Módulo: BASES DE DATOS

Unidad 7: Uso de bases de datos no relacionales

Sesión 1: Bases de Datos no relacionales

Descripción:

La presente sesión vamos a ver las características de las bases de datos no relacionales, los diferentes tipos de bases de datos no relacionales que hay y como almacena la información cada una de ellas.

Para entender esto mejor vamos a ver ejemplos de SGBDs no relacionales de estos tipos así podremos identificar sus elementos, las opciones que nos dan y sus características principales.

Criterios de Evaluación:

- RA07_a: Se han caracterizado las bases de datos no relacionales.
- RA07_b: Se han evaluado los principales tipos de bases de datos no relacionales.
- RA07_c: Se han identificado los elementos utilizados en estas bases de datos.
- RA07_d: Se han identificado distintas formas de gestión de la información según el tipo de base de datos no relacionales.

Objetivos:

- Conocer las características de las bases de datos no relacionales.
- Identificar los tipos de bases de datos no relacionales.
- Conocer distintas formas de gestión de la información según el tipo de base de datos no relacionales.

Recursos:

- Acceso a Internet.
- Software ofimático.

- eXist.
- Neodatis.
- Cassandra.

Conceptos a revisar previamente:

- Características de las bases de datos no relacionales.
- Tipos de bases de datos no relacionales.
- Elementos de las bases de datos no relacionales.
- Sistemas gestores de bases de datos no relacionales

Resolución de la práctica:

Bases de datos documentales

Son **almacenes de datos de documentos**, se suelen almacenar como conjuntos clave y valor, donde la clave es un ID y el valor es un fichero de tipo XML, JSON, otros formatos e incluso como texto sin formato, que son modelos de datos eficientes e intuitivos para los desarrolladores.

Clave	Documento
1001	<EMPLEADOS> <EMP_ROW> <EMP_NO>7369</EMP_NO> <APELLIDO>SANCHEZ</APELLIDO> <OFICIO>EMPLEADO</OFICIO> <SALARIO>1040</SALARIO> </EMP_ROW> <EMP_ROW> <EMP_NO>7499</EMP_NO> <APELLIDO>ARROYO</APELLIDO> <OFICIO>VENDEDOR</OFICIO> <SALARIO>1500</SALARIO> </EMP_ROW> </EMPLEADOS>
1002	<departamentos> <DEP_ROW> <DEPT_NO>20</DEPT_NO> <DNOMBRE>INVESTIGACION</DNOMBRE> </DEP_ROW> <DEP_ROW> <DEPT_NO>30</DEPT_NO> <DNOMBRE>VENTAS</DNOMBRE> </DEP_ROW> </departamentos>

Imagen: Ejemplo BD Documental

La información se puede recuperar en base a su **clave de documento que lo identifica de manera unívoca**. Estas claves se pueden establecer automáticamente o de forma manual, y también se pueden indexar o usar hash sobre ellos para que sea más eficiente. También permite hacer consultas más avanzadas sobre el contenido del documento.

En la imagen *"Ejemplo BD Documental"*, vemos como tiene un valor que es la clave que se puede generar de varias maneras y un segundo valor que es el documento.

Normalmente la manera de almacenar documentos es que correspondan con toda la información una entidad. La información de dicha entidad es específica para cada aplicación, por lo que un solo documento puede contener la información que contendríamos en una o varias tablas relacionales.

Lo mejor de este tipo de base de datos es que los documentos no tienen que tener la

misma estructura lo que redunda en la **flexibilidad del modelo**, aunque si tiene una **estructura jerárquica**. Permite que evolucionen según las necesidades de las aplicaciones.

El modelo de documentos funciona bien con catálogos, perfiles de usuario, configuraciones y sistemas de administración de contenido en los que cada documento es único y evoluciona con el tiempo.

Algunos ejemplos de este tipo son eXists, MongoDB, CouchDB.

eXist

Es una base de datos documental, de código abierto y que almacena **archivos nativos XML** de acuerdo al modelo de datos XML. El motor de base de datos está completamente escrito en Java.

Con el SGBD vienen incorporadas herramientas que permiten ejecutar consultas directamente sobre la BD.

