

TFC

# **Bases de datos XML nativas**

Memoria

**Manuel Sotillo Pérez**

## Índice

0 – Objetivos propuestos.....	5
1 – Estrategias de almacenamiento de datos codificados en XML .....	6
1.1 - ¿Qué es XML?.....	6
1.2 – Documentos centrados en datos y en contenido .....	6
1.3 – ¿Qué opciones hay para almacenar esos datos?.....	7
1.3.1 – Almacenamiento directo de ficheros .....	7
1.3.2 – Almacenamiento sobre una base de datos .....	7
1.3.3 – Almacenamiento sobre una base de datos XML.....	10
2 – SGBD XML-enabled.....	11
2.1 –SQL XML (Estándar SQL 2003).....	11
2.2 – Oracle .....	13
2.2 – DB2.....	15
3 – SGBD XML nativos .....	17
3.1 – Características básicas de un SGBD XML nativo y diferencias con un SGBD relacional .....	17
3.2 – Tamino.....	17
Arquitectura .....	17
Almacenamiento de documentos .....	18
Índices.....	19
Acceso a los datos.....	19
Otras funcionalidades .....	20
3.3 – eXist.....	20
Arquitectura .....	20
Almacenamiento de documentos .....	21
Índices.....	21
Acceso a datos .....	22
Otras funcionalidades .....	22
4 – Conclusiones sobre SGBD .....	23
4.1 – Que tipo de SGBD utilizar. ....	23
4.2 – Próximos pasos en el desarrollo de SGBDs.....	23
4.3 – SGBD XML seleccionado.....	24
5 – XPath 1.0.....	25
5.1 – Que es XPath.....	25
5.2 – Modelo de datos.....	25
Nodos.....	25
Expresiones .....	26
Funciones .....	28
5.3 – Como se expresan las selecciones de nodos.....	30
5.3.1 – Rutas relativas y absolutas.....	30
5.3.2 – Composición del location-path .....	30
5.3.3 – Escritura abreviada de expresiones. ....	32
5.4 – Ejemplos. ....	33
6 – XPath 2.0.....	39
6.1 – Diferencias con XPath 1.0.....	39
6.2 – Modelo de datos.....	39
Contexto .....	40
Expresiones .....	41
Secuencias .....	41
Tipos de datos .....	42
Nodos.....	42

Funciones .....	43
6.3 – Funcionalidades añadidas.....	43
6.3.1 – Funciones de entrada .....	43
6.3.2 – Tratamiento iterativo .....	43
6.3.3 – Tratamiento condicional .....	44
6.3.4 – Cuantificadores existenciales .....	45
6.3.5 – Operadores lógicos en expresiones .....	45
6.3.6 – Combinación de secuencias.....	46
6.3.7 – Comentarios .....	46
6.3.8 – Expresiones aritméticas .....	47
6.3.8 – Instance of .....	47
6.4 – Ejemplos .....	47
7 – XQuery .....	51
7.1 – Que es XQuery .....	51
7.2 – Modelo de datos.....	51
Expresiones .....	51
Axis .....	51
7.3 – Sentencias FLWOR. ....	52
FOR .....	52
LET .....	54
Uso conjunto de FOR y LET .....	55
WHERE.....	55
ORDER BY .....	56
RETURN.....	57
7.4 – Creación de nodos. ....	58
7.5 – Creación de variables.....	59
7.6 – Declaración de funciones. ....	60
7.7 – Módulos. ....	61
Declaración de versión .....	62
Declaración de modulo .....	62
Prólogo.....	62
Ejemplo de uso de módulos .....	66
7.8 – JOINS, como hacerlos .....	67
7.8.1 INNER JOIN .....	67
7.8.2 OUTER JOIN .....	69
7.8.3 FULL OUTER JOIN .....	70
7.9 – Funciones agregadas.....	71
7.10 – Algunos ejemplos más. ....	72
8 – Actualización de datos.....	76
8.1 – XUpdate.....	76
Inserción de nodos .....	77
Actualización de contenido .....	78
Borrado de elemento .....	78
Cambio de nombre .....	78
8.2 – XQuery Update Facility .....	79
Insert.....	79
Delete.....	80
Replace.....	80
Rename .....	81
Uso con las sentencias FLWOR y condicionales .....	81
8.3 – Extensiones de actualización de eXist .....	81
Insert.....	82
Replace.....	82
Value.....	82
Delete.....	82
Rename .....	83
8.4 – Ejemplos .....	83

9 – Acceso desde programas.....	87
9.1 – Métodos de acceso .....	87
XML-RPC.....	87
SOAP.....	88
Web-DAV.....	89
Librerías específicas del lenguaje .....	89
9.2 – XML:DB. ....	90
9.3 – XQJ.....	91
9.4 – Implementación de eXist.....	94
10 – Aplicación de ejemplo.....	95
Objetivo .....	95
Herramientas utilizadas.....	95
Diagrama de clases .....	95
Descripción de las clases.....	97
ExcepciónGestorBD.java.....	97
ElementoBD.java .....	97
Usuario.java .....	98
RenderArbol.java .....	99
UsuarioTableModel.java .....	99
GestorBD.java .....	99
DialogoVerUsuario.java.....	104
EstructuraBDSwing.java.....	105
Instalación de la aplicación .....	108
Requisitos para el funcionamiento .....	108
Proceso de instalación.....	108
Compilación desde la línea de comandos.....	110
Funcionamiento de la aplicación .....	110
Gestión de colecciones.....	114
Borrado de recursos .....	115
Consulta de usuarios .....	116
Ejecutar consultas .....	116
Mejoras futuras a realizar.....	117
11 – Conclusiones.....	119
12 – Anexos .....	121
1 – Instalación de eXist.....	121
2 – Definición de una colección.....	128
3 – Añadir un documento a una colección.....	129
4 – Query Dialog.....	131
9 – Código fuente de las clases desarrolladas.....	133
ExcepciónGestorBD.java.....	133
ElementoBD.java .....	134
Usuario.java .....	135
RenderArbol.java .....	136
UsuarioTableModel.java .....	138
GestorBD.java .....	140
DialogoVerUsuario.java.....	146
EstructuraBDSwing.java.....	148
Bibliografía .....	158

## 0 – Objetivos propuestos

Los objetivos marcados con la realización de este trabajo son los siguientes:

- Conocer las distintas opciones a la hora de almacenar datos y documentos con formato XML
  - SGBD XML-enabled
  - SGBD XML nativos: características básicas y modo de almacenamiento de documentos.
  - Conocer las diferencias entre SGBD relacionales y XML
    - Saber decidir que tipo de gestor es adecuado para cada situación.
- Familiarizarse con los SGBD nativos y el acceso y manejo de la información.
  - Instalación y administración de un SGBD.
  - Aprender los lenguajes de consulta existentes y las posibilidades que ofrecen
    - XPath 1.0/2.0
    - XQuery
  - Conocer los lenguajes de actualización de datos
- Conocer distintos modos de acceso al SGBD, conocer las APIs disponibles centrándonos en el lenguaje Java
  - XMLDB
  - XQJ
- Integración de todos los conocimientos adquiridos, desarrollando una aplicación que acceda y gestione datos almacenados en el SGBD.

## 1 – Estrategias de almacenamiento de datos codificados en XML

### 1.1 - ¿Qué es XML?

XML (extensible Markup Language) es un metalenguaje que nos proporciona una manera sencilla de definición de lenguajes de etiquetas estructurados, en otras palabras, XML define un conjunto de reglas semánticas que nos permiten la organización de información de distintas maneras. Es un estándar definido por el W3C, ofrece muchas ventajas

- Bien formado
- Extensible: Permite ampliar el lenguaje mediante nuevas etiquetas y la definición de lenguajes nuevos
- Existe facilidad para la conversión entre los distintos vocabularios definidos
- Fácil de leer: Al estar codificado textualmente cualquier persona puede leerlo con cierta facilidad.
- Autodescriptivo: La estructura de la información de alguna manera está definida dentro del mismo documento
- Intercambiable: Portable entre distintas arquitecturas
- Para su lectura e interpretación es necesario un parser, y hay productos y versiones libres.

Desde su definición y debido a estas ventajas, el estándar ha sido ampliamente aceptado y adoptado para el almacenamiento e intercambio de información y junto con este uso se ha creado la necesidad de almacenar dicha información.

### 1.2 – Documentos centrados en datos y en contenido

Para enfrentarnos a dicha necesidad y definir el mejor modo de hacerlo primero debemos hacer una pequeña reflexión sobre los tipos de documento que nos podemos encontrar: documentos centrados en datos y en contenido:

Documentos centrados en datos (data centric)

Son documentos para el intercambio de datos (generalmente entre máquinas). Suelen ser documentos con estructuras regulares y bien definidas. Los distintos datos que transmiten son partículas atómicas bien definidas. Tendrán como origen o destino una base de datos

Documentos centrados en el contenido (document centric)

Por el contrario los documentos centrados en el contenido son documentos con una estructura irregular (aunque posean formato no es tan estricto y definido, decimos en este caso que son semi-estructurados). El origen y destino de este tipo de documentos suelen ser personas, y suelen ser creados a mano.

Esta clasificación de documentos no es siempre directa y clara, y en múltiples ocasiones el contenido estará mezclado o será difícil de catalogar en un tipo u otro (podemos tener un documento centrado en datos donde uno de los datos sea una parte de codificación libre, o un documento centrado en el contenido con una parte regular con formato estricto y bien definido), el interés esta distinción es disponer de un punto de partida para poder seleccionar o descartar el modo en el que vamos a almacenar los datos.

### **1.3 – ¿Qué opciones hay para almacenar esos datos?**

A la hora de almacenar estos documentos se nos plantean varias opciones

- Almacenamiento directo del fichero
- Almacenar el documento en una base de datos (SGBD relacional)
  - Directamente como un campo en una tabla
  - Mapeo basado en tablas
  - Mapeo basado en objetos
- Almacenar el fichero en una base de datos XML

Veamos ventajas e inconvenientes de cada una de estas opciones.

#### **1.3.1 – Almacenamiento directo de ficheros**

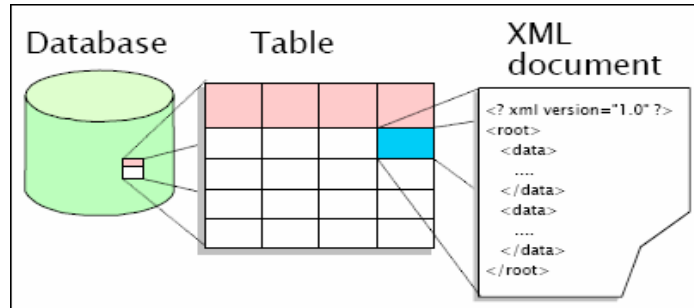
Es una opción pobre, y las opciones que podemos hacer sobre ellos son limitadas y definida por el sistema. No se puede realizar operaciones sobre el contenido y deberemos limitarnos al movimiento del documento como unidad.

#### **1.3.2 – Almacenamiento sobre una base de datos**

En documentos centrados en datos se puede realizar un mapeo entre los distintos elementos definidos en el documento y el modelo de datos del SGBD.

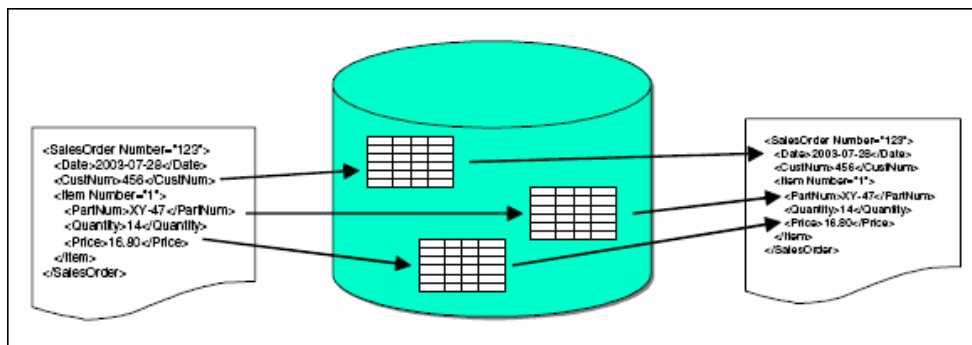
Esta posibilidad está centrada en documentos 'data centric' debido a que una estructura regular y bien controlada es fácilmente transformable en un esquema relacional, sin embargo, posee el inconveniente que sólo se almacenan los datos que nos interesan conservar, y partes del documento son perdidas por el camino (por ejemplo el formato, comentarios, instrucciones de proceso...) y a la hora de reconstruir el documento a partir de los datos almacenados obtendremos otro distinto.

- Directamente sobre un campo



La opción de almacenar el documento entero sobre un campo CLOB en una tabla en la base de datos tiene la ventaja que se mantiene el formato del documento, pero el gran inconveniente de no poder realizar en principio ninguna operación de consulta sobre su contenido.

- Mapeo basado en tablas



Los datos en el documento son vistos como filas, y estos están agrupados dentro de tablas.

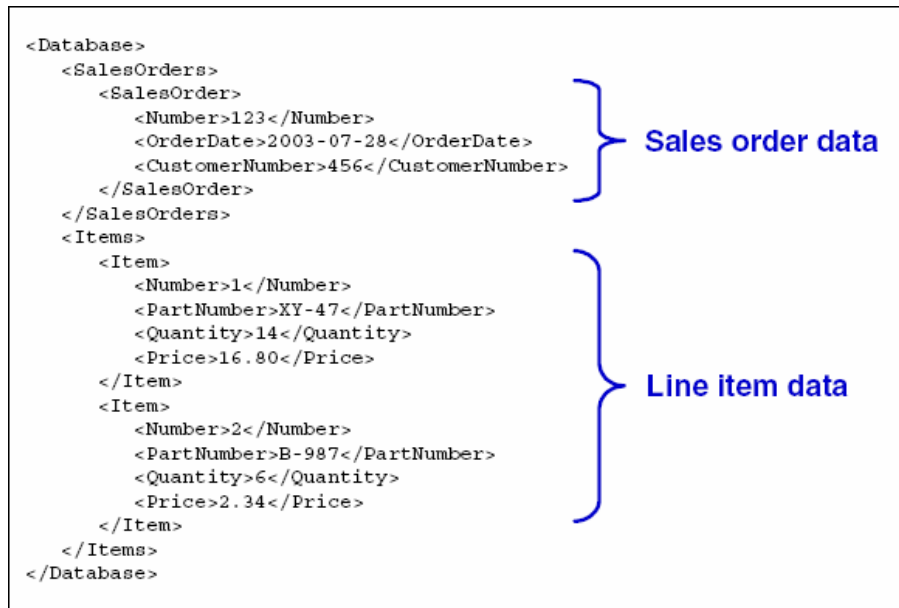
Cada fila corresponderá a una tupla en una tabla, y cada dato dentro de esa fila corresponderá a un campo de la tabla



```

<Database>
  <SalesOrders>
    <SalesOrder>
      <Number>123</Number>
      <OrderDate>2003-07-28</OrderDate>
      <CustomerNumber>456</CustomerNumber>
    </SalesOrder>
  </SalesOrders>
  <Items>
    <Item>
      <Number>1</Number>
      <PartNumber>XY-47</PartNumber>
      <Quantity>14</Quantity>
      <Price>16.80</Price>
    </Item>
    <Item>
      <Number>2</Number>
      <PartNumber>B-987</PartNumber>
      <Quantity>6</Quantity>
      <Price>2.34</Price>
    </Item>
  </Items>
</Database>

```



- Mapeo basado en objetos

Aunque hablemos de mapeo basado en objeto no nos referimos al uso de un SGBD-OO, seguimos trabajando sobre un SGBD relacional.

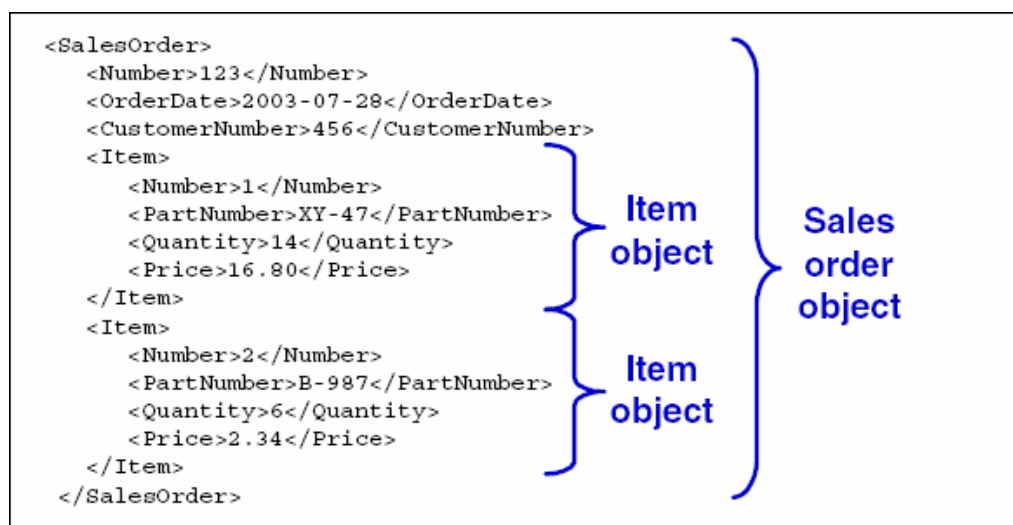
En este caso los datos del documento son tratados como objetos serializados; cada objeto va a una tabla y las propiedades del mismo a las columnas, las relaciones entre los objetos son modeladas como relaciones foreign key/primary key

Podemos ver este caso como una generalización del mapeo basado en tablas

```

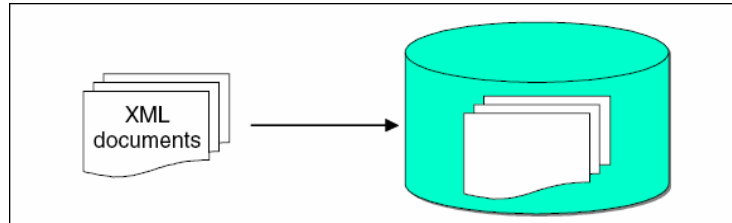
<SalesOrder>
  <Number>123</Number>
  <OrderDate>2003-07-28</OrderDate>
  <CustomerNumber>456</CustomerNumber>
  <Item>
    <Number>1</Number>
    <PartNumber>XY-47</PartNumber>
    <Quantity>14</Quantity>
    <Price>16.80</Price>
  </Item>
  <Item>
    <Number>2</Number>
    <PartNumber>B-987</PartNumber>
    <Quantity>6</Quantity>
    <Price>2.34</Price>
  </Item>
</SalesOrder>

```



Hay que volver a remarcar la necesidad de que la estructura del documento XML case a la perfección con el esquema de la base de datos.

### 1.3.3 – Almacenamiento sobre una base de datos XML



Por el contrario, si nuestro documento está centrado en el contenido, la estructura del mismo será irregular, y en esta situación no podemos controlar todas las posibles estructuras del documento, y realizar un mapeo sobre la base de datos será prácticamente imposible.

Además podemos tener la necesidad de reobtener el mismo documento que almacenamos en un momento dado; en esta situación la solución más aceptable consiste en la utilización de un SGBD XML nativo.

Como añadido podemos también nombrar que los SGBD XML nativos implementan algoritmos de búsqueda e índices específicos que aceleran el acceso a los documentos.

## 2 – SGBD XML-enabled.

Un SGBD XML-enabled es un SGBD tradicional con capacidad para el tratamiento y obtención de documentos XML. Vamos a enumerar las posibilidades básicas que ofrecen. La definición y evaluación de SGBD XML nativos quedará reflejada en el capítulo 3.

Hay que tener en cuenta que los grandes fabricantes de SGBD están aumentando las capacidades de tratamiento XML de sus productos, incluyendo características de SGBD XML nativos, por lo que en un futuro próximo es posible que la frontera entre estos dos tipos de gestores quede diluida.

Los siguientes puntos pretenden ser una revisión general sobre las características que ofrecen los SGBD para almacenar documentos XML sobre tablas con la finalidad de llegar a comprender mejor las diferencias con un SGBD XML nativo y poder decidir que tipo de sistema es mejor usar.

*Los ejemplos incluidos son simplemente informativos.*

### **2.1 –SQL XML (Estándar SQL 2003)**

SQL XML forma parte del estándar SQL 2003. En su sección 14, 'XML-Related Specifications SQL XML' se define el modo de trabajo conjunto de SQL y XML; el contenido central es la obtención de datos XML partiendo de datos en tablas relacionales; con esto no se genera ninguna tabla ni ningún tipo de esquema, y el documento debe de ser validado por el receptor del mismo.

Este documento es un estándar ISO sujeto a derechos de propiedad, y por tanto no se puede distribuir libremente, pero puede obtenerse referencia al mismo en:

[http://www.wiscorp.com/sql\\_2003\\_standard.zip](http://www.wiscorp.com/sql_2003_standard.zip)

Para obtener XML partiendo de consultas tenemos a nuestra disposición un conjunto de funciones que nos permiten dicha transformación; las aquí mencionadas solo son ejemplos sencillos de las funcionalidades básicas:

-XMLELEMENT: Con el podemos crear un nuevo elemento XML, mapeando sobre el un campo de la base de. Este operador permite el uso anidado.

<pre>SELECT e.id, XMLELEMENT ( NAME "Emp", e.fname    ' '    e.lname ) AS "result" FROM employees e WHERE ... ;</pre>	<pre>ID      result ----- 1001 &lt;Emp&gt;John Smith&lt;/Emp&gt; 1206 &lt;Emp&gt;Bob Martin&lt;/Emp&gt;</pre>
<pre>SELECT e.id, XMLELEMENT ( NAME "Emp", 'Employee ', XMLELEMENT (NAME "name", e.lname ), ' was hired on ', XMLELEMENT (NAME "hiredate", e.hire ) ) AS "result" FROM employees e WHERE ... ;</pre>	<pre>ID      result ----- 1001 &lt;Emp&gt;       Employee &lt;name&gt;Smith&lt;/name&gt;       was hired on       &lt;hiredate&gt;2000-05-24&lt;/hiredate&gt;       &lt;/Emp&gt; 1206 &lt;Emp&gt;       Employee &lt;name&gt;Martin&lt;/name&gt;       was hired on       &lt;hiredate&gt;1996-02-01&lt;/hiredate&gt;       &lt;/Emp&gt;</pre>

-XMLATTRIBUTES: Para mapear una columna sobre un atributo de un elemento

<pre>SELECT XML2CLOB( XMLELEMENT (NAME "EmployeeSalary", XMLATTRIBUTES (e.empno AS "id"), XMLELEMENT (NAME "Firstname", e.firstname), XMLELEMENT (NAME "Lastname", e.lastname), XMLELEMENT (NAME "TotalSalary", (e.salary+e.bonus+e.comm) )) FROM employee e WHERE SEX = 'F'</pre>	<pre>&lt;EmployeeSalary id="000010"&gt;   &lt;Firstname&gt;CHRISTINE&lt;/Firstname&gt;   &lt;Lastname&gt;HAAS&lt;/Lastname&gt;   &lt;TotalSalary&gt;000057970.00&lt;/TotalSalary&gt; &lt;/EmployeeSalary&gt;XMLFOREST</pre>
--	---

-XMLFOREST: Una manera abreviada de concatenación de elementos; para cada ítem de la lista genera un elemento XML

<pre>select XML2CLOB ( XMLELEMENT (NAME "EmployeeSalary", XMLATTRIBUTES (e.empno AS "id"), XMLFOREST (e.firstnme AS "Firstname", e.lastname AS "lastname", e.salary+e.bonus+e.comm as "salary") )) FROM employee e WHERE SEX = 'F'</pre>	<pre>&lt;EmployeeSalary id="000010"&gt;   &lt;Firstname&gt;CHRISTINE&lt;/Firstname&gt;   &lt;lastname&gt;HAAS&lt;/lastname&gt;   &lt;salary&gt;000057970.00&lt;/salary&gt; &lt;/EmployeeSalary&gt;</pre>
--	--

-XMLCONCAT: Toma un conjunto de expresiones XML y las devuelve como un único elemento

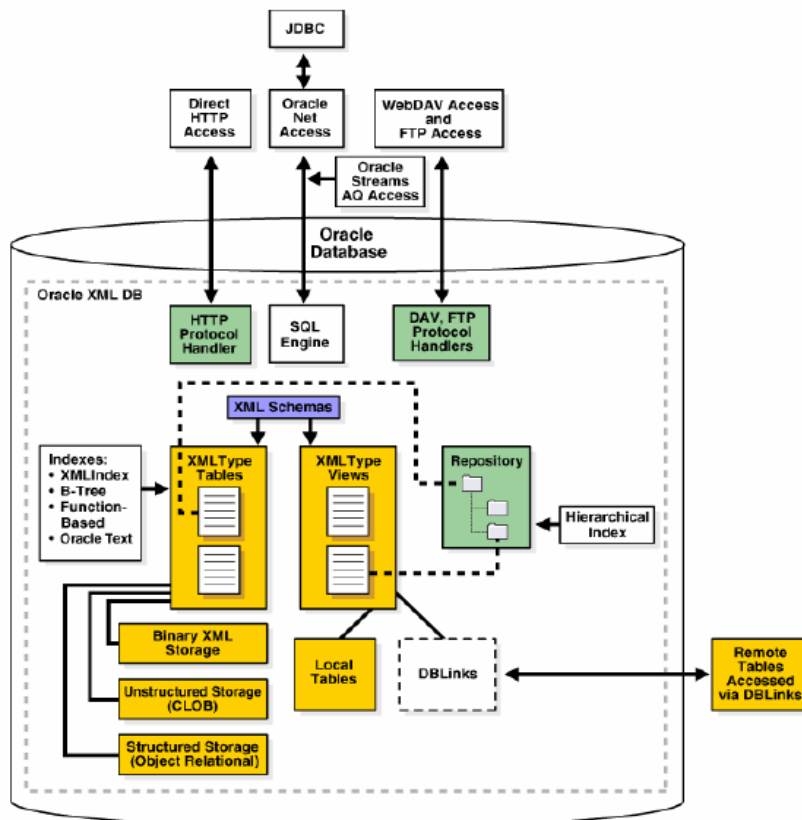
<pre>select XML2CLOB( XMLCONCAT( XMLELEMENT(NAME "Employee", XMLATTRIBUTES (e.firstnme    ' '    e.lastname as "Name"), XMLELEMENT (NAME "Salary", e.salary) ), XMLELEMENT (NAME "Employee", XMLATTRIBUTES (e.firstnme    ' '    e.lastname as "Name"), XMLELEMENT (NAME "Bonus", e.bonus) ), XMLELEMENT (NAME "Employee", XMLATTRIBUTES (e.firstnme    ' '    e.lastname as "Name"), XMLELEMENT (NAME "Commission", e.comm) ))) from employee e where sex = 'F'</pre>	<pre>&lt;Employee Name="CHRISTINE HAAS"&gt;   &lt;Salary&gt;0052750.00&lt;/Salary&gt; &lt;/Employee&gt; &lt;Employee Name="CHRISTINE HAAS"&gt;   &lt;Bonus&gt;0001000.00&lt;/Bonus&gt; &lt;/Employee&gt; &lt;Employee Name="CHRISTINE HAAS"&gt;   &lt;Commission&gt;0004220.00&lt;/Commission&gt; &lt;/Employee&gt;</pre>
--	---

-XMLAGG: Para generar secuencias de valores a partir; para cada elemento generado evalúa la función y devuelve una secuencia de valores

<pre>select XML2CLOB ( XMLELEMENT( NAME "Department", XMLATTRIBUTES(e.workdept as "Name"), XMLAGG ( XMLELEMENT(NAME "Employee", e.firstname  ' '  e.lastname) order by e.lastname) ) ) from employee e group by workdept</pre>	<pre>&lt;Department Name="A00"&gt;   &lt;Employee&gt;CHRISTINE HAAS&lt;/Employee&gt;   &lt;Employee&gt;VINCENZO LUCCHESSE&lt;/Employee&gt;   &lt;Employee&gt;SEAN O'CONNELL&lt;/Employee&gt; &lt;/Department&gt; &lt;Department Name="B01"&gt;   &lt;Employee&gt;MICHAEL THOMPSON&lt;/Employee&gt; &lt;/Department&gt; &lt;Department Name="C01"&gt;   &lt;Employee&gt;SALLY KWAN&lt;/Employee&gt;   &lt;Employee&gt;HEATHER NICHOLLS&lt;/Employee&gt;   &lt;Employee&gt;DOLORES QUINTANA&lt;/Employee&gt; &lt;/Department&gt;</pre>
--	--

Existe un estándar posterior todavía en fase de borrador, SQL 2006, donde se intenta definir el funcionamiento conjunto de SQL con XML y lenguajes de acceso como XQuery, pero está todavía en una fase temprana de desarrollo y tardará en aprobarse y ser adoptada por los fabricantes.

## 2.2 – Oracle



El componente encargado de dar soporte a las funcionalidades XML en Oracle es XML DB. Ofrece funciones como XML parser, validador de esquemas e interprete XSLT y funciones SQL XML.

El tratamiento de datos XML en Oracle gira en torno al tipo XMLType.

XMLType es un tipo de datos abstracto que puede ser usado para definir un campo de una tabla y para definir una tabla de XMLType. Este tipo de dato provee métodos para la validación contra un esquema, consulta y transformación del mismo,

Este tipo de campo puede ser almacenado siguiendo distintas estrategias.

- Unstructured storage: Almacenado en un campo CLOB, almacenamiento en modo texto.
- Structured Storage: Almacenado en un conjunto de tablas
- Binary XML: Documento procesado y almacenado de forma binaria en un formato propio

Para acceder a los datos puede usarse tanto XQuery como SQL XML, aunque cuando es posible internamente convertirá todas las consultas a SQL. El objetivo es que se pueda usar indistintamente cualquiera de los dos lenguajes de consulta para el acceso tanto a datos estructurados como XML.

Se da soporte para el estándar SQL XML e incluso para el estándar SQL 2006 en su fase previa.

Como ejemplo, si tuviésemos las siguientes tablas definidas

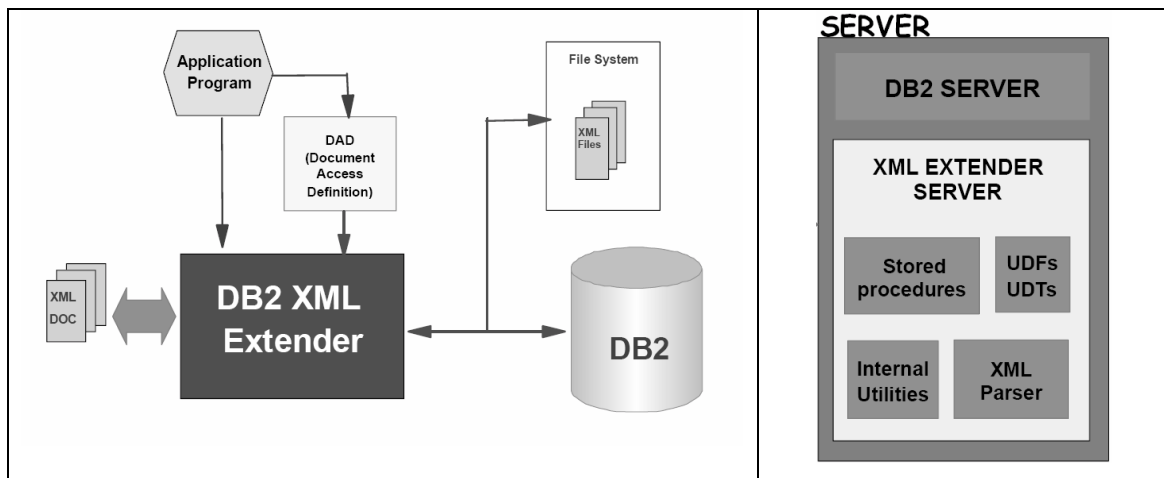
```
CREATE TABLE FXTRADE {
  CURRENCY1      CHAR (3),
  CURRENCY2      CHAR (3),
  AMOUNT         NUMERIC (18,2),
  SETTLEMENT     DATE,
  ACCOUNT        AccountType }

CREATE TYPE AccountType as OBJECT{
  BANKCODE       VARCHAR (100),
  BANKACCT       VARCHAR (100) }
```

Con la siguiente consulta `SELECT * FROM FXTRADE` Obtendríamos el siguiente resultado.

```
<? version="1.0"?>
<ROWSET>
  <ROW num="1">
    <CURRENCY1>GBP</CURRENCY1>
    <CURRENCY2>JPY</CURRENCY2>
    <AMOUNT>10000</AMOUNT>
    <SETTLEMENT>20010325</SETTLEMENT>
    <ACCOUNT>
      <BANKCODE>812</BANKCODE>
      <BANKACCT>00365888</BANKACCT>
    </ACCOUNT>
  </ROW>
  <!-- additional rows ... -->
</ROWSET>
```

## 2.2 – DB2



El núcleo de funcionalidad para el tratamiento de XML para DB2 lo proporciona el XML Extender; este componente engloba un parser XML, un procesador XSL, soporte para funciones y tipos definidos por el usuario (UDF y UDT) y forma parte del SGBD desde la versión 7.

Este componente da soporte para SQL XML

Ofrece tres modos de tratamiento básicos

- Almacenar referencia al sistema de archivos (el SGBD sirve únicamente como puntero al archivo)
- XML Columns para el almacenamiento del documento XML en una columna de la base de datos. A través de estos datos se dan soporte al tratamiento nativo de los documentos.
  - XMLVARCHAR para documentos pequeños
  - XMLCLOB igual que el anterior, pero pensado para documentos de cualquier tipo
  - XMLFILE, que es un puntero a un documento XML, pero almacenado fuera de la base de datos
- XML Collections: Almacenamiento de los elementos que componen el documento XML en un conjunto de tablas. la definición del mapeo se realiza a través de archivos DAD

Archivos DAD (Data Access Definition). Son documentos donde está contenida la relación entre elementos del documento y su lugar en la base de datos. Existen dos tipos distintos

- SQL mapping document: Sobre una sentencia SQL y como las columnas devueltas son mapeadas en el documento XML. Solamente se usan para publicar datos en formato XML.

- Relational Database Node Mapping (RDB). En este caso se especifica como los nodos de un documento XML son mapeados sobre las tablas de la BB.DD. Con este tipo de documento se permiten sentencias SELECT e INSERT, es decir, el mapeo es bidireccional.

Cuando al XML extender se les suministra estos documento DAD como parámetros a sus procedimientos almacenados, los interpreta y produce el resultado, por ejemplo, con la siguiente consulta

```
SELECT Orders.Number AS SONumber,
Orders.Customer AS CustNumber,
Items.Number AS ItemNumber,
Items.Part AS PartNumber
FROM Orders, Items
WHERE (SONumber = Items.SONumber) AND
((SONumber = 123) OR (SONumber = 124))
ORDER BY SONumber, ItemNumber
```

### Aplicando el siguiente archivo DAD

```
<element_node name="SalesOrder">
  <attribute_node name="Number">
    <column name="SONumber" />
  </attribute_node>
  <element_node name="CustomerNumber">
    <text_node>
      <column name="CustNumber" />
    </text_node>
  </element_node>
  <element_node name="Item" multi_occurrence="YES">
    <attribute_node name="Number">
      <column name="ItemNumber" />
    </attribute_node>
    <element_node name="PartNumber">
      <text_node>
        <column name="PartNumber" />
      </text_node>
    </element_node>
  </element_node>
</element_node>
```

### Obtendríamos el siguiente resultado

```
<?xml version="1.0"?>
<SalesOrder Number="123">
  <CustomerNumber>456</CustomerNumber>
  <Item Number="1">
    <PartNumber>XY-47</PartNumber>
  </Item>
  <Item Number="2">
    <PartNumber>B-987</PartNumber>
  </Item>
</SalesOrder>
```



## 3 – SGBD XML nativos

### **3.1 – Características básicas de un SGBD XML nativo y diferencias con un SGBD relacional**

Una base de datos XML nativa es aquella que:

- Define un modelo de datos XML: Definir que elementos son lógicamente significativos. Todos tendrán en cuenta los elementos, atributos, texto y orden en el documento, aunque es posible que de un sistema a otro los elementos reflejados en el modelo varíen
- Utiliza el documento como unidad mínima de almacenamiento
- Puede usar cualquier estrategia de almacenamiento: El almacenamiento físico de los documentos puede realizarse sobre un SGBD relacional tradicional, sobre ficheros con estructura propia o cualquier otro método.

La principal diferencia es que una base de datos XML nativa provee de su propio modelo de datos, mientras que un sistema XML Enabled tiene su propio modelo de datos y añade una capa de software que permite de alguna manera almacenar documentos XML y recuperar los datos generando nuevos documentos XML.

Podemos decir también que un SGBD XML-Enabled solo puede manejar y almacenar los documentos que encajan dentro del modelo definido para ellos, mientras que un SGBD XML nativo debe manejar todos los tipos de documentos posibles.

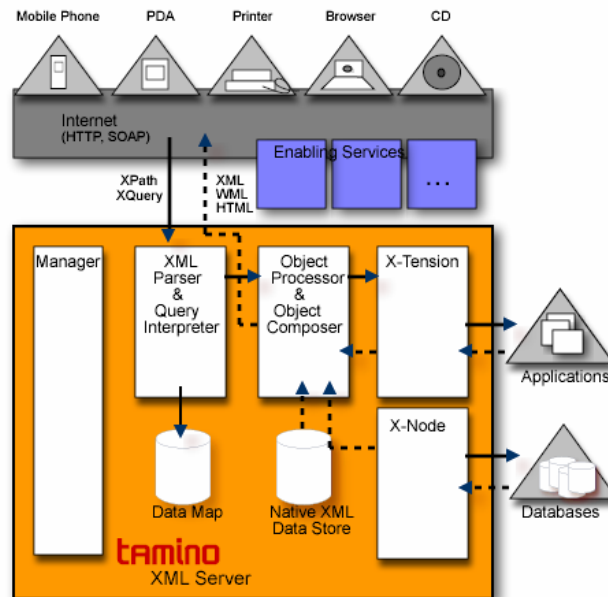
Una vez definido que es un SGBD XML nativo estudiaremos dos implementaciones distintas, una comercial y otra open source.

### **3.2 – Tamino**

Tamino es el SGBD nativo de la empresa SoftwareAG, es producto comercial de alto rendimiento y disponibilidad, además de ser uno de los primeros SGBD XML nativos disponibles.

#### **Arquitectura**

La arquitectura de Tamino tiene los siguientes componentes básicos



- Native XML Data Store + XML Engine: Es el componente central de la arquitectura, contiene el parser XML, el almacén nativo de datos y el intérprete de consultas.
- Data Map: Almacén de metadatos, contiene información sobre como validar esquemas, almacenar e indexar información, mapeo de estructuras...
- X-Node: Es el componente de dar acceso a base de datos externas, mapeando los datos a estructuras XML. El acceso a esta información es transparente para el usuario y se accede a ella cada vez que se necesita, es decir, que no es replicada.
- X-Tension: Permite la definición de funciones de usuario para ampliar las prestadas por Tamino
- Tamino Manager: Herramienta gráfica de gestión.

### Almacenamiento de documentos

Los documentos se almacenan en una base de datos propia y no se transforma en otro modelo. Existe un espacio separado para documentos y para índices.

Un doctype es el elemento raíz de un DTD o XML Schema, es decir, el elemento que define el comienzo y el final del documento.

La base de datos está estructurada en colecciones, una colección es un conjunto de documentos, de modo que es una estructura de árbol donde cada documento pertenece a una única colección.

Cada colección tiene asociado varios doctypes. El elemento raíz del documento XML define a que doctype estará asociado el documento, en caso de no poseer ninguno este se crea dinámicamente. Esto posibilita el almacenamiento de documentos sin formato definido.

La colección también tiene asociado un Schema con información tanto física como lógica de la colección. La parte lógica define las relaciones y propiedades de los documentos XML y la física contiene información sobre el almacenamiento e indexación de los mismos.

También se pueden almacenar documentos no-XML, para estos existe un doctype especial llamado nonXML.

El gestor asigna a cada documento un identificador, y el usuario puede asignarle un nombre, el cual debe de ser único dentro de cada doctype, y puede ser usado para acceder directamente al fichero a través de su URL.

Los elementos de configuración del sistema también son documentos XML almacenados en la colección system, por lo que pueden ser accedidos y manipulados por las herramientas estándar proporcionadas.

## Índices

Provee índices que son mantenidos automáticamente cuando los documentos son añadidos, borrados o modificados

- Simple text indexing: Indexa palabras dentro de elementos.
- Simple Estándar Indexing: Indexación por valor de los elementos.
- Structure Index: Mantiene todos los caminos posibles en los doctype
- Reference Index

Modificando el esquema de la colección se pueden definir distintos tipos de índices para optimizar consultas

- Compound Index
- Multipath Index
- Reference Index

## Acceso a los datos

El lenguaje de consulta de datos es XQuery, cuya implementación es XQuery 4; podemos ejecutar dichas sentencias desde las aplicaciones de gestión suministradas o desde programas, accediendo a la BB.DD. desde las librerías disponibles para Java (XMLDB), C, .NET, ActiveX y JScript.

Permite también la modificación de documentos, con operaciones de inserción, borrado, reemplazo y renombrado utilizando extensiones propias.

