

UD6.- Bases de datos xml.



Caso práctico

A la empresa BK Programación no dejan de llegar nuevos proyectos, está claro que sus productos finales convencen a sus clientes y éstos están bastante contentos con ellos.

Ana está orgullosa de haber participado y aportado su granito de arena en el desarrollo y mantenimiento de diferentes aplicaciones, algunas bastante complejas.

Este fin de semana se ha tomado un respiro. Sabe que el lunes comienza con algo nuevo. Debe ayudar a **María** y **Juan** en la tarea de compatibilizar diferentes formatos de datos que intercambian varias academias de Inglés de una misma firma.

Las academias gestionan la información relativa a sus cursos, alumnos, profesores, distribuidores de material, libros de lectura, artículos, apuntes, exámenes, etc. con programas diferentes y basados también en sistemas de bases de datos diferentes.

Hasta ahora, las academias han hecho uso de XML para intercambiar los datos entre sus sistemas, pero cada vez surge algún problema nuevo, sobre todo con el almacenamiento, consulta y recuperación de ciertos documentos de texto. Por tanto, han decidido acudir a BK Programación para buscar la mejor solución. No les importa empezar desde cero y, si es necesario, utilizar una base de datos nativa XML, tal y como les ha sugerido **Ada** en su primera entrevista.

Ana apura la tarde del domingo pues esa misma noche va a comenzar su nueva aventura, ¡repasar XML y las tecnologías relacionadas! **Ana**, es muy profesional y no deja para mañana lo que puede hacer hoy.



Materiales formativos de FP Online propiedad del Ministerio de Educación y Formación Profesional.

[Aviso Legal](#)

1.- Introduccion.

Atendiendo al nivel de estructuración, podemos decir que existen tres tipos de datos:



- ✔ **Datos estructurados.** Son los que tienen un formato estricto. Toda la información recogida se ajusta al mismo formato, como por ejemplo: los datos tabulados en filas y columnas de una tabla.
- ✔ **Datos desestructurados.** No tienen ninguna estructura, como un documento de texto o un archivo de vídeo.
- ✔ **Datos semi-estructurados.** Tienen cierta estructura, pero no toda la información recogida tiene la misma forma, y además puede ir variando de manera dinámica.

Las bases de datos tradicionales, como las Bases de Datos Relacionales (👉 BDR) son apropiadas para almacenar datos estructurados, pero la cuestión es que, en la actualidad, son muchas las situaciones en las que interesa almacenar grandes volúmenes de datos no estructurados, e incluso integrarlos con datos estructurados. Pero ¿cómo hacerlo? Aquí es donde entra en juego XML (eXtensible Markup Language).

XML define un conjunto de reglas semánticas que permiten la organización de información de distintas maneras. Es un estándar definido por el W3C y ofrece muchas ventajas, entre ellas:

- ✔ Es un lenguaje **bien formado**. No puede haber etiquetas sin finalizar.
- ✔ **Extensible**. Permite ampliar el lenguaje mediante nuevas etiquetas y la definición de lenguajes nuevos.
- ✔ **Fácil de leer**.
- ✔ **Autodescriptivo**. La estructura de la información de alguna manera está definida dentro del mismo documento
- ✔ **Intercambiable**. Portable entre distintas arquitecturas.
- ✔ Para su lectura e interpretación es necesario un 👉 parser, y hay productos y versiones libres.

Debido fundamentalmente a estas ventajas y a su sencillez, **el estándar XML ha sido ampliamente aceptado y adoptado para el almacenamiento e intercambio de información**, y como consecuencia de este uso se ha creado la necesidad de almacenar dicha información.

Y ¿cómo se almacena la información en formato XML de cara a su recuperación y consulta? Existen varias aproximaciones para organizar y almacenar información XML que fundamentalmente van a depender de los diferentes archivos o documentos XML que nos podemos encontrar (centrados en datos y centrados en texto).

Pero en definitiva, lo que se busca es una estrategia que permita almacenar y recuperar datos poco estructurados con la eficiencia de las Bases de Datos convencionales, y es por ello que surgen las Bases de Datos XML.



Citas para pensar

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

Claude Bernard



Para saber más

En el siguiente enlace puedes consultar diferentes tecnologías basadas en XML.



[Tecnologías XML](#)

1.1.- Documentos XML centrados en datos y en texto.

Como te hemos comentado antes, **los tipos de documentos XML que nos podemos encontrar** son:

- ✓ **Documentos centrados en datos**, con las siguientes características:
 - Muchos elementos de datos de pequeño tamaño.
 - Con estructura regular y bien definida.
 - Datos muy estructurados o semi-estructurados.
 - Dirigidos a utilización automática (por máquinas).
 - Ejemplos: Facturas, Pedidos Ficha de alumno.
- ✓ **Documentos centrados en texto, contenido o documentos** con las siguientes características:
 - Pocos elementos.
 - Con grandes cantidades de texto.
 - Con estructuras impredecibles en tamaño y contenido.
 - Datos poco estructurados.
 - Orientados a ser interpretadas por humanos.
 - Enfocados a sistemas documentales y de gestión de contenidos.
 - Ejemplos: Libros, Informes, Memorias, Artículos bibliográficos.

En la siguiente imagen ampliable puedes apreciar las características indicadas para estos dos tipos de documentos.



En el siguiente enlace tienes un ejemplo de cada tipo de documento.

 [Ejemplo de documento centrado en datos y documento centrado en texto.](#) (0.02 MB)



Para saber más

Te proponemos el siguiente enlace para repasar el significado de documento XML bien formado.

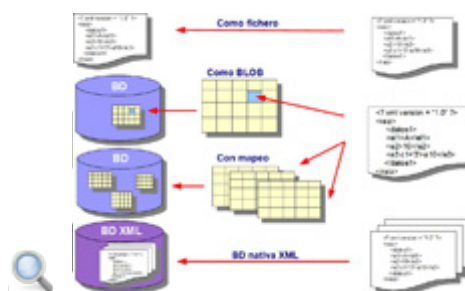
 [Documento XML bien formado](#)

1.2.- Opciones de almacenamiento.

Para almacenar documentos XML tenemos las siguientes opciones:

- ✓ **Almacenamiento directo del fichero.** Es una opción pobre ya que las operaciones que podemos hacer sobre ellos son limitadas, las que proporcione el sistema de archivos.
- ✓ **Almacenar el documento en una base de datos existente** (base de datos relacional, orientada a objetos u objeto-relacional). En este caso existen las siguientes posibilidades:
 - **Directamente como una columna tipo binario grande (📄 BLOB) dentro de una tabla.** En este caso se almacena el documento intacto. Es una buena estrategia si el documento XML contiene contenido estático, que solo será modificado cuando se reemplaza el documento completo. Almacenar el documento completo en formato de texto es fácil de implementar dado que no se necesita 📄 mapeo o traducción, pero limita las consultas y búsquedas de contenido.
 - **Mediante mapeo basado en tablas, o basado en objetos.** En ambos casos hay que realizar una transformación para ajustarlo a la estructura de la BD antes de almacenarlo, y normalmente esto conlleva prescindir de cierta información, que supondrá que el documento y su formato no se almacena y recupera de manera intacta. Además, en el caso de documentos centrados en texto, no podemos controlar todas las posibles estructuras del documento, y por tanto realizar un mapeo sobre la base de datos será prácticamente imposible.
- ✓ **Almacenar el documento en una base de datos nativa XML.** El documento, tanto si es centrado en datos como en texto, se almacena y recupera de forma intacta. Como veremos, será la mejor opción, sobre todo si el documento está basado en texto.

La siguiente imagen ampliable ilustra las anteriores opciones de almacenamiento:



En la actualidad, cada vez más, las BD convencionales ofrecen posibilidades de almacenamiento XML, por lo que se habla de BD XML-compatible o BD XML-enabled. Por tanto, podemos hablar de dos **tipos de Sistemas de Bases de Datos que soportan documentos XML**:

- ✓ **BD XML-compatible:** desglosan un documento XML en su correspondiente modelo relacional o de objetos.
- ✓ **BD XML Nativas:** respetan la estructura del documento, permiten hacer consultas sobre dicha estructura y recuperan el documento tal y como fue insertado originalmente

Las principales diferencias entre ambos tipos de BD XML son las siguientes:

- ✓ Una BD XML nativa proporciona un modelo de datos propio, mientras que un BD XML-compatible tiene ya un modelo de datos y añade una capa de software que permite de alguna manera almacenar documentos XML y recuperar los datos generando nuevos documentos XML.
- ✓ Una BD XML nativa debe manejar todos los tipos de documentos posibles, mientras que una BD XML-compatible solo puede manejar y almacenar los documentos que encajan dentro del modelo definido para ellos.



Para saber más

No dejes de visitar la página de Rounald Bourret, uno de los más importantes investigadores y consultores sobre BD XML. Aunque en inglés, en esta página encontrarás mucha información sobre el desarrollo de las bases de datos XML y sus especificaciones.

[Página de Rounald Bourret sobre BD XML](#)

2.- Bases de Datos Nativas XML.



Caso práctico

Esta mañana, **María** ha estado hablando con una amiga del gremio. Su amiga lleva varios años trabajando en proyectos con tecnologías XML y además ha asistido personalmente a algunas conferencias del mismísimo Rounald Bourret, por lo que está más que al tanto de lo último en estas tecnologías.



María le ha pedido consejo sobre algunas de las bases de datos nativas XML, actualmente en el mercado.


Las **Bases de Datos nativas XML NXD** (Native XML Database) aunque existen desde hace años, en la actualidad siguen evolucionando, por lo que existen ciertas diferencias entre los productos de este tipo que podemos encontrar en el mercado. Lo que si está bastante claro, es que **una NXD o BD XML debe cumplir las siguientes propiedades:**

- ✔ **Define un modelo lógico de datos XML**, estableciendo los elementos que son significativos, de forma que el documento XML se pueda almacenar y recuperar de manera intacta, esto es, con todos sus componentes. Por tanto, el modelo, como mínimo, debe tener en cuenta: los elementos, atributos, texto, secciones CDATA (Character data), y preservar el orden en el documento.
- ✔ **El documento XML es la unidad lógica de almacenamiento**, esto es, la unidad mínima de almacenamiento.
- ✔ **No tiene ningún modelo de almacenamiento físico subyacente concreto.** Pueden ser construidas sobre bases de datos relacionales, jerárquicas, orientadas a objetos o bien mediante formatos de almacenamiento propietarios.

Lo que realmente cambia en estas BD respecto a las convencionales, es el formato que soportan, ya que, están especializadas en almacenar documentos XML, almacenarlos y recuperarlos con todos sus componentes.

Otras características de las BD Nativas XML son las siguientes:

- ✔ **Documentos y colecciones.** Los documentos se pueden agrupar en unidades denominadas colecciones.
- ✔ **Indexación.** Permiten la creación de índices para acelerar las consultas realizadas frecuentemente.

- ✔ **Identificador único.** A cada documento XML se le asocia un identificador único, por el que será reconocido dentro del repositorio.
- ✔ **Consultas.** Soportan uno o más lenguajes de consulta. Entre ellos, uno de los más populares es XQuery (XML Query)
- ✔ **Actualizaciones.** Poseen diferentes estrategias para actualizar documentos, entre ellas la más popular es Update XQuery.
- ✔ **Validación.** No siempre se realiza validación de documentos, esto es, no necesitan un **DTD** o un XML Schema (W3C) para almacenar documentos, basta con que los documentos sean XML bien formados.
- ✔ Soportan  transacciones, accesos concurrentes, control de accesos y backup como cualquier otro sistema de bases de datos.

Y ¿en qué situaciones puede resultar imprescindible su uso? Fundamentalmente en las siguientes situaciones:

- ✔ Existencia de documentos con anidamientos profundos.
- ✔ Importancia de preservar la integridad de los documentos.
- ✔ Frecuentes búsquedas de contenido.



Autoevaluación

Señala si la siguiente afirmación es verdadera o falsa.

Una BD XML nativa solo puede almacenar documentos XML válidos.

☐ Verdadero ☐ Falso

Falso

Es falso, en la mayoría de las BDs XML nativas basta con que los documentos sean XML bien formados.



Para saber más

Consulta el siguiente enlace para recordar o bien familiarizarte con lo que son los DTDs y los esquemas XML.



[Los DTDs y esquemas XML](#)

2.1.- Estrategias de almacenamiento.

Podemos diferenciar o clasificar las BD XML nativas en función del tipo de almacenamiento que utilicen, que puede ser:



- ✓ Almacenamiento **basado en texto**.
- ✓ Almacenamiento **basado en el modelo**.
- ✓ Soluciones desarrolladas **específicamente** para la gestión de documentos XML.

¿En qué consiste el almacenamiento basado en texto y basado en modelo?

- ✓ El **almacenamiento basado en texto** consiste en almacenar el documento XML entero en forma de texto (fichero de texto), y proporcionar alguna funcionalidad de base de datos para acceder a él.

Se suelen aplicar técnicas de compresión para reducir el espacio de almacenamiento, utilizar índices adicionales para mejorar el acceso a la información, y se pueden definir sobre BD tradicionales o sistemas de ficheros. Básicamente existen dos posibilidades:

- ✦ Almacenar el documento como un binario largo (BLOB) en una base de datos relacional, o mediante un fichero, y proporcionar algunos índices sobre el documento que aceleren el acceso a la información.
- ✦ Almacenar el documento en un almacén adecuado con índices, soporte para transacciones, etc.
- ✓ El **almacenamiento basado en modelo** consiste en definir un modelo de datos lógico, como DOM, para la estructura jerárquica de los documentos XML y almacenar el modelo binario del documento en un almacén existente o bien específico. En esta caso las posibilidades que existen son:
 - ✦ Traducir el DOM a tablas relacionales como elementos, atributos, entidades, etc.
 - ✦ Traducir el DOM a objetos en una BDOO.
 - ✦ Utilizar un almacén creado especialmente para esta finalidad

A continuación, te indicamos algunos ejemplos de BD XML nativas clasificadas según su sistema de almacenamiento:

- ✓ **Sistema propietario:** XIndice, Virtuoso, Tamino XML Server.
- ✓ **Sistema relacional:** eXist, DBCOM, XDB
- ✓ **Sistema orientado a objetos:** Ozone, MindSuite XDB.


¿Y qué ventajas proporcionan las BD XML nativas sobre otros sistemas de almacenamiento? Las **principales ventajas** son las siguientes:

- ✓ No necesitan mapeos adicionales.
- ✓ Conservan la integridad de los documentos.
- ✓ Permiten almacenar documentos heterogéneos.



Para saber más

Para refrescar tus conocimientos sobre el modelo de objetos de documentos DOM te recomendamos que visites el siguiente enlace. También obtendrás información sobre el parser SAX (Simple API for XML):

 [Los modelos DOM](#)



Autoevaluación

Señala si la siguiente afirmación es verdadera o falsa.

El almacenamiento basado en modelo supone almacenar el documento como texto en un almacén adecuado con índices y que soporte transacciones.

☐ Verdadero ☐ Falso

Falso

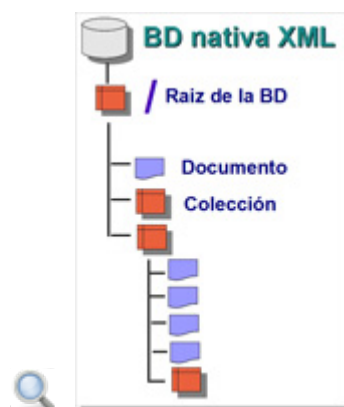
Es falso, esta posibilidad de almacenamiento la contempla la estrategia de almacenamiento basado en texto.

2.2.- Colecciones y documentos.

En general, una BD XML tiene una estructura jerárquica organizada en colecciones y documentos XML. La estructura jerárquica comienza con un **nodo raíz** ('/'), del que parten colecciones y documentos.

Una colección:

- ✓ Es un conjunto de documentos agrupados, normalmente, en función de la información que contienen.
- ✓ Puede contener otras colecciones.



Un documento:

- ✓ Información XML.
- ✓ Información de otro tipo y entonces se le denomina non-XML data (Datos no-XML).

Comparando con una base de datos relacional o un sistema de archivos:

- ✓ Las colecciones juegan en las bases de datos nativas el papel de las tablas en las DB relacionales, o de un directorio en un sistemas de archivos.
- ✓ Los documentos juegan el papel de las filas de una tabla de una BD relacional o un fichero en un sistema de archivos.

