UNIDAD 5

BASES DE DATOS XML

CONTENIDOS

OBJETIVOS

El alumno al término de esta unidad debe ser capaz de:

- Instalar y utilizar bases de datos XML nativas.
- Realizar consultas en documentos y colecciones XML, utilizando lenguajes XPath y XQuery.
- Desarrollar aplicaciones para acceder a los datos de la base de datos XML nativa.
- Desarrollar aplicaciones para añadir, modificar y eliminar documentos XML de la base de datos.

5.1 BASES DE DATOS NATIVAS XML

A diferencia de las bases de datos relacionales (centradas en los datos), las bases de datos nativas XML, no poseen campos, ni almacenan datos, lo que almacenan son documentos XML, son bases de datos centradas en documentos y la unidad mínima de almacenamiento es el documento XML.

Podemos definir una base de datos nativa XML (o XML nativa), como un sistema de gestión de información que cumple con lo siguiente:

- Define un modelo lógico para un documento XML (y no para los datos que contiene el documento), y almacena y recupera los documentos según este modelo.
- Tiene una relación transparente con el mecanismo de almacenamiento, que debe incorporar las características *ACID* de cualquier SGBD *(Atomicity, Consistency, Isolation and Durability - Atomicidad, Consistencia, Aislamiento y Durabilidad).*
- Incluye un número arbitrario de niveles de datos y complejidad.
- Permite las tecnologías de consulta y transformación propias de XML, XQuery, XPath, XSLT, etc., como vehículo principal de acceso y tratamiento.

Ventajas de las bases de datos XML:

- 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, se pueden considerar como desventajas de las bases de datos XML:

- Puede resultar difícil indexar documentos para realizar búsquedas.
- No suelen ofrecer funciones para la agregación (crucial para el procesamiento de transacciones en línea OLTP -*Online Transaction Processing*) en muchos casos hay que reintroducir todo el documento para modificar una sola 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 mayoria de los SGBD incorporan mecanismos para extraer y almacenar datos en formato XML. A continuación se muestra un ejemplo de soporte de datos XML en Oracle y en MySQL:

– La siguiente select de ORACLE devuelve las filas de la tabla EMPLE en formato XML:

```
SELECT XMLELEMENT("EMP_ROW",
          XMLFOREST(EMP_NO  ,APELLIDO,OFICIO, DIR, FECHA_ALT ,
          SALARIO , COMISION ,DEPT_NO )) FILA_EN_XML
     FROM EMPLE;
```

- **Creación de una tabla que almacena datos del tipo XML (el tipo de dato XMLTYPE permite almacenar y consultar datos XML):**

```
CREATE TABLE TABLA_XML_PRUEBA (COD NUMBER, DATOS XMLTYPE);
```

- **Insertar filas en formato XML:**

```
INSERT INTO TABLA_XML_PRUEBA VALUES (1,
XMLTYPE('<FILA_EMP><EMP_NO>123</EMP_NO><APELLIDO>RAMOS MARTÍN</APELLIDO>
<OFICIO>PROFESORA</OFICIO><SALARIO>1500</SALARIO></FILA_EMP>'));

INSERT INTO TABLA_XML_PRUEBA VALUES (1, XMLTYPE('<FILA_EMP><EMP_NO>124</EMP_NO>
<APELLIDO>GARCÍA SALADO</APELLIDO><OFICIO>PONTANERO</OFICIO>
<SALARIO>1700</SALARIO></FILA_EMP>'));
```

- **Extracción de datos de la tabla, se utilizan expresiones XPath:**

Visualiza los apellidos de los empleados:

```
SELECT EXTRACTVALUE(DATOS,'/FILA_EMP/APELLIDO') FROM TABLA_XML_PRUEBA;
```

Visualiza el nombre del empleado con número de empleado = 123

```
SELECT EXTRACTVALUE(DATOS,'/FILA_EMP/APELLIDO') FROM TABLA_XML_PRUEBA
WHERE EXISTSNODE(DATOS, '/FILA_EMP[EMP_NO=123]') = 1;
```

- En el siguiente ejemplo se muestra como MySQL devuelve los datos de una tabla en formato XML. Por ejemplo, disponemos de la BD en MySQL *ejemplo* y se desea obtener los datos de la tabla *departamentos* en formato XML. El usuario de la BD y su clave se llama también *ejemplo*.