## Otras funcionalidades

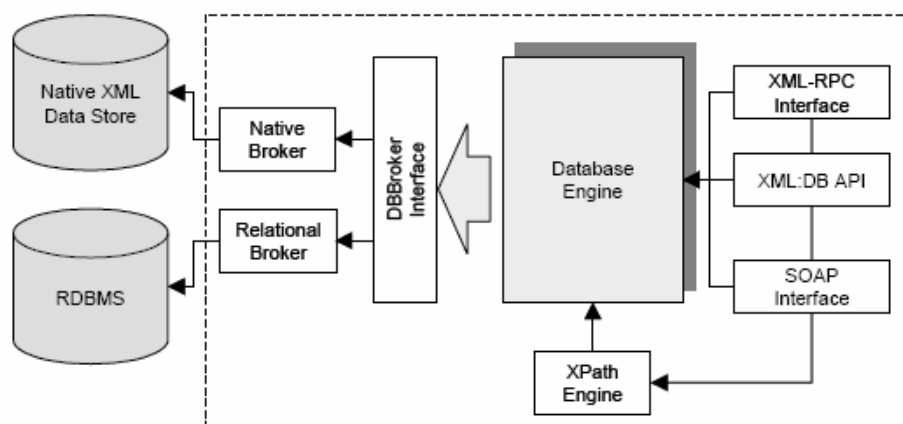
Es un SGBD completo, con todas las funcionalidades que se pueden pedir a un SGBD moderno, como soporte para transacciones, multiusuario, log de operaciones, herramientas para cargas masivas, sistema completo de backups, es un producto escalable y con buen rendimiento.

### 3.3 – eXist

Es un SGBD XML nativo open source que tiene las funcionalidades básicas de cualquier gestor nativo además de integrar algunas técnicas avanzadas como búsquedas de términos, búsquedas por proximidad de términos y búsquedas basadas en expresiones regulares.

## Arquitectura

La arquitectura de eXist posee los siguientes componentes



El motor de base de datos está completamente escrito en Java, posee distintos módulos para el almacenamiento de datos. En estos momentos el almacén principal está compuesto por una base de datos nativa además de incluir la posibilidad de almacenamiento sobre BB.DD. relacionales. En principio esta arquitectura oculta completamente la implementación física del motor de la base de datos.

Es destacable que el motor de base de datos es muy compacto y puede funcionar tanto en modo servidor, incrustado en una aplicación o en un contenedor J2EE (como Tomcat).

Como gestor XML nativo soporta los estándares de consulta XPath y XQuery además de extensiones propias para la actualización. Con el SGBD se dan aplicaciones que permiten ejecutar consultas directamente sobre la BB.DD.

### **Almacenamiento de documentos**

Los documentos se almacenan en colecciones, las cuales pueden estar anidadas; desde un punto de vista práctico el almacén de datos funciona como un sistema de ficheros. Cada documento está en una colección.

Los documentos no tienen que tener una DTD ó XML Schema asociado, y dentro de una colección pueden almacenarse documentos de cualquier tipo.

El almacén central nativo de datos es el fichero dom.dbx; es un fichero paginado donde se almacenan todos los nodos del documento de acuerdo al modelo DOM del W3C. Dentro del mismo archivo existe también un árbol B+ que asocia el identificador único del nodo con su posición física.

En el fichero 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

### **Índices**

El gestor automáticamente indexa todos los documentos utilizando índices numéricos para identificar los nodos del mismo (elementos, atributos, texto y comentarios).

Los índices están basados en árboles B+ y definidos a nivel de colección para mejorar el rendimiento. Durante la indexación se asigna un identificador único a cada documento de la colección, que es almacenado también junto al índice.

El almacén para el índice de elementos y atributos está en el fichero elements.dbx. Para ahorrar espacio los nombres de los nodos no son utilizados para construir el índice, en su lugar se asocia el nombre del elemento y de los atributos con unos identificadores numéricos en una tabla de nombres, y cada entrada del índice consiste en entradas con clave < id colección, id nombre> y una relación de valores de document-id y node-id que le corresponden.

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

También pueden definirse índices específicos para acelerar consultas,

## Acceso a datos

El SGBD puede funcionar de distintos modos

- Funcionando como servidor autónomo (ofreciendo servicios XML-RPC, WebDAV y REST).
- Insertado dentro de una aplicación Java.
- En un servidor J2EE, ofreciendo servicios XML-RPC, SOAP y WebDAV

Dependiendo del modo en que funcione el servidor podremos acceder de un modo u otro; el modo más común es a través del API XML:DB para Java, aunque está en proyecto la creación de un driver XQJ que dejará obsoleto este modo de acceso.

## Otras funcionalidades

Posee funciones de backup a través del cliente Java ó de la consola Unix, cuando se hace el backup se exporta el contenido de la base de datos como documentos XML estándar y se crea una jerarquía de directorios a imagen de la jerarquía de colecciones. Igualmente también se guarda información sobre configuración de índices, usuarios y un archivo especial llamado `__contents__.xml` con metadatos sobre el contenido (permisos, propietario, fechas de modificación...)

Tiene también funcionalidades de recuperación en caso de caída del sistema, deshaciendo transacciones incompletas y rehaciendo las terminadas pero no reflejadas, pero hay que decir que este soporte es solo para caídas de sistema y que no se puede acceder a soporte para transacciones desde las APIS.

## 4 – Conclusiones sobre SGBD

### 4.1 – Que tipo de SGBD utilizar.

Las bases de datos relacionales tienen un modelo probado y más estable, que permiten una gestión muy efectiva de la concurrencia y del soporte a las transacciones. Son capaces de almacenar datos de una manera muy efectiva y compacta.

Por el contrario el modelo relacional no encaja con la naturaleza jerárquica y semi-estructurada de los documentos XML (por ejemplo, no tiene en cuenta el orden de los datos en el documento); para almacenar los datos de este modo hace falta una capa de proceso y transformación que sobrecarga de trabajo al gestor, y no siempre es posible una conversión óptima.

También hay que resaltar que es posible que no se pueda reconstruir el documento original que generó los datos.

Las bases de datos XML son perfectas para almacenar documentos en los que tenemos que preservar el formato exacto del mismo.

Son capaces de manejar documentos sin estructurar; resaltar que el almacenamiento y recuperación de datos es directa e implementan índices específicos que facilitan la consulta del documento (incluso por partículas de texto) y recuperación, por lo que el rendimiento de consultas sobre este tipo de datos está más optimizado.

No hay una regla definitiva para decantarnos por uno u otro sistema, pero como regla general podemos decir que si se utiliza XML como medio de comunicación o intercambio de datos, las estructuras serán regulares y fácilmente transformables a un modelo relacional, por lo que en esta situación puede ser la elección más recomendable.

Si por el contrario nos encontramos con que la naturaleza de los documentos es de presentación de datos y su estructura es variable y compleja (por ejemplo XML como soporte para almacenamiento de libros) o si nos vemos en la necesidad de realizar búsquedas de texto complejas o de recuperar los documentos como se recibieron la elección de un SGBD XML nativo es la más recomendable, ya que nos aseguramos la conservación intacta del documento y una alta efectividad en el manejo y consulta de los datos textuales.

### 4.2 – Próximos pasos en el desarrollo de SGBDs

Aunque existen SGBD XML nativos desde hace algún tiempo en el mercado, es una tecnología que aun no ha alcanzado todo su potencial de desarrollo.

El grado de madurez dentro de la industria es bajo (por ejemplo, los estándares del lenguaje de consulta XQuery 1.0 están publicados en el año 2006 y no se

dispone de un lenguaje de actualización unificado). Existen interfaces estándar para acceso a BB.DD relacionales, pero aún no existe dicho estándar para acceso a SGBD XML nativos (aunque está la definición XMM:DB y hay iniciativas en este aspecto con los drivers XQJ <http://www.jcp.org/en/jsr/detail?id=225> )

Después de ver las capacidades de distintos gestores parece que el destino de estas dos tecnologías y modelos de almacenamiento han tomado un camino convergente: los gestores relacionales comienzan a tener capacidades de almacenamiento XML nativo muy desarrolladas y proporcionan soporte para lenguajes como XQuery. Por su parte los SGBD XML nativos permiten el mapeo de datos en tablas relacionales en documentos XML.

Respecto a los lenguajes de consulta de bases de datos es el estándar XQuery 1.0, de reciente publicación; para bases de datos relacionales tradicionales existe el estándar SQL XML donde se recogen extensiones de SQL para el tratamiento de XML. En principio estas tecnologías no compiten entre ellas sino que son complementarias, y se están realizando trabajos para la integración de los dos lenguajes.

El objetivo es una integración del modelo relacional y modelo XML para poder acceder indistintamente a tablas o documentos de manera transparente para el usuario, es decir, poder utilizar indistintamente SQL ó XQuery sobre tablas o documentos XML y poder recuperar los datos de manera transparente.

### **4.3 – SGBD XML seleccionado**

Hemos visto características de SGBD relacionales XML-enabled así como SGBD XML nativos,

Si bien es cierto que existen versiones de libre distribución de algunos productos comerciales (por ejemplo Oracle Express Edition) que pueden ser usados para la realización de la parte práctica, ya que incluyen posibilidad de almacenamiento nativo de documentos XML nos decantaremos por una base de datos XML nativa pura.

Sobre los dos SGBD XML nativos revisados TAMINO es un producto comercial del cual no se puede conseguir una licencia para uso académico ni de pruebas por un tiempo aceptable.

El otro gestor nativo visto, eXist, es un producto open source de libre distribución; puede usarse sin restricciones y es de fácil acceso, y como añadido da soporte a todas las tecnologías básicas que son el objetivo del presente trabajo, por lo que es el producto seleccionado.

Para las instrucciones de instalación del producto ir al Anexo 1.



## 5 – XPath 1.0

### **5.1 – Que es XPath.**

XPath es un lenguaje pensado para la selección de elementos de un documento XML para su proceso.

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

En su versión 1.0 fue diseñado para su uso con XSLT y XPointer y su objetivo era el tratamiento de documentos. Puede obtenerse la especificación completa de XPath 1.0 en

<http://www.w3.org/TR/xpath>

### **5.2 – Modelo de datos.**

El modelo de datos es la representación que XPath tiene de un documento XML, es decir, las partes del mismo que son importantes para el. Para XPath un documento XML es un árbol de nodos, casi todas las partes del documento están representadas dentro de este modelo.

#### **Nodos**

Como hemos dicho antes, un nodo es una representación lógica de una parte de un documento XML. En XPath existen 7 tipos de nodos:

- Root node
- Element node
- Attribute node
- Text node
- Namespace node
- Comment node
- Processing instruction node

#### *Root node:*

Es el documento en si mismo. El elemento principal del documento es hijo del root node.

### *Element node*

Cada parte del documento está representado por un nodo element. El nombre está formado por el URI del namespace del elemento y el nombre del mismo, aunque también se puede trabajar con el QName, que es el prefijo del namespace del elemento y el nombre local.

`Prefijo:nombrelocal`

El valor textual de un elemento es la concatenación de text nodes descendientes.

Posee un identificador único.

### *Attribute node*

Cada atributo del documento está representado por un nodo atributo. El elemento al que está asociado dicho atributo es su parent node.

Estos atributos tienen nombre y valor textual.

### *Text node*

Representa el contenido textual de un elemento, evidentemente tiene valor de cadena pero no nombre.

### *Namespace node*

Para cada namespace definidos, existen namespace nodes; cada elemento tendrá sus nodos namespace.

### *Comment node*

Representa un comentario, excepto las que ocurren dentro del tipo de documento.

### *Processing Instruction node*

Representa una processing instruction dentro del documento, excepto las que ocurren dentro del tipo de documento.

## **Expresiones**

El formato de las expresiones en XPath no sigue el estándar XML.

Existen los siguientes tipos de expresiones

- Basics: Literales
- Llamadas a funciones
- Node-sets (nodos seleccionados por la expresión)
- Booleanas
- Numéricas
- Cadenas

Las expresiones son evaluadas, y pueden devolver 4 tipos distintos de resultados

- Boolean
- Node-Set
- Number
- String

#### *Boolean*

Los valores de expresiones booleanas se escriben como true() ó false(), debido a que estos dos nombre pueden ser nombre válidos de etiquetas

#### *Node-Set*

Es un conjunto de nodos, técnicamente desordenados (aunque tendrá el orden en que aparezcan en el documento) y que no contiene duplicados.

#### *Number*

Los número son siempre en punto flotante, no hay manera de representar número sencillos.

#### *String*

Es una secuencia de caracteres Unicode.

Estas expresiones se evaluarán respecto a un contexto, es decir, la misma sentencia XPath ejecutada en contextos distintos devolverá resultados distintos Podemos asimilar el contexto al nodo donde el procesador XPath está situado en el momento de ejecutar la sentencia. El contexto viene definido principalmente por

- Context-node, o nodo de referencia sobre el que se lanza la expresión
- Context size, número de elementos hermanos del context-node.
- Context position, número que indica la posición del context-node dentro de sus hermanos.

Al consultar un documento, el contexto generalmente será el nodo raíz del mismo, o al consultar una base de datos una colección, pero cuando se usa junto con XSLT el contexto de la consulta variará en función del nodo tratado en cada momento.

## Funciones

La lista de funciones que componen el núcleo de la especificación XPath se dividen en los siguientes bloques

- Boolean
- Node-Set
- Numeric
- String

### Boolean

boolean(arg)	<p>Convierte el argumento a boolean, siguiendo estas reglas</p> <ul style="list-style-type: none"> <li>• Si el argumento es un boolean devuelve su valor.</li> <li>• Un node-set devuelve false si está vacío, en caso contrario devuelve true.</li> <li>• Un string devuelve true si su longitud es distinta de cero.</li> <li>• Un numérico devolverá true si el argumento es positivo ó negativo, false si es cero ó NaN.</li> <li>• Si no es ninguno de estos tipos dependerá del tipo en particular.</li> </ul>
not(arg)	Devuelve true si el argumento es false, false si el argumento es true.
true()	Devuelve true.
false()	Devuelve false.
lang()	<p>Devuelve true si en el nodo de contexto está especificado el atributo xml:lang (&lt;para xml:lang="en"/&gt;) y el valor de este coincide con el valor del string que se le da como parámetro, false en caso contrario.</p> <p>Si el nodo de contexto es el especificado como ejemplo, lang("en") devuelve true.</p>

### Node-Set

number count(node-set)	Devuelve el número de elementos del node-set.
id(arg)	El objeto que se pasa como parámetro se convierte a string, y se devuelven todos los elementos con in parámetro ID igual que el argumento
number last()	Devuelve un número que contiene el tamaño del contexto.
string local-name(node-set)	Devuelve la parte local del nombre expandido del primer elemento del node-set.
string name(node-set)	Devuelve el QName del primer elemento del node-set.
string namespace-uri(node-set)	Devuelve el URI del nombre expandido, si no existe, si es nulo o el node-set está lacio retorna una cadena vacía
number position()	Devuelve la posición del elemento de contexto dentro del contexto.

### Numeric

number(arg)	<p>Convierte el argumento a numérico</p> <ul style="list-style-type: none"> <li>• Si es una cadena con un número válido devuelve el valor, en caso contrario devuelve NaN</li> <li>• Si es un boolean devuelve 1 si es true, 0 si es false</li> <li>• Si es un node set este es convertido a string y después a numérico igual que si el parámetro fuese una cadena</li> <li>• Si no es ninguno de estos tipos dependerá del tipo en particular.</li> </ul>
sum(node-set)	Devuelve el valor de la suma de todos los elementos del node-set convertido a numérico.
floor(number)	Devuelve el entero menor más cercano al parámetro
ceiling(number)	Devuelve el entero mayor más cercano al parámetro
round(number)	Devuelve el entero más cercano al parámetro

### String

string string(object)	<p>Convierte el objeto pasado como parámetro a cadena de caracteres.</p> <ul style="list-style-type: none"> <li>• Un node-set devuelve el valor de convertir a string el primer elemento.</li> <li>• Un numérico <ul style="list-style-type: none"> <li>o NaN devuelve NaN</li> <li>o Valor cero devuelve 0</li> <li>o El valor infinito es convertido a (-)Infinity</li> <li>o Si es un valor entero devuelve (-)valor si número decimales</li> <li>o Cualquier otro caso devuelve (-)valor con separador decimal y al menos un decimal</li> </ul> </li> <li>• Booleano devuelve 'true' ó 'false'.</li> <li>• Si no es ninguno de estos tipos dependerá del tipo en particular.</li> </ul>
string concat(string, string, string*)	Devuelve la concatenación de todos sus argumentos
boolean starts-with(string, string)	Devuelve true si la primera cadena comienza con la segunda cadena, false en caso contrario.
boolean contains(string, string)	Devuelve true si la primera cadena contiene la segunda cadena, false en caso contrario.
string substring-before(string, string)	Devuelve una subcadena formada por los caracteres de la cadena de parámetro hasta que se encuentra la primera ocurrencia de la segunda cadena.
string substring-after(string, string)	Devuelve una subcadena formada por los caracteres de la cadena de parámetro desde que se encuentra la primera ocurrencia de la segunda cadena hasta el final.

string substring(string, number, number?)	Devuelve una cadena compuesta por los caracteres del primer argumento, comenzando en la posición que indique el Segundo argumento hasta el final, si hay tercer argumento indica la longitud de la cadena a devolver
number string-length(string?)	Devuelve la longitud en caracteres del argumento. Si no hay parámetro se convierte a cadena el nodo de contexto y se aplica la función.
string normalize-space(string?)	Devuelve la cadena del argumento normalizada, eliminando espacios en blanco iniciales y finales y reduciendo secuencias de espacios a uno solo. Si no hay parámetro se convierte a cadena el nodo de contexto y se aplica la función.
string translate(string, string, string)	Devuelve una cadena que es el resultado de sustituir caracteres en el primer parámetro. Esta sustitución viene indicada por los parámetros 2 y 3, de modo que si en la cadena 1 encontramos un carácter de la cadena 2 se sustituye por el mismo carácter de la cadena 3 situado en la misma posición, esto se ve mejor con un ejemplo, translate("bar", "abc", "ABC") devuelve "BAr"

### **5.3 – Como se expresan las selecciones de nodos.**

La selección de nodos la vamos a realizar mediante el location-path, con esta expresión indicamos los nodos que queremos seleccionar del documento.

#### **5.3.1 – Rutas relativas y absolutas**

Hemos comentado con anterioridad que las expresiones se evalúan respecto a un contexto, y que el contexto es importante ya que puede definir los resultados que obtenemos con la expresión; cuando ejecutamos una sentencia XPath tenemos dos maneras de especificar el contexto de la expresión.

- Relativa: La selección se realizará evaluando desde el nodo de contexto actual.
- Absoluta: De este modo indicamos que la evaluación de la expresión se realiza desde el nodo raíz, la forma de indicarlo es comenzando la expresión con el carácter '/'

#### **5.3.2 – Composición del location-path**

Con esta expresión indicamos los nodos que queremos seleccionar del documento. Debido a la naturaleza jerárquica de un documento XML lo que haremos será ir especificando para cada nivel del árbol que nodos queremos seleccionar, por tanto un location-path contendrá distintos pasos para cada nivel del árbol, los cuales se llaman location step. Los location step están separados entre si por el carácter '/'.

El formato del location path es:

```
{/} location-step / location-step /location-step...
```

Cada location step tiene un formato definido y está potencialmente compuesto por tres partes distintas

```
Axis::node-test [predicado]
```

**Axis:** Especificamos el ‘camino’ para acceder a los nodos que necesitamos, igual que al dar una dirección de una calle decimos ‘en la segunda calle a la derecha’ con el eje especificamos la dirección que debemos considerar dentro del árbol (padre, hijo, atributos...), veremos las opciones disponibles un poco más adelante.

**Node-test:** Dentro del camino especificado especificamos que tipo de nodos queremos seleccionar. Cada Axis tiene un tipo principal de nodo, para los atributos el tipo es attribute, para los namespaces es tipo es namespace y para el resto el tipo es element. Podemos filtrar de las siguientes maneras

- QName: Solo evalúa a true si el literal especificado es igual al nombre del nodo (es decir, filtramos por el nombre de los elementos)
- Tipo de nodo: Podemos usar los valores `text()`, `comment()`, `processing-instruction()`, `node()`
- \*: Selecciona todos los nodos

**Predicado:** De todos los nodos seleccionados podemos filtrar en función de la condición que indiquemos en el predicado. El predicado es una expresión que se evalúa a boolean, y si es true devuelve el nodo.

- Si el resultado de la expresión es un número se evalúa a true si corresponde con la posición del nodo dentro del contexto, a false en caso contrario.
- Si el resultado de la expresión no es un número es convertido a boolean usando la función `boolean()` (ver apartado anterior)

Se pueden concatenar más de un predicado

### *Relación de ejes (Axis) disponibles*

child	Hijos del nodo actual.
ancestor	Todos los ancestros del nodo actual hasta llegar al nodo raíz
ancestor-or-self	Todos los ancestros del nodo actual hasta llegar al nodo raíz y el mismo nodo actual
attribute	Incluye solamente a los atributos
descendant	Hijos del nodo actual, sin importar el grado de profundidad
descendant-or-self	Hijos del nodo actual, sin importar el grado de profundidad y el mismo nodo actual

following	Todos los nodos descendientes del nodo raíz que se encuentran después de la posición del nodo actual (según el orden del documento) y no son descendientes del mismo
following-sibling	Nodos que son hijos del nodo padre del nodo actual y que le siguen en el orden del documento
namespace	Contiene los nodos de namespace del nodo
parent	Padre del nodo
preceding	Todos los nodos descendientes del raíz que se encuentran antes de la posición del nodo actual (según el orden del documento) y no son ancestros del mismo
preceding-sibling	Nodos que son hijos del nodo padre y que se encuentran anteriores al nodo actual
self	Es el nodo actual

### 5.3.3 – Escritura abreviada de expresiones.

Es posible la escritura abreviada de determinadas partes de una sentencia XPath, esto hace que sean más ágiles de escribir y más manejables. Las posibles abreviaturas que son

`child::` puede ser omitido, ya que es el axis por defecto, estas dos expresiones son equivalentes

```

/child::div/child::para
/div/para

```

`attribute::nombre` puede ser sustituido por '@'

```

child::para[attribute::type="warning"]
para[@type="warning"]

```

`self::node` puede ser sustituido por '.'

```

self::node()/child::para
./para

```

`parent::node()` puede ser sustituido por '..'

```

parent::node()/child::title
../title

```

`descendant-or-self::node()` puede ser sustituido por '//'

```

/descendant-or-self::node()/child::para
//para

```



#### **5.4 – Ejemplos.**

Después de la descripción y formato de las funcionalidades veamos aquí una serie de ejemplos que ilustrarán mejor el funcionamiento.

Para poder ejecutarlas deberemos cargar el siguiente juego de pruebas en una colección en la base de datos. Para ver como definir una colección nueva mirar el Anexo 2; para ver como realizar la carga de un documento en una colección mirar el Anexo 3.

*(\*) Resaltar que para el correcto funcionamiento de los ejemplos es necesario que el contexto en el 'Query Dialog' esté situado en la colección donde hemos cargado los datos. Para ver el funcionamiento del Query Dialog ir al Anexo 4*

Igualmente hay ejemplos donde se ha expuesto varias maneras de realizar la misma consulta, utilizando notaciones completas y abreviadas

```
<?xml version="1.0" encoding="UTF-8"?>
<envio>
  <poliza>
    <numero>99000000</numero>
    <tomador>Manuel Gonzalez Sanchez</tomador>
    <asegurado nombre="Carlos" apellidos="Sanchez">
      <garantia vigor="S">
        <tipo>Accidentes</tipo>
        <capital>10000</capital>
      </garantia>
      <garantia vigor="S">
        <tipo>Dental</tipo>
        <capital>500</capital>
      </garantia>
    </asegurado>
    <asegurado nombre="Juan" apellidos="López">
      <garantia vigor="N">
        <tipo>Vida</tipo>
        <capital>60000</capital>
      </garantia>
    </asegurado>
  </poliza>
  <poliza externa="S">
    <numero>99000001</numero>
    <tomador>Pedro Martin Gomez</tomador>
    <asegurado nombre="Pedro" apellidos="Martin">
      <garantia vigor="S">
        <tipo>Vida</tipo>
        <capital>80000</capital>
      </garantia>
    </asegurado>
  </poliza>
  <poliza>
    <numero>99000002</numero>
    <tomador>Alfredo Salas Perez</tomador>
    <asegurado nombre="Lucas" apellidos="Montero">
      <garantia vigor="S">
        <tipo>Vida</tipo>
        <capital>90000</capital>
      </garantia>
    </asegurado>
    <asegurado nombre="Carmen" apellidos="Sanchez">
    </asegurado>
    <asegurado nombre="Maria" apellidos="Fernandez">
      <garantia vigor="S">
        <tipo>Vida</tipo>
        <capital>90000</capital>
      </garantia>
      <garantia vigor="N">
        <tipo>Accidentes</tipo>
        <capital>4000</capital>
      </garantia>
      <garantia vigor="S">
        <tipo>Dental</tipo>
        <capital>300</capital>
      </garantia>
    </asegurado>
  </poliza>
</envio>
```

**Ejemplo 1**

<b>Objetivo</b>	<b>Seleccionar los tomadores de todas las pólizas</b>
<b>Sentencia</b>	<code>/envio/poliza/tomador</code> <code>/child::envio/child::poliza/child::tomador</code>
<b>Resultado</b>	<code>&lt;tomador&gt;Manuel Gonzalez Sanchez&lt;/tomador&gt;</code> <code>&lt;tomador&gt;Pedro Martin Gomez&lt;/tomador&gt;</code> <code>&lt;tomador&gt;Alfredo Salas Perez&lt;/tomador&gt;</code>

**Ejemplo 2**

<b>Objetivo</b>	<b>Obtener el número total de pólizas</b>
<b>Sentencia</b>	<code>count(/envio/poliza)</code> <code>count(//poliza)</code> <code>count(/descendant-or-self::node()/poliza)</code>
<b>Resultado</b>	3

**Ejemplo 3**

<b>Objetivo</b>	<b>Obtener el primer asegurado de cada póliza</b>
<b>Sentencia</b>	<code>/envio/poliza/asegurado[1]</code>
<b>Resultado</b>	<code>&lt;asegurado nombre="Carlos" apellidos="Sanchez"&gt;</code> <code>&lt;garantia vigor="S"&gt;</code> <code>&lt;tipo&gt;Accidentes&lt;/tipo&gt;</code> <code>&lt;capital&gt;10000&lt;/capital&gt;</code> <code>&lt;/garantia&gt;</code> <code>&lt;garantia vigor="S"&gt;</code> <code>&lt;tipo&gt;Dental&lt;/tipo&gt;</code> <code>&lt;capital&gt;500&lt;/capital&gt;</code> <code>&lt;/garantia&gt;</code> <code>&lt;/asegurado&gt;</code> <code>&lt;asegurado nombre="Pedro" apellidos="Martin"&gt;</code> <code>&lt;garantia vigor="S"&gt;</code> <code>&lt;tipo&gt;Vida&lt;/tipo&gt;</code> <code>&lt;capital&gt;80000&lt;/capital&gt;</code> <code>&lt;/garantia&gt;</code> <code>&lt;/asegurado&gt;</code> <code>&lt;asegurado nombre="Lucas" apellidos="Montero"&gt;</code> <code>&lt;garantia vigor="S"&gt;</code> <code>&lt;tipo&gt;Vida&lt;/tipo&gt;</code> <code>&lt;capital&gt;90000&lt;/capital&gt;</code> <code>&lt;/garantia&gt;</code> <code>&lt;/asegurado&gt;</code>

**Ejemplo 4**

<b>Objetivo</b>	Seleccionar los primeros asegurados de las pólizas solo si tienen más de dos garantías contratadas
<b>Sentencia</b>	/envio/poliza/asegurado[1] [count(garantia)>1]
<b>Resultado</b>	<pre> &lt;asegurado nombre="Carlos" apellidos="Sanchez"&gt;   &lt;garantia vigor="S"&gt;     &lt;tipo&gt;Accidentes&lt;/tipo&gt;     &lt;capital&gt;10000&lt;/capital&gt;   &lt;/garantia&gt;   &lt;garantia vigor="S"&gt;     &lt;tipo&gt;Dental&lt;/tipo&gt;     &lt;capital&gt;500&lt;/capital&gt;   &lt;/garantia&gt; &lt;/asegurado&gt; </pre>

**Ejemplo 5**

<b>Objetivo</b>	Seleccionar los asegurados que tengan contratada la garantía dental
<b>Sentencia</b>	/envio/poliza/asegurado[garantia[tipo="Dental"]]
<b>Resultado</b>	<pre> &lt;asegurado nombre="Carlos" apellidos="Sanchez"&gt;   &lt;garantia vigor="S"&gt;     &lt;tipo&gt;Accidentes&lt;/tipo&gt;     &lt;capital&gt;10000&lt;/capital&gt;   &lt;/garantia&gt;   &lt;garantia vigor="S"&gt;     &lt;tipo&gt;Dental&lt;/tipo&gt;     &lt;capital&gt;500&lt;/capital&gt;   &lt;/garantia&gt; &lt;/asegurado&gt; &lt;asegurado nombre="Maria" apellidos="Fernandez"&gt;   &lt;garantia vigor="S"&gt;     &lt;tipo&gt;Vida&lt;/tipo&gt;     &lt;capital&gt;90000&lt;/capital&gt;   &lt;/garantia&gt;   &lt;garantia vigor="N"&gt;     &lt;tipo&gt;Accidentes&lt;/tipo&gt;     &lt;capital&gt;4000&lt;/capital&gt;   &lt;/garantia&gt;   &lt;garantia vigor="S"&gt;     &lt;tipo&gt;Dental&lt;/tipo&gt;     &lt;capital&gt;300&lt;/capital&gt;   &lt;/garantia&gt; &lt;/asegurado&gt; </pre>

**Ejemplo 6**

<b>Objetivo</b>	Seleccionar las pólizas con el indicador externa
<b>Sentencia</b>	/envio/poliza[@externa]
<b>Resultado</b>	<pre> &lt;poliza externa="S"&gt;   &lt;numero&gt;99000001&lt;/numero&gt;   &lt;tomador&gt;Pedro Martin Gomez&lt;/tomador&gt;   &lt;asegurado nombre="Pedro" apellidos="Martin"&gt;     &lt;garantia vigor="S"&gt;       &lt;tipo&gt;Vida&lt;/tipo&gt;       &lt;capital&gt;80000&lt;/capital&gt;     &lt;/garantia&gt;   &lt;/asegurado&gt; &lt;/poliza&gt; </pre>

**Ejemplo 7**

<b>Objetivo</b>	Seleccionar todas las garantías que tienen el indicador vigor con valor 'N'
<b>Sentencia</b>	<code>//garantia[@vigor='N'] /descendant-or-self::node()/garantia[@vigor='N']</code>
<b>Resultado</b>	<code>&lt;garantia vigor="N"&gt;   &lt;tipo&gt;Vida&lt;/tipo&gt;   &lt;capital&gt;60000&lt;/capital&gt; &lt;/garantia&gt; &lt;garantia vigor="N"&gt;   &lt;tipo&gt;Accidentes&lt;/tipo&gt;   &lt;capital&gt;4000&lt;/capital&gt; &lt;/garantia&gt;</code>

**Ejemplo 8**

<b>Objetivo</b>	Seleccionar todos los capitales de garantías que tienen el indicador vigor = 'S'
<b>Sentencia</b>	<code>//capital[ancestor::garantia[@vigor='S']]</code>
<b>Resultado</b>	<code>&lt;capital&gt;10000&lt;/capital&gt; &lt;capital&gt;500&lt;/capital&gt; &lt;capital&gt;80000&lt;/capital&gt; &lt;capital&gt;90000&lt;/capital&gt; &lt;capital&gt;90000&lt;/capital&gt; &lt;capital&gt;300&lt;/capital&gt;</code>

**Ejemplo 9**

<b>Objetivo</b>	Seleccionar los tomadores de las pólizas, solo si la póliza tiene dos o más asegurados
<b>Sentencia</b>	<code>//tomador[count(following-sibling::asegurado)&gt;=2]</code>
<b>Resultado</b>	<code>&lt;tomador&gt;Manuel Gonzalez Sanchez&lt;/tomador&gt; &lt;tomador&gt;Alfredo Salas Perez&lt;/tomador&gt;</code>

**Ejemplo 10**

<b>Objetivo</b>	Seleccionar todos los asegurados cuyo nombre empieza por la letra 'J'
<b>Sentencia</b>	<code>/envio/poliza/asegurado[substring(@nombre,1,1)='J'] //asegurado[substring(@nombre,1,1)='J']</code>
<b>Resultado</b>	<code>&lt;asegurado nombre="Juan" apellidos="López"&gt;   &lt;garantia vigor="N"&gt;     &lt;tipo&gt;Vida&lt;/tipo&gt;     &lt;capital&gt;60000&lt;/capital&gt;   &lt;/garantia&gt; &lt;/asegurado&gt;</code>

**Ejemplo 11**

<b>Objetivo</b>	Obtener la suma de capitales de las garantías en vigor
<b>Sentencia</b>	<code>sum(//capital[parent::garantia[@vigor = 'S']])</code>
<b>Resultado</b>	270800

**Ejemplo 12**

<b>Objetivo</b>	Seleccionar las pólizas con alguna garantía cuyo capital sea de 80000 euros
<b>Sentencia</b>	/envio/poliza[count(asegurado/garantia[capital=80000])>0]
<b>Resultado</b>	<pre> &lt;poliza externa="S"&gt;   &lt;numero&gt;99000001&lt;/numero&gt;   &lt;tomador&gt;Pedro Martin Gomez&lt;/tomador&gt;   &lt;asegurado nombre="Pedro" apellidos="Martin"&gt;     &lt;garantia vigor="S"&gt;       &lt;tipo&gt;Vida&lt;/tipo&gt;       &lt;capital&gt;80000&lt;/capital&gt;     &lt;/garantia&gt;   &lt;/asegurado&gt; &lt;/poliza&gt; </pre>

## 6 – XPath 2.0.

XPath 2.0 es una evolución significativa de XPath 1.0. El objetivo sigue siendo la selección de nodos de un árbol origen y la creación de un árbol destino con dichos nodos (aquí no se incluye la creación de nuevos nodos).

Si XPath 1.0 estaba más orientado al procesamiento de documentos junto con XSLT, XPath 2.0 se diseñó conjuntamente con XQuery 1.0 y XSLT 2.0 orientándolo a la consulta de información en bases de datos y documentos.

Como consecuencia del diseño conjunto, el modelo de datos es compartido por XPath 2.0 y XQuery 1.0, además de mantener compatibilidad hacia atrás, es decir, una sentencia XPath 1.0 es una sentencia XPath 2.0 válida.

Hay que resaltar que XPath 2.0 es un subconjunto de XQuery, por lo que según avanzamos en el conocimiento de XPath 2.0 también aprendemos XQuery.

La especificación XPath 2.0 es significativamente mayor que la 1.0, (la primera estaba formada por un único documento, mientras que la segunda está formada por varios documentos con una extensión muy superior), en este documento trataremos los aspectos más importantes del lenguaje. La especificación completa de XPath 2.0 puede consultarse en:

<http://www.w3.org/TR/xpath20/>

### **6.1 – Diferencias con XPath 1.0.**

Enumeramos las diferencias más significativas.

- Soporte para los tipos primitivos de datos XML Schema
- Se reemplaza el concepto de node-set por el de secuencia.
- Posibilidad de funciones lógicas en expresiones
- Se define una función para el tratamiento de secuencias: for
- Se define una función para el tratamiento condicionales: if
- Se añaden los cuantificadores existenciales y universales (some y every)
- Operadores de combinación de secuencias: intersect, union, except
- Posibilidad de incluir comentarios

### **6.2 – Modelo de datos**

La definición completa del modelo de datos de puede encontrar en

<http://www.w3.org/TR/xpath-datamodel/>

El modelo de datos está basado en la especificación “InfoSet”, pero teniendo en cuenta los requisitos necesarios para XQuery 1.0/XPath 2.0, es decir, poseen un modelo de datos compartido

- Soporte para los tipos definidos en XML Schema
- Representación de colecciones de documentos y valores complejos
- Valores atómicos formateados
- Soporte para secuencias ordenadas

Igual que en XPath 1.0, el árbol de datos un documento XML está compuesto de nodos. Todos los nodos tiene una identidad única, los valores atómicos no tienen identidad.

El orden de los nodos del documento es estable y no cambia durante la ejecución de una consulta.

- El primer nodo es el nodo raíz.
- Un nodo aparece siempre antes que sus hijos y descendientes
- Los nodos namespace siguen inmediatamente al nodo del que dependen.
- Los nodos attribute siguen inmediatamente al nodo namespace del elemento, si no hay nodo namespace, sigue inmediatamente al elemento
- Nodos children y descendant aparecen antes que los following-sibling
- El orden de los hermanos es el orden en el que ocurran en el padre

## Contexto

El concepto de contexto engloba todo aquello que puede afectar al resultado de la expresión; podemos distinguir los elementos que afectan a la expresión en dos grupos

- Contexto estático: Engloba toda la información que podemos obtener con el análisis de la sentencia
  - Namespaces definidos
  - Namespace por defecto para elementos, tipos, funciones
  - Definición de esquemas
  - Definición de variables
  - Signatura de funciones
  - Colecciones y documentos disponibles.
- Contexto dinámico: Toda la información disponible en el tiempo de ejecución de la sentencia
  - Context ítem: Elemento procesado en un momento dado
  - Context position: Posición del context ítem en la secuencia procesada
  - Context size: Tamaño de la secuencia donde se encuentra el context ítem
  - Valores reales de las variables definidas
  - Implementación de las funciones
  - Fecha y hora actual
  - Documentos y colecciones realmente disponibles



## Expresiones

Se amplían el tipo de expresiones permitidas, resumiendo podemos encontrarnos expresiones, las cuales iremos viendo en puntos siguientes

- Básicas: Literales, referencias a variables, llamadas a funciones
- Expresiones de localización: Location path
- Secuencias y expresiones sobre secuencias
- Expresiones aritméticas
- Tratamiento iterativo FOR
- Expresiones condicionales IF/THEN/ELSE

## Secuencias

Una secuencia es un conjunto ordenado de elementos en la que pueden aparecer elementos repetidos, es decir, no es igual la secuencia  $\{/a/b, /d/e\}$  que  $\{/d/e, /a/b\}$

En XPath 2.0 todo son secuencias, no se puede decir una expresión devuelve un valor numérico, sino una secuencia que contiene un valor numérico. Las secuencias no pueden ser anidadas, si añadimos una secuencia a otra secuencia, por ejemplo (1, 2,3, (4,5) ,6) lo que tenemos realmente es (1, 2, 3, 4, 5, 6)

Se permite la construcción de secuencias; la secuencia está delimitada por los caracteres ( ) y cada uno de los valores están separados por el carácter ','. Algunos ejemplos de secuencias válidas

- Los números del 1 al 4: (1, 2, 3, 4)
- Todos los nodos A y B del nodo de contexto: (A, B)
- Usando variables, si cantidad tiene valor 7: (1, \$cantidad) = (1,7)

Podemos especificar secuencia de enteros ascendente usando el operador to. Las siguientes secuencias son iguales

- (1,5 to 10,20)
- (1,5,6,7,8,9,10,20)

Si en alguna expresión debemos hacer referencia al tipo de secuencia (por ejemplo con el operador instance of) la forma de expresarlo es la siguiente

```
----empty-sequence()-----
|--tipo--indicador ocurrencia--|
```

Tipo indica el contenido de la secuencia, puede tener ser de los siguientes tipos

```
element(-----)
  |----nombre-----|
  |--*-----| |--(tipo)---|
```

```

attribute (-----)
    |---nombre-----|
    |--*-----|    |--(tipo)-|

schema-element ('nombre')

schema-attribute ('nombre')

processing-instruction (---NCName----)
    |-literal-|

comment ()

text ()

node ()

```

Y el indicador de ocurrencia puede tomar los siguientes valores

- \* : cero o mas ocurrencias
- ? : cero o una ocurrencia
- + : una o más ocurrencias

Por ejemplo, con `node()*` nos referimos a una secuencia de cualquier número de nodos, con `attribute()?` Nos referimos a un atributo opcional. En los ejemplo 8 y 9 puede verse el uso de los operadores sobre secuencia.

## Tipos de datos

Se soportan los 19 tipos de datos básicos definidos en la especificación XML Schema ( <http://www.w3.org/TR/xmlschema-2/#built-in-primitive-datatypes> ), pero se le añaden 5 tipos más

- `xs:untyped` – Para un tipo de dato que no ha sido validado
- `xs:untypedAtomic` – Para un tipo de dato atómico no validado
- `xs:anyAtomicType` – Para cualquier tipo atómico, es la base de todos los tipos de datos atómicos.
- `xs:dayTimeDuration` – Derivado de `xs:duration` pero restringido a días, horas, minutos y segundos
- `xs:yearMonthDuration` – Derivado de `xs:duration` pero limitado al año, mes.

## Nodos

Los tipos de nodos definidos son los mismos definidos en la especificación XPath 1.0 (ver apartado 5.2), se ha añadido el tipo de nodo `document-node` que sustituye al tipo de nodo `root`. El tipo de nodo `namespace` se mantiene por compatibilidad, pero cae en desuso. Si se necesita información acerca del namespace de un elemento debe recurrirse a las funciones `in-scope-prefixes` y `namespace-uri-for-prefix`.

