

INDICE

<u>1.1.</u>	<u>INTRODUCCIÓN</u>	<u>1</u>
<u>1.2.</u>	<u>SQL FRENTE A NOSQL</u>	<u>2</u>
<u>1.3.</u>	<u>VENTAJAS Y DESVENTAJAS DE LOS SISTEMAS NOSQL</u>	<u>3</u>
<u>1.4.</u>	<u>TIPOS DE BASES DE DATOS NOSQL</u>	<u>3</u>
<u>1.5.</u>	<u>BASES DE DATOS EXIST</u>	<u>3</u>
<u>1.5.1.</u>	<u>INSTALACIÓN DE EXIST</u>	<u>4</u>
<u>1.5.2.</u>	<u>PRIMEROS PASOS CON EXIST</u>	<u>8</u>
<u>1.5.3.</u>	<u>EL CLIENTE DE ADMINISTRACIÓN DE EXIST</u>	<u>9</u>
<u>1.5.3.1.</u>	<u>SUBIR UNA COLECCIÓN A LA BASE DE DATOS</u>	<u>10</u>
<u>1.6.</u>	<u>LENGUAJES DE CONSULTAS XPATH Y XQUERY</u>	<u>12</u>
<u>1.6.1.</u>	<u>EXPRESIONES XPATH</u>	<u>13</u>
<u>1.6.2.</u>	<u>AXIS XPATH</u>	<u>20</u>
<u>1.6.3.</u>	<u>CONSULTAS XQUERY</u>	<u>22</u>
<u>1.6.4.</u>	<u>OPERADORES Y FUNCIONES MÁS COMUNES EN XQUERY</u>	<u>28</u>
<u>1.7.</u>	<u>ACCESO A EXIST DESDE JAVA</u>	<u>30</u>
<u>1.7.1.</u>	<u>LA API XQJ (XQUERY)</u>	<u>30</u>

1.1. Introducción

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

El término NoSQL surge con la llegada de la Web 2.0, ya que hasta entonces solo 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 puede subir contenido, lo que provoca un crecimiento exponencial de los datos.

Esto plantea los problemas de administrar y acceder a toda esa información almacenada en bases de datos relacionales. Una de las soluciones, propuesta por las empresas para solucionar estos problemas de accesibilidad, fue utilizar más máquinas, sin embargo, fue una solución cara y no acabó con el problema.

La otra solución fue crear nuevos sistemas de gestión de datos diseñados para un uso específico, que con el tiempo han dado lugar a soluciones robustas, apareciendo así el movimiento NoSQL.

Entonces, hablar de bases de datos NoSQL es hablar de estructuras que nos permiten almacenar información en aquellas situaciones donde las bases de datos relacionales generan ciertos problemas, principalmente por problemas de escalabilidad y rendimiento de las bases de datos relacionales al acceder a cantidades, inmensos datos y miles de usuarios concurrentes y millones de las consultas diarias se encuentran.

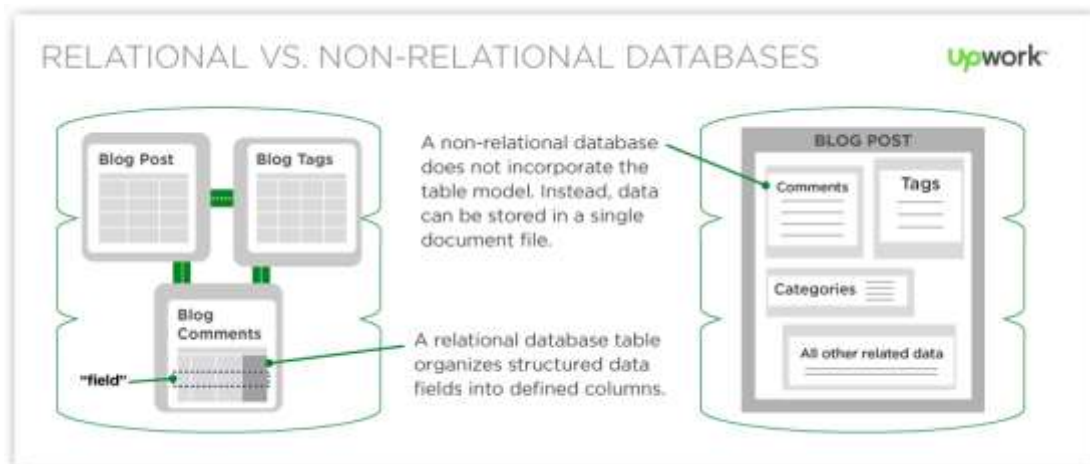
1.2. SQL frente a NoSQL

Las bases de datos relacionales centran su interés en la confiabilidad de las transacciones bajo el conocido principio **ACID**, acrónimo de Atomicidad, Consistencia, Aislamiento y Durabilidad:

- ✓ **Atomicidad:** Asegurarse de que la transacción se complete o no se ejecute en absoluto, sin quedarse a medio camino de fallas.
- ✓ **Coherencia:** Asegurar el estado de validez de los datos en todo momento.
- ✓ **Aislamiento:** Garantice la independencia entre transacciones.
- ✓ **Durabilidad:** Asegura la persistencia de la transacción en caso de cualquier falla.

Cuando la magnitud y el dinamismo de los datos se vuelven importantes, el principio ACID de los modelos relacionales queda en segundo plano en comparación con el rendimiento, la disponibilidad y la escalabilidad, las características más características de las bases de datos NoSQL. Hoy en día, los sistemas de datos modernos en Internet están más en línea con el conocido principio BASE:

- ✓ **Basic Availability:** prioridad de disponibilidad de datos.
- ✓ **Soft state:** Prioriza la propagación de datos, delegando el control de inconsistencias a elementos externos.
- ✓ **Eventually consistency:** se supone que las inconsistencias temporales progresarán hasta un estado final estable.



1.3. Ventajas y Desventajas de los Sistemas NoSQL

<https://www.techrepublic.com/blog/10-things/10-things-you-should-know-about-nosql-databases/>

1.4. Tipos de Bases de Datos NoSQL

<https://en.wikipedia.org/wiki/NoSQL>

1.5. Bases de Datos eXist

eXist es un SGBD libre de código abierto que almacena datos XML de acuerdo a un modelo de datos XML. El motor de 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 los datos y para multitud de protocolos como SOAP, XML-RPC, WebDav y REST. Con el SGBD se dan aplicaciones que permiten ejecutar consultas directamente sobre la BD.

Los documentos XML se almacenan en colecciones, las cuales pueden estar anidadas; des de un punto de vista práctico el almacén de datos funciona como un sistema de ficheros. Cada documento está en una colección, las colecciones serian como carpetas. No es necesario que los documentos tengan una DTD o un XML Schema asociado (XSD), y dentro de una colección pueden almacenarse documentos de cualquier tipo.

En la carpeta **eXist\webapp\WEB-INF\data** es donde se guardan los archivos más importantes de la BD, entre ellos están:

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

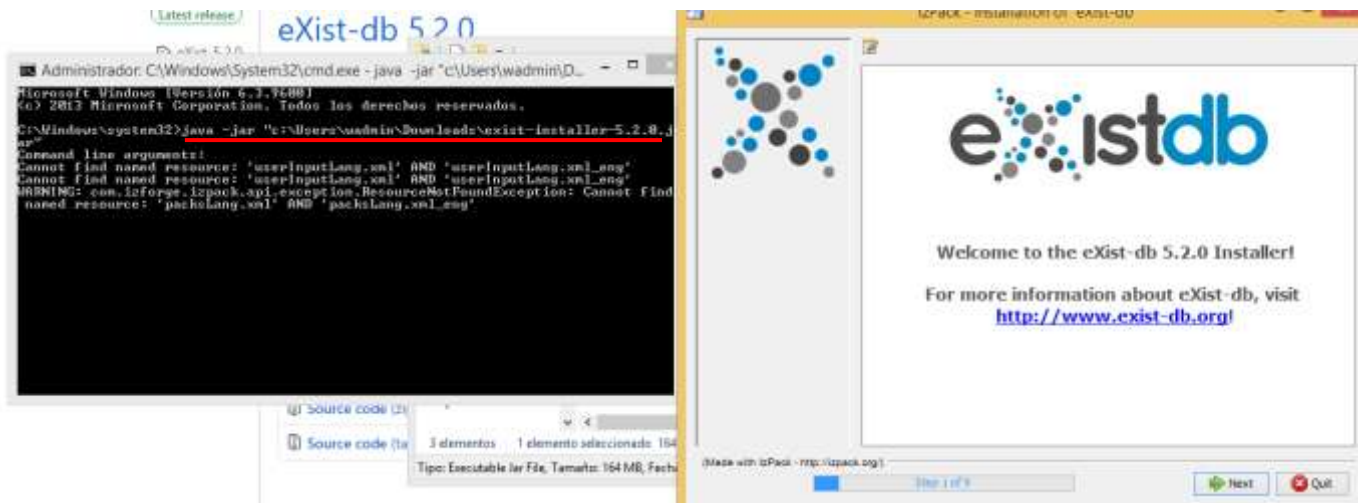
1.5.1. Instalación de eXist

eXist puede funcionar de distintos modos:

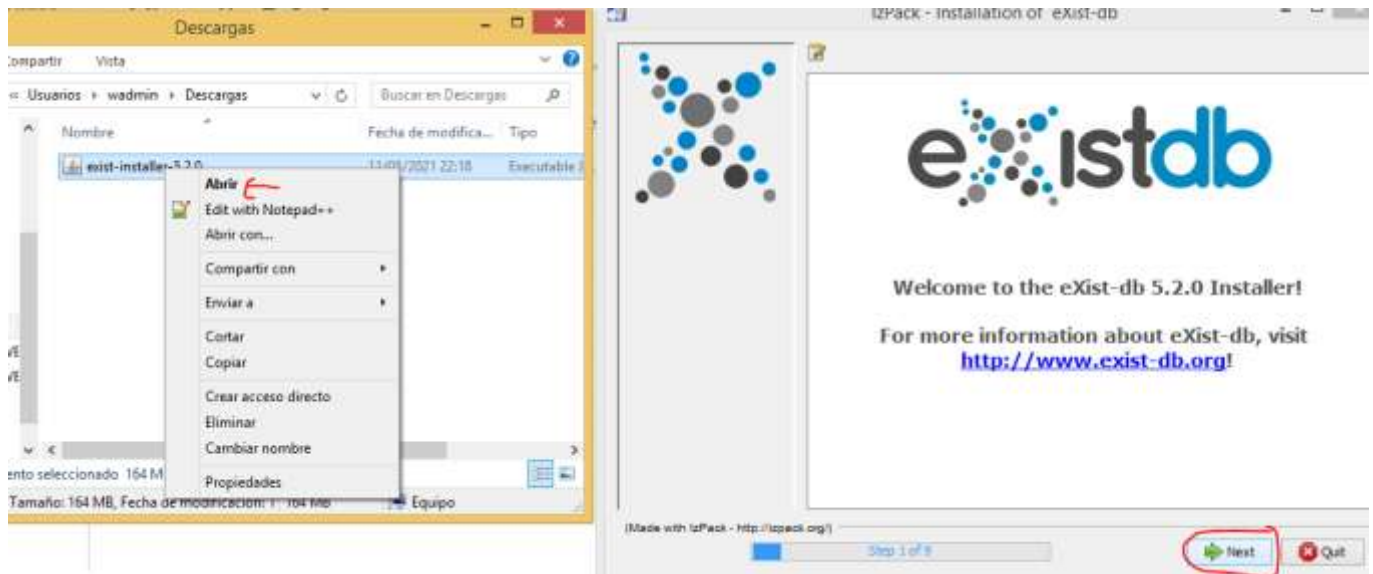
- ✓ Funcionando como servidor autónomo (ofreciendo servicios de llamada remota a procedimientos que funcionan sobre Internet como XML-RPC, WebDAV y REST).
- ✓ Insertado dentro de una aplicación Java.
- ✓ En un servidor J2EE, ofreciendo servicios XML-RPC, SOAP, y WebDAV.