  Desde la línea de comando y en la carpeta donde se encuentra instalado MySQL (por ejemplo, desde la carpeta D:\xampp\mysql\bin>) escribiremos la siguiente orden para obtener los datos de la tabla *departamentos*:

```
mysql --xml -u ejemplo -p -e "select * from departamentos;" ejemplo
```

  Donde --xml, se utiliza para producir una salida en XML.
  -u, a continuación se pone el nombre de usuario de la BD *ejemplo*.
  -p, al pulsar en la tecla [Intro] para ejecutar la orden nos pide la clave del usuario (*ejemplo*)
  -e, ejecuta el comando y sale de MySQL.

El resultado es:

```
<?xml version="1.0"?>
<resultset statement="select * from departamentos"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <row>
       <field name="dept_no">10</field>
       <field name="dnombre">CONTABILIDAD</field>
       <field name="loc">SEVILLA</field>
  </row>
  <row>
       <field name="dept_no">20</field>
       <field name="dnombre">INVESTIGACIÓN</field>
       <field name="loc">MADRID</field>
  </row>
  <row>
       <field name="dept_no">30</field>
       <field name="dnombre">VENTAS</field>
       <field name="loc">BARCELONA</field>
  </row>
  <row>
       <field name="dept_no">40</field>
       <field name="dnombre">PRODUCCIÓN</field>
       <field name="loc">BILBAO</field>
  </row>
</resultset>
```


Si deseamos redireccionar la salida al fichero *departamentos.xml* escribiremos en la misma línea:

```
mysql --xml -u ejemplo -p -e "select * from departamentos;" ejemplo
>departamentos.xml
```

5.2 BASE 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; 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 *XML Schema* asociado, y dentro de una colección pueden almacenarse documentos de cualquier tipo.

En la carpeta eXist\webapp\WEB-INF\data es donde se guardan los ficheros más importantes de la BD:

- ***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 del 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.
- ***elements.dbx,*** es donde se guarda el índice de elementos y atributos. El gestor automáticamente indexa todos los documentos utilizando índices numéricos para identificar los nodos del mismo (elementos, atributos, texto y comentarios). Durante la indexación se asigna un identificador único a cada documento de la colección, que es almacenado también junto al índice.

- ***words.dbx,*** por defecto eXist indexa todos los nodos de texto y valores de atributos dividiendo el texto en palabras. En este fichero se almacena esta información. Cada entrada del índice está formada por un par <id de colección, palabra> y después una lista al nodo que contiene dicha palabra.

5.2.1 INSTALACIÓN DE EXIST

eXist puede funcionar de distintos modos:

- Funcionando como servidor autónomo (ofreciendo servicios de llamada remota a procedimientos que funciona 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://exist-db.org/exist/download.xml podremos descargar la última versión de la BD. En este caso se instalará la versión estable ***eXist-setup-1.4.2-rev16251.jar***. Se instala utilizando el fichero JAR proporcionado en la web que ha de invocarse desde la línea de comandos (método recomendado: *java -jar eXist-setup-1.4.2-rev16251.jar*) o bien haciendo doble clic sobre el icono correspondiente una vez descargado.

En el proceso de instalación, se nos pide introducir la ruta donde está instalado Java JDK (mínimo versión 5, C:\Program Files\Java\jdk1.6.0_29). También indicaremos donde se desea que se instale la BD, nos pedirá contraseña para el usuario administrador de la BD *admin* y marcaremos que instale todas las aplicaciones, véase la Figura 5.1.

Figura 5.1. Instalación de eXist.

Una vez instalado para poder trabajar con la BD primero hay que iniciar el servidor y luego escribimos en el navegador: http://localhost:8080/exist/ . También se puede instalar el servidor como un servicio (*Install eXist as Service*), en este caso se iniciaría la BD de forma automática. En la Figura 5.2 se muestra el menú de eXist.

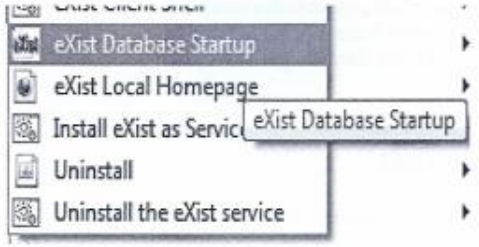

Figura 5.2. Menú de eXist.