## Funciones

El núcleo de funciones XPath 1.0 es un conjunto reducido y se enumeró en el apartado anterior. El conjunto de funciones definidas para XQuery/XPath 2.0 es extenso y sobrepasa los límites de este documento enumerarlas todas, pero puede tenerse una referencia detallada de las mismas en

<http://www.w3.org/TR/xpath-functions/>

### **6.3 – Funcionalidades añadidas**

#### **6.3.1 – Funciones de entrada**

Las funciones de entrada dan acceso a los datos a procesar; tienen importancia porque con ellas y desde la misma expresión podemos decidir que datos queremos procesar.

Existen dos funciones de entrada

- `doc(arg)`: tratamiento del documento especificado por `arg`, si no lo encuentra se genera un error
- `collection(arg)`: tratamiento de una colección, si no encuentra la colección especificada se generará un error. En caso de no especificarla se entenderá que es la colección por defecto.

#### **6.3.2 – Tratamiento iterativo**

Una sentencia FOR consta de tres partes

- Range variable: variable sobre la que se almacenan los nodos
- Binding sequence: nodos que se van a tratar
- Return expression: Secuencia de retorno

```
for 'range-var' in 'binding sequence' return 'expression'
```

Para obtener la secuencia de retorno se consigue evaluando la 'return expression' para cada uno de los elementos que se obtienen con la 'binding sequence' asociado a la 'range variable'. El orden del resultado es el mismo que el orden del documento de origen.

Si se especifican varias 'binding sequence' la 'return-expression' se evalúa para todas las ocurrencias del producto cartesiano de las variables implicadas.

```
for $a in fn:distinct-values(book/author)
return (book/author[. = $a][1], book[author = $a]/title)
```

En este caso, para cada uno de los nodos 'author' seleccionados se evalúa la sentencia RETURN, y en este caso devolvería primero el nodo autor y después nodos book; el resultado sería similar a

```
<author>Stevens</author>
<title>TCP/IP Illustrated</title>
<title>Advanced Programming in the Unix environment</title>
<author>Abiteboul</author>
<title>Data on the Web</title>
```

En este otro ejemplo veremos el uso de una sentencia FOR con varias 'binding sequence'

```
for $i in (10, 20),
    $j in (1, 2)
return ($i + $j)
```

En este ejemplo obtendremos una secuencia de retorno compuesta por los valores (11, 21, 12, 22)

### 6.3.3 – Tratamiento condicional

La sentencia condicional consta de tres partes

- test expresión: expresión a evaluar
- then expression:
- else expression:

```
if ('test expression') then 'then expression' else 'else
expression'
```

Se evalúa la 'test expression', y si el valor es true se retorna el valor de 'then expression', en caso contrario se devuelve el valor de 'else expression'. La evaluación se efectúa según estas reglas

- Si es una secuencia vacía se devuelve false.
- Si es una secuencia y el primer elemento es un nodo devuelve true.
- Si es un tipo `xs:boolean` devuelve dicho valor.
- Si es de tipo `xs:string`, `xs:anyURI`, `xs:untypedAtomic` o derivado de estos devuelve true si la longitud es mayor que cero, false si es cero.
- Si es cualquier elemento numérico, si contiene el valor cero ó NaN devuelve false, true en caso contrario
- Cualquier otra situación se produce un error

Hay que destacar que la partícula ELSE no es opcional, y debe estar presente, si se desea que por esa opción no se devuelva nada deberá ponerse ELSE() para que retorne una secuencia vacía.

```
if ($widget1/unit-cost < $widget2/unit-cost)
then $widget1
else $widget2
```

En este caso se devolvería el nodo widget con menor coste por unidad.

### 6.3.4 – Cuantificadores existenciales

Una expresión con un cuantificador existencial siempre retornará un valor booleano y consta de tres partes

- **quantifier:** some ó every
- **in clause:** asocia una variable con una secuencia de nodos a través de su binding sequence (ver sentencia FOR). Puede haber varias cláusulas, y la secuencia final será el producto cartesiano de todas ellas
- **test-expression:**

```
quantifier $var in 'binding-sequence' satisfies 'test-expression'
```

La 'test-expression' es evaluada para cada uno de los elementos creados con las cláusulas in, devolviendo un valor booleano para cada una de ellas (según las reglas expuestas en el caso de la sentencia condicional); el valor de retorno final depende del cuantificador:

- **some:** devuelve true si alguna 'test-expression' ha devuelto true, false en caso contrario
- **every:** devuelve true si todas las 'test-expression' han devuelto true, false en caso contrario

```
every $part in /parts/part satisfies $part/@discounted
```

Esta expresión devuelve true si todos los elementos part tienen un atributo discounted

```
some $x in (1, 2, 3), $y in (2, 3, 4)
  satisfies $x + $y = 4
every $x in (1, 2, 3), $y in (2, 3, 4)
  satisfies $x + $y = 4
```

Estas dos expresiones se evalúan para 9 elementos, y la primera devuelve true mientras que la segunda devuelve false.

### 6.3.5 – Operadores lógicos en expresiones

Se permiten dos operadores lógicos, and y or, en expresiones. Una expresión lógica devolverá true ó false.

```
op1 and op2
op1 or op2
```

Donde operador es una expresión de comparación; las opciones disponibles son

- Valor, para valores simples: eq, ne, lt, le, gt, ge
- Generales, para secuencias de cualquier longitud: =, !=, <, <=, >, >=
- Nodo

- Identidad, es cierto si devuelve el mismo nodo: is
- De posición: <<, >>

Aquí vemos distintos ejemplos de expresiones con operadores lógicos

```
book/author = "Kennedy" or book1/author = "Marc"
//product[weight gt 100 or length lt 75]
```

### 6.3.6 – Combinación de secuencias

Existen 3 operadores de combinación de secuencias

- union: toma dos secuencias y devuelve una secuencia nueva con los nodos que aparecen en cualquiera de las dos secuencias.
- intersect: toma dos secuencias y devuelve una secuencia nueva con los nodos que aparecen en las dos secuencias
- except: toma dos secuencias, y devuelve una secuencia con todos los nodos de la primera que no están en la segunda.

Si tenemos que \$uno = (1, 2, 3) y que \$dos= (3, 4, 5)

\$uno union \$dos devuelve (1, 2, 3, 4, 5)

\$uno intersect \$dos devuelve (3)

\$uno except \$dos devuelve (1, 2)

### 6.3.7 – Comentarios

Tenemos la posibilidad de añadir comentarios en las sentencias XPath 2.0, de modo que al almacenarlas podamos incluir junto con ellas el texto que creamos oportuno de modo que no afecte a su funcionamiento. El formato del comentario es el siguiente

```
`(:' -comentario- `:)'
```

Por ejemplo

```
(: Obtenemos los asegurados que tengan una garantía con un capital de
60000 ó 80000 euros en vigor:)
for $a in //asegurado
return
if($a/garantia[(capital = 60000 or capital = 80000) and @vigor eq
'S'])
then ($a)
else ()
```

### 6.3.8 – Expresiones aritméticas

Podemos expresar operaciones aritméticas, para ello disponemos de los siguientes elementos

- Operadores unarios, con mayor precedencia que los binarios (salvo parentización)
  - +
  - -
- Operadores binarios
  - Aditivos
    - Suma: +
    - Resta: - (la resta en una operación debe ir precedida de un espacio en blanco o pasará a formar parte del elemento anterior)
  - Multiplicativos
    - Multiplicación: \*
    - División: div e idiv (esta última devuelve un entero)
    - módulo

Las siguientes son expresiones XPath válidas, y pueden utilizarse para obtener resultados dentro de otras expresiones

```
-3 + 2    → -1
7 div 3   → 2.3333333
7 idiv 3  → 2
17 mod 5  → 4
```

### 6.3.8 – Instance of

`'expr'` instance of `'expresión de secuencia'`

Este operador devuelve true en el caso de que 'expr' coincida con el tipo de secuencia especificada (ver operadores de secuencia en el modelo de datos)

## 6.4 – Ejemplos

Para la ejecución de estos ejemplos se ha utilizado el mismo juego de pruebas que en los ejemplos desarrollados en XPath 1.0

*\*Se considera en las sentencias que son lanzadas desde el Query Dialog conectado a la colección donde hemos cargado el fichero con el juego de pruebas, si no fuese así, para que las sentencias funcionasen habría que incluir la función de entrada, por ejemplo, si la colección donde la hemos cargado se llama test deberíamos modificar las expresiones del siguiente modo:*

```
for $a in //poliza por
for $a in collection('db/test')//poliza
```

```
//asegurado[garantia[tipo='Vida']] por
collection('/db/Pruebas')//asegurado[garantia[tipo='Vida']]
```

**Ejemplo 1: Tratamiento iterativo**

<b>Objetivo</b>	<b>Obtener los nodos numero y tomador de todas las pólizas</b>
<b>Sentencia</b>	<pre>for \$a in //poliza return (\$a/numero,\$a/tomador)</pre>
<b>Resultado</b>	<pre>&lt;numero&gt;99000000&lt;/numero&gt; &lt;tomador&gt;Manuel Gonzalez Sanchez&lt;/tomador&gt; &lt;numero&gt;99000001&lt;/numero&gt; &lt;tomador&gt;Pedro Martin Gomez&lt;/tomador&gt; &lt;numero&gt;99000002&lt;/numero&gt; &lt;tomador&gt;Alfredo Salas Perez&lt;/tomador&gt;</pre>

**Ejemplo 2: Tratamiento condicional**

<b>Objetivo</b>	<b>Obtener los nodos numero de pólizas con más de dos asegurados</b>
<b>Sentencia</b>	<pre>for \$a in //poliza return if(count(\$a/asegurado)&gt;2) then (\$a/numero) else ()</pre>
<b>Resultado</b>	<pre>&lt;numero&gt;99000002&lt;/numero&gt;</pre>

**Ejemplo 3: Operador existencial some**

<b>Objetivo</b>	<b>Obtener todos los asegurados que tienen alguna garantía que no está en vigor</b>
<b>Sentencia</b>	<pre>for \$a in //asegurado return if(some \$x in \$a/garantia satisfies \$x/@vigor='N') then (\$a) else ()</pre>
<b>Resultado</b>	<pre>&lt;asegurado nombre="Juan" apellidos="López"&gt;   &lt;garantia vigor="N"&gt;     &lt;tipo&gt;Vida&lt;/tipo&gt;     &lt;capital&gt;60000&lt;/capital&gt;   &lt;/garantia&gt; &lt;/asegurado&gt; &lt;asegurado nombre="Maria" apellidos="Fernandez"&gt;   &lt;garantia vigor="S"&gt;     &lt;tipo&gt;Vida&lt;/tipo&gt;     &lt;capital&gt;90000&lt;/capital&gt;   &lt;/garantia&gt;   &lt;garantia vigor="N"&gt;     &lt;tipo&gt;Accidentes&lt;/tipo&gt;     &lt;capital&gt;4000&lt;/capital&gt;   &lt;/garantia&gt;   &lt;garantia vigor="S"&gt;     &lt;tipo&gt;Dental&lt;/tipo&gt;     &lt;capital&gt;300&lt;/capital&gt;   &lt;/garantia&gt; &lt;/asegurado&gt;</pre>



**Ejemplo 4: Operador existencial every**

<b>Objetivo</b>	Obtener todas las pólizas donde todas las garantías de los asegurados estén en vigor
<b>Sentencia</b>	<pre>for \$a in //poliza return if (every \$x in \$a/asegurado/garantia satisfies \$x/@vigor='S') then (\$a) else ()</pre>
<b>Resultado</b>	<pre>&lt;poliza externa="S"&gt;   &lt;numero&gt;99000001&lt;/numero&gt;   &lt;tomador&gt;Pedro Martin Gomez&lt;/tomador&gt;   &lt;asegurado nombre="Pedro" apellidos="Martin"&gt;     &lt;garantia vigor="S"&gt;       &lt;tipo&gt;Vida&lt;/tipo&gt;       &lt;capital&gt;80000&lt;/capital&gt;     &lt;/garantia&gt;   &lt;/asegurado&gt; &lt;/poliza&gt;</pre>

**Ejemplo 5: Operadores lógicos**

<b>Objetivo</b>	Obtener los asegurados que tienen contratada una garantía con un capital de 60000 ó 80000 euros sólo si está en vigor
<b>Sentencia</b>	<pre>for \$a in //asegurado return if (\$a/garantia[(capital = 60000 or capital = 80000) and @vigor eq 'S']) then (\$a) else ()</pre>
<b>Resultado</b>	<pre>&lt;asegurado nombre="Pedro" apellidos="Martin"&gt;   &lt;garantia vigor="S"&gt;     &lt;tipo&gt;Vida&lt;/tipo&gt;     &lt;capital&gt;80000&lt;/capital&gt;   &lt;/garantia&gt; &lt;/asegurado&gt;</pre>

**Ejemplo 6: Combinación de secuencias**

<b>Objetivo</b>	Obtener los asegurados con las garantías de vida y Accidentes
<b>Sentencia</b>	<pre>//asegurado[garantia[tipo='Vida']] intersect //asegurado[garantia[tipo='Accidentes']]</pre>
<b>Resultado</b>	<pre>&lt;asegurado nombre="Maria" apellidos="Fernandez"&gt;   &lt;garantia vigor="S"&gt;     &lt;tipo&gt;Vida&lt;/tipo&gt;     &lt;capital&gt;90000&lt;/capital&gt;   &lt;/garantia&gt;   &lt;garantia vigor="N"&gt;     &lt;tipo&gt;Accidentes&lt;/tipo&gt;     &lt;capital&gt;4000&lt;/capital&gt;   &lt;/garantia&gt;   &lt;garantia vigor="S"&gt;     &lt;tipo&gt;Dental&lt;/tipo&gt;     &lt;capital&gt;300&lt;/capital&gt;   &lt;/garantia&gt; &lt;/asegurado&gt;</pre>

**Ejemplo 7: Operadores aritméticos**

<b>Objetivo</b>	Obtener las pólizas impares del envío
<b>Sentencia</b>	<code>//poliza[(numero mod 2) = 1]</code>
<b>Resultado</b>	<pre>&lt;poliza externa="S"&gt;   &lt;numero&gt;99000001&lt;/numero&gt;   &lt;tomador&gt;Pedro Martin Gomez&lt;/tomador&gt;   &lt;asegurado nombre="Pedro" apellidos="Martin"&gt;     &lt;garantia vigor="S"&gt;       &lt;tipo&gt;Vida&lt;/tipo&gt;       &lt;capital&gt;80000&lt;/capital&gt;     &lt;/garantia&gt;   &lt;/asegurado&gt; &lt;/poliza&gt;</pre>

**Ejemplo 8: Operadores de secuencia**

<b>Objetivo</b>	Obtener los elementos capital
<b>Sentencia</b>	<pre>for \$a in //* return if(\$a instance of element(capital))       then(\$a)       else()</pre>
<b>Resultado</b>	<pre>&lt;capital&gt;10000&lt;/capital&gt; &lt;capital&gt;500&lt;/capital&gt; &lt;capital&gt;60000&lt;/capital&gt; &lt;capital&gt;80000&lt;/capital&gt; &lt;capital&gt;90000&lt;/capital&gt; &lt;capital&gt;90000&lt;/capital&gt; &lt;capital&gt;4000&lt;/capital&gt; &lt;capital&gt;300&lt;/capital&gt;</pre>

Se obtienen todos los nodos del documento, y solo se devuelven si son de tipo capital

**Ejemplo 9: Operadores de secuencia**

<b>Objetivo</b>	Obtener los elementos asegurados con 0 ó una garantía
<b>Sentencia</b>	<pre>for \$a in //asegurado return if(\$a//garantia instance of element(garantia)?)       then(\$a)       else()</pre>
<b>Resultado</b>	<pre>&lt;asegurado nombre="Juan" apellidos="López"&gt;   &lt;garantia vigor="N"&gt;     &lt;tipo&gt;Vida&lt;/tipo&gt;     &lt;capital&gt;60000&lt;/capital&gt;   &lt;/garantia&gt; &lt;/asegurado&gt; &lt;asegurado nombre="Pedro" apellidos="Martin"&gt;   &lt;garantia vigor="S"&gt;     &lt;tipo&gt;Vida&lt;/tipo&gt;     &lt;capital&gt;80000&lt;/capital&gt;   &lt;/garantia&gt; &lt;/asegurado&gt; &lt;asegurado nombre="Lucas" apellidos="Montero"&gt;   &lt;garantia vigor="S"&gt;     &lt;tipo&gt;Vida&lt;/tipo&gt;     &lt;capital&gt;90000&lt;/capital&gt;   &lt;/garantia&gt; &lt;/asegurado&gt; &lt;asegurado nombre="Carmen" apellidos="Sanchez"/&gt;</pre>

Se obtienen todos los nodos asegurado, y para cada uno vemos si la secuencia de garantías corresponde con una secuencia de 0 ó 1 elemento (indicado con el operador '?').

## 7 – XQuery

### 7.1 – Que es XQuery.

XQuery es un lenguaje pensado para la consulta y procesamiento de bases de datos XML y documentos. Surge de la necesidad de disponer de un lenguaje estándar de consultas para acceder a la creciente cantidad de información en formato XML almacenada.

Es una importante extensión de XPath 2.0, al que se le han añadido muchas funciones; como consecuencia de esto, cualquier sentencia XPath 2.0 (y por tanto XPath 1.0 también) es una sentencia XQuery 1.0 válida y debe devolver los mismos resultados.

*Por este motivo, en esta parte de la memoria nos centraremos en aspectos que proporciona XQuery 1.0 sobre XPath 2.0, entendiendo que otros como sentencias condicionales, existenciales etc... son aplicables.*

Al igual que ocurría con XPath 2.0, la definición completa del lenguaje XQuery es muy extensa y está formada por varios documentos, en este documento trataremos los aspectos más importantes. La especificación completa de XQuery 1.0 puede consultarse en:

<http://www.w3.org/TR/xquery/>

### 7.2 – Modelo de datos.

*El modelo de datos de XQuery 2.0 es compartido con XPath 2.0. Para una descripción más detallada ir a la sección modelo de datos de XPath.*

### **Expresiones**

Al igual que en XPath las expresiones XQuery no tienen formato XML y como consecuencia del modelo de datos siempre devuelven una secuencia.

Existe una proposición de estándar para definir una sintaxis XQuery con formato XML llamada XqueryX, la cual podemos consultar en:

<http://www.w3.org/TR/xqueryx/>

### **Axis**

Una implementación de motor para XQuery puede ser Full Axis Feature, es decir, que soporte todos los Axis definidos, pero la definición XQuery dice los Axis ancestro, ancestro-or-self, following, following-sibling, preceding, preceding-sibling son opcionales.

Igualmente, no se reconoce el Axis namespace.

### 7.3 – Sentencias FLWOR.

Es una expresión que permite la unión de variables sobre conjuntos de nodos y la iteración sobre el resultado. FLWOR es la contracción de For Let Where Order Return.

El funcionamiento de la sentencia es el siguiente: con las sentencias FOR y LET (podemos usar una de las dos ó las dos) se consigue una secuencia de variables ligadas a nodos (tuplas); con la sentencia WHERE (opcional) podemos seleccionar o descartar determinadas tuplas de esa secuencia; la sentencia ORDER (opcional) la utilizamos para reordenar la secuencia de tuplas y la sentencia RETURN es evaluada para cada una de las tuplas.

Veamos el formato y funcionamiento de cada una de estas partículas.

*\*Se considera en las sentencias que son lanzadas desde el Query Dialog conectado a la colección donde hemos cargado el fichero con el juego de pruebas, si no fuese así, para que las sentencias funcionasen habría que incluir la función de entrada, por ejemplo, si la colección donde la hemos cargado se llama test deberíamos modificar la expresión del siguiente modo:*

```
for $a in //tomado por
for $a in collection('db/test')//tomador

let $a := //numero por
let $a := collection('/db/Pruebas')//numero
```

#### **FOR**

El objetivo de la sentencia FOR es obtener una secuencia de tuplas; podemos asimilar el concepto de tupla a un valor o nodo del árbol del documento de origen.

El formato de la sentencia FOR es similar al formato en XPath 2.0, consta de:

- Var: variable sobre la que se almacenan los nodos
- Binding sequence: nodos que se van a tratar

```
for 'var' in 'binding sequence'
```

Con esto conseguimos una secuencia de tuplas asociada a la variable. Para cada ocurrencia que consigamos en la 'binding-sequence' obtenemos una tupla.

Se pueden especificar varias variables en la misma sentencia, si lo hacemos de este modo obtendremos una tupla para el producto cartesiano de todas ellas; sobre el juego de pruebas lo veremos mejor con este ejemplo

<pre>for \$a in //numero return &lt;poliza&gt;{\$a}&lt;/poliza&gt;</pre>	<pre>&lt;poliza&gt;   &lt;numero&gt;99000000&lt;/numero&gt; &lt;/poliza&gt; &lt;poliza&gt;   &lt;numero&gt;99000001&lt;/numero&gt; &lt;/poliza&gt; &lt;poliza&gt;   &lt;numero&gt;99000002&lt;/numero&gt; &lt;/poliza&gt;</pre>
<pre>for \$a in //tomador return &lt;poliza&gt;{\$a}&lt;/poliza&gt;</pre>	<pre>&lt;poliza&gt;   &lt;tomador&gt;Manuel Gonzalez Sanchez&lt;/tomador&gt; &lt;/poliza&gt; &lt;poliza&gt;   &lt;tomador&gt;Pedro Martin Gomez&lt;/tomador&gt; &lt;/poliza&gt; &lt;poliza&gt;   &lt;tomador&gt;Alfredo Salas Perez&lt;/tomador&gt; &lt;/poliza&gt;</pre>
<pre>for \$a in //numero, \$b in //tomador return &lt;poliza&gt;{\$a,\$b}&lt;/poliza&gt;</pre>	<pre>&lt;poliza&gt;   &lt;numero&gt;99000000&lt;/numero&gt;   &lt;tomador&gt;Manuel Gonzalez Sanchez&lt;/tomador&gt; &lt;/poliza&gt; &lt;poliza&gt;   &lt;numero&gt;99000000&lt;/numero&gt;   &lt;tomador&gt;Pedro Martin Gomez&lt;/tomador&gt; &lt;/poliza&gt; &lt;poliza&gt;   &lt;numero&gt;99000000&lt;/numero&gt;   &lt;tomador&gt;Alfredo Salas Perez&lt;/tomador&gt; &lt;/poliza&gt; &lt;poliza&gt;   &lt;numero&gt;99000001&lt;/numero&gt;   &lt;tomador&gt;Manuel Gonzalez Sanchez&lt;/tomador&gt; &lt;/poliza&gt; &lt;poliza&gt;   &lt;numero&gt;99000001&lt;/numero&gt;   &lt;tomador&gt;Pedro Martin Gomez&lt;/tomador&gt; &lt;/poliza&gt; &lt;poliza&gt;   &lt;numero&gt;99000001&lt;/numero&gt;   &lt;tomador&gt;Alfredo Salas Perez&lt;/tomador&gt; &lt;/poliza&gt; &lt;poliza&gt;   &lt;numero&gt;99000002&lt;/numero&gt;   &lt;tomador&gt;Manuel Gonzalez Sanchez&lt;/tomador&gt; &lt;/poliza&gt; &lt;poliza&gt;   &lt;numero&gt;99000002&lt;/numero&gt;   &lt;tomador&gt;Pedro Martin Gomez&lt;/tomador&gt; &lt;/poliza&gt; &lt;poliza&gt;   &lt;numero&gt;99000002&lt;/numero&gt;   &lt;tomador&gt;Alfredo Salas Perez&lt;/tomador&gt; &lt;/poliza&gt;</pre>

Observamos que en el Segundo caso con dos variables, devuelve todas las posibles combinaciones de las dos secuencias.

Fijémonos también que la sentencia RETURN es evaluada para cada una de las tuplas, por eso en el primer y segundo caso obtenemos 3 elementos <poliza> mientras que en el último caso obtenemos 9.

→Esta característica nos da la posibilidad de realización de JOINS entre distintos archivos o fuentes de datos

- *Variable posicional*

En la cláusula FOR también podemos indicar la variable posicional, la cual identifica la posición de un elemento en la secuencia de tuplas generada, la manera de especificarlo es con la partícula at,

```
for $a at $p
```

La variable \$p contendrá la posición del elemento \$a tratado en cada momento. En este ejemplo caso vamos a crear un elemento newaseg, que sólo contiene parámetros, de los cuales el último nos indica la posición en la secuencia generada

<pre>for \$a at \$p in //asegurado return &lt;newaseg       pol="{ \$a/ancestor::poliza/numero}"       nombre="{ \$a/@nombre}"       posic="{ \$p}"/&gt;</pre>
<pre>&lt;newaseg nombre="Carlos" pol="99000000" posic="1"/&gt; &lt;newaseg nombre="Juan" pol="99000000" posic="2"/&gt; &lt;newaseg nombre="Pedro" pol="990000001" posic="3"/&gt; &lt;newaseg nombre="Lucas" pol="990000002" posic="4"/&gt; &lt;newaseg nombre="Carmen" pol="990000002" posic="5"/&gt; &lt;newaseg nombre="Maria" pol="990000002" posic="6"/&gt;</pre>

**LET**

El objetivo de la sentencia LET es el mismo de la sentencia FOR, obtener una secuencia de tuplas, pero la forma de hacerlo es distinta. El formato de la sentencia es el siguiente:

```
Let $var := 'binding sequence'
```

Igualmente podemos especificar la unión con varias variables en una misma sentencia.

Mientras que en la sentencia FOR obtenemos una tupla con cada elemento de entrada, con la sentencia LET obtenemos una única tupla. Veamos la diferencia con la sentencia FOR utilizando los mismos ejemplos y comparando resultados:

<pre>let \$a := //numero return &lt;poliza&gt;{\$a}&lt;/poliza&gt;</pre>	<pre>&lt;poliza&gt;   &lt;numero&gt;99000000&lt;/numero&gt;   &lt;numero&gt;990000001&lt;/numero&gt;   &lt;numero&gt;990000002&lt;/numero&gt; &lt;/poliza&gt;</pre>
<pre>let \$a := //tomador return &lt;poliza&gt;{\$a}&lt;/poliza&gt;</pre>	<pre>&lt;poliza&gt;   &lt;tomador&gt;Manuel Gonzalez Sanchez&lt;/tomador&gt;   &lt;tomador&gt;Pedro Martin Gomez&lt;/tomador&gt;   &lt;tomador&gt;Alfredo Salas Perez&lt;/tomador&gt; &lt;/poliza&gt;</pre>

<pre>let \$a := //numero, \$b := //tomador return &lt;poliza&gt;{\$a, \$b}&lt;/poliza&gt;</pre>	<pre>&lt;poliza&gt;   &lt;numero&gt;99000000&lt;/numero&gt;   &lt;numero&gt;99000001&lt;/numero&gt;   &lt;numero&gt;99000002&lt;/numero&gt;   &lt;tomador&gt;Manuel Gonzalez Sanchez&lt;/tomador&gt;   &lt;tomador&gt;Pedro Martin Gomez&lt;/tomador&gt;   &lt;tomador&gt;Alfredo Salas Perez&lt;/tomador&gt; &lt;/poliza&gt;</pre>
---	---

Observemos que la sentencia RETURN sólo se evalúa para una única tupla, y por tanto solo obtenemos un elemento <poliza>.

### Uso conjunto de FOR y LET

Se puede usar conjuntamente las partículas FOR y LET; en este caso para cada tupla devuelta por la sentencia FOR se evalúa la sentencia LET, y después se evalúa RETURN

<pre>for \$a in //numero let \$b:=\$a/ancestor::poliza/tomador return &lt;poliza&gt;{\$a, \$b}&lt;/poliza&gt;</pre>	<pre>&lt;poliza&gt;   &lt;numero&gt;99000000&lt;/numero&gt;   &lt;tomador&gt;Manuel Gonzalez Sanchez&lt;/tomador&gt; &lt;/poliza&gt; &lt;poliza&gt;   &lt;numero&gt;99000001&lt;/numero&gt;   &lt;tomador&gt;Pedro Martin Gomez&lt;/tomador&gt; &lt;/poliza&gt; &lt;poliza&gt;   &lt;numero&gt;99000002&lt;/numero&gt;   &lt;tomador&gt;Alfredo Salas Perez&lt;/tomador&gt; &lt;/poliza&gt;</pre>
---	---

### WHERE

La cláusula WHERE es una cláusula opcional que sirve para seleccionar determinadas tuplas del flujo generado por las sentencias FOR y LET.

La sintaxis de la sentencia es

```
WHERE 'expresión'
```

Donde 'expresión' es evaluado a booleano para cada tupla del flujo, tratándola si el resultado es true, descartándola si es false. La forma de obtener un booleano de la expresión es igual que en la sentencia condicional de XPath 2.0 (apartado 6.3).

<pre>for \$a in //numero where \$a &gt; 99000001 return &lt;poliza&gt;{\$a}&lt;/poliza&gt;</pre>	<pre>&lt;poliza&gt;   &lt;numero&gt;99000002&lt;/numero&gt; &lt;/poliza&gt;</pre>
--	---

<pre>for \$a in //tomador where     case(substring(\$a,1,6)) =     'MANUEL' return &lt;tomadores&gt;{\$a}&lt;/tomadores&gt;</pre>	<pre>&lt;tomadores&gt;   &lt;tomador&gt;     Manuel Gonzalez Sanchez   &lt;/tomador&gt; &lt;/tomadores&gt;</pre>
---	--

La sentencia FOR del primer ejemplo trata todos los nodos número (3), pero solo devuelve el único que cumple la condición. Ocurre lo mismo en la segunda sentencia, donde se tratan todos los tomadores, pero solo se seleccionan los que las 6 primeras posiciones contengan 'MANUEL' independientemente de si están en mayúsculas o minúsculas.

### ORDER BY

La sentencia ORDER se evalúa antes de la sentencia RETURN y después de la sentencia WHERE. Reorganiza la secuencia de tuplas generadas por las sentencias FOR y LET filtradas por WHERE, pero el orden en la que las evalúa la sentencia RETURN viene marcada por el orden que le especifique la cláusula ORDER BY.

La forma de ordenar la secuencia de tuplas viene dada a través de las order-specs; se pueden especificar varias order-specs.

```
order by 'expr' 'order-modifier'
stable order by 'expr' 'order-modifier'
```

Se puede especificar el orden de la ordenación del valor

- ascending
- descending

Igualmente se puede especificar como tratar las tuplas que no contengan el elemento por el que se ordena

- empty greatest
- empty least

Si se especifica stable al comienzo de la sentencia queremos decir que en el caso de existir dos tuplas que contengan valores idénticos en todos los order-spec que hemos utilizado se mantenga el orden de entrada (*stable es una opción no reconocida –al menos de momento- en eXist*).

Veamos un ejemplo del uso de order by

<pre>for \$a in //tomador return \$a</pre>	<pre>&lt;tomador&gt;Manuel Gonzalez Sanchez&lt;/tomador&gt; &lt;tomador&gt;Pedro Martin Gomez&lt;/tomador&gt; &lt;tomador&gt;Alfredo Salas Perez&lt;/tomador&gt;</pre>
<pre>for \$a in //tomador order by \$a ascending return \$a</pre>	<pre>&lt;tomador&gt;Alfredo Salas Perez&lt;/tomador&gt; &lt;tomador&gt;Manuel Gonzalez Sanchez&lt;/tomador&gt; &lt;tomador&gt;Pedro Martin Gomez&lt;/tomador&gt;</pre>
<pre>for \$a in //tomador order by \$a descending return \$a</pre>	<pre>&lt;tomador&gt;Pedro Martin Gomez&lt;/tomador&gt; &lt;tomador&gt;Manuel Gonzalez Sanchez&lt;/tomador&gt; &lt;tomador&gt;Alfredo Salas Perez&lt;/tomador&gt;</pre>



## RETURN

La sentencia RETURN, como hemos dicho anteriormente es evaluada una vez para cada tupla dentro de la secuencia de tuplas obtenida con las sentencias FOR, LET, WHERE y en el orden especificado por ORDER BY.

La potencia de esta sentencia radica (al contrario que en XPath) en que puede ser cualquier expresión (incluida una sentencia FOR) y que nos permite la creación de nuevos elementos (ver apartado de creación de nodos para más detalle) y por tanto poder cambiar la estructura del documento.

En los ejemplos anteriores hemos visto como se devolvía la variable tratada o sencillamente se envolvía todo con un elemento raíz para ilustrar diferencias entre FOR y LET, pero veamos ahora algún ejemplo más complejo

### Ejemplo 1

Objetivo	Obtener los asegurados de la primera póliza, cambiando el formato convirtiendo los atributos en elementos
Sentencia	<pre>for \$a in /envio/poliza[1]/asegurado return &lt;newaseg&gt;       &lt;nombre&gt;{\$a/@nombre}&lt;/nombre&gt;       &lt;apellidos&gt;{\$a/@apellidos}&lt;/apellidos&gt; &lt;/newaseg&gt;</pre>
Resultado	<pre>&lt;newaseg&gt;   &lt;nombre&gt;Carlos&lt;/nombre&gt;   &lt;apellidos&gt;Sanchez&lt;/apellidos&gt; &lt;/newaseg&gt; &lt;newaseg&gt;   &lt;nombre&gt;Juan&lt;/nombre&gt;   &lt;apellidos&gt;López&lt;/apellidos&gt; &lt;/newaseg&gt;</pre>

### Ejemplo 2

Objetivo	Obtener un nuevo elemento póliza que contenga un parámetro nuevo con el número de asegurados
Sentencia	<pre>for \$a in //poliza return &lt;poliza numaseg="{count(\$a/asegurado)}"&gt;       &lt;/poliza&gt;</pre>
Resultado	<pre>&lt;poliza numaseg="2"/&gt; &lt;poliza numaseg="1"/&gt; &lt;poliza numaseg="3"/&gt;</pre>

### Ejemplo 3

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

<b>Sentencia</b>	<pre> for \$a in //poliza return &lt;poliza numaseg="{count(\$a/asegurado)}"&gt;     {for \$b in \$a/asegurado     return         &lt;asegurado numgar="{count(\$b/garantia)}"&gt;             {\$b/@nombre, \$b/@apellidos}         &lt;/asegurado&gt; }     &lt;/poliza&gt; </pre>
<b>Resultado</b>	<pre> &lt;poliza numaseg="2"&gt;     &lt;asegurado numgar="2"&gt;Carlos Sanchez&lt;/asegurado&gt;     &lt;asegurado numgar="1"&gt;Juan López&lt;/asegurado&gt; &lt;/poliza&gt; &lt;poliza numaseg="1"&gt;     &lt;asegurado numgar="1"&gt;Pedro Martin&lt;/asegurado&gt; &lt;/poliza&gt; &lt;poliza numaseg="3"&gt;     &lt;asegurado numgar="1"&gt;Lucas Montero&lt;/asegurado&gt;     &lt;asegurado numgar="0"&gt;Carmen Sanchez&lt;/asegurado&gt;     &lt;asegurado numgar="3"&gt;Maria Fernandez&lt;/asegurado&gt; &lt;/poliza&gt; </pre>

#### **7.4 – Creación de nodos.**

Hemos visto en los ejemplos anteriores que en la cláusula RETURN podíamos especificar la creación de nuevos nodos en el resultado de la consulta, veamos con más profundidad este aspecto.

Existen dos modos de creación de elementos

- Escribiendo las etiquetas de comienzo y cierre.
- Utilizando los constructores de elementos y atributos

Cuando se utiliza un constructor, las expresiones que se utilizan para componerlos y tienen que ser evaluadas se indican entre llaves {----} ya que si no la expresión sería tomada como un literal.

##### *- Utilizando etiquetas de comienzo y cierre*

```

for $a in /envio/poliza[1]/asegurado
return <newaseg poliza="{ $a/ancestor::poliza/numero}">
    <nombre>{ $a/@nombre}</nombre>
    <apellidos>{ $a/@apellidos}</apellidos>
</newaseg>

```

En este caso, consultamos los asegurados de la primera póliza, y los recuperamos dándoles un formato nuevo, creando el elemento newaseg con un parámetro que es el número de póliza, que contiene a su vez dos elementos hijos, nombre y apellidos; vemos como el contenido de los elementos está delimitado por '{ }' y que el contenido de las mismas es una expresión que se evalúa en ejecución (en este caso obteniendo el valor de los parámetros)

*- Utilizando los constructores de elementos y atributos*

```

for $a in /envio/poliza[1]/asegurado
return element newaseg{
  attribute poliza { $a/ancestor::poliza/numero },
  element nombre { $a/@nombre },
  element apellidos { $a/@apellidos }
}

```

Vemos que existen las secuencias de creación de nodos

```

element nombre {      } para elementos
attribute nombre {    } para atributos

```

En ambos casos entre llaves se especifica el contenido de los elementos; dentro de cada elemento si tiene un atributo lo especificamos primero, y después los elementos hijos. Los elementos secuenciales dentro de cada elemento van separados con comas.

El resultado de estas dos expresiones es idéntico, y es

```

<newaseg poliza="99000000">
  <nombre nombre="Carlos"/>
  <apellidos apellidos="Sanchez"/>
</newaseg>
<newaseg poliza="99000000">
  <nombre nombre="Juan"/>
  <apellidos apellidos="López"/>
</newaseg>

```

**7.5 – Creación de variables.**

Es posible la declaración de variables en un documento XQuery, la manera de hacerlo es

```

declare variable $nombre-----:= 'valor'---
                        |-as 'tipo'--| |--external--|

```

Si declaramos la variable y le asignamos un valor, y no definimos tipo, el tipo que se le asigna está obtenido a partir de 'valor', pero si le definimos un tipo (por ejemplo xs:integer) el valor que le damos tiene que ser coherente.

Si se define con la variable external se entiende que el valor de la variable tiene que ser suministrado desde fuera de la sentencia (por ejemplo, desde un entorno que permita la asociación de variables, o desde un programa java que llame, realizando el bind de la variable)

**Ejemplo 1**

<b>Objetivo</b>	Obtener solamente los número de las pólizas cuya posición es la indicada en el vector
<b>Sentencia</b>	<pre>declare variable \$vector:=(1,2); for \$pol at \$pos in //poliza return if(some \$x in \$vector satisfies \$x=\$pos) then (\$pol/numero) else ()</pre>
<b>Resultado</b>	<pre>&lt;numero&gt;99000000&lt;/numero&gt; &lt;numero&gt;99000001&lt;/numero&gt;</pre>

Como vemos, solamente se devuelven las pólizas cuya posición es 1 y 2; si cambiásemos esos valores el resultado de la consulta cambiaría

**Ejemplo 2**

<b>Objetivo</b>	Obtener as garantías con un importe 10000
<b>Sentencia</b>	<pre>declare variable \$importe as xs:integer:=10000; for \$a in //garantia[capital=\$importe] return \$a</pre>
<b>Resultado</b>	<pre>&lt;garantia vigor="S"&gt;   &lt;tipo&gt;Accidentes&lt;/tipo&gt;   &lt;capital&gt;10000&lt;/capital&gt; &lt;/garantia&gt;</pre>

En este caso definimos explícitamente el tipo de la variable, diciendo que \$importe es de tipo entero; si el valor asignado no coincide con dicho tipo se genera un error, por ejemplo, la sentencia

```
declare variable $importe as xs:integer:="Pedro";
```

Generaría un error al no ser "Pedro" un número entero válido.

**7.6 – Declaración de funciones.**

Como añadido a las funciones definidas en el estándar se permite la definición de nuevas funciones:

```
Declare function 'nom' ('param') as retorno---{'cuerpo'};--
|-external-|
```

El nombre de la función debe de ser un QName completo, es decir, debe tener el prefijo del namespace y un nombre de función; si la función está definida dentro del mismo módulo (ver definición de módulos en el punto 7.7) debe de usarse el namespace 'local'

```
local:nombre_función
```

La definición de parámetros se realiza del siguiente modo

```
$nombre -----
      |--as 'tipo'--|
```

Si no se define tipo, por defecto se asume item()

Si se usa la opción 'external' indicamos que la definición de la función queda fuera del entorno de la consulta; de otro modo dentro del cuerpo de la función deberemos especificar las sentencias que permiten obtener el resultado.

### Ejemplo

<b>Objetivo</b>	Obtener un nuevo elemento poliza con un parámetro número y un elemento tomador
<b>Sentencia</b>	<pre>declare function local:tom-por-pol(\$pol) as element() {     for \$b in //poliza     where \$b/numero=\$pol     return \$b/tomador };  for \$a in //poliza return &lt;newpol num="{ \$a/numero }"&gt;{     local:tom-por-pol(\$a/numero) } &lt;/newpol&gt;</pre>
<b>Resultado</b>	<pre>&lt;newpol num="99000000"&gt;   &lt;tomador&gt;Manuel Gonzalez Sanchez&lt;/tomador&gt; &lt;/newpol&gt; &lt;newpol num="99000001"&gt;   &lt;tomador&gt;Pedro Martin Gomez&lt;/tomador&gt; &lt;/newpol&gt; &lt;newpol num="99000002"&gt;   &lt;tomador&gt;Alfredo Salas Perez&lt;/tomador&gt; &lt;/newpol&gt;</pre>

Declaramos una función (tom-por-pol, que al estar en el mismo modulo tiene como prefijo 'local'), que recibe como parámetro un ítem, con el cual localizaremos el tomador de la póliza cuyo número sea igual a dicho ítem (es el cuerpo de la función).