Aunque depende de cada implementación, en muchos casos:

- ✓ Cada colección puede tener más de un DOCTYPE asociado.
- ✓ El elemento raíz del documento XML define a que DOCTYPE estará asociado el documento, en caso de no poseer ninguno, éste se crea dinámicamente. Esto posibilita el almacenamiento de documentos sin formato definido.
- ✓ La colección también puede tener asociado un Schema con información tanto física como lógica de la colección. La parte lógica define las relaciones y propiedades de los documentos XML y la física contiene información sobre el almacenamiento e indexación de los mismos.
- ✓ También se pueden almacenar documentos no-XML, para estos existe un DOCTYPE especial llamado non-XML.



Citas para pensar

A veces sentimos que lo que hacemos es tan solo una gota en el mar, pero el mar sería menos si le faltara una gota.

Madre Teresa de Calcuta



Autoevaluación

Señala si la siguiente afirmación es verdadera o falsa.

Una colección solo puede almacenar documentos XML.

☐ Verdadero ☐ Falso

Falso




Es falso, también puede almacenar colecciones y documentos non-XML.

2.3.- Gestores nativos XML comerciales y libres

Te indicamos a continuación algunos ejemplos de gestores nativos XML clasificados según sean de código libre o código propietario. Los comerciales, generalmente ofrecen versiones de prueba con limitaciones de tiempo de uso y de funcionalidades. Los libres, a veces también ofrecen dualidad de licencia, comercial y libre.



Algunos de estos gestores nativos XML son los siguientes:


- ✓ **Gestores nativos XML comerciales** o de código propietario son:
 - ✓ **TaminoXML Server.** Es un gestor nativo XML de la empresa SoftwareAG. Es un producto comercial de alto rendimiento y disponibilidad, además de ser uno de los primeros SGBD XML nativos disponibles. Algunas de sus características son las siguientes:
 - Los documentos se almacenan en una base de datos propia y no se transforman en otro modelo.
 - Existe un espacio separado para documentos y para índices.
 - Soporta el lenguaje de consultas XQuery y APIs para Java, C, y .NET, entre otras.
 - ✓ **TEXTML** de Isiasoft. Los documentos se almacenan en su formato nativo, sin ser mapeados.
 - Permite almacenar documentos sin DTD o esquema.
 - Proporciona índices de acuerdo a la propia estructura del documentos.
 - Permite la utilización de varios índices al mismo tiempo.
 - Incluye API para Java, WebDAV, OLE DB y .NET.
- ✓ **Gestores nativos XML libres** o de código abierto son:
 - ✓ **eXist.** Utiliza un sistema de almacenamiento propio (árboles B + y archivos paginados). Se puede ejecutar como un servidor de base de datos independiente, como una biblioteca de Java embebida, o en el motor `servlet` de una aplicación Web.
 - Los documentos se almacenan en una jerarquía de colecciones.
 - Permite almacenar documentos sin DTD o esquema.
 - Soporta el lenguaje de consulta XQuery y sus extensiones, como XUpdate, así como API para Java.
 - ✓ **Mongo DB.** Es un  SGBDNoSQL orientado a documentos de código abierto y escrito en C++, que en lugar de guardar los datos en tablas lo hace en estructuras de datos  BSON (similar a  JSON) con un esquema dinámico.




Para saber más

Puedes consultar una relación extensa de bases de datos nativas XML,

comerciales y libres, en el siguiente enlace:

 [Relación de BD XML nativas, comerciales y libres](#)

Si no sabes lo que es un árbol B+, consulta este enlace para conocer estas estructuras de datos:

 [Información sobre árboles B+](#)



Citas para pensar

El éxito no se logra sólo con cualidades especiales. Es sobre todo un trabajo de constancia, de método y de organización.

J.P. Sergent

3.- Base de datos eXist.



Caso práctico

Juan y María están muy satisfechos con **Ana**, realmente les está ayudando bastante con sus conocimientos de XML. Además, justo el gestor XML nativo que propuso **Ana** desde el principio, es uno de los que a **María** le sugirió su amiga. **Juan** le pregunta a **Ana** —¿Has trabajado antes con el gestor eXist?



Ana le contesta —Pues....., sí. En clase, hicimos algunas prácticas con esta BD XML y realmente me gustó, es sencilla y está optimizada para consultas. Además tiene una versión libre. A lo que **Juan** le dice —Pues ahora es tu momento, serás nuestra profesora estos primeros días, para familiarizarnos con eXist.

eXist-db (o eXist para abreviar) es un proyecto de software de código abierto para bases de datos NoSQL basadas en tecnología XML . Se clasifica como un sistema de base de datos orientado a documentos NoSQL y una base de datos XML nativa (y brinda soporte para documentos XML , JSON , HTML y binarios). A diferencia de la mayoría de los sistemas de gestión de bases de datos relacionales (RDBMS) y las bases de datos NoSQL, eXist-db proporciona XQuery y XSLT como sus lenguajes de consulta y programación de aplicaciones.

eXist-db fue creado en 2000 por Wolfgang Meier. Las versiones principales lanzadas fueron 1.0 en septiembre de 2006, 2.0 en febrero de 2013, 3.0 en febrero de 2017, 4.0 en febrero de 2018 y 5.0.0-RC1 en junio de 2018.

Características

eXist-db permite a los desarrolladores de software conservar documentos XML / JSON / Binary sin escribir middleware extenso. eXist-db sigue y amplía muchos estándares XML de W3C , como XQuery . eXist-db también admite interfaces REST para la interfaz con formularios web de tipo AJAX . Las aplicaciones como XForms pueden guardar sus datos usando solo unas pocas líneas de código. La interfaz WebDAV para eXist-db permite a los usuarios "arrastrar y soltar" archivos XML directamente en la base de datos de eXist-db. eXist-db indexa automáticamente los documentos utilizando un sistema de indexación de palabras clave.

Estándares y tecnologías soportados.

eXist-db tiene soporte para los siguientes estándares y tecnologías:

- ✔ **XPath** - lenguaje de ruta XML.
- ✔ **XQuery** - lenguaje de consulta XML.
- ✔ **XSLT** - Transformaciones de lenguaje extensibles de hojas de estilo.
- ✔ **XSL-FO** - Objetos de formato XSL.
- ✔ **WebDAV** - Creación y control de versiones distribuidos en la web.
- ✔ **REST** - Transferencia de estado representacional (codificación de URL).
- ✔ **RESTXQ** - anotaciones RESTful para XQuery.
- ✔ **XInclude** - procesamiento del archivo del lado del servidor (soporte limitado).
- ✔ **XML-RPC** - un protocolo de llamada a procedimiento remoto.
- ✔ **XProc** - un lenguaje de procesamiento XML Pipeline.
- ✔ **API de XQuery para Java.**

Enlace a la pagina oficial de  [eXist-DB](http://exist-db.org).

3.1.- Instalación eXist.


Requisitos del sistema

eXist-db se ejecuta en cualquier sistema en el que se ejecuta Java. Así que todas las versiones recientes de Linux, macOS y Windows están bien. Se deben cumplir los siguientes requisitos:

- ✓ Al menos Java versión 8 (desde eXist-db 3.0)
- ✓ Aproximadamente 200Mb de espacio en disco para la instalación.
- ✓ Al menos 512Mb de memoria para la ejecución.

Instalando eXist-db

Los primeros pasos para instalar eXist-db son descargar el instalador y ejecutarlo:

- ✓ Descargue el instalador siguiendo el enlace de descarga en la página de  [inicio de eXist-db](#). Le recomendamos que descargue la última versión estable. Actualmente la 4.6.1.
El instalador es un único archivo llamado eXist-db-setup-[version].jar (por ejemplo eXist-db-setup-3.6.1.jar).
- ✓ Inicia el instalador eXist-db:
 - En Mac y Windows, simplemente haga doble clic en el archivo descargado.jar.
 - En las distribuciones de Linux con una interfaz de escritorio gráfica (por ejemplo, Ubuntu), puede iniciar el instalador haciendo que el archivo .jar sea ejecutable, haga clic derecho en él y seleccione la opción "Abrir con ... Java".
 - Si es necesario, puede iniciar el instalador desde la línea de comandos con el siguiente comando:

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

- ✓ Para una utilizar métodos de instalación avanzada sigue las indicaciones del  [artículo de instalación avanzada](#)

Después de iniciar el instalador, siga las instrucciones que se describen a continuación para completar la instalación. Le recomendamos que acepte las opciones predeterminadas, ya que están diseñadas para facilitar el uso de eXist-db. Los paneles de diálogo del instalador son los siguientes:

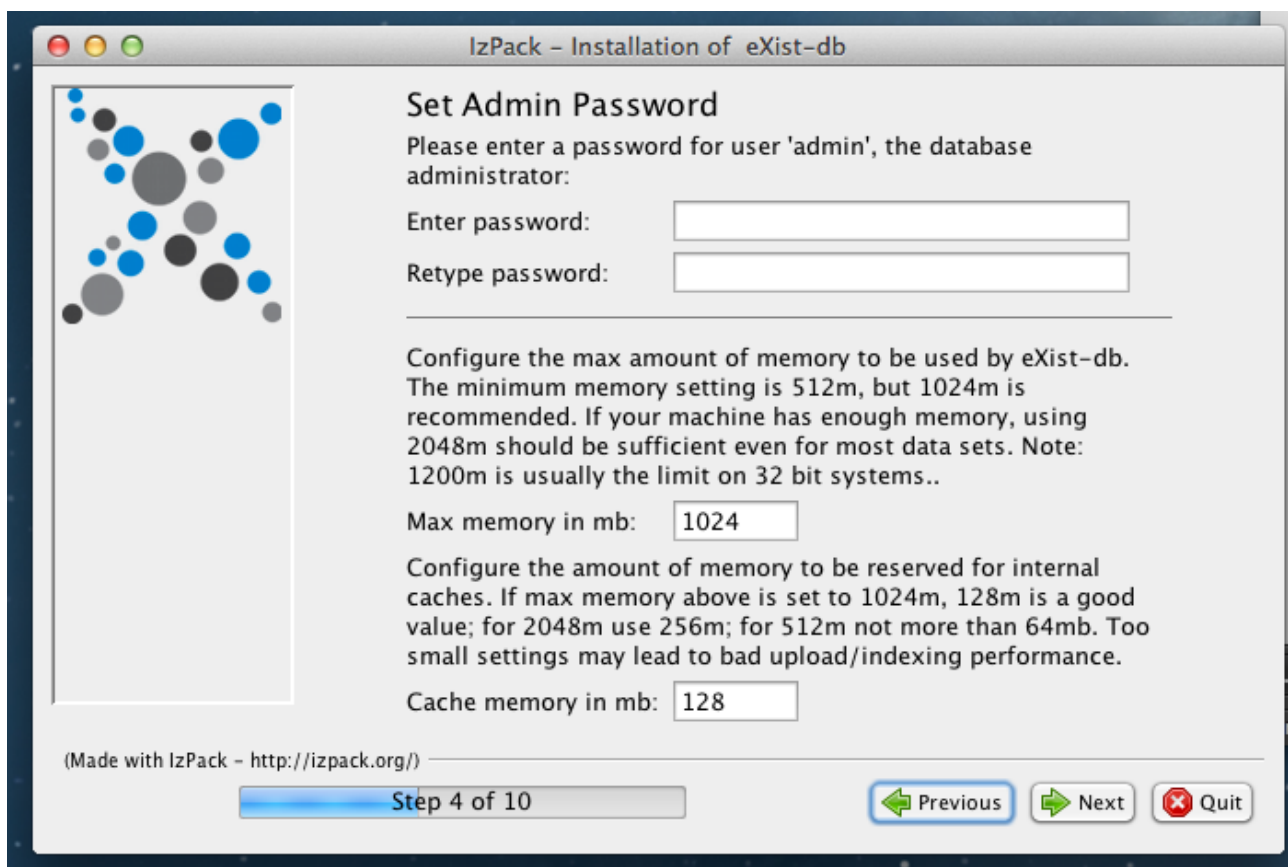
Directorio de instalación:

Se le preguntará dónde instalar eXist-db en su disco duro. El instalador le sugerirá un directorio apropiado, pero si lo desea, puede instalar eXist-db en cualquier parte de su sistema.

Directorio de datos

El directorio de datos es donde eXist-db guarda sus archivos de datos. El instalador sugerirá mantener los archivos de datos dentro del directorio de la aplicación, pero puede seleccionar una ubicación diferente si lo desea.

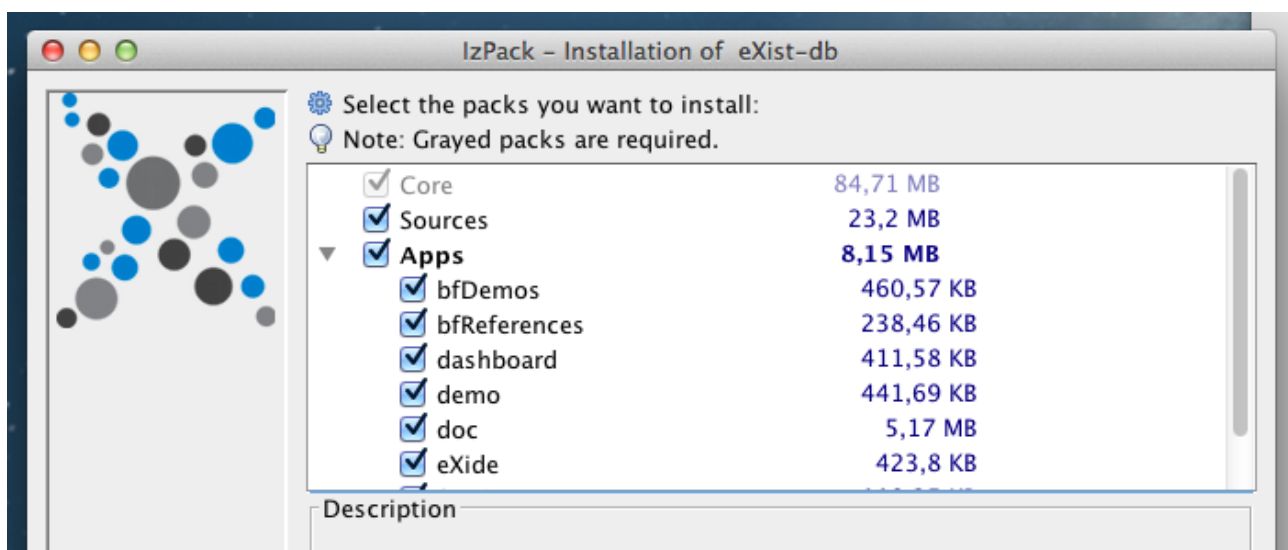
Contraseña de administrador y configuración de memoria:

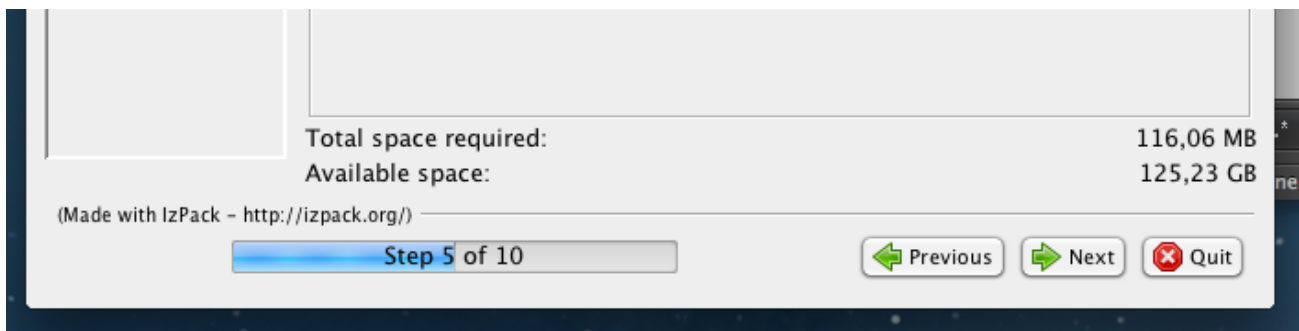


La Contraseña de administrador es una contraseña para la cuenta de administrador de eXist-db, o más comúnmente, la cuenta de "administrador". Esta cuenta de administrador le pertenece a usted, y ciertas funciones clave en eXist-db solo pueden ser realizadas por el administrador. Si bien puede dejar en blanco la contraseña de administrador, le recomendamos que establezca la contraseña para poder acceder de forma segura a su instalación de eXist-db. ¿Por qué? Tenga en cuenta que mientras eXist-db se está ejecutando, otros usuarios pueden acceder a él en su red local (ya sea en su casa u oficina, en una cafetería o en un tren). Por lo tanto, asegurar su cuenta de administrador en eXist-db es una buena manera de proteger sus datos y evitar que otros abusen de la cuenta.