En el sitio <http://www.exist-db.org/exist/apps/homepage/index.html> podremos descargar la última versión de la BD. En este caso se ha instalado la versión **exist-installer-5.2.0.jar**, es necesario tener instalada la versión Java 8. Se instala utilizando el fichero JAR proporcionado en la web:

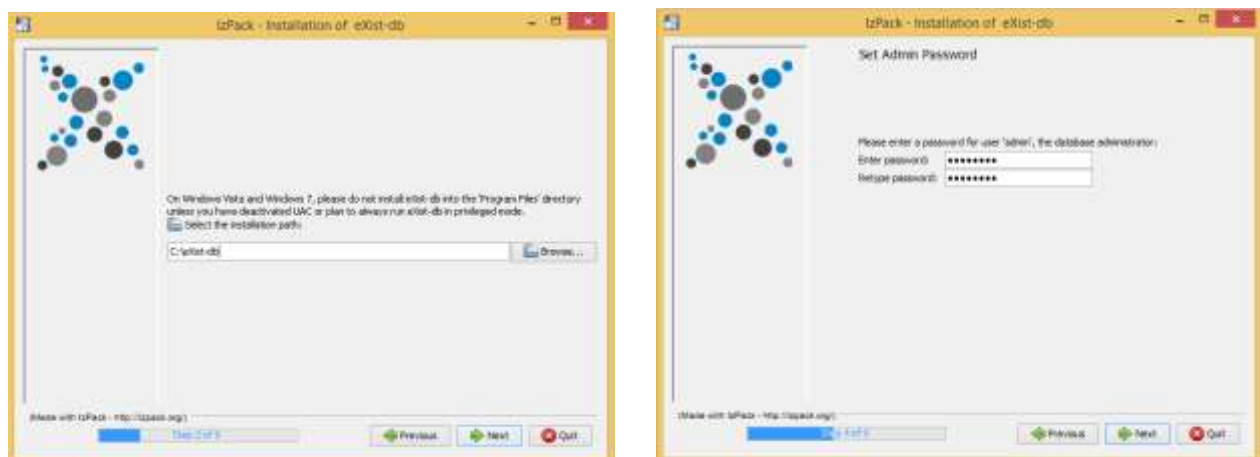
- 1) Que ha de invocarse desde la línea de comandos (**java -jar exist-installer-5.2.0.jar**).



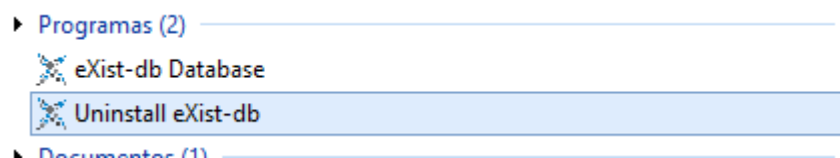
- 2) O bien, haciendo doble clic sobre el icono correspondiente una vez descargado, y siguiendo el asistente.



El proceso de instalación es bastante intuitivo, simplemente seguir el asistente y estar atento a los pasos 2, donde se indica el directorio de instalación, y 4 donde se indica la password del administrador, en nuestro caso pondremos '**abc123..**' para acordarnos.

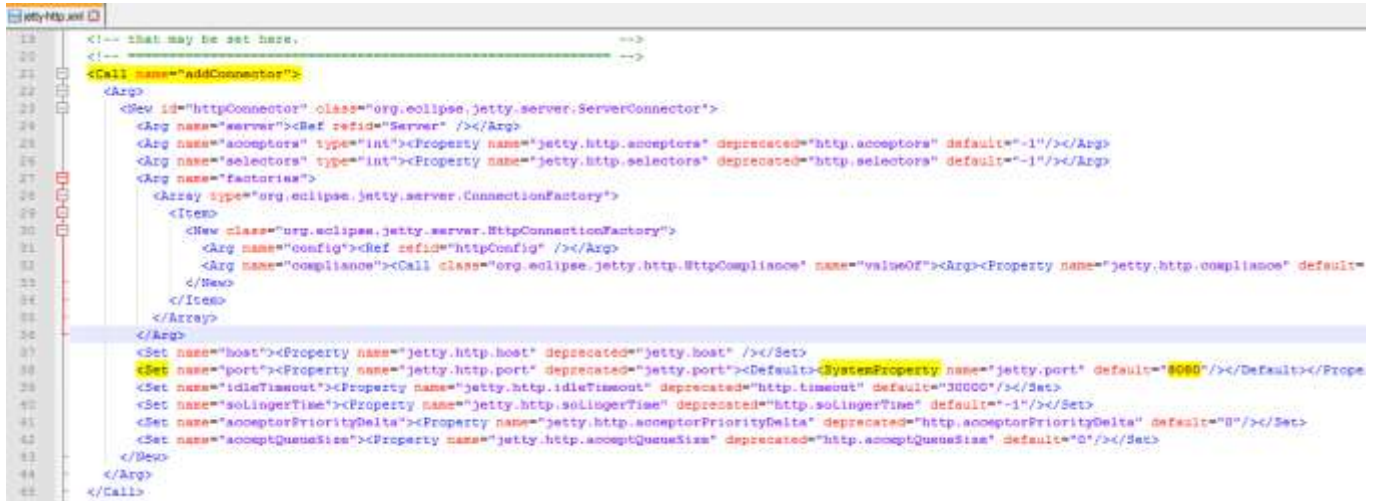


Una vez instalada, para arrancar la base de datos se puede hacer desde el acceso directo que se crea en el escritorio (si ha si se ha decidido en la instalación), o también desde el menú de la aplicación seleccionando **eXist-db XML Database**. También se puede lanzar el fichero **start.jar** que se encuentra en la carpeta **eXist**.



Al lanzar la BD puede ocurrir que esta no se inicie y se visualice un error de arranque de la BD. El problema ocurrirá cuando al intentar conectar con la BD, el puerto que utiliza eXist está ocupado por otra aplicación. eXist se instala en un servidor web (**jetty**) y ocupa el puerto **8080**, como la mayoría de servidores web (Apache, Tomcat, Lampp...).

Para resolver el problema, cambiamos el puerto. Para ello se edita el fichero de configuración: **C:\eXist-db\etc\jetty\jetty-http.xml**, en la etiqueta **<Call name="addConnector">** cambiaremos el puerto, y en la propiedad **SystemProperty** de la etiqueta **<default="8080">** escribimos un nuevo valor, por ejemplo 8083:

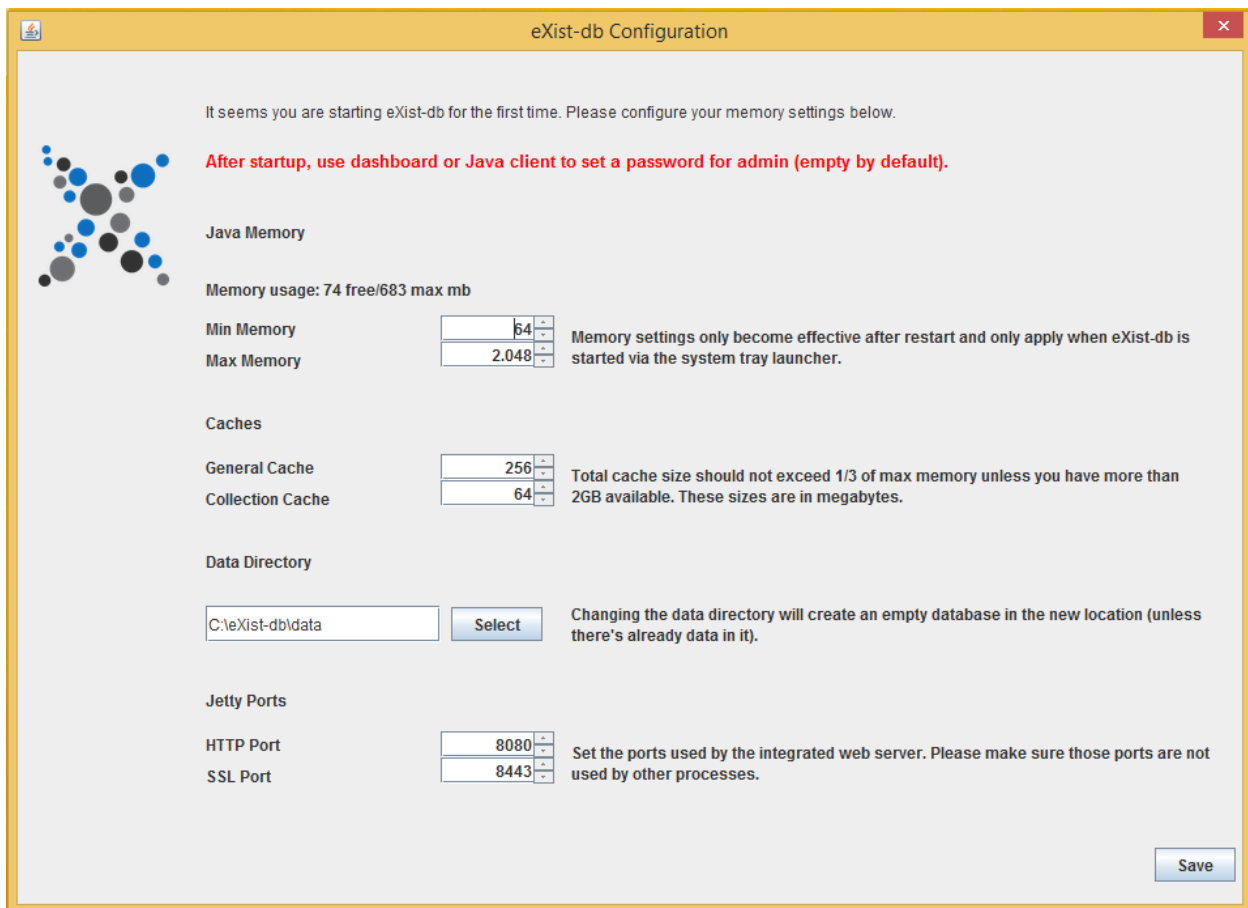


```

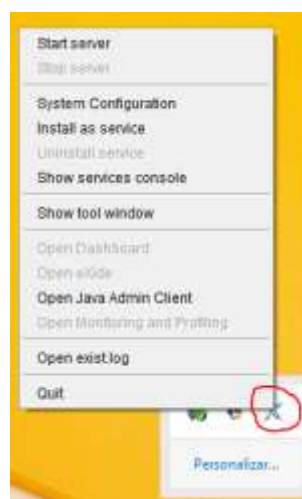
13 <!-- That may be set here. -->
14 <!-- -->
21 <Call name="addConnector">
22   <Arg>
23     <New id="httpConnector" class="org.eclipse.jetty.server.ServerConnector">
24       <Arg name="server"><Ref refid="Server" /></Arg>
25       <Arg name="acceptors" type="int"><Property name="jetty.http.acceptors" deprecated="http.acceptors" default="-1"/></Arg>
26       <Arg name="selectors" type="int"><Property name="jetty.http.selectors" deprecated="http.selectors" default="-1"/></Arg>
27       <Arg name="factories">
28         <Array type="org.eclipse.jetty.server.ConnectionFactory">
29           <Item>
30             <New class="org.eclipse.jetty.server.HttpConnectionFactory">
31               <Arg name="config"><Ref refid="httpConfig" /></Arg>
32               <Arg name="compliance"><Call class="org.eclipse.jetty.http.HttpCompliance" name="valueOf"><Arg><Property name="jetty.http.compliance" default="
33             </New>
34           </Item>
35         </Array>
36       </Arg>
37       <Set name="host"><Property name="jetty.http.host" deprecated="jetty.host" /></Set>
38       <Set name="port"><Property name="jetty.http.port" deprecated="jetty.port"><Default><SystemProperty name="jetty.port" default="8080"/></Default></Property>
39       <Set name="idleTimeout"><Property name="jetty.http.idleTimeout" deprecated="http.timeout" default="30000"/></Set>
40       <Set name="soLingerTime"><Property name="jetty.http.soLingerTime" deprecated="http.soLingerTime" default="-1"/></Set>
41       <Set name="acceptorPriorityDelta"><Property name="jetty.http.acceptorPriorityDelta" deprecated="http.acceptorPriorityDelta" default="0"/></Set>
42       <Set name="acceptQueueSize"><Property name="jetty.http.acceptQueueSize" deprecated="http.acceptQueueSize" default="0"/></Set>
43     </New>
44   </Arg>
45 </Call>