Al arrancar eXist se ejecuta ***startup.bat*** (Windows) o ***startup.sh*** en (Linux), que se encuentra en la carpeta *bin* de eXist. Una vez arrancado eXist no se puede cerrar el programa mientras se esté trabajando con dicho SGBD. Si se hace, automáticamente el sistema deja de funcionar. En la Figura 5.3 se muestra la pantalla inicial de la BD eXist.

También es importante tener en cuenta que es posible que el puerto de conexión de la base de datos (el 8080) se encuentre bloqueado por alguna aplicación (por ejemplo, por el Firewall de Windows). Hay que asegurar que el puerto está abierto o habilitado.

Figura 5.3. Vista inicial de eXist.

5.2.2 PRIMEROS PASOS CON EXIST

Lo primero que vamos a hacer es cargar un fichero XML que nos va a servir como base de datos para realizar consultas utilizando XQuery y XPath, así pues será necesario crear una colección y luego almacenar el documento XML en la colección.

En el menú de la izquierda bajando la pantalla, aparece el menú de *Administración*, seleccionamos el elemento ***Admin*** y accedemos a la administración de la BD eXist, nos conectamos con el usuario ***admin***, y escribimos la contraseña que hayamos elegido en la instalación. Véase la Figura 5.4.

Figura 5.4. Panel de administración de eXist.

Desde la ventana de administración podremos crear usuarios si seleccionamos ***User Management***, se pueden instalar los ejemplos que vienen con la BD para hacer pruebas de consultas desde ***Examples Setup***, o se pueden realizar copias de seguridad desde la opción ***Backups***.

En nuestro caso deseamos crear una colección de documentos XML y subir los documentos *empleados.xml* y *departamentos.xml* para luego realizar consultas. Hacemos clic en ***Browse Collections***, creamos la colección ***Pruebas***, haciendo clic en ***Create Collection***. Entramos en la colección creada y subimos los documentos que se encuentran en la carpeta de la unidad *empleados.xml* y *departamentos.xml*. En la Figura 5.5 vemos como queda la colección creada con los dos documentos XML subidos. Recuerda que los documentos XML deben estar bien formados para no tener problemas en la subida de los mismos a la base de datos.

Figura 5.5. Colección *Pruebas*.

Una vez creada la colección hay que abrir el gestor de consultas de eXist, para ello regresamos a la ventana inicial pulsando el enlace *Home*, y desde ahí se selecciona ***XQuery Sandbox*** que aparece en el menú de la izquierda dentro de *Examples*. Desde el gestor de consultas se podrán realizar consultas a los documentos XML de nuestra colección, en la Figura 5.6 se muestra la ventana para la realización de consultas, en la parte superior se escribe la consulta y en la inferior se muestra el resultado. El resultado de las consultas son etiquetas XML.

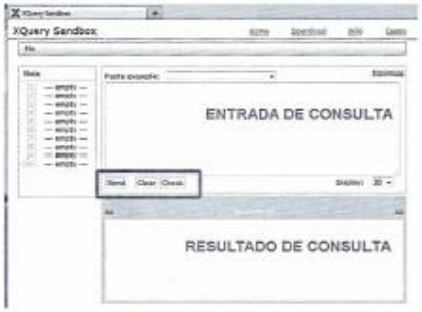

Figura 5.6. ***XQuery Sandbox***, gestor de consultas de eXist.

5.2.3 EL CLIENTE DE ADMINISTRACIÓN DE EXIST

Para ejecutar el cliente abrimos ***eXist Client Shell***, desde el menú de programas de eXist (Inicio -> Programas -> eXist XML Database), o bien ejecutando *eXist\bin\client.bat*. También se descarga el cliente desde la URL *http://localhost:8080/exist/webstart/**exist.jnlp*** y luego se ejecuta. A continuación se muestra la ventana donde pide que se configure la conexión o se elija una ya creada. Tecleamos el nombre de usuario, su contraseña, la URL de la BD y un título para la conexión, estos datos se quedarán guardados para posteriores conexiones, véase la Figura 5.7.

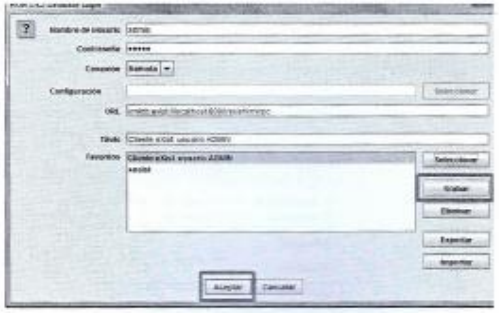

Figura 5.7. Ejecución de *eXist Client Shell* y configuración de la conexión.

