

ACCESO A DATOS

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Bases de datos nativas XML
Almacenamiento
Conexiones

Bases de datos XML

Las **bases de datos XML nativas** son bases de datos que almacenan documentos y datos XML de una forma muy **eficiente**. Igual que en las bases de datos relacionales, permiten que los datos se almacenen, consulten, combinen, aseguren, indexen, etc.

Las bases de datos XML nativas **no se basan en tablas**, sino en los llamados **contenedores**. Cada contenedor puede almacenar **grandes cantidades de documentos XML** o datos, que tienen alguna relación entre ellos.

Los contenedores también pueden tener subcontenedores. La gran diferencia con las bases de datos relacionales es que la **estructura** de los datos XML en un contenedor **no tiene por qué ser fija**. Podremos almacenar **distintas unidades de negocio** sin mucha relación dentro del mismo contenedor, la infraestructura lo permite, y podemos hacerlo, otra cosa es que sea más o menos recomendable.

Las bases de datos XML nativas no son consultadas por sentencias SQL, son **consultadas por expresiones Xpath**.

Xpath es un estándar mundial establecido por el “**W3C**” para **navegar** a través de documentos **XML**. Es un lenguaje que se puede utilizar para consultar datos de documentos XML. Las consultas Xpath se pueden utilizar para **escanear el contenido** de documentos XML.

Este lenguaje se basa en una **representación de árbol** del documento XML, y **selecciona nodos** según diferentes **criterios**.

Cuando se consulta una base de datos XML nativa:

1. el usuario generalmente **abre un contenedor**
2. y posteriormente, envía dicha **expresión Xpath contra todos los documentos XML** en la base de datos.
3. A continuación, se devuelve un conjunto de documentos XML.

Ventajas y desventajas de las bases de datos XML con respecto a las relacionales

Ventajas

- Las bases de datos XML nativas son capaces de **almacenar, mantener y consultar mayores cantidades de documentos XML** en comparación a las relacionales.
- A diferencia de las relacionales, **no es necesario configurar tablas**, y por tanto, no se necesita realizar diseños complicados antes de configurar la base de datos.

Una tabla de **base de datos clásica** tiene la desventaja de ser solo **bidimensional**, por lo que la estructura “más profunda” debe implementarse utilizando **claves secundarias**, lo que puede hacer que el diseño de una base de datos sea bastante complicado.

- La **facilidad de importación o exportación** hacia/desde otras aplicaciones con otros formatos.

Nativas XML

Antiguamente estaba muy difundida la consideración que las bases de datos nativas son más lentas en las consultas que las bases de datos relacionales, pero hoy en día ya no es del todo correcta.

Las bases de datos XML actuales son seguramente **tan rápidas como las relacionales**, y además, **más fáciles de mantener y soportar**, con respecto a datos XML se refiere.

Desventajas

- Las bases de datos relacionales están muy bien establecidas, es tecnología ya probada. Las bases de datos XML son un fenómeno algo **más reciente** ya que algunas de las primeras bases de datos XML **no** tienen mucho **más de una década** de antigüedad, por lo que la experiencia aún es limitada.
- **Xpath no es** un lenguaje excesivamente **fácil** de aprender, mientras que SQL está muy extendido. **No muchos desarrolladores y administradores** de bases de datos dominan Xpath. SQL está más relacionado con el lenguaje y la forma humana de ‘pedir’ cualquier cosa en general, por lo que es algo más sencillo de aprender.
- **No todas las aplicaciones gestoras de datos** soportan dicho lenguaje, tendríamos que buscar cuales son los compatibles con las bases de datos mencionadas.

Gestores XML libres y comerciales

A continuación, veremos algunas herramientas gestoras de bases de datos XML nativas. Se presentarán herramientas comerciales de pago y otras de código abierto o libres.

Sistemas gestores de bases de datos XML nativas **de pago**:

- *Excelon XIS*:

Con este gestor, las empresas pueden aprovechar de manera **completa y rentable** la extensibilidad y flexibilidad de XML para **crear, auditar y cambiar** continuamente aplicaciones que almacenan y manipulan **cantidades limitadas** de datos XML.

