1. INTRODUCCIÓN

Las bases de datos NoSQL son aquellas que no siguen el modelo clásico de sistema de gestión de bases de datos relacionales (RDBMS). Se caracterizan porque no usan el SQL como lenguaje principal de consultas, y además, en el almacenamiento de los datos no se utilizan estructuras fijas almacenamiento.

El término NoSQL surge con la llegada de la web 2.0, ya que hasta ese momento sólo subían contenido a la red aquellas empresas que tenían un portal, pero con la llegada de aplicaciones como Facebook, Twitter o Youtube, en las que el usuario interactúa en la web, cualquier usuario podía subir contenido, provocando así un crecimiento exponencial de los datos.

Surgen así los problemas para gestionar y acceder a toda esa información almacenada en bases de datos relacionales. Una de las soluciones, propuestas por las empresas para solucionar estos problemas de accesibilidad, fue la de utilizar más máquinas, sin embargo, era una solución cara y no terminaba con el problema.

La otra solución fue la de crear nuevos sistemas gestores de bases de datos pensados para un uso específico, que con el paso del tiempo han dado lugar a soluciones robustas, apareciendo así el movimiento NoSQL.

Así pues, hablar de bases de datos NoSQL es hablar de estructuras que nos permiten almacenar información en aquellas situaciones en las que las bases de datos relacionales generan ciertos problemas, debido principalmente a problemas de escalabilidad y rendimiento de las bases de datos relaciones, donde se dan cita miles de usuarios concurrentes y millones de consultas diarias. Por tanto, las beses de datos NoSQL intentan resolver problemas de almacenamiento masivo, alto desempeño, procesamiento masivo de transacciones (sitios con alto tránsito) y, en general, ser alternativas NoSQL a problemas de persistencia y almacenamiento masivo (voluminoso) de información para las organizaciones.

2. CARACTERÍSTICAS

A diferencia de las bases de datos relacionales (centradas en los datos), las **bases de datos nativas XML** (*NXD, Native XML Database*) no guardan tablas con filas y columnas, sino que almacenan documentos XML. Son bases de datos centradas en documentos y la unidad mínima de almacenamiento es el documento XML.

Una base de datos nativa XML se puede definir como un sistema de gestión de información que cumple con lo siguiente:

- **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, 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.
- Tiene una relación transparente con el mecanismo de almacenamiento, que debe incorporar las **características ACID** de cualquier SGBD: atomicidad (*atomicity*), consistencia (*consistency*), aislamiento (*isolation*) y durabilidad (*durability*).
- Permite las **tecnologías de consulta y transformación propias de XML** (Xpath, XQuery, XSLT) como vehículo principal de acceso y tratamiento.

Lo que realmente cambia en estas bases de datos 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 bases de datos 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.
- **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 de copias de seguridad, como cualquier otro sistema de bases de datos.

La utilización de bases de datos nativas XML puede resultar imprescindible en las siguientes situaciones:

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

Las bases de datos nativas XML presentan las siguientes ventajas:

- Ofrecen un acceso y almacenamiento de información ya en formato XML, sin necesidad de incorporar código adicional
- La mayoría incorpora un motor de búsqueda de alto rendimiento
- Es muy sencillo añadir nuevos documentos XML al repositorio.
- Se pueden almacenar datos heterogéneos.

Por el contrario, las bases de datos nativas XML tienen los siguientes inconvenientes:

- Puede resultar difícil indexar documentos para realizar búsquedas.
- No suelen ofrecer funciones para la agregación (crucial para el procesamiento de transacciones en línea). En muchos casos hay que reintroducir todo el documento para modificar una sóla línea.

- Se suele almacenar la información en XML como un documento o como un conjunto de nodos, por lo que su síntesis para formar nuevas estructuras sobre la marcha puede resultar complicada y lenta.

En la actualidad la mayoría de SGBD incorporan mecanismos para extraer y almacenar datos en formato XML.

3. SISTEMAS GESTORES

Los **sistemas gestores XML nativos** son una alternativa natural para aplicaciones que trabajan con documentos XML en todos sus niveles (interfaz, gestión y almacén de datos). Al igual que ocurre con otros sistemas gestores, los sistemas gestores XML nativos soportan transacciones, acceso multiusuario, lenguajes de consulta, índices, etc.

En general, todos los sistemas gestores XML nativos tienen como misión principal el almacenamiento y la gestión de documentos XML y para ello implementan las siguientes características:

- XML permite asignar una estructura a documentos XML mediante esquemas XML (XML Schema) o DTD. Por lo tanto, los sistemas nativos que permiten asociar esquemas a documentos XML deben permitir comprobar la adecuación de los datos a esos esquemas (validación).
- Los sistemas nativos deben permitir almacenar y recuperar documentos XML de acuerdo con la especificación XML. Como mínimo, el modelo debe incluir elementos y atributos, y respetar el orden de los elementos en el documento y la estructura de árbol.
- Para una recuperación eficiente de los datos, éstos deben estar indexados. El modo de indexación en este tipo de sistemas no es igual que el seguido en sistemas relacionales, sino que debe ser adecuado y concreto para la estructura de datos XML.
- Como sistema gestor, cualquier sistema XML nativo debe soportar concurrencia y seguridad.
- Por último, los sistemas XML nativos deben dar soporte a toda la tecnología asociada a XML. Algunos ejemplos de esta tecnología son XPath, XQuery y XSLT.

Algunos ejemplos de sistemas gestores XML nativos son:

- **Sistemas privativos.** Generalmente ofrecen versiones de prueba con limitaciones de tiempo de uso y de funcionalidades, como por ejemplo, TaminoXML Server y TEXTML Server. Entre estos, destacan:
  1) Tamino XML Server (de SoftwareAG): http://www.softwareag.com/corporate/products/az/webmethods/default.asp
  2) Qizx (de Qualcomm): https://aws.amazon.com/marketplace/pp/B00RM04ZFM
  3) TEXTML Server (de Ixiasoft): http://www.ixiasoft.com/en/products/textml-server/overview/
- **Sistemas bajo licencias de software libre.** A veces también ofrecen dualidad de licencia (comercial y libre). Entre estos, destacan:
  1) eXist DB: http://exist-db.org/exist/apps/homepage/index.html
  2) BaseX: http://basex.org/
  3) Berkeley DB XML (de Oracle Corporation): https://www.oracle.com/database/berkeley-db/xml.html

TaminoXML Server es un producto comercial de alto rendimiento y disponibilidad, además de ser uno de los primeros SGBD XML nativos disponibles. Ofrece una alternativa pura XML para almacenar y recuperar documentos. Utiliza una estructura interna propia sin necesidad de otros sistemas gestores subyacentes (por ejemplo, relacionales) que requieran una transformación.

Sus principales 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. Por un lado, se define una estructura para almacenar los documentos, y por otro, los índices asociados que potenciarán la recuperación de datos de esos documentos.
- Permite almacenar documentos que siguen un esquema XML o DTD (validados) o que no lo siguen (bien formados). Esta opción es menos potente pero da más flexibilidad.
- Los documentos se organizan en colecciones. Una colección es un conjunto de documentos, de modo que forma una estructura de árbol donde cada documento pertenece a una única colección. Además de almacenar documentos XML, dentro de una colección se pueden almacenar otros tipos de documentos de texto o binarios (no XML).
- Los elementos de configuración del sistema son también documentos XML almacenados en la colección *system*.
- Soporta diferentes maneras de acceder a los datos: servicios SOAP, APIs para Java (XMLDB, XQJ) y APIs para .NET. Soporta lenguajes de consultas como XQuery.

eXist DB 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.

