M6 – UF3
PERSISTENCIA EN BD NATIVAS
XML

Eduard Lara

INDICE

1. ACCESO FICHEROS XML CON DOM

- XML es un metalenguaje. Es decir, define lenguajes. Nos permite jerarquizar y estructurar la información.
- Los ficheros XML se pueden utilizar para proporcionar datos a una aplicación. Tienen la gran ventaja de ser archivos de texto lo que permite exportar y recuperar datos de una plataforma a otra (solo es necesario conocer la codificación de los caracteres)
- Para leer los ficheros XML se utiliza un procesador XML o “parser”. Los procesadores más utilizados son: DOM y SAX.
- Los procesadores son independientes del lenguaje de programación y existen versiones particulares para Java, VisalBasic, C, etc.

1. ACCESO FICHEROS XML CON DOM

- DOM: Modelo de Objetos de Documento
- Se trata de una API para manipulación de documentos XML y HTML
- Almacena toda la estructura de un documento en memoria en forma de árbol; con nodos padre, nodos hijo y nodos finales (que no tienen descendientes)
- Requiere una buena cantidad de recursos de memoria y tiempo sobre todo si los ficheros XML a procesar son bastante grandes y complejos

1. ACCESO FICHEROS XML CON DOM

- Para poder trabajar con DOM en Java necesitamos las clases e interfaces que componen el paquete **org.w3c.dom** y el paquete **javax.xml.parsers**.
- Las dos clases más importantes son **DocumentBuilderFactory** y **DocumentBuilder**.
- Los programas Java que utilicen DOM necesitan estas y otras interfaces:
  - **Document** → Es un objete que equivale a un ejemplar de un documento XML. Permite crear nuevos nodos en el documento
  - **Element** → Cada elemento del documento XML tiene un equivalente en un objeto de este tipo. Expone propiedades y métodos para manipular los elementos del documento y sus atributos
  - **NodeList** → Contiene una lista con los nodos hijos de un nodo
  - **Attr** → Permite acceder a los atributos de un nodo.
  - **Text** → Son los datos carácter de un elemento

1. ACCESO FICHEROS XML CON DOM

Escritura de archivos – Ejemplo 1

```
public static void main (String args[]) throws IOException {
    File fichero = new File ("AleatorioEmpleado.dat");
    RandomAccessFile file = new RandomAccessFile(fichero, "r");
    int id, dep, posicion=0;
    Double salario;
    char apellido[] = new char[10], aux;

    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    try {
        DocumentBuilder builder = factory.newDocumentBuilder();
        DOMImplementation implementation = builder.getDOMImplementation();
        Document document = implementation.createDocument (null,"Empleados", nu
        document.setXmlVersion("1.0");


        for ( ; ;){
            file.seek(posicion);
            id = file.readInt();
            for (int i = 0; i < apellido.length ; i++) {
                aux =file.readChar();
                apellido[i] = aux;
            }
            String apellidos = new String (apellido);
            dep = file.readInt();
            salario = file.readDouble();
            if (id>0) {
                Element raiz = document.createElement ("empleado");
                document.getDocumentElement().appendChild(raiz);
                CrearElemento ("id", Integer.toString(id), raiz, document);
                CrearElemento ("apellido",apellidos.trim(), raiz, document);
                CrearElemento ("dep", Integer.toString(dep), raiz, document);
                CrearElemento ("salario", Double.toString(salario),raiz, docume
            }
            posicion = posicion + 36;
            if (file.getFilePointer() == file.length()) break;
        }

        Source source = new DOMSource (document);
        Result result = new StreamResult (new java.io.File ("Empleados.xml"));
        Transformer transformer = TransformerFactory.newInstance().newTransformer();

        transformer.transform (source, result);
    } catch (Exception e ) {
        System.err.println ("Error: " + e);
    }
    file.close();
}

static void CrearElemento (String datoEmpleado, String valor, Element raiz, Document document) {
    Element elem = document.createElement (datoEmpleado);
    Text text = document.createTextNode(valor);
    raiz.appendChild (elem);
    elem.appendChild (text);
}
```

1. ACCESO FICHEROS XML CON DOM

Escritura de archivos – Ejemplo 1

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
try {
    DocumentBuilder builder = factory.newDocumentBuilder();
    DOMImplementation implementation = builder.getDOMImplementation();
    Document document = implementation.createDocument (null,"Empleados", null);
    document.setXmlVersion("1.0");
```

Se crea una instancia de DocumentBuilderFactory para construir el parser. Se debe encerrar entre try-catch porque se puede producir la excepción ParserConfigurationException.

Creamos un documento vacío de nombre *document* con el nodo raíz de nombre *Empleados* y asignamos la versión XML. La interfaz DOMImplementation permite crear objetos Document con nodo raíz*

El método CrearElemento genera los 4 nodos-hijo (<id>, <apellido>,<dep> y <salario>) de cada nodo <Empleado>. Primero genera el elemento (**Element**), después el texto o valor (**Text**) y los asocia con la raiz o nodo padre. Recibe como parámetros sus textos o valores que deben estar en formato String, el nodo al que se va añadir (*raiz*) y el documento (*document*). Luego lo vemos con más detalle. Para crear un elemento usamos el método **createElement (String)** llevando como parámetro el nombre que se pone entre las etiquetas < y >.

```
static void CrearElemento (String datoEmpleado, String valor, Element raiz, Document document) {
    Element elem = document.createElement (datoEmpleado);
    Text text = document.createTextNode(valor);
    raiz.appendChild (elem);
    elem.appendChild (text);
}
```

1. ACCESO FICHEROS XML CON DOM

Escritura de archivos – Ejemplo 1

La especificación DOM no define ningún mecanismo para generar un fichero XML a partir de un árbol DOM. Para eso usaremos el paquete **javax.xml.transform** que permite especificar una fuente y un resultado. La fuente y el resultado pueden ser ficheros, flujos de datos o nodos DOM entre otros. Primero crearemos la fuente (**Source**) XML a partir del documento (**document**); después se crea el resultado (**Result**) en el fichero *Empleados.xml.* A continuación, se obtiene un **TransformerFactory** y se realiza la transformación del documento al fichero

```
    Source source = new DOMSource (document);
    Result result = new StreamResult (new java.io.File ("Empleados.xml"));
    Transformer transformer = TransformerFactory.newInstance().newTransformer();
    transformer.transform (source, result);
} catch (Exception e ) {
    System.err.println ("Error: " + e);
}
file.close();
```

1. ACCESO FICHEROS XML CON DOM

Escritura de archivos – Ejemplo 2

```
try {
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    DocumentBuilder db = dbf.newDocumentBuilder();
    Document doc = db.newDocument();

    // Definimos el elemento raíz del documento
    Element eRaiz = doc.createElement("concesionario");
    doc.appendChild(eRaiz);

    // Definimos el nodo que contendrá los elementos
    Element eCoche = doc.createElement("coche");
    eRaiz.appendChild(eCoche);

    // Atributo para el nodo coche
    Attr attr = doc.createAttribute("id");
    attr.setValue("1");
    eCoche.setAttributeNode(attr);

    // Definimos cada uno de los elementos y le asignamos un valor
    Element eMarca = doc.createElement("marca");
    eMarca.appendChild(doc.createTextNode("Renault"));
    eCoche.appendChild(eMarca);

    Element eModelo = doc.createElement("modelo");
    eModelo.appendChild(doc.createTextNode("Megano"));
    eCoche.appendChild(eModelo);

    Element eCilindrada = doc.createElement("cilindrada");
    eCilindrada.appendChild(doc.createTextNode("1.5"));
    eCoche.appendChild(eCilindrada);

    // Clases necesarias finalizar la creación del archivo XML
    TransformerFactory transformerFactory = TransformerFactory.newInstance();
    Transformer transformer = transformerFactory.newTransformer();
    DOMSource source = new DOMSource(doc);
    StreamResult result = new StreamResult(new File("concesionario.xml"));

    transformer.transform(source, result);
} catch(Exception e) {
      e.printStackTrace();
}
```

concesionario.xml

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <concesionario>
3      <coche id="1">
4          <marca>Renault</marca>
5          <modelo>Megano</modelo>
6          <cilindrada>1.5</cilindrada>
7      </coche>
8  </concesionario>
```

1. ACCESO FICHEROS XML CON DOM

Lectura de archivos - Ejemplo 1

```java
public static void  main(String args[]) {
    File file = new File("concesionario.xml");
    Document doc=null;
    try {
        DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
        DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
        doc = dBuilder.parse(file);
    } catch(Exception e) {
        e.printStackTrace();
    }
    doc.getDocumentElement().normalize();
    NodeList nList = doc.getElementsByTagName("coche");
    System.out.println("Número de coches: " + nList.getLength());

    for(int temp = 0; temp < nList.getLength(); temp++) {
        Node nNode = nList.item(temp);
        if(nNode.getNodeType() == Node.ELEMENT_NODE) {
            Element eElement = (Element) nNode;
            System.out.println("\nCoche id: " + eElement.getAttribute("id"));
            System.out.println("Marca: " + eElement.getElementsByTagName("marca").item(0).getTextContent());
            System.out.println("Modelo: " + eElement.getElementsByTagName("modelo").item(0).getTextContent());
            System.out.println("Cilindrada: " + eElement.getElementsByTagName("cilindrada").item(0).getTextContent());
        }
    }
}
```

concesionario.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<concesionario>
  <coche id="1">
    <marca>Renault</marca>
    <modelo>Megane</modelo>
    <cilindrada>1.5</cilindrada>
  </coche>
  <coche id="2">
    <marca>Seat</marca>
    <modelo>León</modelo>
    <cilindrada>1.6</cilindrada>
  </coche>

  <coche id="3">
    <marca>Suzuki</marca>
    <modelo>Vitara</modelo>
    <cilindrada>1.9</cilindrada>
  </coche>
</concesionario>
```

1. ACCESO FICHEROS XML CON DOM

Institut Joan d'Àustria

Lectura de archivos - Ejemplo 1

- A la hora de leer un archivo XML a través de DOM debemos instanciar una serie de clases antes de poder tratar el fichero. Primero utilizaremos la conocida clase File para cargar nuestro fichero.
- Posteriormente y ya dentro de un try/catch (para tratar los excepciones) parsearemos el fichero con estas clases: Utilizaremos una instancia de DocumentBuilderFactory para poder construir el parser con DocumentBuilder y Document, y cargar el documento con el método **parse()**

```
File file = new File("concesionario.xml");
Document doc=null;
try {
    DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
    DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
    doc = dBuilder.parse(file);
} catch(Exception e) {
    e.printStackTrace();
}
```

1. ACCESO FICHEROS XML CON DOM

Lectura de archivos - Ejemplo 1

- A continuación usaremos getDocumentElement() para acceder al nodo raíz del documento y normalize() para elimina nodos vacíos si los hubiera
- Para obtener la lista de nodos (NodeList) de todo el documento, utilizaremos el método Document.getElementsByTagName ("coche").
- Por último, se realiza un bucle para recorrer esa lista de nodos. Por cada nodo se obtienen sus etiquetas y sus valores.

```
doc.getDocumentElement().normalize();
NodeList nList = doc.getElementsByTagName("coche");
System.out.println("Número de coches: " + nList.getLength());

for(int temp = 0; temp < nList.getLength(); temp++) {
    Node nNode = nList.item(temp);
    if(nNode.getNodeType() == Node.ELEMENT_NODE) {
        Element eElement = (Element) nNode;
        System.out.println("\nCoche id: " + eElement.getAttribute("id"));
        System.out.println("Marca: " + eElement.getElementsByTagName("marca").item(0).getTextContent());
        System.out.println("Modelo: " + eElement.getElementsByTagName("modelo").item(0).getTextContent());
        System.out.println("Cilindrada: " + eElement.getElementsByTagName("cilindrada").item(0).getTextContent());
    }
}
```

1. ACCESO FICHEROS XML CON DOM

Lectura de archivos - Ejemplo 2

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
try {
    DocumentBuilder builder = factory.newDocumentBuilder();
    Document document = builder.parse(new File ("empleados.xml"));
    document.getDocumentElement().normalize();

    System.out.printf ("Elemento raiz : %s %n", document.getDocumentElement().getNodeName());

    NodeList empleados = document.getElementsByTagName("EMP_ROW");
    System.out.printf ("Nodos empleado a recorrer: %d %n", empleados.getLength());
    for (int i = 0; i < empleados.getLength(); i++) {
        Node emple = empleados.item(i);
        if (emple.getNodeType() == Node.ELEMENT_NODE){
            Element elemento = (Element) emple;
            System.out.printf("ID = %s %n",
                    elemento.getElementsByTagName("EMP_NO").item(0).getTextContent());
            System.out.printf(" * Apellido = %s %n",
                    elemento.getElementsByTagName("APELLIDO").item(0).getTextContent());
            System.out.printf(" * Departamento = %s %n",
                    elemento.getElementsByTagName("DEPT_NO").item(0).getTextContent());
            System.out.printf(" * Salario = %s %n",
                    elemento.getElementsByTagName("SALARIO").item(0).getTextContent());
        }
    }
}catch (Exception e) {
    e.printStackTrace();
}
```

empleados.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<EMPLEADOS>
<TITULO>DATOS DE LA TABLA EMPLE</TITULO>
  <EMP_ROW>
    <EMP_NO>7369</EMP_NO>
    <APELLIDO>SANCHEZ</APELLIDO>
    <OFICIO>EMPLEADO</OFICIO>
    <FECHA_ALT>1990-12-17</FECHA_ALT>
    <SALARIO>1040</SALARIO>
    <DEPT_NO>20</DEPT_NO>
  </EMP_ROW>
  <EMP_ROW>
    <EMP_NO>7499</EMP_NO>
    <APELLIDO>ARROYO</APELLIDO>
    <OFICIO>VENDEDOR</OFICIO>
    <FECHA_ALT>1990-02-20</FECHA_ALT>
    <SALARIO>1500</SALARIO>
    <COMISION>390</COMISION>
    <DEPT_NO>30</DEPT_NO>
  </EMP_ROW>
  <EMP_ROW>
    <EMP_NO>7521</EMP_NO>
    <APELLIDO>SALA</APELLIDO>
    <OFICIO>VENDEDOR</OFICIO>
    <FECHA_ALT>1991-02-22</FECHA_ALT>
    <SALARIO>1625</SALARIO>
    <COMISION>650</COMISION>
    <DEPT_NO>30</DEPT_NO>
  </EMP_ROW>
```

1. PRACTICA XML CON DOM

Practica 1

Crea un programa en java que genere un documento XML que contenga varios elementos como el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
- <Albumes>
  - <album>
      <autor>Joaquín Sabina</autor>
      <titulo>Nos sobran los motivos</titulo>
      <formato>MP3</formato>
    </album>
  - <album>
      <autor> Camaron de la Isla </autor>
      <titulo> La leyenda del tiempo </titulo>
      <formato> MP3 </formato>
    </album>
</Albumes>
```

Practica 2

Crea un programa Java que lea el documento anterior y muestre toda la información que contenga.