- *GoXML DB*:

Es una base de datos XML nativa con un motor de consultas de **alto rendimiento**. Los documentos XML **se almacenan directamente**, lo que elimina la necesidad de descomponer datos o configurar cómo se almacenarán los datos.

- *Infonyte DB*:

La tecnología de base de datos de Infonyte se distingue por el soporte nativo para XML, la **independencia** de la plataforma y el **uso moderado de los recursos** del sistema. Por lo tanto, se adapta perfectamente a los requisitos tanto de arquitecturas de componentes como de **dispositivos pequeños**.

En estos mercados emergentes, su liderazgo tecnológico les da una ventaja competitiva sobre las soluciones actuales.

Sistemas gestores de bases de datos XML nativas **libres o de código abierto**:

- *Exist DB*:

Es un proyecto de software de código abierto para **bases de datos NoSQL** construido sobre tecnología XML. Está clasificado como un sistema de base de datos **orientado a documentos** NoSQL y una base de datos XML nativa.

También proporciona **soporte** para documentos:

- **XML**,
- **JSON**,
- **HTML**,
- **binarios**.

A diferencia de la mayoría de los sistemas de administración de bases de datos relacionales, proporciona **Xquery** y **XSLT** como lenguajes de **programación** de aplicaciones y **consultas**.

- *X-Hibe DB:*

Es una poderosa base de datos XML nativa diseñada para desarrolladores de software que requieren **funciones avanzadas de procesamiento y almacenamiento** de datos XML dentro de sus **aplicaciones**.

- *Tamino DB:*

Es una base de datos XML nativa. Si la comparamos con una base de datos relacional, tiene la desventaja de no ser muy popular. Sin embargo, si para trabajar con ella se parte de unos **datos ya estructurados**, se pueden encontrar muchas **ventajas** y posiblemente sea una opción preferente, **frente** a un motor **SQL** tradicional.

Instalación y configuración de EXISTDB

Dedicaremos este apartado para realizar la instalación y configuración necesaria para la base de datos XML nativa ExistDB. Para ello, deberemos de tener en cuenta unos requisitos de sistema recomendados (hay unos mínimos menos restrictivos) que son los siguientes:

- Si instalamos una versión final superior a la versión 3.0 tendremos que tener instalado previamente el **JDK Java 8**.
- Debemos asignar **128 MB de memoria caché**.
- Debemos de tener **1024 MB de memoria RAM** disponibles para poderle asignar al aplicativo.

Una vez descargado el archivo .jar (<https://exist-db.org/exist/apps/homepage/index.html>), se realizará la instalación.

Para ello, hacemos doble clic en el fichero y lanzaremos directamente la aplicación.

En Linux: ejecutamos el comando:

java -jar eXist-db-setup-[version].jar

desde la carpeta del paquete .jar.

En el proceso de instalación se nos preguntara sobre el directorio habitual de nuestra aplicación por si queremos modificarlo. También se nos ofrecerá un directorio de donde cogerá los ficheros de información, podemos dejar el que nos ofrece Exist o cambiarlo al que decidamos.

A continuación, nos encontraremos la configuración de memoria y también la contraseña del usuario Admin. Esta cuenta pertenecerá a la persona que está llevando a cabo la instalación, por lo que determinadas funcionalidades en este gestor, solo podrán ser ejecutadas con dicho usuario. Así que una buena recomendación es **cumplimentar el campo password del usuario Admin**. con una **contraseña fuerte**.

Finalmente, en este punto, es necesario configurar y establecer la **cantidad de memoria RAM** que queremos darle a nuestro proceso Java y a nuestra **memoria Caché**.

El instalador dice:

En Windows y Linux, utilice el elemento del menú de inicio de eXist-db o el icono del escritorio para iniciar eXist-db. En Mac OS, haga doble clic en el icono eXist-db.app dentro de la carpeta en la que está instalado eXist-db.