A continuación se muestra la ventana del *Cliente de Administración eXist* (Figura 5.8), 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 y restaurarlas, administrar usuarios y realizar consultas XPath, entre otras operaciones.

Figura 5.8. Cliente de administración de eXist.

Si pulsamos al botón *Consultar la BD usando XPath*, aparece la ventana de consultas ***Query Dialog***, desde aquí podremos elegir 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 *Submit* y en la parte inferior se muestra el resultado, también se puede ver la traza de ejecución seguida en la ejecución de la consulta. En la Figura 5.9 se muestra la ventana de *Query Dialog*.

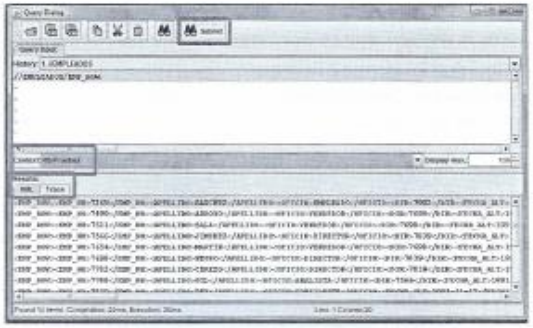

Figura 5.9. Ventana de consultas ***Query Dialog*** de eXist.

5.3 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 ficheros XML hasta bases de datos relacionales 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.

5.3.1 EXPRESIONES XPATH

XPath es un lenguaje que permite seleccionar nodos de un documento XML y calcular valores a partir de su contenido. Existen dos 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 encontrarnos con los siguientes tipos de nodos:

- ***nodo 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 empezaremos con la sintaxis abreviada, así pues los descriptores 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 con // 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 *Pruebas*, que contiene los documentos *departamentos.xml* y *empleados.xml*, cuyas estructuras se muestras en la Figura 5.10. Realizar desde ***Query Dialog*** o ***XQuery Sandbox*** las siguientes consultas:

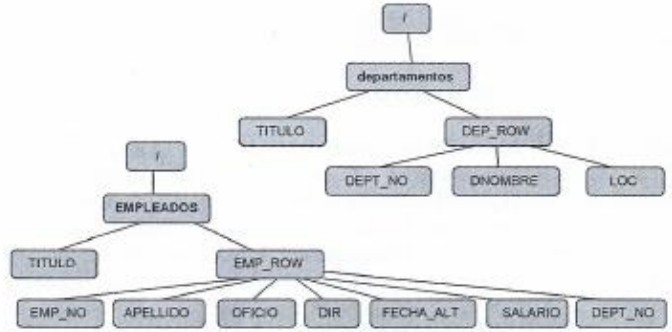

Figura 5.10. Estructuras de *departamentos.xml* y *empleados.xml*.

- `/`, si ejecutamos esta orden en el ***Query Dialog*** y se está dentro del contexto /db/Pruebas devuelve todos los datos de 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 *Pruebas*. Si se ejecuta desde XQuery Sandbox, se obtiene otro resultado, pues el contexto no es el mismo.
- `/departamentos`, devuelve todos los datos de los departamentos, es decir incluye todas las etiquetas que cuelgan del nodo raíz *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 departamentos de cada DEP_ROW, entre etiquetas.
- `/departamentos/DEP_ROW/DNOMBRE/text()`, devuelve los nombres de departamentos de cada DEP_ROW, ya sin las etiquetas.
- `//LOC/text()`, devuelve todas las localidades, de toda la colección (//), solo hay 4.

- `//DEPT_NO`, devuelve todos los números de departamentos, entre etiquetas. Observa que en este caso devuelve 18 filas en lugar de 4, es porque recoge todos los elementos DEPT_NO de la colección y, recuerda que en la colección también hemos incluido el documento *empleados.xml*, que tiene 14 empleados con su DEPT_NO, véase la Figura 5.11.