Imagen: Logo de eXist

Características:

- Los **documentos XML se almacenan en colecciones**, las cuales pueden estar anidadas; desde un punto de vista práctico el almacén de datos funciona como un sistema de fichero. Cada documento está en una colección.
- No es necesario que los documentos tengan una DTD o un XML Schema asociado, y dentro de **una colección pueden almacenarse documentos de cualquier tipo**.
- Soporta los estándares de consulta XPath, XQuery y XSLT.
- Soporta indexación de documentos.
- Soporte para la actualización de datos y para multitud de protocolos como SOAP, XML-RPC, WebDav y REST.

Bases de datos en columnas

Organizan los **datos en columnas y filas**, es en cierta manera similar a una base de datos relacional desde un punto de vista conceptual pero es mucho más simple y tiene un enfoque desnormalizado en la que vamos a añadir datos dispersos.

Cada columna que almacenemos tendrá grupos datos que están relacionados de forma lógica, como se ve en la imagen "*Ejemplo BD Columnas*", a cada ID de Cliente le asociamos una columna Identidad con la información de nombre y apellidos, y también una columna Contacto que tiene los datos relativos al contacto de una persona como puede ser Teléfono o Email y podría aparecer cualquier otra información que estuviera relacionada, pero como vemos no tienen que aparecer todos.

ID_Cliente	Identidad
1	**Nombre:** Pablo **Apellidos:** Jiménez
2	**Nombre:** Sofia **Apellidos:** Martín
3	**Nombre:** Javier **Apellidos:** Cerezo
4	**Nombre:** Pedro **Apellidos:** Casas

ID_Cliente	Contacto
1	**Telefono:** 9113654574 **Email:** pablo@foc.es
2	**Telefono:** 953657281
3	**Email:** javier@foc.es
4	**Telefono:** 645663721 **Email:** pedro@foc.es

Imagen: Ejemplo BD Columnas

Esta estructura, en la que las filas pueden variar dinámicamente es una gran ventaja que lo hace muy adecuado para datos con esquemas variables.

Normalmente las claves de las filas son el índice, aunque también en algunas implementaciones se pueden hacer índices secundarios para recuperar datos por el valor de las columnas en vez de por el ID.

Bases de datos de clave / valor

Básicamente **son tablas hash grandes en las que cada clave es única y el valor se guarda usando una función hash adecuada**. Una función hash simplemente es un método por el que un conjunto de datos se convierten en un rango de salida finito, por lo que vamos a pasar esta función hash a los datos que queremos almacenar y nos va a devolver un valor que usaremos de clave.

Los valores que guardamos en la base de datos no se almacenan de forma que se puedan entender a simple vista, si no que es el SGBD el que interprete la información.

Clave	Valor
AE036F	1101111000101011111000101010101101010101...
AAB312	1111010101010010101010100101010100111001...
E814CA	1100101010101010100111000111101101010101...
F4B5C1	0001110101001011101010101010101101110101...

Imagen: Ejemplo BD Clave Valor

Así que no se podrá acceder a la información de dentro si no que habrá que captar el valor completo, se utiliza mucho para almacenar objetos binarios como pueden ser documentos, imágenes o vídeos. La mayoría de las bases de datos de este tipo solo permiten consulta, inserción y eliminación, siendo la modificación una sobrescritura de todo el valor completo.

Este tipo de bases de datos es uno de los más populares, **son muy óptimas para las aplicaciones que hacen búsquedas por medio de la clave**, pero si necesitas información que haya dentro del valor pierden eficiencia.

Son **altamente divisibles y permiten escalado horizontal** a niveles que otros tipos de bases de datos no son capaces de alcanzar. Por lo que es fácil distribuir estas bases de datos entre diferentes nodos. Se utilizan mucho en juegos, dispositivos conectados a internet y en aplicaciones como las historias de las redes sociales.

Algunos ejemplos de este tipo son Cassandra, Redis, BigTable o HBase.

Cassandra

Cassandra es un SGBD no relacional desarrollado por Facebook en Java y de código abierto, con la idea de que no tenga fallos y tenga una alta disponibilidad.