También puede iniciar eXist-db haciendo doble clic en start.jar o llamando a "java -jar start.jar" en una línea de comando.

Al iniciar eXist-db de una de estas formas se mostrará una pantalla de presentación al iniciar la base de datos. Después del inicio, encontrará un icono de eXist-db en la bandeja del sistema/barra de menú. Haga clic (Windows y Mac OS) o haga clic derecho (Linux) en el icono para obtener un menú emergente con más opciones.

El paquete de instalación

Siguiendo con el proceso de instalación y configuración, el siguiente paso sería el paquete de instalación.

Estos son algunos de los distintos **paquetes que podemos agregar** a la instalación:

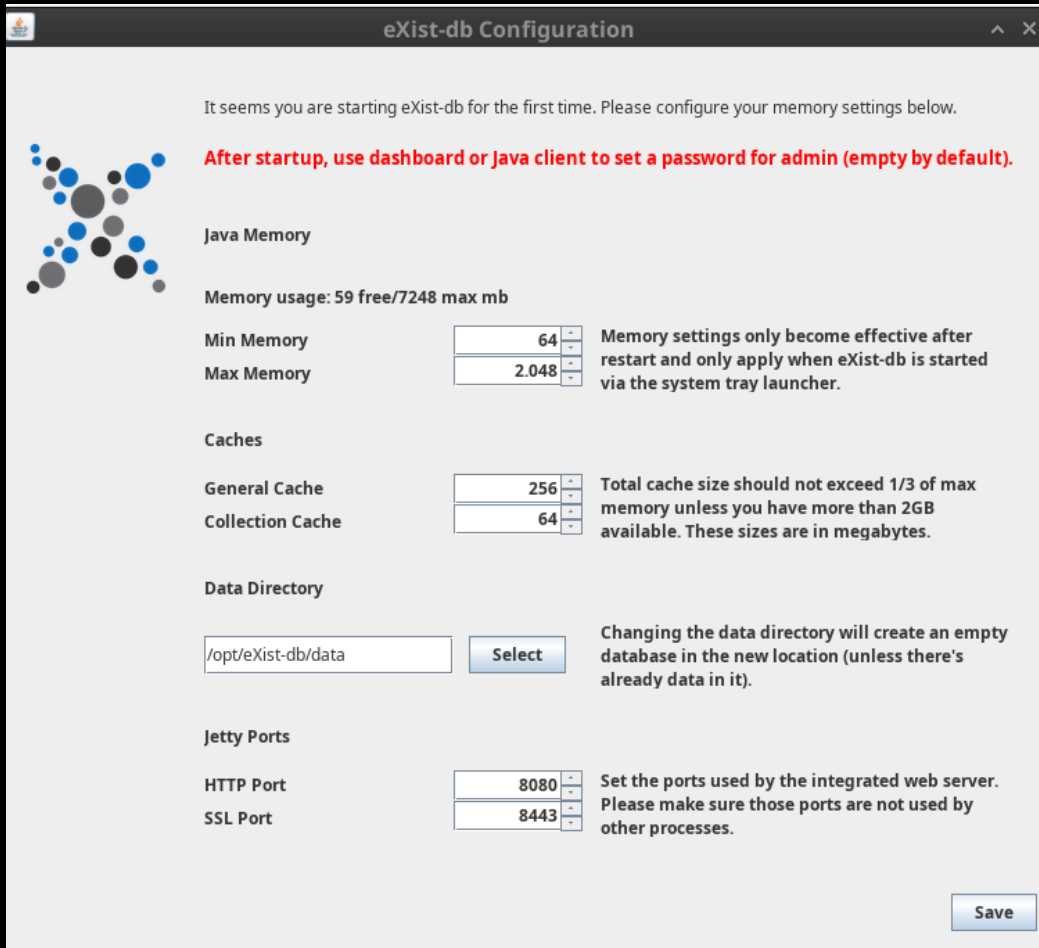
- El paquete “**core**” es requerido para la instalación, ya que es uno de los elementos que nos permitirá **‘correr’ la base de datos**.
- El paquete “**sources**” es opcional; deseleccionaríamos dicho paquete si tuviéramos problemas de espacio, si no, **es preferible instalarlo**.
- El paquete “**apps**” nos permite seleccionar o deseleccionar una serie de **apps que serán instaladas cuando ExistDB arranque por primera vez**. Es recomendable **dejarlas** para dar los primeros pasos.