En la consulta tratamos todas las pólizas, y en el cuerpo de construcción del elemento newpol efectuamos la llamada a la función definida anteriormente.

### 7.7 – Módulos.

Hasta ahora hemos visto expresiones XQuery, incluidas las declaraciones de variables y de funciones, y aunque hasta ahora no hemos sido conscientes de ello estas expresiones o consultas están siempre dentro de un módulo.

Una consulta puede estar compuesta de uno o varios módulos; un módulo es una expresión XQuery precedida de un prólogo opcional.

Toda consulta tiene un módulo principal o main module el cual tiene que tener el cuerpo de la expresión ó query body, que es el que contendrá la expresión que se evaluará y definirá el resultado (es decir, todas las expresiones vistas hasta ahora son cuerpos de un módulo).

Si un módulo no tiene query body (es decir, no tiene expresión) entonces estamos hablando de un library module.

Igualmente el módulo puede tener una declaración de versión; por tanto, la estructura de un módulo es la siguiente

```
-----prólogo-----query body-----
|-Decl. de versión--|      |--decl. de modulo-----prólogo-----|
```

### Declaración de versión

Si hay declaración de versión está al principio del módulo, y sirve para indicar que dicho módulo debe de ser ejecutado por un procesador XQuery que soporte la versión indicada.

```
--xquery version---`ver`-----`;`
|---encoding---`formato`----|
```

Ver es una cadena que indica que versión debe soportar (de momento sólo "1.0") y formato especifica la codificación del archivo ("UTF-8", "UTF-16", "US-ASCII")

```
xquery version "1.0";
xquery version "1.0" encoding "utf-8";
```

### Declaración de modulo

Esta expresión sirve para indicar que es una librería, el formato es el siguiente

```
--module namespace---`nombre`--- = URI;
```

Con nombre definimos el prefijo y con URI definimos el namespace de todas las variables y funciones definidas en el módulo

```
module namespace math = "http://example.org/math-functions";
```

### Prólogo

Con el prólogo definimos el entorno de ejecución de la consulta; esta dividido en dos partes, una primera que contiene los establecimientos de valores (namespace, otros imports...) y una segunda que contiene la declaración de variables y funciones (esto es así porque la segunda parte se ve afectada por la primera); se pueden especificar múltiples opciones

```

-----
|-Default namespace--`; '*-|  |-declaracion variables-----`; '*--|
|-Setter-----|           |-declaración de funciones-|
|-namespace-----|       |-declaración de opciones--|
|-import-----|

```

### Default namespace

Sirve para poder utilizar nombres sin prefijos dentro del módulo (más corto y más sencillo), podemos especificar un namespace por defecto para las variables y otro para las funciones

```

declare default element namespace "http://uoc.org/names";
declare default function namespace "http://uoc.org/funciones";

```

### Namespace

Declara un namespace y lo asocia con su prefijo para poder usarlo dentro del módulo

```

declare namespace uoc = "http://uoc.org";

```

### Setter

Sirve para modificar los comportamientos del motor de ejecución

```

---BoundarySpace-----`; '--
|-DefaultCollation-----|
|-BaseURI-----|
|-Construction-----|
|-OrderingMode-----|
|-EmptyOrder-----|
|-CopyNamespace-----|

```

### *Boundary Space*

Indicamos si los espacios en blanco de comienzo y final de un elemento se deben mantener o son descartados

```

declare boundary-space preserve;
declare boundary-space strip;

```

### *Base URI*

Especificamos la URI base para resolver URIs relativas dentro del módulo; sólo se puede especificar una.

```

declare base-uri "http://uoc.org";

```

### *Construction*

Definir el funcionamiento de los constructores de elementos, puede tomar dos valores, `preserve`, con lo que el tipo del elemento construido es `xs:anyType` y todos los elementos dentro de el mantienen sus tipos ó `strip`, y todos los elementos se construyen siendo del tipo `xs:untyped`.

```
declare construction preserve;
declare construction strip;
```

### *Ordering mode*

En sentencias de selección XPath (con `/` y `//`) y sobre operadores de secuencias `union`, `intersect` y `except` especificamos si queremos el orden del documento (`ordered`) o el orden que imponga la implementación del interprete XQuery (`unordered`).

```
declare ordering ordered;
declare ordering unordered;
```

### *Empty order*

Controlamos la colocación de las secuencias vacías y valores NaN dentro de un `order by`, podemos especificar que sean tratadas como el valor mayor o menor.

```
declare default order empty greatest;
declare default order empty least;
```

### *Copy namespace*

Especificamos la asignación a un namespace cuando un elemento es copiado dentro de otro nuevo definido con un constructor de elementos

```
--declare copy-namespaces----preserve-----inherit-----`;'
|---no-preserve-| |---no-inherit---|
```

### Import

Podemos importar en el modulo actual tanto xml schemas como otros módulos.

#### *Importar Schemas*

Cuando añadimos un xml schema importamos la definición de elementos, atributos y tipos dentro del mismo.

```
import schema---namespace---`nombre'---`=---"URI"-----`;'
|--- default element namespace---| |---at-"URI"---|
```



Podemos especificar un prefijo o que sea el schema por defecto; si especificamos un prefijo no puede ser ni xml, xmlns ni estar duplicado respecto a otros prefijos definidos

Se pueden especificar varias partículas 'at' para especificar varias localizaciones para el schema;

```
import schema namespace pref="http://www.uoc.oedu/funciones"

import schema default element namespace
"http://example.org/abc";
```

### *Importar módulos*

Con esta sentencia definimos un namespace y le damos un prefijo, al igual que con los schemas no puede ser xml, xmlns ni otro prefijo definido; le asociamos las implementaciones definidas en URI (el URI definido en el módulo y en la sentencia import debe de ser el mismo).

Si no coincide el URI con la localización del módulo deberá incluirse la partícula 'at' dándole la dirección del módulo:

La importación de módulos no es transitiva, por lo que no tendremos acceso a funciones y módulos que importen módulos que importamos, es decir, que si importamos el módulo M1 y este a su vez importa el módulo M2, no tendremos acceso a las funciones de M2

```
import module namespace---`nombre`---`=/--"URI"-----;
|at-"URI"-----|

import module namespace nspres="ns"
import module namespace nspres="ns" at "file:///C:\modulo"
```

### Declaración de variables

Ver punto 7.5

### Declaración de funciones

Ver punto 7.6

### Declaración de opciones

Con estas sentencias podemos definir el comportamiento de una implementación concreta. Aunque estas opciones solo las entenderá la implementación específica su sintaxis hace que puedan ser parseadas correctamente por el resto.

```
declare option exq:output "encoding = iso-8859-1";
declare option exq:strip-comments "true";
```

## Ejemplo de uso de módulos

Vamos a utilizar el ejemplo de definición de función del punto 7.6, adaptándolo convenientemente de modo que la función esté en un módulo que importaremos.

El módulo estará en un archivo llamado modulo.xquery y tiene el siguiente contenido:

```
module namespace mismod = "mismodulos";

declare function mismod:tom-por-pol($pol) as element()
{
    for $b in //poliza
    where $b/numero=$pol
    return $b/tomador
};
```

Hemos declarado el namespace “mismodulos” y definido la función tom-por-pol dentro de el (prefijo mismod).

La consulta que hará uso del módulo la podemos introducir directamente en el Query dialog; es la misma consulta que en el punto 7.6, sustituyendo la declaración de la función por una sentencia import para poder usar el módulo:

```
import module namespace misfunc="mismodulos"
                        at "file:///C:\libreria\modulo.xquery";
for $a in //poliza
return <newpol num="{ $a/numero }">{
    misfunc:tom-por-pol($a/numero)
}
</newpol>
```

En este caso se importa un fichero dándole una ruta local completa; si no especificamos la ruta completa y damos sencillamente el nombre del fichero por defecto exist los busca en %exist%\tools\wrapper\bin\

Hay que resaltar que el namespace de la sentencia import y la del módulo deben coincidir, sino se produce un error de ejecución.

## 7.8 – JOINS, como hacerlos

En el modelo relacional, una de las acciones más frecuentes es la realización de joins o consultas donde se ven involucradas más de una tabla o fuente de datos; también es posible la realización de consultas que mezclen más de un fichero ó colección de base de datos de modo que podamos generar un documento que sea la fusión de los anteriores, veamos de que manera.

En los ejemplos de esta sección utilizaremos el juego de pruebas utilizado en las secciones previas, pero además deberemos cargar un documento nuevo en la base de datos con el siguiente contenido

```
<?xml version="1.0" encoding="UTF-8"?>
<personas>
  <persona>
    <nombre>Manuel Gonzalez Sanchez</nombre>
    <direcc>c\pez</direcc>
    <ciudad>Madrid</ciudad>
    <edad>45</edad>
  </persona>
  <persona>
    <nombre>Pedro Martin Gomez</nombre>
    <direcc>paseo de las letras</direcc>
    <ciudad>Salamanca</ciudad>
    <edad>27</edad>
  </persona>
  <persona>
    <nombre>Federico Carrasco</nombre>
    <direcc>Plaza Grande</direcc>
    <ciudad>Cadiz</ciudad>
    <edad>23</edad>
  </persona>
</personas>
```

Es una pequeña base de datos de personas que usaremos para combinar con la base de datos de pólizas

En los ejemplos puestos se entenderá que estamos conectados con el Query Dialog en la colección donde hemos cargado los juegos de pruebas y que estos tienen los siguientes nombres

- Pólizas: test\_xpath.xml
- Personas: test\_xpath\_personas.xml

### 7.8.1 INNER JOIN

Con inner join entendemos la consulta en la que se mezclan datos de dos fuentes y solo se mantienen cuando hay correspondencia entre los dos (por ejemplo, si hacemos un join entre una tabla de pedidos y una tabla de clientes solo se mantendrán los pedidos con clientes y clientes con pedidos).

Este tipo de JOIN es posible realizarlo mediante la sentencia FOR (punto 7.3); en esta sentencia es posible especificar varias fuentes de datos ligándolas a distintas variables, y como resultado se obtiene un conjunto de tuplas compuesto por el producto cartesiano de todas las tuplas de cada variable ligada ó conjunto de datos (ver que esto es igual que una SELECT SQL donde se consulta más de una tabla); si especificamos una condición de relación en una de las fuentes a través de un valor habremos realizado un JOIN, veámoslo con un ejemplo:

Si efectuamos la siguiente consulta, para obtener los distintos tomadores que existen en la base de datos de pólizas

```
for $a in distinct-values(doc("test_xpath.XML")//tomador)
return <nombre>{$a}</nombre>
```

Obtenemos la siguiente salida, existiendo 3 tomadores distintos

```
<nombre>Manuel Gonzalez Sanchez</nombre>
<nombre>Pedro Martin Gomez</nombre>
<nombre>Alfredo Salas Perez</nombre>
```

Vamos ahora a efectuar una consulta que obtenga datos de los tomadores de pólizas completando los datos del tomador con los existentes en el fichero de personas

### Ejemplo 1

<b>Objetivo</b>	Obtener una relación de los datos personales completos de los tomadores de pólizas existentes
<b>Sentencia</b>	<pre>for \$a in   distinct-values(doc("test_xpath.XML")//tomador),   \$b in     doc("test_xpath_personas.XML")//persona[<b>nombre=\$a</b>] return &lt;contacto&gt;   &lt;nombre&gt;{\$a}&lt;/nombre&gt;   &lt;datospers&gt;{\$b/ciudad,     \$b/edad}   &lt;/datospers&gt; &lt;/contacto&gt;</pre>
<b>Resultado</b>	<pre>&lt;contacto&gt;   &lt;nombre&gt;Manuel Gonzalez Sanchez&lt;/nombre&gt;   &lt;datospers&gt;     &lt;ciudad&gt;Madrid&lt;/ciudad&gt;     &lt;edad&gt;45&lt;/edad&gt;   &lt;/datospers&gt; &lt;/contacto&gt; &lt;contacto&gt;   &lt;nombre&gt;Pedro Martin Gomez&lt;/nombre&gt;   &lt;datospers&gt;     &lt;ciudad&gt;Salamanca&lt;/ciudad&gt;     &lt;edad&gt;27&lt;/edad&gt;   &lt;/datospers&gt; &lt;/contacto&gt;</pre>

Vemos que en la sentencia FOR ligamos dos variables, una sobre el fichero de pólizas obteniendo todos los tomadores (\$a), y otra que la ligamos con todas las personas (\$b); además en la variable \$b condicionamos las personas que leemos sean iguales que el tomador leído (`persona[nombre=$a]`).

De este modo obtenemos el producto cartesiano de los dos ficheros restringiendo a las tuplas con el mismo nombre de persona, de este modo hemos efectuado el JOIN; en la sentencia return devolvemos datos tanto de \$a como de \$b.

En el resultado solamente aparecen dos tomadores que son en los que existen coincidencias con el fichero de personas.

### 7.8.2 OUTER JOIN

Con outer join entendemos la consulta en la que se mezclan datos de dos fuentes y se mantienen cuando hay correspondencia entre los dos y los de una de las fuentes aun cuando no existe correspondencia.

En este caso el enfoque es distinto, y utilizando las posibilidades de incluir cualquier expresión en la sentencia return podemos efectuar un outer join

#### Ejemplo 1

<b>Objetivo</b>	Obtener una relación de los tomadores de las pólizas, completándolos con los datos personales en caso de que existan en el fichero de clientes y si no existen mostrar sencillamente el nombre
<b>Sentencia</b>	<pre> for \$a in distinct-values(doc("test_xpath.XML")//tomador) return &lt;contacto&gt;   &lt;nombre&gt;{\$a}&lt;/nombre&gt;   {     for \$b in       doc("test_xpath_personas.XML")//persona[nombre=\$a]     return &lt;datospers&gt;{\$b/direcc,\$b/edad}&lt;/datospers&gt;   } &lt;/contacto&gt; </pre>
<b>Resultado</b>	<pre> &lt;contacto&gt;   &lt;nombre&gt;Manuel Gonzalez Sanchez&lt;/nombre&gt;   &lt;datospers&gt;     &lt;direcc&gt;c\pez&lt;/direcc&gt;     &lt;edad&gt;45&lt;/edad&gt;   &lt;/datospers&gt; &lt;/contacto&gt; &lt;contacto&gt;   &lt;nombre&gt;Pedro Martin Gomez&lt;/nombre&gt;   &lt;datospers&gt;     &lt;direcc&gt;paseo de las letras&lt;/direcc&gt;     &lt;edad&gt;27&lt;/edad&gt;   &lt;/datospers&gt; &lt;/contacto&gt; &lt;contacto&gt;   &lt;nombre&gt;Alfredo Salas Perez&lt;/nombre&gt; &lt;/contacto&gt; </pre>

En este caso para cada tomador se devuelve un contacto, y dentro de la sentencia return accedemos al fichero de personas; si existen datos los completará, pero si no existen el tomador se devuelve igualmente; la salida es igual que en el caso anterior pero añadiendo el tercer tomador que no existe en el fichero de personas.

Podemos decir que este es un Left Outer Join, aplicando el mismo método podríamos realizar un Right Outer Join, manteniendo los datos de las personas que no tienen pólizas.

### 7.8.3 FULL OUTER JOIN

Con full outer join entendemos la consulta en la que se mezclan datos de dos fuentes y se mantienen los datos de las dos fuentes, tanto cuando hay correspondencia como cuando no existe (en los dos sentidos)

#### Ejemplo 2

Objetivo	Obtener una relación de los tomadores de las pólizas, completándolos con los datos personales en caso de que existan en el fichero de clientes y si no existen mostrar sencillamente el nombre, además <u>añadiremos las personas de la base de datos de personas sin pólizas</u>
Sentencia	<pre> for \$a in distinct-values(doc("test_xpath.XML")//tomador) return &lt;contacto&gt;   &lt;nombre&gt;{\$a}&lt;/nombre&gt;   {     for \$b in       doc("test_xpath_personas.XML")//persona[nombre=\$a]     return &lt;datospers&gt;{\$b/direcc,\$b/edad}&lt;/datospers&gt;   } &lt;/contacto&gt;,  for \$c in   doc("test_xpath_personas.XML")//persona where empty(doc("test_xpath.XML")//poliza[tomador=\$c/nombre]) return &lt;contacto&gt;   {\$c/nombre}   &lt;datospers&gt;{\$c/direcc,\$c/edad}&lt;/datospers&gt; &lt;/contacto&gt; </pre>

<b>Resultado</b>	<pre> &lt;contacto&gt;   &lt;nombre&gt;Manuel Gonzalez Sanchez&lt;/nombre&gt;   &lt;datospers&gt;     &lt;direcc&gt;c\pez&lt;/direcc&gt;     &lt;edad&gt;45&lt;/edad&gt;   &lt;/datospers&gt; &lt;/contacto&gt; &lt;contacto&gt;   &lt;nombre&gt;Pedro Martin Gomez&lt;/nombre&gt;   &lt;datospers&gt;     &lt;direcc&gt;paseo de las letras&lt;/direcc&gt;     &lt;edad&gt;27&lt;/edad&gt;   &lt;/datospers&gt; &lt;/contacto&gt; &lt;contacto&gt;   &lt;nombre&gt;Alfredo Salas Perez&lt;/nombre&gt; &lt;/contacto&gt; &lt;contacto&gt;   &lt;nombre&gt;Federico Carrasco&lt;/nombre&gt;   &lt;datospers&gt;     &lt;direcc&gt;Plaza Grande&lt;/direcc&gt;     &lt;edad&gt;23&lt;/edad&gt;   &lt;/datospers&gt; &lt;/contacto&gt; </pre>
------------------	--

Esta es la misma consulta anterior donde después de la sentencia RETURN del primer FOR se concatena otra sentencia for donde obtenemos las personas que no son tomadores de las pólizas:

```

for $c in
  doc("test_xpath_personas.XML")//persona
  where empty(doc("test_xpath.XML")//poliza[tomador=$c/nombre])
return <contacto>
  {$c/nombre}
  <datospers>{$c/direcc,$c/edad}</datospers>
</contacto>

```

La salida es idéntica a la anterior, pero añadiendo a un nuevo contacto, Federico Carrasco, que no tiene ninguna póliza

## **7.9 – Funciones agregadas**

Otra de las operaciones más frecuentes realizando consultas relacionales es la obtención de datos agrupados (usando GROUP BY), obteniendo estadísticas de grupo como numero de registros, sumatorios, medias etc...Este tipo de consultas se pueden realizar haciendo uso de la función distinct-values, la cual ya hemos visto en la sección anterior.

La función distinct-values() devuelve una secuencia de elementos no repetidos donde el orden no es significativo, a partir de ella podremos usar funciones de agregación sobre los elementos de dichos grupos (count, avg, max, min) como muestra el siguiente ejemplo

<b>Objetivo</b>	Obtener un resumen sobre las garantías contratadas en las pólizas, obteniendo un elemento resgar con un atributo tipo especificando la clase de garantía y una atributo numgar especificando el número de garantías total. Además para cada garantía obtendremos un elemento que nos dará el importe total, otro la media de capital por garantía, otro elemento con el importe máximo contratado y otro con el importe mínimo
<b>Sentencia</b>	<pre> for \$tipogar in distinct-values(//tipo) let \$totimp:=sum(//garantia[tipo=\$tipogar]/capital) let \$medimp:=avg(//garantia[tipo=\$tipogar]/capital) let \$maximp:=max(//garantia[tipo=\$tipogar]/capital) let \$minimp:=min(//garantia[tipo=\$tipogar]/capital) return   &lt;resgar tipo="{ \$tipogar}"     numgar="{count(//garantia[tipo=\$tipogar])}"&gt;     &lt;imp_total&gt;{\$totimp}&lt;/imp_total&gt;     &lt;imp_medio&gt;{\$medimp}&lt;/imp_medio&gt;     &lt;imp_max&gt;{\$maximp}&lt;/imp_max&gt;     &lt;imp_min&gt;{\$minimp}&lt;/imp_min&gt;   &lt;/resgar&gt; </pre>
<b>Resultado</b>	<pre> &lt;resgar tipo="Accidentes" numgar="2"&gt;   &lt;imp_total&gt;14000&lt;/imp_total&gt;   &lt;imp_medio&gt;7000&lt;/imp_medio&gt;   &lt;imp_max&gt;10000&lt;/imp_max&gt;   &lt;imp_min&gt;4000&lt;/imp_min&gt; &lt;/resgar&gt; &lt;resgar tipo="Dental" numgar="2"&gt;   &lt;imp_total&gt;800&lt;/imp_total&gt;   &lt;imp_medio&gt;400&lt;/imp_medio&gt;   &lt;imp_max&gt;500&lt;/imp_max&gt;   &lt;imp_min&gt;300&lt;/imp_min&gt; &lt;/resgar&gt; &lt;resgar tipo="Vida" numgar="4"&gt;   &lt;imp_total&gt;320000&lt;/imp_total&gt;   &lt;imp_medio&gt;80000&lt;/imp_medio&gt;   &lt;imp_max&gt;90000&lt;/imp_max&gt;   &lt;imp_min&gt;60000&lt;/imp_min&gt; &lt;/resgar&gt; </pre>

### **7.10 – Algunos ejemplos más.**

Como añadido a los ejemplos explicativos de las distintas expresiones en los apartados anteriores, se incluyen las sentencias siguientes.



**Ejemplo 1**

<b>Objetivo</b>	Obtener los asegurados de la póliza 99000000, obteniendo un nuevo formato con un campo nombre , un contador de garantías y un total de importe
<b>Sentencia</b>	<pre>for \$a in /envio/poliza[numero=99000000]/asegurado return &lt;newaseg&gt;     &lt;nombre&gt;{concat(\$a/@nombre,                     ' ',                     \$a/@apellidos)}     &lt;/nombre&gt;     &lt;numgars imptotal="{sum(\$a/garantia/capital)}"&gt;         {count(\$a/garantia)}     &lt;/numgars&gt; &lt;/newaseg&gt;</pre>
<b>Resultado</b>	<pre>&lt;newaseg&gt;   &lt;nombre&gt;Carlos Sanchez&lt;/nombre&gt;   &lt;numgars imptotal="10500"&gt;2&lt;/numgars&gt; &lt;/newaseg&gt; &lt;newaseg&gt;   &lt;nombre&gt;Juan López&lt;/nombre&gt;   &lt;numgars imptotal="60000"&gt;1&lt;/numgars&gt; &lt;/newaseg&gt;</pre>

**Ejemplo 2**

<b>Objetivo</b>	Obtener una relación de pólizas ordenadas por el número de asegurados, y dentro de cada una la relación de asegurados ordenados por el número de garantías
<b>Sentencia</b>	<pre>for \$a in //poliza order by count(\$a/asegurado) return &lt;pol id="{ \$a/numero}"         numaseg="{count(\$a/asegurado)}"&gt;   {for \$b in \$a/asegurado   order by count(\$b/garantia)   return     &lt;aseg numgar="{count(\$b/garantia)}"       nombre="{ \$b/@nombre}"&gt;     &lt;/aseg&gt;} &lt;/pol&gt;</pre>
<b>Resultado</b>	<pre>&lt;pol id="99000001" numaseg="1"&gt;   &lt;aseg numgar="1" nombre="Pedro"/&gt; &lt;/pol&gt; &lt;pol id="99000000" numaseg="2"&gt;   &lt;aseg numgar="1" nombre="Juan"/&gt;   &lt;aseg numgar="2" nombre="Carlos"/&gt; &lt;/pol&gt; &lt;pol id="99000002" numaseg="3"&gt;   &lt;aseg numgar="0" nombre="Carmen"/&gt;   &lt;aseg numgar="1" nombre="Lucas"/&gt;   &lt;aseg numgar="3" nombre="Maria"/&gt; &lt;/pol&gt;</pre>

**Ejemplo 3**

<b>Objetivo</b>	<b>Obtener unas estadísticas del envío que contenga el número total de pólizas, asegurados, garantías e importe contratado en vigor</b>
<b>Sentencia</b>	<pre> for \$a in /envio return element resumen{     element numpolizas{         attribute numaseg{count(\$a//asegurado)},         count(\$a//poliza)},     element numgar{         count(\$a//garantia)},     element impVigor{         sum(\$a//garantia[@vigor="S"]/capital)} } </pre>
<b>Resultado</b>	<pre> &lt;resumen&gt;   &lt;numpolizas numaseg="6"&gt;3&lt;/numpolizas&gt;   &lt;numgar&gt;8&lt;/numgar&gt;   &lt;impVigor&gt;270800&lt;/impVigor&gt; &lt;/resumen&gt; </pre>

**Ejemplo 4**

<b>Objetivo</b>	<b>Obtener unas estadísticas del importe contratado por tipo de garantía</b>
<b>Sentencia</b>	<pre> for \$tipogar in distinct-values(//tipo) return &lt;resumengar&gt;   &lt;tipo&gt;{\$tipogar}&lt;/tipo&gt;   &lt;importe&gt;{     sum(//garantia[tipo=\$tipogar]/capital)   }&lt;/importe&gt; &lt;/resumengar&gt; </pre>
<b>Resultado</b>	<pre> &lt;resumengar&gt;   &lt;tipo&gt;Accidentes&lt;/tipo&gt;   &lt;importe&gt;14000&lt;/importe&gt; &lt;/resumengar&gt; &lt;resumengar&gt;   &lt;tipo&gt;Dental&lt;/tipo&gt;   &lt;importe&gt;800&lt;/importe&gt; &lt;/resumengar&gt; &lt;resumengar&gt;   &lt;tipo&gt;Vida&lt;/tipo&gt;   &lt;importe&gt;320000&lt;/importe&gt; &lt;/resumengar&gt; </pre>

**Ejemplo 5**

<b>Objetivo</b>	Obtener un resumen de asegurados por garantías mediante el uso de una función. La función asegTipoGar devuelve una secuencia de asegurados de esa garantía.
<b>Sentencia</b>	<pre> declare function local:asegTipoGar(\$tipo) as element()* {     for \$b in //asegurado[garantia[tipo=\$tipo]]     return &lt;newaseg nomb="{ \$b/@nombre}"         apell="{ \$b/@apellidos}"&gt;         &lt;/newaseg&gt; };  for \$tipogar in distinct-values(//tipo) return &lt;resumengar&gt;     &lt;tipo&gt;{\$tipogar}&lt;/tipo&gt;     {local:asegTipoGar(\$tipogar)} &lt;/resumengar&gt; </pre>
<b>Resultado</b>	<pre> &lt;resumengar&gt;   &lt;tipo&gt;Accidentes&lt;/tipo&gt;   &lt;newaseg nomb="Carlos" apell="Sanchez"/&gt;   &lt;newaseg nomb="Maria" apell="Fernandez"/&gt; &lt;/resumengar&gt; &lt;resumengar&gt;   &lt;tipo&gt;Dental&lt;/tipo&gt;   &lt;newaseg nomb="Carlos" apell="Sanchez"/&gt;   &lt;newaseg nomb="Maria" apell="Fernandez"/&gt; &lt;/resumengar&gt; &lt;resumengar&gt;   &lt;tipo&gt;Vida&lt;/tipo&gt;   &lt;newaseg nomb="Juan" apell="López"/&gt;   &lt;newaseg nomb="Pedro" apell="Martin"/&gt;   &lt;newaseg nomb="Lucas" apell="Montero"/&gt;   &lt;newaseg nomb="Maria" apell="Fernandez"/&gt; &lt;/resumengar&gt; </pre>

## 8 – Actualización de datos

Normalmente los datos son algo vivo y evoluciona con el tiempo, lo cual nos lleva a la necesidad de modificar los mismos.

Hasta ahora hemos visto los modos de consulta de documentos XML, no hemos entrado en los modos de actualización.

La posibilidad más básica es la recuperación un documento, modificarlo mediante un programa específico (a través de algún API como SAX ó DOM) y sustituir al documento original; esta solución es muy costosa (es posible que debamos tratar un fichero de varios megas para actualizar un dato), y nada flexible ya que tendríamos que generar un programa para cada consulta. Esto hace evidente la necesidad de lenguajes que permitan modificar el contenido del mismo de manera declarativa.

Esta es posiblemente la mayor laguna actual en los SGBD nativos, la falta de un lenguaje estándar definido y aceptado para la actualización de documentos XML que obliga a cada fabricante a adoptar una solución propia y a los desarrolladores a la utilización de APIs y lenguajes distintos.

Veamos las distintas opciones que tenemos a nuestro alcance

### **8.1 – XUpdate.**

Es un lenguaje para permitir la actualización de documentos XML, podemos encontrar la definición completa en:

<http://xmldb-org.sourceforge.net/xupdate/index.html>

Los requisitos planteados con este lenguaje son

- Declarativo: Debe definir como consultar y actualizar contenido XML
- Debe ser simple y directo
- Debe de ser compatible con XML Namespaces, XPath y XPointer
- Se permitirán extensiones futuras
- Poseer sintaxis XML
- Debe de ser independiente de cualquier parser o modelo.
- Se debe poder actualizar una parte o el documento completo
- Podrá actuar específicamente con el API XML:DB

Deberá tener las siguientes posibilidades

- Dar posibilidad de consulta básicas como expresión base para la actualización
- Se podrá actualizar, insertar, eliminar contenido del documento
- Se permitirá el uso de operadores lógicos and, or, not

XUpdate utiliza XPath como herramienta de consulta de documentos y de selección condicional de nodos a actualizar.

*(\*) Los ejemplos aquí mostrados están extraídos de la Web oficial de XUpdate, debido a la imposibilidad de probar con la instalación de eXist.*

Una modificación se representa con un elemento del tipo `<xupdate:modifications>` dentro de un documento XML, este elemento debe tener un atributo 'version' obligatorio (para la versión actual debe ser 1.0).

```
<?xml version="1.0"?>
<xupdate:modifications version="1.0"
    xmlns:xupdate="http://www.xmldb.org/xupdate">
    ... ..
</xupdate:modifications>
```

Dentro de este elemento y para efectuar las distintas operaciones de actualización podremos incluir los siguientes elementos:

- `xupdate:insert-before` -> inserta el nodo como `preceding-sibling` del nodo de contexto
- `xupdate:insert-after` -> inserta el nodo como `following-sibling` del nodo de contexto
- `xupdate:append` -> inserta el nodo como `child` del nodo de contexto
- `xupdate:update` -> Actualiza el contenido de los nodos seleccionados
- `xupdate:remove` -> elimina los nodos seleccionados
- `xupdate:rename` -> permite cambiar el nombre de un elemento o atributo

### Inserción de nodos

Como hemos dicho hay tres elementos básicos para la inserción de nodos: `insert-before`, `insert-after` y `append`; los dos primeros insertan el nodo como hermano del nodo contexto (el primero como `preceding-sibling` y el segundo como `following-sibling`) mientras que el tercero inserta el nuevo nodo como hijo del nodo contexto (`child`). Necesitan un atributo `select` que contendrá una sentencia XPath para seleccionar el nodo contexto para la inserción:

Se permiten los siguientes elementos dentro de los tres mencionados arriba:

- `xupdate:element`
- `xupdate:attribute`
- `xupdate:text`
- `xupdate:processing-instruction`
- `xupdate:comment`

Las sentencias de actualización tienen la siguiente forma

```
<xupdate:insert-after select="/addresses/address">
  <xupdate:element name="town">
    San Francisco
  </xupdate:element>
</xupdate:insert-after>
```

```
<xupdate:insert-before select="/addresses/address">
  <xupdate:element name="address">
    <xupdate:attribute name="id">2</xupdate:attribute>
  </xupdate:element>
</xupdate:insert-before >

<xupdate:append select="/addresses" child="last()">
  <xupdate:element name="address">
    <town>San Francisco</town>
  </xupdate:element>
</xupdate:append>
```

### Actualización de contenido

Con `xupdate:update` podemos modificar el contenido de los nodos de contexto seleccionados

Con esta sentencia cambiaríamos la ciudad de la segunda dirección a 'New York'

```
<xupdate:update select="/addresses/address[2]/town">
  New York
</xupdate:update>
```

### Borrado de elemento

Con `xupdate:remove` podemos eliminar los nodos seleccionados

Con esta sentencia eliminaríamos el primer elemento `address` de `addresses`

```
<xupdate:remove select="/addresses/address[1]"/>
```

### Cambio de nombre

Con `xupdate:rename` podemos cambiar el nombre de elementos y atributos creados con anterioridad

Con esta sentencia cambiaríamos el elemento `town` por `city`

```
<xupdate:rename select="/addresses/address/town">
  city
</xupdate:rename>
```

A pesar de tener un comienzo interesante el último borrador es del año 2000, por lo que es un proyecto sin ningún tipo de proyección.

## 8.2 – XQuery Update Facility

Como se ha mencionado antes, la principal carencia del lenguaje XQuery es la falta de definición de sentencias para la actualización de datos. El diseño de XQuery se ha realizado teniendo en cuenta esta necesidad, y XQuery Update Facility 1.0 es una extensión del estándar para cubrirla. Aunque todavía no es un estándar si se encuentra en un estado bastante avanzado y podemos consultarlo en:

<http://www.w3.org/TR/xquery-update-10/>

Divide las expresiones en tres tipos

- non-updating expression: Expresión XQuery normal
- Basic updating expression: Sentencia insert, delete, replace, rename o llamada a función de actualización
- Updating expression: Es una Basic updating expression o cualquier otra expresión que contiene una sentencia de actualización (por ejemplo una sentencia FLWOR con una sentencia update en el return)

*\*Debido a que eXist no soporta XQuery Update, los ejemplos aquí puestos están extraídos del documento proporcionado por el W3C; las pruebas de actualización se realizarán utilizando las sentencias proporcionadas por eXist (ver apartado siguiente)*

### Insert

Esta sentencia inserta copias de 0 o más nodos especificados en origen en las posiciones indicadas en destino. Tanto origen como destino son expresiones de no pueden ser expresiones de actualización.

```
insert---node---- 'origen' -----after----- 'destino'
      |-nodes-|           |--before-----|
                        |-----into--|
                        |--as---first---|
                        |--last--|
```

Origen especifica los datos que vamos a insertar, bien nodos de una expresión bien un literal, destino especifica donde queremos insertar los nodos de origen

- Node/nodes se puede usar indistintamente
- Con before, el nodo se inserta como preceding-sibling del nodo destino, si se insertan varios su orden es mantenido
- Con after, el nodo se inserta como following-sibling del nodo destino, si se insertan varios su orden es mantenido
- Con into el nodo es insertado como hijo del nodo destino, si se insertan varios su orden es mantenido. La posición de inserción será dependiente de la implementación

- Si se añade `as first`, el nodo(s) serán los primeros hijos del nodo destino
- Si se añade `as last`, el nodo(s) serán los últimos hijos del nodo destino

```
insert node <year>2005</year>
  after fn:doc("bib.xml")/books/book[1]/publisher

insert node $new-police-report
  as last into fn:doc("insurance.xml")/policies
    /policy[id = $pid]
    /driver[license = $license]
    /accident[date = $accdte]
    /police-reports
```

## Delete

Con esta sentencia borramos 0 o más nodos

```
delete---node-----`destino`
  |-nodes-|
```

Destino debe de ser una expresión de no-actualización y que tenga como resultado una secuencia de nodos.

```
delete node fn:doc("bib.xml")/books/book[1]/author[last()]

delete nodes /email/message
  [fn:currentDate() - date > xs:dayTimeDuration("P365D")]
```

## Replace

Con esta sentencia reemplazamos una secuencia de 0 o más nodos. En este caso debemos tener en cuenta que podemos reemplazar bien el contenido del nodo bien el nodo mismo.

```
replace-----node---`destino`-with---`expr.valor`
  |-value of-|
```

Tanto como destino como `expr.valor` no pueden ser expresiones de actualización.

- Si no especificamos `value of` vamos a reemplazar un nodo, por lo que el nuevo nodo va a ocupar su posición en la jerarquía; esto implica que un nodo elemento puede ser reemplazado por un nodo elemento, un nodo atributo solo por un nodo atributo etc...

```
replace node fn:doc("bib.xml")/books/book[1]/publisher
with fn:doc("bib.xml")/books/book[2]/publisher
```



```
replace value of node fn:doc("bib.xml")/books/book[1]/price
with fn:doc("bib.xml")/books/book[1]/price * 1.1
```

## Rename

Se utiliza esta sentencia para cambiar la propiedad name con un nuevo valor.

```
rename node 'destino' as 'nombre'
```

Destino tiene que ser una expresión de no actualización y devolver una secuencia de nodos.

```
rename node fn:doc("bib.xml")/books/book[1]/author[1]
as "principal-author"
```

## Uso con las sentencias FLWOR y condicionales

La forma de uso de la sentencia FLWOR no varía, pero hay que tener en cuenta que solo se puede especificar una sentencia de actualización en la cláusula RETURN.

```
for $p in /inventory/part
let $deltap := $changes/part[partno eq $p/partno]
return
  replace value of node $p/quantity
  with $p/quantity + $deltap/quantity
```

Igualmente, en la expresión condicional sólo se admiten sentencias de actualización en las expresiones THEN y ELSE

```
if ($e/@last-updated)
then replace value of node
  $e/last-updated with fn:currentDate()
else insert node
  attribute last-updated {fn:currentDate()} into $e
```

## **8.3 – Extensiones de actualización de eXist**

A falta de un estándar definido y estable para la actualización de documentos XML, eXist ofrece su propia solución, podemos consultar la documentación disponible en:

[http://exist.sourceforge.net/update\\_ext.html](http://exist.sourceforge.net/update_ext.html)

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 (por ejemplo, si es un borrado el nodo no estará disponible)

\* Hay que destacar que estas extensiones están pensadas para la actualización de documentos persistentes, y que si se ejecutan sobre datos temporales en memoria (por ejemplo, una secuencia de nodos recuperados con LET) no tendrá efecto.

Todas las sentencias de actualización comienzan con la partícula UPDATE y a continuación la instrucción:

### Insert

```
update---insert---'origen'---into-----'destino'
                        |-following--|
                        |-preceding--|
```

Con esta sentencia añadimos el contenido de 'origen' en 'destino'. 'Destino' debe de ser una expresión que devuelva una secuencia de nodos.

El lugar de inserción se especifica del siguiente modo:

- into: El contenido se añade como último hijo de los nodos especificados.
- following: El contenido se añade inmediatamente después de los nodos especificados
- preceding: El contenido se añade inmediatamente antes de los nodos especificados

### Replace

```
Update---replace---'destino'---with---'nuevo valor'
```

Sustituye el nodo especificado en 'destino' con 'nuevo valor'. Destino debe devolver un único ítem

- Si es un elemento, 'nuevo valor' debe ser también un elemento
- Si es un nodo de texto ó atributo su valor será actualizado con la concatenación de todos los valores de 'nuevo valor'.

### Value

```
Update---value---'destino'---with---'nuevo valor'
```

Actualiza el valor del nodo especificado en 'destino' con 'nuevo valor'. Si 'destino' es un nodo de texto ó atributo su valor será actualizado con la concatenación de todos los valores de 'nuevo valor'.

### Delete

```
update delete expr
```

Elimina todos los nodos indicados con expr del documento

## Rename

Update---rename---`destino`---as---`nuevo nombre`

Renombra los nodos devueltos por 'destino' (debe devolver una relación de nodos o atributos) usando el valor del primer elemento de 'nuevo nombre'.

## 8.4 – Ejemplos

Los resultados de los ejemplos se darán de manera acumulada, es decir, el resultado del ejemplo 5 supone que se han procesado los ejemplos 1, 2, 3 y 4.

