva 2	Autoría: Matisse Software Inc. Tipo de licencia: Copyright (Cita). Procedencia: Realización de consultas OQL
Diapositiva 2	Autoría: Isabel M. Cruz Granados Licencia: Uso educativo-no comercial. Procedencia: Dibujo realizado por la autora.

1 2 3 4 5 6 7 8 9 10 11

[Resumen textual alternativo](#)

Autoevaluación

Señala las opciones correctas. Las consultas OQL con SELECT:

- No permiten recuperar objetos relacionados.

- No permiten realizar consultas navegacionales.

- Permiten realizar JOINs.

- Permiten recuperar el OID de un objeto almacenado.

[Mostrar retroalimentación](#)

Solución

1. Incorrecto
2. Incorrecto
3. Correcto
4. Correcto

Para saber más

En el siguiente enlace dispones de la guía completa, en inglés, sobre el OQL de Matisse:

[Guía oficial del OQL de Matisse.](#) (1.61 MB)

5.4.- Ejecución de sentencias OQL vía JDBC.

Veremos ahora cómo realizar consultas a una base de objetos Matisse mediante sentencias OQL embebidas en Java. Para ello, la conexión a la base de objetos se realizará vía JDBC. Matisse proporciona dos formas de manipular objetos vía JDBC: mediante JDBC puro o mediante una mezcla de JDBC y Matisse.

1. Para crear una conexión vía JDBC puro utilizaremos `java.SQL.*` y `com.lang="de">Matisse`.`.SQL`.`MtDriver`, no siendo necesario en este caso el paquete `com.lang="de">Matisse`.
 - ✓ La cadena de conexión será de la forma: `String url = "JDBC:mt://" + hostname + "/" + dbname;`
 - ✓ La conexión se realizará mediante `Connection jcon = DriverManager.getConnection(url);`
2. Para crear una conexión vía JDBC y Matisse (a través de `MtDatabase`) se necesita `java.SQL.*` y `com.Matisse.MtDatabase`.
 - ✓ La cadena de conexión será de la forma: `MtDatabase db = new MtDatabase(hostname, dbname);`
 - ✓ La conexión se realizará mediante: `db.open();` y `Connection jcon = db.getJDBCConnection();`

Una vez realizada la conexión por uno u otro método, ya podremos ejecutar sentencias OQL vía JDBC, (tal y como si de una sentencia SQL se tratara) en la aplicación java. El esquema de consulta es el siguiente:

Desde el siguiente enlace puedes descargar el proyecto completo de consultas OQL vía JDBC.

[Código con ejemplo de consultas OQL vía JDBC en Matisse](#) (0,19 MB)

Citas para pensar

"Vale más saber alguna cosa de todo, que saberlo todo de una sola cosa".

Blaise Pascal

Autoevaluación

Señala la opción correcta. Para crear una conexión vía JDBC puro en Matisse:

- No es posible, siempre hay que utilizar una mezcla entre JDBC y Matisse.
- Además de `java.sql.*` se necesita `com.matisse.sql.MtDriver`.
- Se necesita `java.sql.*` y `com.matisse.MtDatabase`
- No se pueden realizar este tipo de conexiones en Matisse

No es correcto, prueba de nuevo.

Muy bien, vamos por buen camino.

Incorrecto, esto es justo para la mezcla Matisse y JDBC.

No es la respuesta correcta, sí que se pueden realizar.

Solución

1. Incorrecto
2. Opción correcta
3. Incorrecto
4. Incorrecto

Para saber más

En el siguiente enlace puedes consultar la Guía Matisse para programadores Java y ver diferentes ejemplos de consultas OQL y en modo nativo.

[Guía oficial de Matisse para programadores Java.](#) (0,64 MB)

6.- Características de las bases de datos objeto-relacionales.

Caso práctico

Hoy **Antonio** va a estar todo el día con **Juan** y **Ana**. Va a aprovechar que hoy comienzan a repasar las posibilidades de las bases de datos objeto-relacionales y como al fin y al cabo son bases de datos relacionales, quiere hacerles unas preguntillas sobre unas dudas que tiene sobre SQL, y de paso empaparse de aquellas características que tienen que ver con la orientación a objetos.



Las BDOR las podemos ver como un híbrido de las BDR y las BDOO que intenta aunar los beneficios de ambos modelos, aunque por descontado, ello suponga renunciar a algunas características de ambos.

Los objetivos que persiguen estas bases de datos son:

- ✓ Mejorar la representación de los datos mediante la orientación a objetos.
- ✓ Simplificar el acceso a datos, manteniendo el sistema relacional.

En una BDOR se siguen almacenando tablas en filas y columnas, aunque la estructura de las filas no está restringida a contener escalares o valores atómicos, sino que las columnas pueden almacenar tipos estructurados (tipos compuestos como vectores, conjuntos, etc.) y las tablas pueden ser definidas en función de otras, que es lo que se denomina herencia directa.

Y eso, ¿cómo es posible?

Pues porque internamente tanto las tablas como las columnas son tratados como objetos, esto es, se realiza un mapeo objeto-relacional de manera transparente.

Como consecuencia de esto, aparecen **nuevas características**, entre las que podemos destacar las siguientes:

- ✓ **Tipos definidos por el usuario.** Se pueden crear nuevos tipos de datos definidos por el usuario, y que son compuestos o estructurados, esto es, será posible tener en una columna un atributo multivaluado (un tipo compuesto).
- ✓ **Tipos Objeto.** Posibilidad de creación de objetos como nuevo tipo de dato que permiten relaciones anidadas.
- ✓ **Reusabilidad.** Posibilidad de guardar esos tipos en el gestor de la BDOR, para reutilizarlos en tantas tablas como sea necesario
- ✓ **Creación de funciones.** Posibilidad de definir funciones y almacenarlas en el gestor. Las funciones pueden modelar el comportamiento de un tipo objeto, en este caso se llaman métodos.
- ✓ **Tablas anidadas.** Se pueden definir columnas como arrays o vectores multidimensionales, tanto de tipos básicos como de tipos estructurados, esto es, se pueden anidar tablas
- ✓ **Herencia** con subtipos y subtablas.

Estas y otras características de las bases de datos objeto-relacionales vienen recogidas en el estándar SQL 1999 .

Autoevaluación

Señala si la siguiente afirmación es verdadera o falsa.

En una base de datos objeto-relacional, las columnas de una tabla pueden almacenar tipos estructurados.

- Verdadero Falso

Verdadero

Es verdadero, esa es una de las nuevas características de este tipo de bases de datos.

6.1.- El estándar SQL99.

La norma ANSI SQL1999 (abreviadamente, SQL99) extiende el estándar SQL92 de las Bases de Datos Relacionales, y da cabida a nuevas características orientadas a objetos preservando los fundamentos relacionales.

Algunas extensiones que contempla este estándar y que están relacionadas directamente con la orientación a objetos son las siguientes:

✓ **Extensión de tipos de datos.**

- ◆ Nuevos tipos de datos básicos para datos de caracteres de gran tamaño, y datos binarios de gran tamaño (Large Objects)
- ◆ Tipos definidos por el usuario o tipos estructurados.
- ◆ Tipos colección, como los arrays, **set**, **bag** y **list**.
- ◆ Tipos fila y referencia

✓ **Extensión de la semántica de datos.**

- ◆ Procedimientos y funciones definidos por el usuario, y almacenados en el gestor.
- ◆ Un tipo estructurado puede tener métodos definidos sobre él.