Una vez hechas las selecciones oportunas, hacemos click en Next, y **finalizaremos la instalación**.

Para lanzar la base de datos en Linux o en Windows, simplemente ejecutaremos el **acceso directo** que nos ha generado la instalación en el menú de inicio (En Linux se nos habrá creado un .desktop en la carpeta de instalación). Aparecerá un **logo inicial** con una imagen Splash mientras las aplicaciones seleccionadas previamente se van instalando. Una vez tenemos nuestra base de datos instalada localizaremos un nuevo icono de bandeja del sistema que nos dará acceso a todas las herramientas que nos ofrece ExistDB y nos permitirá **apagar o reiniciar la base de datos**.

Si queremos **lanzar nuestra base de datos desde línea de comandos** nos dirigiremos al **directorio de instalación** de la base de datos y llamaremos a “launcher.sh” o “launcher.bat” dependiendo si usamos Linux o Windows. (Nota: En Linux es: /usr/local/eXist-db/bin/).

Instalación Exist. Dashboard



The image shows the 'eXist-db Configuration' window. It has a title bar with the eXist logo and window controls. The main content area is white with a light blue header. On the left is a logo consisting of a cluster of blue and grey dots. The text is in a sans-serif font. The configuration options are grouped into sections: Java Memory, Caches, Data Directory, and Jetty Ports. Each section has input fields and a 'Select' button for the directory. A 'Save' button is at the bottom right. A red warning message is at the top right.

It seems you are starting eXist-db for the first time. Please configure your memory settings below.

After startup, use dashboard or java client to set a password for admin (empty by default).

Java Memory

Memory usage: 59 free/7248 max mb

Min Memory	<input type="text" value="64"/>	Memory settings only become effective after restart and only apply when eXist-db is started via the system tray launcher.
Max Memory	<input type="text" value="2.048"/>	

Caches

General Cache	<input type="text" value="256"/>	Total cache size should not exceed 1/3 of max memory unless you have more than 2GB available. These sizes are in megabytes.
Collection Cache	<input type="text" value="64"/>	

Data Directory

Changing the data directory will create an empty database in the new location (unless there's already data in it).

Jetty Ports

HTTP Port	<input type="text" value="8080"/>	Set the ports used by the integrated web server. Please make sure those ports are not used by other processes.
SSL Port	<input type="text" value="8443"/>	

(Nota de apunte: En Linux, nos aseguramos de ejecutar el `launcher.sh` como `sudo` para poder guardar los cambios de configuración java).

Una vez instalado y lanzado podemos navegar hasta el [Dashboard](#) de la base de datos, abriéndolo desde el **icono de sistema** o accediendo directamente a:

<http://localhost:8080/exist/>

Integración ExistDB

Vamos a repasar algunas opciones adicionales que ahora tenemos para traer nuestra base de datos ExistDB a nuestro entorno local. Al ingresar en la URL de la base de datos ExistDB, que es nuestra base de datos XML nativa, encontramos el botón de descarga. Si hacemos clic en él y navegamos por la web interactiva, veremos diferentes formas de descarga o soporte. Nos enfocaremos en las tres primeras opciones que nos resultan interesantes:

- La primera opción, que ya discutimos en la unidad, consiste en descargar directamente la **última release estable**, es decir, la última versión disponible de nuestra base de datos. Solo tenemos que hacer clic en el enlace correspondiente y proceder con la descarga.
- La segunda opción es útil si queremos tener un entorno virtualizado localmente. Si deseamos utilizar Docker, crear un **Docker Compose** y levantar la imagen de esta base de datos en nuestro puerto local, podemos descargarla, realizar un pull y utilizarla con Docker. Al levantar la imagen, podemos establecer conexiones directas desde nuestra aplicación.
- Por último, quisiera mencionar **Maven Artifacts**. Al conectarnos a Maven Artifacts, podemos obtener las últimas versiones de los arquetipos creados por esta compañía para esta base de datos. Podemos revisar los cambios más recientes y, específicamente, al explorar uno de ellos, descargar directamente el **JAR** o revisar el archivo **POM** para ver las versiones de los arquetipos, entre otros detalles. Aquí encontraremos las versiones más recientes utilizadas, lo cual puede ser de gran utilidad si queremos incorporar la dependencia Maven directamente en nuestra aplicación para utilizarla con Java.

Quisiera destacar que estas tres opciones también están disponibles para la base de datos, y les recomiendo revisarlas para determinar si son útiles para integrar en sus aplicaciones.

Estrategias de almacenamiento

Las bases de datos nativas XML pueden ser clasificadas **dependiendo del tipo de almacenamiento que vayan a utilizar**, de este modo, a continuación, veremos los diferentes **modos de almacenamiento** de los que disponemos en una base de datos nativa XML:

- Almacenamiento basado en **texto**.
- Almacenamiento basado en **modelo**.
- Almacenar en **local**.

Almacenamiento **basado en texto**:

Esta modalidad consiste en **almacenar el documento completo** en base de datos, y dotar a la misma, de algún tipo de **funcionalidad** para que se pueda **acceder fácilmente** a él.

En este tipo de almacenamiento suele:

- realizarse la **compresión** de ficheros.
- añadir algunos **índices** específicos para aumentar la eficiencia.

Las **posibilidades** para esto serían dos:

- Almacenar el fichero binario de tipo **BLOB**:
 - en un **sistema relacional**.
 - en un soporte o **almacén orientado a dicha operación con**:
 - **índices**,
 - soporte de transacciones,
 - etc.

Almacenamiento **basado en el modelo**:

Para este caso, se utilizaría un **modelo de datos lógico** como por ejemplo puede ser **DOM**, para definir la **estructura y la jerarquía** de los documentos XML que se vayan a almacenar.

Se guardaría el modelo del documento en un **almacén definido** previamente.

Para esto, tenemos algunas **posibilidades** como:

- Traducir desde **DOM** a **tablas** de una base de datos convencional relacional.
- Traducir el objeto **DOM** a **objetos** en una base de datos orientada a objetos.
- Usar un **almacén de datos** creado específicamente para esta utilidad.

Almacenar una forma modificada del documento **en local**:

Podremos usar este tipo de almacenamiento, cuando la **cantidad** de ficheros a almacenar **no** sea muy **elevada**, y **no** se realicen **numerosas actualizaciones** ni **transferencias** de los mismos. Realmente consiste en almacenar una **forma modificada** del fichero XML base completo, directamente en **sistema de archivos**.

Evidentemente tiene diferentes **limitaciones** como **escalabilidad, flexibilidad, recuperación y seguridad**.

Establecimiento y cierre de conexiones

A continuación, realizaremos una **conexión** con la base de datos ExistDB como ejemplo.

Para ello deberemos de **tener en cuenta** ciertos aspectos:

- `Org.xmlldb.api`: Nos enriquecerá el código con las **Interfaces** y **DatabaseManager**.
- `Org.xmlldb.api.base`: Con esta librería accederemos a:
 - las **colecciones**,
 - los **objetos** Database,
 - **Resource**,
 - **ResourceIterator**,
 - **ResourceSet**,
 - y algunos más.
- `Org.xmlldb.api.modules`: Accederemos a:
 - los servicios de **transacciones**,
 - **XMLResource**,
 - servicios de XpathQueryService,
 - y otros varios relacionados.

Supondremos que tenemos en nuestra **aplicación Java ya desarrollada**, una **clase dedicada al acceso** a capa de datos, donde dicha aplicación:

- **accederá**,
- establecerá **conexión**,
- realizará algunas **consultas**,
- y obtendrá **resultados**.