2. ACCESO FICHEROS XML CON SAX

- SAX (API Simple para XML) es un conjunto de clases e interfaces que ofrecen una herramienta muy útil para el procesamiento de documentos XML. Permite analizar documentos de forma secuencial (es decir, no carga todo el fichero en memoria como hace DOM), esto implica:
  - Poco consumo de memoria aunque los documentos sean de gran tamaño
  - Impide tener una visión global del documento
  - El acceso es secuencial NO aleatorio
  - SAX NO permite generar archivos XML
- Por tanto, SAX es útil cuando queremos buscar información en ficheros grandes XML o cuando no tenemos acceso completo al documento (acceso mediante *Stream*)
- SAX es una API totalmente escrita en Java e incluida dentro del JRE que nos permite crear nuestro propio parser de XML

2. ACCESO FICHEROS XML CON SAX

- La lectura de un documento XML produce eventos que ocasiona la llamada a métodos:

Documento XML (alumnos.xml)	**Métodos asociados a eventos del documento**
<?xml version=”1.0”?>	startDocument()
<listadealumnos>	startElement()
<alumno>	startElement()
<nombre>	startElement()
Juan	characters()
</nombre>	endElement()
<edad>	startElement()
19	characters()

2. ACCESO FICHEROS XML CON SAX

</edad>	endElement()
</alumno>	endElement()
<alumno>	startElement()
<nombre>	startElement()
María	characters()
</nombre>	endElement()
<edad>	startElement()
20	characters()
</edad>	endElement()
</alumno>	endElement()
</listadealumnos>	endElement()
	endDocument()

2. ACCESO FICHEROS XML CON SAX

- Como se observa en la tabla:
  - La etiqueta de inicio (<?xml versión = “1.0”?>) → provoca el evento **startDocument()**
  - El final de document → provoca el evento **endDocument()**
  - La etiqueta de inicio de un elemento → provoca el evento **startElement()**
  - La etiqueta de final de un elemento → provoca el evento **endElement()**
  - Los caracteres entre etiquetas → provocan el evento **characters()**
- Los objetos que poseen los métodos que tratarán los eventos son:
  - **ContentHandler** → recibe las notificaciones de los eventos que ocurren en el documento
  - **DTDHandler** → recoge eventos relacionados con la DTD (Definición de Tipo de Documento)
  - **ErrorHandler** → define métodos de tratamientos de errores
  - **EntityResolver** → sus métodos se llaman cada vez que hay una referencia a una entidad
  - **DefaultHandler** → clase que provee una implementación por defecto para todos sus métodos, el programador definirá los métodos que serán utilizados por el programa. Esta clase es la que extenderemos para poder crear nuestro parser de XML. En el ejemplo que veremos a continuación, la clase se llama *GestionContenido* y se tratan solo los eventos básicos: inicio y fin de documento, inicio y fin de etiqueta encontrada, encuentra datos carácter.

2. ACCESO FICHEROS XML CON SAX

Institut Joan d'Àustria

Ejemplo lectura SAX

```
public static void main(String[] args) throws Exception {
    //A continuación se crea objeto procesador XML - XMLReader
    // Durante la creación de este objeto se puede producir una excepción SAXException.
    SAXParserFactory parserFactory = SAXParserFactory.newInstance();
    SAXParser parser = parserFactory.newSAXParser();
    XMLReader procesadorXML = parser.getXMLReader();

    // A continuación, mediante setContentHandler establecemos que la clase que gestiona los
    //eventos provocados por la  lectura del XML será GestionContenido
    SaxReader gestor = new SaxReader();
    procesadorXML.setContentHandler(gestor);
    // Por último, se define el fichero que se va leer mediante InputSource y se procesa el
    // documento XML mediante el método parse() de XMLReader */
    InputSource fileXML = new InputSource ("concesionario.xml");
    procesadorXML.parse(fileXML);
}
```

2. ACCESO FICHEROS XML CON SAX

Ejemplo lectura SAX

La clase SaxReader implementa los métodos necesarios para crear nuestro parser de XML.

Es decir, definimos los métodos que serán llamados al provocarse los eventos comentados anteriormente: startDocument, startElement, characters, etc.

Si quisieramos tratar más eventos definiríamos el método asociado en esta clase.

SaxReader.java

```
2
3  import org.xml.sax.Attributes;
6
7  public class SaxReader extends DefaultHandler {
8      public SaxReader(){
9          super();
10     }
11     public void startDocument() throws SAXException {
12         System.out.println("**********************************************************************" );
13         System.out.println("Comienzo del documento XML");
14     }
15
16     public void endDocument() throws SAXException {
17         System.out.println("Final del documento XML");
18         System.out.println("**********************************************************************" );
19     }
20
21     public void endElement(String uri, String localName, String qName) throws SAXException {
22         System.out.println("---END ELEMENT (<"+qName+">)--------------------------------" );
23     }
24
25     public void characters(char[] ch, int start, int length) throws SAXException {
26         String contenido = new String(ch, start, length).trim();
27         if(contenido.length() > 0){
28             System.out.println("-----El valor del campo es: " + new String(ch, start, length));
29         }
30     }
31
32     public void startElement(String uri, String localName, String qName, Attributes attributes)  throws SAXException {
33         System.out.println("---START ELEMENT (<"+qName+">)--------------------------------" );
34         if(attributes.getLength()>0){
35             System.out.print("----Atributos de la etiqueta :" );
36             for (int i=0;i<attributes.getLength();i++){
37                 System.out.println("----"+attributes.getQName(i)+":"+attributes.getValue(i)+" ");
38             }
39         }
40     }
41 }
42
```

2. ACCESO FICHEROS XML CON SAX

Ejemplo lectura SAX

concesionario.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<concesionario>
  <coche id="1">
    <marca>Renault</marca>
    <modelo>Megane</modelo>
    <cilindrada>1.5</cilindrada>
  </coche>
  <coche id="2">
    <marca>Seat</marca>
    <modelo>León</modelo>
    <cilindrada>1.6</cilindrada>
  </coche>

  <coche id="3">
    <marca>Suzuki</marca>
    <modelo>Vitara</modelo>
    <cilindrada>1.9</cilindrada>
  </coche>
</concesionario>

```

