CONSULTAS	
/	Etiquetas de los documentos que cuelgan del nodo raíz
/nombre_nodo	Etiquetas de los nodos que cuelgan del nodo indicado
/nombre_nodo/node()	Etiquetas de los nodos que cuelgan del nodo indicado. No incluye el nodo en cuestión.
/nombre_nodo/text()	Sólo muestra el texto y no las etiquetas.
//	Mismo que / pero independientemente del documento
/*	Cualquier etiqueta
[]	Para seleccionar elementos concretos. Se introduce la condición en su interior.
<,>,<=,>=,=,!=	Comparación
or, and y not	Enlazar condiciones
|	Unir varias rutas
/ruta[@atributo_nodo **operador comparación** valor]	Atributos de los nodos cumplan una condición
(nodo1, nodo2)	Mostrar varios nodos. Se puede hacer con el operador |

FUNCIONES	
[número]	Devuelve el elemento que ocupa el lugar indicado
last()	Selecciona el último elemento del conjunto seleccionado
count()	Cuenta el número de elementos seleccionados
sum()	Devuelve la suma del elemento seleccionado
max()	Devuelve el máximo del elemento seleccionado
min()	Devuelve el mínimo del elemento seleccionado
avg()	Devuelve la media del elemento seleccionado
name()	Devuelve el nombre del elemento seleccionado
concat(cad1, cad2,…)	Concatena cadenas
start-with(cad1,cad2): starts-with	Obtiene los elementos donde la cad1 comienza por cad2 (cad1 like cad2%)
contains(cad1,cad2):	Obtiene los elementos en los cuales la cad1 contiene a la cad2 (cad1 like %cad2$)
string-length(argumento)	Devuelve el número de caracteres
div()	Realiza divisiones
mod()	Calcula el resto de la división
data(expresión Xpath)	Devuelve el texto de los nodos de la expresión sin las etiquetas
number(argumento)	Convierte a número una cadena, booleano o nodo
abs(num)	Devuelve el valor absoluto
ceiling(num)	Devuelve el entero más pequeño >= que num
floor(num)	Devuelve el entero más grande <= que num
round(num)	Redondea
string(argumento)	Convierte a cadena
compare(exp1,exp2)	Devuelve 0 si son iguales, 1 si exp1>exp2 y -1 si exp1<exp2
substring(cadena, comienzo, num)	Extrae de la cadena a partir de comienzo un num de caracteres
Substring(cadena,comienzo)	Extrae de la cadena a partir de comienzo hasta el final
lower-case(cadena)	Convierte a minúsculas
upper-case(cadena)	Convierte a mayúsculas

translate(cadena1,caract1,caract2)	Reemplaza en la cadena1 los caract1 por los caract2
ends-with(cadena1,cadena2)	Devuelve true si cadena1 termina en cadena2
year-from-date(fecha)	Devuelve el año (formato fecha AÑO-MES-DIA)
month-from-date(fecha)	Devuelve el mes
day-from-date(fecha)	Devuelve el día

RECORRIDO EN LOS EJES	
ancestor	Selecciona los antepasados (padres, abuelos, etc.) del nodo actual
ancestor-or-self	Selecciona los antepasados (padres, abuelos, etc.) del nodo actual y él mismo
attribute	Selecciona los atributos del nodo actual
child	Selecciona los hijos del nodo actual
descendant	Selecciona los descendientes (hijos, nietos, etc.) del nodo actual
descendant-or-self	Selecciona los descendientes (hijos, nietos, etc.) del nodo actual y él mismo
following	Selecciona todo el documento después de la etiqueta de cierre del nodo actual
following-sibling	Selecciona todos los hermanos que siguen al nodo actual
parent	Selecciona el padre del nodo actual
self	Selecciona el nodo actual

ACTUALIZACIONES	
update insert ELEMENTO into EXPRESION XPATH	Añadir como último hijo de los nodos especificados
update replace NODO with 'VALOR_NUEVO'	Sustituir el nodo especificado por un valor nuevo
update value NODO with 'VALOR_NUEVO'	Sustituir el VALOR del nodo especificado por un valor nuevo
update delete expresión xpath	Eliminar los nodos indicados
update rename NODO as NUEVO_NOMBRE	Renombra los nodos indicados

```
private static void conecta(){
        server = new ExistXQDataSource();

        XQDataSource server = new ExistXQDataSource();
        try {
            server.setProperty("serverName", "localhost");
            server.setProperty("port", "8080");
            server.setProperty("user", "dam2");
            server.setProperty("password", "dam2");

            con = server.getConnection();
        } catch (XQException ex) {
            Logger.getLogger(EjerciciosPractica.class.getName()).log(Level.SEVERE, null, ex);
        }

        System.out.println("Conectado con exito \n");
    }
```

```
private static void desconecta(){

        try {
            con.close();
        } catch (XQException ex) {
            Logger.getLogger(EjerciciosPractica.class.getName()).log(Level.SEVERE, null, ex);
        }

        System.out.println("\n Desconectado con exito");
    }
```

```
public void realizarConsulta (String inputConsulta) {
    try {
      XQPreparedExpression xqConsulta = connection.prepareExpression(inputConsulta);
      XQResultSequence xqResultado = xqConsulta.executeQuery();

      XQResultItem resultItem;
      while (xqResultado.next()) {
        resultItem = (XQResultItem) xqResultado.getItem();

        if (resultItem.getItemType().getBaseType() == XQItemType.XQBASETYPE_STRING) {
          System.out.println(resultItem.getAtomicValue());
        } else {
          System.out.println(eliminarNamespace(resultItem));
        }
      }
    } catch (XQException ex) {
      ex.printStackTrace();
    }
  }
```

```
private static void modificacion(String textoDML){
        try {

            XQExpression expresion;

            expresion = con.createExpression();

            expresion.executeCommand(textoDML);

        } catch (XQException ex) {
            Logger.getLogger(EjerciciosPractica.class.getName()).log(Level.SEVERE, null, ex);
        }

        System.out.println("Modificacion realizada con exito");
    }
```