### Ejemplo 1: Inserción

Objetivo	Insertar un nuevo asegurado en la póliza 99000001
Sentencia	<pre>update insert &lt;asegurado nombre="Carmen" apellidos="Requejo"&gt; &lt;/asegurado&gt; into /envio/poliza[numero=99000001]</pre>

### Ejemplo 2: Inserción

Objetivo	Insertar una nueva garantía al primer asegurado de la póliza 99000001 de modo que quede la primera de todas ellas
Sentencia	<pre>update insert &lt;garantia vigor="S"&gt;   &lt;tipo&gt;Invalidez&lt;/tipo&gt;   &lt;capital&gt;15000&lt;/capital&gt; &lt;/garantia&gt; preceding /envio/poliza[numero=99000001]/asegurado[1]/garantia[1]</pre>

Resultado de las inserciones: Después de la actualización, si consultamos la póliza vemos que tiene dos asegurados y que la garantía se ha añadido correctamente al primero

<pre>//poliza[numero=99000001] &lt;poliza externa="S"&gt;   &lt;numero&gt;99000001&lt;/numero&gt;   &lt;tomador&gt;Pedro Martin Gomez&lt;/tomador&gt;   &lt;asegurado nombre="Pedro" apellidos="Martin"&gt;     &lt;garantia vigor="S"&gt;       &lt;tipo&gt;Invalidez&lt;/tipo&gt;       &lt;capital&gt;15000&lt;/capital&gt;     &lt;/garantia&gt;     &lt;garantia vigor="S"&gt;       &lt;tipo&gt;Vida&lt;/tipo&gt;       &lt;capital&gt;80000&lt;/capital&gt;     &lt;/garantia&gt;   &lt;/asegurado&gt;   &lt;asegurado nombre="Carmen" apellidos="Requejo"/&gt; &lt;/poliza&gt;</pre>
--

**Ejemplo 3: Reemplazo**

<b>Objetivo</b>	Sustituir el elemento tomador de la póliza 99000001 por uno nuevo
<b>Sentencia</b>	update replace /envio/poliza[numero=99000001]/tomador with <pagador>nuevo tomador</pagador>

**Ejemplo 4: Reemplazo**

<b>Objetivo</b>	Actualizar el atributo indicador de vigor de la primera garantía del primer asegurado de la póliza 99000001
<b>Sentencia</b>	update replace /envio/poliza[numero=99000001]/asegurado[1]/garantia[1]/@vigor with estado='N'

Resultado: Después de las sustituciones, si consultamos la póliza vemos el tomador ha cambiado por el nuevo, y el indicador de la primera garantía ha cambiado a 'N'

<pre>//poliza[numero=99000001] &lt;poliza externa="S"&gt;   &lt;numero&gt;99000001&lt;/numero&gt;   &lt;pagador&gt;nuevo tomador&lt;/pagador&gt;   &lt;asegurado nombre="Pedro" apellidos="Martin"&gt;     &lt;garantia vigor="N"&gt;       &lt;tipo&gt;Invalidez&lt;/tipo&gt;       &lt;capital&gt;15000&lt;/capital&gt;     &lt;/garantia&gt;     &lt;garantia vigor="S"&gt;       &lt;tipo&gt;Vida&lt;/tipo&gt;       &lt;capital&gt;80000&lt;/capital&gt;     &lt;/garantia&gt;   &lt;/asegurado&gt;   &lt;asegurado nombre="Carmen" apellidos="Requejo"/&gt; &lt;/poliza&gt;</pre>	
---	--

**Ejemplo 5: Actualización**

<b>Objetivo</b>	Actualizar el tipo de la primera garantía del primer asegurado de la póliza 99000001 por Rentas.
<b>Sentencia</b>	update value //poliza[numero=99000001]/asegurado[1]/garantia[1]/tipo with 'Rentas'

**Ejemplo 6: Actualización**

<b>Objetivo</b>	Actualizar el atributo indicador de vigor de la primera garantía del primer asegurado de la póliza 99000001 con un nuevo estado 'T'
<b>Sentencia</b>	update value //poliza[numero=99000001]/asegurado[1]/garantia[1]/@vigor with 'T'

Después de la actualización si nos fijamos en la primera garantía del primer asegurado vemos que es de tipo Rentas y su atributo vigor tiene valor 'T'

```
//poliza[numero=99000001]
<poliza externa="S">
  <numero>99000001</numero>
  <pagador>nuevo tomador</pagador>
  <asegurado nombre="Pedro" apellidos="Martin">
    <garantia vigor="T">
      <tipo>Rentas</tipo>
      <capital>15000</capital>
    </garantia>
    <garantia vigor="S">
      <tipo>Vida</tipo>
      <capital>80000</capital>
    </garantia>
  </asegurado>
  <asegurado nombre="Carmen" apellidos="Requejo"/>
</poliza>
```

### Ejemplo 7: Borrado

<b>Objetivo</b>	Borrar todas las garantías de Rentas de la póliza 99000001
<b>Sentencia</b>	update delete //poliza[numero=99000001]//garantia[tipo='Rentas']

### Ejemplo 8: Borrado

<b>Objetivo</b>	Borrar el apellido de los asegurados con nombre Carmen
<b>Sentencia</b>	update delete //poliza[numero=99000001]/asegurado[@nombre='Carmen']/@apellidos

Después de la actualización observamos que el asegurado Carmen ya no tiene apellido y que el primer asegurado ya no tiene la garantía de rentas

```
//poliza[numero=99000001]
<poliza externa="S">
  <numero>99000001</numero>
  <pagador>nuevo tomador</pagador>
  <asegurado nombre="Pedro" apellidos="Martin">
    <garantia vigor="S">
      <tipo>Vida</tipo>
      <capital>80000</capital>
    </garantia>
  </asegurado>
  <asegurado nombre="Carmen"/>
</poliza>
```

### Ejemplo 9: Cambio de nombre

<b>Objetivo</b>	Cambiar el nombre a los elementos capital de la poliza 99000001 por importe
<b>Sentencia</b>	update rename //poliza[numero=99000001]//capital as 'importe'

**Ejemplo 10: Cambio de nombre**

<b>Objetivo</b>	<b>Cambiar el nombre del atributo vigor por estado</b>
<b>Sentencia</b>	<code>update rename //poliza[numero=99000001]//garantia/@vigor as 'estado'</code>

Después de la actualización observamos que ha cambiado el nombre del elemento y ahora es importe, que el nombre del atributo ha cambiado a estado

```
//poliza[numero=99000001]
<poliza externa="S">
  <numero>99000001</numero>
  <pagador>nuevo tomador</pagador>
  <asegurado nombre="Pedro" apellidos="Martin">
    <garantia estado="S">
      <tipo>Vida</tipo>
      <importe>80000</importe>
    </garantia>
  </asegurado>
  <asegurado nombre="Carmen"/>
</poliza>
```

## 9 – Acceso desde programas

En esta sección vamos a ver las distintas opciones que existen a la hora de acceder al SGBD y recuperar la información almacenada en el mismo; no entraremos a estudiar ni probar las APIs del tratamiento de documentos XML (SAX, DOM); es decir, nos ocuparemos del modo de acceder al SGBD y recuperar o generar documentos XML, pero no del procesamiento de los mismos.

### 9.1 – Métodos de acceso

Cada fabricante de SGBD proporciona distintos métodos de acceso a los datos almacenados en el mismo, vamos a enumerar algunos de ellos (centrándonos en las opciones proporcionadas por eXist):

#### **XML-RPC**

Es un protocolo simple de llamada a métodos remotos (Remote Procedure Call), que usa como protocolo de transporte HTTP y XML como método de codificación de mensajes (es una petición HTTP POST cuyo cuerpo es XML). Podemos encontrar la definición completa en:

<http://www.xmlrpc.com/>

Un ejemplo de llamada a un método simple sin parámetros usando XML-RPC

```
<?xml version='1.0'?>
<methodCall>
  <methodName>title_or_id</methodName>
  <params>
    <param>
      <value><i4>40</i4></value>
    </param>
  </params>
</methodCall>
```

Y obtendríamos la siguiente respuesta

```
<?xml version='1.0'?>
<methodResponse>
  <params>
    <param>
      <value><string>xml basics</string></value>
    </param>
  </params>
</methodResponse>
```

Podemos encontrar distintas implementaciones libres en distintos lenguajes para facilitarnos el uso de este protocolo, el servidor RPC de eXist está implementado usando las librerías creadas por Hannes Wallnoefer las cuales forman ahora en el proyecto Apache (<http://ws.apache.org/xmlrpc/>).

Podemos ver una referencia completa de las funcionalidades ofrecidas por eXist a través de RPC en

<http://exist.sourceforge.net/api/org/exist/xmlrpc/RpcAPI.html>

## SOAP

Es un estándar que especifica el modo en que pueden comunicarse dos objetos en dos procesos distintos ó en máquinas distintas. A diferencia de CORBA o DCOM el formato de intercambio de mensajes es XML, y se puede utilizar cualquier protocolo, aunque generalmente se usará HTTP. Se puede pensar en SOAP como una evolución de XML-RPC.

A diferencia del anterior SOAP si es un estándar soportado por el W3C; excede de los límites del documento el estudio de este protocolo, aunque puede encontrarse referencia completa del mismo en:

<http://www.w3.org/TR/soap/>

Una llamada soap tiene el siguiente aspecto

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getBookTittle>
      <ref>ABD343</ref>
    </getBookTittle>
  </soap:Body>
</soap:Envelope>
```

## Obteniendo una respuesta

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getBookTittleResponse>
      <getBookTittleResult>
        <tittle>XML Basics</tittle>
      </getBookTittleResult>
    </ getBookTittleResponse>
  </soap:Body>
</soap:Envelope>
```

eXist ofrece funcionalidad SOAP haciendo uso del Axis SOAP toolkit de Apache, y con la instalación se ofrecen dos Web Services para la consulta y actualización de documentos

`http://<host>:8080/exist/services/Query`

`http://<host>:8080/exist/services/Admin`

Puede consultarse los servicios completos ofrecidos por estos servicios en

<http://exist.sourceforge.net/api/org/exist/soap/Query.html>

<http://exist.sourceforge.net/api/org/exist/soap/Admin.html>



## Web-DAV

WebDAV es una extensión del protocolo HTTP para permitir el manejo y edición de archivos en sitios Web (Web Distributed Authoring and Versioning). Podemos encontrar información completa sobre el protocolo en:

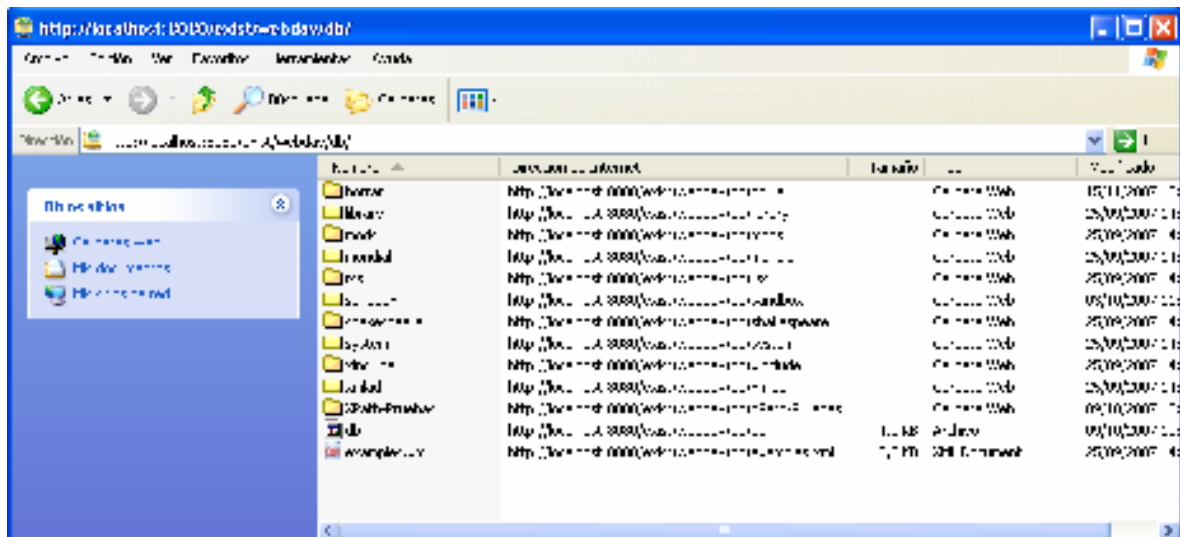
<http://www.webdav.org/>

<http://www.ignside.net/man/servidores/webdav.php>

La implementación de WebDAV de eXist por defecto es accesible desde:

`http://<host>:8080/exist/webdav/db`

El siguiente documento (<http://exist.sourceforge.net/webdav.html>) nos muestra como acceder a eXist utilizando WebDAV, incluso configurándolo como un sitio de red dentro del explorador de archivos de Windows. En la imagen podemos ver como aparece una conexión desde el explorador contra la base de datos



Las colecciones aparecen como carpetas (marcadas como carpeta Web), pudiendo copiar y mover archivos, al igual que ver su contenido.

## Librerías específicas del lenguaje

Dependiendo del producto podremos disponer de librerías implementadas sobre los lenguajes más populares (Java, .Net, C, JavaScript), a partir de aquí nos centraremos en las posibilidades existentes para el lenguaje Java.

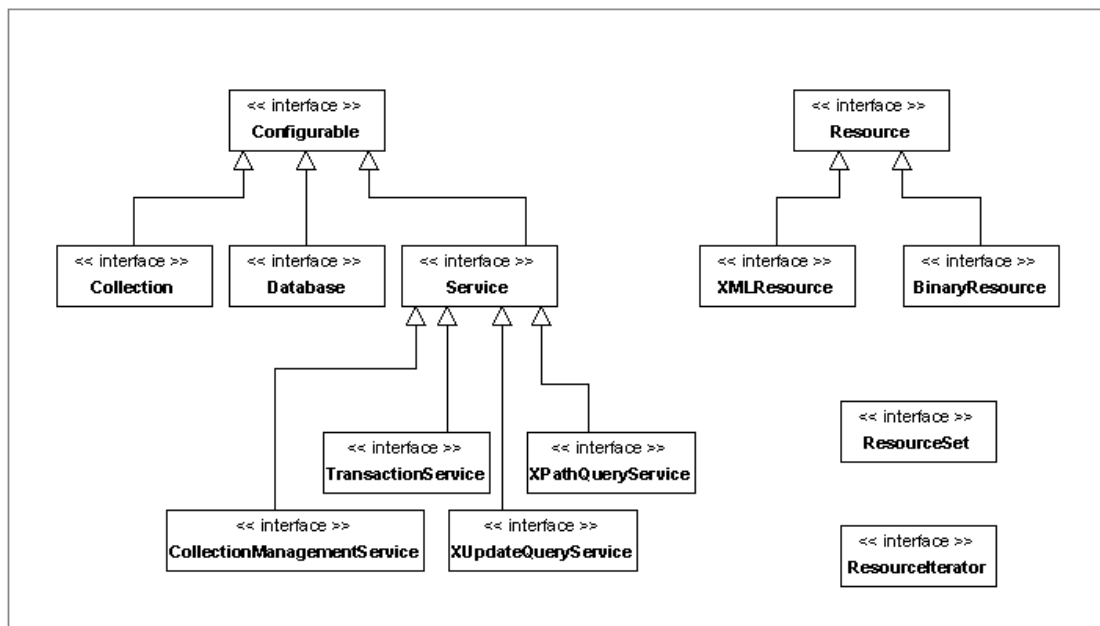
A la hora de acceder a bases de datos XML nativas desde programas Java aun no existe un estándar definido. Existen varias propuestas de estandarización, pero ninguna de ellas está todavía completa ni reconocida, por lo que de momento hay que remitirse a los drivers de acceso que proporcione el fabricante (generalmente soportará alguno de los estándares mencionados a continuación en algún grado). En los puntos siguientes veremos cuales son las alternativas más importantes

## 9.2 – XML:DB.

El objetivo de XML:DB es la definición de un método común de acceso a SGBD XML, permitiendo la consulta, creación, consulta y modificación de contenido. Como objetivo tiene el ser modular, y para ello los módulos del API están definidos en conjuntos que forman un nivel de la especificación. Los módulos que conforman cada nivel son los siguientes:

- Core Level 0
  - API Base
    - Configurable
    - Collection
    - Database
    - Resource
    - ResourceIterator
    - ResourceSet
    - Service
  - XML Resource
- Core Level 1 (incluye Core Level 0)
  - XPathQueryService

Aunque existen definiciones de más componentes, la jerarquía especificada disponible hasta el momento es la siguiente



La estructura de XML:DB gira en torno a los siguientes componentes básicos

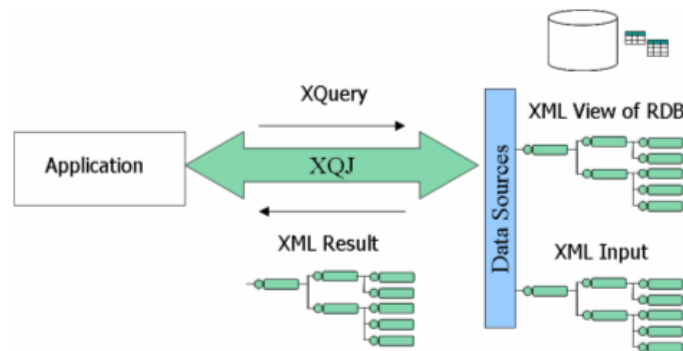
- Driver: Encapsula la lógica de acceso a una base de datos determinada.
- Collection: Contenedor jerárquico para recursos y otras colecciones.
- Resource: Archivo de datos, hay dos tipos
  - XMLResource: Representa un documento XML o parte de un documento obtenido con una consulta.
  - BinaryResource
- Service: Implementación de una funcionalidad que extiende el núcleo de la especificación.

Puede obtenerse la definición disponible del estándar en

<http://xmldb-org.sourceforge.net/xapi/xapi-draft.html>

La última actualización del working draft es del año 2001 y la actividad desde el año 2005 es prácticamente nula, por lo que aparentemente es un estándar con corto recorrido.

### 9.3 – XQJ.



XQJ es una propuesta de estandarización de interfaz java para el acceso a bases de datos XML nativas (el último borrador es de Agosto de 2007) basado en el modelo de datos XQuery; el objetivo es el mismo que se perseguía con JDBC: un método común, sencillo y estable de acceso a SGBD XML nativos.

A diferencia de XML:DB, el proyecto XQJ está muy activo, y se encuentra con el respaldo de SUN.

Los objetivos de la definición de este estándar son

- Estándar independiente del fabricante.
- Soporte al estándar XQuery 1.0
- Facilidad de uso.
- Potenciar el uso de esta tecnología en la comunidad Java.

El borrador del estándar puede encontrarse en:

<http://jcp.org/aboutJava/communityprocess/pr/jsr225/index.html>

Al igual que en JDBC la filosofía gira en torno al origen de datos y la conexión a este, y partiendo de la conexión poder lanzar peticiones al sistema

Una implementación XQJ 1.0 compatible tiene que implementar como mínimo las siguientes interfaces

- XQDataSource
- XQConnection
- XQExpression
- XQPreparedExpression
- XQDynamicContext
- XQStaticContext
- XQItem
- XQResultItem
- XQSequence
- XQResultSequence
- XQItemAncestor
- XQMetaData
- XQItemType
- XQSequenceType
- XQDataFactory

*(\*)Los ejemplos mostrados aquí están obtenidos del borrador del estándar indicado más arriba, no es posible hacer unas pruebas prácticas ya que aunque eXist está implementando su driver XQJ, aun no está disponible)*

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

```
VendorXQDataSource vds = new VendorXQDataSource();
vds.setServerName("my_database_server");
vds.setUser("john");
vds.setPassword("topsecret");
XQConnection conn = vds.getConnection();
```

XQConnection: representa una sesión con la base de datos, manteniendo información de estado, transacciones, expresiones ejecutadas y resultados. Se obtiene a través de un XQDataSource

```
XQDataSource ds = ...;
XQConnection conn = ds.getConnection();
```

XQStaticContext: representa el contexto estático para la ejecución de expresiones en esa conexión. Se puede obtener el contexto estático por defecto a través de la conexión. Si se efectúan cambios en el contexto estático no afecta a las expresiones ejecutándose en ese momento, solo en las creadas con posterioridad a la modificación. También es posible especificar un contexto estático para una expresión en concreto de modo que ignore el contexto de la conexión.

```
XQConnection conn = xqds.getConnection();
XQStaticContext cntxt = conn.getStaticContext();
System.out.println(cntxt.getBaseURI());
```

**XQDynamicContext**: Representa el contexto dinámico de una expresión, como puede ser la zona horaria y las variables que se van a utilizar en la expresión.

**XQExpression**: Objeto creado a partir de una conexión para la ejecución de una expresión una vez, retornando un `XQResultSetSequence` con los datos obtenidos, podemos decir que en un paso se evalúa el contexto estático o expresión y el dinámico. La ejecución se produce llamando al método `executeQuery`, y se evalúa teniendo en cuenta el contexto estático en vigor en la creación del objeto y de las variables ligadas con los métodos `bindXXX`.

```
XQExpression expr = conn.createExpression();
String es = "declare variable $i as xs:integer external; " +
"$i + 1 ";
expr.bindInt(new QName("i"), 21, (XQType)null);
XQResultSetSequence result = expr.executeQuery(es);
```

**XQPreparedExpression**: Objeto creado a partir de una conexión para la ejecución de una expresión múltiples veces, retornando un `XQResultSetSequence` con los datos obtenidos, en este caso la evaluación del contexto estático sólo se hace una vez, mientras que el proceso del contexto dinámico se repite. Igual que en `XQExpression` la ejecución se produce llamando al método `executeQuery`, y se evalúa teniendo en cuenta el contexto estático en vigor en la creación del objeto y de las variables ligadas con los métodos `bindXXX`.

```
String es = "declare variable $x as xs:integer external;" +
" for $n in fn:doc('catalog.xml')//item" +
" where $n/price <= $x" +
" return fn:data($n/name)";
XQPreparedExpression expr = conn.prepareExpression(es);
```

**XQItem**: Representación de un elemento en XQuery. Es inmutable y una vez creado su estado interno no cambia.

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

**XQSequence**: Representación de una secuencia del modelo de datos XQuery, contiene un conjunto de 0 o más `XQItem`. Es un objeto recorrible.

**XQResultSequence**: Resultado de la ejecución de una sentencia; contiene, contiene un conjunto de 0 o más `XQResultItem`. Es un objeto recorrible

#### **9.4 – Implementación de eXist.**

Los drivers proporcionados actualmente por eXist para acceder a su base de datos soportan el estándar XML:DB en todos sus niveles definidos, ampliando notablemente sus funcionalidades con un conjunto de librerías bastante extenso, de la cual encontramos referencia en:

<http://exist.sourceforge.net/api/index.html>

Esté método de acceso será sustituido por unos drivers que soporten el estándar XQJ, pero en este momento están bajo desarrollo y no son accesibles para su prueba.

En el capítulo siguiente profundizaremos en el uso de las librerías XML:DB de eXist realizando una aplicación de ejemplo.

## 10 – Aplicación de ejemplo

### Objetivo

El objetivo del desarrollo de este punto es familiarizarse y experimentar con el API XML:DB proporcionado por eXist.

Para ello vamos a desarrollar una aplicación capaz de hacer una gestión básica de los recursos de la base de datos; las funcionalidades de la misma son las siguientes:

- Recuperar y mostrar la jerarquía de colecciones y documentos almacenados en la BB.DD
  - Para los recursos XML se mostrará su contenido
- Gestionar colecciones
  - Añadir una colección
  - Borrar una colección
- Gestionar recursos
  - Añadir un recurso (catalogándolo como binario ó XML)
  - Borrar un recurso
- Consultar los usuarios existentes en la base de datos.
- Permitir la ejecución de consultas XPath/XQuery contra la base de datos (relativas a la colección seleccionada en ese momento)

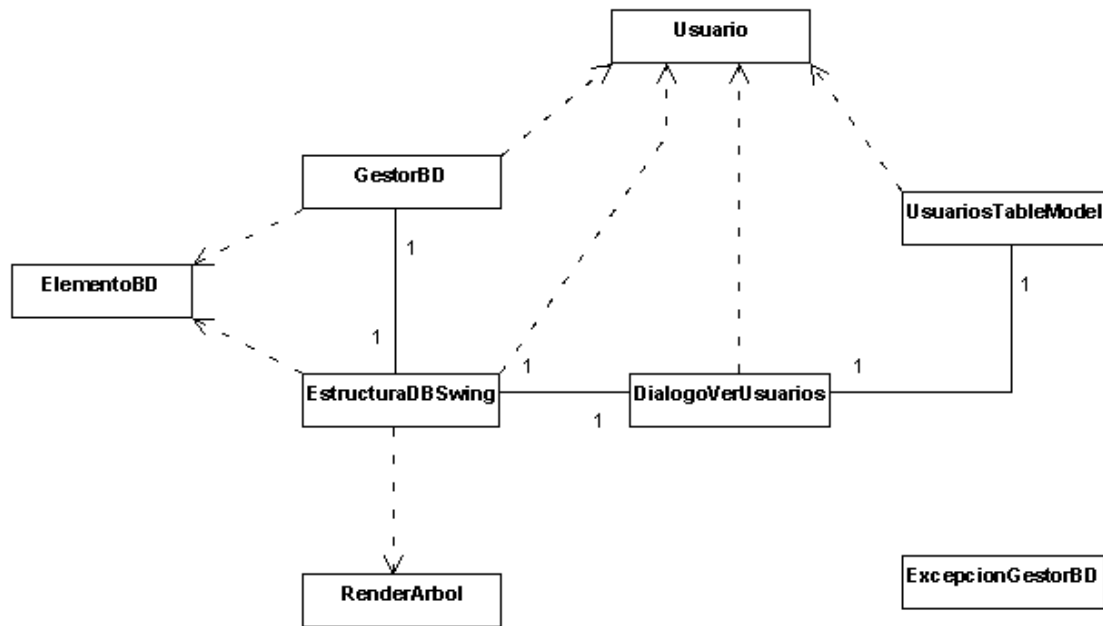
### Herramientas utilizadas

Para el desarrollo de la aplicación se han utilizado las siguientes herramientas

- Java JDK 1.5.0\_09
- eXist 1.1.1
- IDE Eclipse 3.2.2

### Diagrama de clases

La estructura de clases es la siguiente.



*ExcepciónGestorBD.java*

Clase para encapsular las excepciones que se produzcan en el acceso a la base de datos.

*ElementoBD.java*

Representa un ítem que está almacenado en la base de datos, guardando los datos que nos son relevantes (nombre, path y tipo de recurso).

*Usuario.java*

Representa un usuario registrado en la base de datos, manteniendo atributos que necesitamos: nombre, clave, grupo principal, grupos asociados e indicador de administrados.

*GestorBD.java*

Clase principal que encapsula todos los accesos a la base de datos; las aplicaciones que realicen funciones de interfaz con el usuario no accederán a la base de datos, siempre solicitarán los servicios a GestorBD.

La localización de la conexión así como el usuario con el que se ejecuta la lectura de colecciones están fijados como constantes, por lo que se restringe para la correcta ejecución que la aplicación y el SGBD estén en la misma máquina y que el usuario “admin” esté desprotegido (tal y como queda después de una instalación normal).



### *EstructuraBDSwing.java*

Interfaz de la aplicación. Es la interfaz gráfica que usará GestorBD para obtener los datos de la BD y los mostrará en pantalla.

### *RenderArbol.java*

Clase de apoyo encargada de controlar el aspecto estético del JTree usado en EstructuraBDSwing para mostrar la estructura de la base de datos.

### *UsuariosTableModel.java*

Clase de apoyo encargada de controlar el aspecto y contenido de la tabla de presentación de usuarios.

### *DialogoVerUsuarios.java*

Clase que gestiona la ventana de consulta de usuarios.

## **Descripción de las clases**

*(\*)El código completo de las clases puede encontrarse en el Anexo 9*

### **ExcepciónGestorBD.java**

Clase para encapsular las excepciones que se produzcan en el acceso a la base de datos y generar errores personalizados. Deriva de la clase Exception, y tiene dos constructores

```
public ExcepcionGestorBD ()  
public ExcepcionGestorBD (String pTexto)
```

El primero es el constructor por defecto, con un mensaje genérico, y el segundo es el constructor parametrizado para generar un error personalizado

### **ElementoBD.java**

Representa los ítems que pueden estar almacenados en la BD, guardando los datos que nos son relevantes (nombre, tipo y colección). Posee un único constructor con 3 parámetros

```
public ElementoBD(String nombre, int tipo, String coleccion)
```

nombre: nombre del recurso

tipo: Identificador del tipo de recurso

colección: colección donde está almacenado el recurso

Para identificar el tipo de recurso se definen las siguientes constantes

```
public static final int OTRO = 0;
public static final int COLECCION = 1;
public static final int RECURSO_XML = 21;
public static final int RECURSO_BINARIO = 22;
```

Una vez creados no se pueden modificar, por tanto solo existen métodos de lectura

```
public String getNombre()
public String getColeccion()
public int getTipo()
```

Se define también la función `public String getPathCompleto()` que retorna el URI del elemento: cadena de conexión (definida en GestorBD) + jerarquía de colecciones + recurso. También se ha sobrescrito el método `toString` de este modo

```
public String toString(){
    try {
        return URLDecoder.decode(nombre, "UTF-8");
    } catch (UnsupportedEncodingException e) {
        return "--error normalizando nombre--";
    }
}
```

Se utiliza la función `URLDecoder.decode()` por motivos estéticos, ya que el SGBD devuelve los nombres de los recursos expandidos, es decir, si una colección se llama 'parte1 parte2', la función que obtiene su nombre retorna 'parte1%20parte2'.

## Usuario.java

Representa un usuario de la base de datos. Los datos que almacena del usuario son

- Nombre
- Password
- Indicador de administrador
- Grupo primario
- Lista de todos los grupos asignados

Igual que en elemento, no es modificable, por lo que solo existen métodos de lectura; para su creación solo se contempla un constructor parametrizado completo.

```
public Usuario(String nombre, String password, String grupoPrimario,
               boolean esDBA, String[] grupos)
```

## RenderArbol.java

Es una clase auxiliar que sirve para adaptar la presentación del componente JTree utilizado para mostrar la estructura de la base de datos, asignando un icono para una colección (tanto abierta como cerrada) y otro icono para un documento. Implementa el método

```
public Component getTreeCellRendererComponent
```

Cuando este método es llamado devuelve un Component (JLabel en este caso) que será utilizado para dibujarlo en el árbol. En función del tipo de elemento asignaremos un icono u otro (si es una colección además asignaremos un icono distinto si está abierta o cerrada).

Como texto se aplica el valor devuelto por el método toString del elemento; igualmente si el elemento está seleccionado se asigna un color de texto distinto para resaltarlo.

## UsuarioTableModel.java

Es una clase que sirve para controlar el aspecto y contenido de la tabla de usuarios.

El constructor tiene como parámetro un ArrayList que deberá contener la lista de usuarios a mostrar; los métodos implementados son

```
Definir contenido de cabeceras: public String getColumnName(int col)
Indicar el número de columnas: public int getColumnCount()
Indicar el número de filas: public int getRowCount()
Obtener el valor de cada celda: public Object getValueAt(int arg0, int arg1)
```

## GestorBD.java

Esta es la clase principal de la aplicación, y es la encargada de mediar entre la interfaz y el SGBD. En ella se crea la conexión con la base de datos al instanciarla..

Define tres atributos que definen la localización del SGBD; están inicializados con un valor fijo, (usuario con el valor 'admin' y usuarioPwd como una cadena vacía " y URI para la conexión) pero puede fácilmente modificarse para que esta inicialización sea dinámica (desde un fichero o desde un dialogo)

```
public static String URI = "xmldb:exist://localhost:8080/exist/xmlrpc";
private String usuario;
private String usuarioPwd;
```

Ofrece los siguientes servicios o funciones

### -obtenerEstructuraColeccion

Esta función devuelve la estructura de una colección (colecciones y documentos que contiene), para ello tiene un parámetro String donde le damos la ruta de la misma.

```
public DefaultMutableTreeNode obtenerEstructuraColeccion(String
colec) throws ExcepcionGestorBD
```

Si este parámetro es null, por defecto cargará la colección raíz (definida por la constante `DBBroker.ROOT_COLLECTION`).

Debido a la naturaleza jerárquica del contenido se devuelve un árbol de elementos (de hecho se devuelve el nodo raíz del árbol) usando el componente estándar de Java `DefaultMutableTreeNode`. Cada nodo de ese árbol contiene un `ElementoDB` que representa a cada uno de los elementos de esa colección. Es una función recursiva, y se llama a si misma con cada subcolección que encuentra.

Debido al carácter ilustrativo de este ejemplo, se carga la estructura entera de la colección en el árbol en memoria; esto puede ser un inconveniente si la base de datos es muy grande, por lo que en una aplicación comercial el tratamiento debería de ser distinto, por ejemplo limitando el número de subniveles a cargar.

### -leerColeccion

Esta función recibe como parámetro un String con la ruta de una colección y devuelve un objeto `Collection`.

```
public Collection leerColeccion(String colec)
```

Para leer la colección se utiliza el método `getCollection` de la clase `DatabaseManager`

```
colRet = DatabaseManager.getCollection(URI + colec,
                                     usuario,
                                     usuarioPwd);
```

Este método tiene tres cadenas como parámetros, la primera es el localizador de la colección (obtenido con el URI definido en la clase mas la colección a leer), el atributo `usuario` y el atributo `usuarioPwd`.

Si no encuentra la colección especificada devuelve null.

### -leerRecurso

Esta función recibe como parámetros una colección (que podemos obtener con la función anterior) y el nombre del recurso dentro de ella que queremos leer.

```
public Resource leerRecurso(Collection colec, String nombrerec
```

Para efectuar la lectura deberemos llamar al método `getResource` de la colección dándole como parámetro el nombre del recurso

```
res = (Resource) colec.getResource(nombrerec);
```

Si no encuentra el recurso dentro de la colección devuelve un nulo.

### -ejecutarQuery

Esta función se encarga de lanzar una consulta contra la base de datos; recibe dos parámetros, un `String` con la consulta y un segundo `String` con el contexto (colección sobre la que vamos a ejecutar la consulta). Si todo es correcto retorna un `ResourceSet` con los resultados.

```
public ResourceSet ejecutarQuery(String consulta, String contexto)
                                throws ExcepcionGestorBD
```

Al igual que la función `obtenerEstructuraColección`, si como contexto se le pasa es nulo sobreentiende que el contexto de la consulta es la colección raíz del gestor.

Para ejecutar la query, a través de la colección usada como contexto usaremos el servicio requerido, `XQueryService`:

```
XQueryService service =
    (XQueryService) col.getService( "XQueryService", "1.0" );
```

Una vez obtenido el servicio establecemos las propiedades de funcionamiento

```
service.setProperty( OutputKeys.INDENT, "yes" );
service.setProperty( OutputKeys.ENCODING, "UTF-8" );
```

Ahora ya estamos en condiciones de realizar la consulta; para ello tenemos dos métodos, el primero de ellos obtener una `CompiledExpression`, la cual podremos utilizar en varias consultas (este es el método usado en el ejemplo)

```
CompiledExpression compiled = service.compile( "-consulta-" );
result = service.execute( compiled );
```

O ejecutando directamente la consulta si no va a ser reutilizada

```
result = service.query( consulta );
```

Como valor de retorno se obtiene un `ResourceSet` con los elementos devueltos por la consulta.

### -anadirColeccion

Esta función se encarga de añadir una colección a la base de datos, para ello recibe dos parámetros, una Collection, que actuará de contexto y dentro de la cual se insertará la nueva colección y un String con el nombre de la nueva colección.

```
public Collection anadirColeccion(Collection contexto,
                                  String newColec)
    throws ExcepcionGestorBD{
```

Para añadir la colección primero recuperamos de la colección de contexto el servicio requerido (CollectionManagementService), una vez hecho esto llamamos a la función createCollection

```
CollectionManagementService mgtService =
    (CollectionManagementService)contexto.getService(
        "CollectionManagementService",
        "1.0");
newCollection = mgtService.createCollection(
    new String(UTF8.encode(newColec)));
```

A la hora de crear la nueva colección llamando a la función createCollection, le pasamos el nombre recibido convirtiéndolo mediante la función UTF8.encode, de modo que sustituya los caracteres problemáticos (como por ejemplo el espacio en blanco) por una URI correctamente formada.

### -borrarColeccion

Esta es la función encargada de borrar una colección, al igual que la función anadirColección recibe como parámetros una Colección que actuará como contexto de la función y un String con el nombre de la colección a borrar.

```
public void borrarColeccion(Collection contexto,String antColecc)
    throws ExcepcionGestorBD{
```

Recuperamos el servicio de la colección de contexto (CollectionManagementService), y después llamamos a la función removeCollection

```
CollectionManagementService mgtService =
    (CollectionManagementService)contexto.getService(
        "CollectionManagementService",
        "1.0");
mgtService.removeCollection(antColecc);
```

En este caso no se utiliza la función UTF8.encode porque se entiende que el parámetro ya viene en la forma correcta

-anadirRecurso

Esta función es la encargada de añadir un recurso, recibe como parámetro una colección que hará de contexto donde añadiremos el nuevo fichero; recibe también un objeto File con el archivo a añadir y un entero que especifica como será catalogado el recurso, como binario ó xml (Se admiten los valores ElementoBD.*RECURSO\_BINARIO* ó ElementoBD.*RECURSO\_XML*)

```
public void anadirRecurso(Collection contexto, File archivo,
                          int tipoRecurso)
    throws ExcepcionGestorBD{
```

Deberemos crear primero un Resource nuevo, para ello utilizaremos el nombre del archivo y una cadena indicando el tipo (asignada en función del tipo de recurso especificado en el parámetro), que puede tomar los valores "BinaryResource" ó "XMLResource". Una vez creado especificaremos que su contenido es el fichero que recibe como parámetro

```
Resource nuevoRecurso =
    contexto.createResource(archivo.getName(),
                           tipoRecursoStr);
nuevoRecurso.setContent(archivo);
```

En este caso no hace falta obtener ningún servicio, y se llama al método createResource de la colección de contexto.

```
contexto.storeResource(nuevoRecurso);
```

\*Remarcar que si especificamos que el recurso es del tipo XML y el fichero no tiene formato XML se producirá una excepción.

-borrarRecurso

Con esta función eliminaremos un recurso de la base de datos, para ello recibimos una colección, que será el contexto de la cual eliminaremos un recurso y el nombre del recurso a eliminar.

```
public void borrarRecurso(Collection contexto, String nombreRec)
    throws ExcepcionGestorBD{
```

Igual que en el caso anterior no hace falta recuperar ningún servicio ya que la función removeResource lo provee la misma colección; sencillamente leeremos el recurso a borrar y se lo pasaremos a la función de borrado.

```
Resource recursoParaBorrar = leerRecurso(contexto, nombreRec);
contexto.removeResource(recursoParaBorrar);
```

### -leerUsuarios

Con esta función recuperamos los usuarios registrados en la base de datos; no tiene parámetros, y devuelve un ArrayList relleno con objetos Usuario

```
public ArrayList leerUsuarios() throws ExcepcionGestorBD{
```

Para leer los usuarios debemos obtener el servicio de gestión de usuarios (UserManagementService) a través de una colección, para lo que usaremos la colección raíz del sistema, una vez obtenido el servicio deberemos llamar al método getUsers que nos devuelve un array con todos los usuarios.

```
Collection col=
    (CollectionImpl) leerColeccion(DBBroker.ROOT_COLLECTION);
UserManagementService service =
    (UserManagementService) col.getService(
        "UserManagementService", "1.0");
User[] usrEnBD = service.getUsers();
```

### DialogoVerUsuario.java

Esta clase gestiona el diálogo que muestra la lista de usuarios. En el constructor recibe un ArrayList con todos los usuarios que mostrará y su ventana padre.

```
public DialogoVerUsuarios(JFrame pantPadre, final ArrayList
listaUsuarios)
```

Desde esta clase, y con el ArrayList recibido creamos el UsuarioTableModel que utilizaremos para configurar la tabla de contenido; además y como punto más destacable es el gestor de evento para la pulsación del ratón sobre la tabla, que actualizará el contenido del área de texto inferior con los grupos que tiene asociado el usuario seleccionado.

```
tablaUsuarios.addMouseListener(new MouseAdapter() {
    public void mouseClicked(MouseEvent e) {
```

El gestor de acción del único botón de la pantalla sencillamente llama al método dispose para cerrar el diálogo.



## EstructuraBDSwing.java

### -Gestor de selección de elemento en el árbol de estructura

```
arbolEstruc.addTreeSelectionListener(  
    new javax.swing.event.TreeSelectionListener() {  
        public void valueChanged(javax.swing.event.TreeSelectionEvent accion)
```

Cuando se pincha sobre un elemento dentro del árbol de estructura se lanza este método; lo que hacemos es leer el elemento de dicho nodo y obtenemos la colección del elemento seleccionado y en función del tipo que sea actuamos de los siguientes modos

- Colección: Muestra el nombre y la ruta completa de la misma
- Recurso binario: Muestra la colección en la que está y el nombre del recurso
- Recurso xml: Leemos el recurso llamando a GestorBD.leerRecurso, mostrando el contenido en el panel derecho.

### -Gestor de pulsación en botón ejecutar query

```
botonEjQuery.addActionListener(new java.awt.event.ActionListener() {  
    public void actionPerformed(java.awt.event.ActionEvent accion)
```

Este método llama al método GestorBD.ejecutarQuery. La query que se le pasa a la función es la que esté tecleada en ese momento en el área de consultas (se verifica que esté informada, y como contexto para la consulta se pasa la colección seleccionada ó la colección donde se encuentra el elemento seleccionado).

Si no existiese ningún elemento seleccionado, como contexto se pasará un nulo, lo que hará que la consulta se efectúe sobre la colección raíz de la BB.DD

### -Gestor de pulsación en botón añadir recurso

```
botonAnadirRecur.addActionListener(new  
java.awt.event.ActionListener() {  
    public void actionPerformed(java.awt.event.ActionEvent accion)
```

Si el botón está activo (hay una colección seleccionada, en la cual se insertará el nuevo recurso) se llama al método dialogoAnadirRecurso, del cual recuperaremos el fichero a añadir a la BB.DD; si en el dialogo se pulsa cancelar devuelve nulo y no se hace nada, si recibimos un fichero llamamos al método dialogoSeleccionTipoRecurso para saber como debemos catalogarlo en la base de datos.

Si el valor recibido es correcto se llama al método anadirRecurso de GestorBD, pasándole como contexto la colección seleccionada en este momento, el archivo a añadir y como lo catalogaremos.

Como último paso actualizamos el árbol de contenido añadiendo un nodo correspondiente al documento que acabamos de añadir, evitando así tener que recargar la estructura completa de nuevo.

### -Gestor de pulsación en botón borrar recurso

```
botonBorrarRecur.addActionListener(new  
java.awt.event.ActionListener() {  
    public void actionPerformed(java.awt.event.ActionEvent accion)
```

Si el botón está activo (es decir, si existe un recurso seleccionado) se llama a dialogoConfirmacionBorrado para confirmar la acción; en caso de que la respuesta sea afirmativa se llama a GestorBD.borrarRecurso, dando como colección de contexto la colección del recurso a borrar y el nombre del recurso. Si el proceso es correcto se actualizará el árbol de contenido, eliminando el nodo correspondiente al recurso y llamando a la función updateUI del árbol para refrescar el árbol.

