

## 1 BASES DE DATOS NATIVAS XML

Las bases de datos nativas XML almacenan documentos XML, son bases de datos centradas en documentos y la unidad mínima de almacenamiento es el documento XML.

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 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 mayoría 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>FONTANERO</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
```

## 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. Con el SGBD se dan aplicaciones que permiten ejecutar consultas directamente sobre la BD.

**Los documentos XML se almacenan en colecciones**, desde un punto de vista práctico el almacén de datos funciona como un sistema de ficheros. Cada documento está en una colección. 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).
- **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.

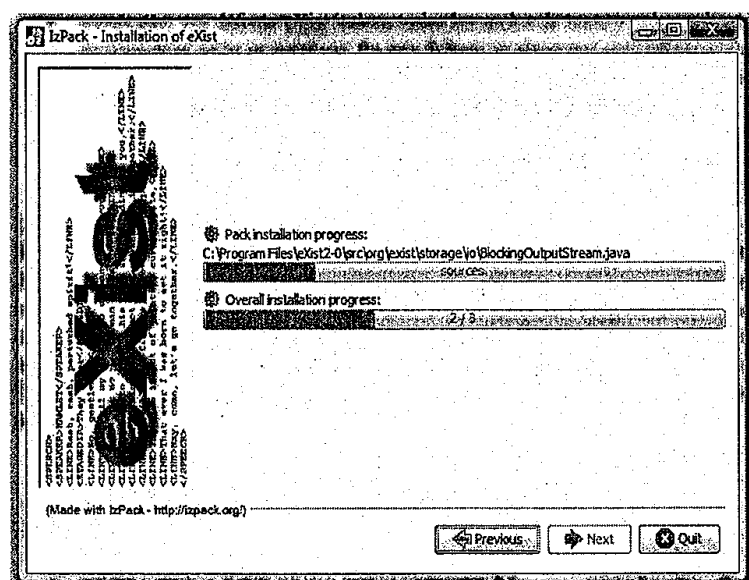
### 2.1 INSTALACIÓN DE EXIST

eXist puede funcionar de distintos modos:

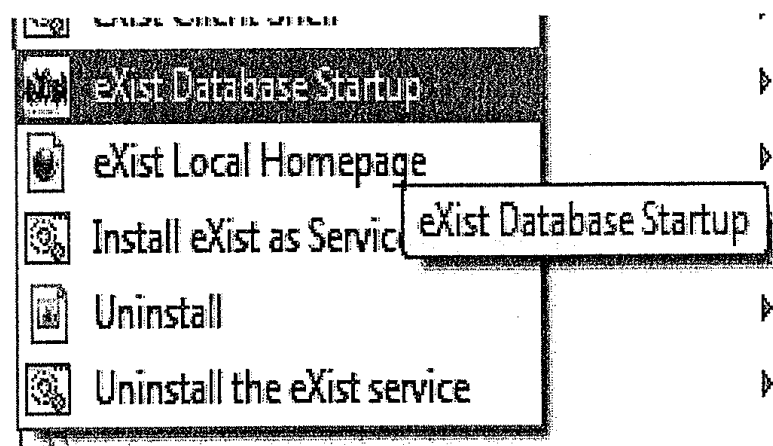
- Funcionando como servidor.
- Insertado dentro de una aplicación Java.
- En un servidor J2EE.

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**.

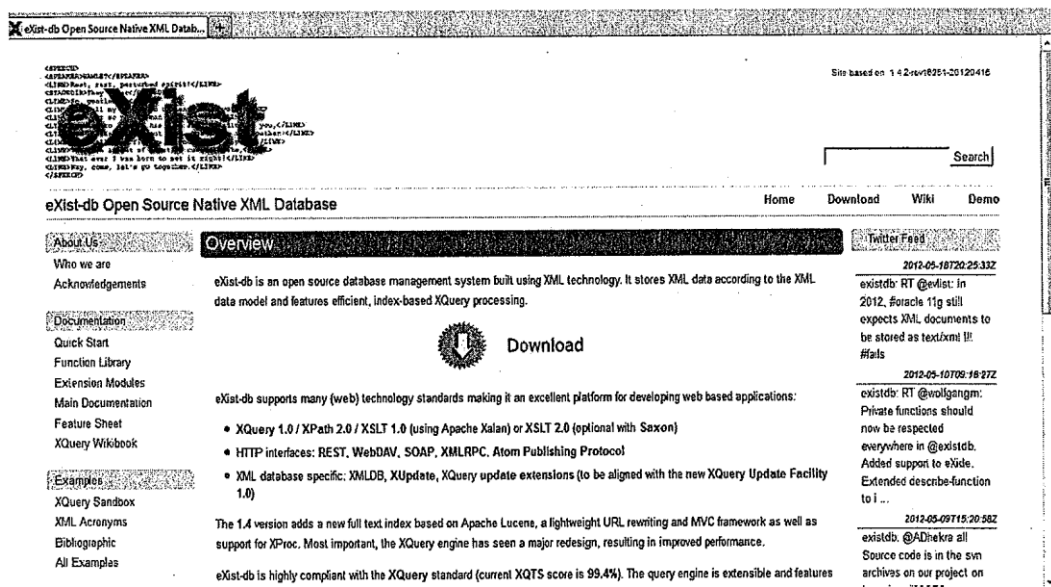
En el proceso de instalación, se nos pide introducir la ruta donde está instalado Java JDK (mínimo versión 5, C:\Program Files\Java\jdk1.6.0\_29). También indicaremos donde se desea que se instale la BD, nos pedirá contraseña para el usuario administrador de la BD *admin* y marcaremos que instale todas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



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. El 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 se muestra la pantalla inicial de la BD eXist.

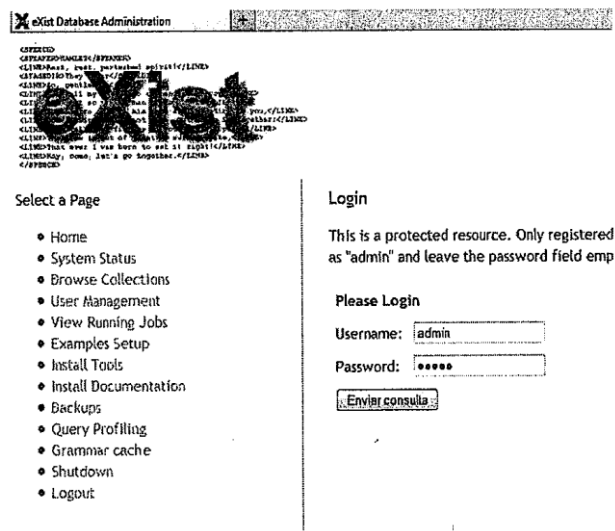


También es importante tener en cuenta que es posible que el puerto de conexión de la base de datos (el 8080) se encuentre bloqueado por alguna aplicación (por ejemplo, por el Firewall de Windows). Hay que asegurar que el puerto está abierto o habilitado.

2.2 PRIMEROS PASOS CON EXIST

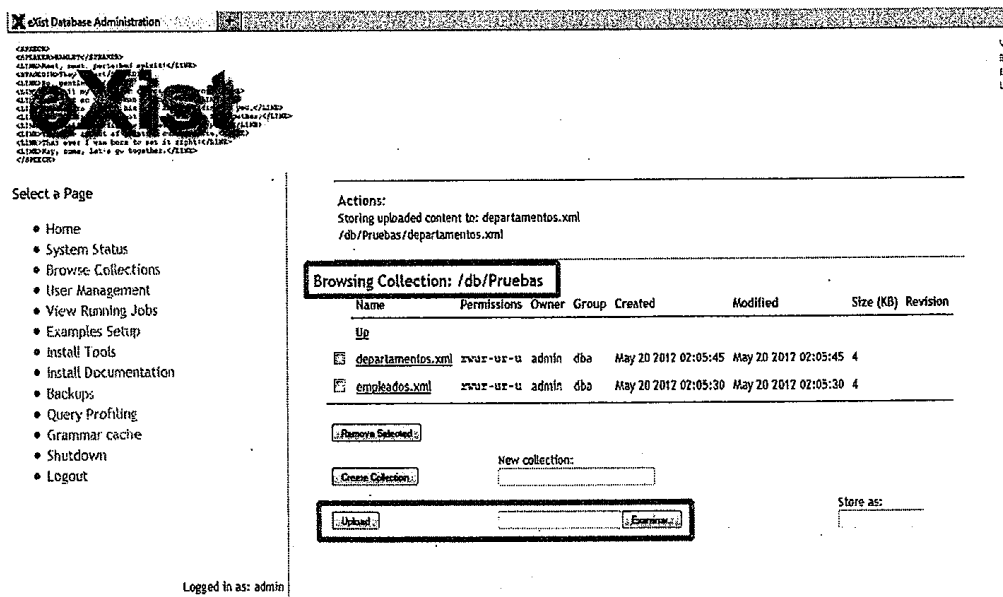
Lo primero que vamos a hacer es cargar un fichero XML que nos va a servir como base de datos para realizar consultas utilizando XQuery y XPath, así pues será necesario crear una colección y luego almacenar el documento XML en la colección.

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.

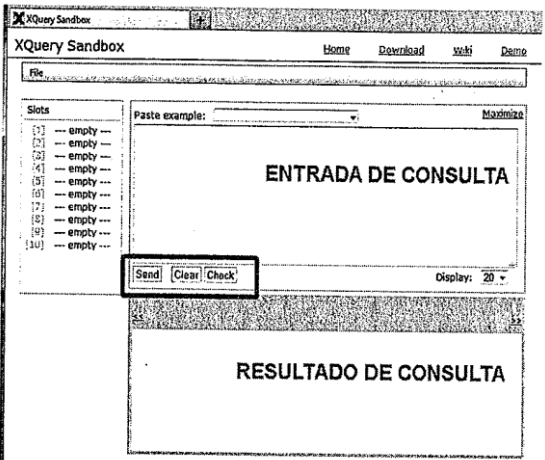


Desde la ventana de administración podremos crear usuarios si seleccionamos **User Management**, se pueden instalar los ejemplos que vienen con la BD para hacer pruebas de consultas desde **Examples Setup**, o se pueden realizar copias de seguridad desde la opción **Backups**.

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** y 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 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.



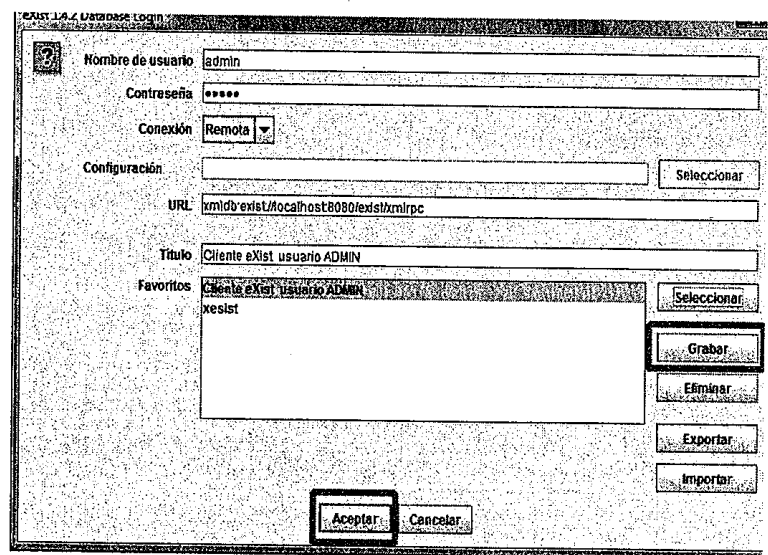
Una vez creada la colección hay que abrir el gestor de consultas de eXist, para ello regresamos a la ventana inicial pulsando el enlace **Home**, y desde ahí se selecciona **XQuery Sandbox** que aparece en el menú de la izquierda dentro de **Examples**. Desde el gestor de consultas se podrán realizar consultas a los documentos XML de nuestra colección, en la 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.



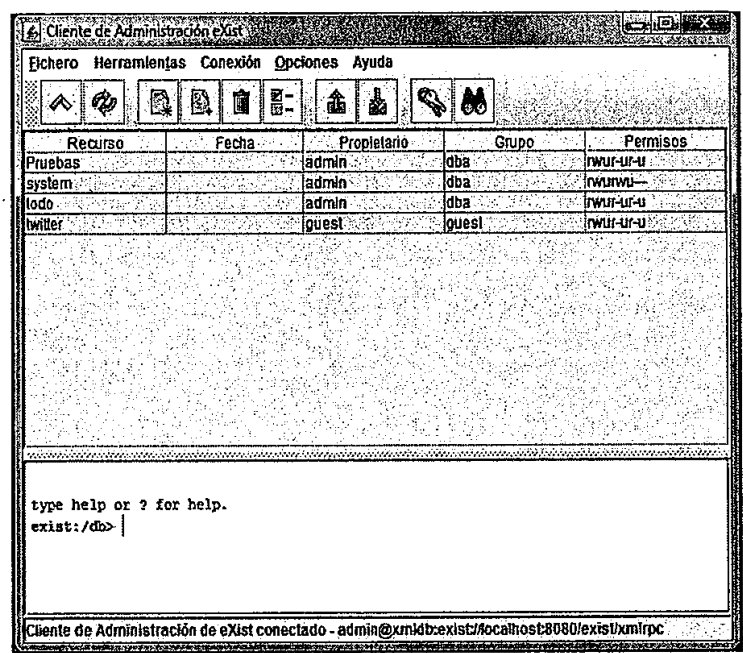
### 2.3 EL CLIENTE DE ADMINISTRACIÓN DE EXIST


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.

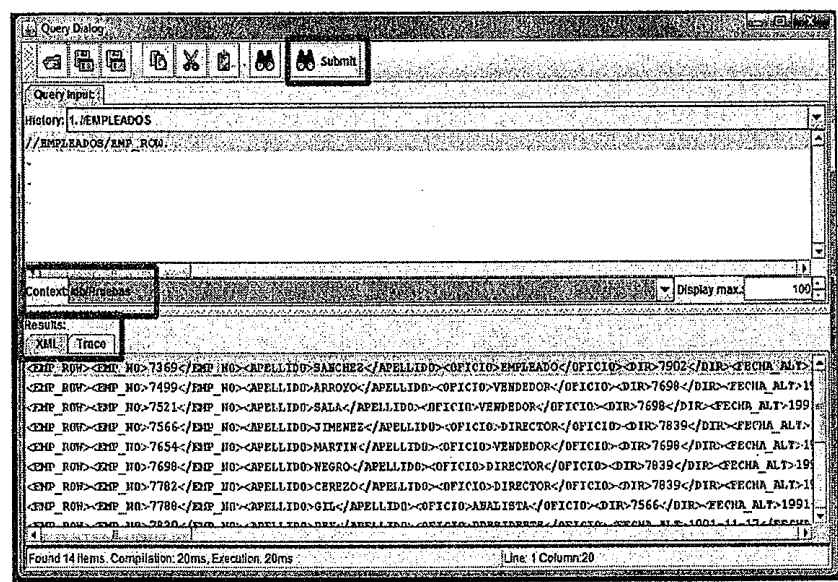
A continuación se muestra la ventana donde pide que se configure la conexión o se elija una ya creada. Teclearemos el nombre de usuario, su contraseña, la URL de la BD y un título para la conexión, estos datos se quedarán guardados para posteriores conexiones.



A continuación 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 y restaurarlas, administrar usuarios y realizar consultas XPath, entre otras operaciones.



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 se muestra la ventana de *Query Dialog*.



### 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. XQuery contiene a XPath, toda expresión de consulta en XPath es válida en XQuery, pero XQuery permite mucho más.

#### 3.1 EXPRESIONES XPATH

XPath es un lenguaje que permite seleccionar nodos de un documento XML y calcular valores a partir de su contenido.



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

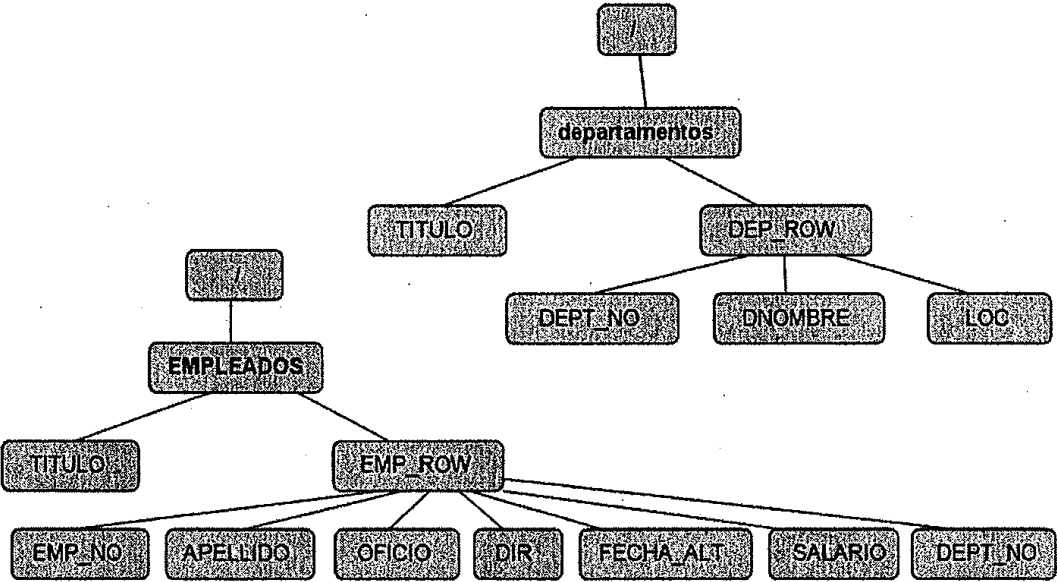
En este tema empezaremos con la sintaxis abreviada, así pues los descriptores se forman simplemente nombrando la etiqueta separada por /.

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.

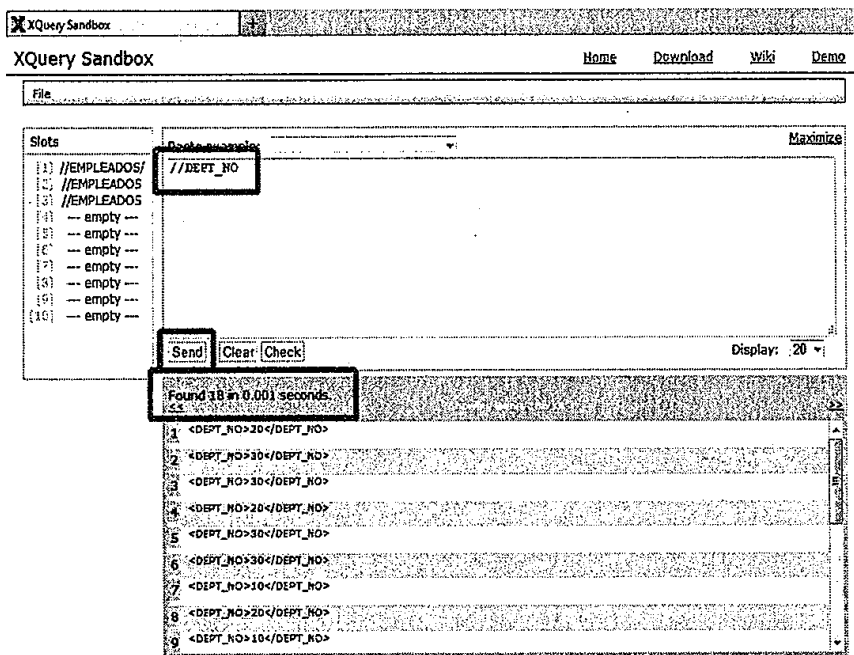
Si el descriptor comienza con // se supone que la ruta descrita puede comenzar en cualquier parte de la colección.

- Ejemplos XPath utilizando una sintaxis abreviada:

A partir de la colección *Pruebas*, que contiene los documentos *departamentos.xml* y *empleados.xml*, cuyas estructuras se muestran en la Figura. Realizar desde **Query Dialog** o **XQuery Sandbox** las siguientes consultas:



- `/`, 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.



- **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**. 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(OFCIO = 'ANALISTA')]/APELLIDO/text()`, selecciona los `APELLIDOS` de los empleados que NO son analistas.
  - `/EMPLEADOS /EMP_ROW[DEPT_NO=10]/APELLID | EMPLEADOS/EMP_ROW[DEPT_NO=10]/OFCIO`, selecciona el `APELLIDO` y el `OFCIO` 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.  
`/EMPLEADOS/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 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.

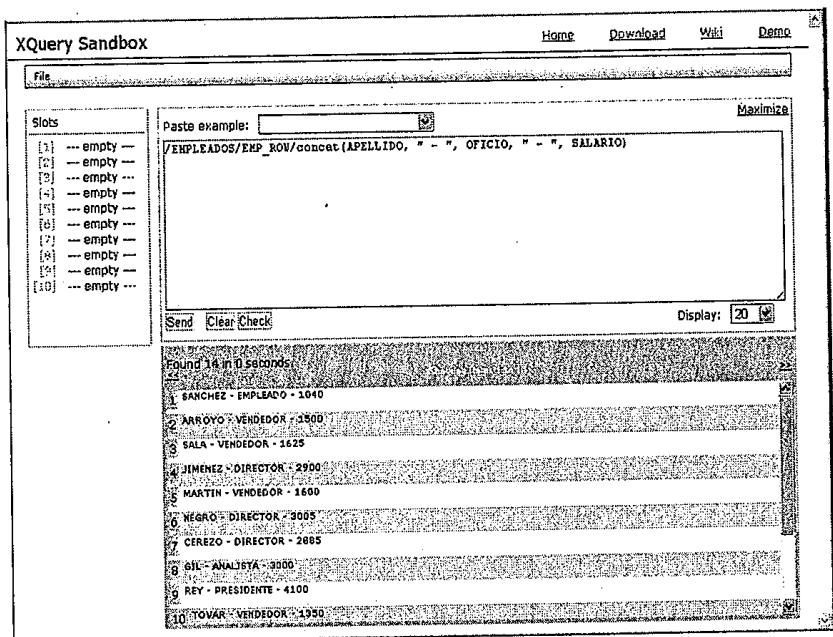
- La función **concat(cad1, cad2,...)** concatena las cadenas.

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.



- La función **starts-with(cad1, cad2)** es verdadera cuando la cadena cad1 tiene como prefijo a la cad2. Ejemplo:

/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 verdadero 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.

**Puedes encontrar más funciones en la URL:**  
[http://www.w3schools.com/xpath/xpath\\_functions.asp](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.



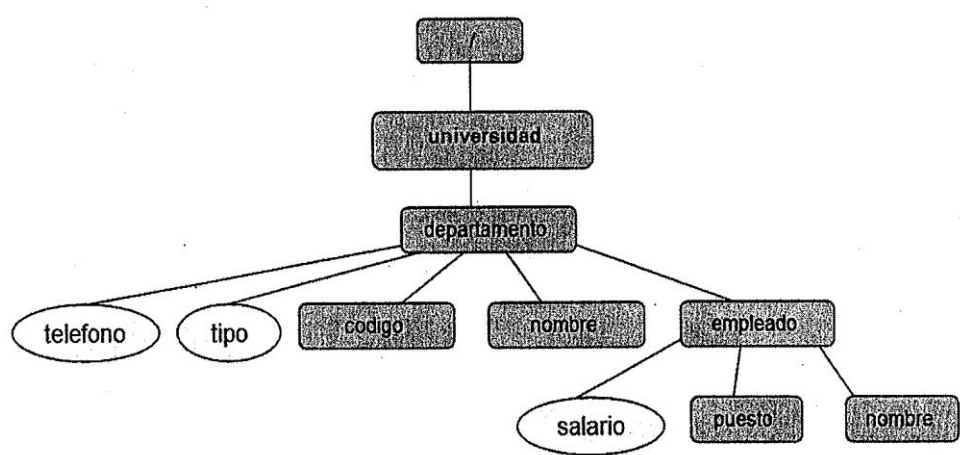
3.2 NODOS ATRIBUTOS XPATH

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.

<pre>&lt;universidad&gt;   &lt;departamento telefono="112233"   tipo="A"&gt;     &lt;codigo&gt;IFC1&lt;/codigo&gt;     &lt;nombre&gt;Informática&lt;/nombre&gt;     &lt;empleado salario="2000"&gt;       &lt;puesto&gt;Asociado&lt;/puesto&gt;       &lt;nombre&gt;Juan Parra&lt;/nombre&gt;     &lt;/empleado&gt;     &lt;empleado salario="2300"&gt;       &lt;puesto&gt;Profesor&lt;/puesto&gt;       &lt;nombre&gt;Alicia Martín&lt;/nombre&gt;     &lt;/empleado&gt;   &lt;/departamento&gt;    departamento telefono="990033"   tipo="A"&gt;     &lt;codigo&gt;MAT1&lt;/codigo&gt;     &lt;nombre&gt;Matemáticas&lt;/nombre&gt;     &lt;empleado salario="1900"&gt;       &lt;puesto&gt;Técnico&lt;/puesto&gt;       &lt;nombre&gt;Ana García&lt;/nombre&gt;     &lt;/empleado&gt;     &lt;empleado salario="2100"&gt;       &lt;puesto&gt;Profesor&lt;/puesto&gt;       &lt;nombre&gt;Mª Jesús Ramos&lt;/nombre&gt;     &lt;/empleado&gt;   &lt;/departamento&gt; &lt;/universidad&gt;</pre>	<pre>&lt;empleado salario="2300"&gt;   &lt;puesto&gt;Profesor&lt;/puesto&gt;   &lt;nombre&gt;Pedro Paniagua&lt;/nombre&gt; &lt;/empleado&gt; &lt;empleado salario="2500"&gt;   &lt;puesto&gt;Tutor&lt;/puesto&gt;   &lt;nombre&gt;Antonia González&lt;/nombre&gt; &lt;/empleado&gt; &lt;/departamento&gt;  &lt;departamento telefono="880833" tipo="B"&gt;   &lt;codigo&gt;MAT2&lt;/codigo&gt;   &lt;nombre&gt;Análisis&lt;/nombre&gt;   &lt;empleado salario="1900"&gt;     &lt;puesto&gt;Asociado&lt;/puesto&gt;     &lt;nombre&gt;Laura Ruiz&lt;/nombre&gt;   &lt;/empleado&gt;   &lt;empleado salario="2200"&gt;     &lt;puesto&gt;Asociado&lt;/puesto&gt;     &lt;nombre&gt;Mario García&lt;/nombre&gt;   &lt;/empleado&gt; &lt;/departamento&gt;  &lt;/universidad &gt;</pre>
--	---

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.



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,' Minimo=', 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="xxxxxxx" codigo="xxxx">
    <director>xxxxxxxxxxxxxxxx</director>
    <poblacion>xxxxxxxxxx</poblacion>
    <cuenta tipo="xxxxxxx">
      <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 cuyos tipo 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.
- Obtén los nodos número de cuenta, nombre de cuenta y el saldo haber de las cuentas con saldo haber mayor de 10000.
- Obtén por cada sucursal con más de 3 cuentas del tipo AHORRO, su código y la suma del saldo debe de esas cuentas.

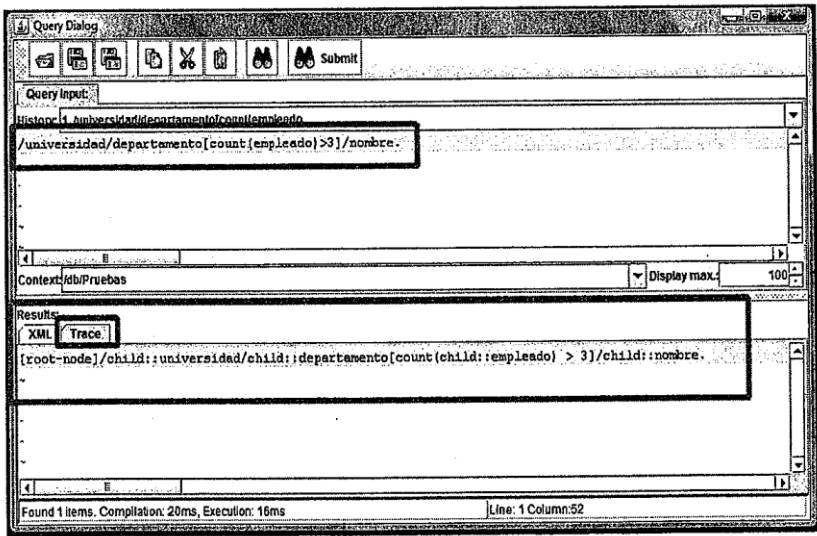
3.3 AXIS XPATH

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

Lasintaxis para utilizar ejes es la siguiente: Nombre\_de\_eje::nombre\_nodo [expresión]

En la ventana del **Query Dialog** del *Cliente de Administración de eXist*, se puede ver la traza de las consultas y en la traza podemos observar los ejes utilizados. Véase la Figura.



Ejemplos:

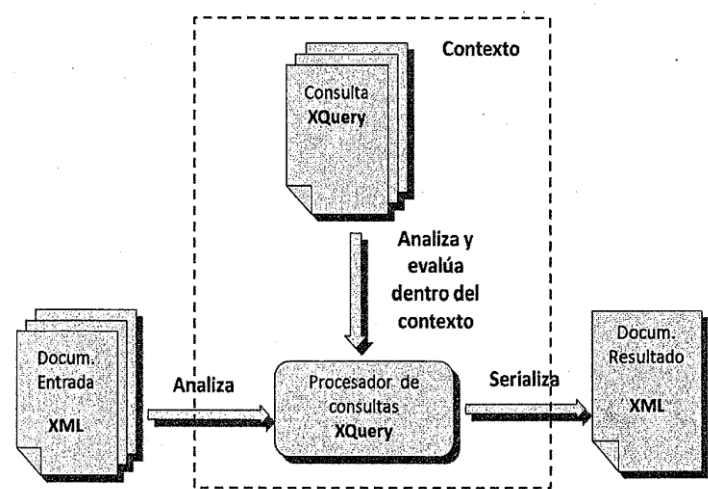
- /universidad/child::\*, es lo mismo que /child::universidad/ child::element () . Devuelve todos los hijos de universidad, es decir los nodos de los departamentos.
- /universidad/departamento/descendant: :\*, devuelve los descendientes del nodo departamento, esto hace lo mismo: /child::universidad/child::departamento/descendant::element()
- /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.
  - //empleado/following-sibling:: node (), selecciona todos los hermanos de los elementos empleado que encuentre en el contexto.  
En este caso //empleado/following-sibling: :empleado [@salario>2100], selecciona todos los hermanos de los elementos empleado que tienen el salario >2100.
  - //empleado [nombre="Ana García"]/following-sibling:: \*, selecciona los nodos hermanos de Ana García.
  - //empleado[nombre="Ana Garcia"]/following-sibling::empleado/nombre/text(), selecciona los nombres de los empleados hermanos de Ana García.