Configure la cantidad máxima de memoria que estará disponible para Java (y eXist-db) y la porción de la misma que estará reservada para cachés internos.

Instalación del paquete:





1. El paquete "core" es requerido para ejecutar eXist-db
2. El paquete "fuentes" es opcional. La eliminación de "fuentes" reduce considerablemente el tamaño de la instalación, pero es mejor instalar todo a menos que esté privado de espacio en el disco.
3. El paquete de "aplicaciones" le permite seleccionar o deseleccionar una serie de aplicaciones que se instalarán en eXist-db cuando se inicie por primera vez. Si eres nuevo en eXist-db, te recomendamos que al menos selecciones "dashboard", "demo", "doc", "eXide" y "fundocs". Los necesitarás para realizar tus primeros pasos en el desarrollo de XQuery.

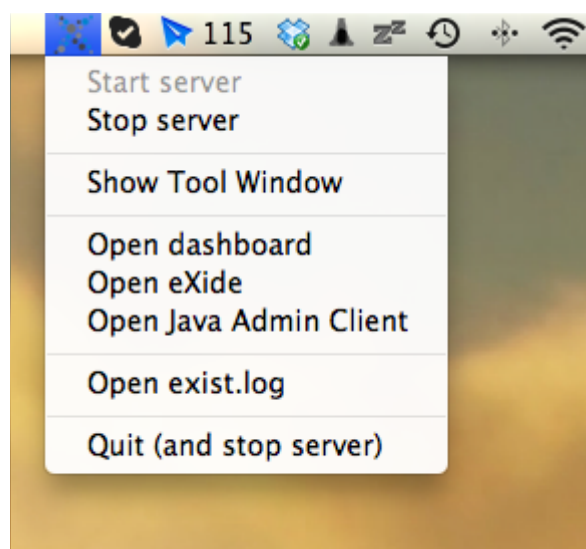
Después de esto, el instalador realizará una serie de pasos adicionales y pantallas para instalar los archivos y configurar el sistema. Una vez terminado, ya está listo para correr.

Lanzando eXist-db



Para lanzar eXist-db en Linux o Windows, seleccione el icono de acceso directo del escritorio de *Inicio de la base de datos eXist-db* o la entrada del Menú Inicio.

eXist-db también se puede ejecutar como un servicio en segundo plano. Para obtener más información, consulte el artículo de [instalación avanzada](#).

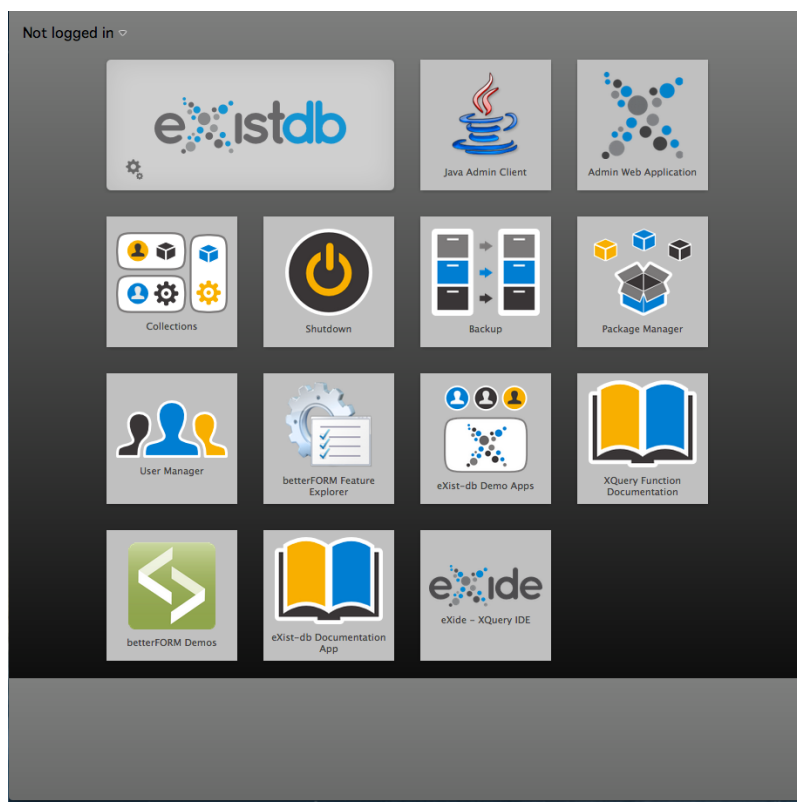
Aparece una pantalla de bienvenida que muestra el logotipo de eXist-db. En el primer arranque, eXist-db cargará las aplicaciones que seleccionó en el instalador. Esto puede llevar un tiempo y solo se hace una vez.



3.2.- Primeros pasos.


Después de una instalación y lanzamiento exitosos, acceda al  [Panel](#) de  [instrumentos de eXist-db](#), a la administración central y al centro de aplicaciones de eXist-db. Si el icono de la bandeja del sistema funciona en su sistema, seleccione *Abrir cuadro de mandos* en el menú emergente. O simplemente abra un navegador web e ingrese la siguiente **URL: <http://localhost:8080/exist/>**

Aparece la siguiente página:



Cerrar la base de datos

El cierre incorrecto de la base de datos puede dañar sus archivos de datos. Siga uno de los siguientes procedimientos para cerrar correctamente eXist-db:


- ✓ El menú emergente de la bandeja del sistema tiene una opción **Detener servidor**. También puede elegir **Salir** (y **detener el servidor**) para cerrar eXist-db y el iniciador de la bandeja del sistema al mismo tiempo.
- ✓ Desde el  [panel de control](#) : seleccione el botón de apagado
- ✓ Desde la línea de comandos, ejecute los scripts de apagado `bin/shutdown.sh` (Linux / Mac) o `shutdown.bat` (DOS / Windows), usando las credenciales de su cuenta de administrador:

```
bin / shutdown.sh -u admin -p youradminpassword
```

```
bin / shutdown.bat -u admin -p youradminpassword
```

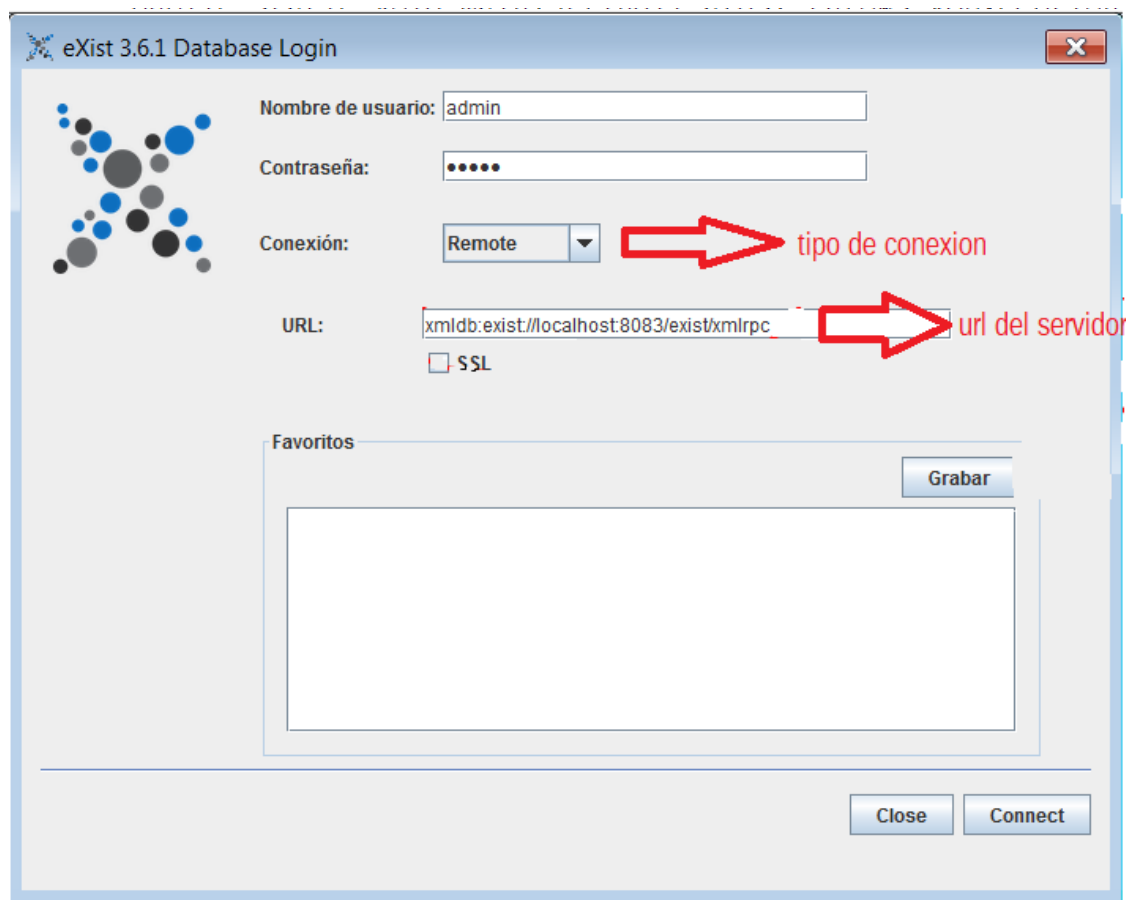
- ✓ Desde la línea de comandos, ejecute el siguiente comando de Java, usando las credenciales de su cuenta de administrador:

```
java -jar start.jar shutdown -u admin -p youradminpassword
```

👍 Desde el  [Java Admin Client](#) : seleccione **Conexión** , **Apagar** desde el menú.

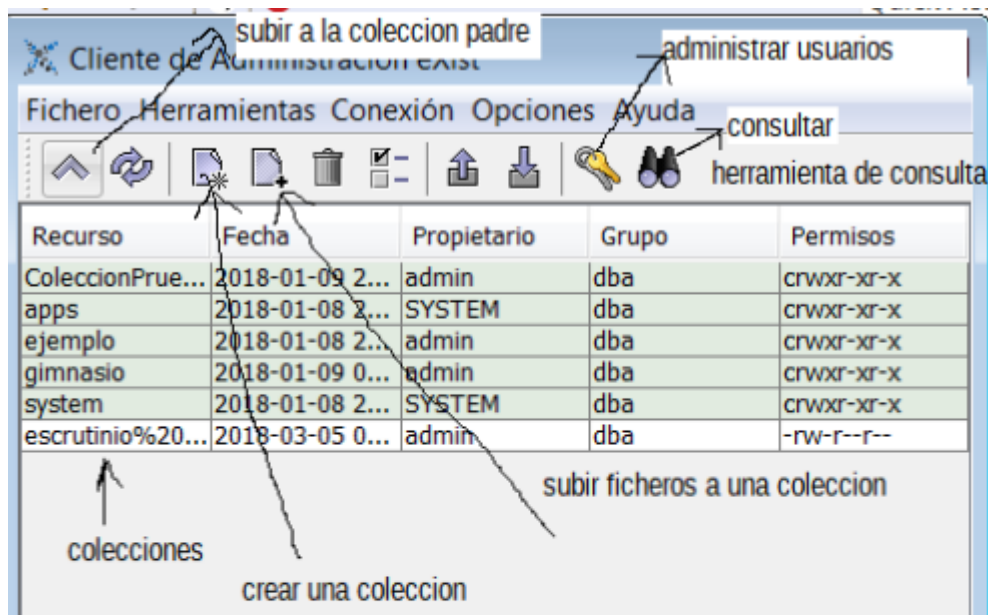
3.3.- Java Admin Client de eXist.

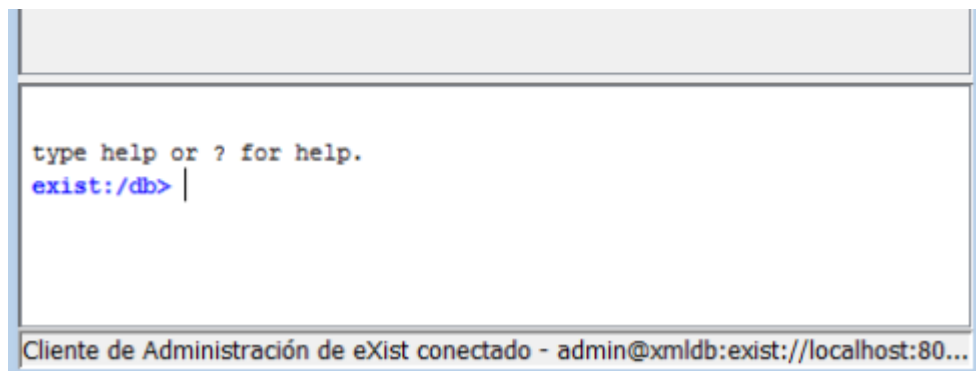
Lo primero que tenemos que hacer para utilizar Java Admin Client de eXist es dar un usuario y contraseña y la url del servidor de eXist.



La ventana principal nos permite crear colecciones, subir ficheros xml a alguna de las colecciones que tenemos creadas, para ello primero la creamos y a continuación hacemos doble clic sobre ella.

Las colecciones pueden estar anidadas. La jerarquía de colecciones se almacena en el fichero `collections.dbx`

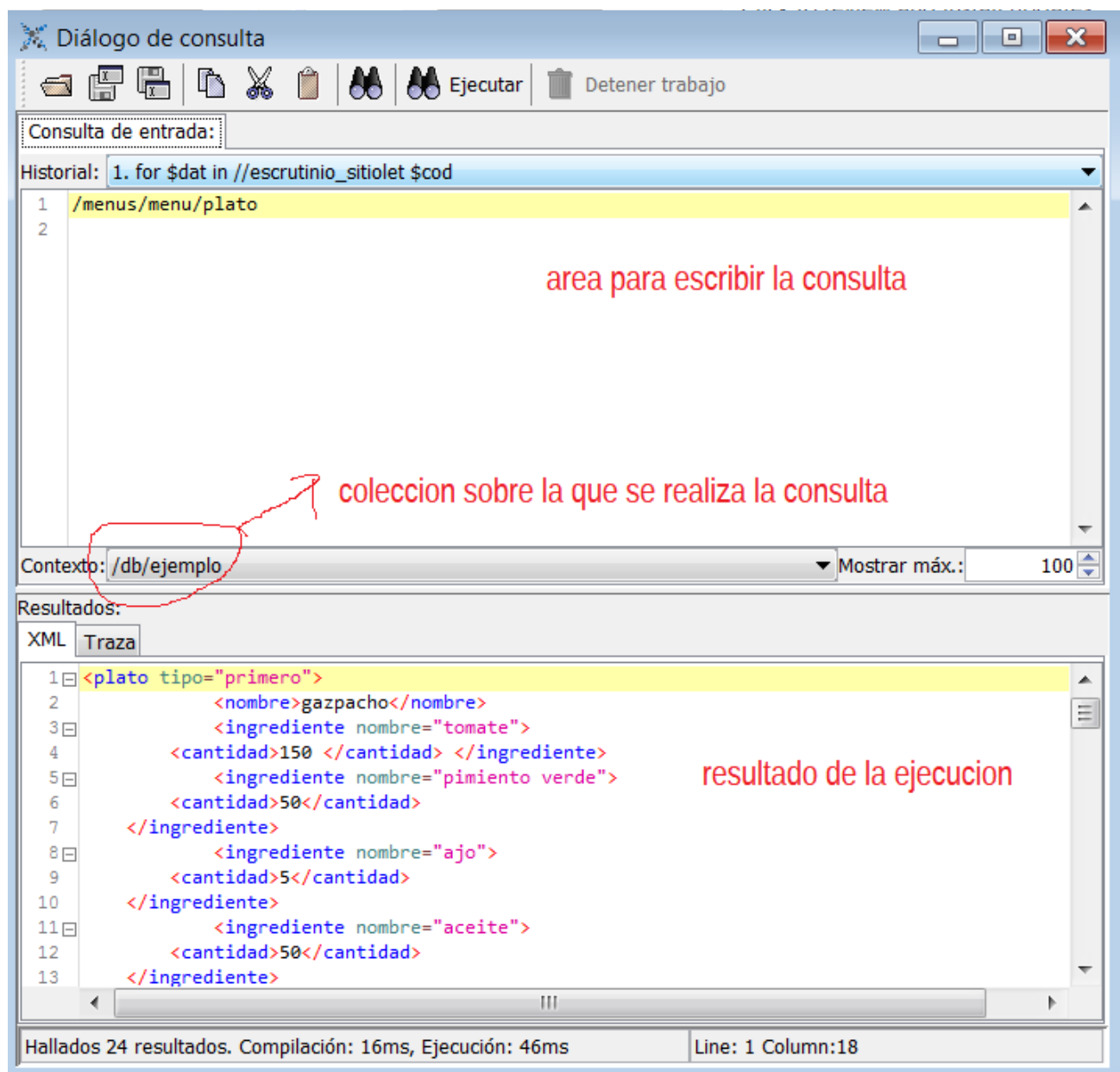




La herramienta de consulta no abre otra ventana, en la que escribimos consultas sobre todos los documentos de la colección activa y las ejecutamos.