```
**********************************************************************
Comienzo del documento XML
---START ELEMENT (<concesionario>)---------------------------------
---START ELEMENT (<coche>)-----------------------------------
----Atributos de la etiqueta :----id:1
---START ELEMENT (<marca>)-----------------------------------
-----El valor del campo es: Renault
---END ELEMENT (<marca>)-------------------------------------
---START ELEMENT (<modelo>)-----------------------------------
-----El valor del campo es: Megane
---END ELEMENT (<modelo>)-------------------------------------
---START ELEMENT (<cilindrada>)-------------------------------
-----El valor del campo es: 1.5
---END ELEMENT (<cilindrada>)---------------------------------
---END ELEMENT (<coche>)---------------------------------------
---START ELEMENT (<coche>)-----------------------------------
----Atributos de la etiqueta :----id:2
---START ELEMENT (<marca>)-----------------------------------
-----El valor del campo es: Seat
---END ELEMENT (<marca>)-------------------------------------
---START ELEMENT (<modelo>)-----------------------------------
-----El valor del campo es: León
---END ELEMENT (<modelo>)-------------------------------------
---START ELEMENT (<cilindrada>)-------------------------------
-----El valor del campo es: 1.6
---END ELEMENT (<cilindrada>)---------------------------------
---END ELEMENT (<coche>)---------------------------------------
---START ELEMENT (<coche>)-----------------------------------
----Atributos de la etiqueta :----id:3
---START ELEMENT (<marca>)-----------------------------------
-----El valor del campo es: Suzuki
---END ELEMENT (<marca>)-------------------------------------
---START ELEMENT (<modelo>)-----------------------------------
-----El valor del campo es: Vitara
---END ELEMENT (<modelo>)-------------------------------------
---START ELEMENT (<cilindrada>)-------------------------------
-----El valor del campo es: 1.9
---END ELEMENT (<cilindrada>)---------------------------------
---END ELEMENT (<coche>)---------------------------------------
---END ELEMENT (<concesionario>)-------------------------------
Final del documento XML
**********************************************************************
```

2. PRACTICA XML CON SAX

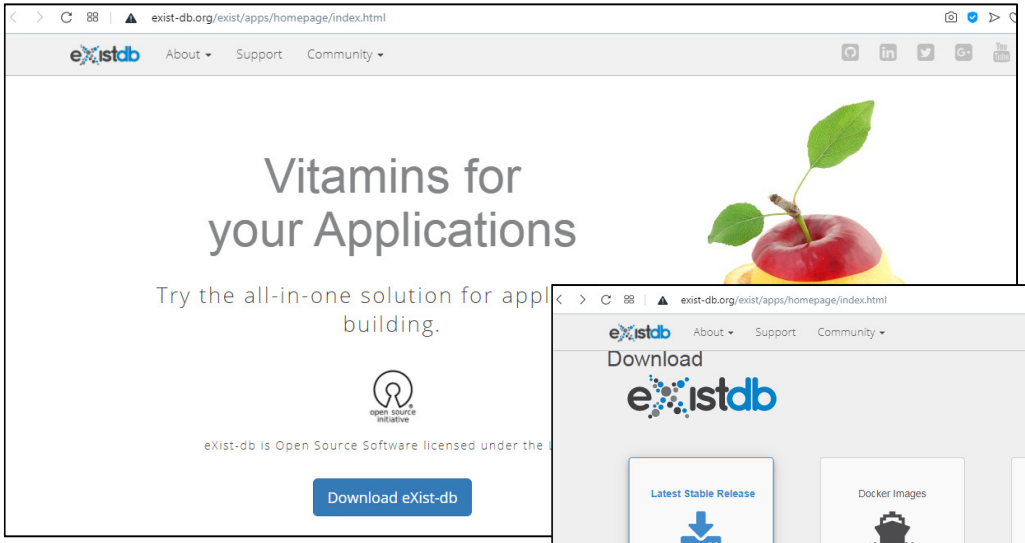
1. Crea un programa en java que lea el documento personas.xml y discos.xml