  - //empleado[nombre="Ana Garcia"]/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 Garcia"]/parent::departamento/nombre, selecciona el nombre del padre de la empleada Ana García.
  - /descendant::departamento [3], selecciona los descendientes del departamento que ocupa la posición 3 en el documento.
  - /child::universidad/child::departamento[count(child::empleado)>3], es lo mismo que /universidad/departamento [count (empleado) >3], obtiene los departamentos con más de 3 empleados.
  - /child: :universidad/child: :departamento/child: :nombre, obtiene las etiquetas con los nombres de los departamentos. Es lo mismo que /universidad/departamento/nombre, y que data(/universidad/departamento/nombre).
  - /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().

### 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 se ve el procesamiento básico de una consulta XQUERY. XQuery contiene a XPath. 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,



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

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

- **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 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:

<pre>for \$emp in /EMPLEADOS/EMP_ROW order by \$emp/APELLIDO return if (\$emp/OFICIO='DIRECTOR') then &lt;DIRECTOR&gt;{\$emp/APELLIDO/text()} &lt;/DIRECTOR&gt; else &lt;EMPLE&gt;{data(\$emp/APELLIDO)}&lt;/EMPLE&gt;</pre>	<p><b>Devuelve los nombres de los empleados, los que son directores entre las etiquetas &lt;DIRECTOR&gt; &lt;/DIRECTOR&gt; y los que no lo son entre las etiquetas &lt;EMPLE&gt;&lt;/EMPLE&gt;.</b></p> <p>&lt;EMPLE&gt;ALONSO&lt;/EMPLE&gt; &lt;EMPLE&gt;ARROYO&lt;/EMPLE&gt; &lt;DIRECTOR&gt;CEREZO&lt;/DIRECTOR&gt; &lt;EMPLE&gt;FERNANDEZ&lt;/EMPLE&gt; &lt;EMPLE&gt;GIL&lt;/EMPLE&gt; &lt;DIRECTOR&gt;JIMENEZ&lt;/DIRECTOR&gt;</p>
<pre>for \$de in doc ('file:///D:/XML /pruebaxquery/NUEVOS_DEP.xml')/NUEVOS DEP/ DEP_ROW return \$de</pre>	<p><b>Devuelve los nodos DEP_ROW de un documento ubicado en una carpeta del disco duro.</b></p>
<pre>for \$prof in /universidad/departamento[@tipo='A']/empleado let \$profe:= \$prof/nombre, \$puesto:= \$prof/puesto where \$puesto='Profesor' return \$profe</pre>	<p><b>Obtiene los nombres de empleados de los departamentos de tipo A, cuyo puesto es Profesor. Esto hace lo mismo:</b></p> <p>for \$prof in /universidad/departamento [@tipo='A']/empleado where \$prof/puesto='Profesor' return \$prof/nombre</p> <p><b>El resultado es:</b></p> <p>&lt;nombre&gt;Alicia Martin&lt;/nombre&gt; &lt;nombre&gt;M<sup>a</sup> Jesús Ramos&lt;/nombre&gt; &lt;nombre&gt;Pedro Paniagua&lt;/nombre&gt;</p>
<pre>for \$dep in /universidad/departamento return if (\$dep/@tipo='A') then &lt;tipoA&gt;{data(\$dep/nombre)}&lt;/tipoA&gt; else &lt;tipoB&gt;{data(\$dep/nombre)}&lt;/tipoB&gt;</pre>	<p><b>Devuelve el nombre de departamento encerrado entre las etiquetas &lt;tipoA&gt;&lt;/tipoA&gt;, si es del tipo = A, y &lt;tipoB&gt;&lt;/tipoB&gt;, si no lo es.</b></p> <p>&lt;tipoA&gt;Informática&lt;/tipoA&gt; &lt;tipoA&gt;Matemáticas&lt;/tipoA&gt; &lt;tipoB&gt;Análisis&lt;/tipoB&gt;</p>
<pre>for \$dep in /universidad/departamento let \$nom:= \$dep/empleado return &lt;depart&gt;{data(\$dep/nombre)} &lt;emple&gt;{count(\$nom)}&lt;/emple&gt;&lt;/depart&gt;</pre>	<p><b>Obtiene los nombres de departamento y los empleados que tiene entre etiquetas:</b></p> <p>&lt;depart&gt;Informática&lt;emple&gt;2&lt;/emple&gt;&lt;/depart&gt; &lt;depart&gt;Matemáticas&lt;emple&gt;4&lt;/emple&gt;&lt;/depart&gt; &lt;depart&gt;Análisis&lt;emple&gt;2&lt;/emple&gt;&lt;/depart&gt;</p>
<pre>for \$dep in /universidad/departamento let \$emp:= \$dep/empleado let \$sal:= \$dep/empleado/@salario return &lt;depart&gt;{data(\$dep/nombre)} &lt;emple&gt;{count(\$emp)}&lt;/emple&gt;&lt;medsal&gt; {avg(\$sal) }&lt;/medsal&gt;&lt;/depart&gt;</pre>	<p><b>Obtiene los nombres de departamento, los empleados que tiene y la media del salario entre etiquetas:</b></p> <p>&lt;depart&gt;Informática&lt;emple&gt;2&lt;/emple&gt; &lt;medsal&gt;2150&lt;/medsal&gt;&lt;/depart&gt; &lt;depart&gt;Matemáticas&lt;emple&gt;4&lt;/emple&gt; &lt;medsal&gt;2200&lt;/medsal&gt;&lt;/depart&gt; &lt;depart&gt;Análisis&lt;emple&gt;2&lt;/emple&gt; &lt;medsal&gt;2050&lt;/medsal&gt;&lt;/depart&gt;</p>



### 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. 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: unión (|), intersect, except.
- Redondeo: floor(), ceiling(), round().
- Funciones de agrupación: count(), min(), max(), avg(), sum().
- Funciones de cadena: concat(), string-length(), starts-with(), ends-with(), substring(), upper-case(), lower-case(), string().
- Uso general: **distinct-values()** extrae los valores de una secuencia de nodos y crea una nueva secuencia con valores únicos, eliminando los nodos duplicados, **empty()** devuelve cierto cuando la expresión entre paréntesis está vacía. 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.

### 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.

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>
```