Sus principales características son las siguientes:

- Los documentos se almacenan en una jerarquía de colecciones. Cada documento está en una colección. Dentro de una colección, también se permite almacenar documentos de cualquier tipo.
- A diferencia de Tamino, en eXist los documentos no tienen que tener un esquema XML o DTD asociado, por lo que solo ofrece funcionalidad para documentos XML bien formados, sin atender a si siguen o no una estructura (validación).
- Proporciona diferentes maneras de acceder a los datos. Se puede utilizar en un servidor Java (J2EE) con servicios XML-RPC, SOAP y WebDAV.
- Soporta el lenguaje de consulta XQuery y sus extensiones, como XUpdate, así como APIs para Java (XMLDB, XQJ).

4. EXIST DB

eXist DB es un SGBD libre de código abierto que almacena datos XML de acuerdo a un modelo de datos XML. El motor de la base de datos está completamente escrito en Java, soporta los estándares de consulta XPath, XQuery y XSLT, además de indexación de documentos y soporte para la actualización de datos, y es compatible con multitud de protocolos como SOAP, XML-RPC, WebDav y REST. Con el SGBD, se proporcionan diversas aplicaciones que permiten ejecutar consultas directamente sobre una base de datos.

Los documentos XML se almacenan en **colecciones**, las cuales pueden estar anidadas. Desde un punto de vista práctico, el almacén de datos funciona como un sistema de ficheros: cada documento está en una colección. No es necesario que los documentos tengan una DTD o un esquema XML asociado, y dentro de una colección, se pueden almacenar documentos de cualquier tipo.

La carpeta ***eXist/webapp/WEB-INF/data*** contiene los ficheros más importantes de la base de datos:

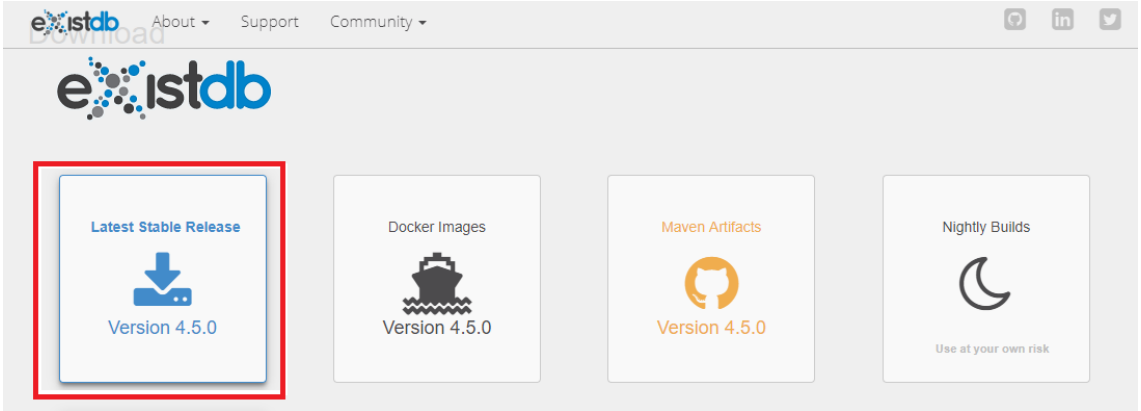
- **dom.dbx.** Es el almacén central nativo de datos. Es un fichero paginado donde se almacenan todos los nodos del documento de acuerdo al modelo DOM del W3C.
- **collections.dbx.** Almacena la jerarquía de colecciones y relaciona ésta con los documentos que contiene. Se asigna un identificador único a cada documento de la colección que es almacenado también junto al índice.

3.1. DESCARGA E INSTALACIÓN

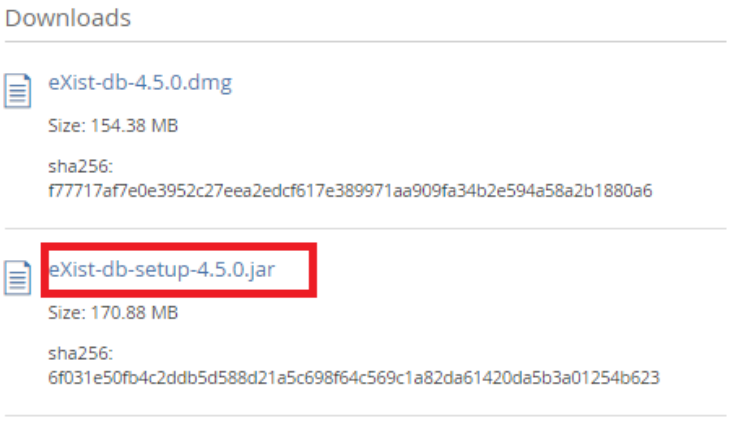
eXist DB puede funcionar de distintos modos:

- Como un servidor autónomo que ofrece servicios de llamada remota a procedimientos sobre internet (XML-RPC, WebDav, y REST).
- Insertado dentro de una aplicación Java.
- En un servidor J2EE, ofreciendo servicios XML-RPC, SOAP y WebDav.

El **sitio web** oficial de eXist DB es: http://exist-db.org/exist/apps/homepage/index.html. Desde aquí se puede descargar la última versión estable de la base de datos (4.5).



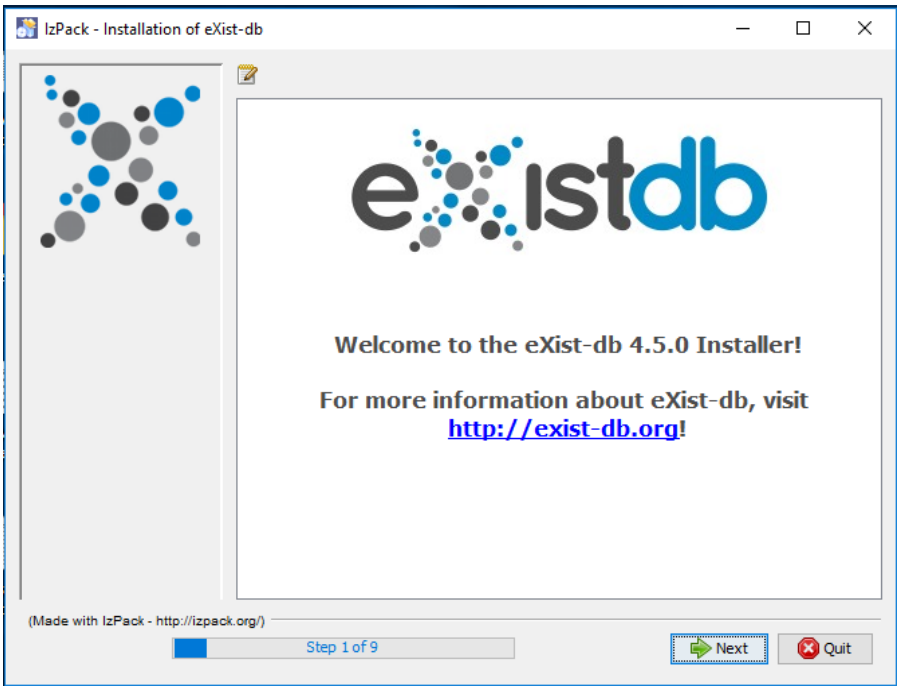
Al pulsar sobre el botón de descarga, el navegador nos redirige al repositorio donde se encuentran los ejecutables de instalación de eXist DB, es necesario tener instalada la versión 8 de Java. A continuación, si se trabaja con un sistema Windows, se pulsa sobre el fichero **eXist-db-setup.4.5.jar** para descargarlo.



Una vez descargado este fichero, la instalación de eXist DB puede iniciarse de dos formas:

- Mediante una invocación desde la línea de comandos:
  `java -jar eXist-db-setup-2.2.jar`
- Haciendo doble clic sobre el icono del fichero descargado para lanzar el instalador gráfico. Es un proceso bastante intuitivo, simplemente seguir el asistente y estar atengo al paso 2, donde se indica el directorio para la instalación, y 4 donde se indica la password del administrador, en nuestro caso pondremos admin para acordarnos.