Las herramientas de la barra por orden permiten:

abrir una consulta almacenada, grabar una consulta en un fichero(podremos utilizarlas en programas java), guardar los resultados en un fichero, copiar, cortar, pegar del portapapeles, compilar la consulta y ejecutar la consulta



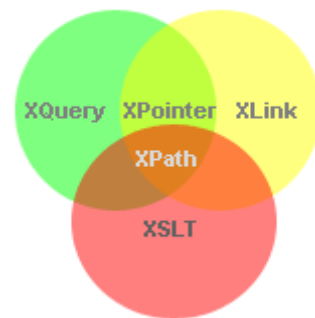
Tienes mas información en la [página de eXist-db](#).

3.4.- Lenguaje de consulta XPath.

XPath (XML Path Language) es un lenguaje que permite construir expresiones que recorren y procesan un documento XML. La idea es parecida a las  [expresiones regulares](#) para seleccionar partes de un texto sin atributos (plain text). XPath permite buscar y seleccionar teniendo en cuenta la estructura jerárquica del XML. XPath fue creado para su uso en el estándar  [XSLT](#), en el que se usa para seleccionar y examinar la estructura del documento de entrada de la transformación. XPath fue definido por el consorcio  [W3C](#).

Características de XPath


- ✓ Es una sintaxis para definir partes de un documento XML.
- ✓ Usa expresiones de ruta para navegar en documentos XML.
- ✓ Contiene una biblioteca de funciones estándar.
- ✓ Es un elemento importante en XSLT y en XQuery.
- ✓ XPath 3.0 es una recomendación de W3C desde Abril de 2014.



Tipos de nodos

Las partes de un documento XML se denominan nodos. Existen 7 tipos de nodos diferentes:

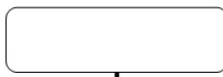
- ✓ **Raíz:** El nodo raíz o nodo documento (root node) no debe confundirse con el elemento raíz. Éste es más bien el nodo padre virtual del elemento raíz.
- ✓ **Elemento** (element node).
- ✓ **Atributo** (attribute node).
- ✓ **Texto** (text node).
- ✓ **Espacio de nombres** (namespace node).
- ✓ **Instrucción de procesamiento** (processing instruction node).
- ✓ **Comentario** (comment node).

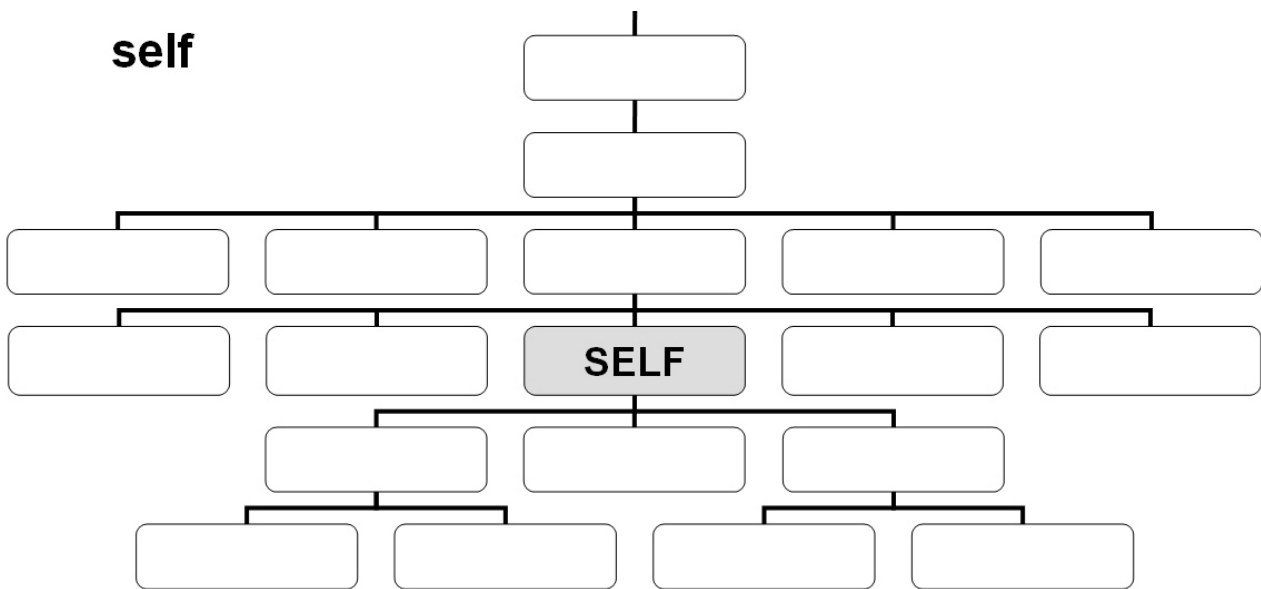
Cada uno de estos nodos puede ser seleccionado con  [XPath](#) para su posterior procesamiento en XSL.

Los ejes

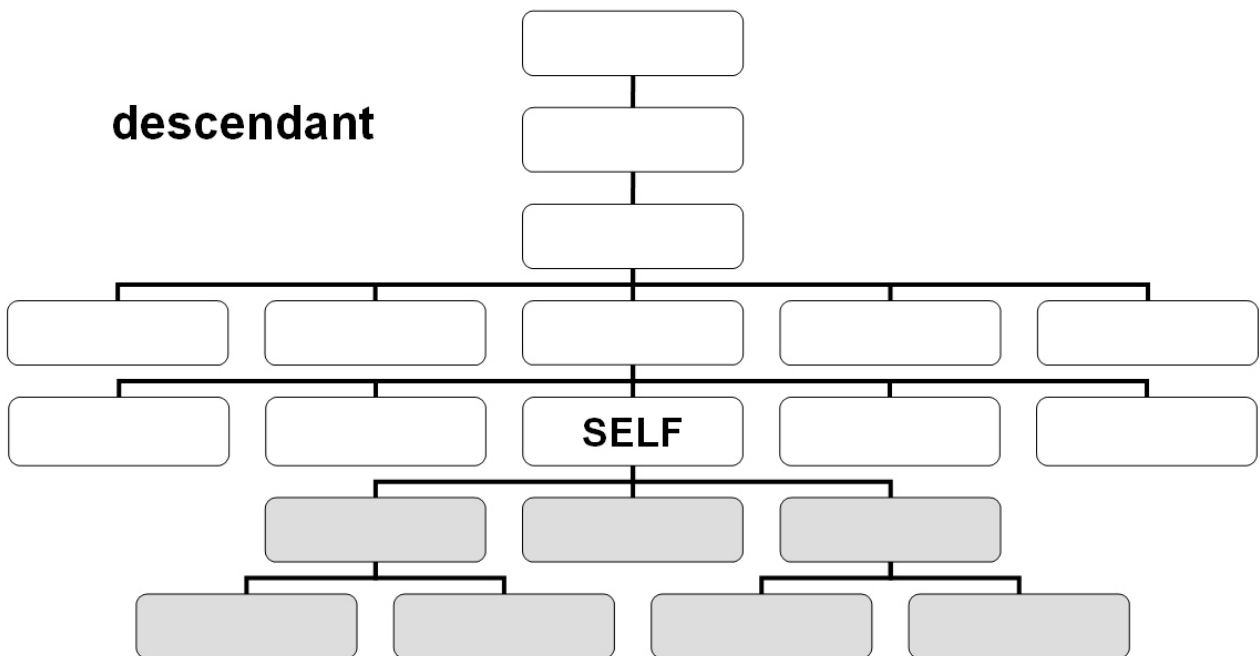
La navegación dentro de un documento XML a través de XPath tiene lugar desde un nodo de contexto, que en las siguientes ilustraciones se representan como 'SELF'. El nodo de contexto es siempre el punto de partida en el que se encuentra el procesador XSLT. Los nombres de los ejes definen las relaciones de parentesco respecto al mismo. En las ilustraciones se muestran 11 de los 13 ejes disponibles. Junto a los ejes que presentamos a continuación, que permiten la navegación en un documento, es posible seleccionar atributos de un nodo o nodos de espacio de nombres a través de los ejes `attribute` o `namespace`.

El eje self contiene el nodo de contexto.

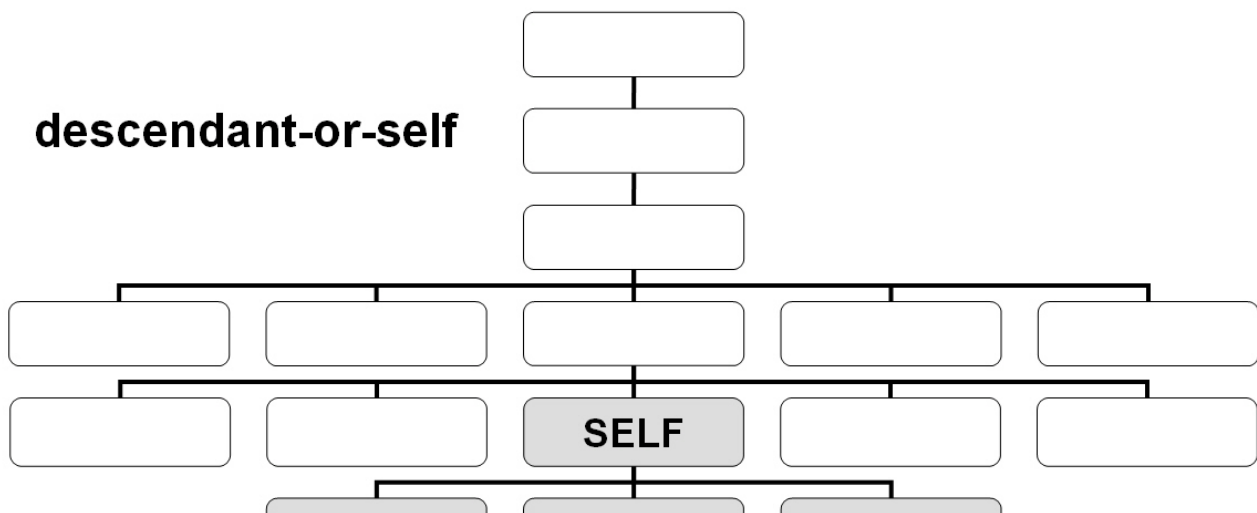


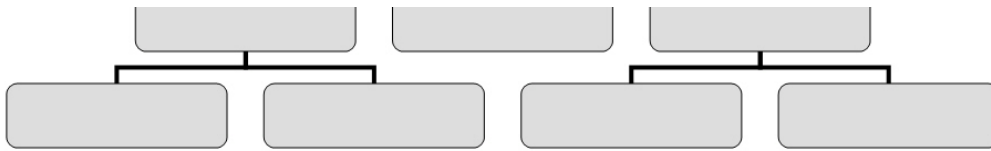
self

El eje child contiene los nodos hijo del nodo de contexto.

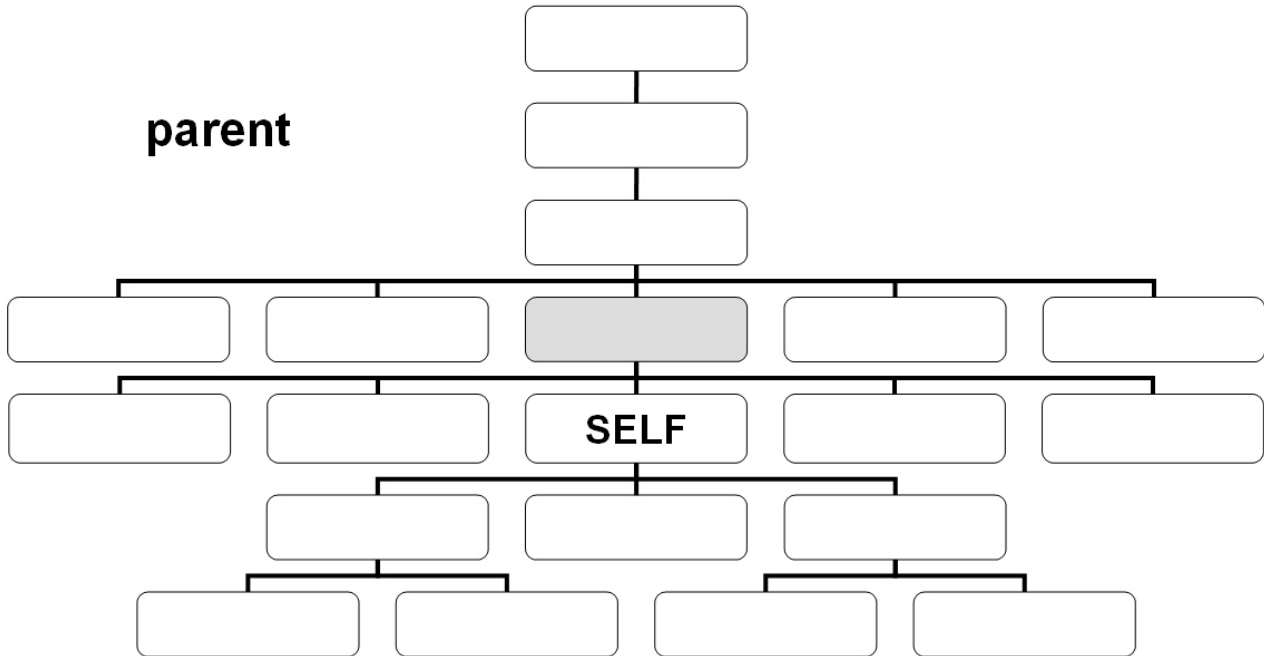
descendant

El eje descendant-or-self contiene el nodo de contexto y los descendientes.

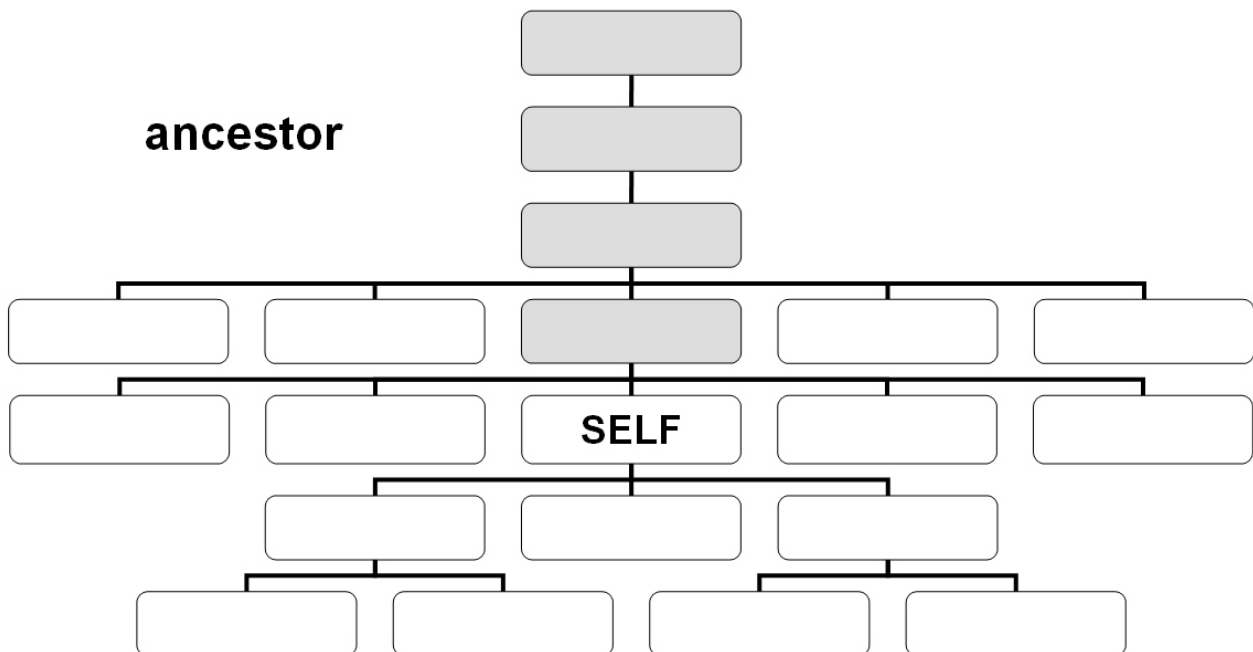
descendant-or-self



El eje parent contiene el padre del nodo de contexto.