Por ejemplo, el siguiente segmento de SQL crea un nuevo tipo de dato, un tipo estructurado de nombre profesor y que incluye en su definición un método, el método sueldo().

```
CREATE TYPE profesor AS (
    id INTEGER,
    nombre VARCHAR (20),
    sueldo_base DECIMAL (9,2),
    complementos DECIMAL (9,2),
    INSTANTIABLE NOT FINAL
    METHOD sueldo() RETURNS DECIMAL (9,2));
    CREATE METHOD sueldo() FOR profesor
    BEGIN
        .....
    END;
```

Citas para pensar

"Lo que embellece al desierto es que en alguna parte esconde un pozo de agua".

Antoine de Saint-Exupery

7.- Gestores de Bases de Datos Objeto-Relacionales.

Caso práctico

Ada ha reunido a **Juan** y **Ana** esta mañana, para hablar sobre la marcha del proyecto, y decirles que **María** empezará esta semana a trabajar algunos ratos con ellos. **Ada** les dice —El cliente de la aplicación de mobiliario de diseño nos ha pedido que le diseñemos también un sitio web para su negocio, de manera que **María** se encargará de ese tema —y pregunta—, ¿os habéis decidido ya por algún sistema de bases de datos en particular?

A lo que **Juan** responde. —Esta semana vamos a valorar algunos sistemas objeto-relacionales, aunque casi estamos seguros de que para esta aplicación lo mejor es una base de objetos.

Ada sonríe y dice —Bueno, aplíquenos en el tema y cuando tengáis la decisión tomada, me lo comunicáis.



Podemos decir que un sistema gestor de bases de datos objeto-relacional (SGBDOR) contiene dos tecnologías; la tecnología relacional y la tecnología de objetos, pero con ciertas restricciones.

A continuación te indicamos algunos ejemplos de **gestores objeto-relacionales**, tanto de código libre como propietario, todos ellos con soporte para Java:

- ✓ De código abierto:
 - ❖ PostgreSQL
 - ❖ Apache Derby
- ✓ De código propietario
 - ❖ Oracle
 - ❖ First SQL
 - ❖ DB2 de IBM

Las normativas SQL99 y SQL2003 son el estándar base que siguen los SGBDOR en la actualidad, aunque como siempre ocurre, cada gestor incorpora sus propias particularidades y diferencias respecto al estándar. Por ejemplo, en Oracle se pueden usar las dos posibilidades de herencia, de tipos y de tablas, mientras que en PostgreSQL solo se puede usar la herencia entre tablas.

7.1.- Instalación del Gestor PostgreSQL.

Para practicar con este tipo de gestores y algunas de las nuevas características que incorporan, hemos elegido PostgreSQL, considerado como el gestor de código abierto más avanzado del mundo.

El código fuente de PostgreSQL está disponible bajo la licencia [BSD](#). Esta licencia te da libertad para usar, modificar y distribuir PostgreSQL en cualquier forma, ya sea junto a código abierto o cerrado. Además PostgreSQL incluye API para diferentes lenguajes de programación, entre ellos Java y .NET

En el siguiente tutorial se describe el proceso de instalación del servidor PostgreSQL en Windows:

[PostgreSQL: Instalación en Windows 7](#)

PostgreSQL: Instalación en Windows 7

Descarga del software

- ✓ Accedemos a la web oficial de PostgreSQL <http://www.postgresql.org/>
- ✓ Desde **Downloads**, hacemos clic en el Sistema Operativo sobre el que vamos a realizar la instalación, en este caso **Windows 7**.
- ✓ Clic sobre “**Download the installer**” que nos llevará a la siguiente página

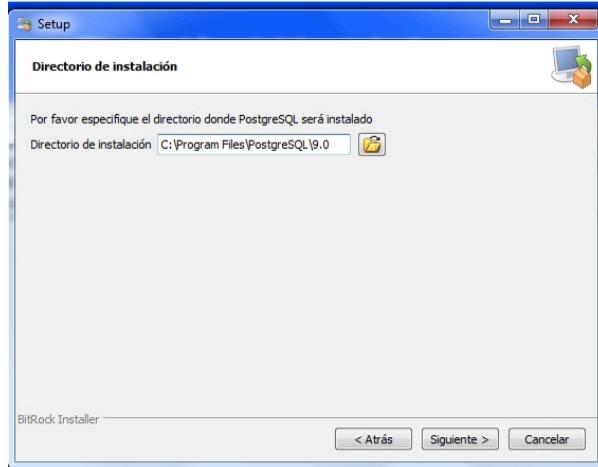
- ✓ Descargamos la última versión para Windows que corresponda: **Win x86-32** o **Win x86-64**(para Sistemas Operativos de 32 bits o 64 bits respectivamente).
- ✓ Guardamos el fichero y ejecutamos el instalador:

Ejecución del Instalador

- ✓ Al hacer **doble clic sobre el instalador** comenzará la instalación de PostgreSQL.
- ✓ La primera pantalla es la de bienvenida. Pulsamos sobre el **botón Siguiente**.

Directorio de Instalación

- ✓ Indicamos el **directorío de Instalación** y damos a siguiente. En este directorio se almacenará el sistema gestor de bases de datos.



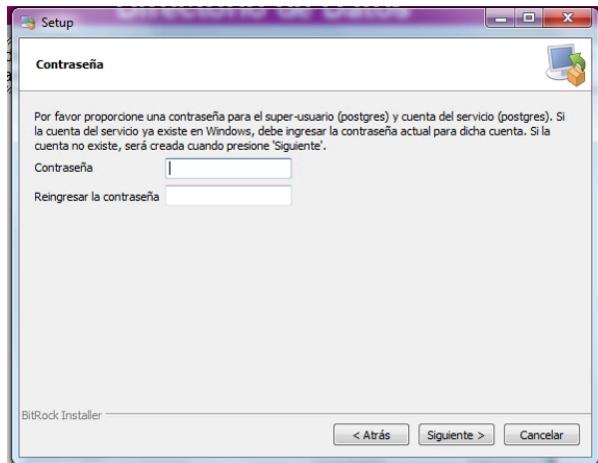
Directorio de Datos

- ✓ Indicamos el **directorío de Datos** y damos a siguiente. En este directorio se almacenarán las bases de datos.



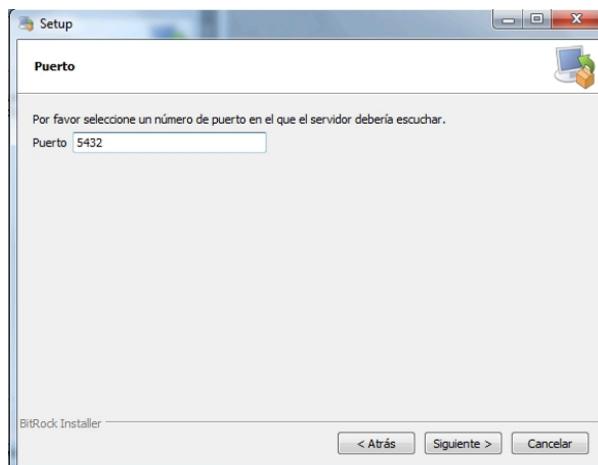
Contraseña Superusuario y Servicio

- ✓ Indicamos la **contraseña de la cuenta del superusuario y del servicio Postgres** y damos a siguiente. El superusuario es el usuario con privilegios de administración.