  Vamos los pasos más detalladamente:



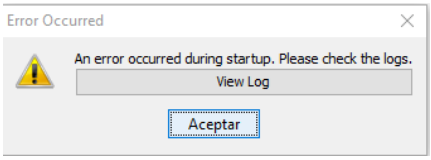
1) Elegir la ruta de instalación de eXist DB.
2) Elegir la ruta donde se almacenarán las bases de datos.
3) Elegir la contraseña para el usuario administrador de las bases de datos. Se recomienda no dejarla en blanco.
4) Seleccionar todos los paquetes/aplicaciones a instalar. Por lo menos, *dashboard*, *demo*, *doc*, *eXide* y *fundocs*, pues éstos serán necesarios para el desarrollo con XQuery.

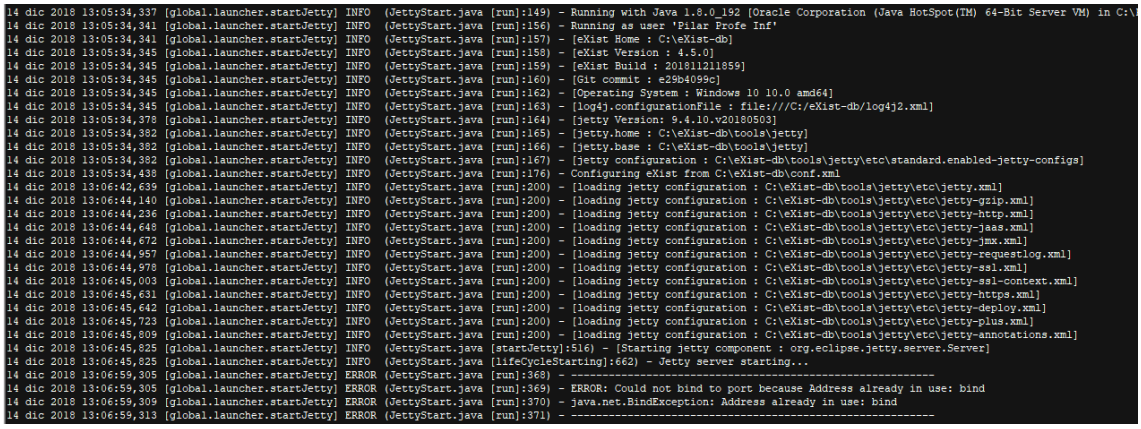
Información básica de instalación: https://exist-db.org/exist/apps/doc/basic-installation

Una vez instalada, para arrancar la base de datos se puede hacer desde el acceso directo que

se crea en el escritorio (si así se ha decidido en la instalación), o también desde el menú de la aplicación seleccionando ***eXist-db Database.*** También se puede lanzar el fichero ***start.jar*** que se encuentra en la carpeta de **eXist.**

Al lanzar la BD puede ocurrir que esta no se inicie y nos visualizce una pantalla de consola de error en el arranque de la BBDD.





El problema ocurrirá cuando al intentar conectar con la BD, porque para un funcionamiento correcto de eXist DB, se debe asegurar que el puerto de conexión de la base de datos esté abierto o habilitado, es decir, que no se encuentre bloqueado u ocupado por alguna aplicación. Exist se instala en un servidor web (**jetty)** y ocupa el puerto **8080**, como la mayoría de servidores web (Apache, Tomcat, Lampp…)

Por defecto, este puerto es el 8080, para resolver el problema, cambiamos el puerto. Para evitar futuros problemas, se recomienda cambiar el puerto de conexión de la base de datos dándole un nuevo valor, por ejemplo 8083, accediendo al fichero **/tools/jetty/etc/jetty.xml.** En la etiqueta < SystemProperty> en el valor del atributo default cambiamos el puerto.

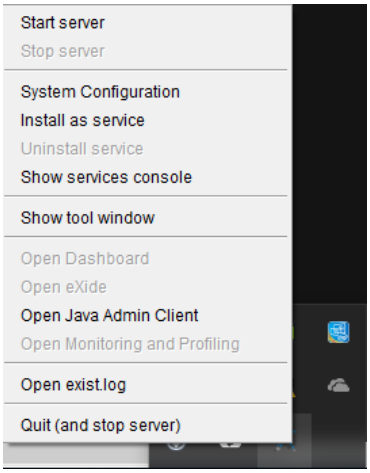
```
<Set name="securePort">
        <Property name="jetty.httpConfig.securePort"
        deprecated="jetty.secure.port">
                <Default>
                        <SystemProperty name="jetty.secure.port" default="8443"/>
                </Default>
        </Property>
</Set>
```

Modificando el puerto por defecto de conexión de la base de datos:

```
<Set name="securePort">
        <Property name="jetty.httpConfig.securePort"
        deprecated="jetty.secure.port">
```

```
                <Default>
                    <SystemProperty name="jetty.secure.port" default="8083"/>
                </Default>
            </Property>
        </Set>
```

Al lanzar la BD también se creará un icono asociado a eXist en la barra de tareas, desde allí podremos iniciar y parar el servidor, y abrir las distintas herramientas de tabajo con esta base de datos.



Para trabajar con la base de datos, primero hay que iniciar el servidor (en Windows, mediante el acceso directo ***eXist-db Database Startup*** del menú de inicio o desde el fichero **/bin/startup.bat**). También se puede instalar el servidor como un servicio (***install eXist as Service*****),** en cuyo caso, la base de datos se iniciaría de forma automática.

3.2. PRIMEROS PASOS

Dashboard es el centro de aplicaciones y de administración de eXist DB. Si el icono de la bandeja del sistema está activo, se puede acceder al panel mediante la opción ***Open Dashboard*** o con la siguiente URL en un navegador web: http://localhost:8083/exist/

La aplicación **Dashboard** permite añadir paquetes del repositorio público de eXist DB (incluyendo documentación), subir datos a la base de datos y realizar tareas administrativas (como crear cuentas de usuario). También contiene la aplicación web **eXide** para consultar la base de datos y crear aplicaciones propias.

Por último, cerrar la base de datos de forma inadecuada puede corromper los ficheros de datos. Por eso, se recomienda usar una de las siguientes maneras para cerrar eXist DB:

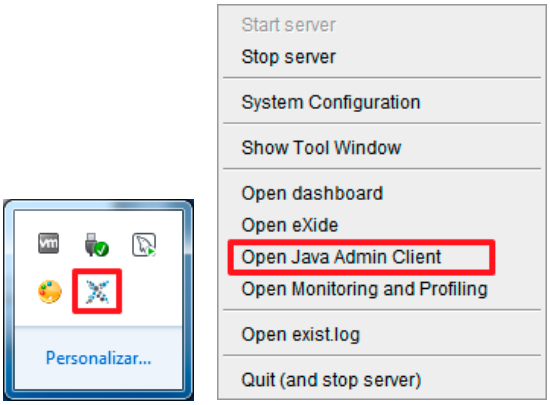
- La bandeja del sistema dispone de la opción ***Stop Server***. También se puede elegir ***Quit (and Stop Server).***
- La aplicación **Dashboard** tiene el botón ***Shutdown***.
- La aplicación **Java Admin Client** tiene la opción ***Connection*** -> ***Shutdown from the Menu***.
- Desde la línea de comandos, el script shutdown.bat (Windows) necesita los datos de acceso de la cuenta de administrador:

```
bin/shutdown.bat –u admin –p password
```

3.3. CLIENTE DE ADMINISTRACIÓN

eXist DB tiene un **Cliente de Administración basado en Java (Java Admin Client)**. Esta aplicación le permite al usuario realizar tareas administrativas, como gestionar de usuarios, modificar ajustes de seguridad, importar directorios completos en lotes, y hacer copias de seguridad o restaurar la base de datos.