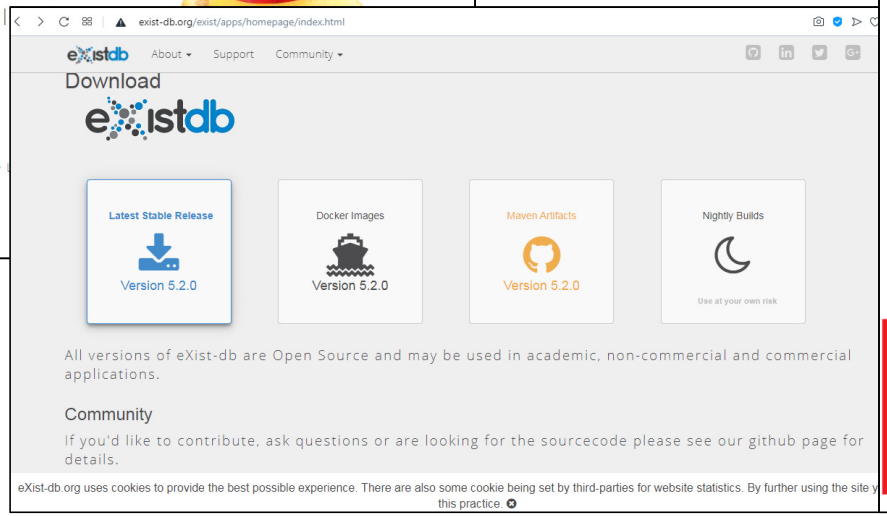
```
<?xml version="1.0" encoding="UTF-8"?>
- <Personas>
  - <persona>
      <nombre>Pepito</nombre>
      <edad>15</edad>
    </persona>
  - <persona>
      <nombre>Juanita</nombre>
      <edad>18</edad>
    </persona>
</Personas>
```

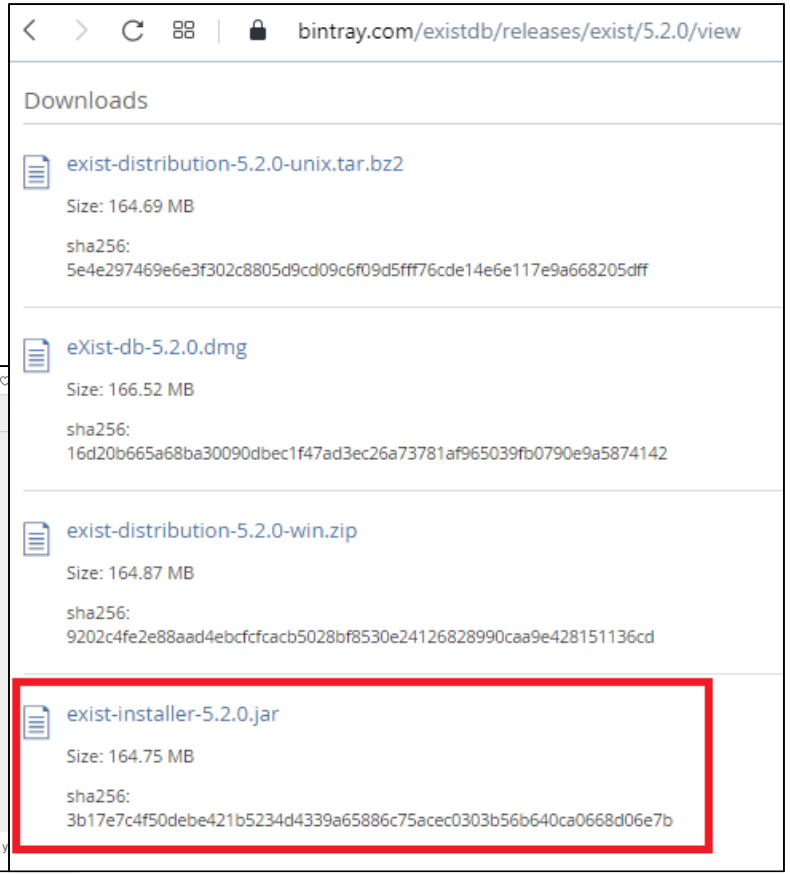
```
<?xml version="1.0" encoding="UTF-8"?>
- <CATALOG>
  - <CD>
      <TITLE>Empire Burlesque</TITLE>
      <ARTIST>Bob Dylan</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Columbia</COMPANY>
      <PRICE>10.90</PRICE>
      <YEAR>1985</YEAR>
    </CD>
  - <CD>
      <TITLE>Hide your heart</TITLE>
      <ARTIST>Bonnie Tyler</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>CBS Records</COMPANY>
      <PRICE>9.90</PRICE>
      <YEAR>1988</YEAR>
    </CD>
  - <CD>
      <TITLE>Greatest Hits</TITLE>
      <ARTIST>Dolly Parton</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>RCA</COMPANY>
      <PRICE>9.90</PRICE>
      <YEAR>1982</YEAR>
    </CD>
  - <CD>
      <TITLE>Still got the blues</TITLE>
      <ARTIST>Gary Moore</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>Virgin records</COMPANY>
      <PRICE>10.20</PRICE>
      <YEAR>1990</YEAR>
    </CD>
  - <CD>
      <TITLE>Eros</TITLE>
      <ARTIST>Eros Ramazzotti</ARTIST>
      <COUNTRY>EU</COUNTRY>
      <COMPANY>BMG</COMPANY>
      <PRICE>9.90</PRICE>
```

3. INSTALACION DE EXIST-DB

Paso 1. Teniendo instalado el jdk de java, debemos ir a la pagina web http://exist-db.org/exist/apps/homepage/index.html y descargar la ultima versión disponible de la base de datos exist-db.

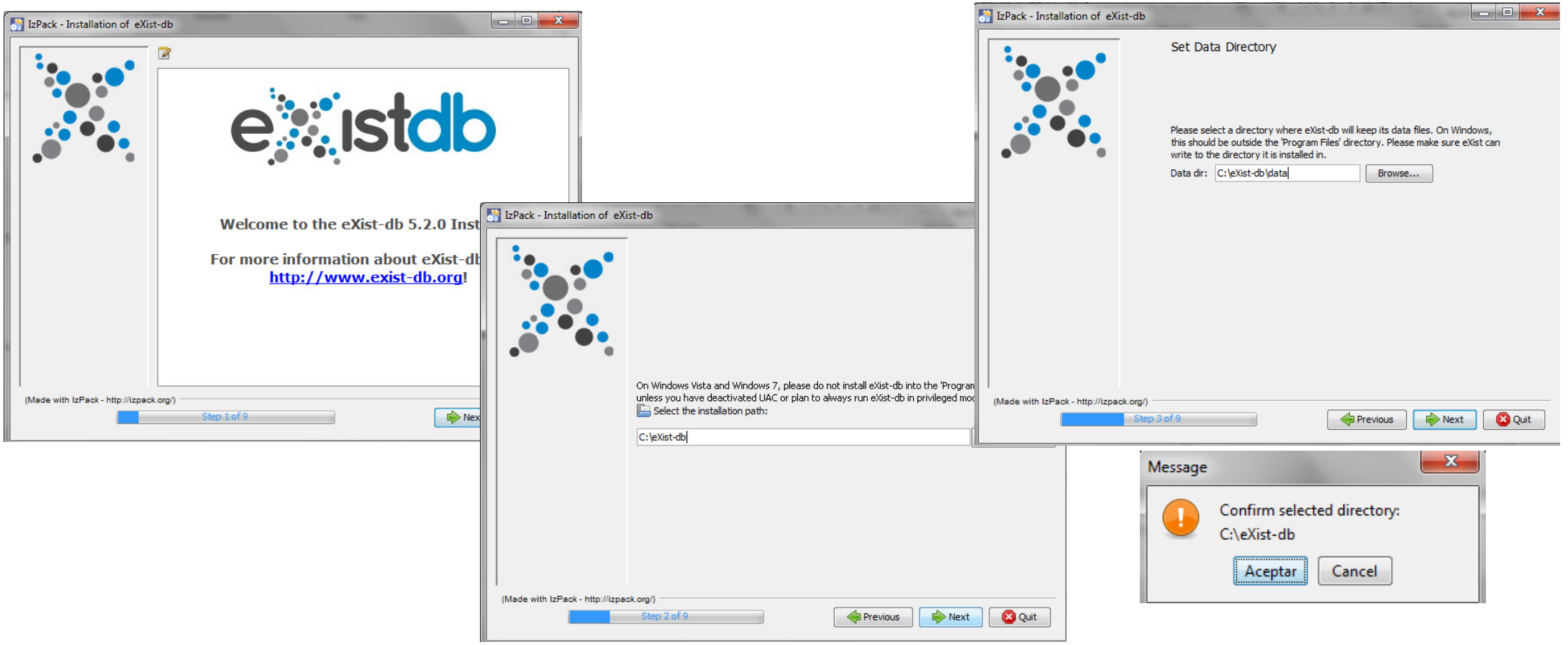






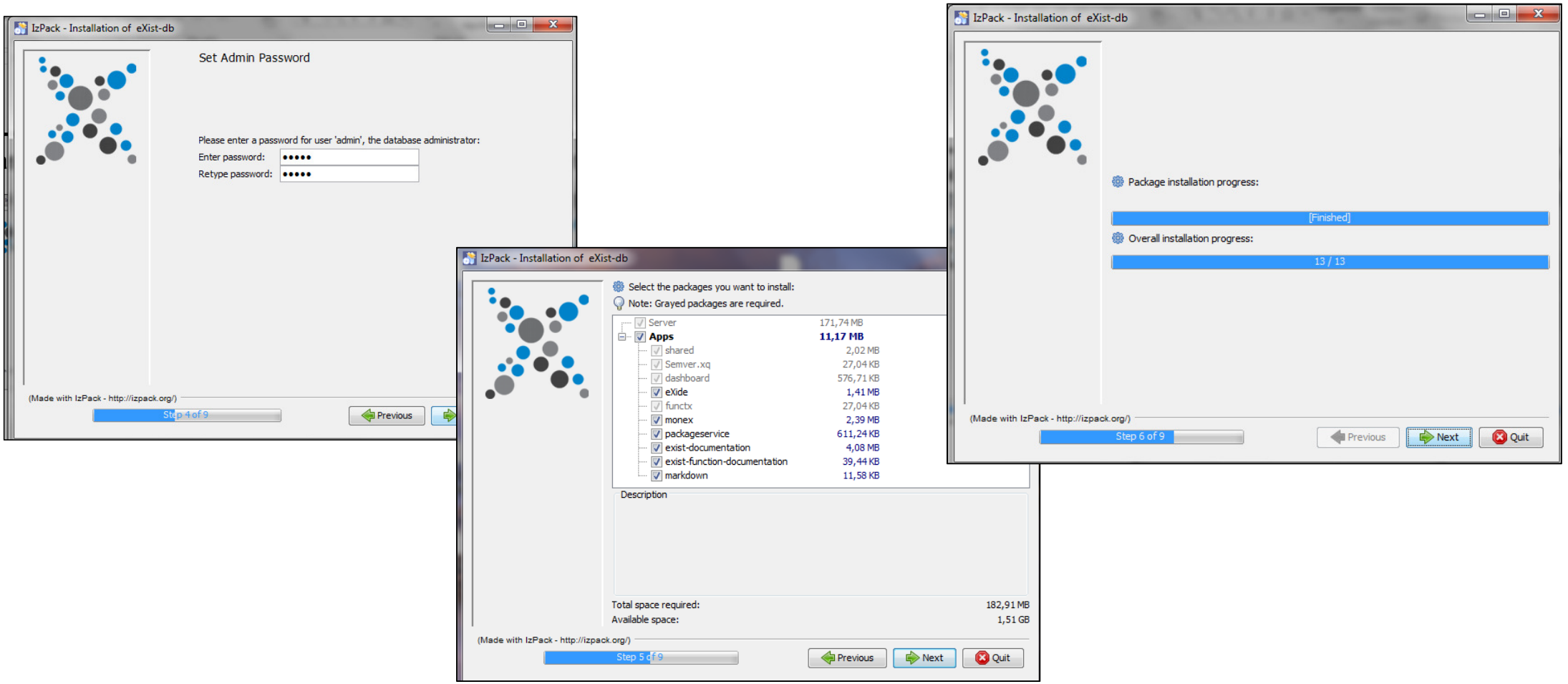
3. INSTALACION DE EXIST-DB

Paso 2. Hacemos doble click en el archivo exist-installer-5.2.0.jar y se inicia la instalación. Indicamos los directorios de instalación:



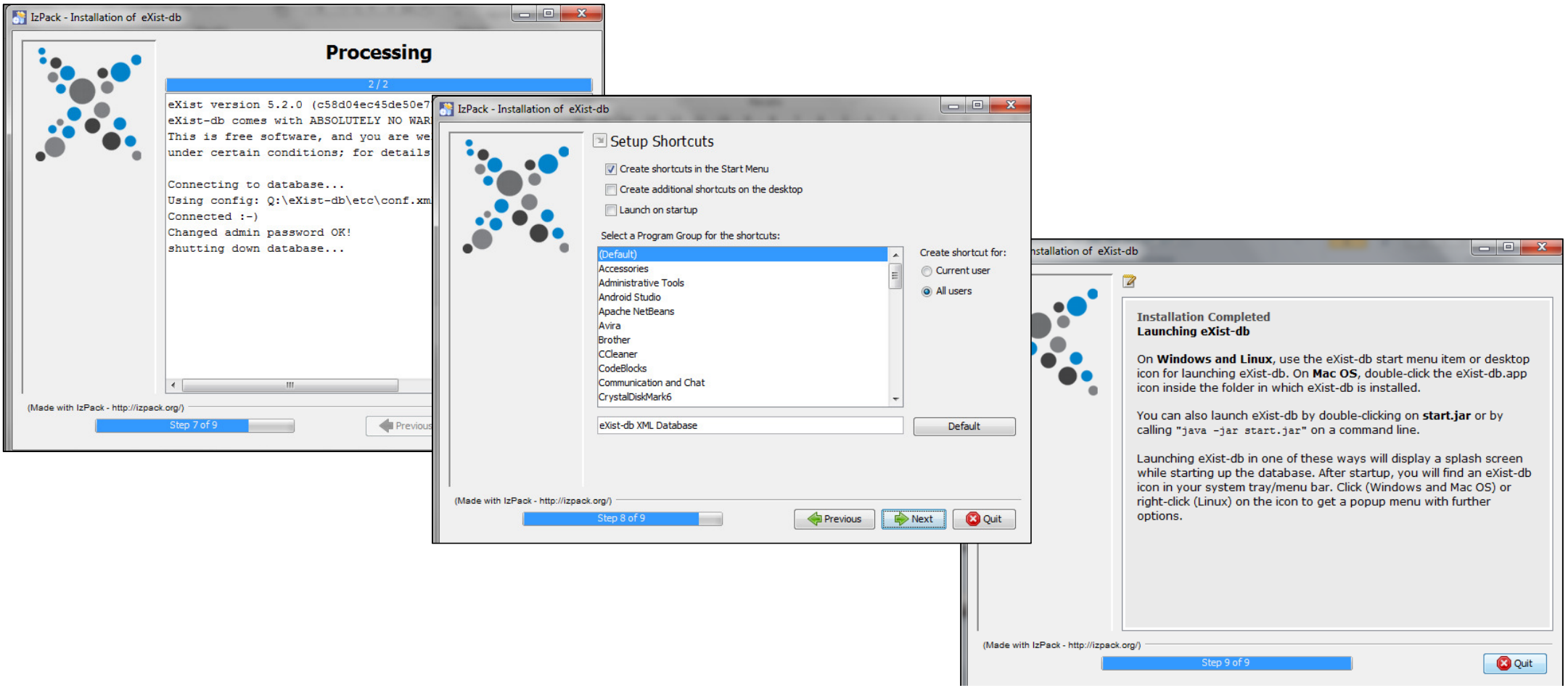
3. INSTALACION DE EXIST-DB

Paso 3. Poner el password “admin” para el usuario administrador “admin”:



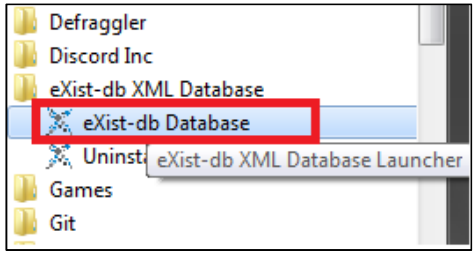
3. INSTALACION DE EXIST-DB

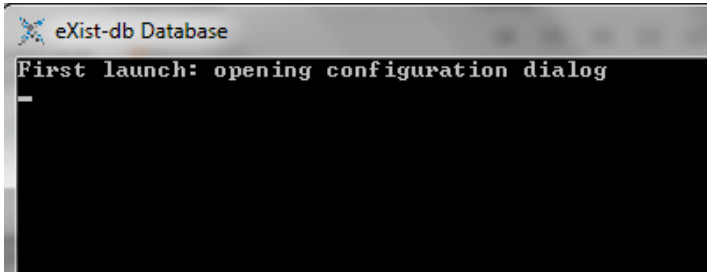
Paso 4. Finalizamos la instalación de exist-db:

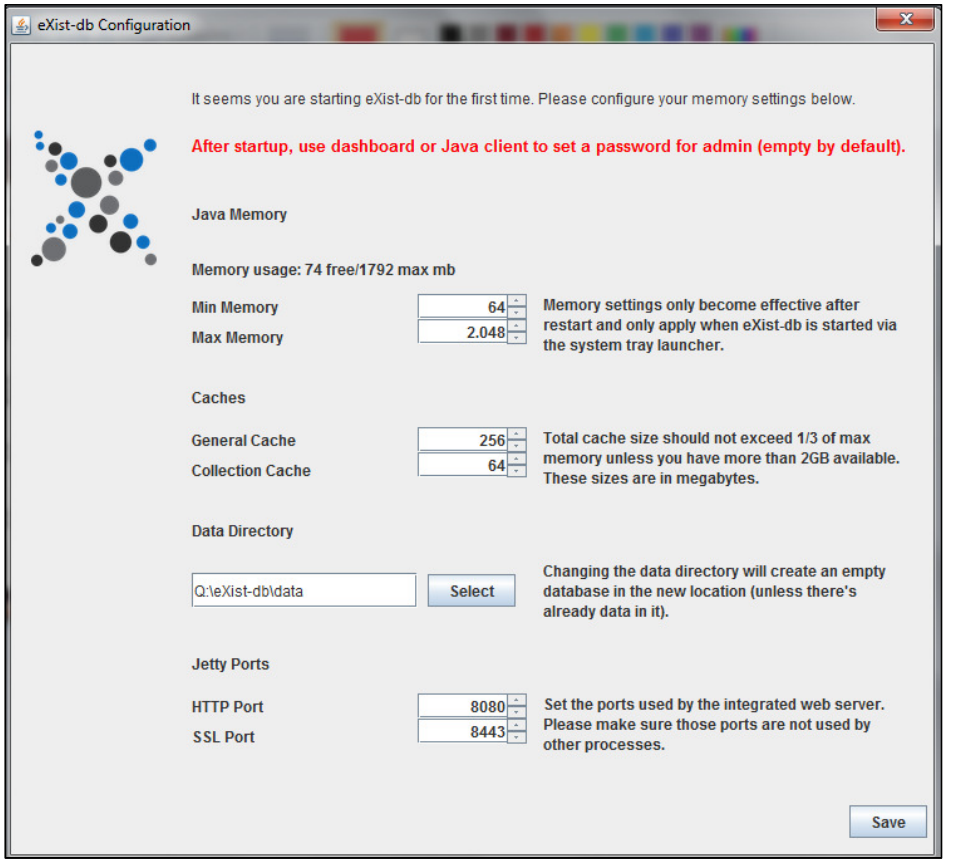


3. INSTALACION DE EXIST-DB

Paso 5. Buscamos el programa exist-db Database para iniciar el servicio por 1º vez. La base de datos exist-db por defecto utiliza el puerto 8080 igual que Tomcat. Hacemos click en Save:

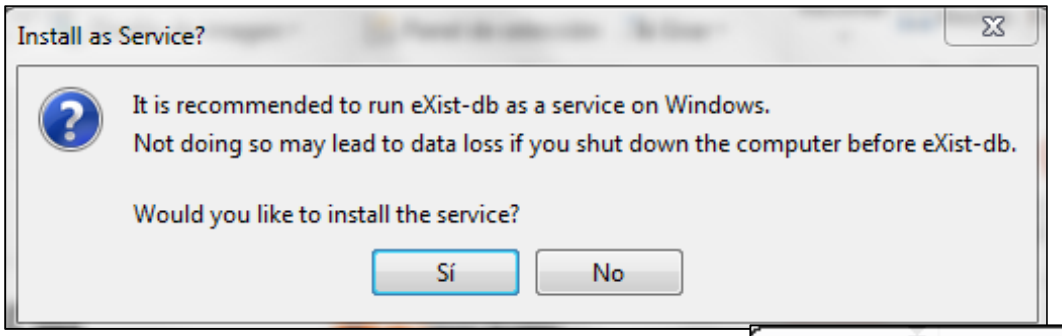


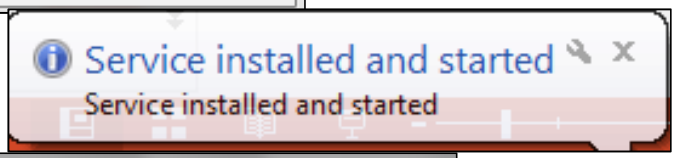


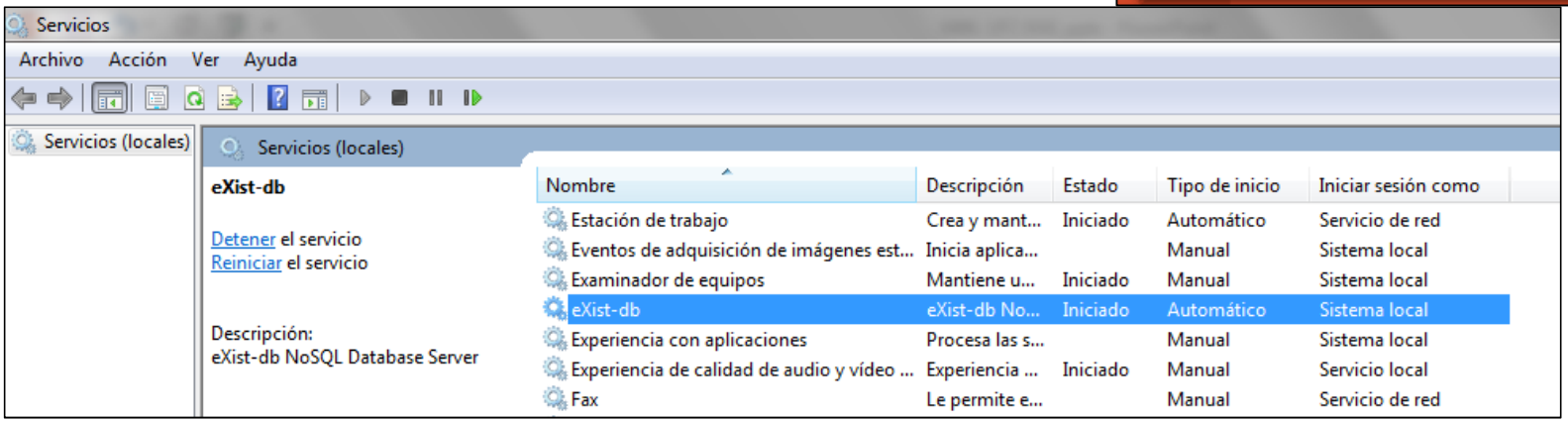


3. INSTALACION DE EXIST-DB

Paso 6. Finalmente instala los programas de gestión de exist-db y nos recomienda que ejecutemos existdb como un servicio de Windows. Podemos observar el servicio activado en services.msc:

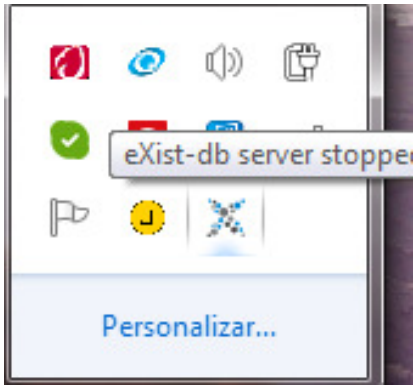


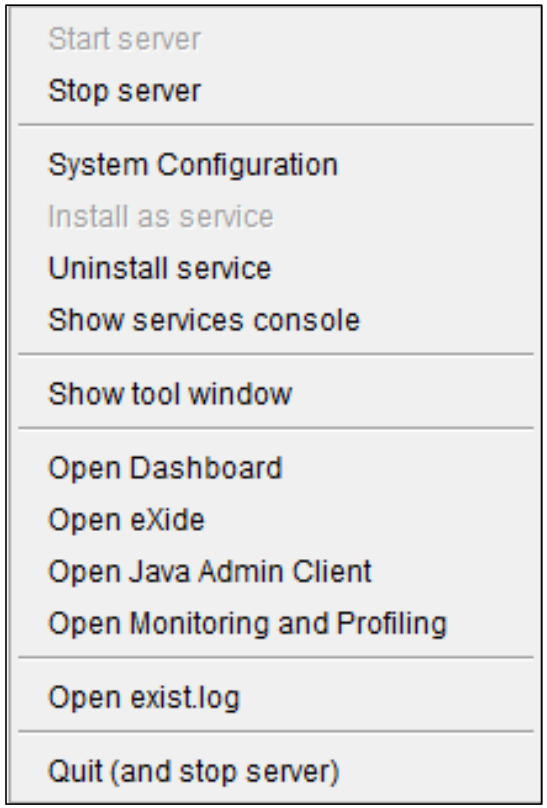




3. INSTALACION DE EXIST-DB

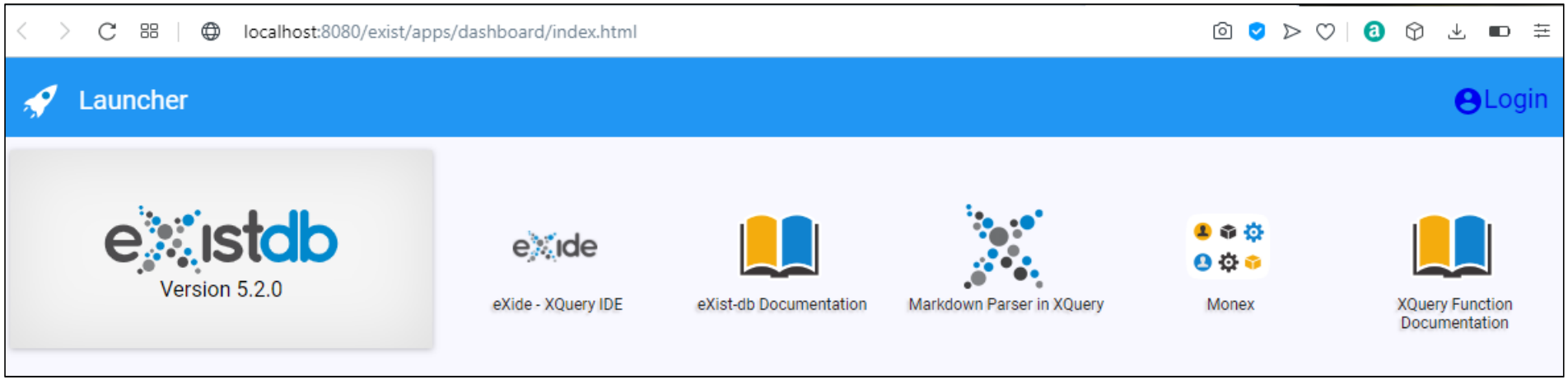
Paso 7. En el área de notificaciones de Windows podemos ver el icono de exist-db. Si hacemos click observamos los diferentes programas que contiene:



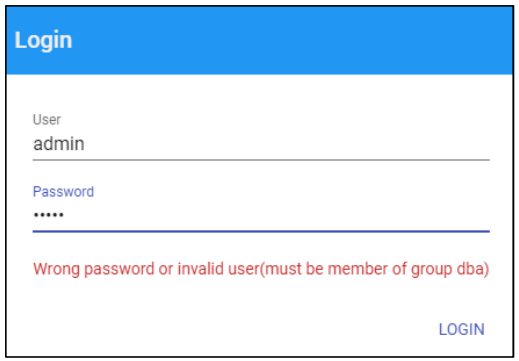


3. INSTALACION DE EXIST-DB

Paso 8. Una vez esta instalado y en marcha el servicio, podemos acceder al panel de control de existdb, escribiendo en un navegador localhost:8080 o a “Open Dashboard” desde el icono de notificaciones. Hacemos click en Login:

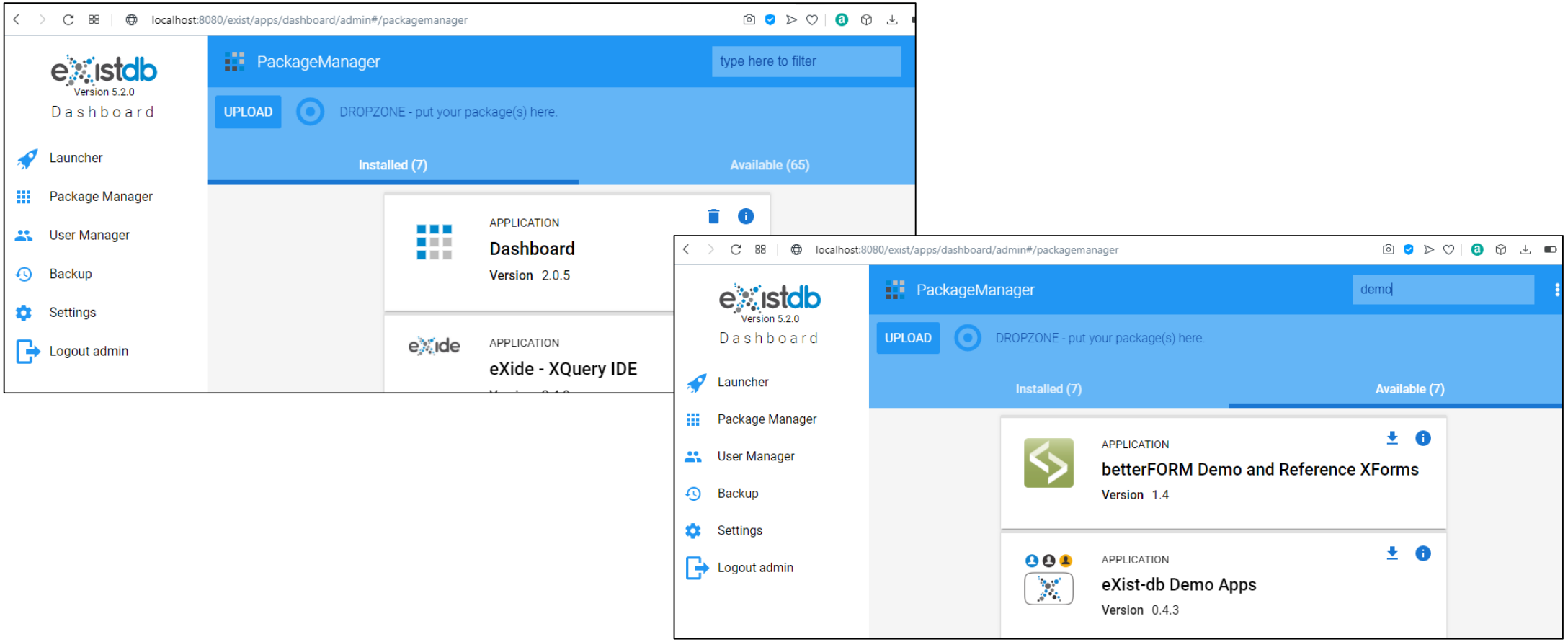


Ponemos usuario: admin
Password: admin



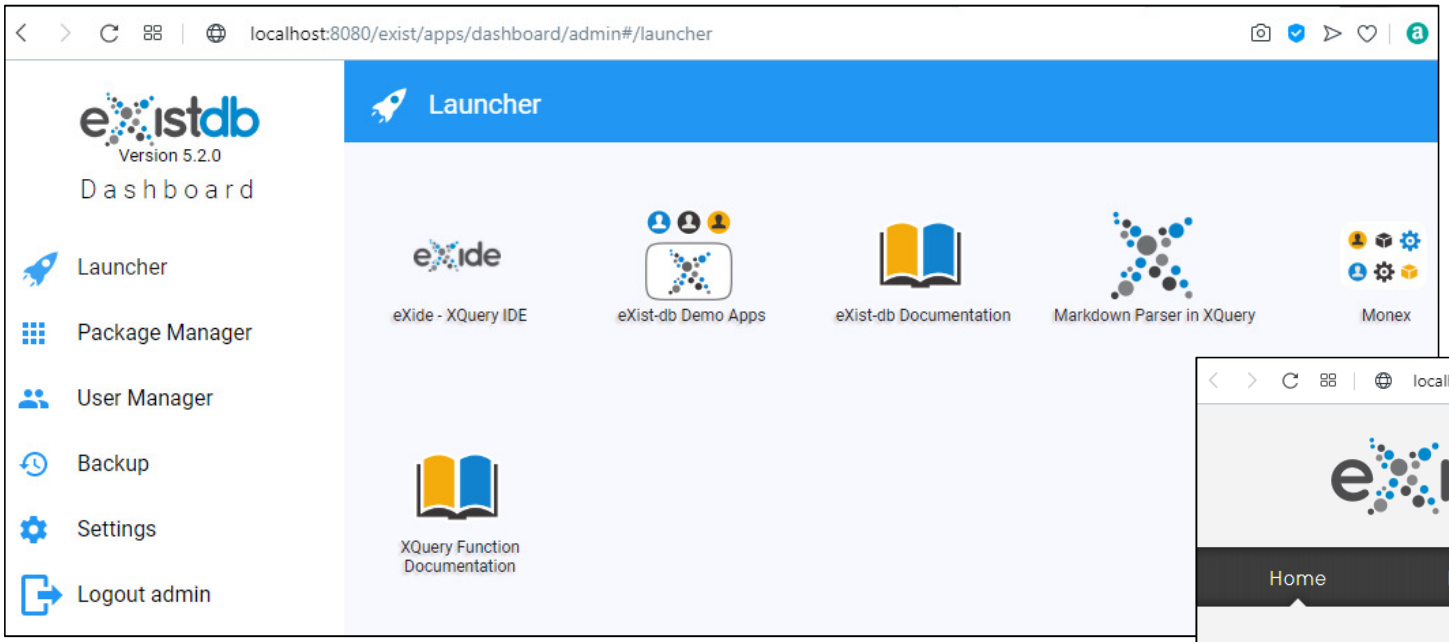
3. INSTALACION DE EXIST-DB

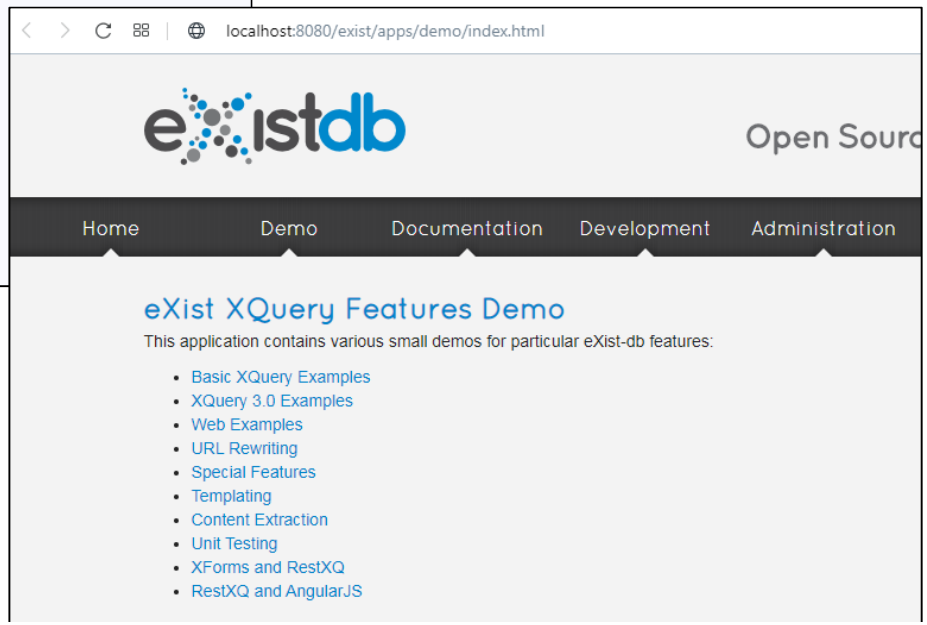
Paso 9. Dentro de la zona del administrador nos dirigimos al Package Manager e instalamos la utilidad Demo Apps:



3. INSTALACION DE EXIST-DB

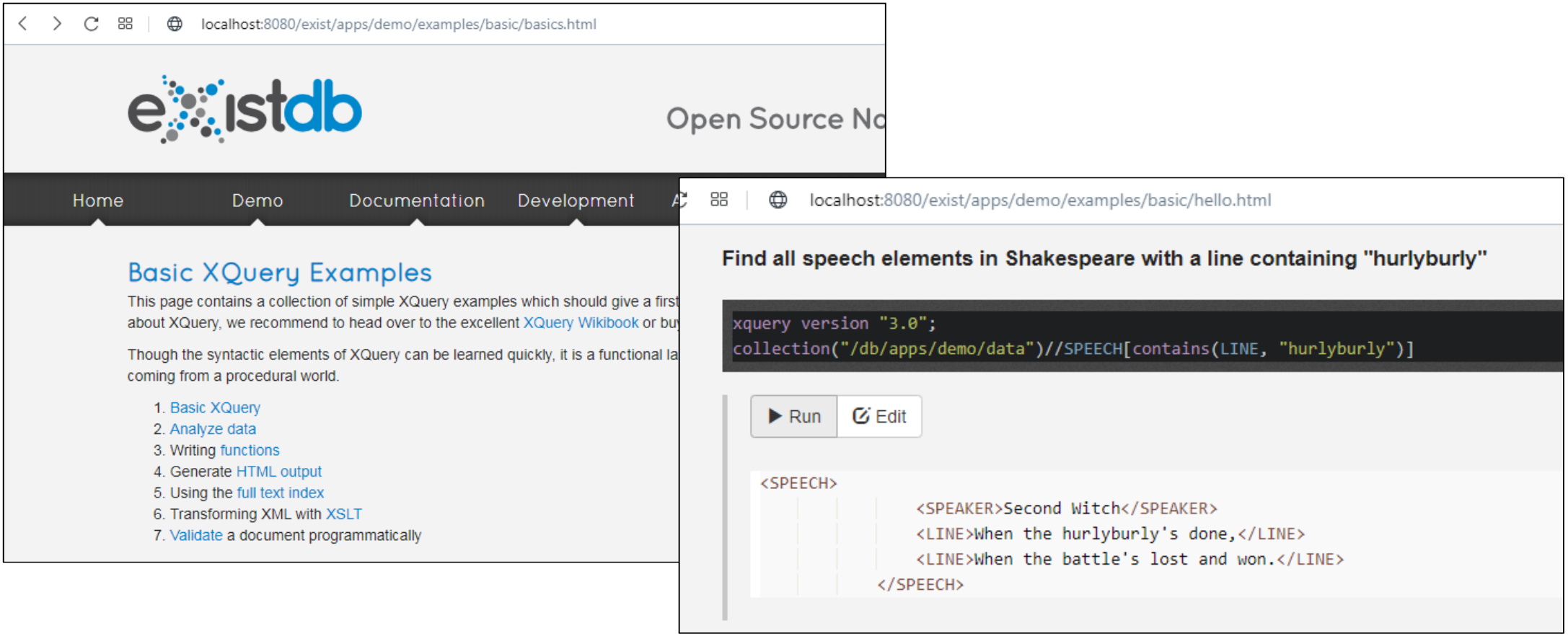
Paso 10. Vamos al Launcher (pagina principal) y hacemos click en Demo Apps:





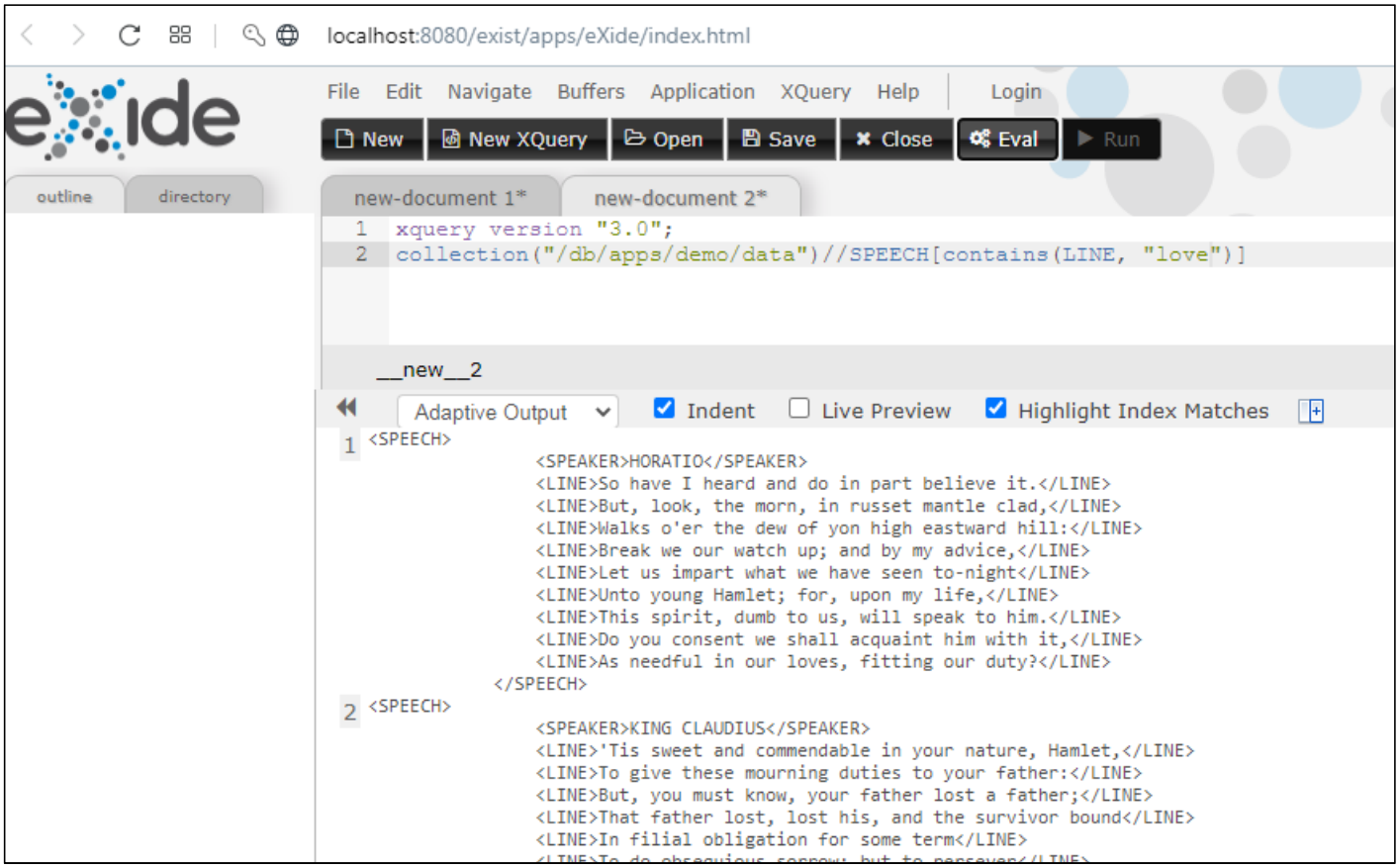
3. INSTALACION DE EXIST-DB

Paso 11. Vamos a Basic XQuery Examples y a Basic XQuery. Escogemos un ejemplo y lo ejecutamos (hacemos click en el botón run):



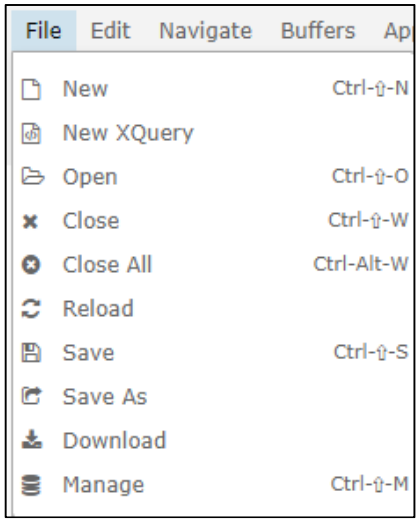
3. INSTALACION DE EXIST-DB

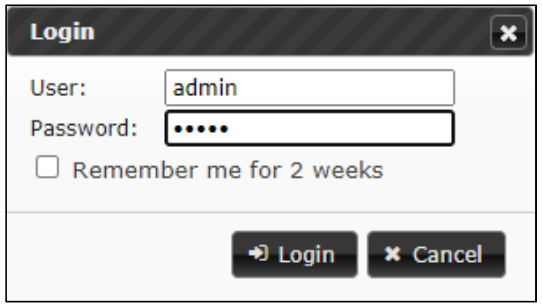
Paso 12. Ahora le damos al boton edit y nos lleva esta consulta a la herramienta eXide. Podemos buscar la palabra “love”, haciendo click en el botón “Eval”:

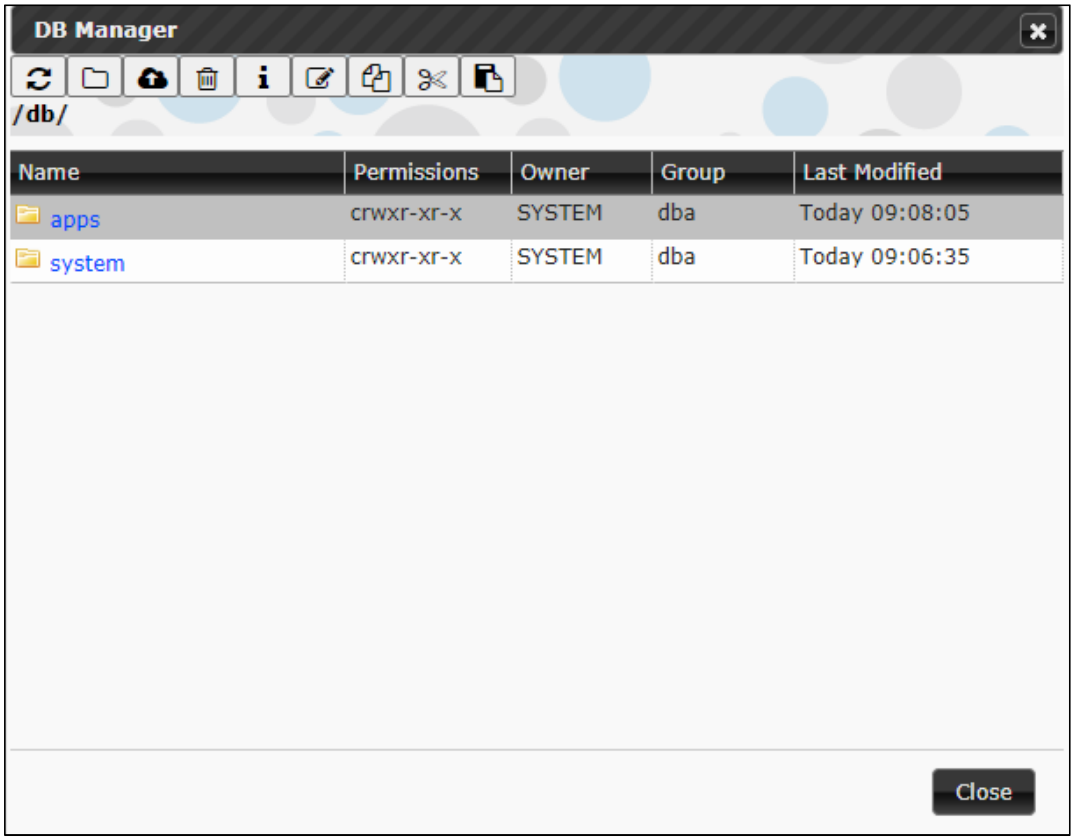


3. INSTALACION DE EXIST-DB

Paso 13. Vamos a crear una base de datos propia. Vamos a File/Manage. Nos logueamos con admin y llegamos al DB Manager:

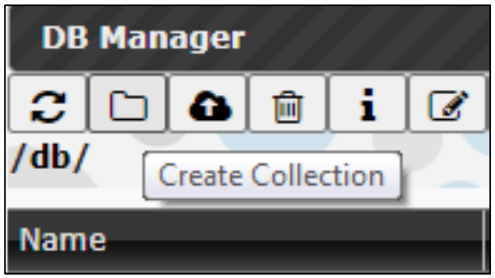


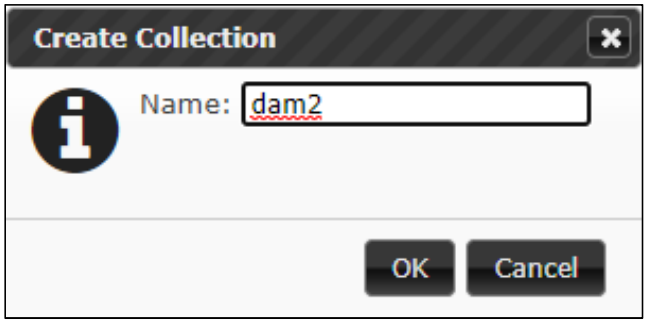


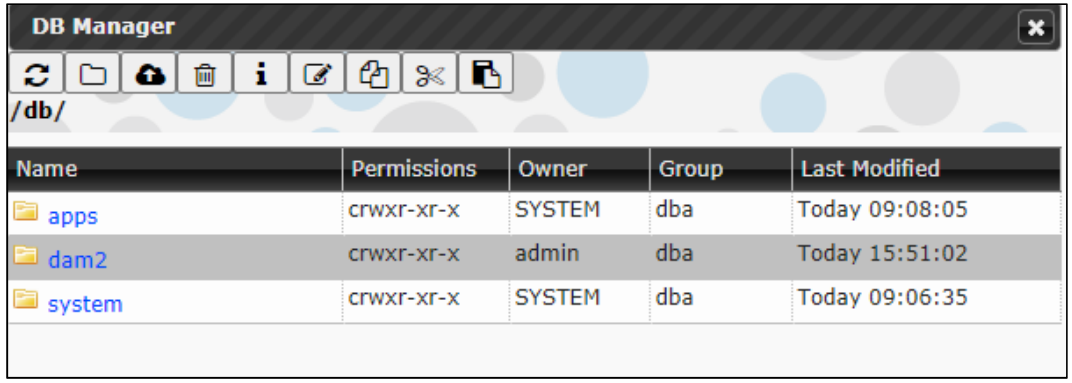


3. INSTALACION DE EXIST-DB

Paso 14. Vamos a crear una colección de nombre dam2:

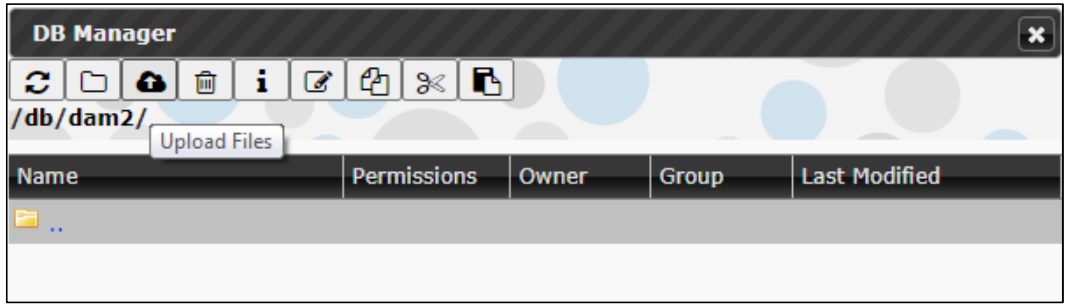