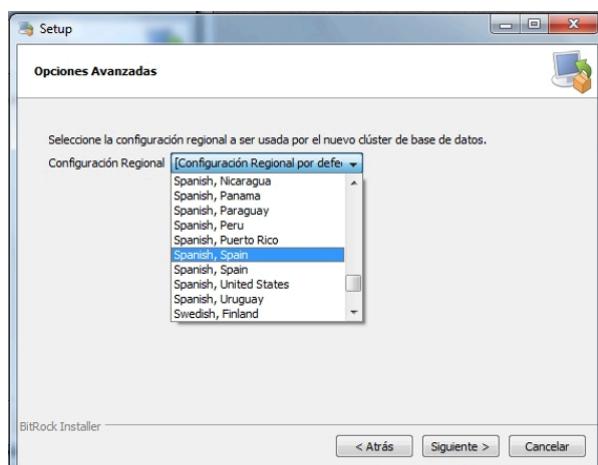
Puerto de escucha del servicio

- ✓ Indicamos el **puerto de escucha del servicio** y damos a siguiente. Dejamos el valor por defecto, **5432**



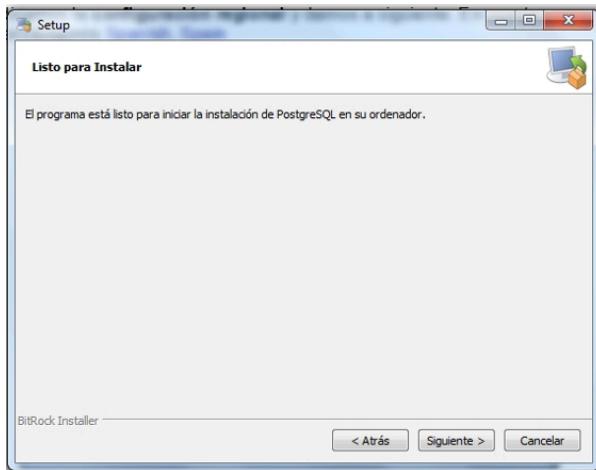
Configuración regional

- ✓ Indicamos la **configuración regional** y damos a siguiente. En nuestro caso seleccionamos **Spanish, Spain**



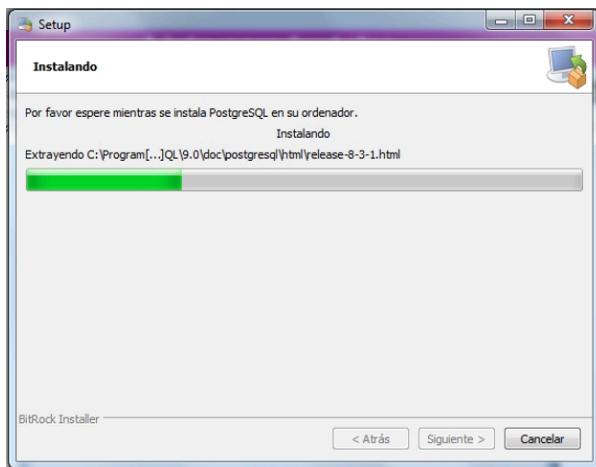
Listo para Instalar

- ✓ Aparece la pantalla que nos indica que ya está **todo listo para la instalación**. Pulsamos en siguiente y comienza la instalación.



Instalando PostgreSQL

- ✓ Aparece la pantalla que nos va indicando el **proceso de la instalación**.



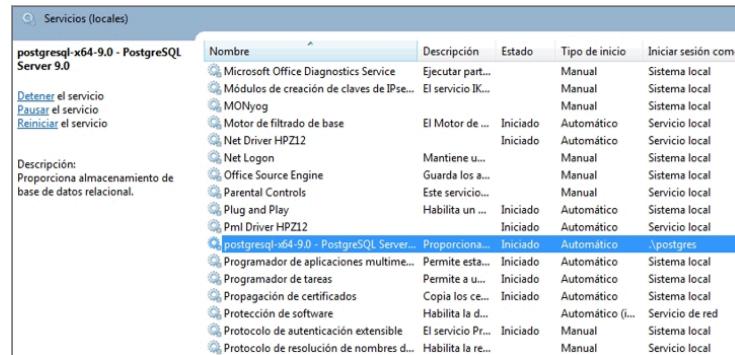
Fin de la instalación de PostgreSQL

- ✓ Aparece la pantalla que nos indica que la **instalación ha finalizado**.
- ✓ Desmarcamos “*¿Lanzar Stack Builder al finalizar?*” (siempre podremos echarle un vistazo después) y pulsamos en **Terminar**.



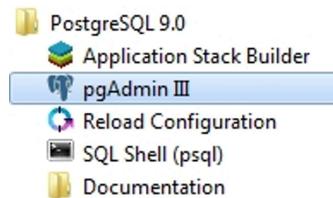
El servicio PostgreSQL

- ✓ Tras la instalación, de manera predeterminada el servicio PostgreSQL se iniciará de forma automática.
- ✓ Desde Herramientas Administrativas/Servicios podemos ver y cambiar la configuración de inicio del servicio, así como detenerlo, pausarlo y reiniciarlo.



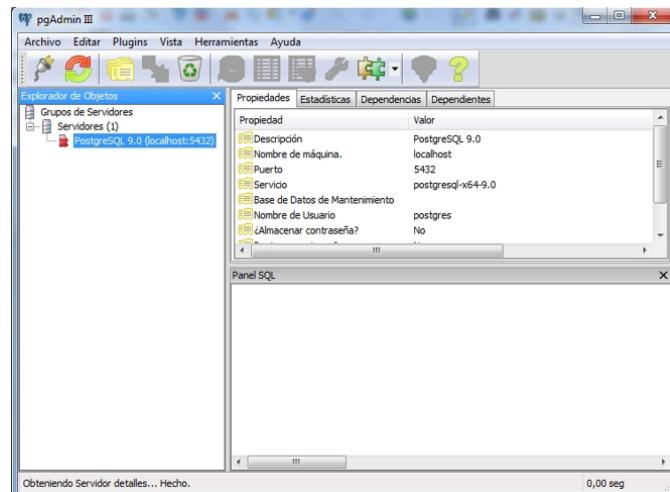
Trabajar con PostgreSQL

- ✓ Debe estar iniciado el servicio PostgreSQL.
- ✓ Desplegamos Inicio/Todos los Programas/ PostgreSQL. Aparecen las diferentes herramientas instaladas por defecto.
- ✓ El programa pgAdmin III permite administrar en modo gráfico PostgreSQL.
- ✓ Ejecutamos pgAdmin III.



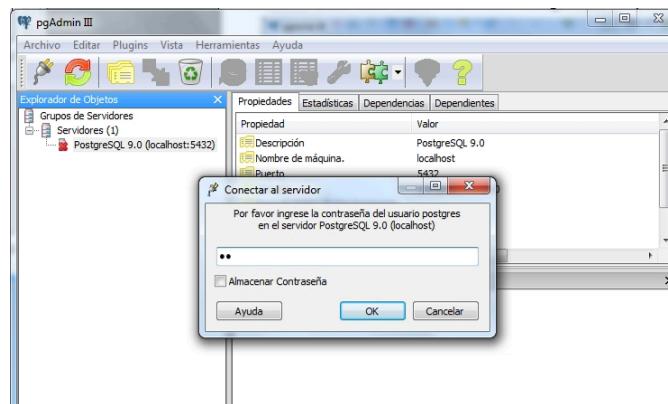
Ejecutar servidor PostgreSQL

- ✓ Hacemos doble clic sobre el servidor PostgreSQL 9.0 (localhost:5432).



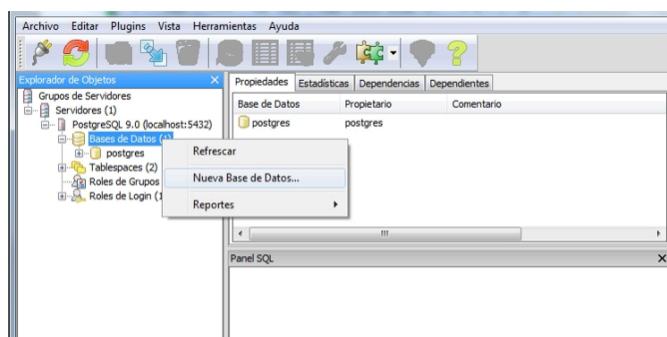
Introducir contraseña servidor PostgreSQL

- ✓ Introducimos la contraseña del superusuario de PostgreSQL establecida en la instalación y pulsamos OK



Trabajar con bases de datos de PostgreSQL

- ✓ Desde el **grupo Base de Datos** podemos **obtener los detalles** las bases de datos PostgreSQL, así como crear una nueva Base de Datos desde el menú contextual.