```

También se puede modificar esta y otras opciones desde la ventana de configuraciones:



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 trabajo con esta base de datos (*Dashboard*, *eXide* y *Java Admin Client*).



Opciones de eXist desde la barra de tareas

1.5.2. Primeros pasos con eXist

La base de datos, una vez arrancada, también se puede abrir desde el navegador escribiendo: <http://localhost:8080/exist/>.



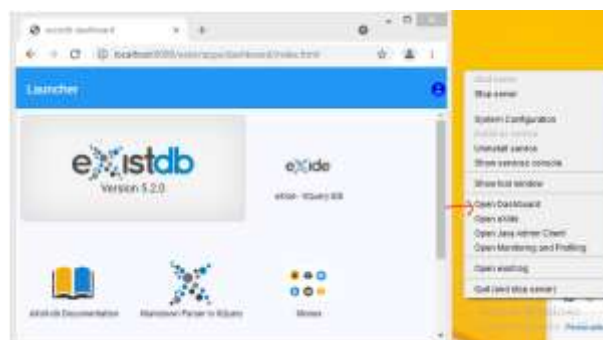
Para hacer consultas y trabajar con la base de datos lo podemos hacer desde varios sitios:

- ✓ Desde el eXide (**Open eXide**): el eXide es una de las herramientas para realizar consultas a documentos de la bd, cargar documentos externos a la BD, crear y borrar colecciones, entre otras cosas.

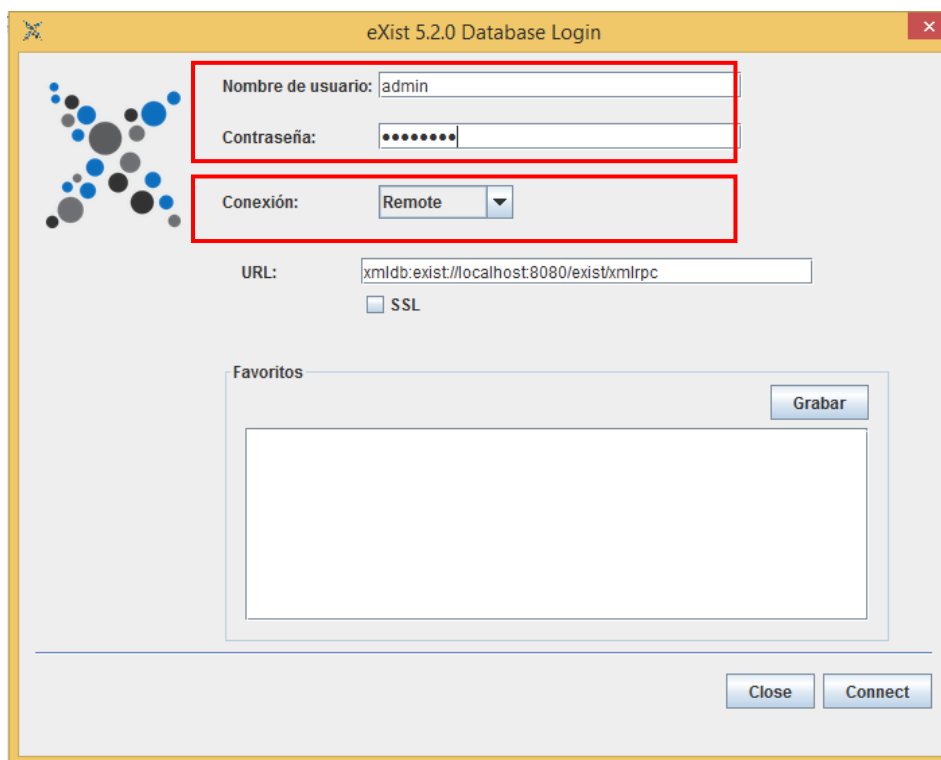
La URL del **eXide** es: <http://localhost:8080/exist/apps/eXide/index.html>



- ✓ Desde el dashboard (**Open Dashboard**): escaparate del administrador de aplicaciones de la BD. Soporta aplicaciones y plugins.




- ✓ Desde el cliente java (**Open Java Admin Client**): el cliente es la herramienta que utilizaremos a lo largo del tema para hacer las consultas. El cliente nos pedirá conexión con el usuario y contraseña (utilizaremos admin/abc123..) y hay que asegurarse de poner correctamente el puerto de la URL: **`xmldb:exist://localhost:8080/exist/xmlrpc`**

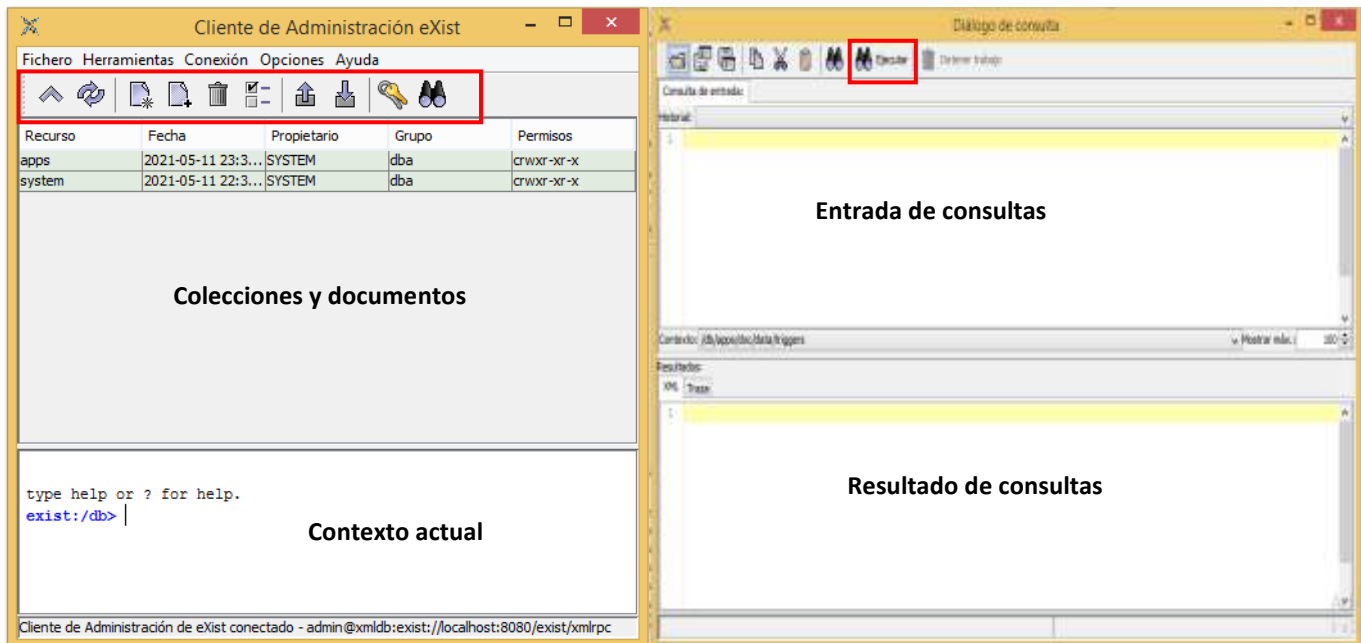


1.5.3.El cliente de administración de eXist

Una vez conectado se muestra la ventana del Cliente de Administración eXist, desde aquí se podrán realizar todo tipo de operaciones sobre la BD, crear y borrar colecciones, añadir y eliminar documentos a las colecciones, modificar los documentos, crear copias de seguridad, restaurarlas, administrar usuarios y realizar consultas XPath, entre otras operaciones. Además podremos navegar por las colecciones (las carpetas) y elegir un contexto a partir de donde se ejecutarán las consultas. Igualmente si se hace doble clic en un documento este se abrirá y también se podrán hacer cambios en los mismos.