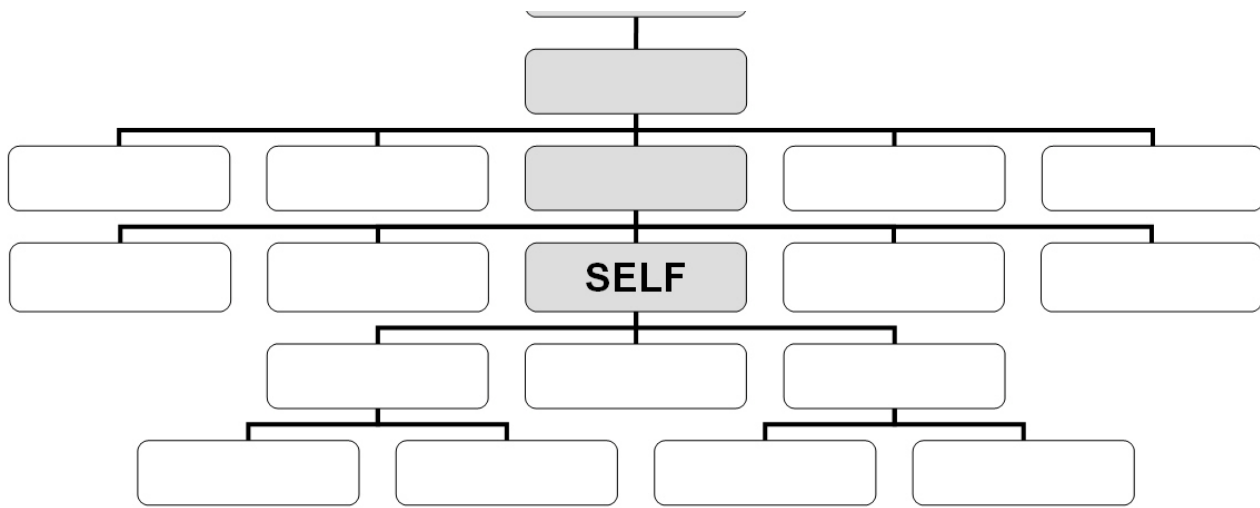


El eje ancestor contiene los ancestros del nodo de contexto.

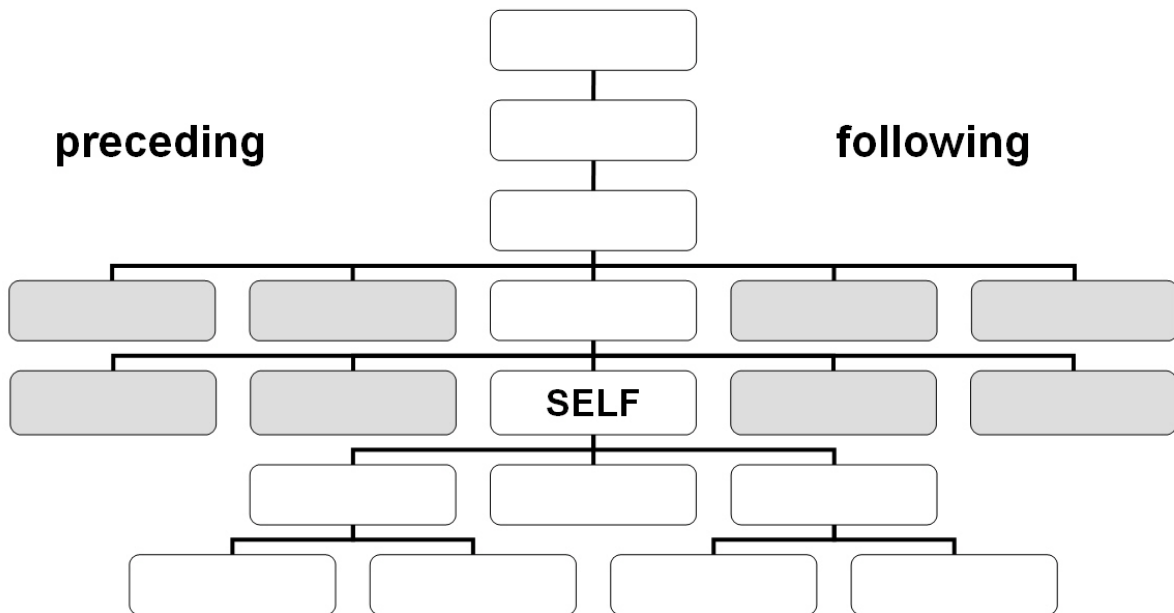


El eje ancestor-or-self contiene el nodo de contexto y sus descendientes.

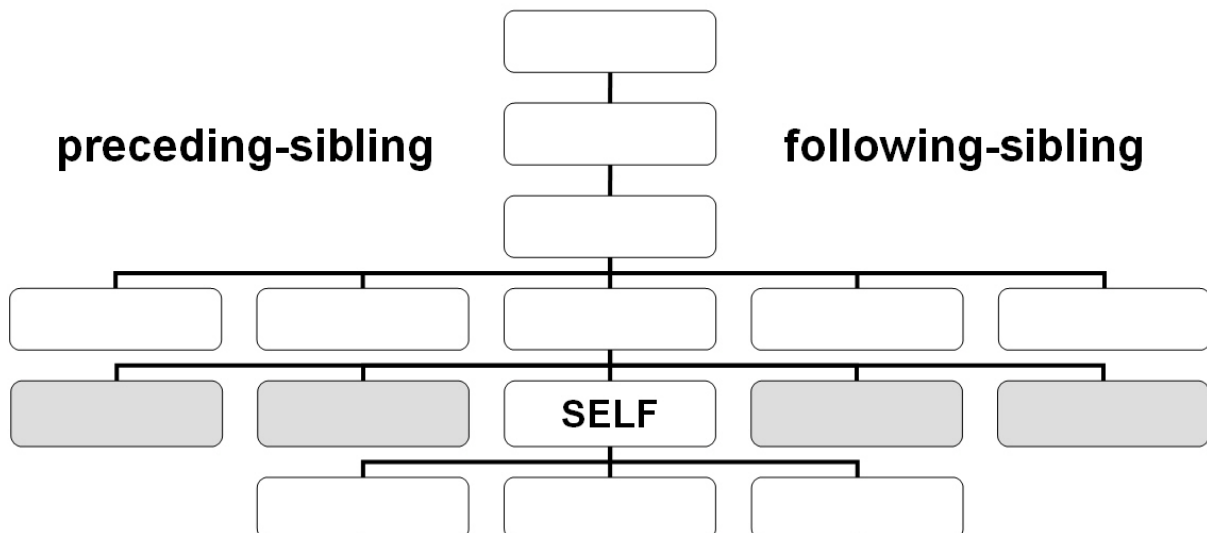




El eje preceding contiene todos los nodos que aparecen antes del nodo de contexto, excluyendo los ancestros. El eje following contiene todos los nodos que aparecen después del nodo de contexto, excluyendo los descendientes.



El eje preceding-sibling contiene los hermanos precedentes del nodo de contexto. El eje following-sibling contiene los hermanos situados detrás del nodo de contexto.





Localización

La localización es la expresión que permite al procesador seleccionar nodos o un conjunto de nodos. Esta localización se conoce como camino o ruta de localización y puede presentar diversas formas:

- ✔ con sintaxis abreviada o completa.
- ✔ como ruta de localización relativa o absoluta.

Sintaxis abreviada o completa

En XPath existen dos formas de sintaxis, que en la práctica se pueden utilizar de manera combinada. La sintaxis abreviada se usa normalmente cuando se trata de nodos y ejes que son seleccionados con mucha frecuencia, mientras que la sintaxis completa se utiliza en caso de nodos y ejes a los que se accede con menor frecuencia. En los siguientes apartados se utilizará la sintaxis completa, recurriendo a la sintaxis abreviada cuando se trate de expresiones más frecuentes.

- ✔ Un ejemplo de sintaxis completa:

```
/child::libro/child::autor/child::nombre/attribut::apellido
```

- ✔ En sintaxis abreviada:

```
/libro/autor/nombre/@apellido
```

En la sintaxis abreviada se prescinde del nombre del eje `child::` y el atributo se introduce anteponiendo el carácter `@`. En los próximos ejemplos se ofrecerán más detalles sobre la sintaxis abreviada.

Equivalencias entre sintaxis completa y abreviada

Sintaxis completa	Sintaxis abreviada
<code>child::</code>	Eje por defecto. Se puede omitir.
<code>attribute::</code>	<code>@</code>
<code>descendant-or-self::node()/</code>	<code>//</code>
<code>self::node()</code>	<code>.</code>

Sintaxis completa	Sintaxis abreviada
parent::node()	..

Rutas relativas y absolutas

La ruta de localización XPath puede ser relativa o absoluta. Una ruta absoluta comienza por el nodo raíz, que es el nodo situado directamente sobre el elemento raíz. Esta distinción es necesaria, ya que desde el elemento raíz no sería posible acceder, por ejemplo, a comentarios o instrucciones que se encuentran fuera del mismo. Las rutas de localización constan de pasos de localización separados por /.

Un ejemplo:

```
/child::Europa/child::pais/child::nombre
```

En este caso se trata de una ruta absoluta que parte del nodo raíz. El nodo raíz se indica mediante una barra diagonal. Desde ahí se selecciona el elemento raíz <Europa>. La expresión generará un resultado si el elemento raíz tiene un nodo hijo <pais> y éste a su vez contiene un nodo hijo <nombre>. Los ejes se definen mediante el nombre del eje seguido de dos signos de dos puntos. En el caso del eje child, puede omitirse la expresión child::. Así, la ruta de localización Europa/pais/nombre equivale a la ruta del ejemplo con sintaxis abreviada. Por el contrario, las rutas de localización relativas necesitan un nodo de contexto. La ruta se evaluará desde esta posición.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="style.xsl"?>
<Europa>
  <pais>
    <nombre>Alemania</nombre>
    <habitantes unidad="millones">82.4</habitantes>
    <capital>Berlín</capital>
    <sigla-pais>D</sigla-pais>
    <prefijo>0049</prefijo>
  </pais>
  <pais>
    <nombre>Francia</nombre>
    <habitantes unidad="millones">58.5</habitantes>
    <capital>Paris</capital>
    <sigla-pais>F</sigla-pais>
    <prefijo>0033</prefijo>
  </pais>
  <pais>
    <nombre>España</nombre>
    <habitantes unidad="millones">39.4</habitantes>
    <capital>Madrid</capital>
    <sigla-pais>E</sigla-pais>
    <prefijo>0034</prefijo>
  </pais>
</Europa>
```

```
</Europa>
```

Ejemplo:

```
child::pais/child::nombre
```

De igual forma que en la ruta absoluta, en este caso se generará un resultado si el nodo de contexto tiene un nodo hijo `<pais>`, que a su vez posee un nodo hijo `<nombre>`.

La descripción aquí presentada de las rutas relativas es incompleta. Podrá encontrar más información sobre rutas relativas en literatura especializada sobre XPath.

Las siguientes son expresiones XPath con sintaxis abreviadas y extendidas. ¿Qué mostrarían estas consultas?

```
/Europa/pais/nombre  
/Europa/pais/habitantes  
/Europa/descendant::habitantes  
/Europa/pais/nombre/following-sibling::habitantes  
/Europa/pais/nombre/following-sibling::habitantes/parent::pais/descendant::capit
```

Filtrar conjuntos de nodos a través de predicados

Las expresiones XPath sirven para la selección de nodos en un árbol XML. Gracias al filtrado de expresiones a través de predicados, es posible usar expresiones XPath para seleccionar grupos de nodos que respondan a criterios más complejos, definiendo el acceso a los nodos de una manera mucho más precisa. Los predicados son expresiones que devuelven un resultado booleano (verdadero o falso), de manera que se puede filtrar de nuevo el resultado del nodo seleccionado a través de la expresión XPath. En caso de que un determinado nodo dentro de un conjunto de nodos se corresponda con el predicado de la expresión (que se cumpla la condición del mismo), éste pasará a formar parte del resultado. Por el contrario, si la condición no se cumple, se excluirá el nodo del resultado. De esta forma es posible afinar la búsqueda dentro de un determinado conjunto inicial de nodos a través de las restricciones realizadas mediante predicados.

Localización de distintos tipos de nodos

Hasta ahora nos hemos ocupado sobre todo con los nodos elemento. No obstante, existen otros tipos de nodo que pueden ser seleccionados para su posterior análisis y procesamiento. La siguiente tabla muestra ejemplos de cómo se realiza tal selección.

<code>nombre/text()</code>	En caso de que se quiera acceder al nodo de texto del elemento <code>nombre</code> se deberá usar la prueba de nodo <code>text()</code> .
<code>direccion/comment()</code>	En caso de que se quiera acceder al nodo de comentario del elemento <code>direccion</code> se deberá usar la prueba de nodo <code>comment()</code> .

<code>dirección/node()</code>	Si se quiere acceder a todos los tipos de nodos (con excepción de los atributos) se deberá usar la prueba de nodo <code>node()</code> .
-------------------------------	---

Ejemplo:

```
//direccion[localidad="Stuttgart"]
```

La expresión `//direccion` selecciona todos los nodos elemento "`direccion`" del documento. En el segundo paso se reduce esta selección mediante un predicado. Sólo aquellos elementos que tengan un elemento hijo "`localidad`" y que el valor del mismo sea "`Stuttgart`" permanecerán en la selección. Los demás serán excluidos.

Es posible, además de la comparación de cadenas, restringir la selección mediante la comparación de otros valores en la expresión de un predicado. Por ejemplo:

Comparación de cadena	<code>direccion[nombre="Pepe López"]</code>
Comparación numérica (mayor)	<code>disco[@calificacion > 3]</code>
Comparación numérica (menor o igual)	<code>//cancion[@año <= 1990]</code>
Combinación de "mayor" y "no igual a"	<code>disco[@calificacion < 2]/cancion[@año != 2000]</code>

Además de la comparación de valores, es posible establecer la coexistencia de otros elementos como criterio para analizar un conjunto de nodos.

<code>/listadedirecciones[direccion]</code>	Comprueba si existe el elemento hijo " <code>dirección</code> ".
<code>/listadedirecciones/direccion[@cat]</code>	Comprueba si existe un atributo " <code>cat</code> " dentro de " <code>direccion</code> ".

Atención: Un predicado sólo comprueba si la condición es verdadera o falsa. Se trata de una conversión implícita a un valor booleano.

Operadores booleanos en predicados

Los predicados pueden contener los operadores booleanos "`and`" y "`or`".

<code>//titulo[@estilo="pop" and @calificacion=4]</code>	Selecciona todos los elementos " <code>titulo</code> " que posean un atributo " <code>estilo</code> " con el valor " <code>pop</code> " y además un atributo " <code>calificacion</code> " con el valor 4.
<code>//titulo[@estilo="pop" or @calificacion>2]</code>	Selecciona todos los elementos " <code>titulo</code> " que posean un atributo " <code>estilo</code> " con el valor " <code>pop</code> " o un atributo " <code>calificacion</code> " cuyo valor sea mayor que 2.

Predicados en cascada

Además del filtrado mediante operadores booleanos, es posible restringir la selección mediante una disposición en serie de predicados. Los predicados en cascada funcionan como una superposición de filtros. En primer lugar se filtra un conjunto de nodos, que será a su vez el conjunto de partida para el segundo predicado, etc.

```
//seccion[parrafo][@tipo='advertencia']
```

En primer lugar se seleccionan todos los elementos seccion. El primer predicado reduce la selección a aquellos elementos que tengan un elemento hijo "parrafo". El conjunto resultante se filtra de nuevo mediante el segundo predicado, de manera que sólo quedaran los elementos que tengan un atributo "tipo" con el valor "advertencia".

Unión de conjuntos de nodos

Hasta ahora se ha definido en cada expresión sólo un conjunto de nodos. No obstante, se dan con frecuencia aplicaciones en las que se deben unir varios conjuntos de nodos. En el ejemplo ofrecido a continuación se seleccionan todos los títulos dentro del estilo pop y todas las compañías discográficas.