Además, como hemos eliminado el recurso seleccionado ya no existe ninguno seleccionado, por lo que se llama al método activarControles con el parámetro ElementoBD.OTRO para deshabilitar todas las opciones, inicializando también las variables nodoSelec y elemSelec.

### -Gestor de pulsación en botón añadir colección

```
botonAnadirColecc.addActionListener(  
    new java.awt.event.ActionListener() {  
        public void actionPerformed(java.awt.event.ActionEvent accion)
```

Si el botón está activo (hay una colección seleccionada, en la cual se insertará una nueva colección hija) se muestra una ventana para capturar el nombre de la nueva colección (dialogoAnadirColeccion). Se verifica que no se haya pulsado cancelar y que el nombre esté informado (longitud > 0). Si se cumplen estas condiciones se llama al método GestorBD.anadirColeccion, pasando como contexto la colección seleccionada y el nombre recogido.

Se añade un nuevo elemento al árbol para que esté disponible sin tener que recargar la estructura entera de nuevo.

### -Gestor de pulsación en botón borrar colección

```
botonBorrarColecc.addActionListener(  
    new java.awt.event.ActionListener() {  
        public void actionPerformed(java.awt.event.ActionEvent accion)
```

Si el botón está activo (es decir, si existe una colección seleccionada) se llama a dialogoConfirmacionBorrado para confirmar la acción; en caso de que la respuesta sea afirmativa se obtiene la colección padre de la colección que queremos borrar, esto tiene un doble sentido, bloquear el borrado de la colección padre y obtener la colección de contexto para llamar al método de borrado (GestorBD.borrarColeccion).

Concluido el borrado se elimina del árbol el nodo correspondiente a la colección eliminada (y por extensión todos sus hijos).

### -Gestor de pulsación en botón ver usuarios

```
botonVerUsuarios.addMouseListener(  
    new java.awt.event.MouseAdapter() {  
        public void mouseClicked(java.awt.event.MouseEvent accion)
```

En este método se obtiene la lista de usuarios del sistema llamando a GestorBD.listaUsuarios para después llamar al método dialogoVerUsuarios (pasándole como parámetro la lista recuperada).

### -Gestor de pulsación en botón salir

```
botonSalir.addMouseListener(new java.awt.event.MouseAdapter() {  
    public void mouseClicked(java.awt.event.MouseEvent accion)
```

Esta función lo único que hace es cerrar la aplicación llamando a la función System.exit

### -activarControles

```
private void activarControles(int elemSelec)
```

Esta función se utiliza para activar los botones de función dependiendo del elemento seleccionado y permitir funciones coherentes; se llama desde el gestor de selección de elemento en el árbol y desde borrado de colección y de elemento.

Recibe como parámetro un entero que indica que tipo de elemento es, admitiendo los valores definidos en la clase ElementoBD. Si es un dato no controlado desactiva todos los botones de acción sobre colecciones y elementos.

### -dialogoAnadirColeccion

```
private String dialogoAnadirColeccion()
```

Esta función muestra un dialogo para recoger el nombre de la nueva colección a añadir, retornando el nombre tecleado; antes de devolverlo se eliminan los espacios en blanco iniciales y finales.

### -dialogoAnadirRecurso

```
private File dialogoAnadirRecurso()
```

Muestra un diálogo estándar de java para la selección de un archivo del sistema de archivos, devolviendo un objeto File que hace referencia al mismo. Sólo permite la selección de un único fichero a la vez.

### -dialogoSeleccionTipoRecurso

```
private int dialogoSeleccionTipoRecurso ()
```

Muestra un dialogo con una lista desplegable para seleccionar el tipo de recurso a almacenar. Devuelve un entero, con los valores definidos en la clase ElementoBD. Si se cancela devuelve ElementoBD.OTRO.

### -dialogoVerUsuarios

```
private void dialogoVerUsuarios(ArrayList listaUsuarios)
```

Crea un objeto DialogoVerUsuarios y le pasa al constructor de dicha clase un array con la lista de usuarios (la lectura se efectúa en el gestor del botón mostrar usuarios, llamando a la función correspondiente de GestorBD).

### -dialogoConfirmacionBorrado

```
private int dialogoConfirmacionBorrado(String mensaje)
```

Muestra un dialogo de confirmación con opciones si/no. Recibe como parámetro una cadena que será la que se muestre en el cuerpo de dialogo (para su uso desde borrado de recurso y de colección)

## Instalación de la aplicación

### Requisitos para el funcionamiento

- Tener instalado un JDK o SDK Java (ver 1.5)
- Tener instalado el SGBD eXist (ver 1.1.2)
  - Debe estar en el mismo equipo que la aplicación
  - No debe cambiarse el puerto por defecto del gestor
  - El usuario admin debe estar en el estado por defecto que deja la instalación, es decir, sin clave.

\*Si estos parámetros no fuesen los correspondientes a la instalación eXist disponible habría que ajustar los valores de las variables URI, usuario, usuarioPwd en la clase GestorBD.

### Proceso de instalación

**(\*)Todos los archivos necesarios para la ejecución de la aplicación (fuentes, fuentes compilados, archivos gráficos y archivo de lanzamiento) se entregan junto a este documento. Las librerías necesarias de eXist no se entregan por el tamaño de las mismas y porque es necesario disponer de una instalación del SGBD lo cual implica que se dispone de ellas.**

Debe crearse un directorio donde se copiarán todos los ficheros proporcionados.

Las 8 clases de la aplicación (que no son necesarias, salvo que queramos modificarla) son;

```
DialogoVerUsuarios.java
ElementoBD.java
EstructuraBDSwing.java
ExcepcionGestorBD.java
GestorBD.java
RenderArbol.java
Usuario.java
UsuariosTableModel.java
```

La relación de archivos .class necesarios es la siguiente (19)

```
DialogoVerUsuarios$1.class
DialogoVerUsuarios$2.class
DialogoVerUsuarios.class
ElementoBD.class
EstructuraBDSwing$1.class
EstructuraBDSwing$2.class
EstructuraBDSwing$3.class
EstructuraBDSwing$4.class
EstructuraBDSwing$5.class
EstructuraBDSwing$6.class
EstructuraBDSwing$7.class
EstructuraBDSwing$8.class
EstructuraBDSwing$9.class
EstructuraBDSwing.class
ExcepcionGestorBD.class
GestorBD.class
RenderArbol.class
Usuario.class
UsuariosTableModel.class
```

Igualmente deberemos tener en el mismo directorio los archivos gráficos utilizados (10).

```
addfile.gif
addfolder.gif
deletefile.gif
deletefolder.gif
docs.gif
ejquery.gif
exit.gif
folder-closed.gif
folder-open.gif
usergroup.gif
```

Además deberemos disponer de las siguientes librerías proporcionados por eXist. Estas 4 librerías se copian en la instalación del SGBD, si el directorio especificado es %exist% la localización de las librerías es la siguiente

```
%exist%\exist.jar
%exist%\lib\core\xmlldb.jar
%exist%\lib\core\xmlrpc-1.2-patched.jar
%exist%\lib\core\log4j-1.2.14.jar
```

Para lanzar la ejecución deberemos ejecutar el archivo 'ejecutar.bat' o teclear la sentencia

```
java -cp .;exist.jar;log4j-1.2.14.jar;xmlldb.jar;xmlrpc-1.2-patched.jar
EstructuraBDSwing
```

Si no disponemos de las librerías copiadas en el mismo directorio de la aplicación deberemos indicar la ruta completa de las mismas en la orden de lanzamiento de la aplicación.

### Compilación desde la línea de comandos

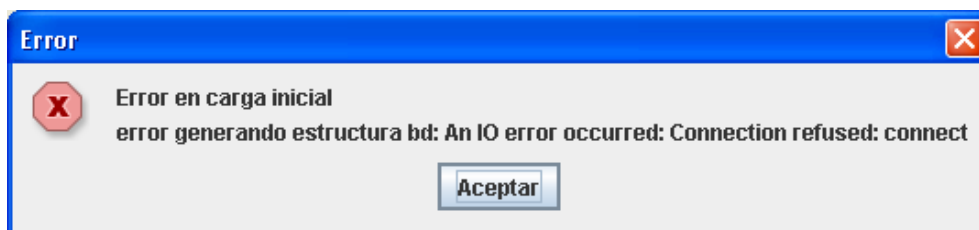
En caso de tener que compilar los archivos fuentes por cualquier motivo (por cambiar los parámetros de conexión o de usuario, o no tener el proyecto creado en un IDE) la secuencia y órdenes de compilación son las siguientes

```
javac ExcepcionGestorBD.java
javac Usuario.java
javac UsuariosTableModel.java
javac DialogoVerUsuarios.java
javac -cp .;exist.jar;xmlldb.jar GestorBD.java
javac ElementoBD.java
javac RenderArbol.java
javac -cp .;xmlldb.jar EstructuraBDSwing.java
```

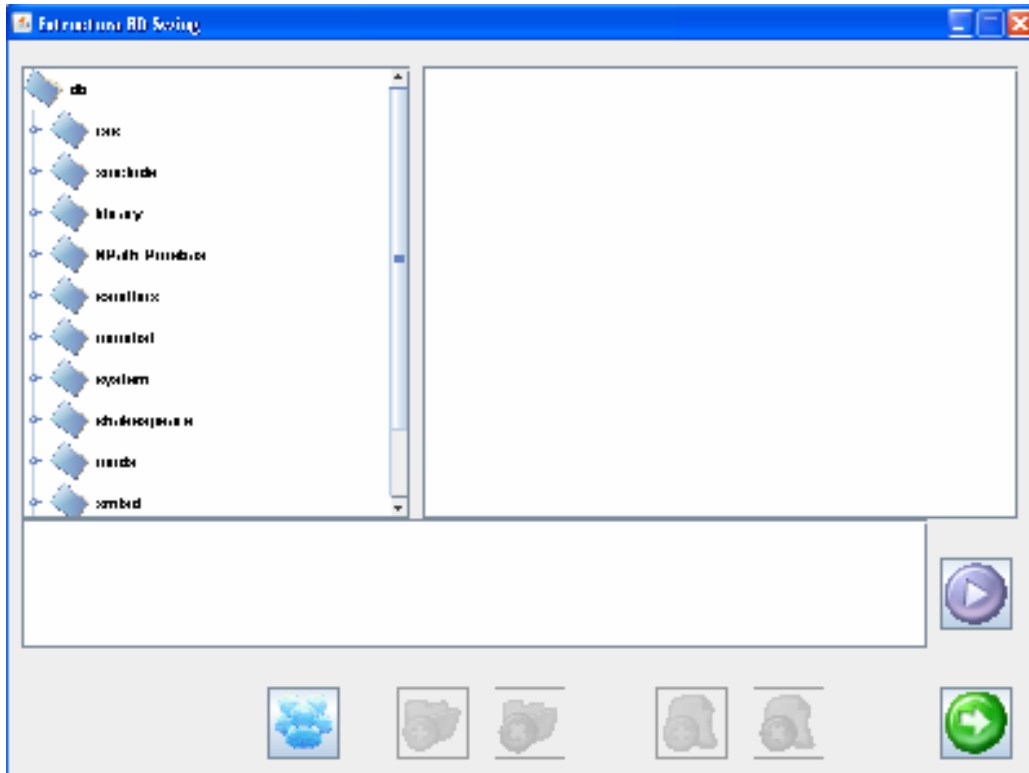
Igual que en el punto anterior, si no disponemos de las librerías copiadas en el mismo directorio de la aplicación deberemos indicar la ruta completa en la sentencia de compilación.

### Funcionamiento de la aplicación

Si cuando lancemos la aplicación el SGBD no se encuentra activo, obtendremos el siguiente mensaje de error










Si todo está correctamente, aparecerá la ventana principal de la aplicación, que tiene el siguiente aspecto



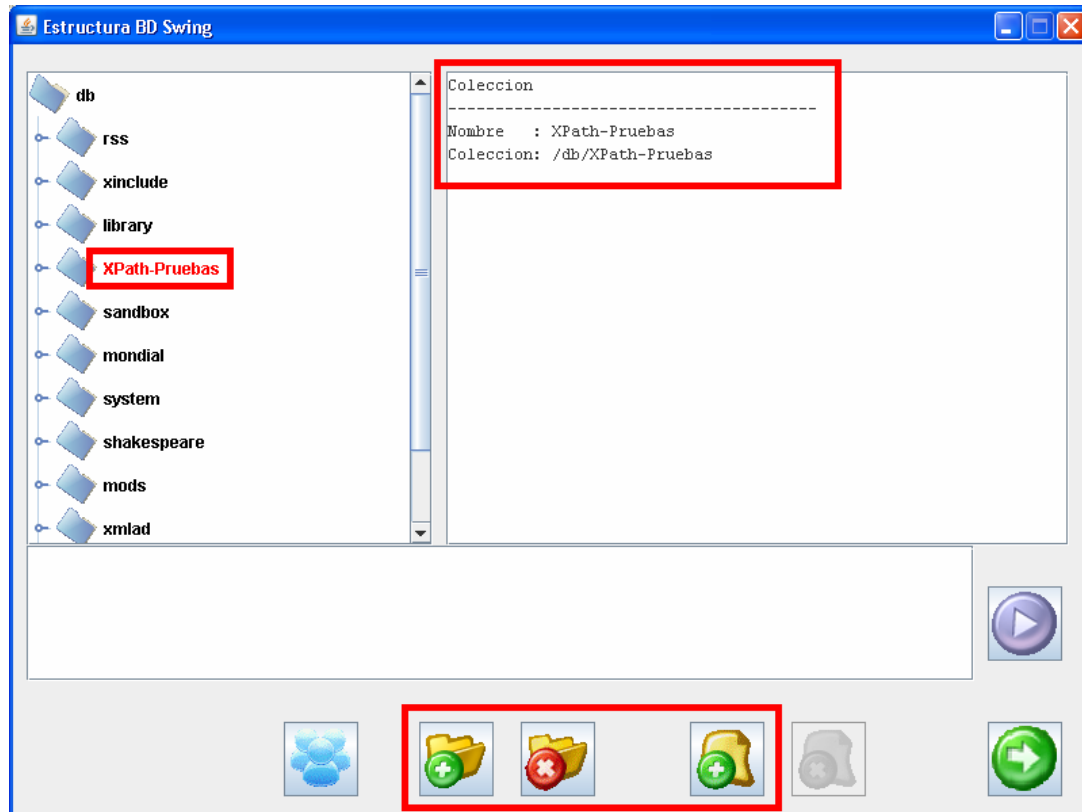
En la parte superior izquierda aparece un árbol con el contenido de la BB.DD, la parte superior derecha es el panel de resultados (información y contenido de elementos seleccionados, resultado de la query procesada). Justo debajo hay un cuadro de texto que corresponde al área de la consulta; en este espacio escribiremos la consulta que queremos hacer.

Además de estas tres áreas existen siete botones, que estarán activos solo si la acción que representan es coherente

	Salir de la aplicación	Siempre activo
	Ejecutar consulta	Siempre activo
	Añadir colección	Activo si hay seleccionada en el árbol una colección
	Borrar colección	Activo si hay seleccionada en el árbol una colección
	Añadir recurso	Activo si hay seleccionada en el árbol una colección
	Borrar recurso	Activo si hay seleccionada en el árbol un recurso
	Consultar lista de usuarios registrados	Siempre activo

Ahora ya podemos navegar por el contenido de la base de datos, accediendo a las distintas colecciones y documentos.

Si en el árbol de contenido pinchamos sobre una colección, el nombre se resaltaré en rojo y se activarán los botones de acción añadir y borrar colección y añadir recurso, además de mostrar en el panel de contenido el nombre y ruta de la misma.

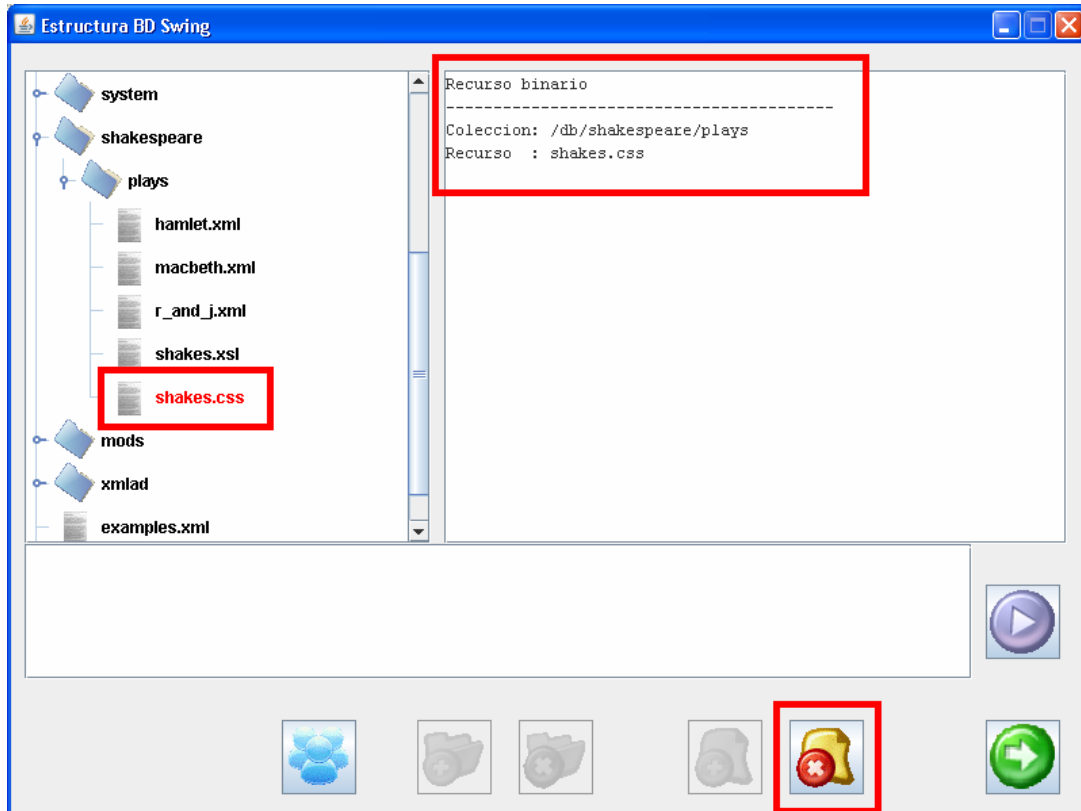


Haciendo doble-click sobre la colección esta se abre mostrando su contenido.

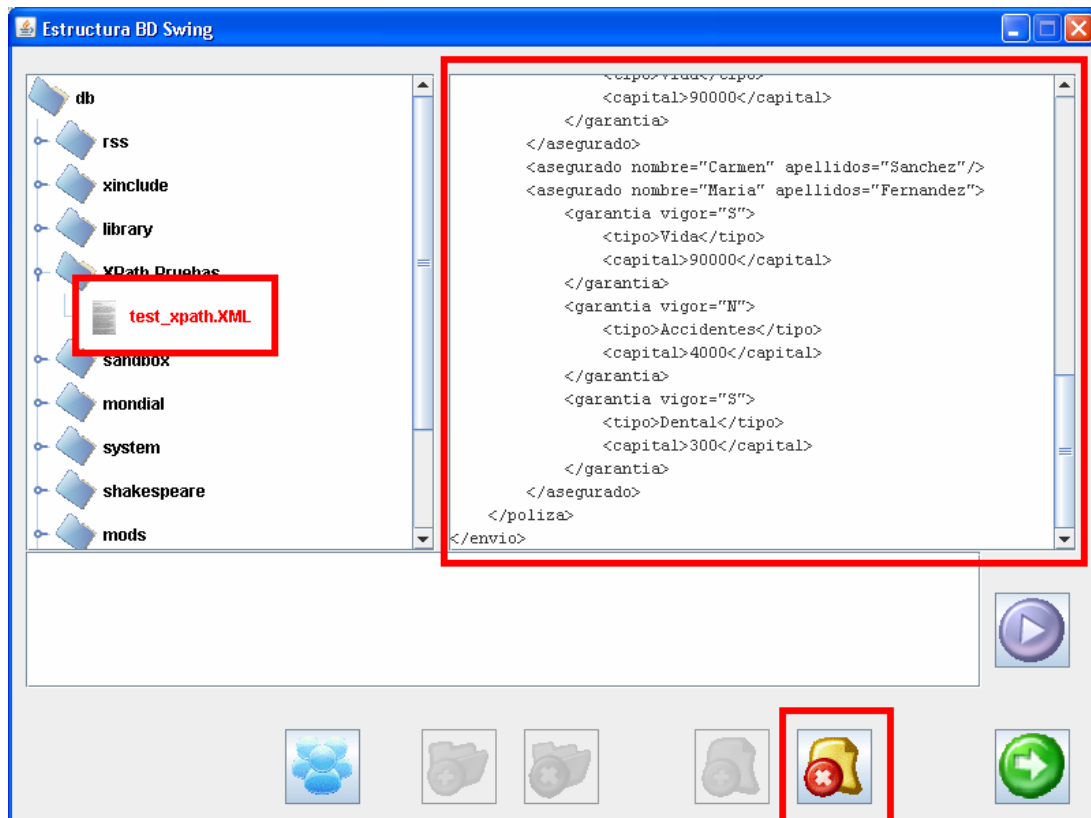
Si desde el árbol de contenido seleccionamos un recurso, se activará la opción de borrado de recursos y se desactivarán las opciones de gestión de colección; en el área de contenido, si es un recurso binario aparecerá la descripción del elemento y si es un recurso XML aparecerá el contenido del mismo

Con recurso binario seleccionado el aspecto de la pantalla es el siguiente




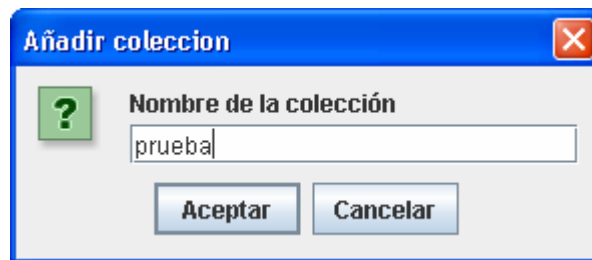


Con un recurso XML seleccionado (muestra el contenido)




## Gestión de colecciones

Con una colección seleccionada podemos efectuar las tareas de gestión de las mismas; si pulsamos sobre el botón añadir colección , se abrirá un dialogo para pedir el nombre; la nueva colección se insertará como hija de la colección seleccionada




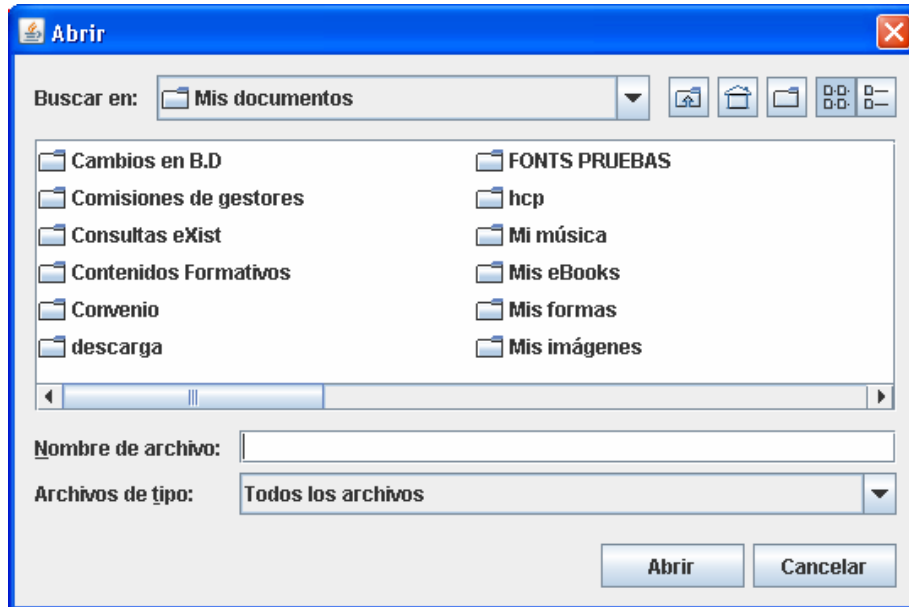
Si pulsamos cancelar la creación de la colección no se efectuará, si pulsamos aceptar la insertaremos y podremos acceder a ella desde el árbol de contenido.

Si con la colección seleccionada pulsamos el botón borrar colección , aparecerá un diálogo de confirmación de borrado

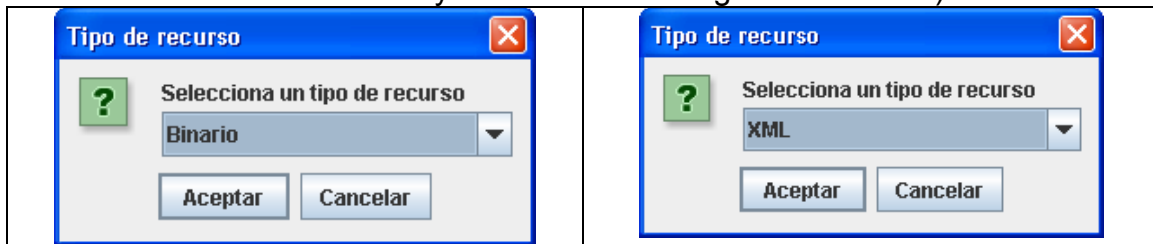


Si pulsamos sobre 'Si, Borrar' se hará efectivo el borrado; esto implica el borrado de todo el contenido de dicha colección.

Igualmente podremos pinchar sobre la opción añadir documento , en este caso se abrirá un dialogo para seleccionar el archivo a añadir




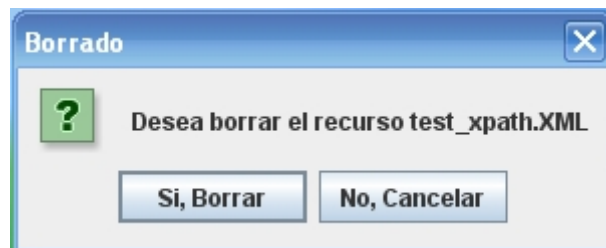
Si pulsamos cancelar el proceso se para, si con el archivo seleccionado pulsamos aceptar aparece la ventana de selección del tipo de recurso, podemos elegir entre añadirlo como recurso XML o recurso binario (si el recurso se marca como XML y no es correcto se genera un error).



Si pulsamos cancelar el proceso se parará, si pulsamos aceptar se añadirá el recurso y estará disponible desde el árbol de contenido.

### Borrado de recursos

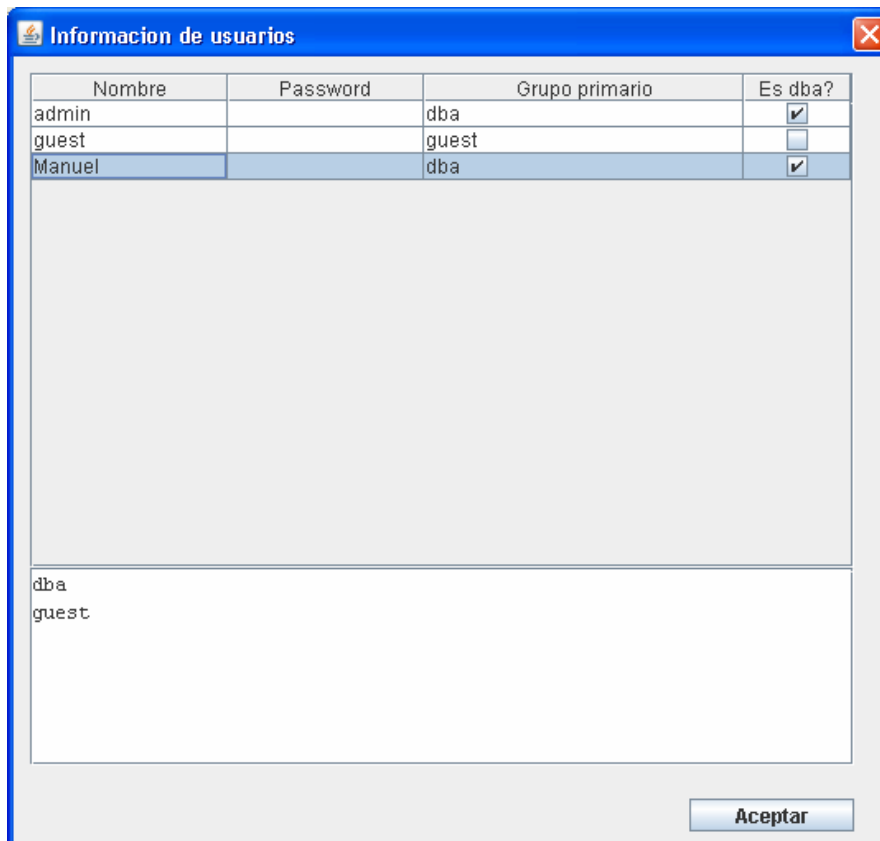
Si tenemos seleccionado un recurso podemos pulsar sobre el botón borrar recurso , si pulsamos aparecerá el dialogo de confirmación de borrado



Si pulsamos sobre 'Si, Borrar' el recurso se eliminará y desaparecerá del árbol de contenido, si pulsamos sobre 'No, Cancelar', la acción se suspenderá.

## Consulta de usuarios


Si pulsamos sobre el botón ver usuarios , se abrirá un diálogo que muestra los usuarios registrados en el SGBD

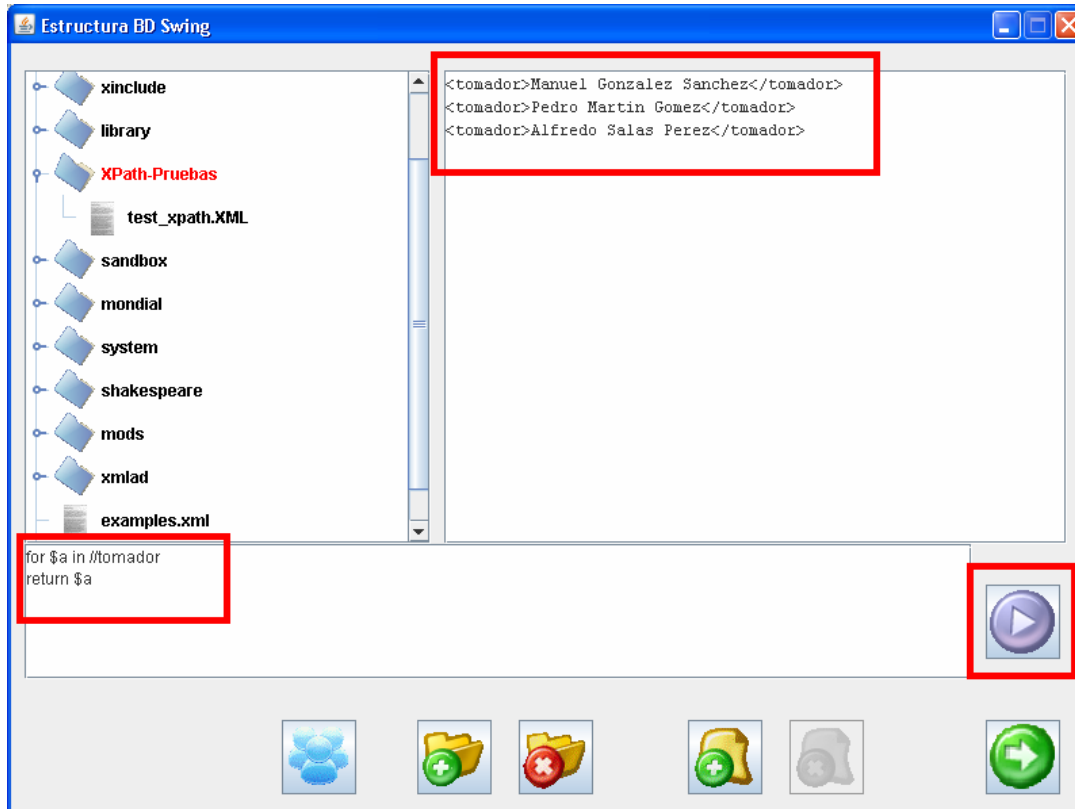


En la parte superior aparece una tabla con los usuarios registrados y los datos principales de los mismos; en la parte inferior aparecerá la lista de grupos asociados al usuario seleccionado.

Si pulsamos en la tabla sobre cualquier usuario se actualizará la lista de grupos en el panel inferior; pulsando 'Aceptar' se cerrará el diálogo y volveremos a la pantalla principal.

## Ejecutar consultas

Tenemos la opción de ejecutar consultas sobre la base de datos; teclearemos la consulta en el área de texto inferior y pulsaremos el botón ejecutar query ; la consulta se efectuará en el contexto de la colección seleccionada o la colección a la que pertenece el recurso seleccionado (si no hay ninguna colección ni recurso seleccionado el contexto será la colección raíz).



Los resultados de la consulta se mostrará en el área de contenido; si la consulta no produce resultados en el área de contenido aparecerá el mensaje 'La consulta no ha devuelto resultados'

### Mejoras futuras a realizar

Vamos a hacer una relación de posibles mejoras o ampliaciones sobre la aplicación desarrollada.

-Parametrización de la url de conexión

Ahora la URL de conexión es fija, apuntando al ordenador local, puerto por defecto (8080), es interesante obtener estos parámetros de manera dinámica, como posibles opciones:

- en la instanciación de la clase GestorBD obteniendo esos datos de un fichero de configuración.
- Pidiendo los datos al comienzo de la aplicación

-Identificador del usuario

Igual que en la URI, el usuario con el que se efectúan las lecturas es el usuario admin sin clave asignada. Se podría presentar una pantalla inicial de login que validase la entrada del usuario contra los registrados y efectuar las lecturas con dicho usuario.

Hay que tener en cuenta en este punto que a la colección system solo pueden acceder usuarios dba

-Ampliar la gestión de usuarios

Se puede efectuar la consulta de usuarios, pero no realizar ninguna gestión de los mismos; podría implementarse opciones de creación y modificación de usuarios y una gestión más potente de la asignación de usuarios a grupos.

-Optimizar la carga del árbol

Al arrancar se carga toda la estructura de la base de datos en el árbol de contenidos; si el contenido de la BB.DD es grande esto puede ser lento, por lo que puede ser una buena idea implementar una carga dinámica de las distintas ramas a las que se va accediendo para optimizar recursos.

## 11 – Conclusiones

El enfoque de este trabajo ha sido el poder adentrarse en el conocimiento de la tecnología de almacenamiento y consulta de documentos XML en bases de datos nativas de una manera natural y progresiva.

El punto de partida ha sido una breve definición de XML y sus características más destacables, para ver después las distintas estrategias de almacenamiento de las que disponemos. Se ha realizado una visual sobre las opciones de SGBD relacionales actuales para el tratamiento de XML (para almacenar extraer los datos codificados en el documento y almacenarlos en tablas ó almacenar el documento entero en un campo y para consultar crear un documento XML a través de funciones específicas), y se han definido cuales son las características de un SGBD XML nativo (principalmente utilizar un modelo de datos basado en documentos). De este modo hemos podido ver los puntos fuertes e inconvenientes de cada modelo y las diferencias entre ellos, pudiendo decidir que sistema es mejor usar en una situación u otra (Punto 4).

Con el SGBD a usar seleccionado comenzamos el estudio de los distintos lenguajes de consulta, comenzando por XPath 1.0, continuando con su evolución XPath 2.0 y terminando con XQuery 1.0; de este modo nos hemos familiarizado gradualmente con estos lenguajes de consulta de documentos y bases de datos XML, asimilando la naturaleza semiestructurada y jerárquica del XML adquiriendo soltura a la hora de obtener datos a través de múltiples ejemplos.

Como siguiente paso se ha visto la problemática actual sobre la actualización de datos y la falta de un estándar definido, así como las distintas posibilidades existentes (sin el uso de programas y sin reemplazar documentos), revisando el futuro estándar definido por el W3C XQuery Update Extensión y efectuando pruebas prácticas con las extensiones ofrecidas por el SGBD eXist.

Finalmente se han estudiado las distintas vías de acceso a la información almacenada en un SGBD, haciendo hincapié en las distintas APIs existentes para Java (XQJ como futuro estándar al igual que JDBC y XMLDB como estándar de facto actual).

Para completar el proceso y como elemento integrador de todo lo aprendido se ha realizado el desarrollo de una aplicación de gestión del SGBD, utilizando las librerías XMLDB proporcionadas por eXist, que nos permite administrar los datos y la realización de consultas XQuery contra ellos.

Las ventajas de XML como lenguaje de comunicación y como soporte a distintos formatos de archivo, su sencillez, extensibilidad y sus capacidades multiplataforma hacen que su aceptación y la cantidad de datos en dicho formato vayan a aumentar todavía en el futuro.

Pero a pesar de ser XML un estándar presente desde 1998 (primera recomendación del W3C <http://www.w3.org/TR/1998/REC-xml-19980210> ) y de la existencia de SGBD XML nativos desde hace tiempo en el mercado, a diferencia del modelo relacional, donde existe una base teórica estable y sólida además de unos estándares plenamente aceptados, este es un modelo por desarrollar y completar (hecho queda reflejado en la publicación del estándar de consulta en el año 2006 y en la no existencia de un lenguaje unificado de actualización a fecha de hoy) y con unas expectativas de desarrollo prometedoras.

La primera versión XQuery muestra una funcionalidad extensa comparada sobre todo con la primera especificación SQL, proporcionando herramientas avanzadas como el tratamiento iterativo y alternativo, conjunto de funciones básicas para el tratamiento de datos muy extensa, declaración de variables, funciones y de módulos...

Esta tecnología satisface la necesidad de almacenamiento, consulta, recuperación y modificación de información poco estructurada, por lo que no compite directamente con la tecnología relacional existente hasta ahora (la cual se muestra poco flexible en este escenario), sino que la complementa (incluso utilizándola para gestionar los almacenes de datos) y en este sentido están enfocados los esfuerzos de desarrollo.



## 12 – Anexos

### 1 – Instalación de eXist.

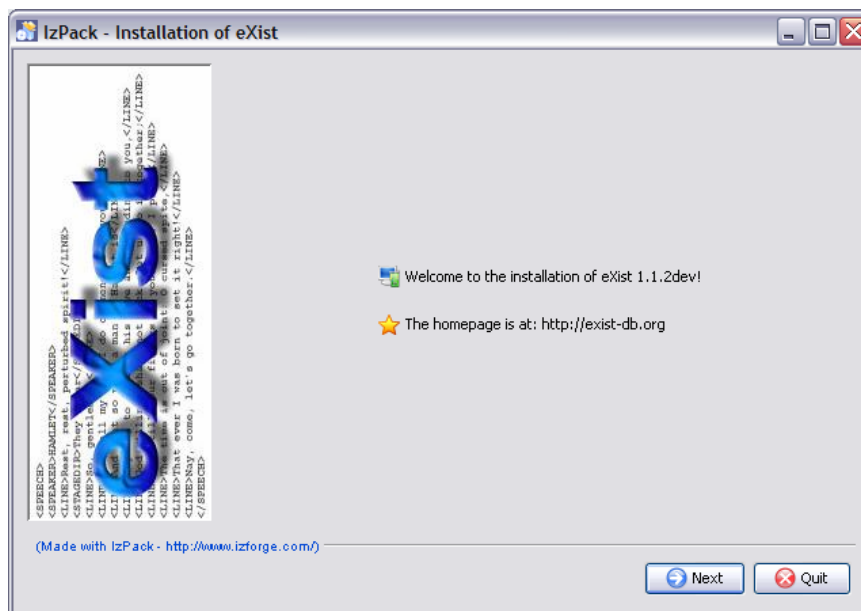
Para instalar eXist primero descargaremos la última revisión de la página Web del producto, en

<http://www.exist-db.org/index.html#download>

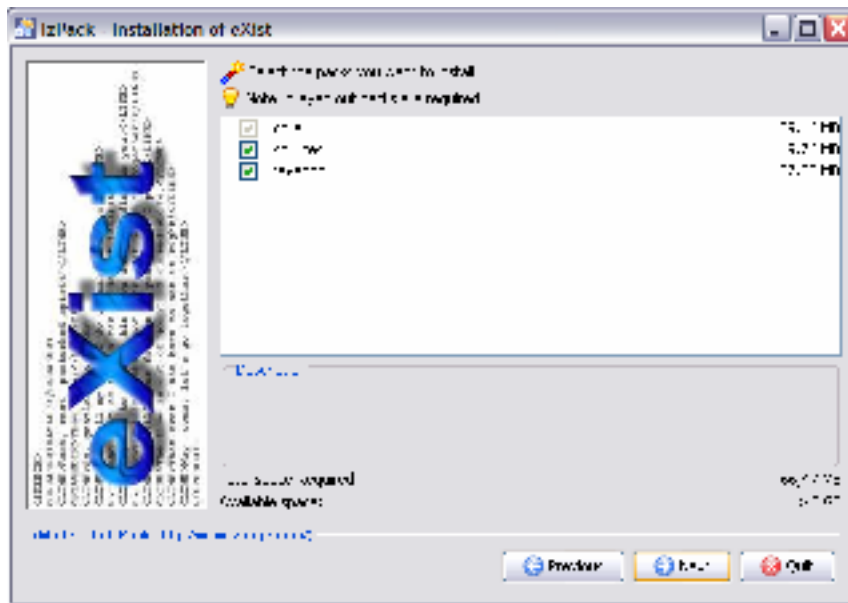
- Es necesario tener instalado en el sistema el Java JDK 1.4.2 ó posterior.