Credenciales

Imagen	Datos de licencia
Todas las capturas de pantalla de esta presentación, salvo la de la diapositiva tienen como datos de licencia: Autoría: Isabel M. Cruz Granados Licencia: Uso educativo-no comercial. Procedencia: Captura de pantalla del programa PostgreSQL de Global Development Group con licencia BSD.	
Diapositiva 12	Autoría: Microsoft Windows 7. Tipo de licencia: Copyright (Cita). Procedencia: Administrador de Servicios.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

[Resumen textual alternativo](#)

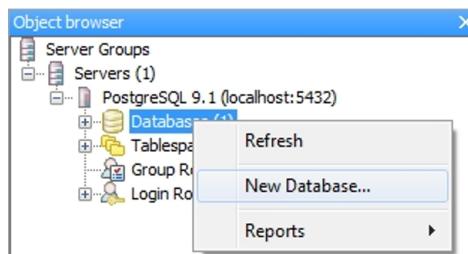
En la instalación se solicita una contraseña para el usuario **postgres**, que es el administrador por defecto de la Base de Datos. La utilidad **pgAdmin (III)** permite realizar esta administración de forma visual.

Para los ejemplos, crearemos una base de datos de prueba denominada **anaconda**. Con la ayuda de pgAdmin, esta tarea puede realizarse en uno cuantos clics:

PostgreSQL: Creación de la base de datos anaconda

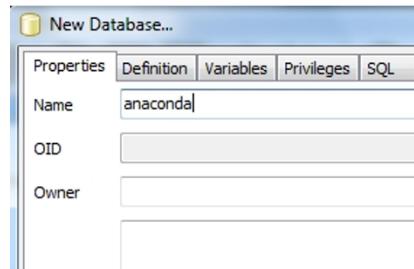
- ✓ Accedemos a **pgAdmin(III)**
- ✓ Nos conectamos al servidor por defecto (**localhost:5432**) haciendo doble clic en su ícono. Para completar la conexión, habrá que introducir la contraseña que le dimos al usuario **postgres** durante la instalación.
- ✓ Una vez autorizados, nos mostrará una serie de nodos. El denominado **Databases** es el que contiene las instancias de las bases de datos creadas hasta el momento (al principio solo habrá una, denominada **postgres** como el usuario)

- ✓ Para crear una nueva base de datos, hacemos clic con el botón derecho sobre el nodo y ejecutamos el comando **New Database**



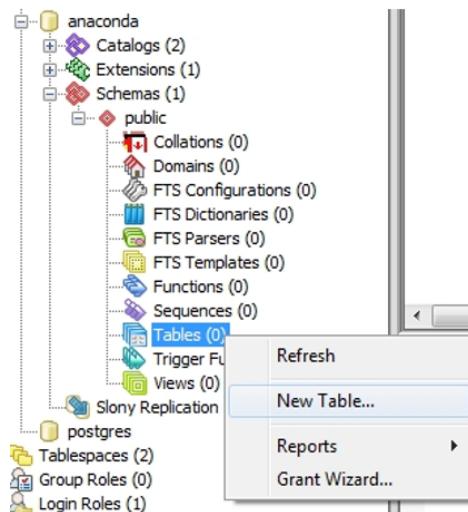
Crear la base de datos anaconda

- ✓ En la ventana **New Database**, escribimos **anaconda**.
- ✓ Si no se especifica lo contrario, la base de datos se creará con la codificación de caracteres **UTF8**. Esto puede cambiarse desde la ficha **Definition**, aunque siempre es una opción conveniente.
- ✓ Es conveniente especificar **Spanish_Spain.1252** en los campos **Collation** y **Character type** de esa misma ficha, para asegurarnos de que la base de datos va a ordenar correctamente los campos de texto, y que no habrá caracteres erróneos.
- ✓ Finalmente, presionamos **OK** para crear la base de datos.



Objetos de anaconda

- ✓ De **anaconda** cuelgan una serie de objetos, el más importante de los cuales es **Schemas** (Esquemas). Los esquemas son los verdaderos contenedores de la información.
- ✓ Una base de datos puede tener varios esquemas, aunque por defecto solo se crea uno denominado **public**.
- ✓ Dentro de un esquema, la información se organiza en unidades denominadas **Tables** (Tablas).
- ✓ Aunque podemos crear nuevas tablas desde el menú contextual con un sencillo clic, nuestro objetivo es aprender a hacerlo con Java desde el Netbeans.



Credenciales

Imagen	Datos licencia
--------	----------------

Todas las capturas de pantalla de esta presentación tienen la siguiente licencia.

Autoría: Isabel M. Cruz Granados
Licencia: Uso educativo-no comercial.
Procedencia: Captura de pantalla del programa PostgreSQL de Global Development Group con licencia BSD.

1 2 3 4
[Resumen textual alternativo](#)

Para saber más

En el siguiente enlace puedes consultar las características generales de PostgreSQL.

[Características generales de PostgreSQL.](#)

7.2.- Tipos de datos: tipos básicos y tipos estructurados.

Como ya sabes, los SGBDOR incorporan un conjunto muy rico de tipos de datos. PostgreSQL no soporta herencia de tipos pero permite **definir nuevos tipos de datos mediante los mecanismos de extensión**.

Te vamos a comentar tres categorías de tipos de datos que encontramos en PostgreSQL:

- ✓ **Tipos básicos:** el equivalente a los tipos de columna usados en cualquier base de datos relacional.
- ✓ **Tipos compuestos:** un conjunto de valores definidos por el usuario con estructura de fila de tabla, y que como tal puede estar formada por tipos de datos distintos.
- ✓ **Tipos array:** un conjunto de valores distribuidos en un vector multidimensional, con la condición de que todos sean del mismo tipo de dato (básico o compuesto). Ofrece más funcionalidades que el array del estándar SQL99.

Entre los **tipos básicos**, podemos destacar:

- ✓ **Tipos numéricos.** Aparte de valores enteros, números de coma flotante, y números de precisión arbitraria, PostgreSQL incorpora también un tipo entero auto-incremental denominado serial.
- ✓ **Tipos de fecha y hora.** Además de los típicos valores de fecha, hora e instante, PostgreSQL incorpora el tipo interval para representar intervalos de tiempo.
- ✓ **Tipos de cadena de caracteres.** Prácticamente los mismos que en cualquier BDR.
- ✓ **Tipos largos.** Como por ejemplo, el tipo BLOB para representar objetos binarios. En la actualidad presentes en muchas BDR como MySQL.

¿No soporta PostgreSQL el tipo estructurado?

Los **tipos compuestos** de PostgreSQL son el equivalente a los **tipos estructurados** definidos por el usuario del estándar SQL99. De hecho, son la base sobre la que se asienta el soporte a objetos.

Por ejemplo, podemos crear un nuevo tipo, el tipo dirección a partir del nombre de la calle (varchar), del número de la calle (integer) y del código postal (integer), de la siguiente forma:

```
CREATE TYPE direccion AS (
    calle varchar,
    numero integer,
    codigo_postal integer);
```

y luego definir la tabla de nombre afiliados, con una columna basada en el nuevo tipo:

```
CREATE TABLE afiliados AS(
    afiliado_id serial,
    nombre varchar(45),
    apellidos varchar(45),
    domicilio direccion);
```

Debes conocer

En el siguiente enlace encontrarás información acerca de los tipos de datos de PostgreSQL:

[Texto Documentación de los tipos de datos en PostgreSQL.](#)

Autoevaluación

Señala si la siguiente afirmación es verdadera o falsa.