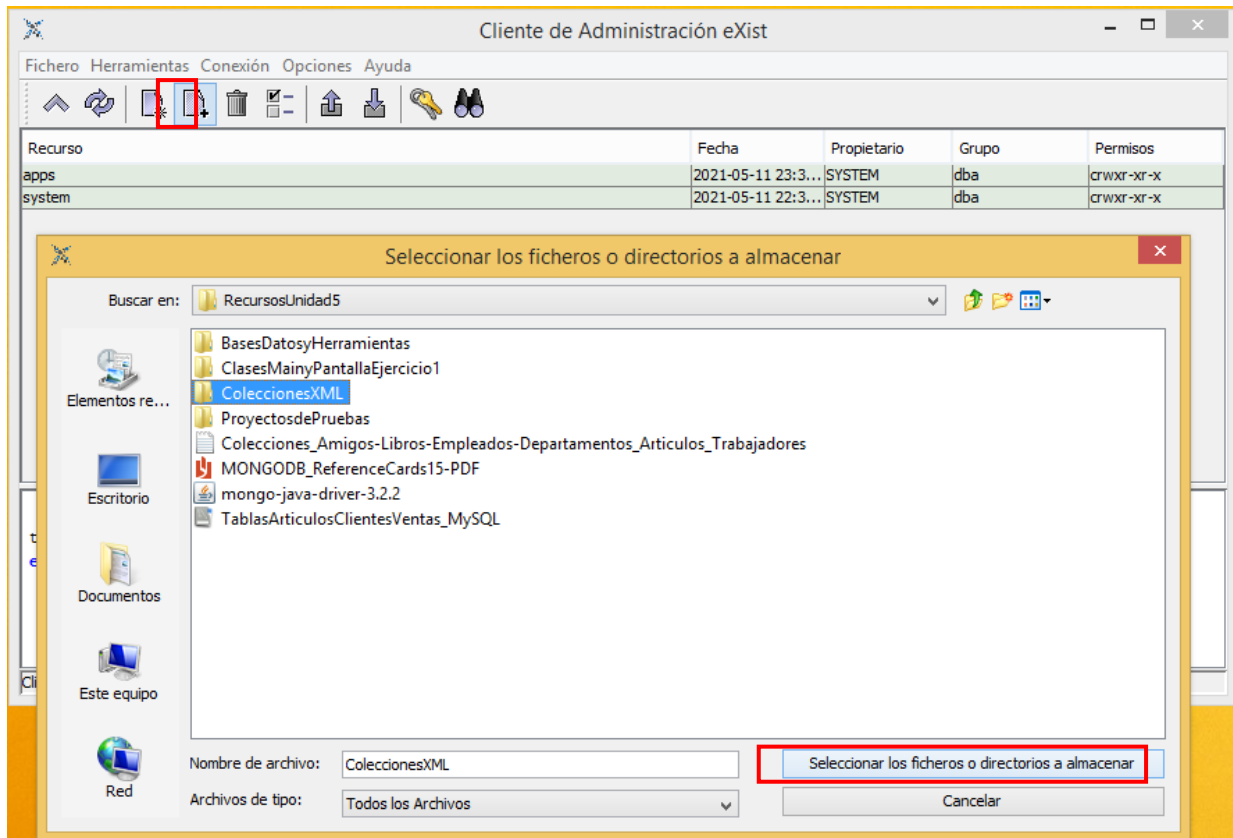


Si pulsamos el botón  (**Consultar la BD usando XPath**), aparece la ventana de consultas Dialogo de consulta, desde aquí se puede elegir también, el contexto sobre el que ejecutaremos las consultas, se pueden también guardar las consultas y los resultados de las consultas en ficheros externos. En la 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.



1.5.3.1. Subir una colección a la base de datos

Lo primero que haremos es subir una colección a la base de datos desde el cliente. Nos conectamos con *Admin*, de momento es el único usuario con el que se va a trabajar. En la ventana, del *Cliente* pulsamos el botón *Almacena uno o más ficheros en la base de datos*, y en la ventana que aparece seleccionamos la carpeta a subir y se pulsa el botón ***Seleccionar los ficheros o directorios***.



Subir una colección a eXist

Una vez pulsada, aparece una ventana de transferencia de datos, y al cerrarla, la colección con los documentos se creará en la base de datos, y se creará dentro de la colección (o carpeta) desde donde se llamó a la subida de documentos. En principio se subirán dentro del contexto *exist:/db>*.

Para hacer consultas sobre esa colección, hacemos doble clic sobre ella para seleccionarla, y una vez dentro pulsamos al botón Consultar la base de datos usando XPath. Así nos aseguramos de hacer las consultas dentro de nuestra colección, ese será el contexto a utilizar. Desde el dialogo de consulta también se podrán guarda las consultas y los resultados en ficheros. Compilar y ejecutar las consultas. Y desde el historial obtendremos las consultas ya realizadas. Si la consulta se realiza mal, aparecerá una ventana Java informando del error.

The screenshot shows the 'Cliente de Administración' (Administration Client) of the eXist XML database. At the top, there is a menu bar with 'Fichero', 'Herramientas', 'Conexión', 'Opciones', and 'Ayuda'. Below it is a toolbar with various icons. A table lists resources with columns 'Recurso' and 'Fecha':

Recurso	Fecha
departamentos.xml	2021-05-12 12:30:35
departamentosnuevo.xml	2021-05-12 12:30:35
empleados.xml	2021-05-12 12:30:35
productos.xml	2021-05-12 12:30:36
sucursales.xml	2021-05-12 12:30:36
universidad.xml	2021-05-12 12:30:36
zonas.xml	2021-05-12 12:30:36

A red arrow points to the 'universidad.xml' entry. Overlaid on this is a 'Diálogo de consulta' (Query Dialog) window. It has a toolbar with icons for saving, executing, and stopping. The 'Consulta de entrada' (Input Query) field contains '/universidad/departamento'. The 'Historial' (History) list shows a previous query '/sucursales/'. The 'Contexto' (Context) field is set to '/db/ColeccionesXML/ColeccionPruebas'. The 'Resultados' (Results) section shows an XML snippet:

```

1 <departamento telefono="112233" tipo="A">
2   <codigo>IFC1</codigo>
3   <nombre>Informática</nombre>
4   <empleado salario="2000">
5     <puesto>Asociado</puesto>
6     <nombre>Juan Parra</nombre>

```

Annotations with arrows point to the 'Guardar la consulta' (Save query) icon, the 'Guardar resultados de consultas' (Save query results) icon, the 'Ejecutar la consulta' (Execute query) button, and the 'Historial de consultas' (Query history) list. A red box highlights the 'Contexto' field.

Haciendo consultas en eXist

1.6. Lenguajes de consultas XPath y XQuery

Ambos son estándares para acceder y obtener datos desde documentos XML, estos lenguajes tienen en cuenta que la información en los documentos está semiestructurada o jerarquizada como árbol.

XPath, es el lenguaje de rutas de XML, se utiliza para navegar dentro de la estructura jerárquica de un XML.

XQuery, es a XML lo mismo que SQL es a las bases de datos relacionales, es decir, es un lenguaje de consulta diseñado para consultar documentos XML. Abarca desde archivos XML hasta bases de datos relaciones con funciones de conversión de registros a XML. XQuery contiene a XPath, toda expresión de consulta en XPath es válida en XQuery, pero XQuery permite mucho más.

1.6.1. Expresiones XPath

XPath es un lenguaje que permite seleccionar nodos de un documento XML y calcular valores a partir de su contenido. Existen varias versiones de XPath aprobadas por el W3C, aunque la versión más utilizada sigue siendo la versión 1.

La forma en que XPath selecciona partes del documento XML se basa en la representación arbórea que se genera del documento. A la hora de recorrer un árbol XML podemos encontrarlos con los siguientes tipos de nodos:

- ✓ **nodos raíz**, es la raíz del árbol, se representa por /.
- ✓ **nodos elemento**, cualquier elemento de un documento XML, son las etiquetas del árbol.
- ✓ **nodos texto**, los caracteres que están entre las etiquetas.
- ✓ **nodos atributo**, son como propiedades añadidas a los nodos elemento, se representan con @.
- ✓ **nodos comentario**, las etiquetas de comentario.
- ✓ **nodos espacio de nombres**, contienen espacios de nombres.
- ✓ **nodos instrucción de proceso**, contienen instrucciones de proceso, van entre las etiquetas <?.....?>.

Por ejemplo. Dado este documento XML:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<universidad>
<espacio xmlns="http://www.misitio.com"
    xmlns:prueba="http://www.misitio.com/pruebas"/>

    <!-- DEPARTAMENTO 1 -->
    <departamento telefono="112233" tipo="A">
    <codigo>IFC1</codigo>
    <nombre>Informática</nombre>
    </departamento>

    <!-- DEPARTAMENTO 2 -->
    <departamento telefono="990033" tipo="A">
    <codigo>MAT1</codigo>
    <nombre>Matemáticas</nombre>
    </departamento>

    <!-- DEPARTAMENTO 3 -->
    <departamento telefono="880833" tipo="B">
    <codigo>MAT2</codigo>
    <nombre>Análisis</nombre>
    </departamento>

</universidad >
```

Nos encontramos los siguientes tipos de nodos:

TIPO NODO	
Elemento	<universidad><departamento><codigo><nombre>
Texto	IFC1, Informática, MAT1, Matemáticas, MAT2, Análisis
Atributo	telefono="112233" tipo="A" telefono="990033" tipo="A", telefono="880833" tipo="B"
Comentario	<!-- DEPARTAMENTO 1 -->, <!-- DEPARTAMENTO 2 -->, <!-- DEPARTAMENTO 3 -->
Espacio de nombres	<espacio xmlns="http://www.misitio.com" xmlns:prueba= http://www.misitio.com/pruebas/ >
Instrucción de proceso	<?xml version="1.0" encoding="ISO-8859-1"?>

Los test sobre los tipos de nodos pueden ser:

- ✓ Nombre del nodo, para seleccionar un nodo concreto, ej.:/universidad
- ✓ **prefix:***, para seleccionar nodos con un espacio de nombres determinado.
- ✓ **text()**, selecciona el contenido del elemento, es decir, el texto, ej.://nombre/text().
- ✓ **node()**, selecciona todos los nodos, los elementos y el texto, ej.:/universidad/node().
- ✓ **processing_instruction()**, selecciona nodos que son instrucciones de proceso.
- ✓ **comment()**, selecciona los nodos de tipo comentario, /universidad/comment().

La sintaxis básica de XPath es similar a la del direccionamiento de ficheros. Utiliza **descriptores de ruta o de camino** que sirven para seleccionar los nodos o elementos que se encuentran en cierta ruta en el documento. Cada descriptor de ruta o paso de búsqueda puede a su vez dividirse en tres partes:

- ✓ **eje**: indica el nodo o los nodos en los que se realiza la búsqueda.
- ✓ **nodo de comprobación**: especifica el nodo o los nodos seleccionados dentro del eje.
- ✓ **predicado**: permite restringir los nodos de comprobación. Los predicados se escriben entre corchetes.

Las expresiones XPath se pueden escribir utilizando una sintaxis abreviada, fácil de leer, o una sintaxis más completa en la que aparecen los nombres de los ejes (AXIS), más compleja. Por ejemplo, estas dos expresiones devuelven los departamentos con más de 3 empleados, la primera es la forma abreviada y la segunda es la completa:

```
/universidad/departamento [count (empleado) > 3]
/child::universidad/child::departamento [count (child::empleado) > 3]
```

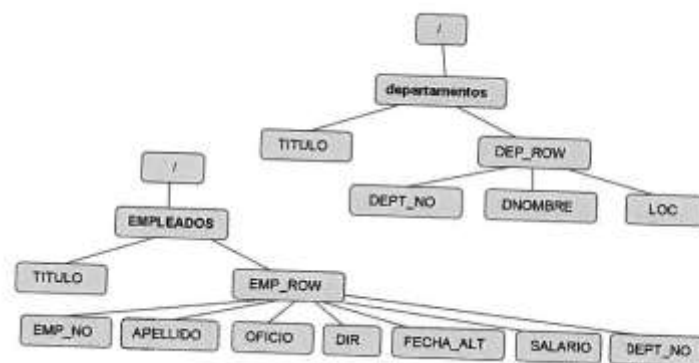
En este tema estudiaremos la sintaxis abreviada, así pues, los descriptors se forman simplemente nombrando la etiqueta separada por / (hay que poner el nombre de la etiqueta tal cual está en el documento XML, recuerda que hace distinción entre mayúsculas y minúsculas).

Si el descriptor comienza con / se supone que es una **ruta desde la raíz**. Para seguir una ruta indicaremos los distintos nodos de paso: /paso1/paso2/paso3... Si las rutas comienzan con / son **rutas absolutas**, en caso contrario serán relativas.

Si el descriptor comienza por // se supone que la ruta descrita puede comenzar en cualquier parte de la colección.

EJEMPLOS XPATH UTILIZANDO UNA SINTAXIS ABREVIADA:

A partir de la colección **ColeccionPruebas**, que contiene los documentos *departamentos.xml* y *empleados.xml*, cuyas estructuras se muestran en la siguiente imagen. Probar desde el diálogo de consultas las siguientes consultas



Estructuras de departamentos.xml y empleados.xml

- ✓ /, si ejecutamos esta orden y estamos dentro del contexto **/db/ColeccionPruebas** devuelve todos los departamentos y los empleados, es decir, incluye todas las etiquetas que cuelgan del nodo departamentos y del nodo EMPLEADOS, que están dentro de la colección *Prueba*.
- ✓ **/departamentos**, devuelve todos los datos de los departamentos.
- ✓ **/departamentos/DEP_ROW**, devuelve todas las etiquetas dentro de cada DEP_ROW.
- ✓ **/departamentos/DEP_ROW/node()**, devuelve todas las etiquetas dentro de cada DEP_ROW, no incluye DEP_ROW.
- ✓ **/departamentos/DEP_ROW/DNOMBRE**, devuelve los nombres de los departamentos de cada DEP_ROW, entre etiquetas.
- ✓ **/departamentos/DEP_ROW/DNOMBRE/text()**, devuelve los nombres de los departamentos de cada DEP_ROW, sin etiquetas.
- ✓ **//LOC/text()**, devuelve todas las localidades, de toda la colección(//), solo hay 4.

- ✓ **//DEPT_NO**, devuelve todos los números de departamentos, entre etiquetas. Observar que en este caso devuelve 18 filas en lugar de 4, es porque recoge todos los elementos DEPT_NO de la colección y en la colección también están incluidos *Empleados.xml*.
- ✓ **El operador *** se usa para nombrar a cualquier etiqueta, se usa como comodín. Por ejemplo:
 - El descriptor **/*/DEPT_NO** selecciona las etiquetas DEPT_NO que se encuentran a 1 nivel de profundidad desde la raíz, en este caso ninguna.
 - **/**/DEPT_NO** selecciona las etiquetas DEPT_NO que se encuentran a dos niveles de profundidad desde la raíz, en este caso 18.
 - **/departamentos/*** selecciona las etiquetas que van dentro de la etiqueta departamentos y sus subetiquetas.
 - **/*** ¿Qué salida produce este descriptor?, depende del contexto en el que se ejecute. Se mostrarán todas las etiquetas de todos los documentos, es decir, se mostrarán todos los documentos de la colección.
 -
- ✓ **Condiciones de selección.** Se utilizarán los corchetes para seleccionar elementos concretos, en las condiciones se pueden usar los comparadores: <, >, <=, >=, =, !=, **or**, **and** y **not** (or, and y not deben escribirse en minúscula). Se utilizará el separador | para unir varias rutas. Ejemplos:
 - **/EMPLEADOS/EMP_ROW[DEPT_NO=10]**, selecciona todos los elementos o nodos (etiquetas) dentro de EMP_ROW de los empleados del DEPT_NO 10.
 - **/EMPLEADOS/EMP_ROW/APELLIDO|/EMPLEADOS/EMP_ROW/DEPT_NO**, selecciona los nodos APELLIDO y DEPT_NO de los empleados.
 - **/EMPLEADOS/EMP_ROW[DEPT_NO=10]/APELLIDO/text()**, selecciona los apellidos de los empleados del DEPT_NO=10.
 - **/EMPLEADOS/EMP_ROW[not(DEPT_NO = 10)]**, selecciona todos los empleados (etiquetas) que NO son del DEPT_NO igual a 10.
 - **/EMPLEADOS/EMP_ROW[not(OFICIO='ANALISTA')]/APELLIDO/text()**, selecciona los APELLIDOS de los empleados que NO son ANALISTAS.
 - **/EMPLEADOS/EMP_ROW[DEPT_NO=10]/APELLIDO | /EMPLEADOS/EMP_ROW[DEPT_NO=10]/OFICIO**, selecciona el APELLIDO y el OFICIO de los empleados del DEPT_NO=10.
 - **/**[DEPT_NO=10]/DNOMBRE/text()**,
/departamentos/DEP_ROW[DEPT_NO=10]/DNOMBRE/text(), estas dos consultas devuelven el nombre del departamento 10.
 - **/**[OFICIO="EMPLEADO"]//EMP_ROW**, devuelve los empleados con OFICIO "EMPLEADO", por cada empleado devuelve todos sus elementos. Busca en cualquier parte de la colección //.

- Esto devuelve lo mismo:
/EMPLEADOS/EMP_ROW[OFICIO="EMPLEADO"]//EMP_ROW.
- */EMPLEADOS/EMP_ROW[SALARIO> 1300 and DEPT_NO=10]*, devuelve los datos de los empleados con SALARIO mayor de 1300 y del departamento 10.
- */EMPLEADOS/EMP_ROW[SALARIO> 1300 and DEPT_NO=20]/APELLIDO | /EMPLEADOS/EMP_ROW[SALARIO> 1300 and DEPT_NO=20]/OFICIO*, devuelve el APELLIDO y el OFICIO de los empleados con SALARIO mayor de 1300 y del departamento 20. Se utiliza el separador | para unir las dos rutas.

✓ **Utilización de funciones y expresiones matemáticas.**

- Un número dentro de los corchetes representa la posición del elemento en el conjunto seleccionado. Ejemplos:
/EMPLEADOS/EMP_ROW[1], devuelve todos los datos del primer empleado.
/EMPLEADOS/EMP_ROW[5]/APELLIDO/text(), devuelve el APELLIDO del quinto empleado.
- La función **last()** selecciona el último elemento del conjunto seleccionado. Ejemplos:
/EMPLEADOS/EMP_ROW[last()], selecciona todos los datos del último empleado.
/EMPLEADOS/EMP_ROW[last()-1]/APELLIDO/text(), devuelve el APELLIDO del penúltimo empleado.
- La función **position()** devuelve un número igual a la posición del elemento actual.
/EMPLEADO/EMP_ROW[position()=3], obtiene los elementos del empleado que ocupa la posición 3.
/EMPLEADOS/EMP_ROW[position()=3]/APELLIDO, selecciona el apellido de los elementos cuya posición es menor de 3, es decir, devuelve los apellidos del primer y segundo empleado.
- La función **count()** cuenta el número de elementos seleccionados. Ejemplos:
/EMPLEADOS/count(EMP_ROW), devuelve el número de empleados.
/EMPLEADOS/count(EMP_ROW[DEPT_NO=10]), cuenta el n° de empleados del departamento 10.
/EMPLEADOS/count(EMP_ROW[OFICIO="EMPLEADO" and SALARIO>1300]), cuenta el n° de empleados con oficio EMPLEADO y SALARIO mayor de 1300.
//[@count(*)=3]*, devuelve elementos que tienen 3 hijos.
//[@count(DEP_ROW)=4]*, devuelve los elementos que contienen 4 hijos DEP_ROW, devolverá la etiqueta de departamentos y todas las subetiquetas.
- La función **sum()** devuelve la suma del elemento seleccionado. Ejemplos:
sum(/EMPLEADOS/EMP_ROW/SALARIO), devuelve la suma del SALARIO.
Si la etiqueta a sumar la considera string hay que convertirla a número utilizando la función number.
sum(/EMPLEADOS/EMP_ROW[DEPT_NO=20]/SALARIO), devuelve la suma de SALARIO de los empleados del DEPT_NO 20.

- La función **max()** devuelve el máximo, **min()** devuelve el mínimo y **avg()** devuelve la media del elemento seleccionado. Ejemplos:
max(/EMPLEADOS/EMP_ROW/SALARIO), devuelve el salario máximo.
min(/EMPLEADOS/EMP_ROW/SALARIO), devuelve el salario mínimo.
min(/EMPLEADOS/EMP_ROW[OFICIO="ANALISTA"]/SALARIO), devuelve el salario mínimo de los empleados con OFICIO ANALISTA.
avg(/EMPLEADOS/EMP_ROW/SALARIO), devuelve la media del salario.
avg(/EMPLEADOS/EMP_ROW[DEPT_NO=20]/SALARIO), devuelve la media del salario de los empleados del departamento 20.

- La función **name()** devuelve el nombre del elemento seleccionado. Ejemplos:
/*[name()='APELLIDO'], devuelve todos los apellidos, entre sus etiquetas.
count(/*[name()='APELLIDO']), cuenta las etiquetas con nombre APELLIDO.

- La función **concat(cad1, cad2, ...)** concatena las cadenas. Ejemplos:
/EMPLEADOS/EMP_ROW[DEPT_NO=10]/concat(APELLIDO," - ",OFICIO)
Devuelve el apellido y el oficio concatenados de los empleados del departamento 10
/EMPLEADOS/EMP_ROW/concat(APELLIDO," - ",OFICIO," - ", SALARIO)
Devuelve la concatenación de apellido, oficio y salario de los empleados.

- La función **starts-with(cad1,cad2)** es verdadera cuando la cadena cad1 tiene como prefijo a la cadena cad2. Ejemplos:
/EMPLEADOS/EMP_ROW[starts-with(APELLIDO,'A')], obtiene los elementos de los empleados cuyo APELLIDO empieza por 'A'.
/EMPLEADOS/EMP_ROW[starts-with(OFICIO,'A')]/concat(APELLIDO," - ",OFICIO)
obtiene APELLIDO y NOMBRE concatenados de los empleados cuyo OFICIO empieza por 'A'.

- La función **contains(cad1, cad2)** es verdadera cuando la cadena cad1 contiene a la cadena cad2.
/EMPLEADOS/EMP_ROW[contains(OFICIO,'OR')]/OFICIO, devuelve los oficios que contienen la sílaba 'OR' .
/EMPLEADOS/EMP_ROW [contains(APELLIDO,'A')]/APELLIDO, devuelve los apellidos que contienen una ' A' .

- La función **string-length(argumento)** devuelve el número de caracteres de su argumento.
/EMPLEADOS/EMP_ROW/concat(APELLIDO,' = ', string-length(APELLIDO)), devuelve concatenados el apellido con su número de caracteres.
/EMPLEADOS/EMP_ROW[string-length(APELLIDO)<4], devuelve los datos de los empleados cuyo APELLIDO tiene menos de 4 caracteres.

- Operador matemático **div()** realiza divisiones en punto flotante.
/EMPLEADOS/EMP_ROW/concat(APELLIDO,' ',SALARIO,' - ',SALARIO div 12),
devuelve los datos concatenados de APELLIDO, SALARIO y el salario dividido por 12.
sum(/EMPLEADOS/EMP_ROW/SALARIO) div count(/EMPLEADOS/EMP_ROW), devuelve la suma de salarios dividido por el contador de empleados.
- Operador matemático **mod()** calcula el resto de la división