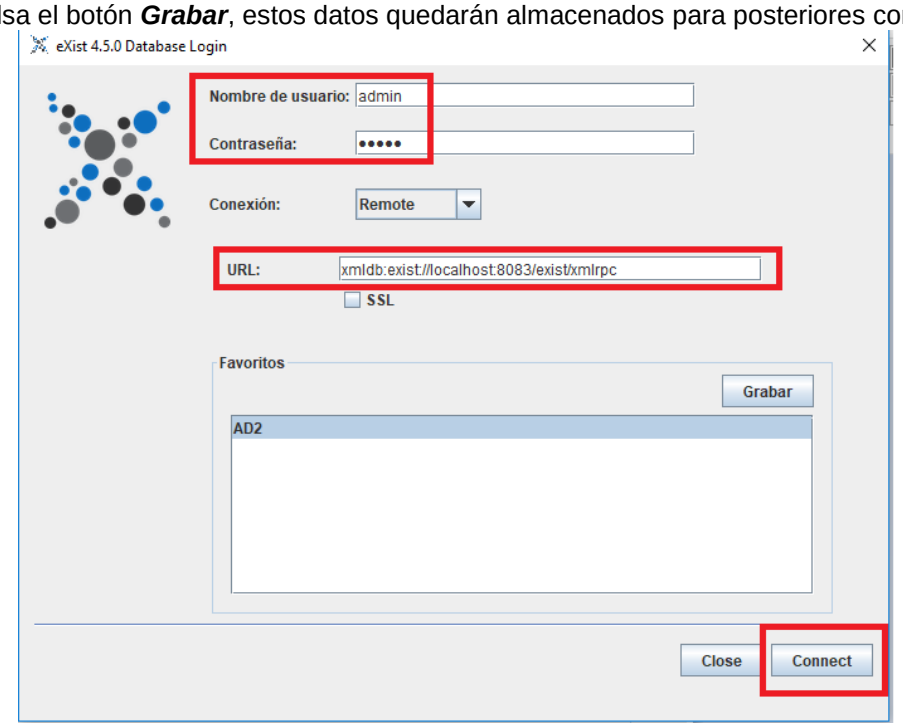
El Cliente de Administración tiene una interfaz gráfica, aunque también se puede usar en la línea de comandos. Para lanzarlo en Windows, se accede mediante el acceso directo ***Java Admin Client*** del menú de inicio o desde la opción ***Open Java Admin Client*** disponible en el menú asociado a eXist DB en la bandeja del sistema.



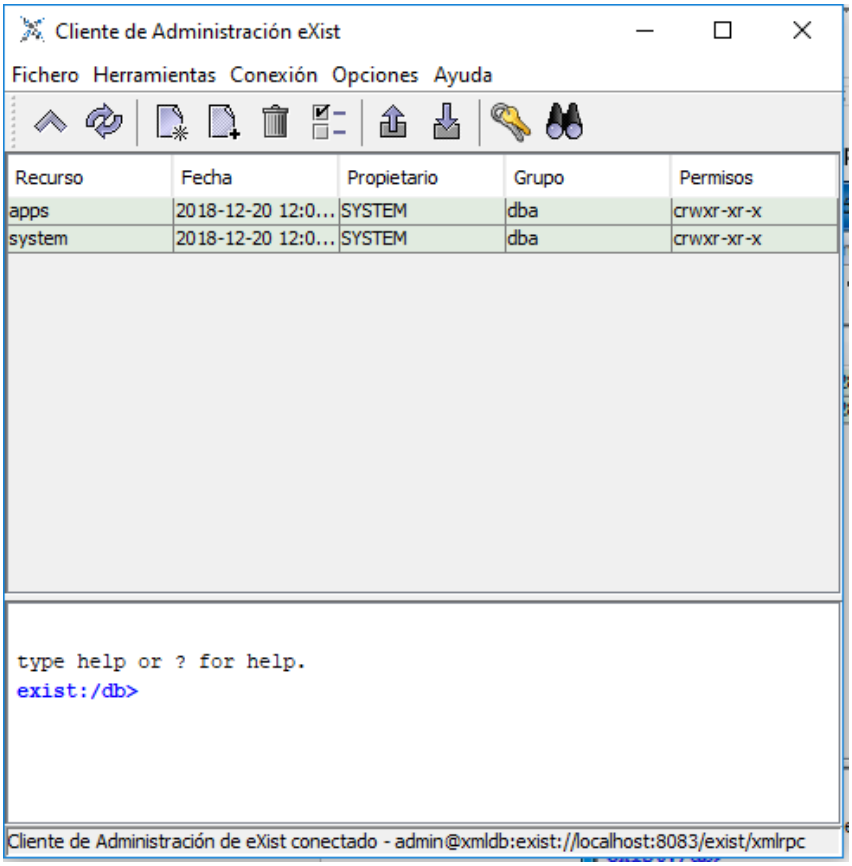
A continuación aparece una ventana de *Database Login*, que pide los siguientes datos para configurar la conexión:

- **Nombre de usuario y contraseña.**
- **Tipo de la conexión (remota o embebida).** El cliente de administración puede conectarse a un servidor "remoto" o puede lanzar una "base de datos embebida", es decir, una base de datos embebida en una aplicación que se ejecuta en el mismo proceso que el cliente. La opción "embebida" es útil para guardar/restaurar una gran cantidad de datos, pues evita la sobrecarga de la red.
- **URL de la base de datos.** Por defecto, es *xmldb:exist://localhost:8083/exist/xmlrpc*.
- Un título para la conexión.

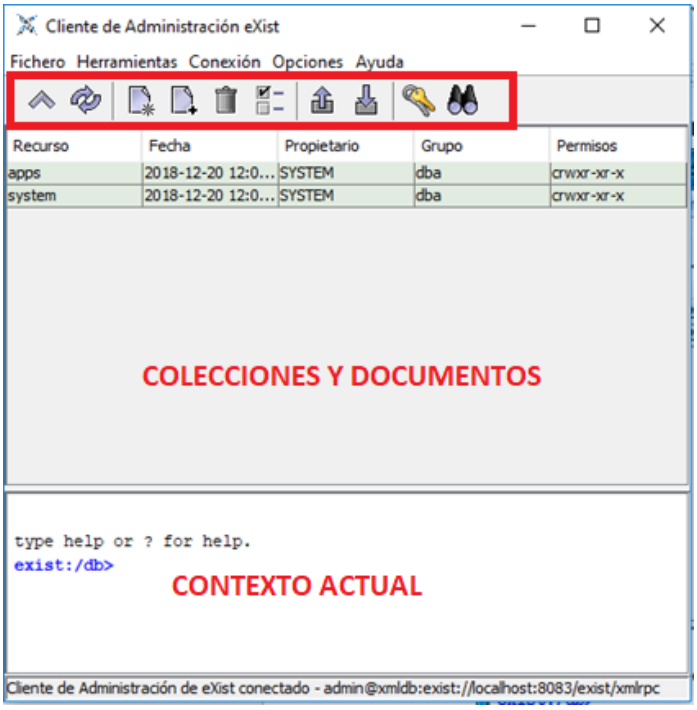
Si se pulsa el botón ***Grabar***, estos datos quedarán almacenados para posteriores conexiones.