Para ello iremos **paso a paso**:

1. Primeramente, indicaremos el `driver` a cargar:

Driver

```
String driver = "org.exist.xmlldb.DatabaseImpl";
```

2. Con esta línea cargaremos el driver instanciado:

Carga Driver

```
Class clase = Class.forName(driver);
```

3. Acto seguido, instanciamos la base de datos:

Instancia

```
Database database = (Database) clase.newInstance();
```

4. Con esta línea registraremos la base de datos:

Registro

```
DatabaseManager.registerDatabase(database);
```

5. A continuación, mostraremos algunas líneas de código para acceder a la colección.

Con ellas, **preparamos el código** para la colección que más tarde consultaremos:

Acceso a la colección

```
Collection coleccion =  
DatabaseManager.getCollection(  
    "xmldb:exist://localhost:8080/exist/xmlrpc/db/",  
    "usuario",  
    "contraseña");  
XPathQueryService service =  
(XPathQueryService) coleccion.getService(  
    "XPathQueryService", "1.0");
```

6. Por último, realizaremos las **consultas** a base de datos.

Los resultados obtenidos se almacenarán en la variable “**resultado**” que posteriormente tendremos que **iterar**, y realizar su procesado:

ResultSet

```
ResourceSet resultado = service.query(  
    "for $b in doc('prueba.xml')//a return $b");
```

7. Una vez hechas las consultas tendremos en cuenta en qué consisten los **conceptos** de:

- **Indexación**: Permiten la **creación de índices** para **acelerar** las **consultas** frecuentes.
- **Identificador único**: Cada documento XML está asociado con un identificador único, a través del cual se puede identificar **en el repositorio**.

Agregar, modificar y eliminar recursos

También es posible **agregar** nuevos **recursos XML y no XML** a la colección (objeto de "Collection").

Para esto, se necesitan las siguientes clases y métodos:

La **clase Collection** representa una **colección de recursos** (resource) almacenados en una base de datos XML. Tiene **métodos** como los siguientes:

- **storeResource** (resource res):

Es el método más relevante para **agregar nuevos recursos** en esta clase. Almacena el **recurso res** proporcionado por el parámetro en la colección.

- **removeResource** (recurso res): **Elimina el recurso** res pasado al recurso a través de parámetros de la colección.
- **listResources** (): Útil para crear y eliminar nuevos recursos; devuelve una **matriz de cadenas** que contiene los ds de todos los recursos que tiene la colección;
- **getResourceCount**() obtiene la cantidad de recursos almacenados en la colección;
- **createResource** (java.lang.String id, java.lang.String type), crea un nuevo recurso vacío en la colección, cuyo **ID y tipo** son pasados por **parámetros**.

Si ya se comprende el proceso de creación de un nuevo recurso, se puede definir el proceso de eliminación más fácilmente. Para esto, la clase Collection vuelve a intervenir. La forma principal de eliminar recursos es: **removeResource** (resource res), que elimina resource res de la colección.

Creación y borrado de colecciones

A continuación, veremos un **ejemplo de agregación y borrado** de colecciones en código:

Agregación y borrado de colecciones

```
public Collection anadirColeccion (
    Collection contexto,
    String newColec) throws ExcepcionGestorBD {
    Collection newCollection = null;
    try {
        //Creamos un nuevo servicio.
        CollectionManagementService mgtService = (
            CollectionManagementService) contexto.getService(
                "CollectionManagementService", "1.0");
        //Creamos una nueva collection con codificación UTF8
        newCollection = mgtService.createCollection(
            new String(UTF(.encode(newColec))))
    } catch (XMLDBException e) {
        throw new ExceptionGestorDB (
            "Error agregando coleccion: " + e.getMesagget());
    }
    return newCollection;
}
```

Si lo que deseamos es borrar dicha colección simplemente tendremos que llamar al método: `removeCollection(String nombre)` de la clase “CollectionManagementService”

Modificación de contenidos XML

Estudiaremos distintas acciones para modificar el contenido de un árbol DOM.