/EMPLEADOS/EMP_ROW/concat(APELLIDO,' ',SALARIO,' - ',SALARIO mod 12),
devuelve los datos concatenados de APELLIDO, SALARIO y el resto de dividir el SALARIO por 12.
/EMPLEADOS/EMP_ROW[(SALARIO mod 12)=4], devuelve los datos de los empleados cuyo resto de dividir el SALARIO entre 12 sea igual a 4.

✓ **Otras funciones.**

- **data(expresión XPath)**, devuelve el texto de los nodos de la expresión sin las etiquetas.
- **number(argumento)**, para convertir a número el argumento, que puede ser cadena, booleano o un nodo.
- **abs(num)**, devuelve el valor absoluto del número.
- **ceiling(num)**, devuelve el entero más pequeño mayor o igual que la expresión numérica especificada.
- **floor(num)**, devuelve el entero más grande que sea menor o igual que la expresión numérica especificada.
- **round(num)**, redondea el valor de la expresión numérica.
- **string(argumento)**, convierte el argumento en cadena.
- **compare(exp1,exp2)**, compara las dos expresiones, devuelve 0 si son iguales, 1 si exp1>exp2, y -1 si exp1<exp2.
- **substring(cadena,comienzo,num)**, extrae de la *cadena*, desde la posición indicada en comienzo el número de caracteres indicado en num.
- **substring(cadena,comienzo)**, extrae de la *cadena*, los caracteres desde la posición indicada por comienzo, hasta el final.
- **lower-case(cadena)**, convierte a minúscula la *cadena*.
- **upper-case(cadena)**, convierte a mayúscula la *cadena*.
- **translate(cadena1,caract1,caract2)**, reemplaza dentro de *cadena1*, los caracteres que se expresan en *caract1*, por los correspondientes que aparecen en *caract2*, uno por uno.
- **ends-with(cadena1,cadena2)**, devuelve true si la *cadena1* termina en *cadena2*.
- **year-from-date(fecha)**, devuelve el año de la fecha, el formato de fecha es AÑO MES-DIA.
- **month-from-date(fecha)**, devuelve el mes de la fecha.
- **day-from-date(fecha)**, devuelve el día de la fecha

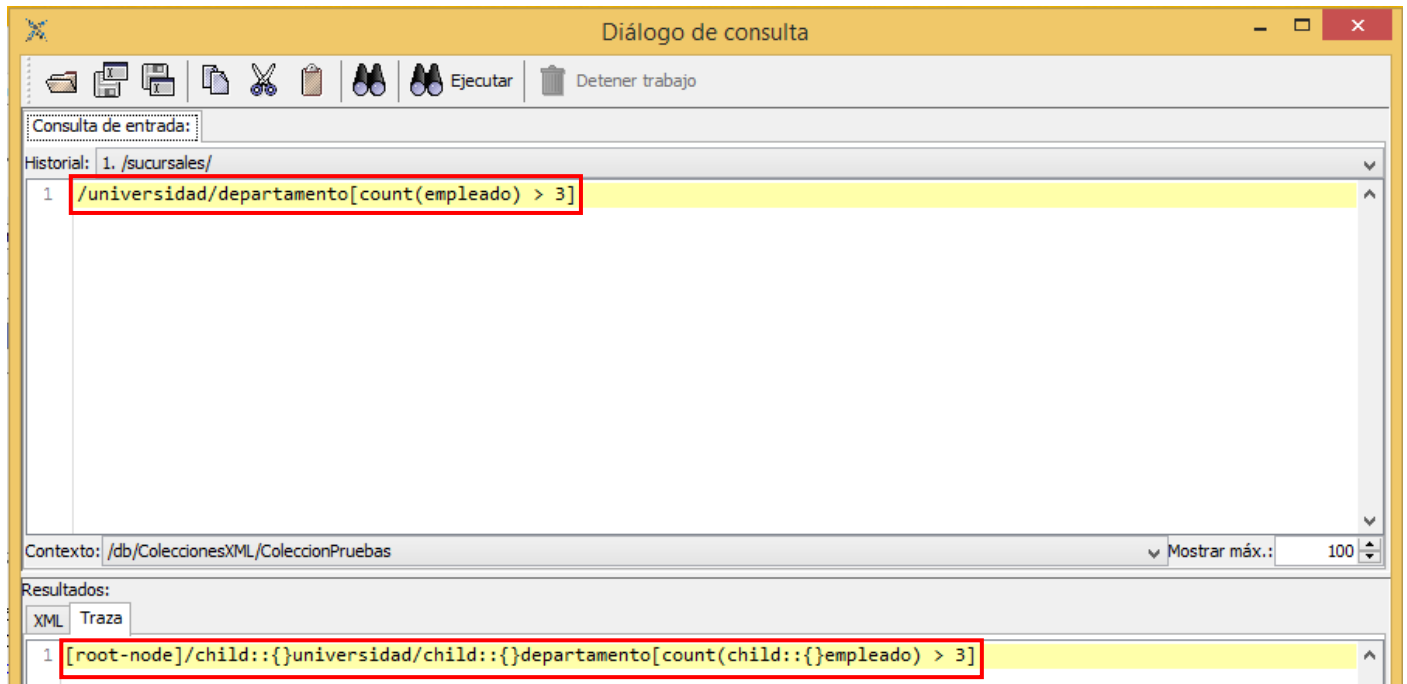
1.6.2.Axis XPath

Un AXIS o eje, especifica la dirección que se va a evaluar, es decir, si nos vamos a mover hacia arriba en la jerarquía o hacia abajo, si va a incluir el nodo actual o no, es decir, define un conjunto de nodos relativo al nodo actual. Los nombres de los ejes son los siguientes:

Nombre de Axis	Resultado
ancestor	Selecciona los antepasados (padres, abuelos, etc.) del nodo actual
ancestor-or-self	Selecciona los antepasados (padres, abuelos, etc.) del nodo actual y el nodo actual en sí
attribute	Selecciona los atributos del nodo actual
child	Selecciona los hijos del nodo actual
descendant	Selecciona los descendientes (hijos, nietos, etc.) del nodo actual
descendant-or-self	Selecciona los descendientes (hijos, nietos, etc.) del nodo actual y el nodo actual en sí
following	Selecciona todo el documento después de la etiqueta de cierre del nodo actual
following-sibling	Selecciona todos los hermanos que siguen al nodo actual
parent	Selecciona el padre del nodo actual
self	Selecciona el nodo actual

La sintaxis para utilizar ejes es la siguiente: ***Nombre_de_eje::nombre_nodo[expresion]***

En la ventana del **Diálogo de consulta**, se puede ver en la parte de *Resultado*, y dentro de la pestaña *Trace*, la traza de las consultas, y en la podemos ver los ejes utilizados.



Ejes en la traza de ejecución de las consultas XPath

Ejemplos con Axis XPath:

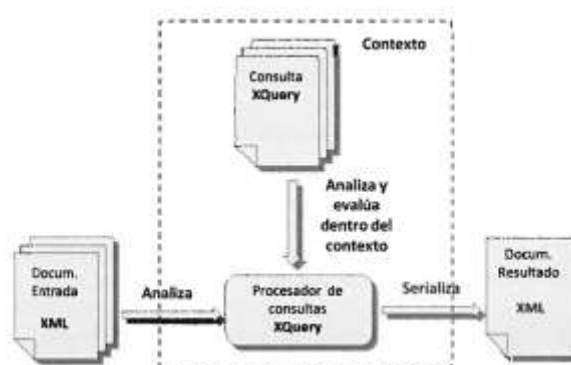
- ✓ **`/universidad/child::*`**, es lo mismo que `/child::universidad/child::element()`. Devuelve todos los hijos de universidad, es decir, los nodos de los departamentos.
- ✓ **`/universidad/departamento/ descendant::*`**, devuelve los descendientes del nodo departamento, esto hace lo mismo: **`/child::universidad/child::departamento/ descendant::element()`**
- ✓ **`/universidad/departamento/ descendant::emplado`**, devuelve los nodos empleado descendientes de los nodos departamento.
- ✓ **`/universidad/ descendant::nombre`**, devuelve todos los elementos nombre descendientes de universidad, tanto nombres de departamentos como de empleados. Si ponemos esto nos devuelve el texto del nombre: **`data(/universidad/ descendant::nombre)`**.
- ✓ **`/universidad/departamento/ following-sibling::*`**, selecciona todos los hermanos de departamento a partir del primero, siguiendo el orden en el documento.
Si ponemos **`/universidad/departamento[2]/ following-sibling::*`**, selecciona todos los hermanos de departamento a partir del segundo.
- ✓ **`//emplado/ following-sibling::node()`**, selecciona todos los hermanos de los elementos empleado que encuentre en el contexto.
En este caso **`//emplado/ following-sibling::emplado[@salario>2100]`**, selecciona todos los hermanos de los elementos empleado que tienen el salario >2100.
- ✓ **`//emplado[nombre="Ana Garcia"]/ following-sibling::*`**, selecciona los nodos hermanos de Ana García.

- ✓ `//empleado[nombre="Ana García"]/following-sibling::empleado[puesto="Profesor"]/nombre/text()`, selecciona los nombres de los empleados hermanos de Ana García que son profesores.
- ✓ `//empleado/parent::departamento/nombre`, selecciona el nombre de los padres de los elementos empleado.
- ✓ `//empleado[nombre="Ana García"]/parent::departamento/nombre`, selecciona el nombre del padre de la empleada Ana García.
- ✓ `//descendant::departamento[1]`, selecciona los descendientes del departamento que ocupa la posición 3 en el documento.
- ✓ `/child::universidad/child::departamento[count(child::empleado) > 3]`, obtiene los departamentos con más de 3 empleados, es lo mismo que `/universidad/departamento[count(empleado) > 3]`.
- ✓ `/child::universidad/child::departamento/child::nombre`, obtiene las etiquetas con los nombres de los departamentos.
- ✓ `/child::universidad/child::departamento/child::nombre/child::text()`, obtiene los nombres de los departamentos.
- ✓ `/child::universidad/child::departamento[attribute::tipo = "B"][count(child::empleado) >= 2]/child::nombre/child::text()`, devuelve el nombre de los departamentos de tipo B y con 2 o más empleados. Es lo mismo que poner `/universidad/departamento[@tipo="B" and count(empleado)>=2]/nombre/text()`

1.6.3.Consultas XQuery

Una consulta en XQuery es una expresión que lee datos de uno o más documentos en XML y devuelve como resultado otra secuencia de datos en XML, en la imagen, se ve el procesamiento básico de una consulta XQUERY. XQuery contiene a XPath, toda expresión de consulta en XPath es válida y devuelve el mismo resultado en XQuery. Xquery nos va a permitir:

- ✓ 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.



Procesamiento de una consulta Xquery

En las consultas XQuery podemos utilizar las siguientes funciones para referimos a colecciones y documentos dentro de la BD, son las siguientes:

- ✓ **collection("/ruta")**, indicamos el camino para referimos a una colección.
- ✓ **doc("/ruta/documento.xml")**, indicamos el camino de un documento de una colección, y el nombre del documento.

Si no indicamos esas funciones la bd busca los elementos en el contexto actual. Así por ejemplo:

1. La consulta **collection(/ColeccionPruebas)**, devuelve el contenido de la colección de ruta absoluta /ColeccionPruebas, es decir, visualiza todos los documentos incluidos en esa colección. Todas las consultas parten de la ruta /db.
2. La consulta **doc("/ColeccionPruebas/productos.xml")** devuelve el documento *productos.xml* completo que se encuentra en *ColeccionPruebas*. Esto hace lo mismo: **doc("/db/ColeccionPruebas/productos.xml")**.

Si en una colección se prevé que pueda haber varias etiquetas raíz con el mismo nombre, es muy importante al hacer consultas sobre un documento indicar el nombre del documento utilizando la función **doc()** para referimos a él.

Otros ejemplos de consultas XQuery utilizando estas funciones:

1. La consulta **collection(/ColeccionPruebas)/departamentos/DEP_ROW** devuelve los nodos DEP_ROW que cuelgan de la etiqueta raíz departamentos, que aparezcan dentro de la colección. Buscará todos los nodos **departamentos/DEP_ROW** que estén dentro de la colección.
2. La consulta **collection(/ColeccionPruebas)/sucursales/sucursal[@codigo='SUC1']** devuelve el nodo sucursal cuyo código es SUC1, y que se encuentra dentro de nodos *sucursales*, y dentro de la colección.
3. **doc('1/ColeccionPruebas/productos.xml')/productos/produc[precio>50]/denominación** esta consulta devuelve los productos cuyo precio sea mayor de 50. Selecciona el documento de la colección con doc, y la consulta solo se realizará para ese documento.
4. **doc("/ColeccionPruebas/universidad.xml")/universidad/departamento[@tipo="A"]**, esta consulta devuelve los departamentos de tipo A, que se encuentran en el documento *universidad.xml*.
5. **También podemos hacer consultas sobre ficheros XML guardados en disco.** En este caso escribiremos la ruta completa de la ubicación del archivo. Por ejemplo, si pongo: **doc("file:///D:/misXMLs/clientes.xml")/clientes/client/nombre**, obtengo los nombres de los clientes del documento *clientes.xml* que se encuentran en la carpeta *misXMLs* de la unidad *D*.

En XQuery las consultas se pueden construir utilizando expresiones **FLWOR** (leído como flower), que corresponde a las siglas de **For, Let, Where, Order** y **Return**. Permite a diferencia de XPath manipular, transformar y organizar los resultados de las consultas. La sintaxis general de una estructura FLWOR es esta:

```

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

```

For: se usa para seleccionar nodos y almacenarlos en una variable, similar a la cláusula from de SQL. Dentro del for escribimos una expresión XPath que seleccionará los nodos. Si se especifica más de una variable en el for se actúa como producto cartesiano. Las variables comienzan con \$.

Las consultas XQuery deben llevar obligatoriamente una orden **Return**, donde indicaremos lo que queremos que nos devuelva la consulta. Por ejemplo, estas consultas devuelven la primera los elementos EMP_ROW, y la segunda los apellidos de los empleados. Unas escritas en Xquery y las otras en XPath:

XQuery	XPath
for \$emp in /EMPLEADOS/EMP_ROW return \$emp	/EMPLEADOS/EMP_ROW
for \$emp in /EMPLEADOS/EMP_ROW return \$emp/APELLIDO	/EMPLEADOS/EMP_ROW/APELLIDO

Let: permite que se asignen valores resultantes de expresiones XPath a variables para simplificar la representación. Se pueden poner varias líneas let una por cada variable, o separar las variables por comas.

En el siguiente ejemplo se crean 2 variables, el APELLIDO del empleado se guarda en **\$nom**, y el OFICIO en **\$ofi**. La salida sale ordenada por OFICIO, y se crea una etiqueta <APE_OFI> </APE_OFI> que incluye el nombre y el oficio concatenado. Se utilizarán las llaves en el return {} para añadir el contenido de las variables. Véase el ejemplo:

```

for $emp in /EMPLEADOS/EMP_ROW
let $nom:=$emp/APELLIDO, $ofi :=$emp/OFICIO
order by $emp/OFICIO
return <APE_OFI> {concat($nom,' ', $ofi)} </APE_OFI>

```

La salida de esta consulta es:

```

<APE OFI>GIL ANALISTA</APE OFI>
<APE=OFI>FERNANDEZ ANALISTA</APE_OFI>

```


En este otro ejemplo se obtienen los nodos:

```
for $emp in /EMPLEADOS/EMP_ROW
let $nom:=$emp/APELLIDO, $oti :=$emp/OFICIO order by $emp/OFICIO
return <APE_OFI> { ($nom,' ', $ofi) } </APE_OFI>
```

La salida de esta consulta es:

```
<APE_OFI>
  <APELLIDO>GIL</APELLIDO> <OFICIO>ANALISTA</OFICIO>
</APE_OFI>
<APE_OFI>
  <APELLIDO>FERNANDEZ</APELLIDO> <OFICIO>ANALISTA</OFICIO>
</APE_OFI>
```

La cláusula let se puede utilizar sin for, prueba el siguiente caso y observa la diferencia:

SIN FOR	CON FOR
<pre>let \$ofi := /EMPLEADOS/EMP_ROW/OFICIO return <OFICIOS>{\$ofi}</OFICIOS></pre>	<pre>for \$ofi in /EMPLEADOS/EMP_ROW/OFICIO return <OFICIOS>{\$ofi}</OFICIOS></pre>
La cláusula let vincula la variable \$ofi con todo el resultado de la expresión. En este caso vincula todos los oficios creando un elemento <OFICIOS> con todos ellos.	La cláusula for vincula la variable \$ofi con cada nodo oficio que encuentre en la colección de datos, creando una elemento por cada oficio. Por eso aparece la etiqueta <OFICIOS> para cada oficio.

- ✓ **Where:** filtra los elementos, eliminando todos los valores que no cumplan las condiciones dadas:
- ✓ **Order by:** ordena los datos según el criterio dado.
- ✓ **Return:** construye el resultado de la consulta en XML, se pueden añadir etiquetas XML a la salida, si añadimos etiquetas los datos a visualizar los encerramos entre llaves {}. Además en el return se puede añadir **condicionales usando if-then-else** y así tener más versatilidad en la salida. Si se usa la condicional, hay que tener en cuenta que la cláusula else es obligatoria y debe aparecer siempre en la expresión condicional, se debe a que toda expresión XQuery debe devolver un valor. Si no existe ningún valor a devolver al no cumplirse la cláusula if, devolvemos una secuencia vacía con **else()**. El siguiente ejemplo devuelve los departamentos de tipo A encerrados en una etiqueta:

```
for $dep in /universidad/departamento
return if ($dep/@tipo='A')
then <tipoA>(data($dep/nombre))</tipoA>
else ()
```

Utilizaremos la función **data()** para extraer el contenido en texto de los elementos. También se utiliza data() para extraer el contenido de los atributos p.e. esta consulta XPath **//empleado/@salario** es errónea, pues salario no es un nodo, pero esta otra consulta **data(//empleado/@salario)** devuelve los salarios.

Dentro de las asignaciones `let` en las consultas XQuery podremos utilizar expresiones del tipo: `let $var:=//empleado/@salario` esto no da error, pero si queremos extraer los datos pondremos `let $var:=data(//empleado/@salario)` o si hay que devolver el salario pondríamos `return data($var)`.

Ejemplos de consultas XQuery:

CONSULTA 1

```
for $emp in /EMPLEADOS/EMP_ROW
order by $emp/APELLIDO
return
if ($emp/OFICIO='DIRECTOR')
then
<DIRECTOR>{$emp/APELLIDO/text()}
</DIRECTOR>
else
<EMPLE>{data($emp/APELLIDO)}
</EMPLE>
```

SALIDA

Devuelve los nombres de los empleados, los que son directores entre las etiquetas `<DIRECTOR>` `</DIRECTOR>`, y los que no lo son entre etiquetas `<EMPLE>` `</EMPLE>`

```
<EMPLE>ALONSO</EMPLE>
<EMPLE>ARROYO</EMPLE>
<DIRECTOR>CEREZO</DIRECTOR>
<EMPLE>FERNANDEZ</EMPLE>
<EMPLE>GIL</EMPLE>
<DIRECTOR>JIMENEZ</DIRECTOR>
<EMPLE>JIMENO</EMPLE>
<EMPLE>MARTIN</EMPLE>
<EMPLE>MUÑOZ</EMPLE>
<DIRECTOR>NEGRO</DIRECTOR>
<EMPLE>REY</EMPLE>
<EMPLE>SALA</EMPLE>
<EMPLE>SANCHEZ</EMPLE>
<EMPLE>TOVAR</EMPLE>
```

CONSULTA 2

```
for $dep in
/universidad/departamento
```

```

let $nom:=$dep/empleado
return
<depart>{data($dep/nombre)}
    <emple>{count($nom)} </emple>
</depart>

```

SALIDA

Devuelve los nombres del departamento y los empleados que tiene entre etiquetas:

```

<depart>Informática<emple>2</emple></depart>
<depart>Matemáticas<emple>4</emple></depart>
<depart>Análisis<emple>2</emple></depart>

```

CONSULTA 3

```

for $dep in
/universidad/departamento
let $emp := $dep/empleado
let
    $sal := $dep/empleado/@salario
return
<depart>{data($dep/nombre)}
    <emple>{count($emp)} </emple>
    <medsal>{avg($sal)} </medsal>
</depart>

```

SALIDA

Obtiene los nombres de departamento, los empleados que tiene y la media del salario entre etiquetas:

```

<depart>Informática<emple>2</emple><medsal>2150</medsal></depart>
<depart>Matemáticas<emple>4</emple><medsal>2200</medsal></depart>
<depart>Análisis<emple>2</emple><medsal>2050</medsal></depart>

```

CONSULTA 4

```

for $dep in
/universidad/departamento
let $emp := $dep/empleado
let $sal := $dep/empleado/@salario
let $maxi := max($dep/empleado/@salario)
let $emplmax := $dep/empleado[@salario = $maxi]
return
<depart>
{data($dep/nombre)}
<numemples>{count($emp)} </numemples>
<medsal>{avg($sal)} </medsal>

```

```

<salariomax>{$maxi}</salariomax>
<emplemax> {$emplmax/nombre/text()} -
            {data($emplmax/@salario)}
</emplemax>
</depart>

```

SALIDA

Obtiene los nombres de departamento, los empleados que tiene y la media del salario entre etiquetas y el empleado con salario máximo.

```

<depart>Informática<numemples>2</numemples><medsal>2150</medsal><salariomax>2300</salariomax><emplemax>Alicia    Martín    -
2300</emplemax></depart>
<depart>Matemáticas<numemples>4</numemples><medsal>2200</medsal><salariomax>2500</salariomax><emplemax>Antonia González -
2500</emplemax></depart>
<depart>Análisis<numemples>2</numemples><medsal>2050</medsal><salariomax>2200</salariomax><emplemax>Mario García -
2200</emplemax></depart>

```

1.6.4. Operadores y funciones más comunes en XQuery

Las funciones y operadores soportados por XQuery prácticamente son los mismos que los soportados por XPath. Soporta operadores y funciones matemáticas, de cadenas, para el tratamiento de expresiones regulares, comparaciones de fechas y horas, manipulación de nodos XML, manipulación de secuencias, comprobación y conversión de tipos y lógica booleana. Los operadores y funciones más comunes se muestran en la siguiente lista.

- ✓ Matemáticos: +, -, *, div (se utiliza div en lugar de la /), idiv (es la división entera), mod.
- ✓ Comparación: =, !=, <, >, <=, >=, not().
- ✓ Secuencia: union(|), 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(), uppers-case(), lower-case(), string().
- ✓ Uso general: **distinct-values()** extrae los valores de un secuencia de nodos y crea una nueva secuencia con valores únicos, eliminando los nodos duplicados, **empty()** devuelve cierto cuando la expresión entre paréntesis esta vacía. Y **exists()** devuelve cierto cuando una secuencia contiene, al menos, un elemento.
- ✓ Los comentarios en XQuery van encerrados entre caras sonrientes: (: Esto es un comentario :)

Ejemplos utilizando el documento EMPLEADOS.xml

CONSULTA 1. Los nombres de oficio que empiezan por P.