**SALIDA:**

```
<res>
  <APELLIDO>SANCHEZ</APELLIDO>
  <DEPT_NO>20</DEPT_NO>
  <DNOMBRE>INVESTIGACION</DNOMBRE>
</res>
<res>
  <APELLIDO>ARROYO</APELLIDO>
  <DEPT_NO>30</DEPT_NO>
  <DNOMBRE>VENTAS</DNOMBRE>
</res>
<res>
  <APELLIDO>SALA</APELLIDO>
  <DEPT_NO>30</DEPT_NO>
  <DNOMBRE>VENTAS</DNOMBRE>
</res>
```

- Utilizando los documentos *departamentos.xml* y *empleados.xml*, obtener por cada departamento, el nombre de departamento, el número de empleados, y la media de salario.

```
for $dep in /departamentos/DEP_ROW
let $d:=$dep/DEPT_NO
let $tot:=sum(/EMPLEADOS/EMP_ROW[DEPT_NO=$d]/SALARIO)
let $cu:=count(/EMPLEADOS/EMP_ROW[DEPT_NO=$d]/EMP_NO)
return <resul>{$dep/DNOMBRE}<sumsalario>{$tot}</sumsalario><totemple>{$cu}</totemple></resul>
```

**SALIDA:**

```
<resul>
  <DNOMBRE>CONTABILIDAD</DNOMBRE>
  <sumasalario>8675</sumasalario>
  <numemple>3</numemple>
</resul>
<resul>
  <DNOMBRE>INVESTIGACION</DNOMBRE>
  <sumasalario>11370</sumasalario>
  <numemple>5</numemple>
</resul>
```

```

<resul>
  < DNOMBRE >VENTAS </DNOMBRE >
  <sumasalario>10415</sumasalario>
  <numemple> 6 </numemple >
</resul>

```

```

<resul>
<DNOMBRE>PRODUCCION</DNOMBRE>
<sumasalario>0</sumasalario>
<numemple>0</numemple>
</resul>

```

- Convertir la salida de la consulta anterior, de manera que el total salario, y el total empleados, sean atributos de cada departamento. Hacemos que la salida que se cree sea una concatenación de los datos a obtener:

```

for $dep in /departamentos/DEP_ROW
let $d:=$dep/DEPT_NO
let $tot:=sum(/EMPLEADOS/EMP_ROW[DEPT_NO=$d]/SALARIO)
let $cu:=count(/EMPLEADOS/EMP_ROW[DEPT_NO=$d]/EMP_NO)
return concat('<departamento sumasalario="', $tot, ' " totemple=" ', $cu, ' ">',
  data($dep/DNOMBRE), '</departamento>')

```

#### **SALIDA:**

```

<departamento sumasalario="8675" totemple="3">CONTABILIDAD</departamento>
<departamento sumasalario="11370" totemple="5">INVESTIGACION</departamento>
<departamento sumasalario="10415" totemple="6">VENTAS</departamento>
<departamento sumasalario="0" totemple="0">PRODUCCION</departamento>

```

- Utilizando los documentos *departamentos.xml* y *empleados.xml*, obtener por cada departamento, el nombre de empleado que más gana.

```

for $emp in /EMPLEADOS/EMP_ROW
let $d:=$emp/DEPT_NO, $nom:=$emp/APELLIDO, $sal:=$emp/SALARIO
let $ndep:=(/departamentos/DEP_ROW[DEPT_NO=$d]/DNOMBRE)
let $salmax:= max(/EMPLEADOS/EMP_ROW[DEPT_NO=$d]/SALARIO)
return
if ($sal=$salmax)
then
<depart>{data($ndep)}<salmax>{data($sal)}</salmax><emple>{data($nom)}</emple></depart>
else ()

```

#### **SALIDA:**

```

<depart>VENTAS <salmax>3005</salmax><emple>NEGRO</emple></depart>
<depart>INVESTIGACION<salmax>3000</salmax><emple>GIL</emple></depart>
<depart>CONTABILIDAD<salmax>4100</salmax><emple>REY</emple></depart>
<depart>INVESTIGACION<salmax>3000</salmax><emple>FERNANDEZ</emple></depart>

```

**Actividad 4:** Sube a la colección *Pruebas* el documento *zonas.xml*, contiene información de las zonas donde se venden los productos del documento *productos.xml*. Utilizando estos dos documentos realiza las siguientes consultas XQuery:

- Obtén los datos denominación, precio y nombre de zona de cada producto, ordenado por nombre de zona.
- Obtén por cada zona, el nombre de zona y el número de productos que tiene.
- Obtén por cada zona, el nombre de la zona, su código y el nombre del producto con menos stock actual.