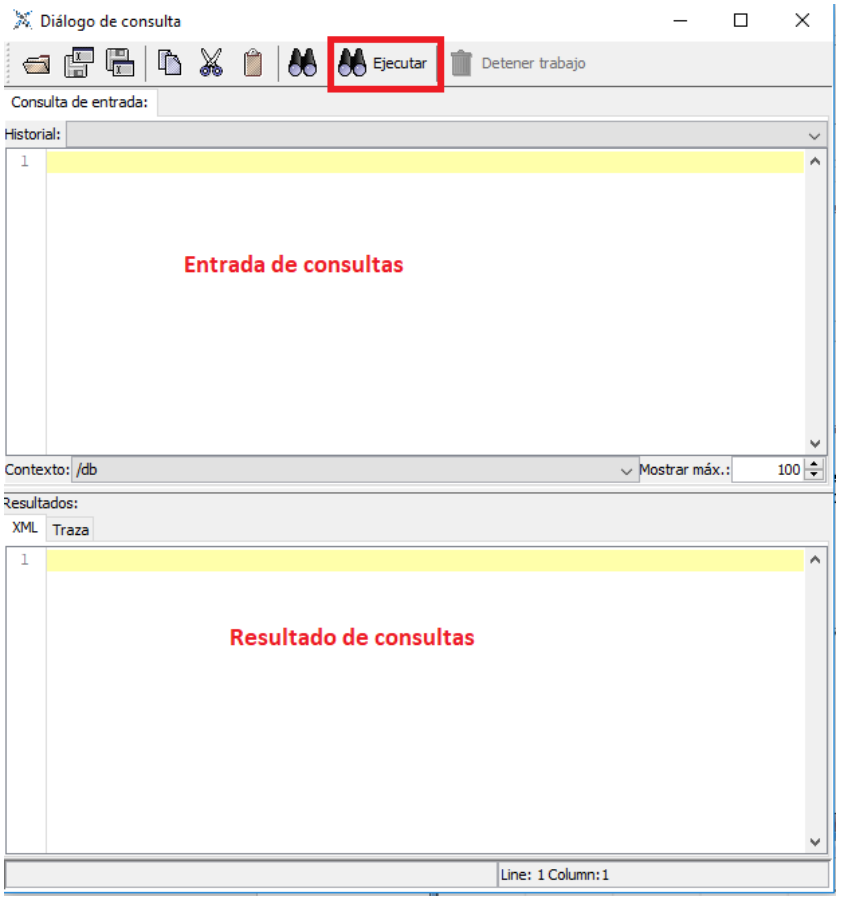
Tras pulsar el botón ***Connect***, aparece la ventana del Cliente de Administración:



El Cliente de Administración permite realizar todo tipo de operaciones sobre la base de datos, como crear y borrar colecciones, añadir y eliminar documentos a las colecciones, modificar los documentos, crear copias de seguridad y restaurarlas, administrar usuarios y realizar consultas con XPath.

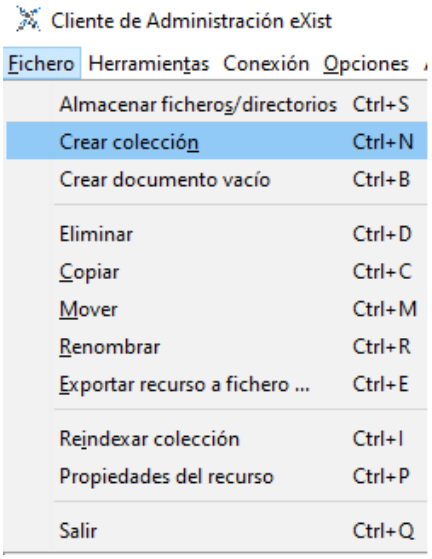


En la parte superior escribimos la consulta, pulsamos el botón *Ejecutar*, y en la inferior se muestra el resultado, también se puede ver la traza de ejecución seguida en la ejecución de la consulta.

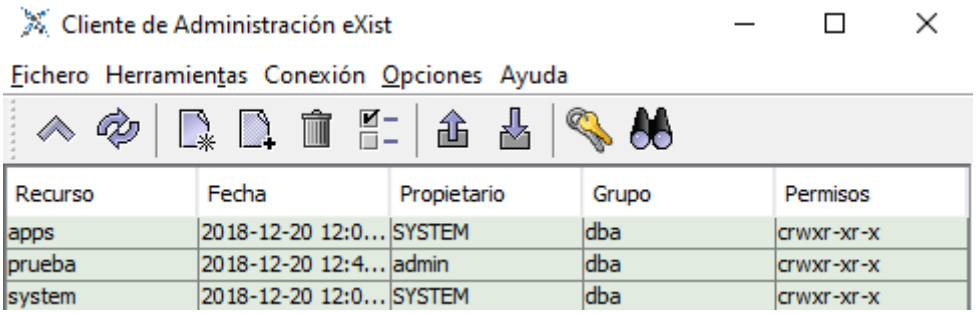


3.3.1. Crear una Colección Nueva

Los documentos XML se agrupan en colecciones. Lo primero que debemos hacer es subir una colección a la base de datos desde el cliente. Nos conectamos como *Admin.* Para crear una nueva colección, se puede hacer clic en el icono de nueva colección o ir al menú ***Fichero*** -> ***Crear Colección***:

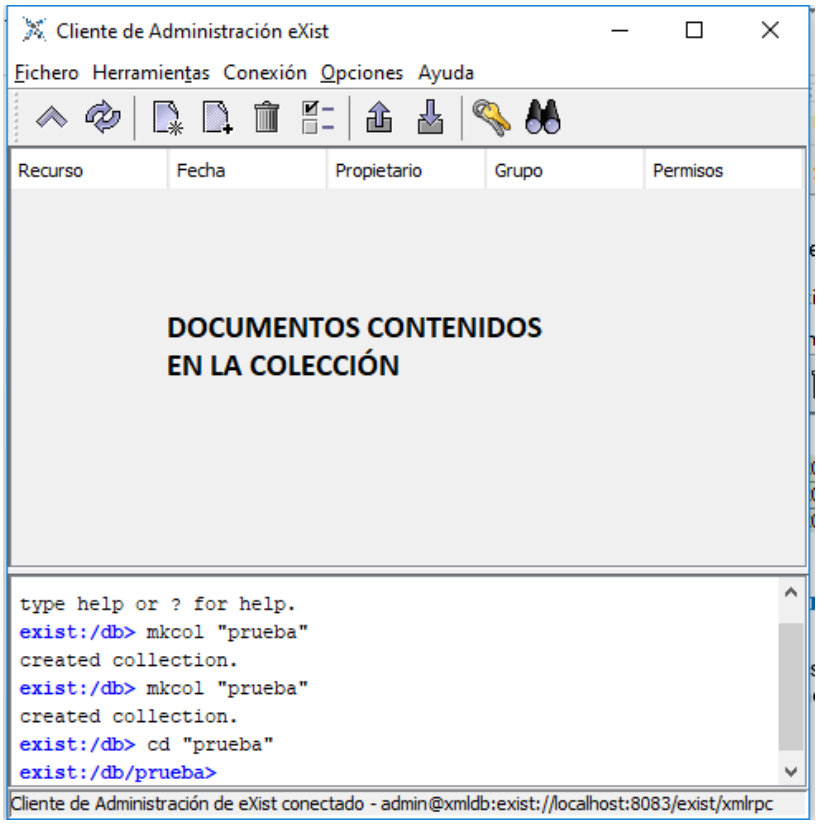


Tras pulsar el botón Aceptar, el Cliente de Administración mostrará la nueva colección creada junto con las anteriores previamente existentes:

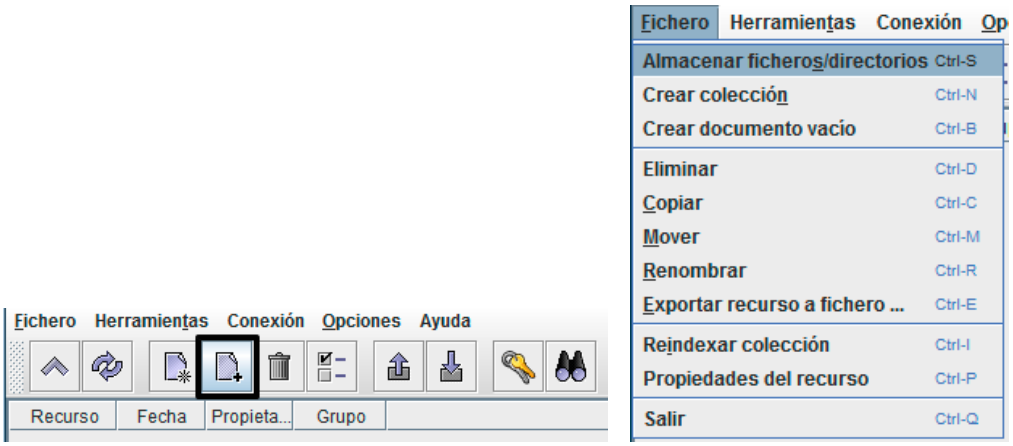


3.3.2. Añadir un Documento a una Colección

Antes de añadir un documento a una colección, se debe estar trabajando sobre la colección deseada. El Cliente de Administración mostrará en la parte superior el contenido de dicha colección y en la parte inferior indicará la colección a la que está conectado:

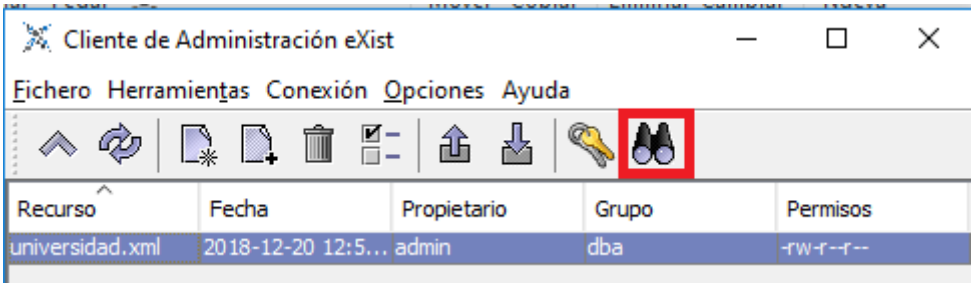


Desde aquí, para añadir un documento a la colección seleccionada, se puede hacer clic en el icono de nuevo documento o ir al menú ***Fichero*** -> ***Almacenar Ficheros/Directorios***:

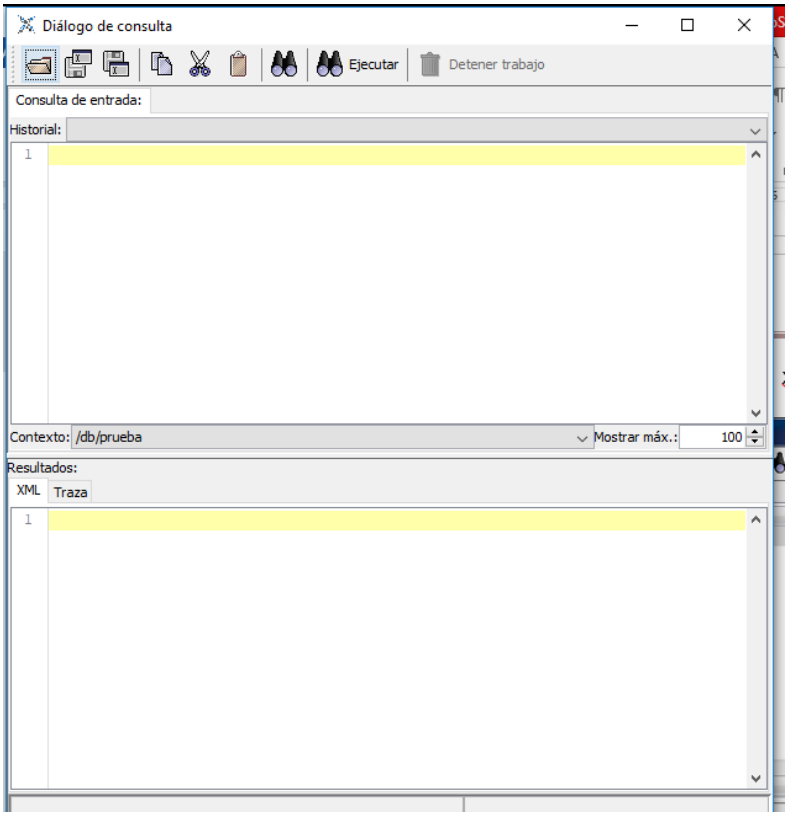


3.3.3. Consultar la Base de Datos usando XPath

Desde dentro de una colección se pueden utilizar expresiones XPath para seleccionar nodos.

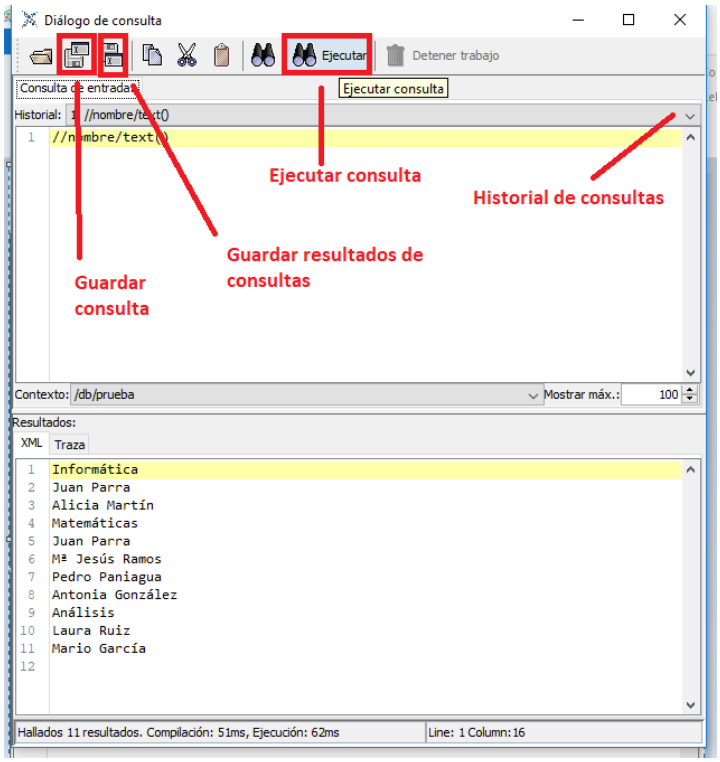


Si se pulsa el icono de consulta con XPath, aparece una ventana de consultas **Diálogo de Consulta (Query Dialog)**:

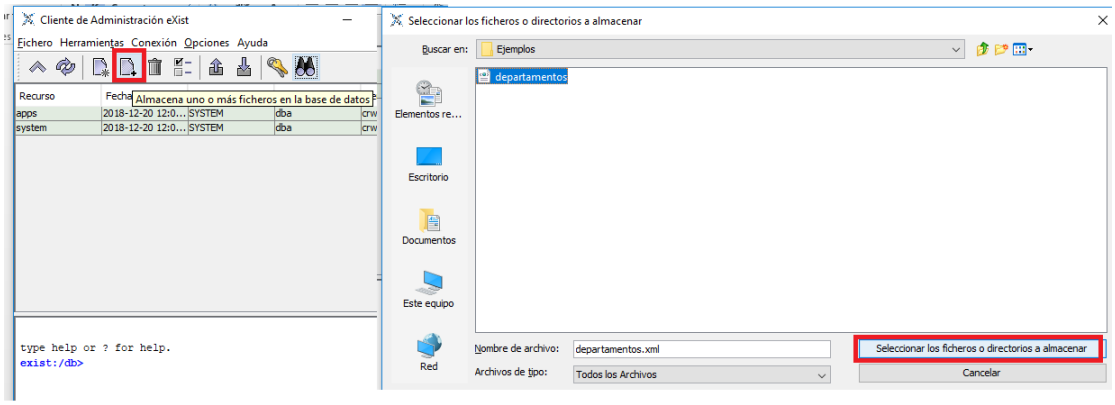


Desde aquí se puede elegir el contexto sobre el que ejecutar las consultas. También se pueden guardar las consultas y los resultados de las consultas en ficheros externos.

En la parte superior, se escribe la consulta y se pulsa el botón ***Ejecutar (Submit).*** En la parte inferior, se muestra el resultado de la consulta y también se puede ver la traza de ejecución seguida en la ejecución de la consulta.