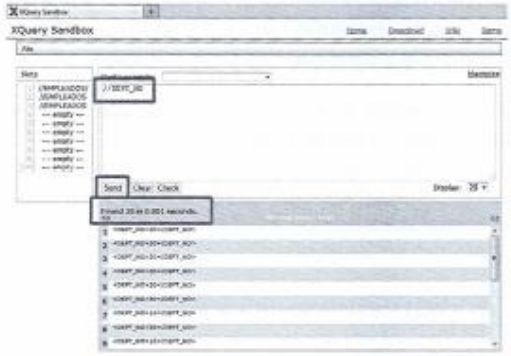

Figura 5.11. ***XQuery Sanbox***, ejecución de la consulta `//DEPT_NO`.

- ***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.
- ***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.
  - Lo función **position()** devuelve un número igual a la posición del elemento actual.
    `/EMPLEADOS/EMP_ROW[position()=3]`, obtiene los elementos del empleado que ocupa la posición 3.
    `/EMPLEADOS/EMP_ROW[position()<3]/APELLIDO`, selecciona el apellido del 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 número de empleados del departamento 10.
    `/EMPLEADOS/count(EMP_ROW[OFICIO="EMPLEADO" and SALARIO>1300])`, cuenta el nº 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 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.

- 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. Véase la Figura 5.12.

Figura 5.12. Ejecución de una consulta de concatenación.

- 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 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.

Puedes encontrar más funciones en la URL: http://www.w3schools.com/xpath/xpath_functions.asp

Actividad 1: Sube el documento *productos.xml* dentro de la colección *Pruebas*. Este documento contiene los datos de los productos de una distribuidora de componentes informáticos. Por cada producto tenemos el código, la denominación, el precio, el stock actual, el stock mínimo y el código de zona. Estos datos son:

```
<produc>
  <cod_prod>xxxxxx</cod_prod>
  <denominacion>xxxxxxxxxxx</denominacion>
  <precio>xxxx</precio>
  <stock_actual>xxx</stock_actual>
  <stock_minimo>xxxx</stock_minimo>
  <cod_zona>xxxx</cod_zona>
</produc>
```

Realiza las siguientes consultas XPath:

- Obtén los nodos denominación y precio de todos los productos.
- Obtén los nodos de los productos que sean placas base.
- Obtén los nodos de los productos con precio mayor de 60 € y de la zona 20.
- Obtén el número de productos que sean memorias y de la zona 10.
- Obtén la media de precio de los micros.
- Obtén los datos de los productos cuyo stock mínimo sea mayor que su stock actual.
- Obtén el nombre de producto y el precio de aquellos cuyo stock mínimo sea mayor que su stock actual y sean de la zona 40
- Obtén el producto más caro.
- Obtén el producto más barato de la zona 20.
- Obtén el producto más caro de la zona 10.

5.3.2 NODOS ATRIBUTOS XPATH

Un nodo puede tener tantos atributos como se desee y para cada uno se le creará un ***nodo atributo***. Los *nodos atributo* NO se consideran como hijos, sino más bien como etiquetas añadidas al nodo elemento. Cada *nodo atributo* consta de un nombre, un valor (que es siempre una cadena) y un posible "espacio de nombres".

Partimos del documento *universidad.xml*, que se encuentra en la carpeta de la unidad. Sube este documento dentro de la colección *Pruebas* de la BD.


```
<universidad>
  <departamento telefono="112233"
tipo="A">
    <codigo>IFC1</codigo>
    <nombre>Informática</nombre>
    <empleado salario="2000">
       <puesto>Asociado</puesto>
       <nombre>Juan Parra</nombre>
    </empleado>
    <empleado salario="2300">
       <puesto>Profesor</puesto>
       <nombre>Alicia Martín</nombre>
    </empleado>
  </departamento>

  <departamento telefono="990033"
tipo="A">
    <codigo>MAT1</codigo>
    <nombre>Matemáticas</nombre>
    <empleado salario="1900">
       <puesto>Técnico</puesto>
       <nombre>Ana García</nombre>
    </empleado>
    <empleado salario="2100">
       <puesto>Profesor</puesto>
       <nombre>Mª Jesús Ramos</nombre>
    </empleado>
```

```
    <empleado salario="2300">
       <puesto>Profesor</puesto>
       <nombre>Pedro Paniagua</nombre>
    </empleado>
    <empleado salario="2500">
       <puesto>Tutor</puesto>
       <nombre>Antonia González</nombre>
    </empleado>
  </departamento>

  <departamento telefono="880833"
tipo="B">
    <codigo>MAT2</codigo>
    <nombre>Análisis</nombre>
    <empleado salario="1900">
       <puesto>Asociado</puesto>
       <nombre>Laura Ruiz</nombre>
    </empleado>
    <empleado salario="2200">
       <puesto>Asociado</puesto>
       <nombre>Mario García</nombre>
    </empleado>
  </departamento>

</universidad >
```