```
//titulo[@estilo="pop"] | //discografica
```

Funciones XPath

Las funciones ofrecen operaciones avanzadas adicionales en la consulta de conjuntos de nodos, así como en el análisis de cadena de nodos de texto y valores de atributos. Las funciones [XPath](#) se pueden usar en expresiones XPath y en predicados, tal como se muestra en el siguiente ejemplo:

```
count(//cancion)  
//disco[count(cancion)>10]
```

Las funciones pueden contener un argumento y devuelven siempre un valor. En el primer ejemplo la función "count" tiene como argumento un conjunto de nodos y devuelve como valor de salida un conjunto de nodos.

Ejemplos de funciones de conjuntos de nodos:

local-name(..)	Devuelve el nombre del elemento padre como cadena.
//cancion[position()=5] (Sintaxis abreviada [5])	Devuelve todas las canciones que ocupen la quinta posición en cada disco.

```
//cancion[position()=last()]
```

Devuelve las canciones que ocupen la última posición en cada disco.

Ejemplos de funciones de cadena:

```
//disco[string-length(titulo) > 20 ]
```

Selecciona todos los discos cuyo título tenga más de 20 caracteres.

```
//disco[starts-with(interprete,'M')]
```

Selecciona todos los discos en los que el nombre del intérprete comience con la letra "M".

Un ejemplo de una función muy útil y usada con mucha frecuencia es "`not()`". Esta función niega una expresión booleana. A continuación se ofrece un ejemplo del empleo de esta función:

```
//disco[not(cancion)]
```

Selecciona todos los discos que no tengan un elemento "`cancion`".

3.5.- El lenguaje de consultas XQuery.



Caso práctico

Ana está orgullosa en su nueva faceta de profesora. **Juan** y **María** la han felicitado y están satisfechos con la introducción que les ha dado sobre Qizx. **Ana** les dice —Antes de empezar con la aplicación en Java, necesitamos familiarizarnos con el lenguaje de consultas XQuery.




A lo que **Juan** le pregunta —¿Conoces algo del lenguaje?"

Ana sonríe y contesta —Pues...sí. En clase vimos también algunos ejemplos sencillos de uso.

Juan, con cara de asombro dice —Está claro que el nuevo Ciclo Formativo de DAM incorpora muchos temas punteros que yo no domino. El próximo curso iniciaré sin falta el estudio de algunos módulos.

XQuery es un lenguaje de consulta diseñado para extraer información de colecciones de datos expresadas en XML. Entre las **principales características de XQuery** vamos a destacar las siguientes:



- ✔ Está **basado en el lenguaje XPath**, (XML Path Language), y se fundamenta en él para realizar la selección de información y la iteración a través del conjunto de datos XML.
- ✔ Es un **lenguaje declarativo**, lo que significa que, en vez de ejecutar una lista de comandos como un  lenguaje procedimental clásico, cada consulta es una expresión que es evaluada y devuelve un resultado, al igual que en SQL.

Podemos decir que **XQuery es a XML lo mismo que SQL es a las bases de datos relacionales**. Sin embargo, aunque XQuery y SQL puedan considerarse similares, el modelo de datos sobre el que se sustenta XQuery es muy distinto del modelo de datos relacional sobre el que sustenta SQL, ya que XML incluye conceptos como jerarquía y orden de los datos que no están presentes en el modelo relacional.

Por ejemplo, a diferencia de SQL, en XQuery es importante y determinante el orden en que se encuentren los datos, ya que no es lo mismo buscar una etiqueta `nombre` dentro de una etiqueta `autor` que todas las etiquetas `nombre` del documento (que pueden estar anidadas dentro de una etiqueta `autor` o fuera).

Los **principales tipos de expresiones de XQuery** son:

- ✓ Expresiones XPath, para navegar por los documentos.
- ✓ Expresiones FLWOR (For, Let, Where, Order, Return) para iterar por los elementos de un conjunto de datos.

Pero XQuery también admite:

- ✓ Constructores para generar nodos y contenido dinámico.
- ✓ Condicionales (IF, THEN ELSE) para construir el resultado en base a alguna condición.
- ✓ Cuantificadores (SOME, ANY) para chequear la existencia de algún elemento que cumpla una condición.
- ✓ Listas a las que se pueden aplicar operadores (UNION,...) y funciones de agregación (AVG, COUNT,...).



Para saber más

En este enlace puedes profundizar sobre el lenguaje XQuery.



[Especificación completa de XQuery](#)

El siguiente enlace te proporciona a su vez enlaces muchos enlaces interesantes a varias tecnologías asociadas a XQuery, así como a XQuery.

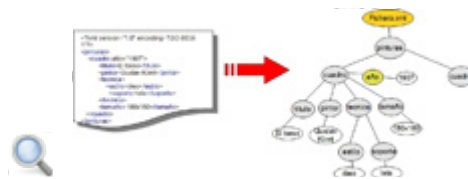


[Tecnologías asociadas con XQuery](#)

3.5.1.- Modelo de datos.

El modelo de datos en que se sustenta XQuery es el modelo de datos de XPath.

XPath modela un documento XML como una estructura jerárquica en forma de árbol. El árbol está **formado por** **👉 nodos**, **y hay siete tipos de nodos**: raíz, elemento, texto, atributo, espacio de nombres, instrucción de procesamiento y comentario.



Los **principales nodos** de la estructura jerárquica o en árbol en un documento XML son: (puedes verlos en la imagen ampliable superior)

- ✓ **Nodo raíz o /**. Es el primer nodo del documento.
- ✓ **Nodo elemento**. Cualquier elemento de un documento XML. Cada nodo elemento posee un padre y puede o no tener hijos. En el caso de que no tenga hijos, es un nodo hoja.
- ✓ **Nodo texto**. Cualquier elemento del documento que no esté marcado con una etiqueta de la DTD del documento XML.
- ✓ **Nodo atributo**. Un nodo elemento puede tener etiquetas que complementen la información de ese elemento. Esto sería un nodo atributo.



Debes conocer

En el siguiente enlace tienes una ilustración gráfica más detallada del paso de un documento XML a un árbol de nodos XML, así como la explicación en detalle de cada tipo de nodo. (Apartado 2.Modelo de datos XPath)



[Nodos de un árbol XML](#)



Autoevaluación

Señala la opción correcta. Entre los nodos de la estructura jerárquica

de un árbol XML están:

- ☐ Nodo etiqueta y nodo atributo.
- ☐ Nodo padre.
- ☐ Nodo hijo.
- ☐ Nodo raíz y nodo elemento.

Es incorrecto, prueba de nuevo.

No es correcto, 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

3.5.2.- Caminos de localización.

¿Cómo se localiza cada uno de esos nodos en el árbol XML? Mediante una expresión XPath conocida como camino o ruta de localización. **Un camino de localización:**



- ✓ Selecciona un conjunto de nodos relativo al nodo de contexto.
- ✓ Puede contener recursivamente expresiones utilizadas para filtrar conjuntos de nodos.
- ✓ Al ser evaluado, devuelve el conjunto de nodos seleccionados por el camino de localización.
- ✓ Se construye siguiendo unas reglas de sintaxis y semántica.

Hay dos **tipos de caminos de localización:**

- ✓ **Caminos relativos.** Son una secuencia de uno o más pasos de localización separados por /.
 - ➡ Los pasos se componen de izquierda a derecha.
- ✓ **Caminos absolutos.** Consiste en / seguido, opcionalmente, por un camino de localización relativo.
 - ➡ Una / por sí misma selecciona el nodo raíz del documento que contiene al nodo contextual.

Los siguientes, son **algunos ejemplos de caminos de localización:**

- ✓ `cuadro` selecciona los elementos `cuadro` hijos del nodo contextual.
- ✓ `cuadro/titulo` selecciona los elementos `titulo` descendientes de los elementos `cuadro` hijos del nodo contextual.
- ✓ `*` selecciona todos los elementos hijos del nodo contextual.
- ✓ `@año` selecciona el atributo `año` del nodo contextual
- ✓ `@*` selecciona todos los atributos del nodo contextual
- ✓ `cuadro[1]` selecciona el primer hijo `cuadro` del nodo contextual
- ✓ `cuadro[@año=1907]` selecciona todos los hijos `cuadro` del nodo contextual que tengan un atributo `año` con valor `1907`.



Debes conocer

En el siguiente enlace tienes ejemplos de caminos de localización en la sintaxis abreviada de XPath (Es el apartado 2.5 Sintaxis abreviada)



[Caminos de localización XPath](#)



Citas para pensar

Razonar y convencer, ¡qué difícil, largo y trabajoso!
¿Sugestionar? ¡Qué fácil, rápido y barato!

Santiago Ramón y Cajal



Autoevaluación

Señala si la siguiente afirmación es verdadera o falsa.

El camino: //curso selecciona los elementos curso descendientes del nodo contextual.


☐ Verdadero ☐ Falso

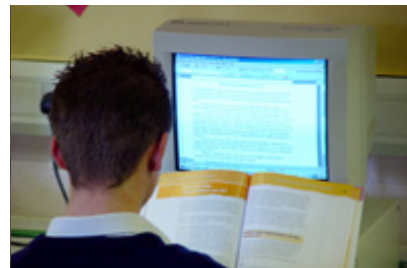
Falso

Es falso, selecciona todos los descendientes curso del raíz del documento.

3.5.3.- Primeras consultas XQuery.

Una **consulta XQuery** es una expresión que lee una **secuencia de datos en XML** y devuelve como resultado **otra secuencia de datos en XML**, donde:

- ✔ Una **secuencia** es un conjunto ordenado de cero o más ítems.
- ✔ Un **ítem** es cualquier tipo de nodo del árbol XML o un  valor atómico.



Las funciones que se pueden invocar para **referirnos a colecciones y documentos dentro de la BD** son las siguientes:

```
collection(camino de la colección)
```

```
doc(camino del documento)
```

Así por ejemplo:

- ✔ La consulta `collection(/Books)`: devuelve el contenido de la colección de ruta absoluta `/Books`.
- ✔ La consulta `doc(/Empresa.xml)`: devuelve el documento `/Empresa.xml` completo.

Otros **ejemplos de consultas XQuery basadas en expresiones XPath** son los siguientes:

- ✔ La consulta `collection(/Books)//book/title` devuelve los nodos `title` de todos los libros (`book`) de la colección `/Books`
- ✔ Si se utilizan espacios de nombres o `namespaces`, entonces la consulta anterior se redactaría de la siguiente forma: `declare namespace t = http://www.qizx.com/namespace/Tutorial; collection(/Books)//t:book/t:title`
- ✔ La consulta `doc(/Empresa.xml)//nombre` devuelve todos los nodos `nombre` del documento `/Empresa.xml`

En la siguiente enlace encontraras diferentes ejemplos de consultas XQuery: ejecutadas directamente sobre una sencilla BD XML denominada 'Cursillos'. Necesitarás descargar el archivo:

 [Acceso a los ejemplos.](#) (0.01 MB)



Para saber más

Te recomendamos que visites el siguiente enlace para profundizar en el concepto y uso de los `namespaces`.

[Espacios de nombres o namespaces en XML](#)

Autoevaluación

Señala si la siguiente afirmación es verdadera o falsa.

La expresión `collecion("/Cursos")//curso[aula=2]/profesor` devuelve los cursos y los profesores con cursos en el aula 2.

☐ Verdadero ☐ Falso

Falso

Es falso, devuelve todos los profesores que dan cursos en el aula 2.

3.5.4.- Expresiones FLWOR.

Los ejemplos de consultas XQuery vistos en el apartado anterior, son la manera más sencilla de realizar búsquedas y selecciones de nodos concretos en una BD XML. Pero existe otra manera mucho más potente de realizar este trabajo, mediante lo que se denomina consultas o **expresiones FLWOR** (leído como flower), siendo FLWOR las siglas de For, Let, Where, Order y Return.

Se trata de una **expresión que permite la unión de variables sobre conjuntos de nodos y la iteración sobre el resultado**. Las diferentes **cláusulas de una expresión FLWOR** son:

- ✓ **For.** Permite seleccionar los nodos que se quieren consultar, guardándose su valor en una variable (identificador que comienza por \$). Al conjunto de valores de la variable se le llama **tupla**.
- ✓ **Let.** (opcional). Asocia valores a variables.
- ✓ **Where** (opcional). Permite filtrar los resultados según una condición.
- ✓ **Order** (opcional). Permite ordenar la secuencia de valores o resultados.
- ✓ **Return.** Genera los valores de salida o devueltos.

En la siguiente imagen, puedes ver un ejemplo de una consulta (izquierda) donde aparecen las 5 cláusulas. La consulta devuelve los nombres de los cursos con 20 plazas, ordenados por nombre de curso (derecha).



Una expresión FLWOR vincula variables a valores con las cláusulas for y let y utiliza esos vínculos para crear nuevas estructuras de datos XML.

A continuación se muestra otro ejemplo de consulta XQuery. La siguiente consulta (izquierda) devuelve los nombres de los cursos con cuota mensual (derecha). Como **cuota** es un atributo del elemento **precio**, recuerda que se le antecede con un carácter @.



Para saber más

En el siguiente enlace tienes muchos ejemplos de consultas XQuery con expresiones FLWOR.



[Ejemplos XQuery con expresiones FLWOR](#)

3.5.5.- XQuery Update Facility.

El lenguaje **XQuery** solo proporciona expresiones para la realización de consultas sobre documentos XML, pero no su actualización (inserción, modificación o eliminación de nodos).

XQuery Update Facility es una extensión de XQuery que permite la actualización de documentos mediante las cláusulas `insert`, `delete`, `replace` y `rename`

El **funcionamiento de las cláusulas de XQuery Update** es el siguiente:

- ✔ **insert.** Permite la inserción de uno o varios nodos antes (`before`) o después (`after`) del nodo indicado. También se puede insertar al principio (`as first into`) o al final del documento (`as last into`).
- ✔ **delete.** Elimina uno o varios nodos del documento.
- ✔ **replace.** Tiene dos funciones:
 - ➡ Modificar el valor del nodo.
 - ➡ Modificar el nodo completo.
- ✔ **rename.** Renombra un nodo (elemento, atributo o instrucciones de proceso) sin afectar a su contenido.

A continuación te mostramos dos ejemplos sencillos, uno de `insert` y otro de `delete`:

- ✔ Eliminar la empresa de `id=2` en el documento `Empresa.xml`: `delete node doc(Empresa.xml)//empresa[@id=2].`
- ✔ Insertar el nodo `tipoAccesible/tipo` al final del documento `/Aulas/aula3` `insert node tipoAccesible/tipo as last into doc(Aulas/aula3.xml)//aula.`

En el siguiente enlace puedes otros ejemplos de actualizaciones usando XQuery Update:

 [Ejemplos de XQuery Update](#)



Para saber más

En el siguiente enlace puedes ver un documento tipo resumen con muchos de los aspectos tratados en esta unidad hasta el momento.

 [XML y Bases de Datos](#)



Autoevaluación

Señala si la siguiente afirmación es verdadera o falsa.

Mediante rename se puede sustituir el valor de un nodo.

☐ Verdadero ☐ Falso

Falso

Es falso, rename permite cambiar el nombre a un nodo, no sustituir su valor.

3.6.- Conexión a base de datos XML. API XML:DB.



Caso práctico

—¡Ha llegado la hora de manejar la base de datos desde Java! —exclama **Ana**, y añade— ¿queréis que os indique cómo crear colecciones y añadir documentos?

Juan contesta —Pues claro **Ana**, ¿sabes cual es el API principal que proporciona este gestor para trabajar con Java?

Ana contesta —De memoria no lo recuerdo, pero sí recuerdo que la propia distribución dispone de datos de prueba que podemos utilizar en nuestros primeros ejemplos.

María, que escucha con atención dice —¡Perfecto!, vamos a ello.



Los datos almacenados en las BD deben poder ser accesibles desde aplicaciones desarrolladas en diferentes lenguajes y, por este motivo, los SGBD se ven obligados a facilitar interfaces de programación para los lenguajes de programación más utilizados. Así, los fabricantes de SGBD, conocedores de la tecnología empleada en su producto, facilitan una API para permitir el acceso, desarrollada de la manera más eficiente posible para su producto, lo que conlleva la aparición de un problema: la API proporcionada para cada SGBD es propia y diferente de las API los otros SGBD y, en consecuencia, las aplicaciones desarrolladas quedan ligadas al SGBD y los programadores tienen que conocer un montón de API diferentes, tantas como número de SGBD diferentes deban enlazar.

Ante la anarquía de API existente para atacar los SGBD de un determinado tipo, suelen aparecer intentos para estandarizar el mecanismo y proporcionar una aPI estándar. **En el caso de las BD-XML nativas ha habido dos procesos de estandarización para el lenguaje Java, que han dado lugar a dos aPI estándares: XML: DB (también llamada XAPI) y XQueryAPI for Java (XQJ).**

3.6.1.- Introducción.

Esta API proporciona una interfaz común para bases de datos nativas o habilitadas para XML y admite el desarrollo de aplicaciones portátiles y reutilizables.

Los componentes básicos empleados por XML: DB API son controladores , colecciones , recursos y servicios .

- ✓ Los **controladores** son implementaciones de la interfaz de la base de datos que encapsulan la lógica de acceso a la base de datos para productos de bases de datos XML específicos. Son proporcionados por el proveedor del producto y deben registrarse con el gestor de la base de datos.
- ✓ Una **colección** es un contenedor jerárquico de recursos y otras subcolecciones. Actualmente, la API define dos recursos diferentes: XMLResource y BinaryResource. An XMLResource representa un documento XML o un fragmento de documento, seleccionado por una consulta XPath ejecutada previamente.
- ✓ Finalmente, se solicitan **servicios** para tareas especiales, como consultar una colección con XPath o administrar una colección.

La API XML:DB principalmente se basa en tres paquetes:

- ✓ **org.xmldb.api:** Interfaces, DatabaseManager
- ✓ **org.xmldb.api.base:** Interfaces, Collection, Configurable, Database, Resource, ResourceIterator, ResourceSet, Service , Classes, ErrorCodes , Exceptions, XMLDBException.
- ✓ **org.xmldb.api.modules:** Interfaces, BinaryResource, CollectionManagementService, TransactionService, XMLResource, XPathQueryService, XUpdateQueryService.

Hay varios ejemplos de XML: DB proporcionados en la pagina de eXist .

En el siguiente ejemplo sencillo , se recupera un documento del servidor eXist y se imprime en la salida estándar. El programa recibe dos argumentos `arg[0]` y `arg[1]`, nombre de la colección y nombre del fichero xml, respectivamente.