*También podemos directamente importar documentos, que crean una colección con dichos documentos en la base de datos, y entonces se creará dentro de la colección (o carpeta) desde donde se llamó a la subida del documento. En principio dentro del contexto ***exist:/db>.***



4. XPATH

XPath (XML Path Language) es el lenguaje de rutas de XML y se utiliza para navegar dentro de la estructura jerárquica de un documento XML.

Un procesador de lenguaje XPath toma como entrada un árbol de nodos del documento origen, selecciona una parte de él y genera un nuevo árbol que puede ser tratado.

Especificación completa de XPath 2.0 del *W3C*:
https://www.w3.org/TR/xpath20/

Tutorial de XPath de *w3schools.com*:
http://www.w3schools.com/xml/xpath_intro.asp

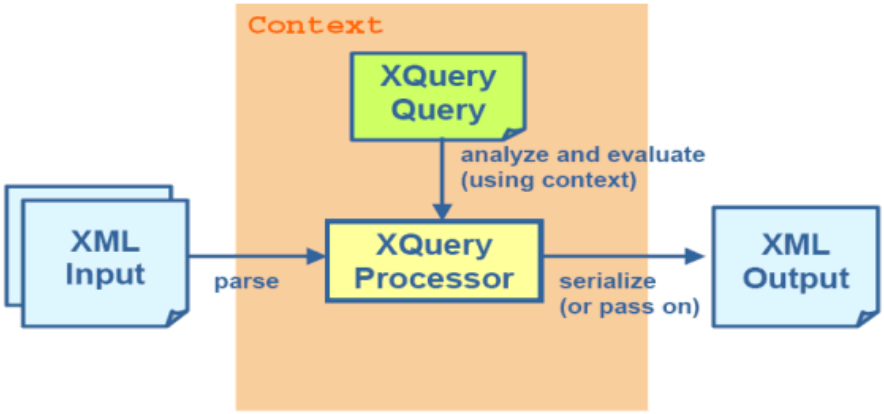
5. XQUERY

XQuery es un lenguaje de consulta diseñado para consultar documentos XML. Abarca desde ficheros XML hasta bases de datos relacionales con funciones de conversión de registros a XML. XQuery contiene a XPath, toda expresión de consulta de XPath es válida en XQuery, pero XQuery permite mucho más.

Una consulta en XQuery es una expresión que lee datos de uno o más documentos XML y devuelve como resultado otra secuencia de datos en XML.

Especificación completa de XQuery 1.0 del *W3C*:
http://www.w3.org/TR/xquery/

Tutorial de XQuery de *w3schools.com*:
http://www.w3schools.com/xml/xquery_intro.asp



XQuery permite realizar lo siguiente:

- Seleccionar información basada en un criterio específico.
- Buscar información en un documento o conjunto de documentos.
- Unir datos desde múltiples documentos o colección de documentos.
- Organizar, agrupar y resumir datos.
- Transformar y reestructurar datos XML, en otro vocabulario o estructura.
- Desempeñar cálculos aritméticos sobre números y fechas.
- Manipular cadenas de caracteres a formato de texto.

5.1. FUNCIONAMIENTO

En XQuery, las consultas siguen la norma **FLWOR (for, let, where, order by, return)**, y a diferencia de XPath, permite manipular, transformar y organizar los resultados de una consulta. La sintaxis general de una sentencia FLWOR es:

```
for <variable> in <expresión XPath>
let <variables vinculadas>
where <condición XPath>
order by <expresión>
return <expresión de salida>
```

La cláusula **for** se usa para seleccionar nodos mediante una o varias expresiones de XPath y almacenarlos en variables. Si se especifica más de una variable dentro del for, se actúa como un producto cartesiano. Las variables declaradas comienzan con $.

La cláusula **let** permite que se asignen valores resultantes de expresiones de XPath a variables para simplificar la representación. Se pueden poner varias líneas let, una por cada variable, o separar las variables mediante comas (,).

La cláusula **where** (opcional) filtra los elementos, eliminando todos los nodos que no cumplan las condiciones dadas.

La cláusula **order by** (opcional) se utiliza para reordenar la secuencia de nodos según el criterio dado.

La cláusula **return** construye el resultado de la consulta en XML. Se pueden añadir etiquetas XML a la salida y los datos a visualizar se encierran entre llaves {}. Además, la cláusula return puede contener condicionales *if-then-else* y así tener más versatilidad en la salida.

En XQuery, el término tupla se refiere a cada uno de los valores que toma una variable.

FOR	Vincula una o más variables a expresiones escritas en XPath, creando un flujo de tuplas en el que cada tupla está vinculada a una de las variables.
LET	Vincula una variable al resultado completo de una expresión añadiendo esos vínculos a las tuplas generadas por una claúsula for, o si no existe ninguna cláusula for, creando una única tupla que contenga esos vínculos.
WHERE	Es una cláusula opcional que sirve para seleccionar nodos generados por las sentencias for y let. La expresión es evaluada a booleano para cada nodo del flujo.
ORDER BY	Reorganiza la secuencia de tuplas generadas por las sentencias for y let filtradas por where. Se puede especificar el orden de la ordenación: ascending | descending.
RETURN	La sentencia return es evaluada una vez para cada tupla dentro de la secuencia de tuplas obtenida con las sentencias anteriores. La evaluación puede ser con cualquier expresión, permitiendo la creación de nuevos elementos y por tanto poder cambiar la estructura del documento.

No es necesario que aparezca ninguna de las cinco cláusulas FLWOR en una consulta. Como XQuery está construido sobre XPath, una expresión XPath es una consulta válida y no tiene ninguna de las 5 cláusulas vistas.

```
for $b in doc("libros.xml")//libro
where $b/@año = "2000"
order by $b/titulo
return $b/titulo

for $b in doc("libros.xml")//libro
where $b/@año = "2000"
return
<TITULO>{$b/titulo}</TITULO>
```

```
<titulo>Data on the Web</titulo>
<titulo>Millenium</titulo>

<TITULO>
    <titulo>Millenium</titulo>
</TITULO>
<TITULO>
    <titulo>Data onthe Web</titulo>
</TITULO>
```

5.1.1. Diferencias entre las Cláusulas for y let

La cláusula **for** vincula una variable con cada nodo que encuentra en la colección de datos:

```
for $b in
doc("libros.xml")/bib/libro/titulo
return <titulos>{ $b }</titulos>
```

```
<titulos>
    <titulo>TCP/IP Illustrated</titulo>
</titulos>
<titulos>
    <titulo>Advanced Programming for Unix
environment</titulo>
</titulos>
<titulos>
    <titulo>Millenium</titulo>
</titulos>
<titulos>
```

	<titulo>Data on the Web</titulo> </titulos> <titulos> <titulo>Economics of Technologyfor Digital TV</titulo> </titulos>

La cláusula **let** vincula una variable con todo el resultado de una expresión:

let $b := doc("libros.xml")/bib/libro/titulo **return** <titulos>{ $b }</titulos>	<titulos> <titulo>TCP/IP Illustrated</titulo> <titulo>Advanced Programming for Unix environment</titulo> <titulo>Millenium</titulo> <titulo>Data on the Web</titulo> <titulo> Economics of Technology for Digital TV</titulo> </titulos>

Si una cláusula let aparece en una consulta que ya posee una o más cláusulas for, los valores de las variables vinculada por la cláusula let se añaden a cada una de las tuplas generadas por la cláusula for.

for $b **in** doc("libros.xml")/bib/libro **let** $c := $b/autor **return** <libro> { $b/titulo, <autores> {count($c)}</autores> } </libro>	<libro> <titulo>TCP/IP Illustrated</titulo> <autores>1</autores> </libro> <libro> <titulo>Advanced Programming for Unix environment</titulo> <autores>1</autores> </libro> <libro> <titulo>Millenium</titulo> <autores>1</autores> </libro> <libro> <titulo>Data on the Web</titulo> <autores>3</autores> </libro> <libro> <titulo>Economics of Technology for Digital TV</titulo> <autores>0</autores> </libro>