- Modificaremos el valor del texto asociado a una etiqueta.
- Podremos modificar el valor de un atributo asociado a una etiqueta.

Modificación del **valor del texto** asociado a una etiqueta:

Valor texto

```
Node nodo = raiz.getElementsByTagName("nombreDelTag").item(0);  
nodo.setTextContent("Otro");
```

- primera línea: obteniendo un nodo.
- segunda línea: con el método “setTextContent” estaremos estableciendo y modificando el valor del texto asociado a dicha etiqueta.

Modificación del **valor de un atributo** asociado a una etiqueta:

Imaginemos que poseemos un objeto de la clase “**Node**” que **representa una etiqueta**. La idea es **convertir** dicho objeto a un objeto de la clase “**Element**” para utilizar el método “**setAttribute**(String)”.

Para ello tendremos que realizar un **casting de “Element”** sobre el nodo deseado. A continuación, podemos ver el código:

Valor atributo

```
Element elemento = doc.getDocumentElement();  
elemento.setAttribute("nombre", "valor");
```

Tal y como podemos observar en el código anterior, extraemos el elemento y trabajamos con el método “setAttribute”, indicándole el nombre de la etiqueta, y a continuación, introduciendo el valor deseado. Para obtener dicho contenido simplemente usaríamos:

```
System.out.println(elemento.getAttribute("nombre"));
```

Transacciones y excepciones

Transacciones XML

En algunos de los ejemplos expuestos en el tema, existe el concepto de transacción: un conjunto de declaraciones ejecutadas que **forman inseparablemente una unidad** de trabajo, de modo que todas se ejecutan o no se ejecutan.

Al administrar las transacciones, el administrador XML proporciona **acceso simultáneo a los datos** almacenados, mantiene la **integridad y seguridad** de los datos, y proporciona un **mecanismo de recuperación de fallos** en la base de datos.

Exist-db admite transacciones **compatibles con ACID**:

- **Atomicidad**. Se deben completar **todas** las operaciones de la transacción, **o** no realizar **ninguna** operación, no se puede dejar a la mitad.
- **Consistencia**. La **transacción finaliza** solo cuando la base de datos permanece en un **estado coherente**.
- **Aislamiento**. Las transacciones sobre una misma información deben ser **independientes** para que no interfieran con sus operaciones, y no generen ningún tipo de error.
- **Durabilidad**. Al final de la transacción, el **resultado** de la misma **se conservará** y no se podrá deshacer incluso si el sistema falla.

Tratamiento excepciones

La clase `XMLDBException` captura todos los errores que ocurren cuando se usa XML (DB para procesar la base de datos).

Esta excepción contendría dos **códigos de error**:

- Un código de error **especificado por cada sistema** de procesamiento XML local (fabricante-proveedor).
- Un código de error **definido en la clase ErrorCodes**.

Si el error que ocurrió en un momento dado es **parte del sistema** de administración XML, `ErrorCode` debe tener un valor de `errorCodes`, **VENDOR_ERROR**.

La clase `ErrorCodes` define los diferentes **códigos de error XML**: DB utilizado por el atributo `errorCodes` de la **excepción XMLDBException**.

Descarga e instalación de ExistDB

Se requiere instalar una base de datos XML nativa para realizar las distintas conexiones. Al menos descarga e instalación.

Para realizar la instalación de la base de datos ExistDb, en primer lugar, nos dirigiremos a la web oficial de la herramienta, en este caso:

<http://exist-db.org/exist/apps/homepage/index.html>

Como podemos ver tenemos diferentes **formas de integración de la base de datos** en nuestro sistema, las más interesantes:

- Descargar **directamente** la **última versión** de base de datos e instalarla.
- Descargar una **imagen de docker** y montar dicha imagen para usar la herramienta.
- **Dependencia maven**, que nos ayudará a **inyectar la dependencia** de dicha base de datos, en el proyecto en el que estemos trabajando.