Imagen: Logo de Cassandra.

El diseño de Cassandra permite guardar grandes cantidades de datos, por lo que podemos manejar cantidades masivas de datos, repartidos entre muchos servidores, proporcionando un servicio de alta disponibilidad sin fallos.

Características:

- **Datos distribuidos**. Está distribuida y replicada en muchos los nodos utilizando una arquitectura peer to peer, en la que los nodos se sincronizan de forma automática teniendo todos el mismo rol sin que ninguno sea un nodo central, este tipo de arquitectura se llama masterless.

- **Escalabilidad horizontal**. Añadiendo nodos, es fácil de aumentar la capacidad de almacenamiento y la latencia. Se basa en utilizar equipos de bajo coste, ahorrando y abriendo la posibilidad de crecer rápidamente si es necesario.

- **Alta disponibilidad**. Al ser un sistema sin nodo central no hay un punto donde todo falle. En caso de que alguno de los nodos falle, la disponibilidad está garantizada, ya que se puede continuar consultando la información a otro nodo.

- **Consistencia**. El único problema es que la consistencia no está garantizada, ya que tiene una alta replicación. Pero esta consistencia es configurable tanto para lecturas como escrituras, aunque esto afecta de manera considerable al rendimiento.

- **Base de datos de columnas y Clave/Valor**: Es un hibrido entre dos esquemas de almacenamiento: las bases de datos de columnas y las de clave-valor. Por lo que usa una clave que apunta a una familia de columnas, lo que le proporciona una mayor compresión de la información y una mejor eficiencia en las respuestas. Este método de almacenamiento es muy eficiente, por ejemplo, para la realización de búsquedas de columnas no necesita leer el registro completo para acceder a unos pocos valores.

- **Lenguaje similar a SQL**, CQL (Cassandra Query Languaje).

- No es posible efectuar joins entre tablas.

- Interfaz de cliente relativamente fácil de usar.

Bases de datos de grafos

Son almacenes de datos que **utilizan un modelo basado en grafos**, utilizando nodos que serían las entidades y aristas entre los nodos que nos dan información de como o en que dirección se relacionan.

La idea de este modelo es hacer **consultas eficazmente recorriendo la red de nodos y aristas**. En el la imagen "*Ejemplo de BD de Grafos*", que representaría una base de datos de la organización de una empresa donde los nodos son empleados y departamentos, y las flechas las relaciones que hay entre los nodos, vemos como la estructura nos facilita cierto tipo de consultas como buscar los empleados de tal departamento o los que dependen de tal empleado.

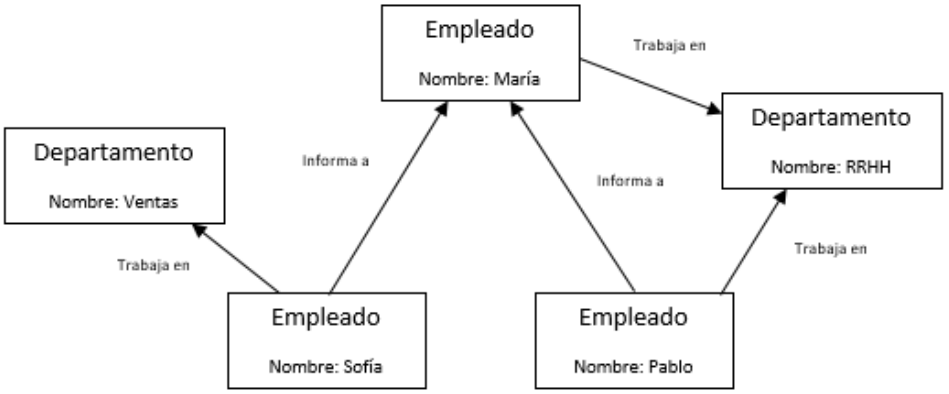


Imagen: Ejemplo BD Clave Valor

Este tipo de bases de datos ofrece una **navegación más eficiente entre relaciones que en un modelo relacional**. Aunque para sacarle el mayor rendimiento la estructura del grafo debe estar normalizada.