El tipo array de PostgreSQL solo permite vectores.

Verdadero Falso

Falso

Es falso, se pueden declarar array multidimensionales.

7.3.- Conexión mediante JDBC.

La conexión de una aplicación Java con PostgreSQL se realiza mediante un conector tipo JDBC.



En la siguiente presentación te proporcionamos un tutorial que detalla tanto la obtención de este driver, como su integración en un proyecto de NetBeans.

PostgreSQL: Obtención e integración del driver JDBC

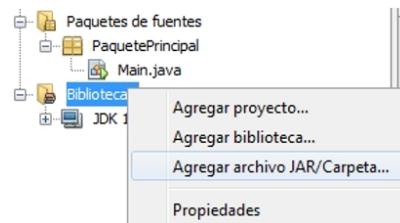
Descarga del driver JDBC de PostgreSQL

- ✓ Accedemos a la web oficial del driver JDBC para PostgreSQL <http://jdbc.postgresql.org>
- ✓ En la pestaña **Home**, seleccionamos el menú **Download** dentro de **About**; o bien, vamos directamente a la página de descargas <http://jdbc.postgresql.org/download.html>.

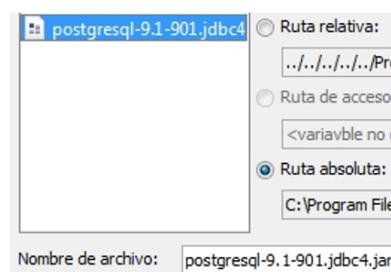
- ✓ En la sección **Current Version**, seleccionamos el driver más adecuado para el Java que tengamos instalada (normalmente, la última disponible para el **JDBC4 Postgres driver**)
- ✓ Guardamos el fichero **.jar** en un directorio, y anotamos la ruta para referencia posterior.

Integración en un proyecto del Netbeans

- ✓ Para integrar el driver en un proyecto del **Netbeans**, clic con el botón derecho en **Bibliotecas** para ejecutar el comando **Agregar archivo JAR/Carpeta**



- ✓ Se introduce la ruta donde se guardó el fichero **.jar** descargado, y listo.



Credenciales

Imagen	Datos de licencia
	Todas las capturas de pantalla de esta presentación, salvo la diapositiva 2, tienen como licencia: Autoría: Isabel M. Cruz Granados Licencia: Uso educativo-no comercial. Procedencia: Captura de pantalla del programa PostgreSQL de Global Development Group con licencia BSD..
Diapositiva 2	Autoría: Isabel M. Cruz Granados Licencia: Uso educativo-no comercial. Procedencia: Captura de pantalla del programa NetBeans, propiedad Sun Microsystems, bajo licencia GNU GPL v2

1 2 3
[Resumen textual alternativo](#)

Una vez integrado en un proyecto, el driver puede utilizarse desde cualquiera de sus clases mediante un:

```
import java.sql.*
```

Recuerda que en JDBC, una base de datos está representada por una [URL](#). La cadena correspondiente tiene una de las tres formas siguientes:

- ✓ jdbc: postgresql: base de datos
- ✓ jdbc: postgresql: //host/base de datos
- ✓ jdbc: postgresql: //host: puerto/base de datos

El nombre de host por defecto, del servidor PostgreSQL, será localhost, y el puerto por el que escucha el 5432. Como ves, esta cadena es idéntica a la empleada por otros sistemas gestores de bases de datos.

Para conectar con la base de datos, utilizaremos el método `DriverManager.getConnection()` que devuelve un objeto `Connection` (la conexión con la base de datos). Una de las posibles sintaxis de este método es:

```
Connection conn = DriverManager.getConnection(url, username, password);
```

Una vez abierta, la conexión se mantendrá operativa hasta que se llame a su método `close()` para efectuar la desconexión. Si al intentar conectar con la base de datos ésta no existe, se generará una excepción del tipo `PSQLException` "FATAL: no existe la base de datos ...". En cualquier caso se requiere un bloque `try-catch`.

```
/*nombre url de la base de datos almacenada en el servidor local. no hay
que poner el puerto ya q es por defecto
String url = "jdbc:postgresql://localhost/anaconda";*/
//conexión con la base de datos
Connection conn = null;
try {
    //realiza la conexión con la base de datos a la que apunta el url
    //resuelve la contraseña del usuario postgres
    conn = DriverManager.getConnection(url, "postgres", "1234");
} catch (SQLException e) {
    //realiza la conexión
    System.out.println(e.getMessage());
} finally {
    //cierra la conexión
    conn.close();
}
```

[Fragmento de código para la conexión con la base de datos anaconda.](#) (1 KB)

Para saber más

En el siguiente enlace encontrarás información detallada sobre el API JDBC de PostgreSQL:

[Documentación del API JDBC de PostgreSQL.](#)

7.4.- Consulta y actualización de tipos básicos.

PostgreSQL implementa los objetos como filas, las clases como tablas, y los atributos como columnas. Hablaremos por tanto de tablas, filas y columnas, tal y como lo hace PostgreSQL.

Para interactuar con PostgreSQL desde Java, vía JDBC, debemos enviar sentencias SQL a la base de datos mediante el uso de comandos.

Por tanto, si nuestra conexión es conn, para enviar un comando Statement haríamos lo siguiente:

- ✓ Crear la sentencia, por ejemplo.: Statement sta = conn.createStatement();
- ✓ Ejecutar la sentencia:
 - ◆ sta.executeQuery(string sentenciaSQL); si **sentenciaSQL** es una consulta (SELECT)
 - ◆ sta.executeUpdate(string sentenciaSQL); si **sentenciaSQL** es una actualización (INSERT, UPDATE O DELETE)
 - ◆ sta.execute(string sentenciaSQL); si **sentenciasql** es un CREATE, **DROP**, o un ALTER

Como ves, en PostgreSQL se utilizan estos comandos como en cualquier otra BDR.

En el siguiente enlace te proporcionamos un ejemplo que ilustra este hecho, con sentencias perfectamente conocidas por ti del modelo relacional. Se crean varias tablas en la base de datos anaconda, se insertan, modifican, eliminan y consultan datos de tipos básicos.

[Ejemplo del envío de comandos SQL mediante JDBC a PostgreSQL.](#) (0.51 MB)

En las próximas secciones, emplearemos estos mismos comandos para estudiar las nuevas características del estándar SQL99.

Citas para pensar

"No es sabio el que sabe donde está el tesoro, sino el que trabaja y lo saca".

Francisco de Quevedo

7.5.- Consulta y actualización de tipos estructurados.

Imaginemos que tenemos el tipo estructurado dirección:

```
CREATE TYPE direccion AS (
    calle varchar, numero integer, codigo_postal varchar);
```

y la tabla afiliados, con una columna basada en el nuevo tipo:

```
CREATE TABLE afiliados(afiliado_id serial, nombre varchar, apellidos varchar, domicilio direccion);
```

¿Cómo insertamos valores en una tabla con un tipo estructurado? Se puede hacer de dos formas:

- ✓ Pasando el valor del campo estructurado entre comillas simples (lo que obliga a encerrar entre comillas dobles cualquier valor de cadena dentro), y paréntesis para encerrar los subvalores separados por comas: INSERT INTO afiliados (nombre, apellidos, dirección).

```
INSERT INTO afiliados (nombre, apellidos, dirección)
VALUES ('Onorato', 'Maestre Toledo', ('Calle de Rufino', 56, 98080));
```

- ✓ Mediante la función ROW que permite dar valor a un tipo compuesto o estructurado.

```
INSERT INTO afiliados (nombre, apellidos, dirección)
VALUES ('Onorato', 'Maestre Toledo', ROW('Calle de Rufino', 56, 98080));
<br />
<br />
<br />
<br />
```

¿Cómo se referencia una subcolumna de un tipo estructurado?