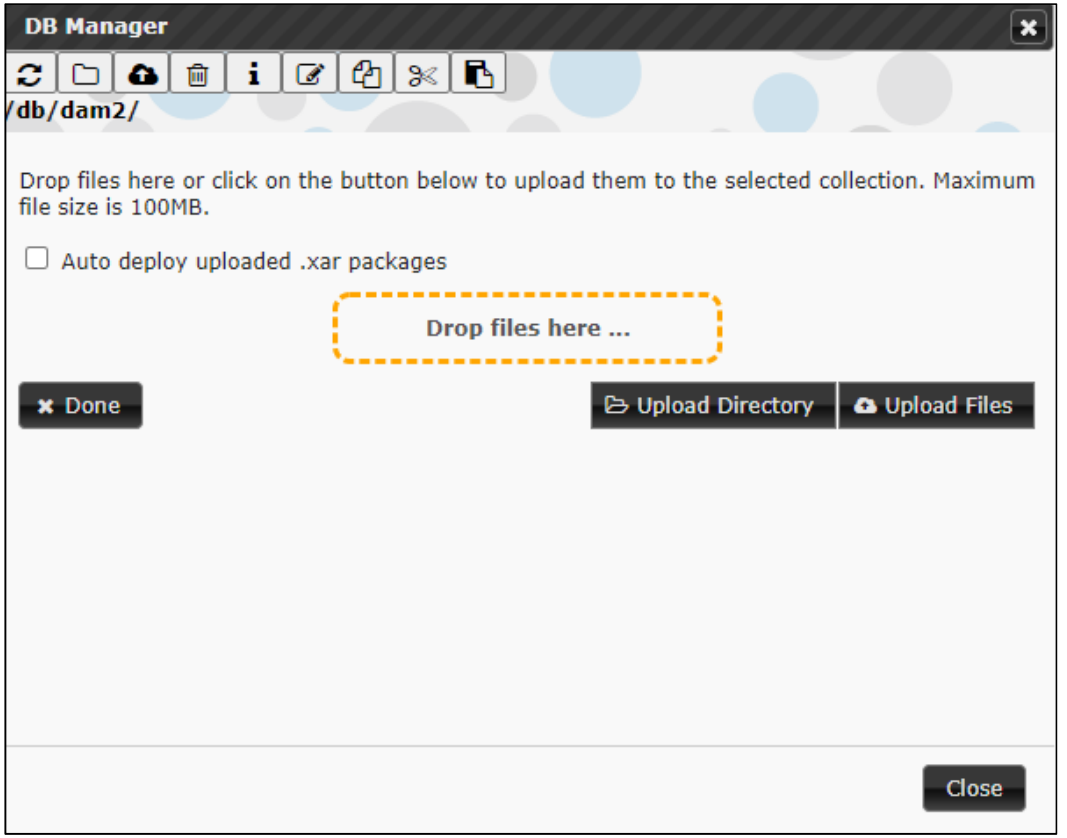




3. INSTALACION DE EXIST-DB

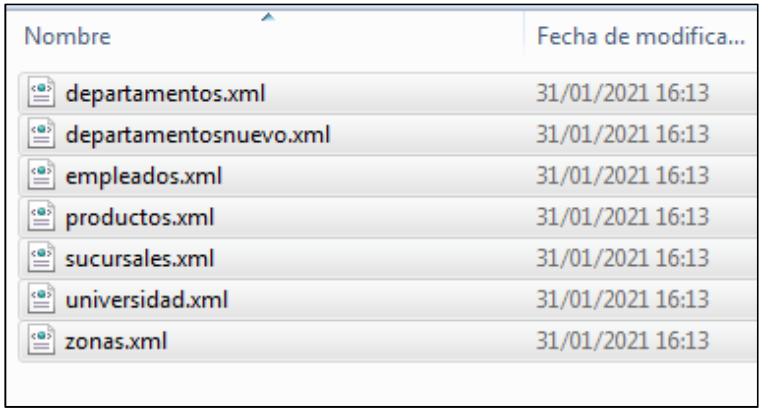
Paso 15. Entramos en la colección dam2 (haciendo doble click) y una dentro de /db/dam2/ hacemos click en el botón de subida de ficheros:

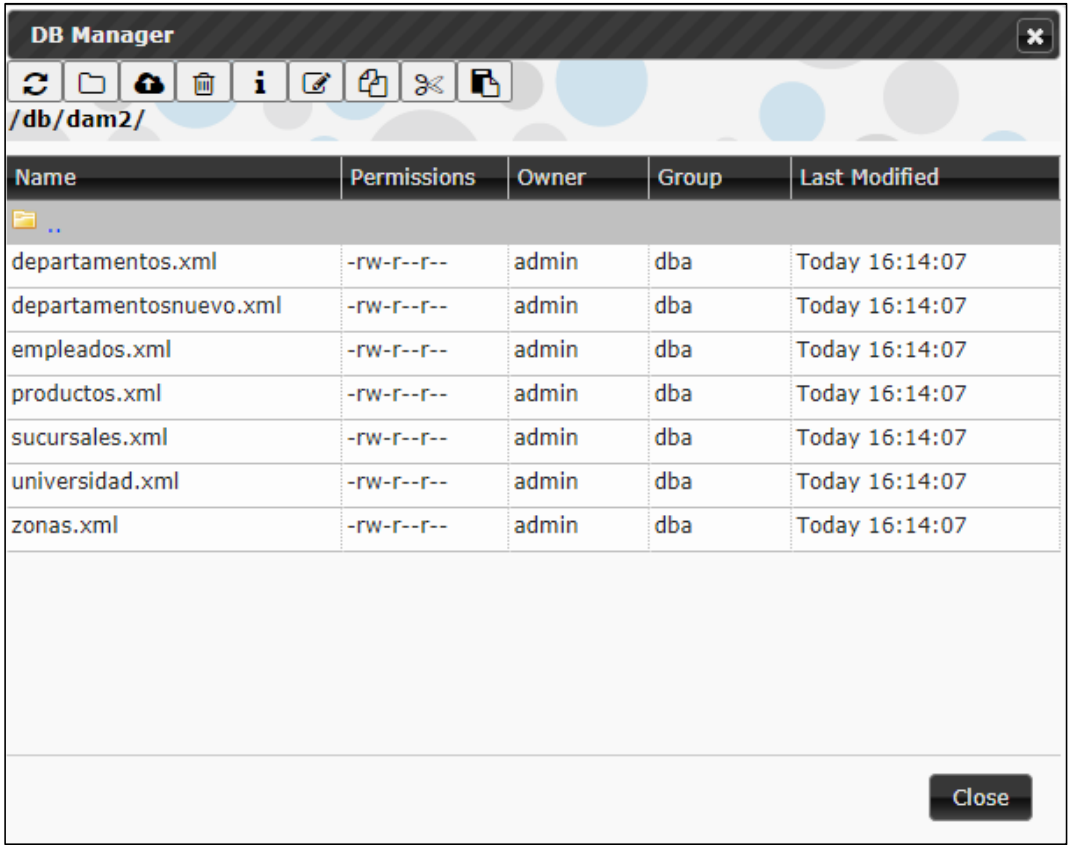




3. INSTALACION DE EXIST-DB

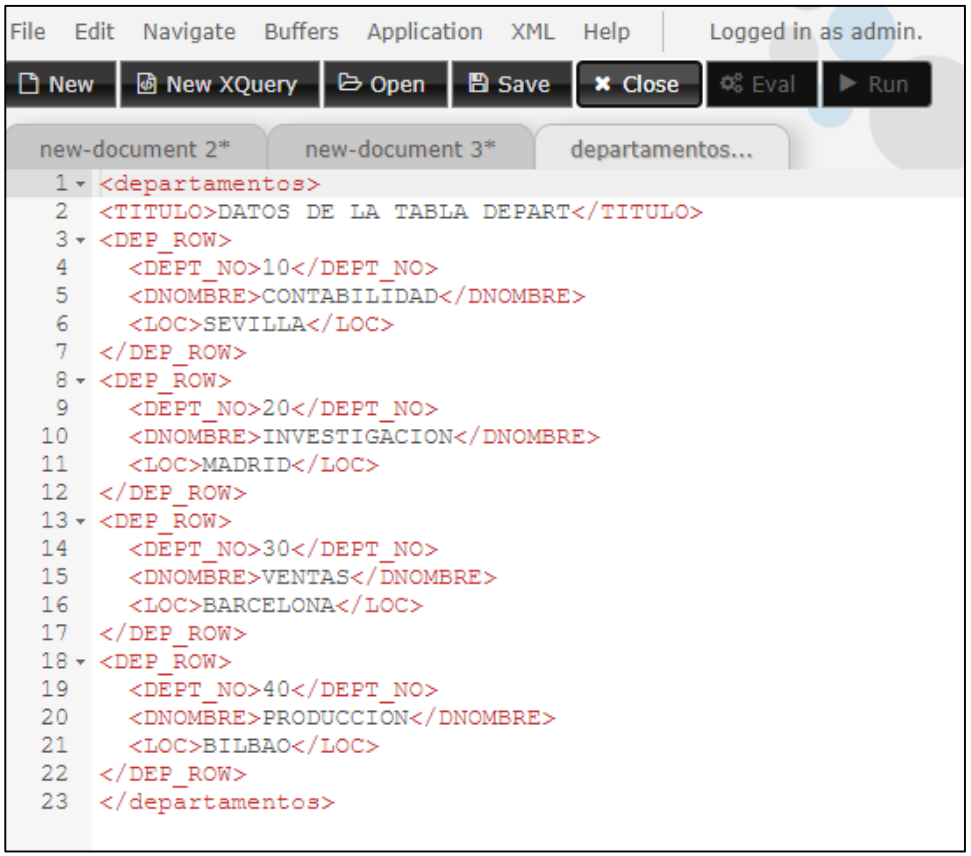
Paso 16. Subimos a la colección dam2 todos los documentos xml contenidos en el archivo “ColecciónPruebas.zip”





3. INSTALACION DE EXIST-DB

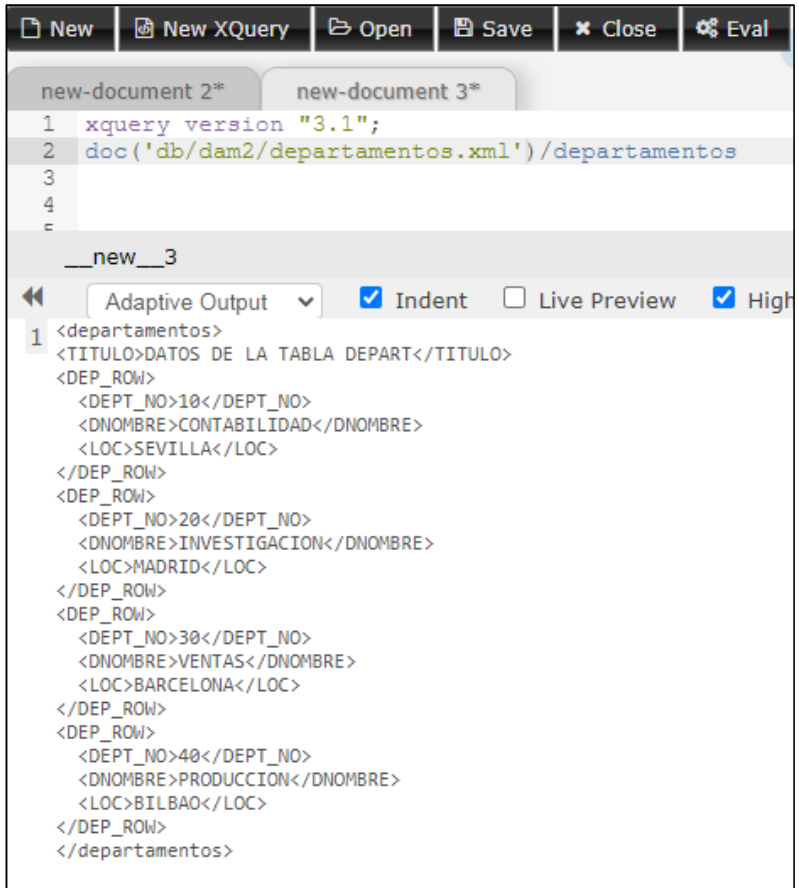
Paso 17. Se puede visualizar cualquier de los documentos subidos haciendo doble click encima de ellos.



3. INSTALACION DE EXIST-DB

Paso 18. Crea en la pestaña new XQuery una primera consulta:

doc('db/dam2/departamentos.xml')/departamentos



4. XPATH Y XQUERY

- Tanto XPath com XQuery son estándares para acceder y obtener datos desde documentos XML.
- Estos lenguajes tienen en cuenta que la información en los documentos está semiestructurada o jerarquizada como árbol.
- **XPath** → Lenguaje de rutas XML, se utiliza para navegar dentro de la estructura jerárquica de un XML
- **XQuery** → es a XML lo mismo que SQL es a las bbdd relacionales, es decir, un lenguaje de consulta diseñado para consultar documentos XML. XQuery contiene a XPath, toda expresión de consulta en XPath es válida en XQuery, pero XQuery permite mucho más

4. CONSULTAS XPATH

Expresiones Xpath

- XPath es un lenguaje que permite seleccionar nodos de un documento XML y calcular valores a partir de su contenido. Existen varias versiones de XPath aprobadas por W3C aunque la versión más utilizada sigue siendo la 1.
- La forma en que XPath selecciona partes del documento XML se basa en la representación arbórea que se genera del documento.
- A la hora de recorrer un árbol XML podemos encontrarnos con los siguientes tipos de nodos:
  - **nodo raíz** → raíz del árbol, se representa por /
  - **nodos elemento** → cualquier elemento del árbol, son las etiquetas del árbol
  - **nodos texto** → los caracteres entre etiquetas
  - **nodos atributo** → propiedades añadidas a los nodos elementos, se representan con @
  - **nodos comentario** → etiquetas de comentario
  - **nodos espacio de nombres** → contienen espacio de nombres
  - **nodos instrucción de proceso** → instrucciones de proceso, van entre <?.....?>

4. CONSULTAS XPATH