Para la realización de esta guía y todas las prácticas del proyecto se ha instalado la versión 1.1.1-newcore-build4311 instalándola en un sistema Windows XP profesional SP2. y JDX 1.6.

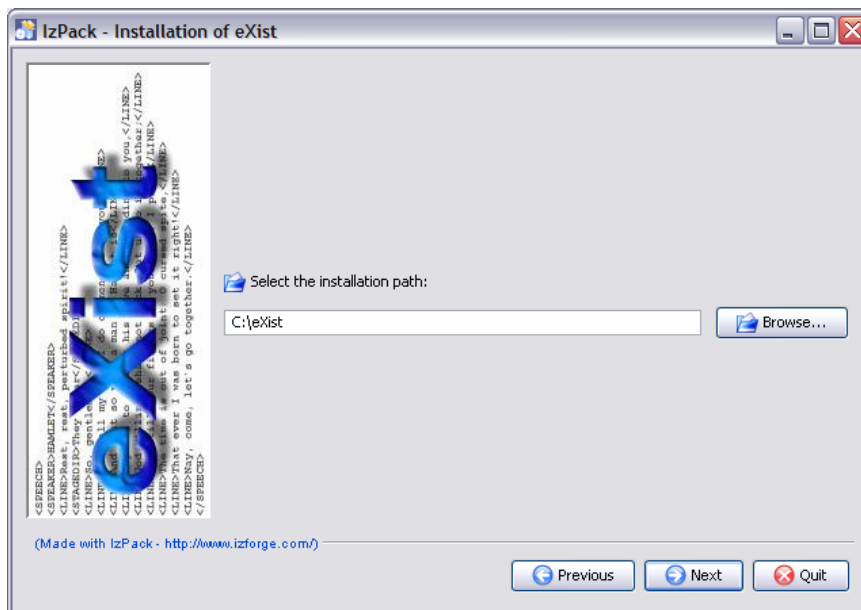
Lanzamos la instalación del producto, para ello haremos doble clic en el archivo eXist-1.1.1-newcore-build4311.jar y se ejecutará el instalador del SGBD.



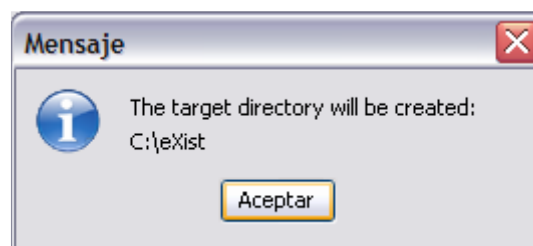
Aparece la pantalla de bienvenida a la instalación, pulsamos Next.



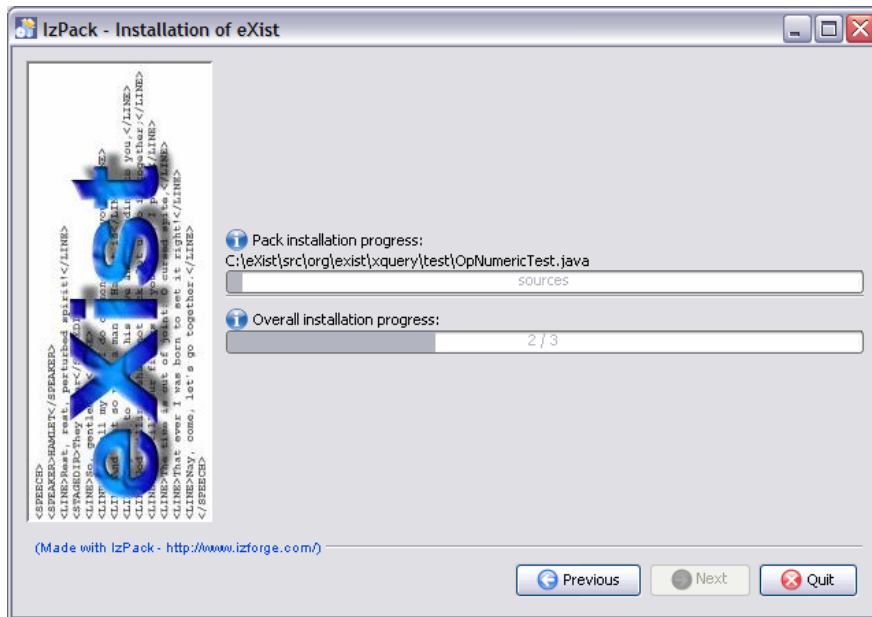
Seleccionamos la instalación de todos los componentes, incluidos la documentación javadoc de las interfaces y pulsamos Next.



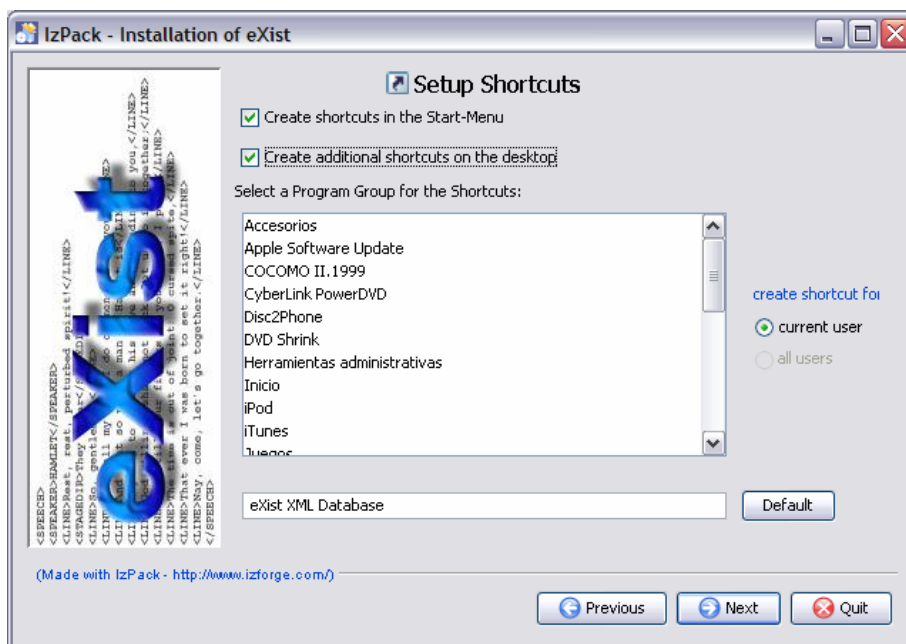
Introducimos la carpeta de destino y pulsamos Next, nos dará el siguiente mensaje de aviso



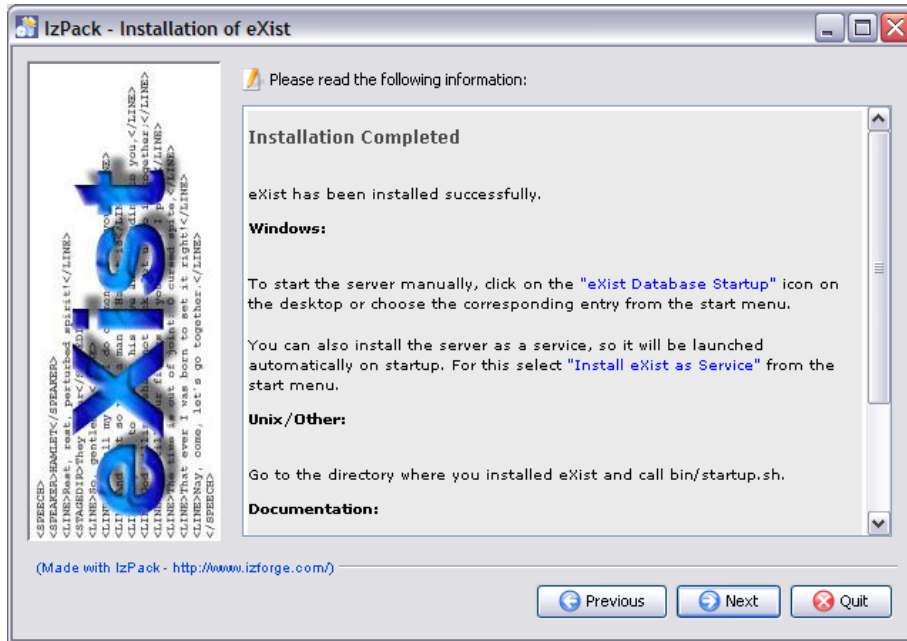
Pulsamos aceptar y comenzará la copia de archivos.



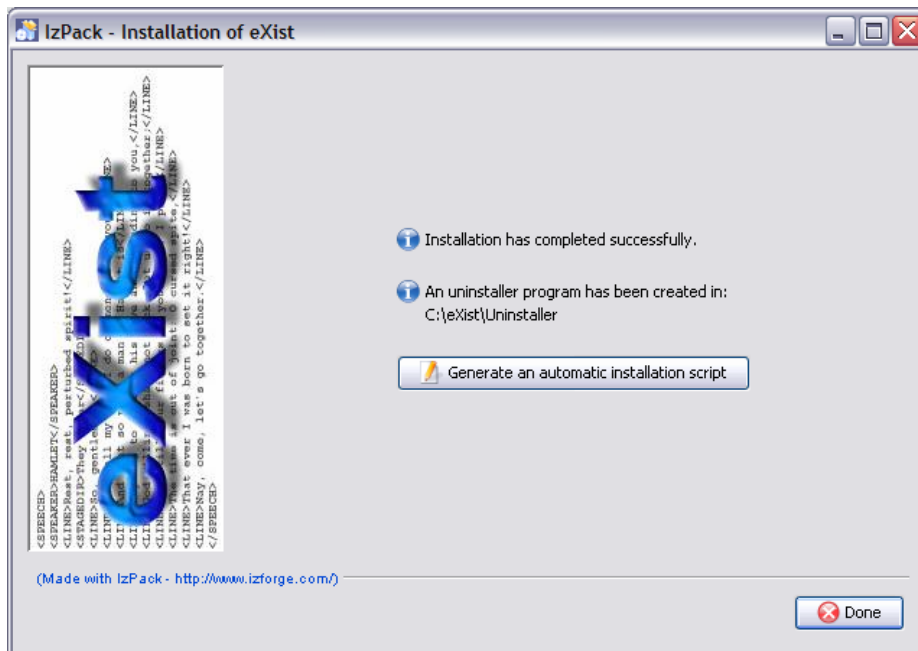
Cuando termine la copia aparecerá la ventana donde le indicaremos el grupo de programas donde nos creará los accesos directos para acceder a los programas,



Pulsamos Next.



Pulsamos Next.



Pulsamos Done. En este punto ya tenemos instalado el SGBD en nuestro sistema.

Para arrancarlo pulsamos en eXist Database Startup, y se iniciará el SGBD, aparecerá una ventana de consola DOS

```

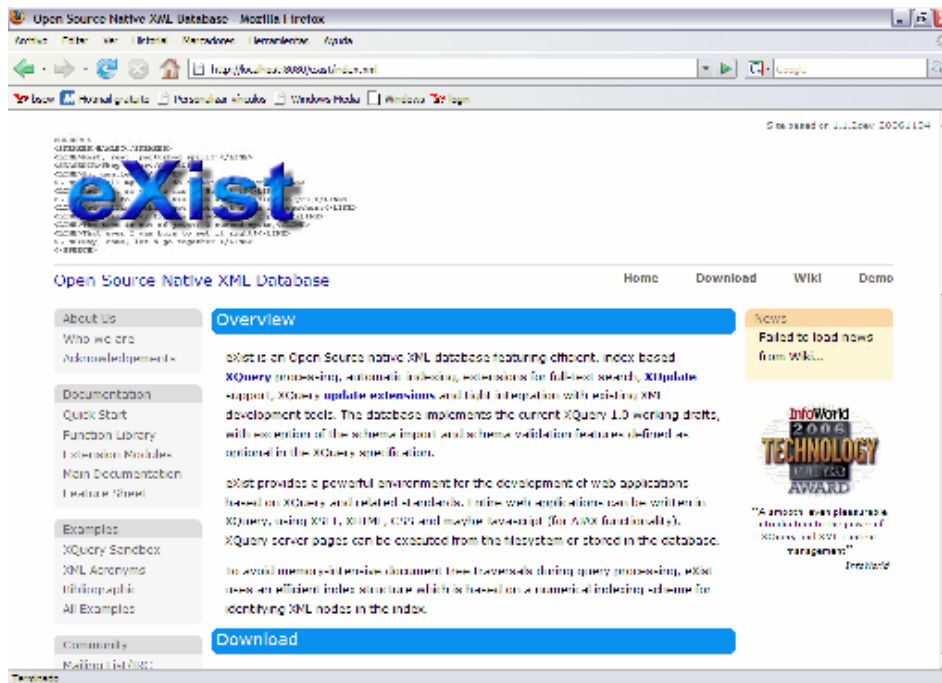
eXist Database Startup
WARN: The following JAR file entries from 'org/exist/start/start.config' aren't
available (this may NOT be a problem):
  C:\exist\tools\irchot\dist\irchot.jar

Looking for a valid Parser...
Checking for Xerces, found version Xerces-J 2.8.1
OK!

Looking for a valid Transformer...
Checking for Xalan, found version Xalan Java 2.7.0
OK!

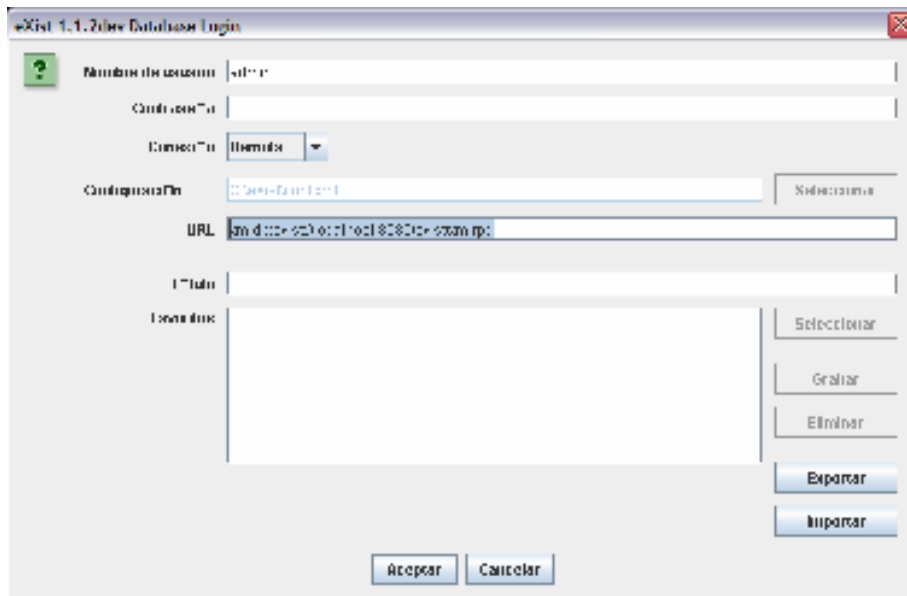
Configuring eXist from C:\exist\conf.xml
Found 1 catalog entries.
Loading catalog: C:\exist\webapp\WEB-INF\catalog.xml
06 oct 2007 11:11:55.149 [main] INFO  <FileResource.java [c<clinit>]:60> - Checki
ng Resource aliases
06 oct 2007 11:11:55.320 [main] INFO  <HttpServer.java [setStatsOn]:1068> - Stat
istics on = false for org.mortbay.jetty.Server@13c0b53
06 oct 2007 11:11:55.330 [main] INFO  <HttpServer.java [start]:654> - Starting J
etty/5.0.0
06 oct 2007 11:11:55.360 [main] INFO  <HttpServer.java [start]:669> - Started or
g.mortbay.http.NCSARequestLog@edf389
06 oct 2007 11:11:56.041 [main] INFO  <HttpContext.java [start]:1604> - Started
WebApplicationContext[eXist XML Database,eXist XML Database]
Logging already initialized. Skipping...
06 oct 2007 11:11:57.673 [main] WARN  <JavaUtils.java [isAttachmentSupported]:13
05> - Unable to find required classes
  javax.activation.DataHandler and javax.ma
  il.internet.MimeMultipart). Attachment
  support is disabled.
06 oct 2007 11:11:58.344 [main] INFO  <SocketListener.java [start]:204> - Starte
d SocketListener on 0.0.0.0:8080
06 oct 2007 11:11:58.344 [main] INFO  <HttpServer.java [start]:690> - Started or
g.mortbay.jetty.Server@13c0b53
  
```

EL SGBD tiene un servidor Web incorporado (Jetty), Ahora podemos acceder al SGBD mediante un navegador, pulsamos en eXist Local Homepage.

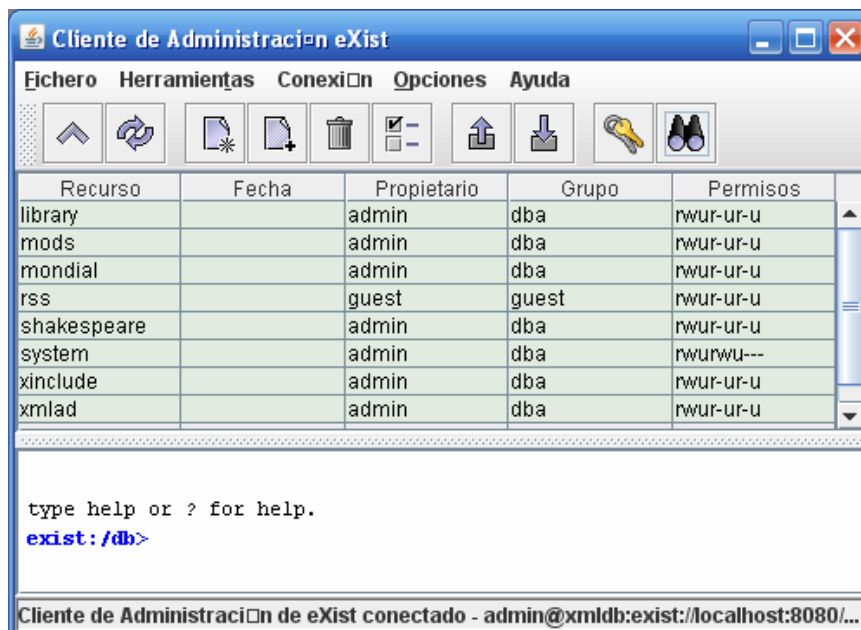


Aunque tiene el mismo aspecto que la Web de eXist estamos accediendo al servidor local, si pulsamos sobre admin ó XQuery Sandbox podemos acceder a la administración del mismo o a la aplicación Web para el lanzamiento de consultas.

Con la instalación también se da el Database Client, una aplicación Java que permite la explotación del SGBD. Pinchamos en “eXist Client Shell” y aparece la siguiente pantalla

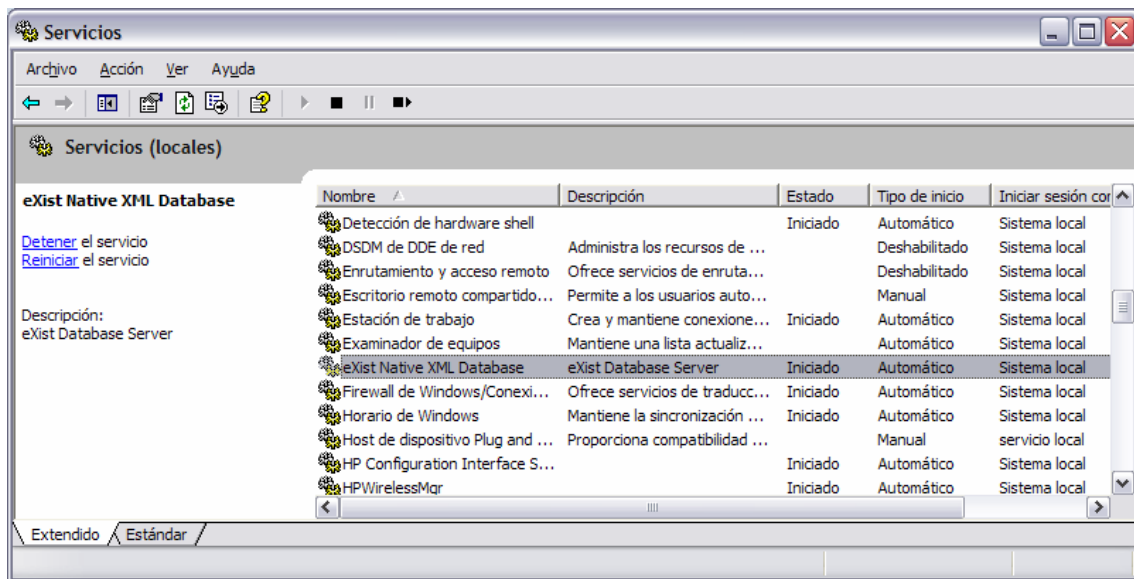


Pide los datos de la conexión; si no hemos tocado ningún parámetro si pulsamos aceptar se conectará como admin. (Por defecto no tiene clave)



Existe la posibilidad en entorno Windows de instalar eXist como un servicio del sistema, de este modo podemos arrancarlo automáticamente al iniciar la sesión y no tener la ventana de la consola abierta. Para ello solo tenemos que pinchar en la opción “Install eXist as a service” dentro del grupo de programas que hemos creado, y automáticamente aparecerá como un servicio.

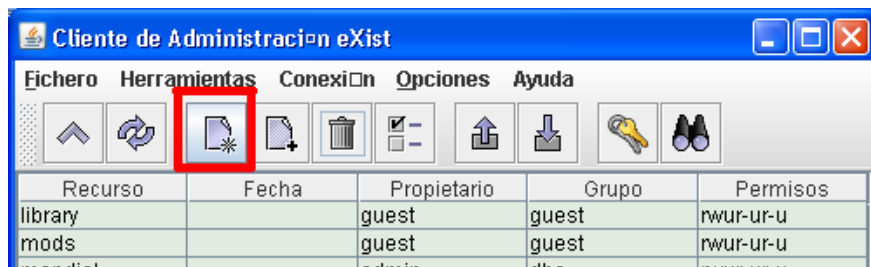
Después de instalarlo como servicio, no se arranca automáticamente, por lo que deberemos reiniciar el sistema o bien iniciarlo manualmente.



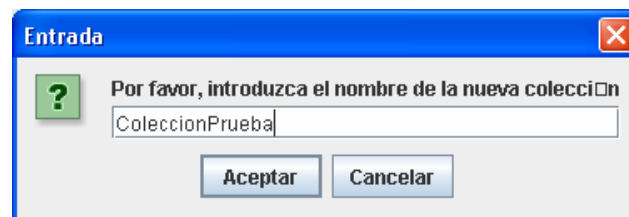
## 2 – Definición de una colección.

Vamos a realizar la creación de la colección desde el 'Client Shell', pero también es posible realizarla accediendo mediante un navegador a la aplicación integrada.

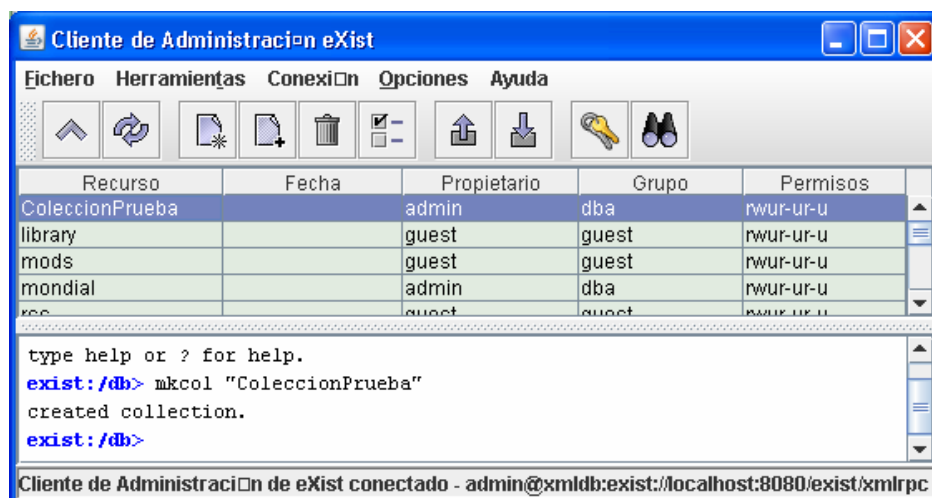
Nos conectamos con nuestro usuario y password, y pinchamos en el icono 'Definir nueva colección' o accedemos al menú Fichero/Crear colección



Aparecerá el siguiente diálogo pidiendo el nombre de la colección, introducimos el nombre y pinchamos sobre Aceptar



Volveremos a la pantalla principal donde aparecerá la nueva colección disponible para trabajar con ella

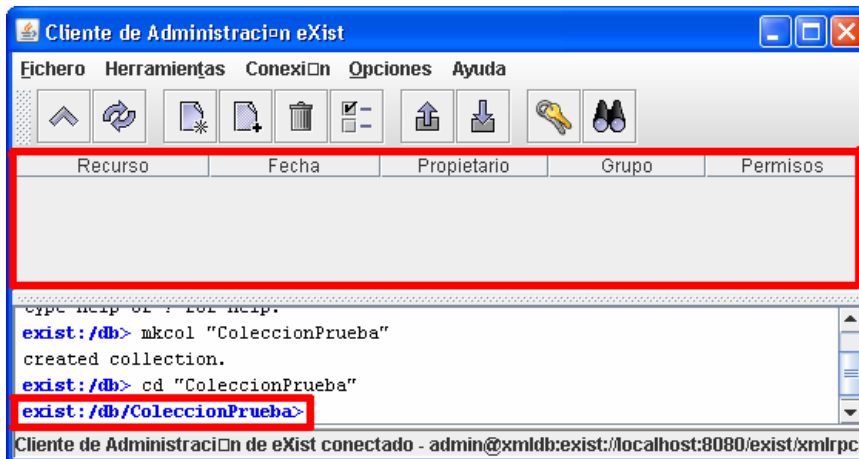


Podemos definir colecciones dentro de otras colecciones siguiendo el mismo procedimiento

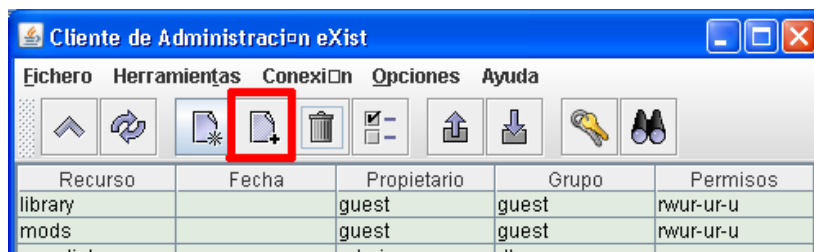


### 3 – Añadir un documento a una colección.

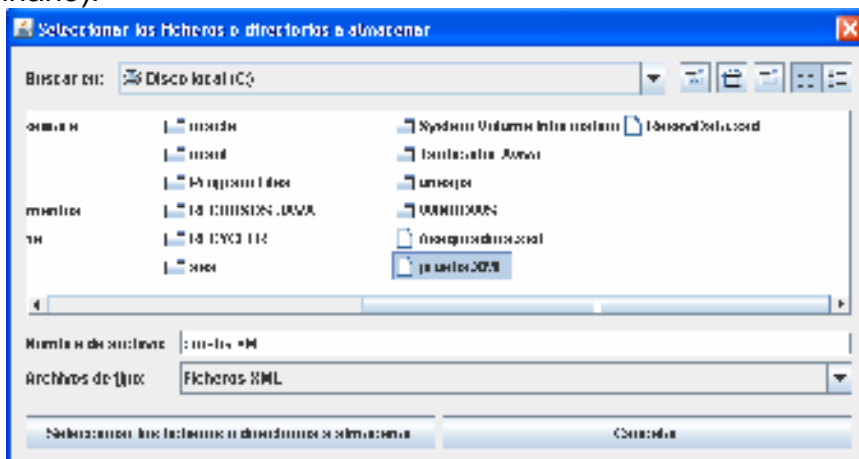
Para añadir un documento a una colección debemos estar trabajando sobre esa colección, desde la pantalla principal hacemos iremos haciendo doble-clic la jerarquía de colecciones hasta que estemos trabajando con la colección a la que queremos añadir el elemento. En la parte superior veremos el contenido de la colección, y en la ventana inferior vemos a la colección a la que estamos conectados.



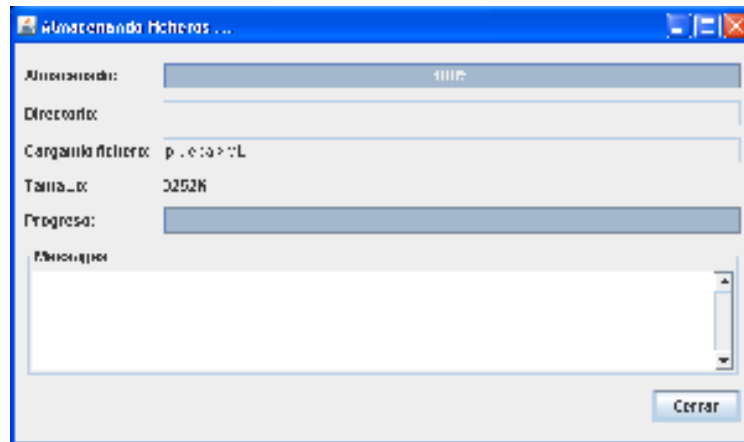
Desde aquí pinchamos en el icono añadir nuevo documento o vamos al menú, Fichero/Almacenar fichero-directorio



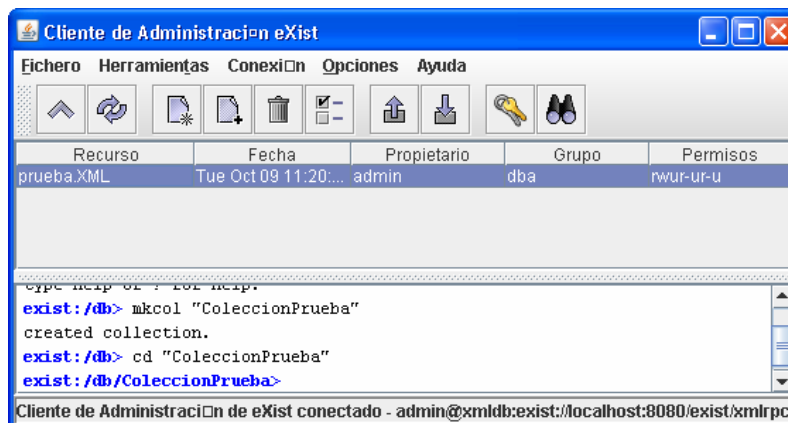
Y se abrirá un diálogo donde podremos seleccionar el archivo a añadir y su tipo (XML o binario).



Pinchamos sobre 'Seleccionar los ficheros o directorios a almacenar'. Aparecerá un dialogo mostrando el progreso de la carga, si todo va correcto terminaremos sin mensajes de error

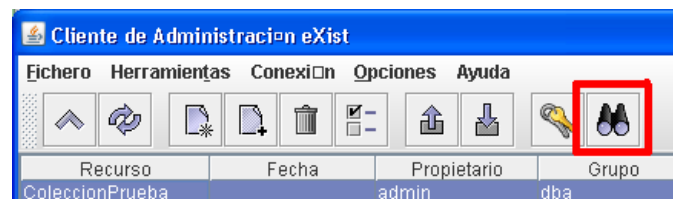


Pulsamos sobre cerrar y veremos la pantalla principal, con los documentos de la colección, y el que acabamos de cargar ya añadido.

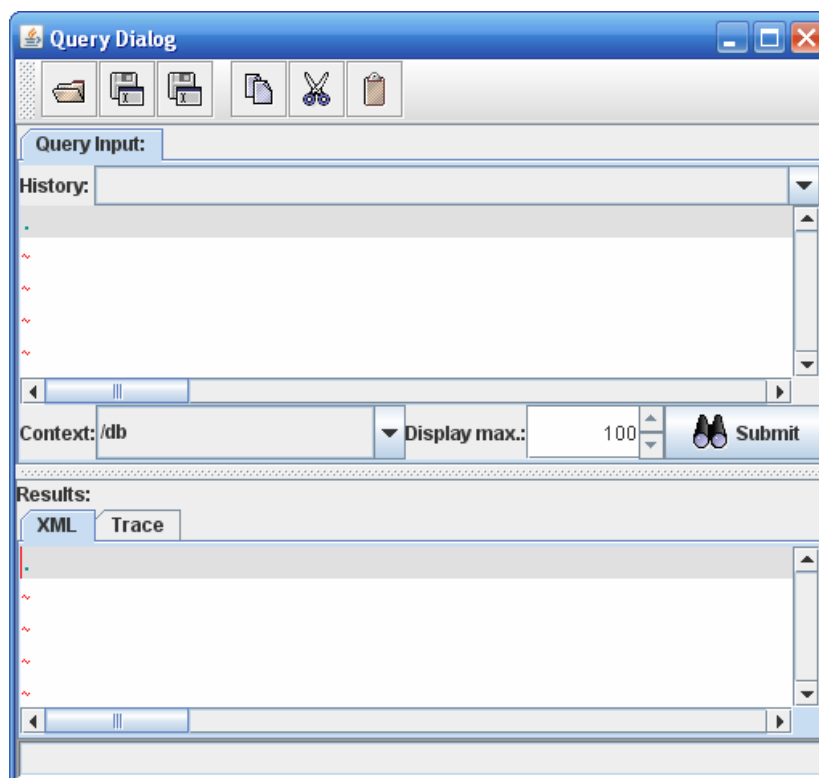


#### 4 – Query Dialog.

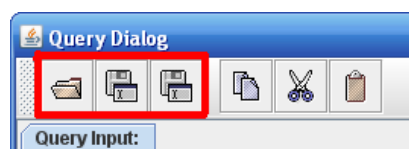
Si desde el cliente de administración pulsamos sobre el botón con los prismáticos ó accedemos al menú Herramientas / Encontrar, accedemos al Query Dialog,



Desde el 'Query Dialog' podemos lanzar consultas interactivas y visualizar los resultados, permitiendo salvar las consultas y resultados que obtengamos

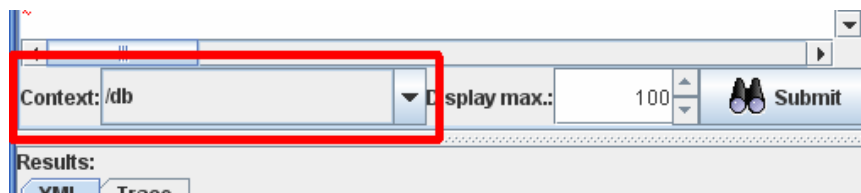


En la barra superior disponemos de tres botones



El primero nos permitirá abrir una consulta guardada; el segundo y tercero nos permitirá guardar la consulta y los resultados de la última consulta ejecutada respectivamente.

Es importante el control que nos permite cambiar el contexto en el que se ejecutará la consulta; por defecto aparecerá la colección desde donde hemos pulsado para acceder al 'Query Dialog', pero es posible cambiarlo desde aquí; si el contexto no está bien seleccionado es posible que los resultados no sean los esperados.