En este documento los elementos o nodos que llevan atributos son los siguientes: los elementos departamento llevan los atributos teléfono y tipo, y los elementos empleado llevan el atributo salario. El árbol del documento se muestra en la Figura 5.13, los atributos se representan con elipses.

Figura 5.13. Árbol del documento *universidad.xml*.

Para referirnos a los atributos de los elementos se usa @ antes del nombre, por ejemplo @telefono, @tipo, @salario. En un descriptor de ruta los atributos se nombran como si fueran etiquetas hijo pero anteponiendo @.

Ejemplos:

- `/universidad/departamento[@tipo]`, se obtienen los datos de los departamentos que tengan el atributo tipo. Si ponemos `data(/universidad/departamento[@tipo])`, nos devuelve los datos sin las etiquetas.
- `/universidad/departamento/empleado[@salario]`, se obtienen los datos de los empleados que tengan el atributo salario.
- `/universidad/departamento[@telefono="990033"]`, se obtienen los datos del departamento cuyo teléfono es 990033. Si ponemos `data(/universidad/departamento[@telefono="990033"])`, devuelve lo mismo pero sin las etiquetas de los elementos.
- `/universidad/departamento[@telefono="990033"]/nombre/text()`, se obtienen el nombre de departamento cuyo teléfono es 990033.
- `//departamento[@tipo='B']`, se obtienen los datos de los departamentos cuyo tipo es B.
- `/universidad/departamento[@tipo="A"]/empleado`, se obtienen los datos de los empleados de los departamentos del tipo A.
- `/universidad/departamento/empleado[@salario>"2100"]`, se obtienen los datos de los empleados cuyo salario es mayor de 2100.
- `/universidad/departamento/empleado[@salario>"2100"]/nombre/text()`, se obtienen los nombres de los empleados cuyo salario es mayor de 2100.
- `/universidad/departamento/empleado[@salario>"2100"] /concat(nombre,' ', @salario)`, se obtienen los datos concatenados del nombre de empleado y su salario, de los empleados cuyo salario es mayor de 2100.
- `/universidad/departamento[@tipo="A"]/count(empleado)`, devuelve el número de empleados que hay en los departamentos de tipo=A.
- `/universidad/departamento[@tipo="A"]/concat(nombre,' ',count(empleado))`, devuelve por cada departamento del tipo=A, la concatenación de su nombre y el número de empleados.
- `/universidad/departamento/concat(nombre,' ',count(empleado))`, devuelve el número de empleados por cada departamento
- `sum(//empleado/@salario)`, devuelve la suma total del salario de todos los empleados, hace lo mismo que esto: `sum(/universidad/departamento/empleado/@salario)`
- `/universidad/departamento/concat(nombre,' Total=', sum(empleado/@salario))`, obtiene por cada departamento la concatenación de su nombre y el total salario.
- `min(//empleado/@salario)`, devuelve el salario mínimo de todos los empleados.
- `/universidad/departamento/concat(nombre,' Mínimo=', min(empleado/@salario))`
  `/universidad/departamento/concat(nombre,' Máximo=', max(empleado/@salario))`
  La primera obtiene por cada departamento la concatenación de su nombre y el mínimo salario, y la segunda el máximo salario.

- `/universidad/departamento/concat(nombre,' Media=', avg(empleado/@salario))`, obtiene por cada departamento la concatenación de su nombre y la media de salario.
- `/universidad/departamento[count(empleado)>3]`, obtiene los datos de departamentos con más de 3 empleados. `/universidad/departamento[count(empleado)>3]/nombre/text()`, en este caso devuelve el nombre de los departamentos con más de 3 empleados.
- `/universidad/departamento[@tipo="A" and count(empleado)>2]/nombre/text()`, devuelve el nombre de los departamentos de tipo A y con más de 2 empleados.