Elegiremos para este caso la primera opción y nos descargaremos la última versión “release” estable de la misma.

Realmente lo que nos estamos descargando es un fichero con **extensión .jar**. Simplemente ejecutaremos dicho fichero .jar para realizar la **instalación** tanto para sistemas Unix como para Windows. Con este pequeño proceso nos bastará para tener instalada nuestra base de datos. Más tarde el usuario tendrá que realizar el **arranque de base de datos** y la **integración** con la aplicación.

ExistDB config file

Actualmente tenemos una base de datos XML nativa asociada a una aplicación Java de registro de clientes.

Se requiere cambiar algunos aspectos de configuración de la base de datos instalada Exist DB:

- Cambiar el tamaño de **caché a 512MB**.
- Cambiar la **ubicación de los ficheros a usuario/Downloads**.

Una vez instalada nuestra base de datos nativa XML nos dispondremos a realizar varias configuraciones:

En primer lugar, comentar, que el **fichero principal de configuración** de dicha base de datos es el fichero "**conf.xml**", como suele ser habitual en numerosos sistemas gestores de base de datos.

Dicho fichero se encuentra alojado en el **directorio raíz** de instalación de nuestra base de datos. (Nota de apunte: en mi instalación está en: /etc/ del directorio de instalación de la base de datos).

A continuación, modificaremos los 2 aspectos que se han requerido y veremos algunos atributos más de configuración del mismo.

Atributos

```
<db-connection
  cacheSize="512M"
  collectionCache="24M"
  database="native"
  files=" ../usuario/Downloads"
  pageSize="4096"
  nodesBuffer="-1">
  <pool
    max="15"
    min="1"
    sync-period="240000"
    wait-before-shutdown="60000" />
  <recovery
    enabled="yes"
```

```
    sync-on-commit="no"
    group-commit="no"
    size="100M"
    journal-dir=" ../data" />
    <watchdog output-size-limit="10000" query-timeout="-1" />
    <default-permissions collection="0775" resource="0775" />
</db-connection>
```

Como podemos comprobar, en este caso simplemente hemos realizado el cambio que se nos requería:

- Atributo “**cacheSize**” hemos seteado a **512 Megabytes**.
- Atributo “**files**” para indicar la **ubicación de los ficheros**.

Como podemos observar en este **bloque de “db-connection”** del **fichero conf.xml**, tenemos numerosos atributos más, sobre los que podremos aprender su funcionalidad, en la página oficial de la base de datos Exist:

<https://exist-db.org/>

Ejemplo de preparación de acceso a base de datos XML

Se requiere instalar una base de datos XML nativa, y dejar preparada una **capa de acceso a datos** con las líneas esenciales, al menos, para realizar conexión con base de datos.

En primer lugar, descargaremos la última versión estable de la aplicación ExistDB. Para ello, nos dirigiremos a la web oficial:

<http://exist-db.org/exist/apps/homepage/index.html>

Ejecutaremos el fichero .jar con doble clic y seguiremos la instalación con los diferentes pasos que hemos aprendido en las páginas anteriores.

Una vez instalado, nos dispondremos a realizar una nueva **paquetería en nuestra aplicación** donde contendrá la **clase de acceso** a datos. Normalmente podremos crear una clase con nomenclatura parecida a:

AcessDataDao.java

dentro de la estructura: "ApplicationName.com.Java.Dao".

Una vez creada la clase añadiremos las siguientes líneas de código para realizar la **conexión con la base de datos**:

```
String driver = "org.exist.xmlldb.DatabaseImpl";  
Class clase = Class.forName(driver);  
Database database = (Database) clase.newInstance();  
DatabaseManager.registerDatabase(database);
```

Con estas líneas de código, estaremos:

- **dando de alta nuestro driver** para la base de datos XML nativa,
- y al mismo tiempo, estaremos **realizando el registro**.

Una vez realizadas estas simples operaciones, a continuación, podríamos preparar diferentes **colecciones** para obtener **resultados realizando consultas**.

<http://exist-db.org/exist/apps/homepage/index.html>