- ✓ Se emplea la notación punto, '.', tras el nombre de la columna entre paréntesis, (tanto en consultas de selección, como de modificación). Por ejemplo:

```
SELECT (domicilio).calle FROM afiliados WHERE (domicilio).codigo_postal=98080
```

devolvería el nombre de la calle Maestre Toledo. Los paréntesis son necesarios para que el gestor no confunda el nombre del campo compuesto con el de una tabla.

Debes conocer

En el siguiente enlace encontrarás ejemplos de cómo se modifican los valores de un tipo estructurado, y más ejemplos sobre consultas e inserciones.

[Documentación sobre los tipos compuestos en PostgreSQL.](#)

Y ¿cómo se elimina el tipo estructurado?

- ✓ Se elimina con DROP TYPE, por ejemplo DROP TYPE dirección;

En el siguiente enlace puedes descargar un proyecto java que incluye más ejemplos de todo esto.

[Ejemplo de columnas de tipos compuestos en PostgreSQL.](#) (0.52 MB)

Autoevaluación

Señala la opción correcta. Para insertar valores en un tipo compuesto de PostgreSQL, utilizamos la siguiente notación:

- (tipo_complejo).subcolumna=valor;.
- La función ROW.
- tipo_complejo.(subcolumna)=valor
- Se pasan los valores del campo compuesto entre comillas dobles

No es la respuesta correcta, prueba de nuevo.

Muy bien, vamos por buen camino.

No es correcto.

No es cierto, en todo caso son comillas simples.

Solución

1. Incorrecto
2. Opción correcta
3. Incorrecto
4. Incorrecto

Para saber más

Puedes ver los tipos definidos por el usuario que soporta Oracle desde el siguiente enlace:

[Tipos definidos por el usuario de Oracle.](#)

7.6.- Consulta y actualización de tipos array.

PostgreSQL permite especificar **vectores multidimensionales como tipo de dato para las columnas** de una tabla. La única condición es que todos sus elementos sean del mismo tipo.

Por ejemplo:

- ✓ Declaración de una columna de tipo vector: nombre_columna tipo_dato[]
- ✓ Declaración de una columna de tipo matriz multidimensional: nombre_columna tipo_dato[][]

donde como se ve, sólo hay que agregar uno o más corchetes '[' al tipo de dato.

Aunque PostgreSQL permite especificar el tamaño de cada dimensión en la declaración, y acepta escribir:

nombre_columna tipo_dato[5] o bien nombre_columna tipo_dato[2][3], las versiones actuales ignoran estos valores en la práctica, de manera que los array declarados de esta forma tienen la misma funcionalidad que los del ejemplo anterior.

En realidad ninguna de estas declaraciones es conforme al estándar SQL99, que sólo contempla el tipo array como columnas de vectores unidimensionales declarados con la palabra reservada array:

En el siguiente ejemplo puedes ver la creación de una tabla con una columna de tipo array de varchar y como se insertan y consultan valores:

[Código java con ejemplo del tipo array en PostgreSQL.](#) (1 KB)

MATRIZ DE NECESIDADES				
NECESIDADES	SER	TENER	HACER	ESTAR
SUBSISTENCIA				
PROTECCIÓN				
AFECTO				
ENTENDIMIENTO				
PARTICIPACIÓN				
OJO				
CREACIÓN				
IDENTIDAD				
LIBERTAD				

```

CREATE TABLE "NEEDS"
(
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    needs_type VARCHAR(255) NOT NULL,
    needs_desc VARCHAR(255) NOT NULL,
    needs_val VARCHAR(255) NOT NULL
);

INSERT INTO "NEEDS" (name, needs_type, needs_desc, needs_val)
VALUES ('SUBSISTENCIA', 'ESTAR', 'Tener que comer para vivir', 'ESTAR');
INSERT INTO "NEEDS" (name, needs_type, needs_desc, needs_val)
VALUES ('PROTECCIÓN', 'ESTAR', 'Tener que sentirse seguro', 'ESTAR');
INSERT INTO "NEEDS" (name, needs_type, needs_desc, needs_val)
VALUES ('AFECTO', 'ESTAR', 'Tener que sentirse querido', 'ESTAR');
INSERT INTO "NEEDS" (name, needs_type, needs_desc, needs_val)
VALUES ('ENTENDIMIENTO', 'ESTAR', 'Tener que ser comprendido', 'ESTAR');
INSERT INTO "NEEDS" (name, needs_type, needs_desc, needs_val)
VALUES ('PARTICIPACIÓN', 'ESTAR', 'Tener que sentirse incluido', 'ESTAR');
INSERT INTO "NEEDS" (name, needs_type, needs_desc, needs_val)
VALUES ('OJO', 'ESTAR', 'Tener que ser visto', 'ESTAR');
INSERT INTO "NEEDS" (name, needs_type, needs_desc, needs_val)
VALUES ('CREACIÓN', 'ESTAR', 'Tener que sentirse creativo', 'ESTAR');
INSERT INTO "NEEDS" (name, needs_type, needs_desc, needs_val)
VALUES ('IDENTIDAD', 'ESTAR', 'Tener que sentirse individualizado', 'ESTAR');
INSERT INTO "NEEDS" (name, needs_type, needs_desc, needs_val)
VALUES ('LIBERTAD', 'ESTAR', 'Tener que sentirse libre', 'ESTAR');

CREATE TABLE "NEEDS_NEEDS"
(
    id SERIAL PRIMARY KEY,
    need_id1 INTEGER NOT NULL,
    need_id2 INTEGER NOT NULL,
    value DECIMAL(10,2) NOT NULL
);

INSERT INTO "NEEDS_NEEDS" (need_id1, need_id2, value)
VALUES (1, 2, 0.5);
INSERT INTO "NEEDS_NEEDS" (need_id1, need_id2, value)
VALUES (1, 3, 0.5);
INSERT INTO "NEEDS_NEEDS" (need_id1, need_id2, value)
VALUES (1, 4, 0.5);
INSERT INTO "NEEDS_NEEDS" (need_id1, need_id2, value)
VALUES (1, 5, 0.5);
INSERT INTO "NEEDS_NEEDS" (need_id1, need_id2, value)
VALUES (2, 3, 0.5);
INSERT INTO "NEEDS_NEEDS" (need_id1, need_id2, value)
VALUES (2, 4, 0.5);
INSERT INTO "NEEDS_NEEDS" (need_id1, need_id2, value)
VALUES (2, 5, 0.5);
INSERT INTO "NEEDS_NEEDS" (need_id1, need_id2, value)
VALUES (3, 4, 0.5);
INSERT INTO "NEEDS_NEEDS" (need_id1, need_id2, value)
VALUES (3, 5, 0.5);
INSERT INTO "NEEDS_NEEDS" (need_id1, need_id2, value)
VALUES (4, 5, 0.5);

```

Debes conocer

En el siguiente enlace encontrarás ejemplos de cómo consultar y actualizar columnas de tipo array.

[Documentación sobre el tipo array en PostgreSQL.](#)

Autoevaluación

Señala las opciones correctas. El tipo array en PostgreSQL:

Requiere que todos sus elementos sean del mismo tipo.

Permite declarar columnas como matriz de 2 dimensiones

Siempre se debe especificar entre corchetes el tamaño de la dimensión.

Solo se pueden especificar vectores de una dimensión.

[Mostrar retroalimentación](#)

Solución

1. Correcto
2. Correcto
3. Incorrecto

4. Incorrecto

7.7.- Funciones del gestor desde Java.