---

• **Utilización de varios for.** La utilización de varios for es muy útil para consultas en documentos XML anidados y también cuando utilizamos varios documentos unidos por una cláusula **where** como una combinación de tablas en SQL.

Ejemplos:

- Esta consulta visualiza por cada departamento del documento *universidad.xml*, el número de empleados que hay en cada puesto de trabajo. Utilizamos un for para obtener los nodos departamento y el segundo for para obtener los distintos puestos de cada departamento.

```
for $dep in /universidad/departamento
for $pue in distinct-values($dep/empleado/puesto)
let $cu:=count($dep/empleado[puesto=$pue])
return <depart>{data($dep/nombre)}<puesto>{data($pue)}</puesto><profes>{$cu}</profes></depart>
```

**SALIDA:**

```
<depart>Informática<puesto>Asociado</puesto><profes>1</profes></depart>
<depart>Informática<puesto>Profesor</puesto><profes>1</profes></depart>
<depart>Matemáticas<puesto>Técnico</puesto><profes>1</profes></depart>
<depart>Matemáticas<puesto>Profesor</puesto><profes>2</profes></depart>
<depart>Matemáticas<puesto>Tutor</puesto><profes>1</profes></depart>
<depart>Análisis<puesto>Asociado</puesto><profes>2</profes></depart>
```

- Esta consulta visualiza por cada departamento del documento *universidad.xml*, el salario máximo y el empleado que tiene ese salario. El primer for obtiene los nodos departamento y el segundo for los empleados de cada departamento. Para sacar el máximo en la salida preguntamos si el salario es el máximo.

```
for $dep in /universidad/departamento
for $emp in $dep/empleado
let $emple:= $emp/nombre
let $sal:= $emp/@salario
return if ($sal = $dep/max(empleado/@salario))
then
    <depart>{data($dep/nombre)} <salamax>{data($sal)}</salamax>
    <empleado>{data($emple)}</empleado></depart>
else ()
```

**También se pueden poner los dos for de la siguiente manera:**

```
for $dep in /universidad/departamento, $emp in $dep/empleado
```

**SALIDA:**

```

<depart>Informática
  <salamax>2300</salamax>
  <empleado>Alicia Martin</empleado>
</depart>

<depart>Matemáticas
  <salamax>2500</salamax>
  <empleado>Antonia González</empleado>

</depart> <depart>Análisis
  <salamax>2200</salamax>
  <empleado>Mario Garcia</empleado>
</depart>

```

- Esta consulta visualiza por cada puesto del documento *universidad.xml*, el empleado con salario máximo y ese salario. El primer for obtiene los distintos puestos de trabajo y el segundo for obtiene los empleados que tienen ese puesto de trabajo. En el if se pregunta si el salario del empleado es el salario máximo de los empleados del oficio del primer for.

```

for $pue in distinct-values (/universidad/departamento/empleado/puesto)
for $emp in /universidad/departamento/empleado[puesto=$pue]
let $sal:= $emp/@salario
let $nom:= $emp/nombre
return
if ($sal = max(/universidad/departamento/empleado[puesto=$pue]/$salario))
then
  <puesto>{data($pue)}<maxsalario>{data($sal)}</maxsalario><emple>{data ($nom)}</emple>
  </puesto>
else ()

```

**SALIDA:**

```

<puesto>Asociado<maxsalario>2200</maxsalario><emple>Mario García</emple></puesto>
<puesto>Profesor<maxsalario>2300</maxsalario><emple>Alicia Martin</emple></puesto>
<puesto>Profesor<maxsalario>2300</maxsalario><emple>Pedro Paniagua</emple></puesto>
<puesto>Técnico<maxsalario>1900</maxsalario><emple>Ana Garcia</emple></puesto>
<puesto>Tutor<maxsalario>2500</maxsalario><emple>Antonia González</emple></puesto>

```

**También podemos resolver la consulta utilizando un solo for, de la siguiente manera:**

```

for $emp in (/universidad/departamento/empleado)
let $pue:=$emp/puesto
let $sal:= $emp/@salario
let $nom:= $emp/nombre
order by $pue
return
if ($sal = max(/universidad/departamento/empleado[puesto=$pue]/@salario))
then
  <puesto>{data($pue)}<maxsalario>{data($sal)}</maxsalario><emple>{data($nom)}</emple> </puesto>
else ()

```

La primera consulta del apartado anterior la podemos escribir con dos for y el where:

```

for $emp in (/EMPLEADOS/EMP_ROW), $dep in /departamentos/DEP_ROW
let $emple:= $emp/APELLIDO
let $d:= $emp/DEPT_NO
where data($d) = data($dep/DEPT_NO)
return <res>{$emple, $d} {$dep/DNOMBRE} </res>

```

**Actividad 5:** Utiliza el documento *sucursales.xml* para realizar las siguientes consultas XQuery:

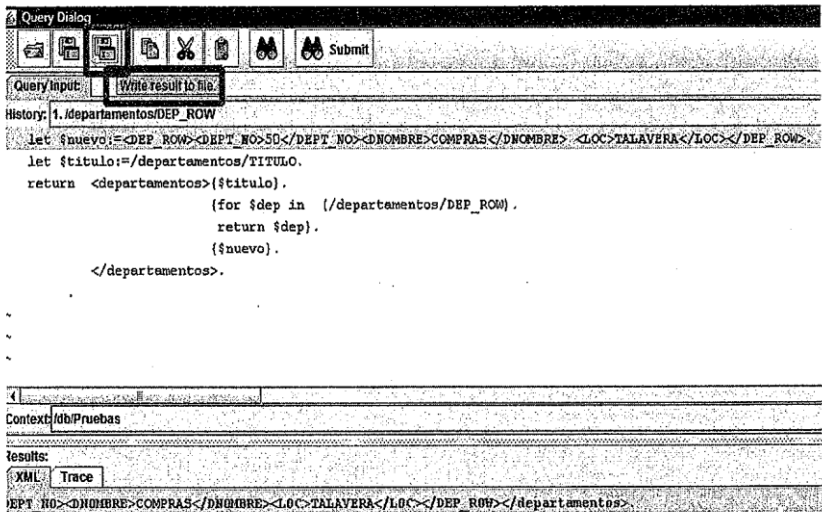
- Obtén por cada sucursal el mayor saldo haber y el nombre de la cuenta que tiene ese saldo.
- Obtén por cada sucursal el nombre de la cuenta del tipo AHORRO cuyo saldo debe sea el máximo. Saca también el máximo.

Utiliza los documentos *productos.xml* y *zonas.xml*

- Visualiza los nombres de productos con su nombre de zona. Utiliza dos for en la consulta.
- Visualiza los nombres de productos con stock\_minimo > 5. su código de zona, su nombre y el director de esa zona. Utiliza dos for en la consulta.

• **Altas, bajas y borrado de nodos en documentos XML.**

- Esta consulta va a generar una salida en la que se van a incluir todos los nodos del documento *departamentos.xml*, más un nuevo nodo a insertar. En la variable *\$nuevo* añadimos el nodo a insertar, en *\$titulo* guardamos el nodo TITULO del documento *departamentos.xml*. En el return creamos una etiqueta llamada *<departamentos>* e incluimos el título, todos los nodos *DEP\_ROW* de *departamentos.xml* (estos se obtienen con el for) y el nodo nuevo. Si se desea guardar la salida generada se pulsa el botón **Write result to file** del **Query Dialog** de eXist, véase la Figura.



- En la siguiente consulta se elimina el nodo cuyo número de departamento es el 10. Utilizamos el mismo documento. Se trata de generar una consulta que obtenga todos los nodos del documento menos el nodo con ese número de departamento. Utilizamos *where* para no seleccionar el departamento 10.

```
let $titulo:=/departamentos/TITULO
return <departamentos>{$titulo}
      {for $dep in (/departamentos/DEP_ROW)
        where data(($dep/DEPT_NO))!=10
        return $dep
      }
</departamentos>
```

- En la siguiente consulta se va a actualizar el departamento cuyo número de departamento es el 20. Vamos a cambiar la localidad (LOC) por GRANADA. Lo que hacemos es sacar los datos del nodo a modificar, para luego insertar ese nodo modificado, en nuestro caso sacamos el nombre del departamento. Y luego se crea el nodo modificado en *\$modif* con todas las etiquetas; de manera que cuando se van generando las etiquetas con los departamentos al llegar al departamento a modificar (20) se devuelve el nodo *\$modif*, y si no es el 20, se visualiza el nodo *\$dep*.

```
let $titulo:=/departamentos/TITULO
let $dnom:=/departamentos/DEP_ROW[DEPT_NO=20]/DNOMBRE/text()
let $modif:=<DEP_ROWXDEPT_NO>20</DEPT_NOXDNOMBRE>{$dnom}</DNOMBRE><LOC>GRANADA
    </LOC> </DEP_ROW>
return <departamentos>{$titulo}
    {for $dep in (/departamentos/DEP_ROW)
      return
        if ($dep/DEPT_NO/text()=20) then $modif
        else $dep
    }
</departamentos>
```

- En la siguiente consulta se van a actualizar los salarios de los empleados del departamento 10, se les sube a todos la cantidad de 200. Lo que se hace para todos los empleados es sacar los nodos a variables, de manera que si el empleado es del departamento 10, se creará un nodo nuevo <EMP\_ROW> con los datos de sus nodos y el salario actualizado. Si el empleado no es del departamento 10 se devuelve el nodo EMP\_ROW leído.

```
for $em in /EMPLEADOS/EMP_ROW
let $no:=$em/EMP_NO, $ape:=$em/APELLIDO, $ofi:=$em/OFICIO, $dir:=$em/DIR
let $fec:=$em/FECHA_ALT, $de:=$em/DEPT_NO, $sal:=$em/SALARIO
let $salnue:=200+ number($em/SALARIO/text ())
return
  if ($em/DEPT_NO/text()=10) then
    <EMP_ROW>{$no, $ape, $ofi, $dir, $fec}<SALARIO>{$salnue}</SALARIO>{$de}</EMP_ROW>
  else $em
```

- En la siguiente consulta se han realizado los cambios necesarios para que la salida sea el documento *empleados.xml* pero con el salario actualizado.

```
let $titulo:=/EMPLEADOS/TITULO
return <EMPLEADOS>
  {$titulo}
  {
    for $em in /EMPLEADOS/EMP_ROW
    let $no:=$em/EMP_NO, $ape:=$em/APELLIDO, $ofi:=$em/OFICIO, $dir:=$em/DIR
    let $fec:=$em/FECHA_ALT, $de:=$em/DEPT_NO, $sal:=$em/SALARIO
    let $salnue:=200+ number($em/SALARIO/text())
    return
      if ($em/DEPT_NO/text()=10) then
        <EMP_ROW>{$no,$ape,$ofi,$dir,$fec}<SALARIO>{$salnue}</SALARIO>{$de}
        </EMP_ROW>
      else $em
    }
  }
</EMPLEADOS>
```



**Actividad 6:** Crea un nuevo documento XML llamado *productos\_nuevo.xml*, que incorpore el siguiente producto: cod\_prod: 1023, denominación: HD externo Seagate expansión 250GB, precio: 70, stock\_actual: 100, stock\_minimo: 20, y el código de zona tiene que ser el código de zona correspondiente a Andalucía.