Actividad 2: Sube el documento sucursales.xml dentro de la colección *Pruebas*. Este documento contiene los datos de las sucursales de un banco. Por cada sucursal tenemos el teléfono, el código, el director de la sucursal, la población y las cuentas de la sucursal. Y por cada cuenta tenemos el tipo de cuenta AHORRO o PENSIONES, el nombre de la cuenta, el numero, el saldohaber y el saldodebe. Estos datos son:

```
<sucursales>
  <sucursal telefono="xxxxxxxx" codigo="xxxx">
      <director>xxxxxxxxxxxxxxx</director>
          <poblacion>xxxxxxxxxxx</poblacion>
          <cuenta tipo="xxxxxxxxx">
                        <nombre>xxxx</nombre>
                              <numero>xxxx</numero>
                              <saldohaber>xxxxxx</saldohaber>
                              <saldodebe>xxxxx</saldodebe>
                    </cuenta>
                . . . . . .
  </sucursal>
  . . . . . .
</sucursales>
```

Realiza las siguientes consultas XPath:

- Obtén los datos de las cuentas bancarias cuyo stipo sea AHORRO.
- Obtén por cada sucursal la concatenación de su código y el número de cuentas del tipo AHORRO que tiene.
- Obtén las cuentas de tipo PENSIONES de la sucursal con código SUC3.
- Obtén por cada sucursal la concatenación de los datos, código sucursal, director y total saldo haber.
- Obtén todos los elementos de las sucursales con más de 3 cuentas.
- Obtén todos los elementos de las sucursales con más de 3 cuentas del tipo AHORRO.
- Obtén los nodos del director y la población de las sucursales con más de 3 cuentas.
- Obtén el número de sucursales cuya población sea Madrid.
- Obtén por cada sucursal, su código y la suma de las aportaciones de las cuentas del tipo PENSIONES.

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()`.

5.3.4 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 Figura 5.15 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.

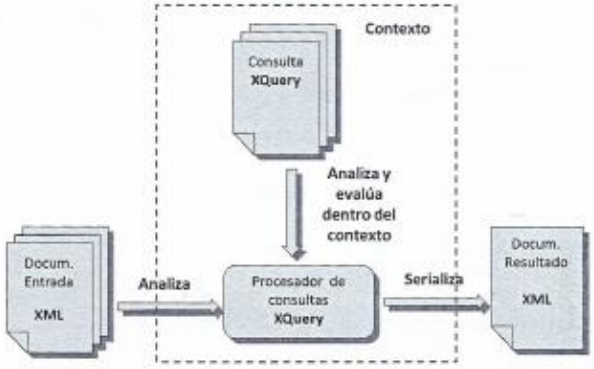

Figura 5.15. Procesamiento de una consulta XQuery.

En XQuery las consultas siguen la norma **FLWOR** (leído como flower), corresponde a las siglas de ***For***, ***Let***, ***Where***, ***Order*** y ***Return***. Permite a diferencia de XPat 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 devuelve 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 cláusula let se puede utilizar sin for, prueba el siguiente caso y observa la diferencia:

SIN FOR	CON FOR
`let $ofi := /EMPLEADOS/EMP_ROW/OFICIO return <OFICIOS>{$ofi}</OFICIOS>`	`for $ofi in /EMPLEADOS/EMP_ROW/OFICIO return <OFICIOS>{$ofi}</OFICIOS>`
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 pueden añadir ***condicionales usando if-then-else*** y así tener más versatilidad en la salida.
  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.ej. 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 $var:=data(//empleado/@salario)`** o **`return data($var)`**

Ejemplos:

`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>`	Devuelve los nombres de los empleados, los que son directores entre las etiquetas <DIRECTOR> </DIRECTOR> y los que no lo son entre las etiquetas <EMPLE></EMPLE>.
	
	`<EMPLE>ALONSO</EMPLE>`
	`<EMPLE>ARROYO</EMPLE>`
	`<DIRECTOR>CEREZO</DIRECTOR>`
	`<EMPLE>FERNANDEZ</EMPLE>`
	`<EMPLE>GIL</EMPLE>`
	`<DIRECTOR>JIMENEZ</DIRECTOR>`
	. . . . .

`for $de in doc('file:///D:/XML /pruebaxquery/NUEVOS_DEP.xml')/NUEVOS_DEP/DEP_ROW return $de`	Devuelve los nodos DEP_ROW de un documento ubicado en una carpeta del disco duro.