Algunos ejemplos de este tipo son Neo4j, InfoGrid o Virtuoso.

Bases de datos de series temporales

Son **conjuntos de datos organizados por tiempo**. Esto suelen admitir muchas operaciones de escritura, ya que la idea es que **recopilen grandes cantidades de información en tiempo real**. No suelen hacerse operaciones de modificación y los eliminados suelen ser masivos.

Están optimizado para estos tipos de datos, por ejemplo se utilizan para almacenar información de sensores, contadores o logs. Algo similar a los que vemos en la imagen "*Ejemplo BD Series Temporales*".

Timestamp	id_dispositivo	Valor
2023-02-07 02:10:43.123	1	36
2023-02-07 02:12:03.513	2	42
2023-02-07 02:12:12.027	7	17
2023-02-07 02:12:26.412	1	38

Imagen: Ejemplo BD Series Temporales

Normalmente son registros pequeños pero gran cantidad de ellos y que crecen de manera rápida.

Algunos ejemplos de este tipo son InfluxDB, Prometheus o TimescaleDB.

Bases de datos orientadas a objetos

Las Bases **de Datos Orientadas a Objetos** (BDOO) son aquellas cuyo modelo de datos está orientado a objetos, y por tanto soportan el paradigma orientado a objetos almacenando métodos y datos. Su origen se debe principalmente a la existencia de problemas para representar cierta información y modelar ciertos aspectos del mundo real.

Las Bases de Datos Orientadas a Objetos simplifican la programación orientada a objetos (POO) almacenando directamente los objetos en la BD y empleando las mismas estructuras y relaciones que los lenguajes de POO.

Las **características** asociadas a las **BDOO** son las siguientes:

- Los datos se **almacenan como objetos**.
- Cada objeto se identifica mediante un **identificador único**, este identificador no es modificable por el usuario.
- Cada objeto **define sus métodos y atributos y la interfaz** mediante la cual se puede acceder a él, el usuario puede especificar qué atributos y métodos pueden ser usados desde fuera.

En definitiva, un **SGBDOO debe cumplir las características de un SGBD**: persistencia, concurrencia, recuperación ante fallos, gestión del almacenamiento secundario y facilidad de consultas; **y las características de un sistema orientado a objetos (OO)**: encapsulación, identidad, herencia y polimorfismo.

Algunos ejemplos de este tipo de bases de datos son NeoDatis, Gemstone o Db4o.

NeoDatis

Es un sistema gestor de base de datos embebido, que soporta el paradigma orientado a objetos, de código abierto, que permite ser explotado desde los lenguajes Java, .Net, Groovy y Android.

Neodatis ODB aparece **para evitar el desfase objeto-relacional entre los mundos de objetos y relacionales**.

Los objetos se pueden almacenar y recuperar en pocas líneas de código de manera nativa, evitando realizar el mapeo con tablas.

Imagen: Logo de *NeoDatis*

Tiene una capa de persistencia nativa y transparente real para Java, .Net y Mono.

Decimos objetos porque la unidad persistente básica es un objeto, no una tabla. Nativo y transparente porque persiste directamente los objetos tal como existen en el lenguaje de programación nativo, sin ninguna conversión.

Características:

- **Simple**: es muy simple e intuitivo: el tiempo de aprendizaje es muy corto. La API es simple y no requiere aprender ninguna técnica de mapeo. Con una sola línea de código se puede almacenar un objeto tal como son y sin necesidad de instalación.
- **Pequeña**: Con incluir un jar/dll se puede empaquetar en la aplicación.
- **Segura y robusta**: Admite transacciones ACID para garantizar la integridad de los datos de la base de datos. Usa la recuperación automática de transacciones en el inicio.
- **Ligera**: usa un solo archivo de base de datos.
- **Multiplataforma**. Se ejecuta en la plataforma Java y .Net.
- **Alta disponibilidad**. Los datos siempre están disponibles. Permite exportar e importa todos los datos a un formato XML estándar
- **Productividad**: Permite almacenar datos con muy pocas líneas de código. No hay necesidad de modificar las clases que deben persistir y no se necesita mapeo.