Crea un nuevo documento XML llamado *productos\_30.xml*, con el <TITULO>DATOS DE LOS PRODUCTOS DE LA ZONA 30</TITULO>, este documento solo debe contener los productos cuyo código de zona sea 30.

Crea una consulta que visualice todo el documento de *productos.xml* pero con los siguientes cambios:

- Sube el precio de cada producto a un 3% más caro.
- Añade 10 unidades a los stock\_actual de todos los productos.

-----

• **Sentencias de actualización de eXist.**

Estas sentencias permiten hacer altas, bajas y modificaciones de nodos y elementos en documentos xml. Se pueden usar las sentencias de actualización en cualquier punto pero si se utiliza en la cláusula RETURN de una sentencia FLWOR, el efecto de la actualización es inmediato. Todas las sentencias de actualización comienzan con la palabra UPDATE y a continuación la instrucción. Son las siguientes:

- **Insert**, se utiliza para insertar nodos. El lugar de inserción se especifica con: **into** (el contenido se añade como último hijo de los nodos especificados); **following** (el contenido se añade inmediatamente después de los nodos especificados), o **preceding** (el contenido se añade antes de los nodos especificados). El formato es:

**update insert ELEMENTO into EXPRESION**  
**update insert ELEMENTO following EXPRESION**  
**update insert ELEMENTO preceding EXPRESION**

<pre>update insert &lt;zona&gt;&lt;cod_zona&gt;50&lt;/cod_zona&gt; &lt;nombre&gt;Madrid-OESTE &lt;/nombre&gt; &lt;director&gt;Alicia Ramos Martín&lt;/director&gt; &lt;/zona&gt; into /zonas</pre>	Inserta una zona en <i>zonas.xml</i> , en la última posición.
<pre>update insert &lt;cuenta tipo="PENSIONES"&gt;&lt;nombre&gt;Alberto Morales &lt;/nombre&gt;&lt;numero&gt;30302900&lt;/numero&gt; &lt;aportacion&gt;5000&lt;/aportacion&gt;&lt;/cuenta&gt; into /sucursales/sucursal[@codigo="SUC1"]</pre>	Inserta una cuenta en el documento <i>sucursales.xml</i> del tipo PENSIONES a la sucursal SUC1.
<pre>for \$de in doc('file:///D:/XML/pruebaxquery/NUE- VOS_DEP.xml') /NUEVOS_DEP/DEP_ROW return update insert \$de into /departamentos</pre>	Inserta en el documento departamentos de la BD los nodos DEP_ROW del documento externo NUEVOS_DEP.xml ubicado en la carpeta D:/XML/pruebaxquery/

- **Replace**, sustituye el nodo especificado en NODO con VALOR NUEVO (ver formato). NODO debe devolver un único ítem: si es un elemento, VALOR\_NUEVO debe ser también un elemento. Si es un nodo de texto o atributo su valor será actualizado con la concatenación de todos los valores de VALOR\_NUEVO.

**update replace NODO with VALOR\_NUEVO**

update replace /zonas/zona[cod zona=50]/director with <directora>Pilar Martin Ramos</directora>	Cambia la etiqueta director de la zona 50 y su contenido, en el documento <i>zonas.xml</i>
update replace /departamentos/DEP_ROW[DEPT_NO=10] with <DEPT_ROW><DEPT_NO>10</DEPT_NO><DNOMBRE>NUEVO10</DNOMBRE> <LOC>TALAVERA</LOC></DEPT_ROW>	Cambia el nodo completo DEP_ROW del departamento 10, por los nuevos datos y las etiquetas que escribamos.

- **Value**, actualiza el valor del nodo especificado en NODO con VALOR\_NUEVO. Si NODO es un nodo de texto ó atributo su valor será actualizado con la concatenación de todos los valores de VALOR\_NUEVO.

**update value NODO with 'VALOR NUEVO'**

update value /EMPLEADOS/EMP_ROW[EMP_NO=7369]/APELLIDO with 'Alberto Montes Ramos'	Cambia el apellido del empleado 7369, del documento <i>empleados.xml</i>
update value /sucursales/sucursal[@codigo='SUC3']/cuenta[1]/@tipo with 'NUEVOTIPO'	Cambia el atributo tipo de la primera cuenta de la sucursal SUC3, del documento <i>sucursales.xml</i> .
for \$em in /EMPLEADOS/EMP_ROW[DEPT_NO=10] let \$sal := \$em/SALARIO return update value \$em/SALARIO with data(\$sal)+200	Cambia el salario de los empleados del departamento 10, del documento <i>empleados.xml</i> , subirles 200.

- **Delete**, elimina los nodos indicados en la expresión: **update delete expr**

update delete /zonas/zona[cod zona=50]	Elimina la zona con código 50, en el documento <i>zonas.xml</i>
--	---

- **Rename**. Renombra los nodos devueltos en NODO (debe devolver una relación de nodos o atributos) por el NUEVO\_NOMBRE.

**update rename NODO as NUEVO\_NOMBRE**

update rename /EMPLEADOS/EMP_ROW as 'fila_emple'	Cambia de nombre el nodo EMP_ROW del documento <i>empleados.xml</i>
--	---

**Actividad 7:** A partir del documento *universidad.xml*.

Añade un empleado al departamento que ocupa la posición 2. Los datos son el salario 2340, el puesto Técnico, y nombre Pedro Fraile.

Actualiza el salario de los empleados del departamento con código MAT1. Suma al salario 100.

Renombra el nodo DEP\_ROW del documento *departamentos.xml* por *filadepar*.

---

## 4 ACCESO A EXIST DESDE JAVA

En este apartado vamos a ver distintas APIs que acceden a la BD eXist para procesar documentos XML.

### 4.1 LA API XMLDB PARA BASES DE DATOS XML

Actualmente, la API está siendo utilizada por la mayoría de bases de datos. La API propuesta se basa en tres paquetes: **org.xmldb.api**, **org.xmldb.api.base** y **org.xmldb.api.modules**.

Los componentes básicos empleados por el código XML: DB API son los drivers, las colecciones, los recursos y los servicios. Veamos cada uno de ellos:

- **Los drivers** son implementaciones de la interfaz que encapsula la lógica de acceso a la base de datos XML. Los proporciona el proveedor del producto y debe ser registrado con el gestor de base de datos. Ejemplo:

```
String driver = "org.exist.xmldb.DatabasImpl";           //Driver para eXist
Class cl = Class.forName(driver);                       //Cargar del driver
Database database = (Database) cl.newInstance();         //Instancia de la BD
DatabaseManager.registerDatabase(database);              //Registro del driver
```

- **Una colección** es un contenedor de recursos y otras sub-colecciones. La API define dos recursos diferentes: XMLResource y BinaryResource. Un XMLResource representa un documento XML o un fragmento del documento, seleccionados por la ejecución de una consulta XPath. Una vez utilizado el recurso se debe de cerrar. Ejemplo:

```
String URI="xmldb:exist://localhost:8080/exist/xmlrpc/db/Pruebas"; //Colección
String usu="admin";                                              //Usuario
String usuPwd="admin";                                          //Clave
Collection col = DatabaseManager.getCollection(URI, usu, usuPwd);
```

- **Los servicios** se solicitan para tareas como consultar una colección con XPath o la gestión de una colección. Ejemplo:

```
XPathQueryService servicio = (XPathQueryService) col.getService("XPathQueryService", "1.0");
ResourceSet result = servicio.query("for $em in /EMPLEADOS/EMP_ROW return $em");
```

Para ejecutar la consulta llamaremos al método ***nombre\_servicio.query(xpath)***, este método devuelve un ***ResourceSet***, que contiene el recurso, en el ejemplo anterior el resultado de la consulta es devuelto en *result*. El siguiente código lo escribiremos para recorrer el recurso *result* devuelto por la consulta anterior:

```
ResourceIterator i; //se utiliza para recorrer un set de recursos
i = result.getIterator();
if (!i.hasMoreResources ()) {
    System.out.println(" LA CONSULTA NO DEVUELVE NADA.");
}
while (i.hasMoreResources ()) {
    Resource r = i.nextResource();
    System.out.println((String) r.getContent());
}
```

Donde ***result.getIterator()*** nos da un iterador sobre el recurso, cada recurso contiene el valor seleccionado por la expresión XPath. El método ***getContent()***, devuelve el contenido del recurso, en nuestro ejemplo devuelve la consulta y el tipo es String.

Para localizar si existe un documento en una colección utilizaremos este código:

```
Collection col= DatabaseManager.getCollection(URI, usu, usuPwd);
XMLResource res = null;
res = (XMLResource)col.getResource("empleados.xml");
if(res == null)
    System.out.println("NO EXISTE EL DOCUMENTO");
```

Para realizar un programa que consulte la BD eXist debemos incluir las siguientes librerías: *exist.jar*, *exist-optional.jar*, *xmldb.jar*, *xml-apis-13.04.jar*, *xmlrpc-client-3.1.1.jar*, *xmlrpc-common-3.1.1.jar* situadas en la instalación de eXist y *log4j-1.2.15.jar*.