`for $prof in /universidad/departamento[@tipo='A']/empleado let $profe:= $prof/nombre, $puesto:= $prof/puesto where $puesto='Profesor' return $profe`	Obtiene los nombres de empleados de los departamentos de tipo A, cuyo puesto es Profesor. Esto hace lo mismo:
	
	`for $prof in /universidad/departamento[@tipo='A']/empleado where $prof/puesto='Profesor' return $prof/nombre`
	
	El resultado es:
	
	`<nombre>Alicia Martin</nombre>`
	`<nombre>Mª Jesús Ramos</nombre>`
	`<nombre>Pedro Paniagua</nombre>`

`for $dep in /universidad/departamento return if ($dep/@tipo='A') then <tipoA>{data($dep/nombre)}</tipoA> else <tipoB>{data($dep/nombre)}</tipoB>`	Devuelve el nombre de departamento encerrado entre las etiquetas <tipoA></tipoA>, si es del tipo = A, y <tipoB></tipoB>, si no lo es.
	
	`<tipoA>Informática</tipoA>`
	`<tipoA>Matemáticas</tipoA>`
	`<tipoB>Análisis</tipoB>`

`for $dep in /universidad/departamento let $nom:= $dep/empleado return <depart>{data($dep/nombre)} <emple>{count($nom)}</emple></depart>`	Obtiene los nombres de 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>`

`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>`	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>`

5.3.5 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 tabla.

- 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(), 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. 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:

- 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>
```

- 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
```

- 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 = 5. Media salario = 2274
Departamento: 30. Num emples = 6. Media salario = 1736
Departamento: 10. Num emples = 3. Media salario = 2892
```

Si se desea devolver el resultado entre etiquetas pondremos en el return (p.ej):

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

```
<depart><cod>20</cod><emples>5</emples><medsal>2274</medsal></depart>
<depart><cod>30</cod><emples>6</emples><medsal>1736</medsal></depart>
<depart><cod>10</cod><emples>3</emples><medsal>2892</medsal></depart>
```


Actividad 3: Utilizando el documento ***productos.xml.*** Realiza las siguientes consultas *XQuery*:

- Obtén por cada zona el número de productos que tiene.
- Obtén la denominación de los productos entres las etiquetas <zona10></zona10> si son del código de zona 10, <zona20></zona20> si son de la zona 20, <zona30></zona30> si son de la 30 y <zona40></zona40> si son de la 40.
- Obtén por cada zona la denominación del o de los productos más caros.
- Obtén la denominación de los productos contenida entre las etiquetas <placa></placa> para los productos en cuya denominación aparece la palabra *Placa Base*, <memoria></memoria>, para los que contienen a la palabra *Memoria* <micro></micro>, para los que contienen la palabra *Micro* y <otros></otros> para el resto de productos.

Utilizando el documento ***sucursales.xml.*** Realiza las siguientes consultas XQuery:

- Devuelve el código de sucursal y el número de cuentas que tiene de tipo AHORRO y de tipo PENSIONES
- Devuelve por cada sucursal el código de sucursal, el director, la población, la suma del total debe y la suma del total haber de sus cuentas.
- Devuelve el nombre de los directores, el código de sucursal y la población de las sucursales con más de 3 cuentas.
- Devuelve por cada sucursal , el código de sucursal y los datos de las cuentas con más saldo debe.
- Devuelve la cuenta del tipo PENSIONES que ha hecho más aportación.

5.3.6 CONSULTAS COMPLEJAS CON XQUERY

Dentro de las consultas XQuery podremos trabajar con varios documentos xml para extraer su información, podremos incluir tantas sentencias for como se deseen, incluso dentro del return. Además podremos añadir, borrar e incluso modificar elementos, bien generando un documento xml nuevo o utilizando las sentencias de actualización de eXist. A continuación se muestran ejemplos de diversa complejidad:

- **Joins de documentos.**

  - Visualizar por cada empleado del documento *empleados.xml*, su apellido, su número de departamento y el nombre del departamento que se encuentra en el documento *departamentos.xml*

```
for $emp in (/EMPLEADOS/EMP_ROW)
let $emple:= $emp/APELLIDO
let $dep:= $emp/DEPT_NO
let $dnom:= (/departamentos/DEP_ROW[DEPT_NO =$dep]/DNOMBRE)
return <res>{$emple, $dep, $dnom} </res>
```