```
<?xml version =”1.0” encoding = “ISO-8859-1”?>
<universidad>
<espacio xmlns=http://www.misitio.com xmlns:prueba=”http://www.misitio.com/pruebas” />
<!-- DEPARTAMENTO -- >
<departamento telefono=”112233” tipo=”A”>
        <codigo>IFC1</codigo>
        <nombre>Informática</nombre>
</departamento>
….
</universidad>
```

- Elementos: <universidad>, <departamento>,<codigo>,<nombre>
- Texto: IFC1, Informática
- Atributo: telefono=”112233”, tipo=”A”
- Comentario: DEPARTAMENTO

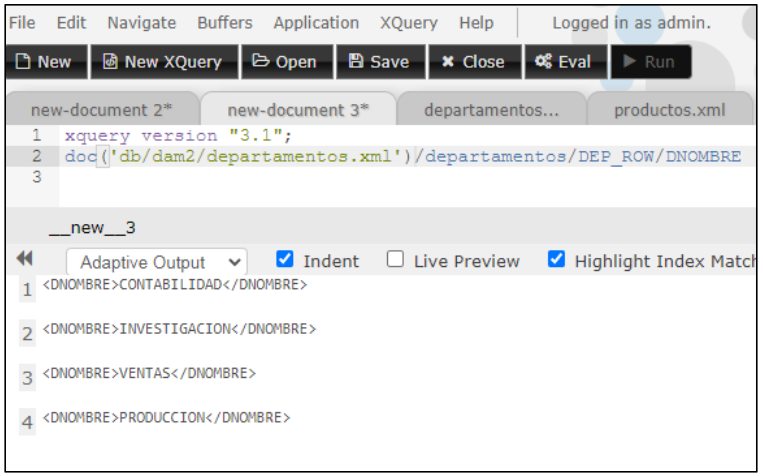
4. CONSULTAS XPATH

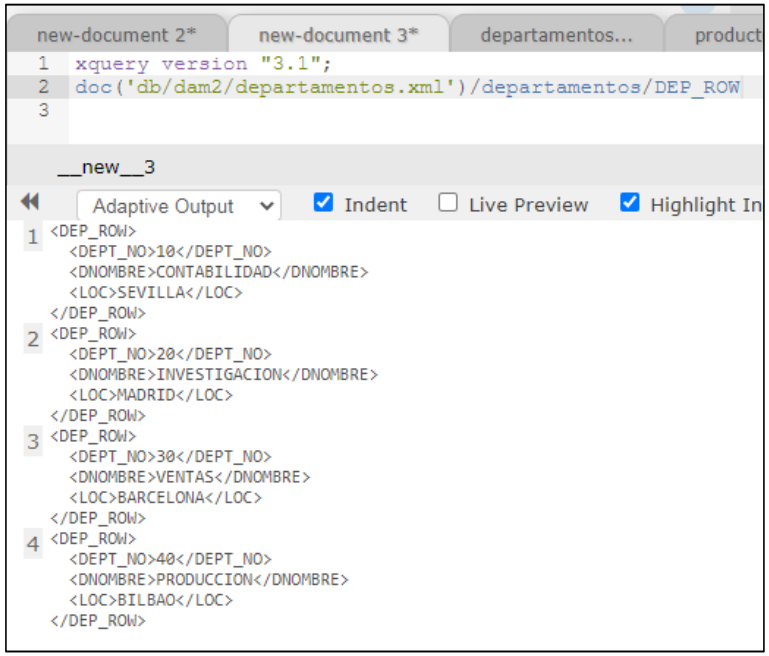
Ejercicio 1. Comprueba el resultado de las siguientes consultas:

a. /departamentos → devuelve todos los datos de departamentos (Esta sería la misma consulta del ejercicio anterior pero sin indicar toda la ruta)

b. /departamentos/DEP_ROW → devuelve las etiquetas de cada DEP_ROW

c. /departamentos/DEP_ROW/DNOMBRE → devuelve nombres de departamentos entre etiquetas





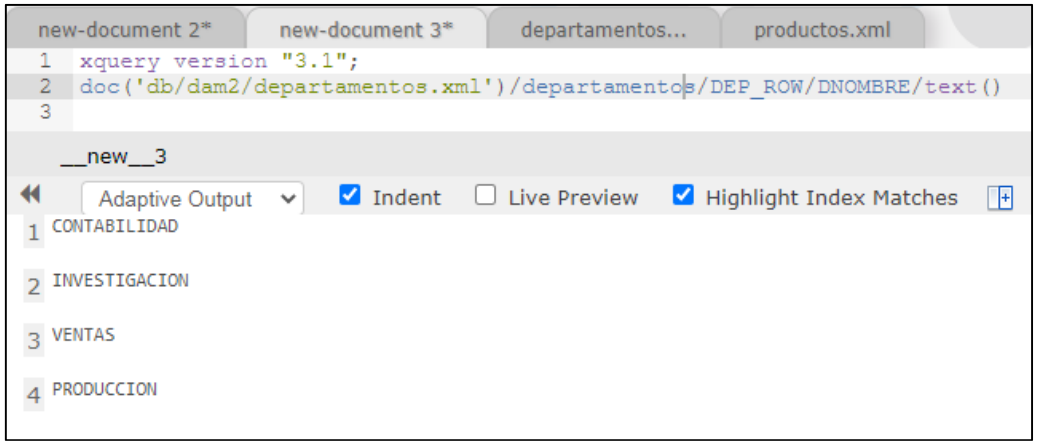
4. CONSULTAS XPATH

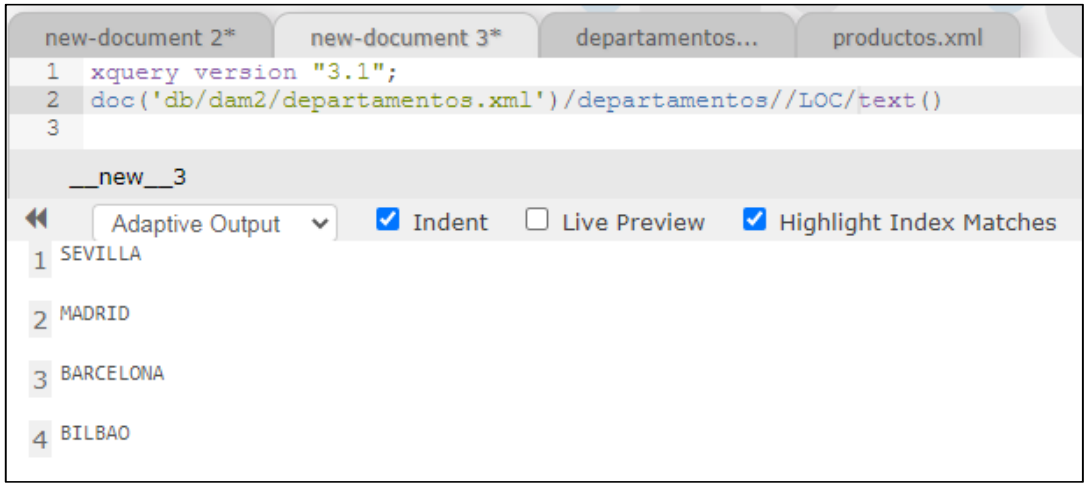
d. /departamentos/DEP_ROW/DNOMBRE/text() → Lo mismo que antes pero sin etiquetas

e. //LOC/text() → localidades

NOTA: / se usa para dar rutas absolutas.

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





4. CONSULTAS XPATH

Ejercicio 2. Averigua el resultado de las siguientes consultas (utilizaremos el documento 'db/dam2/empleados.xml')

a. /EMPLEADOS/EMP_ROW[DEPT_NO=10]
b. /EMPLEADOS/EMP_ROW/APELLIDO|/EMPLEADOS/EMP_ROW/DEPT_NO
c. /EMPLEADOS/EMP_ROW [DEPT_NO=10]/APELLIDO/text()
d. /EMPLEADOS/EMP_ROW [not(OFICIO='ANALISTA')]
e. /EMPLEADOS/EMP_ROW[SALARIO>1300 and DEPT_NO=20]/APELLIDO
f. /EMPLEADOS/EMP_ROW[1]

4. CONSULTAS XPATH

Ejercicio 3. Investiga en la web las siguientes funciones de XPath y pon algún ejemplo utilizando los documentos departamentos.xml y empleados.xml

a. last()
b. position()
c. count()
d. sum(),div(),mod()
e. max(), min(),avg()
f. concat(cadena1, cadena2,...)
g. starts-with (cadena1, cadena2)
h. contains(cad1,cad2)
i. string-length(argumento)

4. CONSULTAS XPATH

Ejercicio 4. Resuelve las siguientes consultas:

a. Devuelve el apellido del penúltimo empleado (NOTA: utilizar last())
b. Obtén los elementos del empleado que ocupa la posición 3 (position())
c. Cuenta el número de empleados del departamento 10
d. Obtén la suma de SALARIO de los empleados del DEPT_NO =20
e. Obtén el salario máximo, el mínimo de los empleados con OFICIO=ANALISTA
f. Obtén la media de salario en el DEPT_NO=10
g. Devuelve la concatenación de apellido, oficio y salario
h. Obtén los elementos de los empleados cuyo apellido empieza por 'A'
i. Devuelve los oficios que contienen la sílaba 'OR'
j. Obtén los datos de los empleados cuyo apellido tiene menos de 4 caracteres

4. CONSULTAS XPATH

Ejercicio 5. Resuelve las siguientes consultas referentes al documento productos.xml. Este documento contiene los datos de los productos de una distribuidora de componentes informáticos.

a) Obtén la denominación y precio de todos los productos
b) Obtén los productos que sean “Placa base”
c) Obtén los productos cuyo precio sea mayor que 60€ y de la zona 20
d) Obtén el número de los productos que sean memorias y de la zona 10
e) Obtén la media de los precios de los micros
f) Obtén los datos de los productos cuyo stock mínimo sea mayor que el stock actual (usa función number())
g) Obtén el producto más caro
h) Obtén el producto más barato de la zona 20

5. CONSULTAS XQUERY

- Una consulta XQuery es una expresión que lee datos de uno o más documentos en XML y devuelve como resultado otra secuencia de datos en XML.
- XQuery contiene a XPath. Es decir, toda expresión de consulta en XPath es válida y devuelve el mismo resultado en XQuery.
- XQuery nos va a permitir:
  - Seleccionar información basada en un criterio específico
  - Buscar información en un documento o conjunto de documentos
  - Unir datos desde múltiples documentos
  - Transformar y reestructurar datos XML en otro vocabulario o estructura
  - ...

5. CONSULTAS XQUERY

- En las consultas XQuery podemos utilizar las siguientes funciones para referirnos a colecciones y documentos dentro de la bbdd:
  - **collection("/ruta")** → indicamos el camino para referirnos a una colección
  - **doc("/ruta/documento.xml")** → indicamos el camino de un documento de una colección
- En XQuery las consultas se pueden construir utilizando expresiones FLWOR que corresponde a las siglas de **For, Let, Where, Order y Return**.
- La sintaxis general es:

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

5. CONSULTAS XQUERY

- **For** → se usa para seleccionar nodos y almacenarlos en una variable, similar a la cláusula FROM de SQL. Dentro del for escribimos una expresión XPath que seleccionará a los nodos. Si se especifica más de una variable en el for se actúa como producto cartesiano. Las variables comienzan con $
- Las consultas XQuery deben llevar obligatoriamente una orden **Return**, donde indicaremos lo que queremos que nos devuelva la consulta.
- Ejemplo comparativo XQuery - XPath

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

5. CONSULTAS XQUERY

- **Let** → permite que se asignen valores resultantes de expresiones XPath a variables para simplificar la representación. Se pueden poner varias líneas let una por cada variable, o separar las variables por comas.
- En el siguiente ejemplo, se crean dos variables $nom y $ofi. La salida sale ordenada por OFICIO, y se crea una etiqueta <APE_OFI> que incluye el nombre y oficio concatenado.

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

5. CONSULTAS XQUERY

- **Where** → para filtrar elementos
- **Order by** → ordena los datos según un criterio dado
- **Return** → construye el resultado de la consulta en XML. Se pueden añadir etiquetas XML a la salida. Si lo hacemos, los datos a visualizar los encerramos entre llaves {}. Además, en el return se puede añadir condicionales usando **if-then-else**
- El siguiente ejemplo devuelve los departamentos de tipo A encerrados en una etique

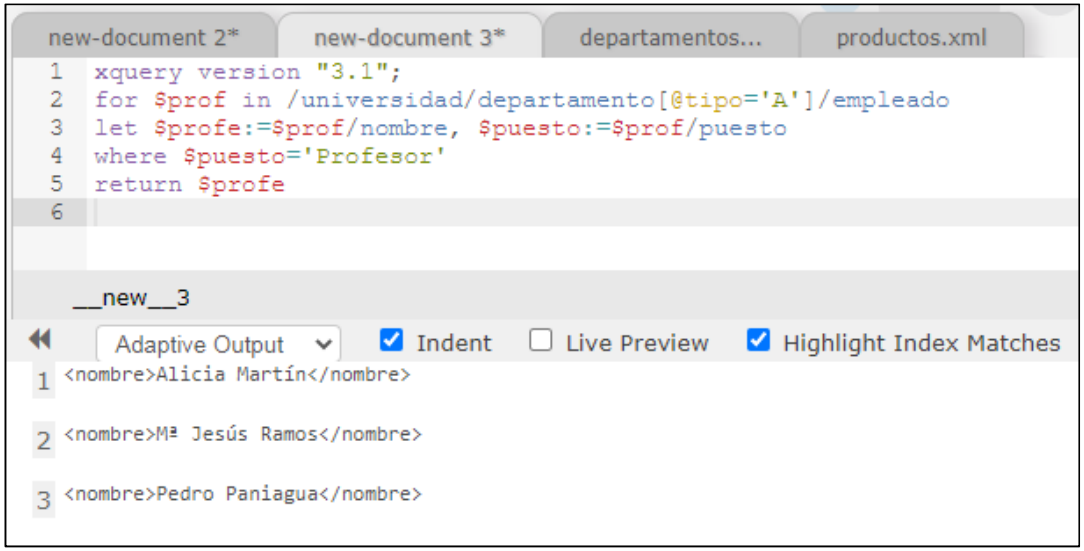
```
for $dep in /universidad/departamento
return if ( $dep/@tipo='A')
        then <tipoA> {data ( $dep/nombre)} </tipoA>
```

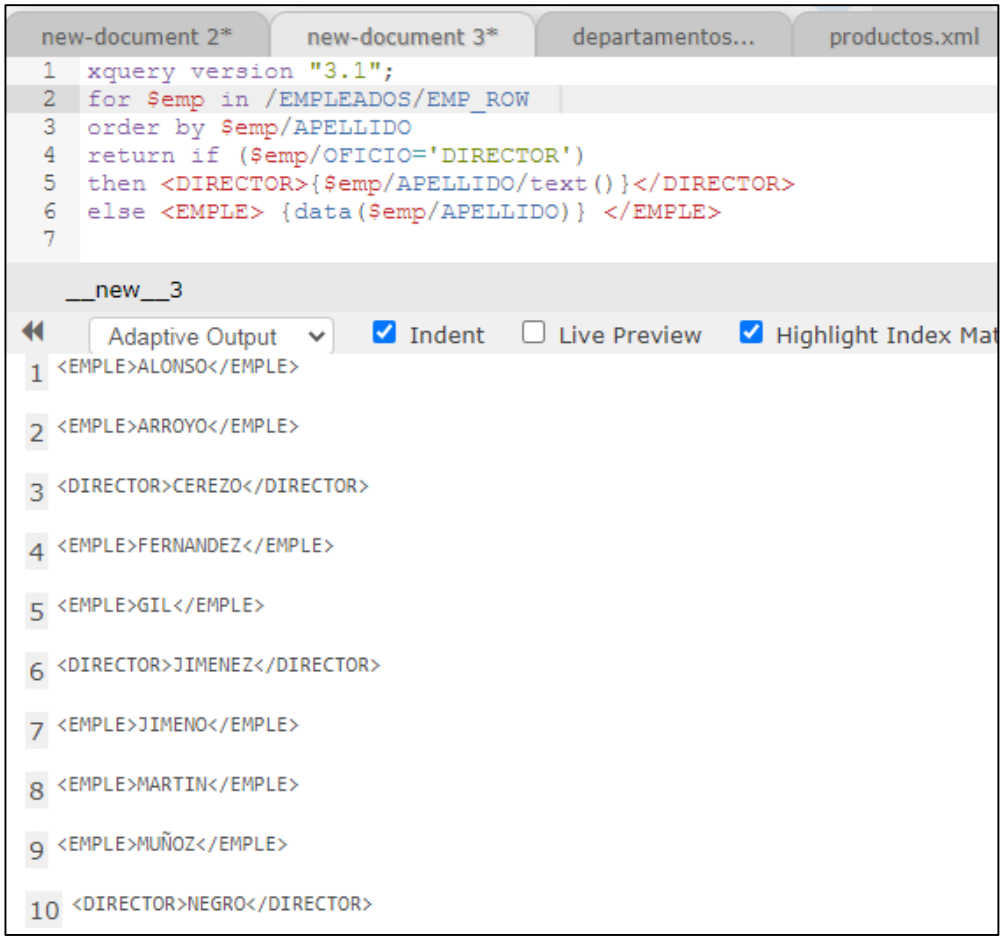
- Utilizaremos la función **data()** para extraer el contenido en texto de los elementos.

5. CONSULTAS XQUERY

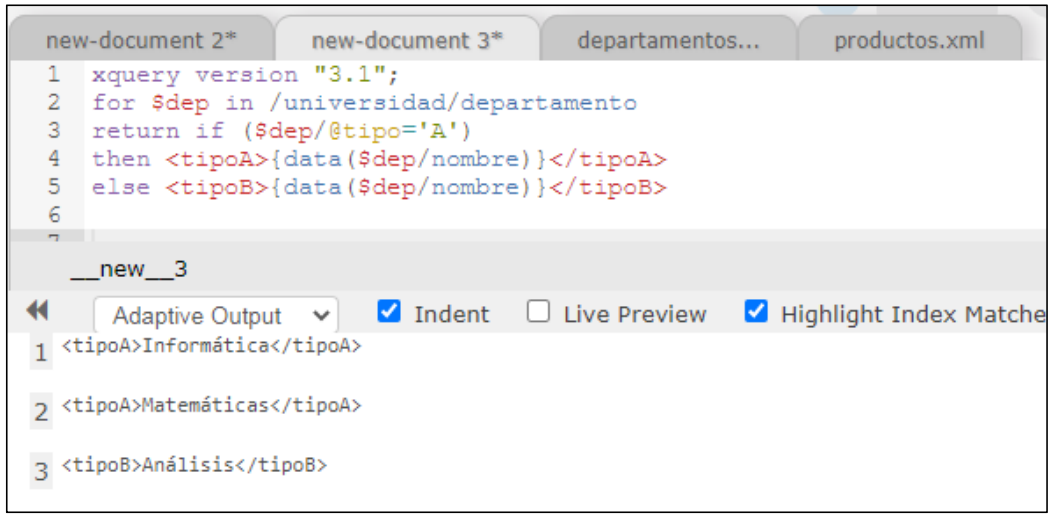
Ejemplos XQuery

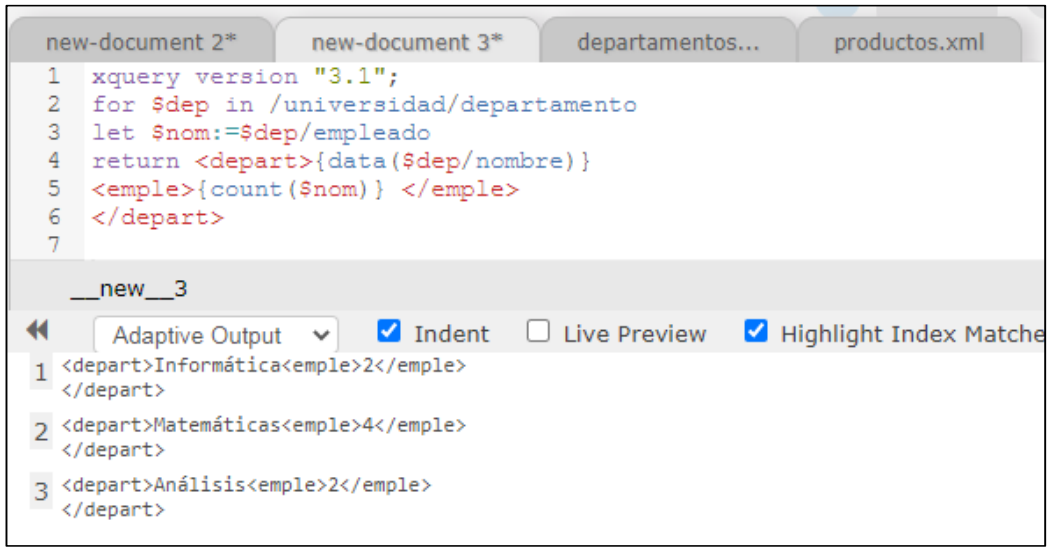
Prueba las siguientes expresiones en eXide:





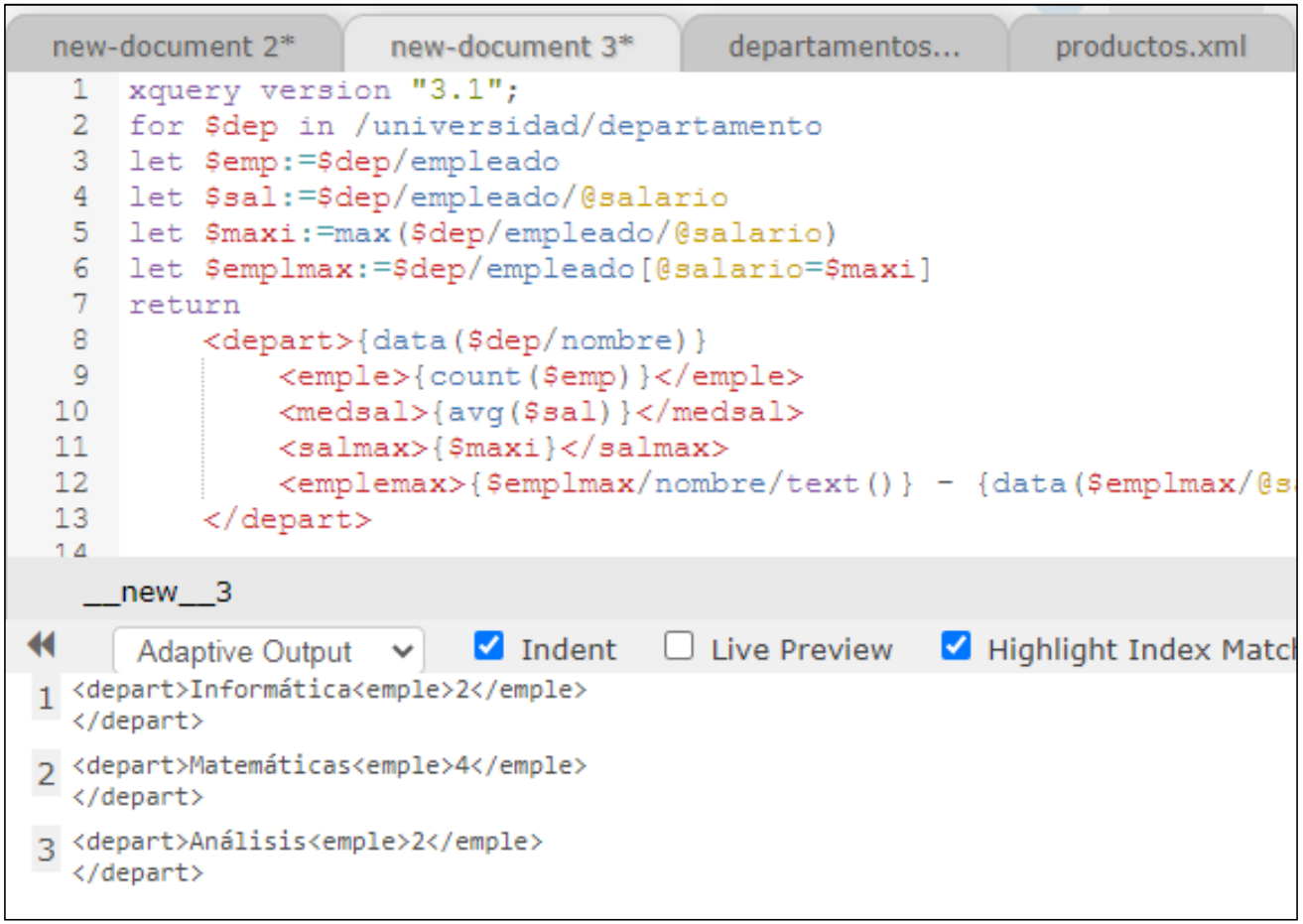
5. CONSULTAS XQUERY






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

5. CONSULTAS XQUERY



5. CONSULTAS XQUERY

Operadores en Xquery

- **Matemáticos: +, - , * , div, idiv (división entera), mod**
- **Comparación: <, > , =, !=, <=, >=, not()**
- **Redondeo: floor(), ceiling(), round()**
- **Funciones de agrupación: count(), min(), max(), avg(), sum()**
- **Funciones de cadena: concat(), string-length(), starts-with(), ends-with(), substring(), upper-case(), lower-case(), string()**
- **Uso general:**
  - distinct-values() → extrae los valores de una secuencia de nodos y crea una nueva secuencia con valores únicos, eliminando los nodos duplicados
  - empty() → devuelve cierto cuando la expresión entre paréntesis está vacía.
  - exists() → devuelve cierto cuando una secuencia contiene, al menos, un elemento
- **Los comentarios en XQuery van encerrados entre caras sonrientes (: blabla :)**

5. CONSULTAS XQUERY

Ejercicio 1. Resuelve las siguientes consultas utilizando el documento EMPLEADOS.xml

a. Obtén los nombres de oficio que empiezan por P
b. Obtén los nombres de oficio y el número de los empleados de cada oficio. Utiliza distinct-values
c. Obtén el número de empleados que tiene cada departamento y la media de salario redondeada

5. CONSULTAS XQUERY

Ejercicio 2. Utilizando el documento productos.xml, resuelve con XQuery:

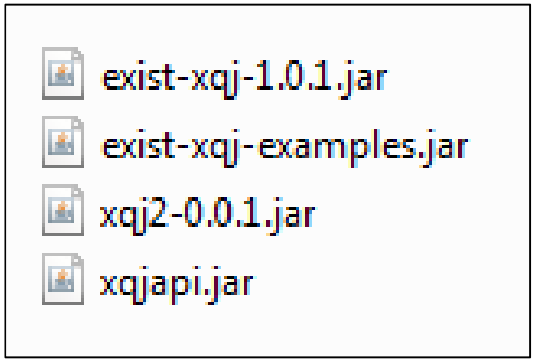
a. Obtén por cada zona el número de productos que tiene
b. Obtén la denominación de los productos entre las etiquetas <zona10></zona10> si son del código de zona 10, <zona20></zona20> si son del código de zona 20, etc.
c. Obtén por cada zona la denominación del o de los productos más caros.
d. Obtén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

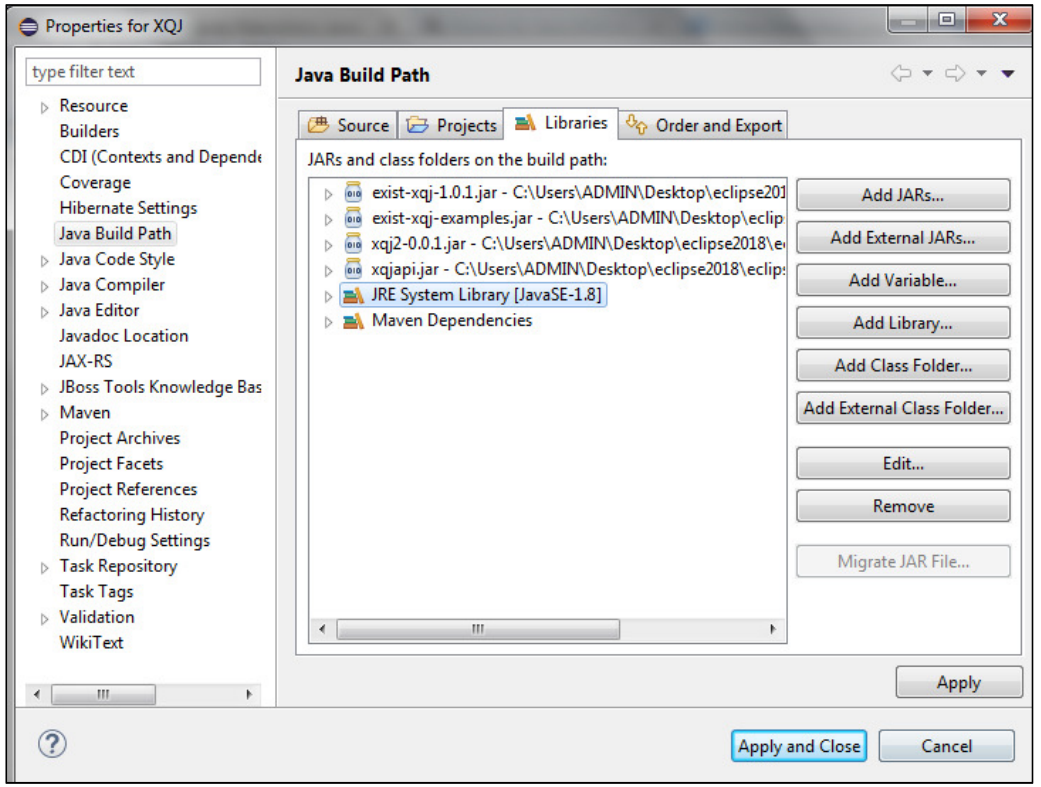
6. ACCESO A EXIST DESDE JAVA

- Ya hemos visto acceso a ficheros XML (DOM y SAX) , ahora veremos una API para acceder a la bbdd eXist
- Las APIs más conocidas son:
  - API XML:DB → cuyo objetivo es la definición de un método común de acceso a SGBD XML. Permite consulta, creación y modificación de contenido. La última actualización fue en el 2001 y aunque se sigue utilizando bastante se puede considerar obsoleta
  - API XQJ → propuesta de estandarización de interfaz Java para el acceso a bbdd XML nativas basado en modelo de datos XQuery. Es similar a JBDC. Es independiente del fabricante, fácil de usar pero solo permite realizar consultas.

6. API XQJ

- Para descargar la API accederemos a http://xqj.net/exist/
- Necesitamos agregar a nuestro proyecto las siguientes librerías:





6. API XQJ

Institut Joan d'Àustria

Configurar una conexión

- **XQDataSource:** identifica la fuente física de datos a partir de la cual crear conexiones. Cada implementación definirá las propiedades necesarias para efectuar la conexión.

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

- **XQConnection:** representa una sesión con la bbdd, manteniendo información de estado, transacciones, expresiones ejecutadas y resultados.

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

6. API XQJ

Clases y métodos para procesar consultas

- **XQExpression:** objeto creado a partir de una conexión para la ejecución de una expresión. Devuelve un **XQResultsetSequence.** La ejecución se produce llamando al método **executeQuery.**
- **XQPreparedExpression:** objeto creado a partir de una conexión para la ejecución de una expresión múltiples veces.
- **XQDynamicContext:** representa el contexto dinámico de una expresión (zona horaria, variables a usar en la expresión)
- **XQStaticContext:** representa el contexto estático para la ejecución de expresiones

6. API XQJ

- **XQItem:** representación de un elemento **XQuery**. Es inmutable y una vez creado su estado interno no cambia.
- **XQResultItem:** representación de un elemento de un resultado.
- Con XQJ no se necesita seleccionar la colección de los documentos XML, la búsqueda la realiza en todas las colecciones. Por tanto, a la hora de hacer consultas indicaremos la colección o el documento de la colección

6. API XQJ

Ejemplo 1 - Consulta Xquery