En el siguiente ejemplo vemos cómo utilizar las clases vistas anteriormente, este ejercicio realiza una conexión con la BD para acceder al documento *empleados.xml* y obtener en XML los empleados del departamento 10.

```
import org.xmldb.api.*;
import org.xmldb.api.base.*;
import org.xmldb.api.modules.*;

public static void verempleados10(String[] args) throws XMLDBException {
    String driver = "org.exist.xmldb.DatabaseImpl"; //Driver para eXist
    Collection col = null; // Colección
    String URI="xmldb:exist://localhost:8080/exist/xmlrpc/db/Pruebas"; //URI colección
    String usu="admin"; //Usuario
    String usuPwd="admin"; //Clave
    try {
        Class cl = Class.forName(driver); //Cargar del driver
        Database database = (Database) cl.newInstance(); //Instancia de la BD
        DatabaseManager.registerDatabase(database); //Registro del driver
    } catch (Exception e) {
        System.out.println("Error al inicializar la BD eXist");
        e.printStackTrace(); }

    col = DatabaseManager.getCollection(URI, usu, usuPwd);
    if(col == null)
        System.out.println(" *** LA COLECCION NO EXISTE. ***");

    XPathQueryService servicio = (XPathQueryService) col.getService("XPathQueryService", "1.0");
```

```

ResourceSet result = servicio.query ("for $em in /EMPLEADOS/EMP_ROW[DEPT_NO=10] return $em");

// recorrer los datos del recurso.
ResourceIterator i;
i = result.getIterator();
if (!i.hasMoreResources())
    System.out.println(" LA CONSULTA NO DEVUELVE NADA.");
while (i.hasMoreResources()) {
    Resource r = i.nextResource();
    System.out.println((String) r.getContent());
}
col.close(); //borramos
} // FIN verempleados10

```

**Actividad 8:** Realiza los cambios necesarios al ejercicio anterior para leer de teclado un departamento y visualizar sus empleados. Utiliza la entrada estándar. El código para la entrada estándar es el siguiente:

```

// import para la entrada estandar
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

System.out.println("Teclea departamento:"); // se lee un dato de tipo cadena
String s = null;
try{
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    s = in.readLine();
}catch(IOException e){
    System.out.println("Error al leer");
    e.printStackTrace ();
}
int dep=Integer.parseInt(s); // convertimos a numérico

```

Realiza un programa Java que inserte, elimine y modifique departamentos del documento *departamentos.xml*. Utiliza las *Sentencias de actualización de eXist* Los datos se leerán de la entrada estándar de teclado. Haz que la función *main()* llame y ejecute los siguientes métodos (no devuelven nada):

- ***Insertadep()***, este método leerá de teclado un departamento, su nombre y su localidad, y deberá añadirlo al documento. Si el código de departamento existe visualiza que no se puede insertar porque ya existe.
- ***Borradep()***, este método leerá de teclado un departamento y deberá borrarlo si existe, si no existe visualiza que no se puede borrar porque ya existe.
- ***Modificadep()***, este método leerá de teclado un departamento, su nombre nuevo y la localidad nueva y deberá actualizar todos los datos si existe, si no existe visualiza que no se puede modificar porque ya existe.

#### 4.1.1. OPERACIONES SOBRE COLECCIONES Y DOCUMENTOS

La API **XMLDB** nos va a permitir además de consultar documentos en la BD, crear y eliminar colecciones, y crear y eliminar documentos.

- **Crear una colección:** para crear una nueva colección, se llama al método **createCollection** del servicio **CollectionManagementService**. El siguiente ejemplo crea la colección NUEVA\_ COLECCION dentro de la colección **col**:

```
CollectionManagementService mgtService =  
    (CollectionManagementService)col.getService("CollectionManagementService","1.0");  
mgtService.createCollection("NUEVA_COLECCION");
```

- **Borrar una colección:** para borrar utilizamos el método `removeCollection` :

```
mgtService = (CollectionManagementService)col.getService("CollectionManagementService","1.0");  
mgtService.removeCollection("NUEVA_COLECCION");
```

- **Crear un nuevo documento:** este ejemplo añade un documento nuevo a la colección **col**, el documento se llama *NUEVOS\_DEP.xml*, y se encuentra en nuestro disco, en la carpeta donde está el programa. Utilizamos el paquete *java.io.File*, para declarar el fichero a subir a la BD, y el método **createResource** para crear el recurso.

```
import java.io.File;  
.....  
File archivo= new File("NUEVOS_DEP.xml");  
if(!archivo.canRead()) System.out.println("ERROR AL LEER EL FICHERO");  
else  
{  
    Resource nuevoRecurso = col.createResource(archivo.getName(), "XMLResource");  
    nuevoRecurso.setContent(archivo);           //comprueba si es un archivo  
    col.storeResource(nuevoRecurso); }  

```

- **Borrar un documento de la colección:** este ejemplo borra el documento creado anteriormente, comprueba si existe. Se utiliza el método **removeResource**:

```
try  
{  
    Resource recursoParaBorrar = col.getResource("NUEVOS_DEP.xml");  
    col.removeResource(recursoParaBorrar);  
}catch(NullPointerException e)  
{  
    System.out.println("No se puede borrar. No se encuentra."); }  

```

### Actividad 9:

Haz un programa Java que cree la colección GIMNASIO y suba los documentos que se encuentran en la carpeta **ColeccionGimnasio**. Los documentos son los siguientes:

- *socios\_gim.xml* contiene información de los socios que asisten a hacer deporte en un Gimnasio.
- *actividades\_gim.xml* contiene información de las actividades que se pueden realizar en el Gimnasio. Hay 3 tipos de actividades:
  - 1 - son actividades de libre horario, el socio no paga cuota adicional por ellas, por ejemplo: aparatos o piscina.
  - 2 - representan actividades que se realizan en grupo, como por ejemplo: aerobico o pilates. El socio paga una cuota adicional de 2€ por cada hora que dedique a la actividad.
  - 3 - representan actividades en las que se alquila un espacio, por ejemplo padel o tenis. El socio paga una cuota adicional de 4€ por cada hora que dedique a la actividad.
- *actividades\_gim.xml* contiene información de las actividades que se pueden realizar en el Gimnasio. Hay 3 tipos de actividades:
- *Uso\_gimnasio.xml*, contiene las actividades que realizan los socios en el Gimnasio durante el año, cada fila representa una actividad realizada por el socio con la fecha (dd/mm/yy), la hora de inicio (por ejemplo 17) y la hora de finalización (por ejemplo 18):

- A partir de esos documentos haz un método java para obtener por cada socio la cuota que tiene que pagar. Obtener el CODSOCIO y la CUOTA\_FINAL

Esta CUOTA\_FINAL será igual a la suma de la CUOTA\_FIJA y las CUOTAS ADICIONALES que dependerán de las actividades realizadas por el socio.

El programa java debe crear un documento XML intermedio que obtenga la cuota adicional a obtener por cada actividad realizada por cada usuario, el documento debe contener estas etiquetas:

```
<datos><COD>xxxxx</COD><NOMBRESOCIO>xxxxxxxx</NOMBRESOCIO>  
  <CODACTIV>xxxxxxx</CODACTIV><NOMBREACTIVIDAD>xxxxxxxx</NOMBREACTIVIDAD>  
  <horas>xxxx</horas><tipoact>xxxx</tipoact><cuota_adicional>xxxx</cuota_adicional>  
</datos>
```

Añade el documento a la colección GIMNASIO. Una vez creado y añadido el documento, el programa java debe obtener la cuota final total, que será la suma de las cuotas adicionales de las actividades mas la cuota fija. Obtén las siguientes etiquetas:

```
<datos>  
  <COD>xxxxxx</COD>  
  <NOMBRESOCIO>xxxxxxxxxx</NOMBRESOCIO><CUOTA_FIJA>xxxxxx</CUOTA_FIJA>  
  <suma_cuota_adic>xxxxx</suma_cuota_adic><cuota_total>xxxxxxx</cuota_total>  
</datos>
```

## 4.2 LA API XQJ (XQUERY)

La API XQJ es una propuesta de estandarización de interfaz Java para el acceso a bases de datos XML nativas basado en el modelo de datos XQuery.

Para descargar la API accedemos a la URL: <http://xqj.net/exist/>. Utilizaremos las siguientes interfaces para conectarnos y obtener los datos de la base de datos:

```
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 el origen de datos a partir del cual se va a crear la conexión. 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 (admin). 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", "admin");
```

- **XQConnection.** Representa una sesión con la base de datos. Por ejemplo:

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

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

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

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

### Procesar los resultados de una consulta:

- **XQPreparedExpression:** Objeto creado a partir de una conexión para la ejecución de una expresión múltiples veces. Devuelve un **XQResultsetSequence** con los datos obtenidos. La ejecución se produce llamando al método **executeQuery**.
- **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=10]");
resultado = consulta.executeQuery();
while(resultado.next())
    System.out.println("Elemento: " +resultado.getItemAsString(null));

```

- **XQResultItem:** Objeto que representa un elemento de un resultado, válido hasta que se llama al método close.

El ejemplo anterior lo podemos poner así:

```

consulta =conn.prepareExpression("/EMPLEADOS/EMP_ROW[DEPT_NO=10]");
resultado = consulta.executeQuery();
XQResultItem r_item;
while(resultado.next()){
    r_item = (XQResultItem) resultado.getItem() ;
    System.out.println("Elemento: " + r_item.getItemAsString(null)); }

```

- **XQItem:** Representación de un elemento en XQuery.
- **XQSequence:** Contiene un conjunto de 0 o más XQItem. Es un objeto recorrible.

Con XQJ la búsqueda la realiza en todas las colecciones. Tampoco admite el uso de *Sentencias de actualización de eXist*.

El siguiente ejemplo mostrará los datos del documento *productos.xml*, si tenemos productos en distintos documentos se mostrarán las etiquetas de todos los documentos:

```

public static void productos (String[] args) throws XQException
{
    XQDataSource server = new ExistXQDataSource();
    server.setProperty("serverName", "localhost");
    XQPreparedExpression consulta;
    XQResultSequence resultado;
    XQConnection conn = server.getConnection();

    System.out.println(" ----- Consulta documentos productos.xml ----- ");
    consulta =conn.prepareExpression("for $de in /productos return $de");
    resultado = consulta.executeQuery();