```
for $ofi in /EMPLEADOS/EMP_ROW/OFICIO
where starts-with(data($ofi), 'P')
return $ofi
```

SALIDA

```
<OFICIO>PRESIDENTE</OFICIO>
```

CONSULTA 2. Obtiene los nombres de oficio y los empleados de cada oficio. Utiliza la función distinct-values para devolver los distintos oficios.

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

SALIDA

```
EMPLEADO = 4
VENDEDOR = 4
DIRECTOR = 3
ANALISTA = 2
PRESIDENTE = 1
```

CONSULTA 3. Obtiene 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 $cu := count(/EMPLEADOS/EMP_ROW/DEPT_NO = $dep)
let $sala := round(avg(/EMPLEADOS/EMP_ROW[DEPT_NO = $dep] / SALARIO))
return concat('Departamento: ', $dep, '. Num emples = ', $cu, '. Media salario = ', $sala)
```

SALIDA

```
Departamento: 20. Num emples = 1. Media salario = 2274
Departamento: 30. Num emples = 1. Media salario = 1736
Departamento: 10. Num emples = 1. Media salario = 2892
```

Si se desea devolver el resultado entre etiquetas pondremos en el return:

```
return <depart>
    <cod>{$dep}</cod>
    <emples>{$cu}</emples>
    <medsal>{$sala}</medsal>
</depart>
```

1.7. Acceso a eXist desde Java

Ya se ha estudiado en la UNIDAD 1 cómo leer documentos XML, accediendo a su estructura y contenido utilizando los parser o analizadores DOM (Modelo de Objetos de Documento) y SAX (API Simple para XML). En este apartado vamos a ver distintas APIs que acceden a la BD eXist para procesar documentos XML. Estas son:

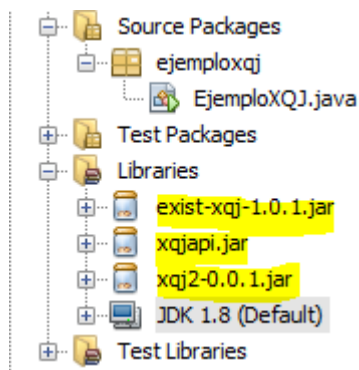
- ✓ **API XML: DB**, cuyo objetivo es la definición de un método común de acceso a SGBD XML, permitiendo la consulta, creación y modificación de contenido. La última actualización de este trabajo es del año 2001 y la actividad desde el año 2005 es prácticamente nula. Sin embargo, aunque apenas se utiliza hay que resaltar que proporciona clases bastante útiles para el manejo de colecciones y documentos. No la estudiaremos.
- ✓ **API XQJ**: es una propuesta de estandarización de interfaz de Java para el acceso a bases de datos XML nativas basado en el modelo XQuery. El objetivo es conseguir un método sencillo y estable de acceso a bases de datos XML nativos, este estándar es un estándar independiente del fabricante, soporta al estándar XQuery 1.0 y muy fácil de utilizar. El inconveniente es que solo se pueden hacer consultas, no se puede operar con colecciones y documentos.

1.7.1. La API XQJ (XQuery)

Es un api similar a JDBC para bases de datos relacionales. Al igual que en JDBC la filosofía gira en torno al origen de datos y la conexión a este, y partiendo de la conexión poder lanzar peticiones al sistema. Para descargar la API accederemos a la URL: <http://xqj.net/exist/>. Descargar el .zip (**exist-xqj-api-1.0.1**).

Para trabajar con esta API necesitamos:

1. Importar las siguientes **librerías**:



2. Los siguientes **import**:

```
import javax.xml.xquery.XQConnection;  
import javax.xml.xquery.XQDataSource;  
import javax.xml.xquery.XQException;  
import javax.xml.xquery.XQPreparedExpression;  
import javax.xml.xquery.XQResultItem;  
import javax.xml.xquery.XQResultSequence;  
import net.xqj.exist.ExistXQDataSource;
```

Configurar una conexión

- ✓ **XQDataSource**: identifica una fuente física de datos a partir de la cual crear conexiones; cada implementación definirá las propiedades necesarias para efectuar la conexión, siendo básicas las propiedades `user` y `password`.

Por ejemplo, este código realiza la conexión con eXist, hay que poner el nombre del servidor (*localhost*), el puerto de la BD (8080), el usuario (*admin*) y su password (*abc123..*). Aunque obligatoria es solo la propiedad `serverName` si estamos en local.

```
XQDataSource server = new ExistXQDataSource();
server.setProperty("serverName", "localhost");
server.setProperty("port", "8080");
server.setProperty("user", "admin");
server.setProperty("password", "abc123..");
```

- ✓ **XQConnection**: representa una sesión con la base de datos, manteniendo información de estado, transacciones, expresiones ejecutadas y resultado. Se obtiene a través de un `XQDataSource`, en nuestro caso es `server`. Ejemplo:

```
XQConnection conn = server.getConnection();
```

También se puede indicar el usuario y password que abre la sesión.

```
XQConnection conn = server.getConnection("admin", "abc123..");
```

La conexión la cerramos escribiendo `conn.close()`;

Clases y métodos para procesar los resultados de una consulta:

- ✓ **XQExpression**: objeto creado a partir de una conexión para la ejecución de una expresión una vez, retornando un **XQResultSetSequence** con los datos obtenidos, podemos decir que en un paso se evalúa el contexto estático o expresión y el dinámico. La ejecución se produce llamando al método **executeQuery**, y se evalúa teniendo en cuenta el contexto estático en vigor.
- ✓ **XQPreparedExpression**: objeto creado a partir de una conexión para la ejecución de una expresión múltiples veces, retornando un **XQResultSetSequence** con los datos obtenidos, en este caso la evaluación del contexto estático solo se hace una vez, mientras que el proceso del contexto dinámico se repite. Igual que en **XQExpression** la ejecución se produce llamando al método **executeQuery**, y se evalúa teniendo en cuenta el contexto estático en vigor.
- ✓ **XQDynamicContext**: representa el contexto dinámico de una expresión, como puede ser la zona horaria y las variables que se van a utilizar en la expresión.
- ✓ **XQStaticContext**: representa el contexto estático para la ejecución de expresiones en esa conexión. Se puede obtener el contexto estático por defecto a través de la conexión. Si se efectúan cambios en el contexto estático no afecta a las expresiones ejecutándose en ese momento, solo en las creadas con posterioridad a la modificación. También es posible especificar un contexto estático para una expresión en concreto, de modo que ignore el contexto de la conexión.
- ✓ **XQItem**: representación de un elemento en **XQuery**. Es inmutable y una vez creado su estado interno no cambia.

- ✓ **XQResultItem**: objeto que representa un elemento de un resultado, inmutable, válido hasta que se llama al método `close` suyo o de la `XQResultSequence` a la que pertenece.

```
XQPreparedExpression consulta;
XQResultSequence resultado;
consulta = conn.prepareExpression("/EMPLEADOS/EMP_ROW[DEPT_NO=20]");
resultado = consulta.executeQuery();
XQResultItem r_item;
while (resultado.next()){
    r_item = (XQResultItem) resultado.getItem();
    System.out.println ("Elemento: " + r_item.getItemAsString(null));
}
```

- ✓ **XQSequence**: representación de una secuencia del modelo de datos *XQuery*, contiene un conjunto de 0 o más *XQItem*. Es un objeto recorrible.
- ✓ **XQResultSequence**: resultado de la ejecución de una sentencia; contiene un conjunto de 0 o más *XQResultItem*. Es un objeto recorrible.

Este ejemplo obtiene los empleados del departamento 10, utilizamos `getItemAsString` para que devuelva los elementos como cadenas:

```
XQPreparedExpression consulta;
XQResultSequence resultado;
consulta = conn.prepareExpression("/EMPLEADOS/EMP_ROW[DEPT_NO=20]");
resultado = consulta.executeQuery();
while (resultado.next()){
    System.out.println ("Elemento: " + resultado.getItemAsString(null));
}
```

Con XQJ no se necesita seleccionar la colección de los documentos XML, la búsqueda la realiza en todas las colecciones. Tampoco admite el uso de sentencias de actualización de `eXist`, con lo cual las inserciones, borrados y actualizaciones resultan bastante engorrosas.

EJEMPLOS:

- ✓ Este método realiza una consulta al documento *productos.xml* de la colección *BDProductosXML*, y devuelve los nodos. Si escribimos esta consulta XQL busca los productos en esa colección.

```
for $pr in collection('/db/ColeccionesXML/BDProductosXML')/productos/produc return $pr
```

Si escribimos esta otra consulta XQJ busca los productos en toda la base de datos y todas las colecciones. Pueden existir varios documentos con nodos */productos/produc*.

```
for $pr in /productos/produc return $pr
```



```

public static void verproductos(){
    try{
        XQDataSource server = new ExistXQDataSource();
        server.setProperty("serverName", "localhost");
        server.setProperty("port", "8080");
        server.setProperty("user", "admin");
        server.setProperty("password", "abc123..");

        //Establecer una sesión
        XQConnection conn = server.getConnection();

        XQPreparedExpression consulta;
        XQResultSequence resultado;

        System.out.println("---Consulta documento productos xml ---");
        consulta = conn.prepareExpression("for $pr in "
            + "collection('/db/ColeccionesXML/BDProductosXML')/productos/produc return $pr");

        resultado = consulta.executeQuery();

        while (resultado.next()){
            System.out.println ("Elemento: " + resultado.getItemAsString(null) );
        }
        conn.close();
    }catch(XQException ex){
        System.out.println("Error al operar");
    }
}

```

✓ Estas instrucciones devuelven el número de productos con precio mayor de 50.

```

public static void verproductos(){
    try{
        XQDataSource server = new ExistXQDataSource();
        server.setProperty("serverName", "localhost");
        server.setProperty("port", "8080");
        server.setProperty("user", "admin");
        server.setProperty("password", "abc123..");

        //Establecer una sesión
        XQConnection conn = server.getConnection();

        XQPreparedExpression consulta;
        XQResultSequence resultado;

        System.out.println("---Consulta documento productos xml ---");
        consulta = conn.prepareExpression ("count(collection('/db/ColeccionesXML/BDProductosXML') "
            + " /productos/produc[precio > 50] )");
        resultado = consulta.executeQuery();
        resultado.next();
    }
}

```

```

        System.out.println("Numero de productos con precio > de 50: " + resultado.getInt());

        conn.close();
    }catch(XQException ex){
        System.out.println("Error al operar");
    }
}

```

- ✓ El siguiente método ejecuta una consulta almacenada en un fichero. El fichero está en la carpeta del proyecto, y se llama ***miconsulta.xq***.

```

public static void ejecutarconsultadefichero(){
    try{
        XQDataSource server = new ExistXQDataSource();
        server.setProperty("serverName", "localhost");
        server.setProperty("port", "8080");
        server.setProperty("user", "admin");
        server.setProperty("password", "abc123..");

        //Establecer una sesión
        XQConnection conn = server.getConnection();

        InputStream query;
        query = new FileInputStream("miconsulta.xq");
        XQExpression xqe = conn.createExpression();
        XQSequence resultado = xqe.executeQuery(query);

        while (resultado.next()){
            System.out.println ("Elemento: " + resultado.getItemAsString(null) );
        }

        conn.close();
    }catch(XQException ex){
        System.out.println("Error en las propiedades del server");
    }catch (FileNotFoundException ex){
        System.out.println("Error fichero.....");
    }
}

```