```
import org.xmldb.api.base.*;
import org.xmldb.api.modules.*;
import org.xmldb.api.*;
import javax.xml.transform.OutputKeys;
import org.exist.xmldb.EXistResource;
public class RetrieveExample {

    private static String URI = "xmldb:exist://localhost:8080/exist/xmlrpc";
    /**
     * args[0] Should be the name of the collection to access
     * args[1] Should be the name of the resource to read from the collection
     */
    public static void main(String args[]) throws Exception {

        final String driver = "org.exist.xmldb.DatabaseImpl";

        // initialize database driver
        Class cl = Class.forName(driver);
```



```

Database database = (Database) cl.newInstance();
database.setProperty("create-database", "true");
DatabaseManager.registerDatabase(database);

Collection col = null;
XMLResource res = null;
try {
    // get the collection
    col = DatabaseManager.getCollection(URI + args[0]);
    col.setProperty(OutputKeys.INDENT, "no");
    res = (XMLResource)col.getResource(args[1]);

    if(res == null) {
        System.out.println("document not found!");
    } else {
        System.out.println(res.getContent());
    }
} finally {
    //dont forget to clean up!

    if(res != null) {
        try { ((EXistResource)res).freeResources(); } catch(XMLDBException x
    }

    if(col != null) {
        try { col.close(); } catch(XMLDBException xe)
            {xe.printStackTrace();}
    }
}
}
}
}

```

En este ejemplo se registra el driver para eXist.

```

// initialize database driver
Class cl = Class.forName("org.exist.xmlldb.DatabaseImpl");
Database database = (Database)cl.newInstance();
database.setProperty("create-database", "true");
DatabaseManager.registerDatabase(database);

```

A continuación se obtiene el objeto colección (`Collection`) del gestor de la base de datos:

```
col = DatabaseManager.getCollection(URI + args[0]);
```

Una vez utilizado el recurso se debe cerrar:

```
col.close()
```

El método `getCollection()` espera una URI con el formato siguiente:

```
xmlldb:[DATABASE-ID]://[HOST-ADDRESS]/db/collection
```

Debido a que se puede registrar más de un controlador de base de datos, se requiere la primera parte del URI (`xmlldb:exist`) para determinar qué clase de controlador se debe usar. El objeto de `DatabaseManager` utiliza la identificación de la base de datos para seleccionar el controlador correcto de su lista de controladores disponibles.

La parte final del URI identifica la ruta a la colección y, opcionalmente, la dirección de host del servidor de base de datos en la red. Internamente, eXist usa dos implementaciones de controladores diferentes: la primera habla con un motor de base de datos remoto que usa llamadas XML-RPC, la segunda tiene acceso directo a una instancia local de eXist-db.


La colección raíz siempre se identifica con `/db`.

Por ejemplo:

```
xmlldb:exist://localhost:8080/exist/xmlrpc/db/shakespeare/plays
```

Si omitimos la dirección de host el driver XML:DB tratará de conectarse a una instancia local de la base de datos. Por ejemplo:

```
xmlldb:exist:///db/shakespeare/plays
```

En este caso, debemos decirle al driver que cree una instancia a la base de datos si no hay ninguna creada. Esto se hace poniendo la propiedad `create-database` de la clase `Database` a `true`. Para más información del uso embebido de eXist-db consulta  [deployment guide](#).

El método `setProperty` se utiliza para establecer parámetros específicos de la base de datos.

La llamada `col.getResource()` recupera el documento, que se devuelve como un archivo `XMLResource`. Todos los recursos tienen un método `getContent()`, que devuelve el contenido del recurso, dependiendo de su tipo. En este caso recuperamos el contenido como tipo `String`.

3.6.2.- Consultas sobre colecciones y documentos.

Para consultar el repositorio, podemos usar el estándar `XPathQueryService` o la clase `XQueryService` de `eXist`. La API XML: DB define diferentes tipos de servicios, que pueden ser proporcionados o no por la base de datos. El método `getService` de clase `Collection` llama a un servicio si está disponible. El método espera que el nombre del servicio sea el primer parámetro, y su versión (como una cadena) como el segundo.

El siguiente es un ejemplo del uso de XML: DB API para ejecutar una consulta de base de datos:

El siguiente ejemplo ejecuta una consulta XPath que se pasa al programa como parámetro.

```
import org.xmldb.api.modules.*;
import org.xmldb.api.*;
import org.exist.xmldb.EXistResource;
public class XPathExample {
    private static String URI = "xmldb:exist://localhost:8080/exist/xmlrpc";
    /**
     * args[0] Should be the name of the collection to access
     * args[1] Should be the XPath expression to execute
     */
    public static void main(String args[]) throws Exception {
        final String driver = "org.exist.xmldb.DatabaseImpl";
        // initialize database driver
        Class cl = Class.forName(driver);
        Database database = (Database) cl.newInstance();
        database.setProperty("create-database", "true");
        DatabaseManager.registerDatabase(database);

        Collection col = null;
        try {
            col = DatabaseManager.getCollection(URI + args[0]);
            XPathQueryService xpqs = (XPathQueryService)col.getService("XPathQuerySe
            xpqs.setProperty("indent", "yes");
            ResourceSet result = xpqs.query(args[1]);
            ResourceIterator i = result.getIterator();
            Resource res = null;
            while(i.hasMoreResources()) {
                try {
                    res = i.nextResource();
                    System.out.println(res.getContent());
                } finally {
                    //dont forget to cleanup resources
                    try { ((EXistResource)res).freeResources(); } catch(XMLDBExcepti
                }
            }
        } finally {
            //dont forget to cleanup
        }
    }
}
```

```

        if(col != null) {
            try { col.close(); } catch(XMLDBException xe) {xe.printStackTrace();}
        }
    }
}

```

Para ejecutar la consulta, se llama al método `service.query (xpath)`. Este método devuelve un `ResourceSet`, que contiene los recursos encontrados por la consulta. `ResourceSet.getIterator ()` nos da un iterador sobre estos recursos. Cada recurso contiene un único fragmento o valor de documento, seleccionado por la expresión XPath.

Internamente, eXist no distingue entre expresiones XPath y XQuery. Por lo tanto, `XQueryService` se asigna a la misma clase de implementación que `XPathQueryService`. Sin embargo, proporciona algunos métodos adicionales. Lo más importante es que cuando se habla con una base de datos integrada, `XQueryService` permite que la expresión XQuery se compile en una representación interna, que luego se puede reutilizar. Con la compilación, el código de ejemplo anterior se vería como sigue:

```

import org.xmldb.api.base.*;
import org.xmldb.api.modules.*;
import org.xmldb.api.*;
import org.exist.xmldb.EXistResource;
public class XQueryExample {
    private static String URI = "xmldb:exist://localhost:8080/exist/xmlrpc";
    /**
     * args[0] Should be the name of the collection to access
     * args[1] Should be the XQuery to execute
     */
    public static void main(String args[]) throws Exception {
        final String driver = "org.exist.xmldb.DatabaseImpl";
        // initialize database driver
        Class cl = Class.forName(driver);
        Database database = (Database) cl.newInstance();
        database.setProperty("create-database", "true");
        DatabaseManager.registerDatabase(database);
        Collection col = null;
        try {
            col = DatabaseManager.getCollection(URI + args[0]);
            XQueryService xqs = (XQueryService) col.getService("XQueryService", "1.0");
            xqs.setProperty("indent", "yes");

            CompiledExpression compiled = xqs.compile(args[1]);
            ResourceSet result = xqs.execute(compiled);
            ResourceIterator i = result.getIterator();
            Resource res = null;
            while(i.hasMoreResources()) {
                try {
                    res = i.nextResource();
                    System.out.println(res.getContent());
                } finally {
                    //dont forget to cleanup resources
                    try { ((EXistResource)res).freeResources(); } catch(XMLDBExcepti

```

```

        }
    }
} finally {
    //dont forget to cleanup
    if(col != null) {
        try { col.close(); } catch(XMLDBException xe) {xe.printStackTrace();}
    }
}
}
}
}

```

El servidor XML-RPC almacena automáticamente en la memoria caché las expresiones compiladas, por lo que la compilación a través del controlador remoto no produce ningún efecto si la expresión ya está almacenada en la memoria caché.

A continuación, nos gustaría almacenar un nuevo documento en el repositorio. Esto se hace creando un nuevo XMLResource, asignándole el contenido del nuevo documento y llamando al método storeResource de la clase Colección.

Primero, un nuevo recurso es creado por el método `Collection.createResource ()`, y espera dos parámetros: el id y el tipo de recurso que se crea. Si el parámetro id es nulo, se generará automáticamente una identificación de recurso única.

En algunos casos, es posible que la colección aún no exista, por lo que debemos crearla. Para crear una nueva colección, llame al método `createCollection` del servicio `CollectionManagementService`. En el siguiente ejemplo, simplemente comenzamos en el objeto de colección raíz para obtener el servicio `CollectionManagementService`.

```

import java.io.File;
import org.exist.xmldb.EXistResource;
import org.xmldb.api.base.*;
import org.xmldb.api.modules.*;
import org.xmldb.api.*;
public class StoreExample {

    private static String URI = "xmldb:exist://localhost:8080/exist/xmlrpc";
    /**
     * args[0] Should be the name of the collection to access
     * args[1] Should be the name of the file to read and store in the collection
     */
    public static void main(String args[]) throws Exception {
        if(args.length < 2) {
            System.out.println("usage: StoreExample collection-path document");
            System.exit(1);
        }
        final String driver = "org.exist.xmldb.DatabaseImpl";

        // initialize database driver
        Class cl = Class.forName(driver);
        Database database = (Database) cl.newInstance();
        database.setProperty("create-database", "true");
        DatabaseManager.registerDatabase(database);
        Collection col = null;
    }
}

```

```

XMLResource res = null;
try {
    col = getOrCreateCollection(args[0]);

    // create new XMLResource; an id will be assigned to the new resource
    res = (XMLResource)col.createResource(null, "XMLResource");
    File f = new File(args[1]);
    if(!f.canRead()) {
        System.out.println("cannot read file " + args[1]);
        return;
    }
    res.setContent(f);
    System.out.print("storing document " + res.getId() + "...");
    col.storeResource(res);
    System.out.println("ok.");
} finally {
    //dont forget to cleanup
    if(res != null) {
        try { ((EXistResource)res).freeResources(); } catch(XMLDBException x
    }
    if(col != null) {
        try { col.close(); } catch(XMLDBException xe) {xe.printStackTrace();
    }
}

private static Collection getOrCreateCollection(String collectionUri) throws
XMLDBException {
    return getOrCreateCollection(collectionUri, 0);
}

private static Collection getOrCreateCollection(String collectionUri, int pathSe
Collection col = DatabaseManager.getCollection(URI + collectionUri);
if(col == null) {
    if(collectionUri.startsWith("/")) {
        collectionUri = collectionUri.substring(1);
    }
    String pathSegments[] = collectionUri.split("/");
    if(pathSegments.length > 0) {
        StringBuilder path = new StringBuilder();
        for(int i = 0; i <= pathSegmentOffset; i++) {
            path.append("/") + pathSegments[i];
        }
        Collection start = DatabaseManager.getCollection(URI + path);
        if(start == null) {
            //collection does not exist, so create
            String parentPath = path.substring(0, path.lastIndexOf("/"));
            Collection parent = DatabaseManager.getCollection(URI + parentPa
            CollectionManagementService mgt = (CollectionManagementService)
            col = mgt.createCollection(pathSegments[pathSegmentOffset]);
            col.close();
            parent.close();
        } else {
            start.close();
        }
    }
    return getOrCreateCollection(collectionUri, ++pathSegmentOffset);
} else {
    return col;
}

```

```
}  
}  
}
```


El método `XMLResource.setContent ()` carga el objeto Java como que se pasa como parámetro. El controlador `eXist` comprueba si el objeto es un archivo. De lo contrario, el objeto se transforma en una cadena llamando al método `toString ()` del objeto. Pasar un archivo tiene una gran ventaja: si la base de datos se ejecuta en modo incrustado, el archivo se pasará directamente al indexador. De esta forma, el contenido del archivo no tiene que cargarse en la memoria. Esto es útil si los archivos son muy grandes.



Para saber más

Utiliza estas dos enlaces para estudiar como conectar java con eXist:

 [Java-eXist1](#) y  [Java -eXist2](#).

El proceso de conexion a la BD paso a paso, se explica en el  [enlace](#).

3.6.3.- Crear y borrar colecciones.

La creación y eliminación de colecciones son proporcionadas por la interfaz `CollectionManagementService` con los métodos:

- ✔ `createCollection(String "nombre_coleccion")`, Para crear una nueva colección en la base de datos.
- ✔ `removeCollection(String "nombre_coleccion")`, Para eliminar una colección.

En ambos métodos, el nombre de la colección para crear o para eliminar es relativo a la colección desde la que se ha obtenido el objeto `CollectionManagementService` que se utiliza para invocar los métodos, el que se obtiene con algún los dos métodos siguientes de la interfaz `Collection`:

1. `getServices()`, Para obtener una lista de todos los servicios proporcionados por la colección.
2. `getService()`, Para obtener un objeto `Service` de entre los servicios que proporciona la colección, como un objeto `CollectionManagementService`.

El método `createCollection()` no genera ningún error en caso de que ya exista una colección con el mismo nombre que la que se pretende crear; se mantiene la colección existente.

El método `removeCollection()` elimina la colección, con todo su contenido.

3.7.- Conexión a base de datos XML. API XJQ.

Estas Bases de Datos son una propuesta estandarizada de la interfaz Java para el acceso a BBDD XML nativas basadas en el modelo de datos XQuery. El objetivo de éstas es conseguir un método fácil y estable de acceso. Es un API bastante similar al JDBC de las bases de datos relacionales.

3.7.1.- Introducción


XQuery API for Java (XQJ) es una interfaz de programación de aplicaciones Java pensada para utilizar el lenguaje XQuery para obtener información de BD-XML nativas, de manera parecida a como JDBC es una API pensada para utilizar el lenguaje SQL para acceder a BDR.

XQuery API for Java (XQJ) es una interfaz de programación de aplicaciones Java pensada para utilizar el lenguaje XQuery para obtener información de BD-XML nativas, de manera parecida a como JDBC es una API pensada para utilizar el lenguaje SQL para acceder a BDR .

XQJ nació en 2003 y su versión definitiva ha sido publicada en 2009. También es conocida como JSR 225 ya que ha sido diseñada como un proyecto JCP (Java Community Process).

Numerosos SGBD-XML nativas facilitan la conectividad desde Java mediante esta XQJ, por lo que podemos desarrollar aplicaciones que accedan a SGBD-XML nativas vía XQJ con la única particularidad de tener que utilizar la implementación del API que facilita cada SGBD.

Para ello necesitamos tener instalados los eXist-db y disponer de la implementación XQJ para este SGBD.

XQJ, está constituida por un gran número de interfaces y unas pocas clases. Cada SGBD debe proveer las clases que implementan las interfaces que dictamina el API . En este enlace puedes descargar la  [API correspondiente a eXist-db](#).

Sera necesario agregar las librerías descargadas e incluirlas en el proyecto.

3.7.2.- Conexión a la BD.

Las aplicaciones que acceden a BD necesitan, como primer paso para poder gestionar los datos de la BD, establecer la conexión con la BD a gestionar. La API XQJ facilita dos interfaces apropiadas para lograr este objetivo:

1. **XQDataSource**, Fábrica para obtener objetos **XQConnection**
2. **XQConnection**, Para referenciar conexiones (sesiones) con un SGBD específico. Toda conexión se logra, forzosamente, a través de un objeto **XQDataSource**

Así, para obtener una conexión, hay que seguir el siguiente esquema:

```
XQDataSource xqs = new ExistXQDataSource();
XQConnection con = xqs.getConnection ( lista_parametros ) ;
```

Una vez se dispone del objeto **XQDataSource** ya estamos en condiciones de crear un objeto **XQConnection** para establecer la sesión de trabajo con el SGBD y eso lo conseguimos con el método **getConnection()** de la interfaz **XQDataSource**.

La API XQJ facilita sobrecargas del método **getConnection()**:

- ✔ **XQConnection**
`getConnection (String username, String passwd)` throws **XQException**.
- ✔ **XQConnection** `getConnection ()` throws **XQException**.

La interfaz **setProperty()** nos permite definir una serie de propiedades antes de intentar establecer la conexión con el método **getConnection()**:

```
void setProperty ( String name, String value ) throws XQException;
```

Entre las propiedades que permite establecer el método **setProperty()** encontramos el nombre de la máquina, el puerto, el nombre de la base de datos, el usuario y la contraseña.

El método **XQDataSource.getConnection()** devuelve, si todo es correcto, un objeto que implementa la interfaz **XQConnection**, el cual utilizaremos para ejecutar cualquier acción **XQuery** o **Update** sobre la BD. Al finalizar el programa hay que recordar siempre de cerrar la conexión con el método **XQConnection.close()**, a fin de liberar todos los recursos asignados por el sistema operativo para mantener la conexión.

La interfaz **XQConnection** facilita métodos para crear expresiones sobre las que ejecutaremos sentencias **XQuery / Update**, y los métodos para validar (**commit()**) o deshacer (**rollback()**) los cambios llevados a cabo durante una transacción.

Ejemplo de conexion :

```

import javax.xml.xquery.XQConnection;
import javax.xml.xquery.XQDataSource;
import javax.xml.xquery.XQException;
import javax.xml.xquery.XQPreparedExpression;
import javax.xml.xquery.XQSequence;
import net.xqj.exist.ExistXQDataSource;
public class HelloWorld {
    public static void main(String[] args) throws XQException {
        XQDataSource ds = new ExistXQDataSource();
        ds.setProperty("serverName", "localhost");
        ds.setProperty("port", "8083");
        XQConnection con = ds.getConnection();
        String query = "<hello-world>{1 + 1}</hello-world>";
        XQPreparedExpression expr = con.prepareExpression(query);
        XQSequence result = expr.executeQuery();
        // prints "<hello-world>2</hello-world>"
        System.out.println(result.getSequenceAsString(null));
        result.close();
        expr.close();
        con.close();
    }
}

```

En eXist-db, el hecho de que el método **XQDataSource.getConnection()** no levante una excepción, no es garantía de que la conexión se haya establecido correctamente.

En el ejemplo anterior la conexión se hace con un usuario guest que es la opción por defecto si no se especifica usuario y contraseña. Para conectarse con un usuario se utilizan el método `setProperty()`:

```

ds.setProperty("user" , "admin");
ds.setProperty("password" , "admin");

```

Por defecto, una conexión opera en modo `auto-commit`, lo que significa que cada instrucción Update es ejecutada y validada en una transacción individual. Esta forma de trabajar se puede desactivar con el método `XQConnection.setAutoCommit()`, y en tal situación, la transacción finalizará efectuando una llamada al método `commit()` o `rollback()`, dando lugar al inicio de una nueva transacción. No hay, pues, una instrucción específica para indicar el inicio de transacción.



Para saber más

Consulta la  [API XQJ de eXist](#) para obtener más información.

3.7.3.- Ejecución de consultas XQuery

La API XQJ facilita los dos mecanismos a través de las interfaces:

- ✔ **XQExpression**, Para la ejecución inmediata de sentencias.
- ✔ **XQPreparedExpression**, Para la ejecución de sentencias parametrizadas.

Para ejecutar una sentencia **XQExpression**, crearemos un objeto XQExpression a partir del método `XQConnection.createExpression()`. Disponemos de dos sobrecargas de este método:

```
XQExpression createExpression throws XQException ;  
XQExpression createExpression ( XQStaticContext properties )throws XQException ;
```

En ambas sobrecargas obtiene un objeto XQExpression que podremos utilizar para ejecutar sentencias XQuery / Update de manera inmediata. La segunda sobrecarga permite indicar las propiedades del contexto estático que se tendrán en cuenta al evaluar la expresión, mientras que la primera sobrecarga toma como contexto estático el asociado a la conexión.

La interfaz XQStaticContext provee un conjunto de métodos get y set para recuperar y establecer las propiedades de un contexto estático. La interfaz XQConnection facilita los métodos `getStaticContext()` y `setStaticContext()` para recuperar y establecer el contexto estático de la conexión.

Una vez tengamos el objeto XQExpression, podremos a través de él ejecutar consultas y otras órdenes de manera inmediata. Recordemos que en XQuery hay que distinguir, como en SQL, las sentencias "consulta" que pueden devolver un conjunto de resultados que habrá que procesar, de las sentencias "no consulta" que permiten ejecutar una orden (inserción, eliminación o actualización e incluso órdenes específicas del SGBD), de la que se nos informa, como mucho, del éxito o fracaso de su ejecución. De ahí que la interfaz XQExpression distingue los métodos `executeQuery` para las "consultas" de los métodos `executeCommand` para las "no consultas":

```
void executeCommand ( java.lang.String cmd ) throws XQException ;  
void executeCommand ( java.io.Reader cmd ) throws XQException ;  
XQResultSequence executeQuery ( java.Lang.String query ) throws XQException ;  
XQResultSequence executeQuery ( java.io.Reader query ) throws XQException ;  
XQResultSequence executeQuery ( java.io.InputStream query ) throws XQException ;
```

Además de los métodos anteriores, disponemos del método `close()` para cerrar la expresión, liberando todos los recursos asociados cuando ya no sea necesaria. La ejecución del método `close()` sobre la conexión cierra todas las expresiones definidas sobre ella. También disponemos del método `isClosed()` para poder averiguar si una expresión está abierta o cerrada:

```
void close throws XQException ;  
boolean isClosed;
```

Fijémonos que el método `executeQuery()` devuelve, si todo va bien, un objeto `XQResultSequence`, interfaz que deriva de la interfaz `XQSequence`, la cual nos facilita un conjunto de métodos para evaluar la secuencia de resultados obtenidos con el método `executeQuery()`:

La interfaz `XQResultSequence` representa una secuencia de elementos siguiendo la definición de XDM (XQuery 1.0 and XPath 2.0 Data Model, de W3C). Un vistazo a la documentación de esta interfaz nos muestra que disponemos de métodos para:

- ✔ Procesar sus elementos: `next()`, `previous()`, `getItem()`, `getPosition()`, `count()`, `first()`, `last()`, `isFirst()`, `isLast()`, `isBeforeFirst()`, `isAfterLast()` y un conjunto de métodos `get` para obtener el contenido de un elemento en el formato adecuado (`getBoolean()`, `getByte()`, `getDouble()`...).
- ✔ Obtener una versión seriada de la secuencia como un objeto `String` (método `getSequenceAsString()`).
- ✔ Obtener una versión seriada de la secuencia (método `getSequenceAsStream()`) como un objeto `XMLStreamReader` (interfaz del API Stax de Java, que permite la iteración, hacia delante, de un documento XML en modo lectura, utilizando los métodos `next()` y `hasNext()`).

Según sea la gestión que tengamos que efectuar, utilizaremos unos u otros métodos. Así, ante un programa de sentencia abierta (en el que el usuario pueda introducir la sentencia a ejecutar), sólo podremos pensar en mostrar el resultado a través de la versión seriada hacia `String` o vía objeto `Transformer` a partir de la seriación hacia `XMLStreamReader`. Y ante un programa de sentencia cerrada (sentencia perfectamente conocida en escribir el programa), dado que se conoce la forma de la respuesta, podemos pensar en efectuar un tratamiento específico.

4.- Bases de Datos MongoDB

MongoDB (que proviene de «humongous») es la base de datos NoSQL líder y permite a las empresas ser más ágiles y escalables. Organizaciones de todos los tamaños están usando MongoDB para crear nuevos tipos de aplicaciones, mejorar la experiencia del cliente, acelerar el tiempo de comercialización y reducir costes.

Es una base de datos ágil que permite a los esquemas cambiar rápidamente cuando las aplicaciones evolucionan. MongoDB ha sido creado para brindar escalabilidad con un elevado rendimiento, tanto para lectura como para escritura, potenciando la computación en memoria (in-memory). La replicación nativa de MongoDB y la tolerancia a fallos automática ofrece fiabilidad a nivel empresarial y flexibilidad operativa.

Las suscripciones de MongoDB ofrecen un servicio de asistencia técnica profesional, licencias comerciales y acceso a características de software de MongoDB Enterprise. Las suscripciones no solo ayudan a los clientes a lograr una infraestructura de TI estable, escalable y segura, sino también a alcanzar sus objetivos empresariales más amplios, tales como reducir los costes, acelerar el tiempo de comercialización y disminuir los riesgos.

MongoDB Enterprise ofrece seguridad avanzada, monitorización on-premises, soporte SNMP, certificaciones de SO y mucho más. El servicio de gestión de MongoDB (MMS) ofrece funcionalidad de monitorización y respaldo en la nube o bien on-premises como parte de MongoDB Enterprise.

4.1.- Instalación, configuración y manejo de una base de datos Mongo.

En el siguiente enlace al documento propiedad de MongoDB, podrás aprender a instalar, configurar y manejar de forma sencilla la base de datos documental MongoDB:

 [Descarga del documento](#) (pdf - 627439 B)

4.2.- Conexión desde JAVA a MongoDB

Las ordenes Java necesarias para conectar a una base de datos MongoDB son:

- ✓ Conectar con la base de datos
`MongoClient clienteMongo = MongoClient.create();`
- ✓ Activar una base de datos
`MongoDatabase baseDatos = clienteMongo.getDatabase("centro");`
- ✓ Obtener un listado de las colecciones
`baseDatos.listCollectionNames();`
- ✓ Cerrar la conexión
`clienteMongo.close();`

En el siguiente ejemplo se muestra la conexión a una base de datos Mongo llamada centro:

```
package com.jcg.java.mongodb;
import com.mongodb.client.MongoClients;
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoDatabase;
public class JavaMongoDbFp {
    public static void main( String args[] ) {
        MongoClient clienteMongo = MongoClient.create();
        MongoDatabase baseDatos = clienteMongo.getDatabase("centro");
        for (String name : baseDatos.listCollectionNames()) {
            System.out.println(name);
        }
        clienteMongo.close();
    }
}
```

- ✓ Seleccionar una colección
`MongoCollection coleccion = database.getCollection("personas");`
- ✓ Buscar el primer documento de una colección
`Document primero = coleccion.find().first();`

En el siguiente ejemplo se muestra como seleccionar una colección y como buscar el primer documento de la misma:

```
package com.jcg.java.mongodb;
import com.mongodb.*;
import com.mongodb.client.MongoClients;
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.model.Projections;
import com.mongodb.client.model.Filters;
import static com.mongodb.client.model.Filters.*;
import static com.mongodb.client.model.Projections.*;
import com.mongodb.client.model.Sorts;
```

```
import java.util.Arrays;
import org.bson.Document;
public class JavaMongoDbFp {
    public static void main( String args[] ) {
        MongoClient clienteMongo = MongoClient.create();
        MongoDBDatabase baseDatos = clienteMongo.getDatabase("centro");
        MongoCollection<Document> coleccion =
        baseDatos.getCollection("personas");
        Document primero = coleccion.find().first();
        System.out.println(primero.toJson());
        clienteMongo.close();
    }
}
```

✔ Realizar una consulta

```
FindIterable<Document> iterDoc = coleccion.find(new Document("tipo", "Alumno"));
```

En el siguiente ejemplo se muestra como realizar una consulta a una base de datos dada.











```
package com.jcg.java.mongodbfp;
import com.mongodb.*;
import com.mongodb.client.FindIterable;
import com.mongodb.client.MongoClients;
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.model.Projections;
import com.mongodb.client.model.Filters;
import static com.mongodb.client.model.Filters.*;
import static com.mongodb.client.model.Projections.*;
import com.mongodb.client.model.Sorts;
import java.util.Arrays;
import java.util.Iterator;
import org.bson.Document;
public class JavaMongoDbFp {
    public static void main( String args[] ) {
        MongoClient clienteMongo = MongoClient.create();
        MongoDBDatabase baseDatos = clienteMongo.getDatabase("centro");
        MongoCollection<Document> coleccion = baseDatos.getCollection("personas");
        //Obtener un objeto iterable de tipo cursor
        FindIterable<Document> iterDoc = coleccion.find(new Document("tipo", "Alum
        int i = 1;
        //Iniciamos el iterador
        Iterator it = iterDoc.iterator();
        while (it.hasNext()) {
            System.out.println(it.next());
            i++;}
        clienteMongo.close();
    }
}
```

En el siguiente ejemplo se realiza una inserción de un documento:

```
package com.jcg.java.mongodbfp;
import com.mongodb.*;
import com.mongodb.client.FindIterable;
import com.mongodb.client.MongoClients;
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.model.Projections;
import com.mongodb.client.model.Filters;
import static com.mongodb.client.model.Filters.*;
import static com.mongodb.client.model.Projections.*;
import com.mongodb.client.model.Sorts;
import java.util.Arrays;
import java.util.Iterator;
import org.bson.Document;
public class JavaMongoDbFp {
    public static void main( String args[] ) {
        MongoClient clienteMongo = MongoClients.create();
        MongoDatabase baseDatos = clienteMongo.getDatabase("centro");
        MongoCollection<Document> coleccion =
            baseDatos.getCollection("personas");
        Document persona = new Document("nombre", "Sergio")
            .append("tipo", "Profesor")
            .append("datos_personales", new Document("telefono", "123456789")
                .append("email", "su@correo.es"))
            .append("asignaturas", Arrays.asList("Física", "Matemáticas"));
        coleccion.insertOne(persona);
        clienteMongo.close();
    }
}
```

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: Isabel M. Cruz Granados. Licencia: Uso Educativo-nc. Procedencia: Captura de pantalla del programa QizxStudio de Qizx Free Engine Edition.		Autoría: Isabel M. Cruz Granados. Licencia: Uso Educativo-nc. Procedencia: Captura de pantalla del programa QizxStudio de Qizx Free Engine Edition.
	Autoría: Isabel M. Cruz Granados. Licencia: Uso Educativo-nc. Procedencia: Captura de pantalla del programa QizxStudio de Qizx Free Engine Edition.		Autoría: Isabel M. Cruz Granados. Licencia: Uso Educativo-nc. Procedencia: Captura de pantalla del programa QizxStudio de Qizx Free Engine Edition.
	Autoría: Emma Gracia Lor. Licencia: CC-by-nc-sa. Procedencia: http://www.flickr.com/photos/paideiaeducacion/5102909303/		Autoría: fsse8info. Licencia: CC BY-SA. Procedencia: http://www.flickr.com/photos/fsse-info/4194980532/
	Autoría: Howard Stanbury. Licencia: CC BY-NC-SA. Procedencia: http://www.flickr.com/photos/stanbury/5852623652/		Autoría: Justin See. Licencia: CC BY. Procedencia: http://www.flickr.com/photos/koalazymonkey/3651288422/
	Autoría: Adam Prince. Licencia: CC BY-NC-SA. Procedencia: http://www.flickr.com/photos/adam_prince/3908404628/		Autoría: Tantek Çelik. Licencia: CC BY-NC. Procedencia: http://www.flickr.com/photos/tantek/539497686/