Recuerda que el estándar SQL99 introduce la posibilidad de crear funciones de usuario que puedan quedar almacenadas en el gestor de bases de datos, utilizando un Lenguaje Procedural (PL). Estas funciones pueden definir el comportamiento de ciertos objetos y entonces, se llaman métodos. Oracle soporta métodos en este sentido, no así el gestor PostgreSQL.

Lo que si podremos hacer en PostgreSQL es crear funciones con prestaciones para los nuevos tipos de datos, tipos estructurados y array, y llamarlas desde una aplicación Java.

En PostgreSQL se pueden construir funciones mediante diferentes lenguajes:

- ✓ **Funciones de lenguaje de consultas o funciones SQL** (Escritas en lenguaje SQL)
- ✓ **Funciones de lenguaje procedural** (Escritas en lenguaje PL/pgSQL, PL/Java, etc)
- ✓ **Funciones de lenguaje de programación** (Escritas en un lenguaje de compilado, como C o C++)

Veremos algún ejemplo de funciones en SQL. Sobre estas funciones debes saber que:

- ✓ **Los parámetros de la función** pueden ser cualquier tipo de dato admitido por el gestor (un tipo básico, un tipo compuesto, un array o alguna combinación de ellos).
- ✓ **El tipo devuelto** puede ser un tipo básico, un array o un tipo compuesto.
- ✓ Su estructura general tiene la forma:

```
CREATE FUNCION nombre_funcion(tipo_1,tipo_2,...)
    RETURN tipo AS $$sentencia_sql$$
    LANGUAGE SQL
```

Y debes tener en cuenta que:

- ✓ Los argumentos de la función SQL se pueden referenciar en las consultas usando la sintaxis \$n:, donde \$1 se refiere al primer argumento, \$2 al segundo, y así sucesivamente. Si un argumento es un tipo compuesto, entonces se usará la notación '' para acceder a sus subcolumnas.
- ✓ Por último, al final de la función hay que especificar que la función se ha escrito en lenguaje SQL mediante las palabras clave **LANGUAGE SQL**.

Por ejemplo, desde Java podemos crear una función que convierte un tipo estructurado, por ejemplo el tipo dirección, en una cadena. Para ello, ejecutaremos un comando **Statement (sta)** con el código SQL de creación de la función:

```
sta.execute ("CREATE FUNCTION cadena_direccion(direccion)"+
    "RETURNS varchar AS 'SELECT $1.calle||' '|| CAST($1.numero AS varchar)" +
    "LANGUAGE SQL"; )
```

Si una función tiene algún parámetro de tipo estructurado, no se podrá eliminar el tipo hasta que no se elimine la función. Se puede usar **DROP C nb_tipo CASCADE** para eliminar de manera automática las funciones que utilizan ese tipo.

En el siguiente enlace te puedes descargar un ejemplo donde se crean funciones del gestor y se invocan desde una aplicación Java.

[Funciones Gestor.](#) (0.51 MB)

Autoevaluación

Señala si la siguiente afirmación es verdadera o falsa.

En PostgreSQL se pueden crear funciones con parámetros de tipo estructurado, pero no de tipo array

Verdadero Falso

Falso

Es falso, se pueden pasar tanto parámetros de tipo estructurado como array.

Para saber más

En el siguiente enlace puedes ver muchos ejemplos de funciones PostgreSQL.

[Tutorial PostgreSQL.](#) Resumen textual alternativo

8.- Gestión de transacciones.

Caso práctico

—¡Por fin ha llegado el viernes! —dice **Ana** a **Juan**, y continúa— después de estas dos últimas semanas revisando sistemas objeto-relacionales ¡he terminado agotada!, pero el esfuerzo me ha compensado. Ahora tengo mucho más claro, las diferencias entre una base de objetos y una objeto-relacional. Pero... —se queda pensativa por un instante y entonces continúa— las transacciones ¿se gestionan de manera similar en ambos tipos de sistemas?

Juan le responde, —justo es eso lo que nos queda por ver, **Ana**. ¿Empezamos hoy o lo dejamos para el lunes?



Ana sonríe y le dice —Creo que **Ada** viene para acá, ¿no oyés sus pasos?

Como en cualquier otro Sistema de Bases de datos, en un Sistema de bases de objetos u objeto relacional, **una transacción** es un conjunto de sentencias que se ejecutan formando una unidad de trabajo, esto es, en forma indivisible o atómica, o se ejecutan todas o no se ejecuta ninguna.

Mediante la gestión de transacciones, los sistemas gestores proporcionan un acceso concurrente a los datos almacenados, mantienen la integridad y seguridad de los datos, y proporcionan un mecanismo de recuperación de la base de datos ante fallos.

Un ejemplo habitual para motivar la necesidad de transacciones es el traspaso de una cantidad de dinero (digamos 10000€) entre dos cuentas bancarias. Normalmente se realiza mediante dos operaciones distintas, una en la que se decrementa el saldo de la cuenta origen y otra en la que incrementamos el saldo de la cuenta destino.

Para garantizar la integridad del sistema (es decir, para que no aparezca o desaparezca dinero), las dos operaciones tienen que completarse por completo, o anularse íntegramente en caso de que una de ellas falle.

Las transacciones deben cumplir el criterio **ACID**.

- ✓ **Atomicidad.** Se deben cumplir todas las operaciones de la transacción o no se cumple ninguna; no puede quedar a medias.
- ✓ **Consistencia.** La transacción solo termina si la base de datos queda en un estado consistente.
- ✓ **Isolation** (Aislamiento). Las transacciones sobre la misma información deben ser independientes, para que no interfieran sus operaciones y no se produzca ningún tipo de error.
- ✓ **Durabilidad.** Cuando la transacción termina el resultado de la misma perdura, y no se puede deshacer aunque falle el sistema.

Algunos sistemas proporcionan también **puntos de salvaguarda** (savepoints) que permiten descartar selectivamente partes de la transacción, justo antes de acometer el resto. Así, después de definir un punto como punto de salvaguarda, puede retrocederse al mismo. Entonces se descartan todos los cambios hechos por la transacción después del punto de salvaguarda, pero se mantienen todos los anteriores.

Originariamente, los puntos de salvaguarda fueron una aportación del estándar SQL99 de las Bases de Datos Objeto-Relacionales. Pero con el tiempo, se han ido incorporando también a muchas Bases de Datos Relacionales como MySQL (al menos cuando se utiliza la tecnología de almacenamiento InnoDB).

Para saber más

Puedes consultar el siguiente enlace para más información sobre los puntos de salvaguarda en MySQL:

[Documentación sobre los puntos de salvaguarda en MySQL.](#)

8.1.- Transacciones en una base objeto-relacional.

Los SGBDOR gestionan transacciones mediante las sentencias COMMIT (confirmar transacción) y ROLLBACK (deshacer transacción).

JDBC permite agrupar instrucciones SQL en una sola transacción. Así, podemos asegurar las propiedades ACID usando las facilidades transaccionales del JDBC.

El control de la transacción lo realiza el objeto Connection. Cuando se crea una conexión, por defecto es en modo AUTO COMMIT= TRUE. Esto significa que cada instrucción individual SQL se trata como una transacción en sí misma, y se confirmará en cuanto que la ejecución termine.

Por tanto, si queremos que un grupo de sentencias se ejecuten como una transacción, tras abrir una conexión Connection conn; habrá que:

- ✓ Poner autocommit=false de la siguiente manera:

```
if (conn.getAutoCommit() )  
    conn.setAutoCommit( false );
```

- ✓ Mediante un bloque try-catch controlar si se deshace conn.rollback o confirma conn.commit la transacción iniciada con esa conexión.

En el siguiente enlace puedes ver como se controlan una secuencia de insert como una transacción en PostgreSQL.

[Ejemplo de transacción en PostgreSQL.](#)

Citas para pensar

"El experimentador que no sabe lo que está buscando no comprenderá lo que encuentra".