Para ver un ejemplo con Neodatis, vamos a crear un proyecto Java

Con una clase Departamento, como la siguiente:

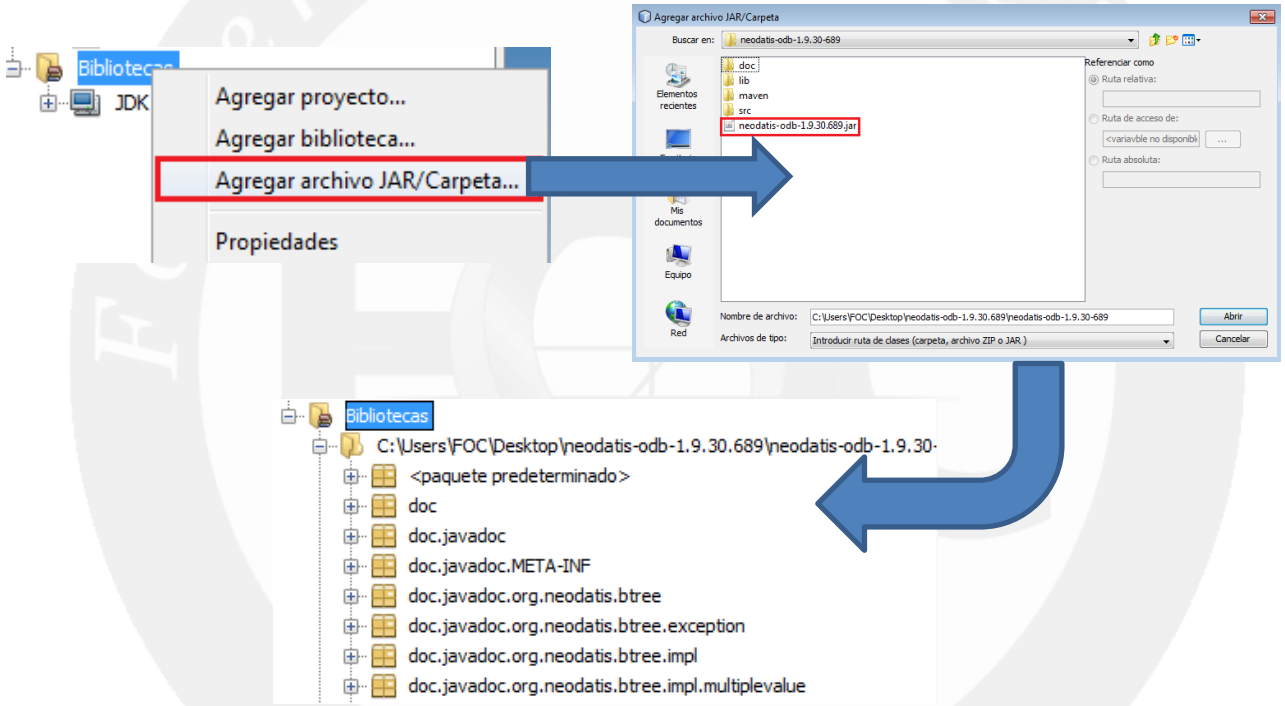
```
public class Departamento {
	private int codDepartamento;
	private String nombre;
	private String localizacion;
	private int numEmpleados;

	public Departamento(int codDepartamento, String nombre, String localizacion, int numEmpleados) {
		this.codDepartamento = codDepartamento;
		this.nombre = nombre;
		this.localizacion = localizacion;
		this.numEmpleados = numEmpleados;
	}
	public int getCodDepartamento() { return codDepartamento; }
	public void setCodDepartamento( int codDepartamento) { this.codDepartamento = codDepartamento; }
```

```
        public String getNombre() { return nombre; }
        public void setNombre(String nombre) {  this.nombre = nombre; }
        public String getLocalizacion() {  return localizacion; }
        public void setLocalizacion(String localizacion) {  this.localizacion = localizacion; }
        public int getNumEmpleados() {  return numEmpleados; }
        public void setNumEmpleados(int numEmpleados) { this.numEmpleados = numEmpleados; }
}
```

Importarmos el jar de NeoDatis a ese proyecto:

- Descargar de la web http://neodatis.wikidot.com/ la librería neodatis.jar.
- Agregar el jar al proyecto como una Biblioteca.