```
public static void main(String[] args) {
    try{
        XQDataSource server = new ExistXQDataSource();
        server.setProperty ("serverName","localhost");
        server.setProperty ("port","8080");
        server.setProperty ("user","admin");
        server.setProperty ("password", "admin");

        XQConnection conn = server.getConnection();

        String s = "for $pr in doc('dam2/productos.xml')/productos/produc return $pr";
        XQPreparedExpression consulta = conn.prepareExpression (s);
        XQResultSequence resultado = consulta.executeQuery();
        while (resultado.next()) {
            System.out.println(resultado.getItemAsString(null));
        }
        conn.close();
    } catch (XQException ex) {
        System.out.println("Error al operar"+ex.getMessage());
    }
}
```

Problems Javadoc Declaration Console Progress Debug Git Staging Variables Expressions

<terminated> Main (17) [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (31 ene. 2021 19:09:50)

```
<produc>
    <cod_prod xmlns="">1010</cod_prod>
    <denominacion xmlns="">Placa Base MSI G41M-P26</denominacion>
    <precio xmlns="">50</precio>
    <stock_actual xmlns="">10</stock_actual>
    <stock_minimo xmlns="">3</stock_minimo>
    <cod_zona xmlns="">10</cod_zona>
  </produc>
<produc>
    <cod_prod xmlns="">1011</cod_prod>
        <denominacion xmlns="">Micro Intel Core i5-2320</denominacion>
        <precio xmlns="">120</precio>
        <stock_actual xmlns="">3</stock_actual>
        <stock_minimo xmlns="">5</stock_minimo>
        <cod_zona xmlns="">10</cod_zona>
  </produc>
```

6. API XQJ

Ejemplo 2 - Introducción de parámetros en consultas Xquery

```
public static void main(String[] args) {
    try{
        XQDataSource server = new ExistXQDataSource();
        server.setProperty ("serverName","localhost");
        server.setProperty ("port","8080");
        server.setProperty ("user","admin");
        server.setProperty ("password", "admin");
        XQConnection conn = server.getConnection();

        //Definimos la variable $x que representa el ID de un departamento
        String s= "declare variable $x as xs:int external;"+  "/EMPLEADOS/EMP_ROW[DEPT_NO=$x]";
        XQPreparedExpression consulta = conn.prepareExpression(s);
        // Introducimos el valor del parámetro de la consulta
        int valor=10;
        consulta.bindInt(new QName("x"), valor, null);
        /* Más opciones en aquí */
        XQResultSequence resultado = consulta.executeQuery();
        while (resultado.next()) {
            System.out.println("Element E2: "+ resultado.getItemAsString(null));
        }
        conn.close();
    } catch (XQException ex) {
        System.out.println("Error al operar"+ex.getMessage());
    }
}
```

Problems Javadoc Declaration Console Progress Debug Git Staging

<terminated> Main (17) [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (31 ene. 2021

```
Element E2: <EMP_ROW>
    <EMP_NO xmlns="">7782</EMP_NO>
        <APELLIDO xmlns="">CEREZO</APELLIDO>
        <OFICIO xmlns="">DIRECTOR</OFICIO>
        <DIR xmlns="">7839</DIR>
        <FECHA_ALT xmlns="">1991-06-09</FECHA_ALT>
        <SALARIO xmlns="">2885</SALARIO>
        <DEPT_NO xmlns="">10</DEPT_NO>
  </EMP_ROW>
Element E2: <EMP_ROW>
    <EMP_NO xmlns="">7839</EMP_NO>
        <APELLIDO xmlns="">REY</APELLIDO>
        <OFICIO xmlns="">PRESIDENTE</OFICIO>
        <FECHA_ALT xmlns="">1991-11-17</FECHA_ALT>
        <SALARIO xmlns="">4100</SALARIO>
        <DEPT_NO xmlns="">10</DEPT_NO>
  </EMP_ROW>
```

6. API XQJ

Ejemplo 3 - Crear un nuevo archivo XML con datos del eXist

```
    //Devuelve todos los datos de los empleados del departamento 10 pero organizados
    //según la estructura de los datos en el documento empleados.xml
    String s= "let $titulo:= /EMPLEADOS/TITULO return " +  "<EMPLEADOS>{$titulo} "+
    "{for $em in /EMPLEADOS/EMP_ROW[DEPT_NO=10] return $em}</EMPLEADOS>";

    XQPreparedExpression consulta = conn.prepareExpression(s);
    XQResultSequence resultat = consulta.executeQuery();
    // Escribimos los datos de la consulta en un fichero xml
    BufferedWriter writer = new BufferedWriter(new FileWriter("empleados.xml"));
    // Cabecera XML
    writer.write("<?xml version='1.0' encoding='UTF-8'?>");
    writer.newLine();
    // Nodos XML
    while (resultat.next()) {
        String cad = resultat.getItemAsString(null);
        System.out.println(cad);
        writer.write(cad);
        writer.newLine();
    }
    writer.close();
    conn.close();
} catch (XQException ex) {
    System.out.println("Error al operar"+ex.getMessage());
}
```

6. PRACTICA 1

Realiza un programa que:

1. Devuelva los productos del catalogo productos.xml
2. Devuelva el número de productos con precio mayor a 50.
3. Devuelva todos los empleados del departamento 10.

6. PRACTICA 2

1. A partir del documento *universidad.xml*, haz un programa que muestre los empleados del departamento cuyo tipo es elegido por el usuario. Si no hay empleados o el tipo de departamento aportado por el usuario no existe, se debe de informar al usuario.
2. A partir de los documentos *productos.xml* y *zonas.xml*, haz un programa que reciba un número de zona por parámetro y genere un documento con nombre *zonaXX.xml* donde XX es la zona solicitada. El documento debe contener los productos de esta zona y las siguientes etiquetas: <cod_prod>, <denominación>, <precio>, <nombre_zona>, <director> y <stock>. Donde el *stock* se calcula restando el *stock* actual y el stock mínimo.