Si en la consulta aparece más de una cláusula for (o más de una variable en una cláusula for), el resultado es el producto cartesiano de dichas variables.

for $t **in**
doc("libros.xml")//titulo,
$e **in**
doc("comentarios.xml")//entrada
where $t = $e/titulo
return <comentario> {$t,
$e/comentario} </comentario>
 	
for $t **in**
doc("libros.xml")//titulo,
	<comentario>
	<titulo>TCP/IP Illustrated</titulo>
	<comentario>Uno de los mejores libros de TCP/IP
	</comentario>
	</comentario>
	<comentario>
	<titulo>Data onthe Web</titulo>

`$e in doc("comentarios.xml")//entrada return <comentario> {$t, $e/comentario} </comentario>`	<comentario>Un libro muy bueno sobre bases de datos. </comentario> </comentario>

5.1.2. Funciones de Entrada

doc(URI) devuelve el nodo documento, o nodo raíz, del documento referenciado en URI. Es la función más habitual para acceder a la información almacenada en archivos.

collection(URI) devuelve una secuencia de nodos referenciados por una URI, sin necesidad de que exista un nodo documento o nodo raíz. Es la más habitual para acceder a una base de datos que tenga capacidad de crear estructuras de datos XML.

```
for $a in collection("ejercicios")//nombre
return <nombres>{$a}</nombres>

for $a in doc("ejercicio3.xml")//nombre
return <nombres>{$a}</nombres>
```

5.2. OPERACIONES Y FUNCIONES COMUNES

XQuery también soporta **expresiones condicionales** del tipo *if-then-else* con la misma semántica que en los lenguajes de programación. Una expresión condicional permite crear una u otra estructura de nodos en el resultado que dependa de los valores de las tuplas filtradas por la cláusula where.

La cláusula else es obligatoria, si no existe ningún valor a devolver, se devuelve una secuencia vacía con 'else ()'.

```
(:A partir del tercer autor aparecerá la expresión 'otros':)
for $b in doc("libros.xml")//libro
return
<libro>
       { $b/titulo }
       {
              for $a at $i in $b/autor
              (:con la clausula at se crea el contador $i:)
              where $i <= 2
              return <autor>{string($a/apellido), ", ",
                     string($a/nombre)}</autor>
       }
       {
              if (count($b/autor) > 2) then <autor>otros</autor>
              else ()
       }
</libro>
```

Se pueden utilizar los **cuantificadores existenciales** some y every. Permite definir consultas que devuelvan algún elemento que satisfaga la condición **some** o consultas que devuelvan los elementos en los que todos sus nodos satisfagan la condición **every**.

```
(:título de los libros en los que al menos un autor
tenga el apellido 'Stevens' y el nombre 'W.':)
for $b in doc("libros.xml")//libro
where some $a in $b/autor
       satisfies ($a/apellido="Stevens" and $a/nombre="W.")
return $b/titulo
```

```
(:título de los libros en los que todos los autores
tengan el apellido 'Stevens' y el nombre 'W.':)
(:cuando se aplica sobre un nodo vacío, siempre devuelve cierto:)
for $b in doc("libros.xml")//libro
where every $a in $b/autor
      satisfies ($a/apellido="Stevens" and $a/nombre="W.")
return $b/titulo
```

```
(:Devuelve el título de todos los libros que
contienen 'Programming' en el título:)
for $b in doc("libros.xml")//libro
where every $p in $b//titulo
      satisfies contains($p,"Programming")
return $b/titulo
```

El conjunto de **funciones** y **operadores** soportados por XQuerey 1.0 es el mismo conjunto de funciones y operadores utilizado en XPath 2.0 y XSLT 2.0:

- *Matemáticas:* +, -, *, , div(*), idiv(*), mod. La división se indica con el operador div ya que el símbolo / es necesario para indicar caminos. El operador idiv es para divisiones con enteros en las que se ignora el resto.
- *Comparación:* =, ! =, <, >, <=, >=, not().
- *Secuencia:* unión (|), intersect, except.
- *Redondeo:* floor(), ceiling(), round().
- *Funciones de agrupación:* count(), min(), max(), avg(), sum().
- *Funciones de cadena:* concat(), string-length(), starts-with(), ends-with(), substring(), upper-case(), lower-case(), string().
- *Uso general:* distinct-values() extrae los valores de una secuencia de nodos y crea una nueva secuencia con valores únicos (eliminando los nodos duplicados), empty() devuelve cierto cuando la expresión entre paréntesis está vacía, exists() devuelve cierto cuando una secuencia contiene, al menos, un elemento.
- Los *comentarios* van encerrados entre caras sonrientes: (: comentario :).

```
(:Consulta para obtener una lista ordenadas de apellidos de
todos los autores y editores:)
for $l in distinct-values (doc("libros.xml")//(autor|editor)/apellido)
order by $l
return <apellidos>{ $l }</apellidos>
```

```
(:Obtener un nodo libro con todos sus nodos hijos salvo el nodo precio:)
for $b in doc("libros.xml")//libro
where $b/titulo = "TCP/IP Illustrated"
return
<libro>
      { $b/@* }
      { $b/* except $b/precio }
</libro>
```

```
(:Obtener los nodos libro que tengan al menos un nodo autor:)
for $b in doc("libros.xml")//libro
where not(empty($b/autor))
return $b
```

Ejemplos de Consultas sobre el documento *ejemplos1.xml*:

Obtener los asegurados de la primera póliza, cambiando el formato y convirtiendo los atributos en elementos.

```
for $a in doc("ejemplos1.xml")/envio/poliza[1]/asegurado
return
<newaseg>
       <nombre>{$a/@nombre}</nombre>
       <apellidos>{$a/@apellidos}</apellidos>
</newaseg>
```

Obtener un nuevo elemento póliza que contenga un elemento nuevo con el número de asegurados.

```
for $a in doc("ejemplos1.xml")//poliza
return
<poliza>numaseg="{count($a/asegurado)}"</poliza>
```

Obtener un nuevo elemento póliza que contenga un atributo nuevo con el número de asegurados, y que para cada póliza reformatee el elemento asegurado, poniendo un atributo con el número de garantías y cuyo contenido sea el nombre.

```
for $a in doc("ejemplos1.xml")//poliza
return
<polizanumaseg="{count($a/asegurado)}">
{
       for $b in $a/asegurado
       return
              <aseguradonumgar="{count($b/garantia)}">
                     {data($b/@nombre), data($b/@apellidos)}
              </asegurado>
}
</poliza>
```

Ejemplos de Consultas sobre el Documento *empleados.xml*:

Obtener los nombre de los oficios que comienzan por la letra P.

```
for $a in doc("empleados.xml")/EMPLEADOS/EMP_ROW/OFICIO
where starts-with(data($a),'P')
return $a
```

Obtener el nombre de los oficios y el número de empleados de cada oficio. Utilizar la función distinct-values para devolver los distintos oficios.

```
for $a in distinct-values(/EMPLEADOS/EMP_ROW/OFICIO)
let $emp := count(/EMPLEADOS/EMP_ROW[OFICIO=$a])
return concat($a,' = ', $emp)
```

Obtener el número de empleados que tiene cada departamento y la media de salario redondeada.

```
for $dep in distinct-values(/EMPLEADOS/EMP_ROW/DEPT_NO)
let $emp := count(/EMPLEADOS/EMP_ROW[DEPT_NO=$dep])
let $sala := round(avg(/EMPLEADOS/EMP_ROW[DEPT_NO=$dep]/SALARIO))
return
       concat('Departamento: ', $dep, ' N° empleados: ', $emp,
       ' Salario medio: ', $sala)
```