Claude Bernard

8.2.- Transacciones en un gestor de objetos.

Las transacciones en un sistema de objetos puro, se gestionan mediante COMMIT y ROLLBACK .

Un fallo causa un rollback de los datos, como normalmente sucede en las bases de datos relacionales. En cuanto a la concurrencia, las bases de objetos verifican en qué momento permiten el acceso en paralelo a los datos. Este acceso concurrente implica que más de una aplicación o hilo podrían estar leyendo o actualizando los mismos objetos a la vez. Esto se suele denominar **commit de dos fases** (dos procesos pueden trabajar sobre el mismo objeto concurrentemente). Para ello se utiliza normalmente bloqueo de datos para lecturas y escrituras.



Veamos un ejemplo sencillo con la base de objetos db4o. **En Db4o:**

- ✓ Siempre se trabaja dentro de una transacción.
- ✓ Al abrir un ObjectContainer se crea e inicia implícitamente una transacción.
- ✓ Las transacciones se gestionan explícitamente mediante commit y rollback.
- ✓ La transacción actual hace commit implícitamente cuando se cierra el ObjectContainer.

Por ejemplo, podemos deshacer una transacción con rollback , restaurando la base de objetos al estado anterior, justo al estado después del último commit o rollback.

En el siguiente ejemplo, al ejecutar db.rollback después de db.store(p3) y db.store(p4), hará que los objetos p3 y p4 no se almacenen en la base de objetos.

```
db.begin(); //se pone en modo de almacenamiento los objetos con el método begin()
db.createObjectContainer();
db.commit(); //finalizar explícitamente la transacción con commit
//...
p损ente p3 = new p损ente("Jes%u00fas", "Fernandez", "fernandez@objeto.com", 200);
p损ente p4 = new p损ente("Pedro", "Alvarez", "pavelschke@mail.com", 300);
//...
db.commit(); //se cierra la transacción
db.rollback(); //se borra la transacción comenzada de p3 y p4 en la DB4O
```

Autoevaluaci%u00f3n

Señala si la siguiente afirmaci%u00f3n es verdadera o falsa.

Una transacci%u00f3n en Db4o se deshace mediante rollback.

- Verdadero Falso

Verdadero

Es verdadero, con rollback se deshacen transacciones.

Para saber m%u00e1s

Desde el siguiente enlace puedes ver una presentaci%u00f3n tipo resumen con muchos de los aspectos tratados en esta unidad.

[Persistencia de objetos y db4o.](#) (0.61 MB)

Anexo I.- Estándar ODMG-93 u ODMG.

El **estándar ODMG** (Object Database Management Group) trata de estandarizar conceptos fundamentales de los Sistemas Gestores de Bases de Datos Orientados a Objetos (SGBDOO) e intenta definir un **SGBDOO como un sistema que integra las capacidades de las bases de datos con las capacidades de los lenguajes de programación orientados a objetos**, de manera que los objetos de la base de datos aparezcan como objetos del lenguaje de programación.

Fué desarrollado entre los años 1993 y 1994 por representantes de un amplio conjunto de empresas relacionadas con el desarrollo de software y sistemas orientados a objetos.

1. Arquitectura del estándar ODMG

La arquitectura propuesta por ODMG consta de:

- ✓ Un **modelo de objetos** que permite que tanto los diseños, como las implementaciones, sean portables entre los sistemas que lo soportan.
 - ✓ Un **sistema de gestión** que soporta un lenguaje de bases de datos orientado a objetos, con una sintaxis similar a un lenguaje de programación también orientado a objetos.
 - ✓ Un **lenguaje de base de datos** que es especificado mediante:
 - ▷ Un Lenguaje de Definición de Objetos (ODL)
 - ▷ Un Lenguaje de Manipulación de Objetos (OML)
 - ▷ Un Lenguaje de Consulta (OQL)
- siendo todos ellos portables a otros sistemas con el fin de conseguir la portabilidad de la aplicación completa.

- ✓ Enlaces con lenguajes Orientados a Objetos como C++, Java, Smalltalk.

El **modelo de objeto ODMG** es el modelo de datos en el que están basados el ODL y el OQL. Este modelo de objeto proporciona los tipos de datos, los constructores de tipos y otros conceptos que pueden utilizarse en el ODL para especificar el esquema de la base de datos de objetos.

Vamos a destacar algunas de las **características más relevantes** del estándar ODMG:

- ✓ Las primitivas básicas de modelado son los **objetos** y los **literales**.
- ✓ Un objeto tiene un **Identificador de Objeto** (OID) y un estado (valor actual) que puede cambiar y tener una estructura compleja. Un literal no tiene OID, pero si un valor actual, que es constante.
- ✓ El estado está definido por los valores que el objeto toma para un conjunto de propiedades. Una propiedad puede ser:
 - ▷ Un atributo del objeto.
 - ▷ Una interrelación entre el objeto y otro u otros objetos.
- ✓ Objetos y literales están organizados en **tipos**. Todos los objetos y literales de un mismo tipo tienen un comportamiento y estado común.
- ✓ Un objeto queda descrito por cuatro características: identificador, nombre, tiempo de vida y estructura.
- ✓ Los tipos de objetos se descomponen en atómicos, colecciones y tipos estructurados.
 - ▷ **Tipos atómicos** o básicos: constan de un único elemento o valor, como un entero.
 - ▷ **Tipos estructurados**: compuestos por un número fijo de elementos que pueden ser de distinto tipo, como por ejemplo una fecha.
 - ▷ **Tipos colección**: número variable de elementos del mismo tipo. Entre ellos:
 - Set: grupo desordenado de elementos y sin duplicados.
 - Bag: grupo desordenado de elementos que permite duplicados.
 - List: grupo ordenado de elementos que permite duplicados.
 - Array : grupo ordenado de elementos que permite el acceso por posición.

Algunos fabricantes sólo ofrecen vinculaciones de lenguajes específicos, sin ofrecer capacidades completas de ODL y OQL.

Anexo.- Licencias de recursos.

Licencias de recursos utilizados en la Unidad de Trabajo.

Recurso (1)	Datos del recurso (1)	Recurso (2)	Datos del recurso (2)
	Autoría: Agustín Díaz. Licencia: CC-BY-SA. Procedencia: http://www.flickr.com/photos/yonmacklein/22105978/		Autoría: Pablo Sanz Almogera. Licencia: CC BY-NC-SA. Procedencia: http://www.flickr.com/photos/pablosanz/2839818050/
	Autoría: Luis Pérez. Licencia: CC BY-NC-SA. Procedencia: http://www.flickr.com/photos/luipermom/2934628860/		Autoría: RonLD. Licencia: CC-BY. Procedencia: http://www.flickr.com/photos/rld/1204516152
	Autoría: Jeff Kubina. Licencia: CC BY-SA. Procedencia: http://www.flickr.com/photos/kubina/912714753		Autoría: Speric. Licencia: CC BY-NC. Procedencia: http://www.flickr.com/photos/ericfarkas/248666729/
	Autoría: Olga Berrios. Licencia: CC BY. Procedencia: http://www.flickr.com/photos/ofernandezberrios/368893337/		

Condiciones y términos de uso de los materiales

Materiales desarrollados inicialmente por el Ministerio de Educación, Cultura y Deporte y actualizados por el profesorado de la Junta de Andalucía bajo licencia Creative Commons BY-NC-SA.



Antes de cualquier uso leer detenidamente el siguiente [Aviso legal](#)

Historial de actualizaciones

Versión: 01.00.09	Fecha de actualización: 21/07/21
Actualización de materiales y correcciones menores.	
Versión: 01.00.00	Fecha de actualización: 11/11/13
Versión inicial de los materiales.	