## 9 – Código fuente de las clases desarrolladas.

### ExcepciónGestorBD.java

```
/*
 * TFC - SGBD XML Nativos
 * @author MANUEL SOTILLO
 *
 * Excepción en conexión con el SGBD
 */
public class ExcepcionGestorBD extends Exception {

    private static final long serialVersionUID = 1L;

    /*
     * Constructor por defecto
     * Se asigna mensaje genérico.
     */
    public ExcepcionGestorBD () {
        super("Problemea en la conexión con la BD");
    }

    /*
     * Constructor parametrizado
     * @param String pTexto Descripción de la excepción
     */
    public ExcepcionGestorBD (String pTexto) {
        super(pTexto);
    }
}
```

## ElementoBD.java

```
/*
 * TFC - SGBD XML Nativos
 * @author MANUEL SOTILLO
 *
 * Representación de un elemento de la base de datos
 */
import java.io.UnsupportedEncodingException;
import java.net.URLDecoder;

public class ElementoBD {

    public static final int OTRO = 0;
    public static final int COLECCION = 1;
    public static final int RECURSO_XML = 21;
    public static final int RECURSO_BINARIO = 22;
    private String nombre=null;
    private String coleccion=null;
    private int tipo;

    /**
     * Constructor parametrizado
     *
     * @param nombre Nombre del recurso
     * @param tipo Tipo del recurso (coleccion,recurso_xml,recurso_binario)
     * @param coleccion Coleccion donde esta almacenado
     */
    public ElementoBD(String nombre, int tipo, String coleccion) {
        super();
        this.nombre = nombre;
        this.coleccion = coleccion;
        this.tipo=tipo;
    }

    public String getNombre() {
        return nombre;
    }

    public String getPathCompleto() {
        return GestorBD.URI + coleccion + "/" + nombre;
    }

    public String getColeccion() {
        return coleccion;
    }

    public int getTipo() {
        return tipo;
    }

    /**
     * Metodo de conversion a cadena
     * @return Nombre del elemento (normalizado)
     */
    public String toString(){
        try {
            return URLDecoder.decode(nombre,"UTF-8");
        } catch (UnsupportedEncodingException e) {
            return "--error normalizando nombre--";
        }
    }
}
```

## Usuario.java

```
/*
 * TFC - SGBD XML Nativos
 * @author MANUEL SOTILLO
 *
 * Encapsula a un usuario de la BB.DD
 */
public class Usuario {
    private String nombre;
    private String password;
    private String grupoPrimario;
    private boolean esDBA;
    private String[] grupos;

    public Usuario(String nombre,
                   String password,
                   String grupoPrimario,
                   boolean esDBA,
                   String[] grupos) {
        super();
        this.nombre = nombre;
        this.password = password;
        this.grupoPrimario = grupoPrimario;
        this.esDBA = esDBA;
        this.grupos = grupos;
    }

    public boolean getEsDBA() {
        return esDBA;
    }

    public String getGrupoPrimario() {
        return grupoPrimario;
    }

    public String[] getGrupos() {
        return grupos;
    }

    public String getNombre() {
        return nombre;
    }

    public String getPassword() {
        return password;
    }
}
```

## RenderArbol.java

```
/*
 * TFC - SGBD XML Nativos
 * @author MANUEL SOTILLO
 *
 * Clase de gestión de aspecto gráfico de árbol de estructura
 */
import java.awt.Component;
import javax.swing.ImageIcon;
import javax.swing.JLabel;
import javax.swing.JTree;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeCellRenderer;

public class RenderArbol extends DefaultTreeCellRenderer {
    private static final long serialVersionUID = 1L;
    ImageIcon carpetaAbierta;
    ImageIcon carpetaCerrada;
    ImageIcon documento;
    JLabel etiqueta = new JLabel();

    public RenderArbol() {
        carpetaCerrada = new ImageIcon("folder-closed.GIF");
        carpetaAbierta = new ImageIcon("folder-open.GIF");
        documento = new ImageIcon("docs.GIF");
    }

    public Component getTreeCellRendererComponent(
        JTree tree,
        Object value,
        boolean sel,
        boolean expanded,
        boolean leaf,
        int row,
        boolean hasFocus) {

        super.getTreeCellRendererComponent(
            tree, value, sel,
            expanded, leaf, row,
            hasFocus);

        DefaultMutableTreeNode nodo = (DefaultMutableTreeNode)value;
        ElementoBD miElem = (ElementoBD) nodo.getUserObject();

        if(miElem.getTipo()==ElementoBD.COLECCION){
            if(expanded){
                etiqueta.setIcon(carpetaAbierta);
            }else{
                etiqueta.setIcon(carpetaCerrada);
            }
        }else{
            etiqueta.setIcon(documento);
        }

        if(sel){
            etiqueta.setForeground(java.awt.Color.RED);
        }else{
            etiqueta.setForeground(java.awt.Color.black);
        }
    }
}
```



```
etiqueta.setText(miElem.toString());  
  
return etiqueta;  
}  
}
```

## UsuarioTableModel.java

```
/*
 * TFC - SGBD XML Nativos
 * @author MANUEL SOTILLO
 *
 * Modelo de datos de tabla de usuarios
 */
import java.io.Serializable;
import java.util.ArrayList;
import javax.swing.table.AbstractTableModel;

public class UsuariosTableModel extends AbstractTableModel
    implements Serializable{

    private static final long serialVersionUID = 1L;
    final String[] columnNames={"Nombre",
                                "Password",
                                "Grupo primario",
                                "Es dba?"};

    ArrayList listaUsr = null;

    public UsuariosTableModel(ArrayList listaUsuarios) {
        super();
        this.listaUsr = listaUsuarios;
    }

    public String getColumnName(int col) {
        return columnNames[col];
    }

    public int getColumnCount() {
        return columnNames.length;
    }

    public int getRowCount() {
        return listaUsr.size();
    }

    public Object getValueAt(int arg0, int arg1) {

        Usuario usr = (Usuario)listaUsr.get(arg0);

        switch (arg1) {
            case 0: if(usr.getNombre()==null){
                    return (new String(""));
                }else{
                    return (new String(usr.getNombre()));
                }
            case 1: if(usr.getPassword()==null){
                    return (new String(""));
                }else{
                    return (new String(usr.getPassword()));
                }
            case 2: if(usr.getGrupoPrimario()==null){
                    return (new String(""));
                }else{
                    return (new String(usr.getGrupoPrimario()));
                }
            default : if(usr.getEsDBA()){
                    return(Boolean.TRUE);
                }else{
                    return(Boolean.FALSE);
                }
        }
    }
}
```

```
    }  
}  
  
public Class getColumnClass(int c) {  
    return getValueAt(0, c).getClass();  
}  
}
```

## GestorBD.java

```

/*****
 * TFC - SGBD XML Nativos
 * @author MANUEL SOTILLO
 *
 * Clase interfaz con el SGBD
 *****/
import java.io.File;
import java.util.ArrayList;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.xml.transform.OutputKeys;
import org.exist.security.User;
import org.exist.storage.DBBroker;
import org.exist.util.UTF8;
import org.exist.xmldb.CollectionImpl;
import org.exist.xmldb.UserManagementService;
import org.exist.xmldb.XQueryService;
import org.xmldb.api.DatabaseManager;
import org.xmldb.api.base.Collection;
import org.xmldb.api.base.CompiledExpression;
import org.xmldb.api.base.Database;
import org.xmldb.api.base.Resource;
import org.xmldb.api.base.ResourceSet;
import org.xmldb.api.base.XMLDBException;
import org.xmldb.api.modules.CollectionManagementService;

public class GestorBD{

    protected static String driver = "org.exist.xmldb.DatabaseImpl";
    public static String URI = "xmldb:exist://localhost:8080/exist/xmlrpc";
    private Database database;
    private String usuario;
    private String usuarioPwd;

    /*****
     * Constructor de la clase
     * @throws ExcepcionGestorBD si existe algún problema en la conexión
     *****/
    public GestorBD() throws ExcepcionGestorBD{
        this.database = null;
        this.usuario = "admin";
        this.usuarioPwd = "";
        conectarBD();
    }

    /*****
     * Efectúa la conexión con el SGBD
     *****/
    private void conectarBD() throws ExcepcionGestorBD{
        try {
            Class cl = Class.forName(driver);
            database = (Database) cl.newInstance();
            DatabaseManager.registerDatabase(database);
        } catch (ClassNotFoundException e) {
            throw new ExcepcionGestorBD(
                "No se encuentra la clase del driver");
        } catch (InstantiationException e) {
            throw new ExcepcionGestorBD(
                "Error instanciando el driver");
        } catch (IllegalAccessException e) {
            throw new ExcepcionGestorBD(
                "Se ha producido una IllegalAccessException");
        } catch (XMLDBException e) {
            throw new ExcepcionGestorBD(

```

```

        "error XMLDB :" + e.getMessage());
    }
}

/*****
 * Obtiene un árbol con la estructura de componentes de la coleccion
 *
 * @param collec Colección a mostrar el contenido, si es null
 *             devolvera la coleccion raiz
 * @return Arbol con la estructura de componentes de la coleccion
 * @throws ExcepcionGestorBD Si ocurre algún error en el proceso
 *****/
public DefaultMutableTreeNode obtenerEstructuraColeccion(String collec)
    throws ExcepcionGestorBD{

    CollectionImpl col;
    String nombrecol;
    String[] listaColecciones;
    String[] listaRecursos;
    DefaultMutableTreeNode contenido;

    try {
        if(collec==null){
            nombrecol = new String(DBBroker.ROOT_COLLECTION);
        }else{
            nombrecol = new String(collec);
        }

        col=(CollectionImpl) leerColeccion(nombrecol);

        contenido =
            new DefaultMutableTreeNode(
                new ElementoBD(
                    nombrecol.substring(nombrecol.lastIndexOf("/") + 1),
                    ElementoBD.COLECCION,
                    nombrecol));

        listaColecciones = col.listChildCollections();
        for(int i=0; i< listaColecciones.length; i++){
            contenido.add(
                obtenerEstructuraColeccion(nombrecol +
                    "/" + listaColecciones[i]));
        }

        listaRecursos = col.getResources();
        for(int i=0; i< listaRecursos.length; i++){
            int tiporec;
            Resource res = leerRecurso(col, listaRecursos[i]);

            if(res.getResourceType().equals("XMLResource")){
                tiporec=ElementoBD.RECURSO_XML;
            }else{
                tiporec=ElementoBD.RECURSO_BINARIO;
            }

            ElementoBD newElemento =
                new ElementoBD(listaRecursos[i],
                    tiporec,
                    nombrecol);
            contenido.add(new DefaultMutableTreeNode(newElemento));
        }

        return contenido;
    } catch (XMLDBException e) {
        throw new ExcepcionGestorBD(
            "error generando estructura bd: " +
            e.getMessage());
    }
}

```

```

}
}

/*****
 * Leemos una colección de la base de datos
 * @param colec Ruta de la colección a leer
 * @return Coleccion leida
 * @throws ExcepcionGestorBD si existe algun problema
 *****/
public Collection leerColeccion(String colec)
    throws ExcepcionGestorBD{

    Collection colRet=null;

    try {
        colRet = DatabaseManager.getCollection(URI + colec,
                                             usuario,
                                             usuarioPwd);
    } catch (XMLDBException e) {
        throw new ExcepcionGestorBD(
            "Error leyendo colección\n" + e.getMessage());
    }
    return colRet;
}

/*****
 * Leemos un recurso de una colección de la base de datos
 * @param colec Coleccion de la que leeremos
 * @param nombrerec Recurso a leer de la colección
 * @return Resource Recurso leido, null en caso contrario
 *****/
public Resource leerRecurso(Collection colec, String nombrerec){

    Resource res=null;
    try {
        res = (Resource)colec.getResource(nombrerec);
    } catch (XMLDBException e) {
        res = null;
    }
    return res;
}

/*****
 * Lanzar una consulta contra la base de datos
 * @param consulta Consulta a ejecutar
 * @param contexto Coleccion de contexto de la consulta, si es null
 *                 se usará la colección raíz del SGBD
 * @return ResultSet Resultado de la Query
 *****/
public ResultSet ejecutarQuery(String consulta, String contexto)
    throws ExcepcionGestorBD{

    ResultSet result=null;

    Collection col;
    try {
        if(contexto==null){
            col = DatabaseManager.getCollection(URI + DBBroker.ROOT_COLLECTION);
        }else{
            col = DatabaseManager.getCollection(URI + contexto);
        }
    }

    XQueryService service =
        (XQueryService)col.getService( "XQueryService", "1.0" );
    service.setProperty( OutputKeys.INDENT, "yes" );
    service.setProperty( OutputKeys.ENCODING, "UTF-8" );
    CompiledExpression compiled = service.compile( consulta );

```

```

        result = service.execute( compiled );

        // podria ser: result = service.query( consulta );
    } catch (XMLDBException e) {
        throw new ExcepcionGestorBD("Error ejecutando query: " +
            e.getMessage());
    }

    return result;
}

/*****
 * Añadir una nueva coleccion a la BB.DD
 * @param contexto Coleccion sobre la que insertaremos la nueva
 * @param newColec Nombre de la nueva coleccion a insertar
 *                (relativa a la coleccion de contexto)
 * @return Collection Nueva colección creada
 * @throws ExcepcionGestorBD Si existe un error en la inserción
 *****/
public Collection anadirColeccion(Collection contexto,
    String newColec)
    throws ExcepcionGestorBD{

    Collection newCollection=null;
    try {
        CollectionManagementService mgtService =
            (CollectionManagementService)contexto.getService(
                "CollectionManagementService",
                "1.0");
        newCollection = mgtService.createCollection(
            new String(UTF8.encode(newColec)));
    } catch (XMLDBException e) {
        throw new ExcepcionGestorBD(
            "Error añadiendo colección: " + e.getMessage());
    }

    return newCollection;
}

/*****
 * Borrar una nueva coleccion a la BB.DD
 * @param contexto Coleccion sobre la que insertaremos la nueva
 * @param newColec Nombre de la nueva coleccion a insertar
 *                (relativa a la coleccion de contexto)
 * @throws ExcepcionGestorBD Si existe un error en la inserción
 *****/
public void borrarColeccion(Collection contexto,
    String antColecc)
    throws ExcepcionGestorBD{

    try {
        CollectionManagementService mgtService =
            (CollectionManagementService)contexto.getService(
                "CollectionManagementService",
                "1.0");
        mgtService.removeCollection(antColecc);
    } catch (XMLDBException e) {
        throw new ExcepcionGestorBD(
            "Error eliminando colección: " + e.getMessage());
    }
}

/*****
 * Añadir una nuevo recurso a la BB.DD
 * @param contexto Coleccion sobre la que insertaremos el archivo
 * @param archivo Archivo a añadir como recurso
 * @param tipoRecurso Tipo del recurso a almacenar (binario ó XML

```

```

* @throws ExcepcionGestorBD Si existe un error en la inserción
*****/
public void anadirRecurso(Collection contexto,
                        File archivo,
                        int tipoRecurso)
                        throws ExcepcionGestorBD{

    try {
        String tipoRecursoStr=null;
        if(tipoRecurso==ElementoBD.RECURSO_BINARIO){
            tipoRecursoStr = "BinaryResource";
        }else{
            if(tipoRecurso==ElementoBD.RECURSO_XML){
                tipoRecursoStr = "XMLResource";
            }else{
                throw new ExcepcionGestorBD(
                    "Error añadiendo colección: " +
                    "tipo de recurso no valido");
            }
        }
        Resource nuevoRecurso =
            contexto.createResource(archivo.getName(), tipoRecursoStr);
        nuevoRecurso.setContent(archivo);
        contexto.storeResource(nuevoRecurso);
    } catch (XMLDBException e) {
        throw new ExcepcionGestorBD(
            "Error añadiendo recurso: " + e.getMessage());
    }
}

/*****
* Añadir un nuevo recurso a la BB.DD
* @param contexto Colección de la que borrarémos un recurso
* @param nombreRec Recurso a borrar
* @throws ExcepcionGestorBD Si existe un error en el borrado
*****/
public void borrarRecurso(Collection contexto, String nombreRec)
                        throws ExcepcionGestorBD{

    try {
        Resource recursoParaBorrar =
            leerRecurso(contexto, nombreRec);
        contexto.removeResource(recursoParaBorrar);
    } catch (XMLDBException e) {
        throw new ExcepcionGestorBD(
            "Error añadiendo colección: " + e.getMessage());
    }
}

/*****
* Obtiene una lista con los usuarios definidos en el SGBD
* @return ArrayList con la lista de Usuarios
* @throws ExcepcionGestorBD Si existe un error en la lectura
*****/
public ArrayList leerUsuarios() throws ExcepcionGestorBD{

    ArrayList listaUsr = new ArrayList();

    try {
        Collection col=
            (CollectionImpl)leerColeccion(DBBroker.ROOT_COLLECTION);
        UserManagementService service =
            (UserManagementService)col.getService(
                "UserManagementService", "1.0");
        User[] usrEnBD = service.getUsers();
        for (int i=0;i<usrEnBD.length;i++){
            listaUsr.add(new Usuario(usrEnBD[i].getName(),
                                    usrEnBD[i].getPassword(),
                                    usrEnBD[i].getPrimaryGroup(),

```



```
        usrEnBD[i].hasDbRole(),
        usrEnBD[i].getGroups());
    }
} catch (XMLDBException e) {
    throw new ExcepcionGestorBD(
        "Error leyendo usuarios: " + e.getMessage());
}

return listaUsr;
}

/*****
 * Método de prueba de clase
 * @param args
 *****/
public static void main(String[] args) {
    GestorBD gestor;
    try {
        gestor = new GestorBD();
        DefaultMutableTreeNode estructura=
            gestor.obtenerEstructuraColeccion("/db");
        System.out.println(estructura);
        System.out.println("numero de hijos : " +
            estructura.getChildCount());
    } catch (ExcepcionGestorBD e) {
        System.out.println("Error durante el proceso");
        System.out.println(e.getMessage());
    }
}
}
```

## DialogoVerUsuario.java

```

/*****
 * TFC - SGBD XML Nativos
 * @author MANUEL SOTILLO
 *
 * Dialogo detalle de usuarios
 *****/
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Rectangle;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.ArrayList;
import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextArea;
import javax.swing.table.TableColumnModel;

public class DialogoVerUsuarios extends JDialog {

    private static final long serialVersionUID = 1L;
    private JPanel panelContenido;
    private JButton botonAceptar;
    private JScrollPane panelTabla;
    private JScrollPane panelGrupos;
    private JTable tablaUsuarios;
    private UsuariosTableModel tableModelUsuario;
    private JTextArea areaGrupos;

    public DialogoVerUsuarios(JFrame pantPadre, final ArrayList listaUsuarios) {
        super(pantPadre);

        panelContenido = new JPanel();
        botonAceptar = new JButton();
        panelTabla = new JScrollPane();
        panelGrupos = new JScrollPane();
        tableModelUsuario = new UsuariosTableModel(listaUsuarios);
        tablaUsuarios = new JTable(tableModelUsuario);

        areaGrupos = new JTextArea();
        areaGrupos.setEditable(false);
        areaGrupos.setBounds(new Rectangle(10, 310, 500, 120));
        areaGrupos.setFont(new Font("Courier", Font.PLAIN, 12));
        panelGrupos = new JScrollPane();
        panelGrupos.setBounds(new Rectangle(10, 310, 500, 120));
        panelGrupos.setViewportView(areaGrupos);

        //--Gestor boton Aceptar
        botonAceptar.setText("Aceptar");
        botonAceptar.setBounds(new Rectangle(410, 450, 100, 20));
        botonAceptar.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                dispose();
            }
        });

        //tabla y componentes que usa
        TableColumnModel cm = tablaUsuarios.getColumnModel();
        cm.getColumn(0).setPreferredWidth(75);
        cm.getColumn(1).setPreferredWidth(75);
    }
}

```

```
cm.getColumnModel(2).setPreferredWidth(150);
cm.getColumnModel(3).setPreferredWidth(20);

panelTabla.setBounds(new Rectangle(10, 10, 500, 300));
panelTabla.setViewportView(tablaUsuarios);

panelContenido.add(botonAceptar);
panelContenido.add(panelTabla);
panelContenido.add(panelGrupos);
panelContenido.setSize(new Dimension(520, 500));
panelContenido.setLayout(null);

/--de entrada si hay algun usuario mostramos
/--los grupos del primero
if(listaUsuarios.size()>0){
    Usuario usr = (Usuario)listaUsuarios.get(0);
    mostrarGruposUsr(usr);
}

/--Gestor de pulsacion sobre tabla de usuarios
tablaUsuarios.addMouseListener(new MouseAdapter(){
    public void mouseClicked(MouseEvent e){
        int fila = tablaUsuarios.rowAtPoint(e.getPoint());
        if (fila > -1){
            Usuario usr = (Usuario)listaUsuarios.get(fila);
            mostrarGruposUsr(usr);
        }
    }
});

/--ventana principal
setSize(new Dimension(540, 510));
setTitle("Informacion de usuarios");
setContentPane(panelContenido);
setLayout(null);
setVisible(true);
}

/**
 * Actualiza el area de grupos del usuario con los grupos que
 * tiene asignado el usuario que recibe como parametro
 *
 * @param usr Usuario del que mostraremos los grupos
 */
private void mostrarGruposUsr(Usuario usr){

    areaGrupos.setText("");

    String[] listagrupos = usr.getGrupos();
    for(int i=0; i<listagrupos.length; i++){
        areaGrupos.append(listagrupos[i] + "\n");
    }
}
}
```

## EstructuraBDSwing.java

```

/*****
 * TFC - SGBD XML Nativos
 * @author MANUEL SOTILLO
 *
 * Interfaz gráfico acceso a SGBD
 *****/
import javax.swing.*;
import java.awt.Font;
import java.awt.Rectangle;
import java.io.File;
import java.util.ArrayList;

import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeModel;
import javax.swing.tree.TreeSelectionModel;
import org.xmldb.api.base.Collection;
import org.xmldb.api.base.Resource;
import org.xmldb.api.base.ResourceIterator;
import org.xmldb.api.base.ResourceSet;
import org.xmldb.api.base.XMLDBException;
import org.xmldb.api.modules.XMLResource;

public class EstructuraBDSwing extends JFrame {

    private static final long serialVersionUID = 1L;
    private JPanel panelPrincipal = null;
    private JScrollPane arbolScrollPane = null;
    private JScrollPane resulScrollPane = null;
    private JScrollPane queryScrollPane = null;
    private JTextArea areaResul=null;
    private JTextArea areaQuery=null;
    private JTree arbolEstruc=null;
    private GestorBD gestorBD=null;
    private DefaultMutableTreeNode estructura=null;

    private ImageIcon iconoEjecutarQuery=null;
    private JButton botonEjQuery=null;
    private ImageIcon iconoSalir=null;
    private JButton botonSalir=null;
    private ImageIcon iconoAnadirColecc=null;
    private JButton botonAnadirColecc=null;
    private ImageIcon iconoBorrarColecc=null;
    private JButton botonBorrarColecc=null;
    private ImageIcon iconoAnadirRecur=null;
    private JButton botonAnadirRecur=null;
    private ImageIcon iconoBorrarRecur=null;
    private JButton botonBorrarRecur=null;
    private ImageIcon iconoVerUsuarios=null;
    private JButton botonVerUsuarios=null;
    private ElementoBD elemSelec; //Ultimo elemento seleccionado en el arbol
    private DefaultMutableTreeNode nodoSelec;

    ArrayList listaUsuarios;

    /**
     * Método constructor de la clase.
     */
    public EstructuraBDSwing() {
        super();
        try {
            gestorBD=new GestorBD();
            elemSelec=null;
            nodoSelec=null;

```

```

        initialize();
    } catch (ExcepcionGestorBD e) {
        JOptionPane.showMessageDialog(this,
            "Error en carga inicial\n" +
            e.getMessage(),
            "Error",
            JOptionPane.ERROR_MESSAGE);
        System.exit(55);
    }
}

/**
 * Método de inicialización del JFrame
 * @return void
 * @throws ExcepcionGestorBD
 */
private void initialize() throws ExcepcionGestorBD {
    this.setSize(800, 600);
    this.setResizable(false);
    this.setTitle("Estructura BD Swing");
    this.setContentPane(inicializaPanel());
}

/**
 * Método que inicializa el panel principal de la aplicación
 * @return JPanel
 */
private JPanel inicializaPanel() throws ExcepcionGestorBD{

    //--boton Salir-----
    iconoSalir = new ImageIcon("exit.GIF");
    botonSalir = new JButton();
    botonSalir.setIcon(iconoSalir);
    botonSalir.setBounds(new Rectangle(720, 500, 55, 55));
    botonSalir.setFocusPainted(false);
    botonSalir.setToolTipText("Salir");

    //--Boton ejecutar query-----
    iconoEjecutarQuery = new ImageIcon("ejquery.GIF");
    botonEjQuery = new JButton();
    botonEjQuery.setIcon(iconoEjecutarQuery);
    botonEjQuery.setBounds(new Rectangle(720, 400, 55, 55));
    botonEjQuery.setFocusPainted(false);
    botonEjQuery.setToolTipText("Ejecutar consulta");

    //--Boton añadir coleccion-----
    iconoAnadirColecc = new ImageIcon("addfolder.GIF");
    botonAnadirColecc = new JButton();
    botonAnadirColecc.setIcon(iconoAnadirColecc);
    botonAnadirColecc.setBounds(new Rectangle(300, 500, 55, 55));
    botonAnadirColecc.setFocusPainted(false);
    botonAnadirColecc.setToolTipText("Ejecutar consulta");

    //--Boton borrar coleccion-----
    iconoBorrarColecc = new ImageIcon("deletefolder.GIF");
    botonBorrarColecc = new JButton();
    botonBorrarColecc.setIcon(iconoBorrarColecc);
    botonBorrarColecc.setBounds(new Rectangle(375, 500, 55, 55));
    botonBorrarColecc.setFocusPainted(false);
    botonBorrarColecc.setToolTipText("Borrar colección");

    //--Boton anadir recurso-----
    iconoAnadirRecur = new ImageIcon("addfile.GIF");
    botonAnadirRecur = new JButton();
    botonAnadirRecur.setIcon(iconoAnadirRecur);
    botonAnadirRecur.setBounds(new Rectangle(500, 500, 55, 55));
}

```

```
botonAnadirRecur.setFocusPainted(false);
botonAnadirRecur.setToolTipText("Añadir recurso");

/--Boton borrar recurso-----
iconoBorrarRecur = new ImageIcon("deletefile.GIF");
botonBorrarRecur = new JButton();
botonBorrarRecur.setIcon(iconoBorrarRecur);
botonBorrarRecur.setBounds(new Rectangle(575, 500, 55, 55));
botonBorrarRecur.setFocusPainted(false);
botonBorrarRecur.setToolTipText("Borrar recurso");

/--Ver usuarios-----
iconoVerUsuarios = new ImageIcon("usergroup.GIF");
botonVerUsuarios = new JButton();
botonVerUsuarios.setIcon(iconoVerUsuarios);
botonVerUsuarios.setBounds(new Rectangle(200, 500, 55, 55));
botonVerUsuarios.setFocusPainted(false);
botonVerUsuarios.setToolTipText("Ver usuarios");

/--Panel con arbol jerarquia BB.DD-----
estructura = gestorBD.obtenerEstructuraColeccion(null);

arbolEstruc = new JTree(new DefaultTreeModel(estructura));
arbolEstruc.setBounds(new Rectangle(0,0,300,300));
arbolEstruc.setCellRenderer(new RenderArbol());
arbolEstruc.getSelectionModel().setSelectionMode(
    TreeSelectionMode.SINGLE_TREE_SELECTION);

arbolScrollPane = new JScrollPane();
arbolScrollPane.setBounds(new Rectangle(10,20,300,350));
arbolScrollPane.setViewportView(arbolEstruc);

/--Panel de resultado-----
areaResul = new JTextArea();
areaResul.setEditable(false);
areaResul.setBounds(new Rectangle(0,0,460,350));
areaResul.setFont(new Font("Courier",Font.PLAIN,12));
resulScrollPane = new JScrollPane();
resulScrollPane.setBounds(new Rectangle(320,20,460,350));
resulScrollPane.setViewportView(areaResul);

/--Panel query-----
areaQuery = new JTextArea();
areaQuery.setEditable(true);
areaQuery.setBounds(new Rectangle(0,0,700,100));
queryScrollPane = new JScrollPane();
queryScrollPane.setBounds(new Rectangle(10,370,700,100));
queryScrollPane.setViewportView(areaQuery);

/--Panel principal-----
panelPrincipal = new JPanel();
panelPrincipal.setLayout(null);
panelPrincipal.add(resulScrollPane, null);
panelPrincipal.add(arbolScrollPane, null);
panelPrincipal.add(queryScrollPane, null);
panelPrincipal.add(botonEjQuery, null);
panelPrincipal.add(botonSalir, null);
panelPrincipal.add(botonAnadirRecur, null);
panelPrincipal.add(botonBorrarRecur, null);
panelPrincipal.add(botonAnadirColecc, null);
panelPrincipal.add(botonBorrarColecc, null);
panelPrincipal.add(botonVerUsuarios, null);

activarControles(ElementoBD.OTRO);

/**-----
 * Gestor de evento cierre de ventana
```

```

*-----*/
addWindowListener(new java.awt.event.WindowAdapter() {
    public void windowClosing(java.awt.event.WindowEvent evt) {
        System.exit(0);
    }
});

/**-----
 * Gestor de evento Seleccion de elemento en el arbol
 *-----*/
arbolEstruc.addTreeSelectionListener(
    new javax.swing.event.TreeSelectionListener() {
        public void valueChanged(javax.swing.event.TreeSelectionEvent accion) {

            nodoSelec =
                (DefaultMutableTreeNode)arbolEstruc.getLastSelectedPathComponent();

            //--verificamos si es null por si se llama al método sin haber
            //--seleccionado ningun nodo aun
            if(nodoSelec!=null){
                elemSelec = (ElementoBD)nodoSelec.getUserObject();
                Collection col;
                try {
                    col = gestorBD.leerColeccion(elemSelec.getColeccion());
                } catch (ExcepcionGestorBD el) {
                    col=null;
                }

                if(col==null){
                    areaResul.setText(";;;Error obteniendo coleccion!!!\n" +
                        elemSelec.getColeccion() );
                    activarControles(ElementoBD.OTRO);
                }else{
                    if(elemSelec.getTipo()==ElementoBD.COLECCION){
                        areaResul.setText(
                            "Coleccion\n" +
                            "-----" + "\n" +
                            "Nombre : " + elemSelec.getNombre() + "\n" +
                            "Coleccion: " + elemSelec.getColeccion() + "\n");
                        activarControles(ElementoBD.COLECCION);
                    }else{
                        Resource res =
                            gestorBD.leerRecurso(col, elemSelec.getNombre());
                        if(res==null){
                            areaResul.setText(
                                ";;;Error obteniendo recurso!!!\n" +
                                "Coleccion: " + elemSelec.getColeccion() + "\n" +
                                "Recurso : " + elemSelec.getNombre());
                            activarControles(ElementoBD.OTRO);
                        }else{
                            try {
                                activarControles(elemSelec.getTipo());

                                if(elemSelec.getTipo()==ElementoBD.RECURSO_XML){
                                    XMLResource xmlres = (XMLResource)res;
                                    areaResul.setText((String) xmlres.getContent());
                                }else{
                                    areaResul.setText("Recurso binario\n" +
                                        "-----" + "\n" +
                                        "Coleccion: " + elemSelec.getColeccion() + "\n" +
                                        "Recurso : " + elemSelec.getNombre());
                                }
                            } catch (XMLDBException e) {
                                areaResul.setText(
                                    ";;;Error accediendo al contenido del recurso!!!\n" +
                                    "Coleccion: " + elemSelec.getColeccion() + "\n" +
                                    "Recurso : " + elemSelec.getNombre());
                            }
                        }
                    }
                }
            }
        }
    }
);

```

```

        activarControles (ElementoBD.OTRO);
    }
}
}
}
} else{
    activarControles (ElementoBD.OTRO);
}
}
});

/**-----
 * Gestor de evento botón Ejecutar Query
 *-----*/
botonEjQuery.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent accion) {

        try {
            if(areaQuery.getText().length()>0){
                ResultSet resultado=null;
                areaResul.setText("");
                DefaultMutableTreeNode nodo =
                    (DefaultMutableTreeNode) arbolEstruc.getLastSelectedPathComponent();

                if(nodo==null){
                    resultado= gestorBD.ejecutarQuery(areaQuery.getText(),null);
                } else{
                    ElementoBD miElem = (ElementoBD)nodo.getUserObject();
                    resultado= gestorBD.ejecutarQuery(areaQuery.getText(),
                                                       miElem.getColeccion());
                }

                ResourceIterator iterator = resultado.getIterator();
                if(!iterator.hasMoreResources()){
                    areaResul.setText("La consulta no ha devuelto resultados");
                } else{
                    while(iterator.hasMoreResources()){
                        Resource res = iterator.nextResource();
                        areaResul.append((String)res.getContent() + "\n");
                    }
                }
            } else{
                areaResul.setText("No hay ninguna query escrita");
            }
        } catch (ExcepcionGestorBD e) {
            areaResul.setText(e.getMessage());
        } catch (XMLDBException e) {
            areaResul.setText(e.getMessage());
        }
    }
});

/**-----
 * Gestor de evento botón Añadir Recurso
 *-----*/
botonAnadirRecur.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent accion) {
        if(botonAnadirRecur.isEnabled()){
            File nuevoFichero = dialogoAnadirRecurso();
            if(nuevoFichero!=null){
                int tipoRecurso = dialogoSeleccionTipoRecurso();
                if(tipoRecurso!=ElementoBD.OTRO){
                    try {
                        Collection col = gestorBD.leerColeccion(elemSelec.getColeccion());
                        gestorBD.anadirRecurso(col, nuevoFichero, tipoRecurso);
                        nodoSelec.insert(new DefaultMutableTreeNode(
                            new ElementoBD(

```



```

        nuevoFichero.getName(),
            tipoRecurso,
            elemSelec.getColeccion()),0);
        arbolEstruc.updateUI();
        areaResul.setText("Recurso añadido correctamente");
    } catch (ExcepcionGestorBD e) {
        areaResul.setText("Error añadiendo en BD: " +
            e.getMessage());
    }
    } else{
        areaResul.setText("Añadir recurso cancelado");
    }
    } else{
        areaResul.setText("Añadir recurso cancelado");
    }
    } else{
        areaResul.setText("Añadir recurso deshabilitado");
    }
    }
    });

/**-----
 * Gestor de evento botón Borrar Recurso
 *-----*/
botonBorrarRecur.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent accion) {
        if(botonBorrarRecur.isEnabled()){
            try {
                Collection col = gestorBD.leerColeccion(elemSelec.getColeccion());

                int op =
                    dialogoConfirmacionBorrado("Desea borrar el recurso " +
                        elemSelec.getNombre());

                if(op==0){
                    gestorBD.borrarRecurso(col, elemSelec.getNombre());

                    DefaultMutableTreeNode padre = (
                        DefaultMutableTreeNode)nodoSelec.getParent();
                    padre.remove(nodoSelec);
                    arbolEstruc.updateUI();
                    activarControles(ElementoBD.OTRO);

                    areaResul.setText("Recurso : " +
                        elemSelec.getNombre() +
                        "\nborrado correctamente");

                    elemSelec=null;
                    nodoSelec=null;
                }
            } catch (ExcepcionGestorBD e) {
                areaResul.setText("Error borrando un recurso: " +
                    e.getMessage());
            }
        } else{
            areaResul.setText("Borrar recurso deshabilitado");
        }
    }
});

/**-----
 * Gestor de evento botón Añadir Coleccion
 *-----*/
botonAnadirColecc.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent accion) {
        if(botonAnadirColecc.isEnabled()){
            String nombNewCol = dialogoAnadirColeccion();
            if(nombNewCol!=null){

```

```

if(nombNewCol.length(>0)
try {
    Collection col =
        gestorBD.leerColeccion(elemSelec.getColeccion());

    //añade el nombre sin expandir, ya lo hace la función
    Collection newCol = gestorBD.anadirColeccion(col, nombNewCol);
    String nombreNewCol = (new String(newCol.getName()));
    String nombreExpandido =
        nombreNewCol.substring(nombreNewCol.lastIndexOf("/") + 1);

    //creando el elemento lo hacemos con el nombre expandido
    nodoSelec.insert(new DefaultMutableTreeNode(
        new ElementoBD(
            nombreExpandido,
            ElementoBD.COLECCION,
            elemSelec.getColeccion() + "/" + nombreExpandido), 0);
    arbolEstruc.updateUI();
    areaResul.setText("Colección añadida correctamente");
} catch (ExcepcionGestorBD e) {
    areaResul.setText("Error añadiendo colección: " +
        e.getMessage());
} catch (XMLDBException e) {
    areaResul.setText("Error añadiendo colección\n" +
        "Obteniendo nombre: " + e.getMessage());
}
else{
    areaResul.setText("No ha introducido ningún nombre");
}
} else{
    areaResul.setText("Añadir colección cancelada");
}
} else{
    areaResul.setText("Añadir colección deshabilitado");
}
}
});

/**-----
 * Gestor de evento botón Borrar Coleccion
 *-----*/
botonBorrarColecc.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent accion) {
        if(botonBorrarColecc.isEnabled()){
            try {
                Collection col =
                    gestorBD.leerColeccion(elemSelec.getColeccion());
                //getNombre devuelve el nombre expandido

                if(col.getParentCollection() != null){
                    int op =
                        dialogoConfirmacionBorrado("Desea borrar la colección " +
                            elemSelec.getColeccion());

                    if(op == 0) {
                        gestorBD.borrarColeccion(col.getParentCollection(),
                            elemSelec.getNombre());
                        DefaultMutableTreeNode padre =
                            (DefaultMutableTreeNode) nodoSelec.getParent();
                        padre.remove(nodoSelec);
                        arbolEstruc.updateUI();
                        activarControles(ElementoBD.OTRO);

                        areaResul.setText("Colección : " + elemSelec.getNombre() +
                            "\nborrada correctamente");

                        elemSelec = null;

```

```

        nodoSelec=null;
    }
    }else{
        areaResul.setText("No se pùede borrar la coleccion raiz");
    }
} catch (ExcepcionGestorBD e) {
    areaResul.setText("Error añadiendo coleccion: " +
        e.getMessage());
} catch (XMLDBException e) {
    areaResul.setText("Error añadiendo coleccion\n" + "" +
        "recuperando coleccion padre: " +
        e.getMessage());
}
}else{
    areaResul.setText("Borrar coleccion deshabilitado");
}
}
});

/**-----
 * Gestor de evento botón Salir
 *-----*/
botonVerUsuarios.addMouseListener(new java.awt.event.MouseAdapter(){
    public void mouseClicked(java.awt.event.MouseEvent accion) {

        try {
            listaUsuarios = gestorBD.leerUsuarios();
            dialogoVerUsuarios(listaUsuarios);
        } catch (ExcepcionGestorBD e) {
            areaResul.setText("Error leyendo usuarios\n" +
                e.getMessage());
        }
    }
});

/**-----
 * Gestor de evento botón Salir
 *-----*/
botonSalir.addMouseListener(new java.awt.event.MouseAdapter(){
    public void mouseClicked(java.awt.event.MouseEvent accion) {
        System.exit(0);
    }
});

return panelPrincipal;
}

/**
 * Activa los botones de la aplicación en función del elemento indicado
 * @param elemSelec tipo de elemento seleccionado en el árbol
 */
private void activarControles(int elemSelec) {

    if(elemSelec==ElementoBD.COLECCION) {
        botonAnadirColecc.setEnabled(true);
        botonBorrarColecc.setEnabled(true);
        botonAnadirRecur.setEnabled(true);
        botonBorrarRecur.setEnabled(false);
    }else{
        if(elemSelec==ElementoBD.RECURSO_XML ||
            elemSelec==ElementoBD.RECURSO_BINARIO) {
            botonAnadirColecc.setEnabled(false);
            botonBorrarColecc.setEnabled(false);
            botonAnadirRecur.setEnabled(false);
            botonBorrarRecur.setEnabled(true);
        }else{
            botonAnadirColecc.setEnabled(false);

```

```
        botonBorrarColecc.setEnabled(false);
        botonAnadirRecur.setEnabled(false);
        botonBorrarRecur.setEnabled(false);
    }
}

/**
 * Muestra el dialogo modal para pedir el nombre de la nueva colección
 * @return Nombre de la coleccion a añadir
 */
private String dialogoAnadirColeccion(){
    String nombre =
        JOptionPane.showInputDialog(this,
                                    "Nombre de la colección",
                                    "Añadir colección",
                                    JOptionPane.QUESTION_MESSAGE);

    if(nombre!=null){
        return nombre.trim();
    }else{
        return nombre;
    }
}

/**
 * Abre un selector de fichero estándar para elegir el fichero a
 * añadir a la coleccion
 * @return File fichero seleccionado
 */
private File dialogoAnadirRecurso(){
    File fichero=null;
    int rv;

    JFileChooser fc = new JFileChooser();
    fc.setMultiSelectionEnabled(false);
    fc.setDialogType(JFileChooser.OPEN_DIALOG);
    rv = fc.showOpenDialog(this);
    if(rv == JFileChooser.APPROVE_OPTION) {
        fichero=fc.getSelectedFile();
    }

    return fichero;
}

/**
 * Muestra el dialo de selección de tipo de recurso a añadir
 * @return tipo de recurso seleccionado
 */
private int dialogoSeleccionTipoRecurso(){
    int tiporec;
    String[] valores = { "XML", "Binario"};
    String valorSelec =
        (String)JOptionPane.showInputDialog(this,
                                            "Selecciona un tipo de recurso",
                                            "Tipo de recurso",
                                            JOptionPane.QUESTION_MESSAGE,
                                            null,
                                            valores,
                                            valores[0]);

    if(valorSelec==null){
        tiporec=ElementoBD.OTRO;
    }else{
        if(valorSelec.equals(valores[0])){
            tiporec=ElementoBD.RECURSO_XML;
        }else{

```

```
        tiporec=ElementoBD.RECURSO_BINARIO;
    }
}
return tiporec;
}

/**
 * Muestra el dialogo con información de los
 * usuarios
 */
private void dialogoVerUsuarios(ArrayList listaUsuarios){

    new DialogoVerUsuarios(this, listaUsuarios);

}

/**
 * dialogo de confirmacion de borrado
 * */
private int dialogoConfirmacionBorrado(String mensaje){
    String string1 = "Si, Borrar";
    String string2 = "No, Cancelar";
    Object[] options = {string1, string2};
    int opBorrado = JOptionPane.showOptionDialog(this,
        mensaje,
        "Borrado",
        JOptionPane.YES_NO_OPTION,
        JOptionPane.QUESTION_MESSAGE,
        null,
        options,
        string1);
    return (opBorrado);
}

/**
 * Metodo principal de la aplicación
 */
public static void main(String args[]) {
    EstructuraBDSwing cliente = new EstructuraBDSwing();
    cliente.setVisible(true);
}
}
```

## Bibliografía

### W3C

**XML Path Lenguaje (XPath) Version 1.0**

Recomendación W3C lenguaje XPath versión 1.0

<http://www.w3.org/TR/xpath>

**XQuery 1.0 and XPath 2.0 Datamodel (XDM)**

Definición del modelo de datos para XQuery 1.0 y XPath 2.0

<http://www.w3.org/TR/xpath-datamodel/>

**XML Path Lenguaje (XPath) Version 2.0**

Recomendación W3C lenguaje XPath versión 2.0

<http://www.w3.org/TR/xpath20/>

**XQuery 1.0: An XML Query lenguaje**

Recomendación W3C lenguaje consulta sobre datos XML

<http://www.w3.org/TR/xquery/>

**XML Schema Part 2: Datatypes Second Edition**

Definición de tipos de datos primitivos de la recomendación XML Schema

<http://www.w3.org/TR/xmlschema-2/#built-in-primitive-datatypes>

**XQuery 1.0 and XPath 2.0 Functions and operators**

Referencia de funciones para XPath 2.0 y XQuery 1.0

<http://www.w3.org/TR/xpath-functions/>

**XQuery Update Facility 1.0**

Recomendación W3C sobre extensión de actualización para XQuery

<http://www.w3.org/TR/xquery-update-10/>

**XML Syntax for XQuery 1.0 (XQueryX)**

Recomendación sobre sintaxis XML para sentencias XQuery

<http://www.w3.org/TR/xqueryx/>

### XUpdate

**XUpdate: XML Update Lenguaje**

Proposición de estándar de lenguaje de actualización de datos XML

<http://xmldb-org.sourceforge.net/xupdate/index.html>

### eXist

**Página principal eXist**

<http://exist.sourceforge.net/>

**XQuery Update Extensions**

Referencia de extensión de actualización de datos XML proporcionada por eXist

[http://exist.sourceforge.net/update\\_ext.html](http://exist.sourceforge.net/update_ext.html)

**Referencia implementación XMLDB**

Javadoc correspondientes a la implementación de XML:DB realizada por eXist

<http://exist.sourceforge.net/api/index.html>

## XML:DB

**XML:DB Database API**

Proposición de estándar de API de acceso a SGBD XML nativos

<http://xmldb-org.sourceforge.net/xapi/xapi-draft.html>

## XQJ

**JSR-000225 XQuery API for Java (XQJ)**

Proposición de estándar de API de acceso a SGBD XML nativos

<http://jcp.org/aboutJava/communityprocess/pr/jsr225/index.html>

**XQJ – The JDBC for XML**

Tutorial de uso XQJ

[http://www.datadirect.com/developer/xquery/docs/xqj\\_tutorial.pdf](http://www.datadirect.com/developer/xquery/docs/xqj_tutorial.pdf)

## Oracle

**XML Technology Center**

Centro de documentación Oracle para desarrollo de aplicaciones usando XML

<http://www.oracle.com/technology/tech/xml/index.html>

**Oracle 9i Applications developers guide**

Guía de desarrollo de aplicaciones Oracle 9.0.1

[http://download-uk.oracle.com/docs/cd/A97329\\_03/web.902/a88894.pdf](http://download-uk.oracle.com/docs/cd/A97329_03/web.902/a88894.pdf)

**Oracle Database 11g XML DB**

Novedades en Oracle XML DB 11g

<http://www.oracle.com/technology/tech/xml/xmldb/Current/11g%20new%20features.ppt.pdf>

## DB2

**XML FOR DB2 Information Integration**

Manual IBM para el tratamiento de datos XML en DB2

<http://www.redbooks.ibm.com/redbooks/pdfs/sq246994.pdf>

**XML and DB2**

Características Extender DB2

[http://www.xml-finland.org/archive/xmldb\\_juna/ibm.pdf](http://www.xml-finland.org/archive/xmldb_juna/ibm.pdf)

Tamino

**Tamino XML Server Documentation**

Centro de documentación de Software AG Tamino

<http://documentation.softwareag.com/crossvision/ins441/overview.htm>

Otros

**Beginning XML 3er Edition**

Editorial: Wrox

Autores: David Hunter, Andrew Watt, Jeff Rafter, John Buckett, Danny Ayers, Nicholas Chase, Joe Fawcett, Tom Gaven, Hill Patterson

ISBN: 0-7645-7077-3

**XML Data Management. Native XML and XML-Enabled Database Systems**

Editorial: Addison Wesley

Autores: Akmal B. Chaudhri, Awais Rashid, Roberto Zicari

ISBN: 0-201-84452-4