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

```

**Actividad 10:** Realiza un programa Java que contenga una función *main()* que lea de la entrada estándar el tipo de departamento cuyos empleados se quieren visualizar. Debe invocar al método *Visualizar(tipo)*, que recibe el tipo y visualiza los empleados de los departamentos del tipo indicado. Si no hay empleados o si el tipo no existe se debe visualizar un mensaje indicándolo. Utiliza el documento *universidad.xml*.

El siguiente ejemplo crea un fichero XML en el disco con nombre *NUEVO\_EMPLE10.xml*, a partir de los datos extraídos de una consulta. Para crear el fichero utilizaremos la **clase File** del paquete **java.io**. La consulta devuelve los empleados del departamento 10, más el título del documento *empleados.xml*, de la base de datos, la consulta es la siguiente:

```
let $titulo:= /EMPLEADOS/TITULO
return <EMPLEADOS>{$titulo}
      {for $em in /EMPLEADOS/EMP_ROW[DEPT_NO=10]
      return $em}</EMPLEADOS>
```

El ejercicio nos queda así:

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import javax.xml.xquery.*;
import net.xqj.exist.ExistXQDataSource;

public class creaFicheroexterno {
    public static void main(String[] args) throws XQException {
        String nom_archivo = "NUEVO_EMPLE10.xml";
        File fichero = new File(nom_archivo);

        XQDataSource server = new ExistXQDataSource();
        server.setProperty("serverName", "localhost");
        server.setProperty("port", "8080");
        XQConnection conn = server.getConnection();
        XQPreparedExpression consulta = conn.prepareExpression
            "let $titulo:= /EMPLEADOS/TITULO return <EMPLEADOS>{$titulo}" + "
            {for $em in /EMPLEADOS/EMP_ROW[DEPT_NO=10]} +
            " return $em)</EMPLEADOS>";
        XQResultSequence result = consulta.executeQuery();
        if (fichero.exists ())
        {
            //borramos el archivo si existe y se crea de nuevo
            if(fichero.delete()) System.out.println("Archivo borrado. Creo de nuevo.");
            else System.out.println("Error al borrar el archivo");
        }
        try{
            BufferedWriter bw = new BufferedWriter(new FileWriter(nom_archivo));
            bw.write("<?xml version='1.0' encoding='ISO-8859-1'?>" + "\n");
            while (result.next()) {
                String cad = result.getItemAsString(null);
                System.out.println(" output "+ cad);           //visualizamos
                bw.write (cad+ "\n");                          // .grabamos en el fichero
            }
            bw.close();                                         // Cerramos el fichero el fichero
        } catch (IOException ioe){ioe.printStackTrace();}
        conn.close ();
    }
}
```

**Actividad 11:** A partir de los documentos *productos.xml* y *zonas.xml*

Realiza un programa Java que cree un documento externo con nombre *zonas20.xml*.

Debe contener los productos de la zona 20 y las siguientes etiquetas para cada producto: `<cod_prod>`, `<denominadon>`, `<precio>`, `<nombre_zona>`, `<director>` y `<stock>`, este stock debe ser el cálculo del `stock_actual` – `stock_minimo`.

---

## 4.3 TRATAMIENTO DE EXCEPCIONES

### XMLDBEXCEPTION:

Esta excepción se lanza cuando se produce un error en la API XML:DB. Contiene dos códigos de error, uno es el código de error XML de la API, y el otro es definido por el proveedor específico. El error del proveedor vendrá definido por *ErrorCodes*. *VENDO\_ERROR*.

Cuando se produce un error con **XMLDBException** podemos acceder a cierta información usando el método *getMessage()* que devuelve una cadena que describe el error.

- El error **NullPointerException**, es el error que se dispara siempre que no se haya inicializado la BD, es el caso de escribir mal el driver.
- Si escribimos mal la URI, se disparará **XMLDBException** a la hora de asignar la colección.
- Si escribimos mal la carpeta donde se encuentra la colección, se disparan los **NullPointerException** siguientes, ya que el servicio no se ha podido crear.
- Si escribimos mal la query se dispara el error **XMLDBException**, a la hora de crear el service.

### XQEXCEPTION:

Cuando se produce un error con **XQException** disponemos de dos métodos que nos van a permitir saber la causa. Estos métodos son *getMessage()* que devuelve una cadena que describe el error y *getCause()* que nos indica la causa.

## 5 EJERCICIOS

1. A partir del documento *departamentos.xml* de la colección *Pruebas* realiza un programa Java para gestionar dicho documento. Utiliza las clases *Main.java* y *Pantalla.java* del Ejemplo 2 de la Unidad 3. El programa debe mostrar la pantalla inicial donde podremos consultar, insertar, borrar y eliminar nodos *<DEPT\_ROW>* del documento *departamentos.xml*.

Comprueba que al insertar un departamento no exista ya en el documento, igualmente a la hora de borrar o de modificar hay que comprobar que el departamento exista. Visualiza los mensajes informativos en cada caso.

Crea una clase para gestionar las operaciones sobre el documento, esta clase debe contener los métodos para conectarnos a *eXist*, insertar, borrar, modificar o consultar en el documento. Utiliza las *Sentencias de actualización de eXist*.

GESTIÓN DE DEPARTAMENTOS

Nº de departamento:  Consultar

Nombre:

Localidad:

Alta Baja Modificación Limpiar

### A PARTIR DE LA COLECCIÓN VENTAS:

2. Haz un programa Java que cree la colección *VENTAS* y suba los documentos que se encuentran en la carpeta ***ColeccionVentas***. Esta colección contiene los siguientes documentos XML:

***clientes.xml***: contiene datos de clientes que compran los productos. Por cada cliente tenemos

```
<clien numero="nn"> <nombre>xxxxxx</nombre>
                    <poblacion>xxxxxxx</poblacion>
                    <tlf>xxxxxxx</tlf>
                    <direccion>xxxxxxxxx</direccion>
</clien>
```

***productos.xml***: contiene los datos de los productos que son comprados por los clientes. Por cada producto tenemos su categoría y precio, que van como atributos, el código de producto, el nombre y el stock del producto:

```
<product categoria="xxx" pvp="xxxx">
  <codigo>xxxxxx</codigo>
  <nombre>xxxxxx</nombre>
  <stock>xxxxxxx</stock>
</product>
```

**facturas.xml:** contiene los datos de las facturas de los clientes. Por cada factura tenemos el número de factura como atributo, la fecha de factura, el importe y el número de cliente, los nodos son:

```
<factura numero="xxxx">
  <fecha>xxxxxx</fecha>
  <importe>xxxxxx</importe>
  <numcliente>xxxxxxx</numcliente>
</factura>
```

**detallefacturas.xml:** contiene el detalle de cada factura, es decir los datos de los productos y las unidades compradas por los clientes. También cada producto de la factura lleva asociado un porcentaje de descuento. Por cada factura se dispone de la siguiente información

```
<factura numero="xxxx">
  <codigo>xxxxxxx</codigo>
  <producto descuento="xxx">
    <codigo>xxx</codigo>
    <unidades>xxxxx</unidades>
  </producto>
  <producto descuento="xxx">
    <codigo>xxxxx</codigo>
    <unidades>xxxx</unidades>
  </producto>
  .....
</factura>
```

REALIZAR LOS SIGUIENTES

EJERCICIOS:

- 3. Realiza una consulta XQuery para obtener las facturas que tiene cada cliente, que aparezca solo el nombre del cliente y el número de factura entre las etiquetas <facturaclientes></facturaclientes>. Debe obtener algo parecido a esto:

```
<facturasclientes> <nombre>Pilar Martín</nombre> <nufac>10</nufac></facturasclientes>
<facturasclientes> <nombre>Pilar Martín</nombre> <nufac>11</nufac></facturasclientes>
<facturasclientes> <nombre>Antonio Reus</nombre> <nufac>12</nufac></facturasclientes>
```

- 4. Realiza una consulta XQuery para obtener el detalle de las ventas de los productos de la factura número 10. Obtén por cada producto de esa factura, el código, nombre, la cantidad vendida, el precio, y el importe que será la suma de las unidades por el precio del producto. Utiliza los documentos *productos.xml* y *detallefacturas.xml*. La salida debe ser similar a lo que se muestra:

```
<detalle><codigo>1</codigo><nombre>Silla Plegable</nombre>
  <cant>10</cant><pvp>100</pvp><importe>1000</importe>
</detalle>
<detalle><codigo>2</codigo><nombre>BH Prisma</nombre>
  <cant>3</cant><pvp>900</pvp><importe>2700</importe>
</detalle>
.....
```

5. Realiza una consulta XQuery para obtener el detalle de las ventas de los productos de cada una de las facturas del documento *detallefacturas.xml*. Crear una salida que muestre para cada factura, el número de factura y el código como atributos dentro de la etiqueta <factura>. Y a continuación los artículos, el código de artículo será un atributo de la etiqueta <articulo>. Utiliza los documentos *productos.xml* y *detallefacturas.xml*. La salida debe ser similar a lo que se muestra:

```
<factura numero="10" codigo="FACT10">
  <articulo codigo="1"><nombre>Silla Plegable</nombre>
    <cant>10</cant><pvp>100</pvp><importe>1000</importe>
  </articulo>
  <articulo codigo="2"><nombre>BH Prisma</nombre>
    <cant>3</cant><pvp>900</pvp><importe>2700</importe>
  </articulo>
</factura>
<factura numero="11" codigo="FACT11">
```

6. Haz un programa Java que genere un fichero XML a partir de la consulta creada anteriormente, llama al nuevo fichero *facturastotal.xml*. Una vez creado sube el fichero a la colección *Ventas* de la base de datos. Recuerda que el documento XML debe estar bien formado porque si no, no se podrá cargar en la base de datos.
7. Utilizando el documento creado anteriormente, realiza una consulta de actualización para actualizar la etiqueta <importe> del documento *facturas.xml*. La actualización consiste en actualizar el importe de cada factura con la suma de los importes de los artículos que componen la factura, obtén la suma de importes por cada factura a partir del documento creado anteriormente (*facturastotal.xml*).
8. Utilizando los documentos *facturas.xml* y *clientes.xml*, realiza una consulta XQuery para obtener todo lo que tiene que pagar cada cliente. La salida debe ser similar a:
- ```
<nombre>Alicia Díaz<totalapagar>0</totalapagar></nombre>
<nombre>Pilar Martín<totalapagar>8180</totalapagar></nombre>
```
9. Utilizando los datos devueltos por la consulta anterior, haz un programa Java para obtener el nombre del cliente que más importe tiene que pagar.