Tras importar **NeoDatis en el proyecto Netbeans,** implementar una clase denominada CrearBDNeoDatis cuyo objetivo sea almacenar unos objetos Departamento en una base de datos orientada a objetos en NeoDatis.

```
import java.io.*;
import org.neodatis.odb.*;
import java.util.*;

public class CrearBDNeoDatis {
   public static void main(String[] args) {

      ODB odb = ODBFactory.open("departamentos.db"); //Abrimos conexión con el fichero de la BD
```

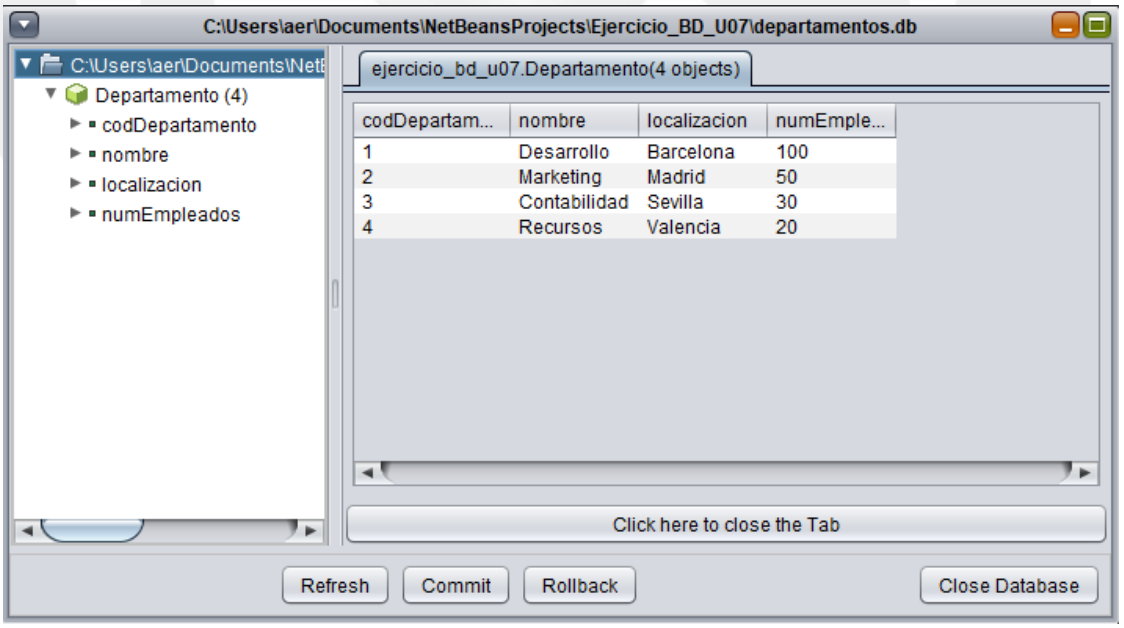
```
        //Creamos 4 objetos Departamento
        Departamento dept1 = new Departamento (1,"Desarrollo","Barcelona",100);
        Departamento dept2 = new Departamento (2,"Marketing","Madrid",50);
        Departamento dept3 = new Departamento (3,"Contabilidad","Sevilla",30);
        Departamento dept4 = new Departamento (4,"Recursos","Valencia",20);

        //Insertamos los objetos en la BD con el método store
        odb.store(dept1);
        odb.store(dept2);
        odb.store(dept3);
        odb.store(dept4);

        odb.close(); //Cerramos la conexion
    }
}
```

Con este código:

- Primero abrimos una conexión con el fichero de la BD, si no estuviera creado se crea en el momento. Esta conexión se almacena en un objeto de tipo ODB.
- Segundo, creamos 4 objetos de tipo Departamento.
- Tercero, insertamos cada uno de los 4 objetos en la base de datos con el método store() del la conexión 'odb'.
- Por último, cerramos la conexión del objeto 'odb'.



Si revisamos el explorador de objetos que nos da Neodatis, vemos que se han insertado correctamente.