

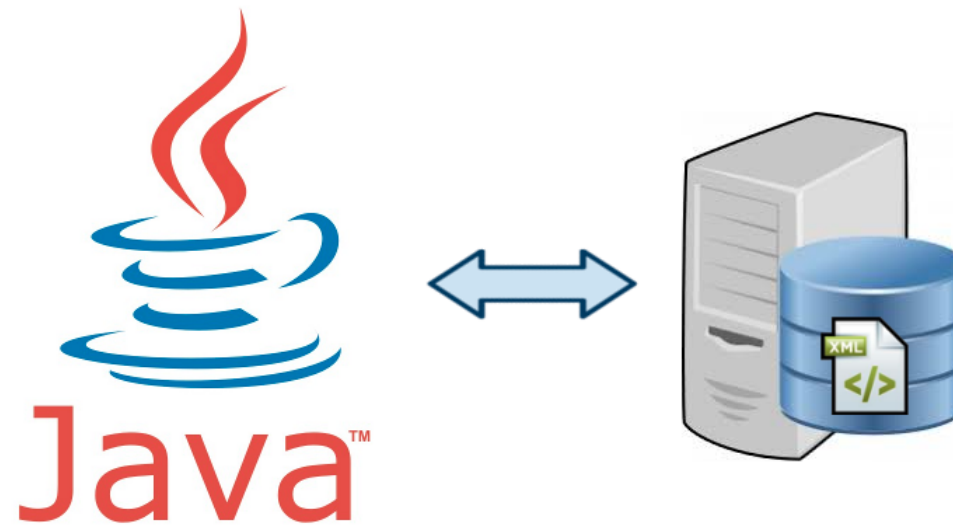
---

## Accés a Dades

---

## Tema 8: Bases de Dades XML

---



## 1 - Introducció

---

L'**XML** és, segons la definició feta pel World Wide Web Consortium (W3C), un format simple basat en text per a representar informació estructurada: documents, dades, configuracions, llibres, transaccions, factures i molt més. Va ser derivat d'un format estàndard més antic, anomenat **SGML**, amb la finalitat de ser més adequat per a la seua utilització en la web.

L'**XML**, avui en dia, és un dels formats més utilitzats per a l'**intercanvi** d'informació estructurada: entre els programes, entre les persones, entre ordinadors i persones, tant a nivell local com a través de les xarxes. El fet que la informació s'intercanvia en format **XML** ha implicat l'aparició de mecanismes que permeten enregistrar aquesta informació en format **XML**, de manera que no siga necessari efectuar traduccions a altres formats.

L'emmagatzematge de la informació en format **XML** no s'ha fet d'una única manera, sinó que han aparegut diverses estratègies d'emmagatzematge que ens interessa conèixer, així com els avantatges i els inconvenients d'aquestes estratègies.

Com en el tema anterior amb els objectes, veurem dues de les estratègies: la primera guardar en un SGBD Relacional, que incorpora conceptes de XML; la segona guardar en un SGBD enfocat estrictament a XML, les **Bases de Dades Natives XML**. Ens interessa, saber les seues principals característiques per poder fer programes (en Java) que puguin guardar o accedir a informació guardada en XML.

## 2 - Estratègies d'emmagatzematge de XML

L'XML és un llenguatge que permet tenir informació estructurada, ja siga per a intercanvi entre plataformes o, simplement, per facilitar un ràpid accés a continguts. La seua àmplia acceptació ha dut la necessitat d'idear mecanismes per poder guardar, de manera fàcil i àgil, grans volums de documents XML.

Hi ha més d'una manera, més d'una estratègia per a guardar la informació XML, però abans de definir-les hem de fer una xicoteta reflexió sobre els tipus de documents XML que ens podem trobar.

### Document centrats en dades

Anomenats en anglès **data-centric**, estan pensats per a l'intercanvi entre plataformes. Solen ser documents amb estructures regulars i ben definides. Les dades que transmeten són simples i ben definides (encara que pot ser que siguen molt extenses), i en moltes ocasions són actualitzables. Acostumen a tenir com a origen una Base de Dades i, en conseqüència, no és massa important la seua persistència com a document XML. Aquest podria ser un exemple:

```
<clients>
  <client>
    <codi>10</codi>
    <raoSocial>Components Informàtics</raoSocial>
  </client>
  <client>
    <codi>20</codi>
    <raoSocial>Institut Obert de Catalunya</raoSocial>
  </client>
</clients>
```

Es veu perfectament la seua estructura fortament jeràrquica, clara i concisa.

### Document centrats en el document

Anomenats en anglès **document-centric**, amb una estructura irregular. L'origen i el destí acostuma a estar en les persones i solen estar fets a mà. Hi ha que poden arribar a tenir un cert format, però no és estricte ni definit i en tal cas parlem de dades semiestructurades. Exemples de documents centrats en el document són els llibres, els correus electrònics, els anuncis, etc.

A continuació veiem un exemple de document XML centrat en el document. Es tracta d'un document descriptiu d'un producte (torró) en el qual l'autor ha inserit diverses marques per facilitar-ne la consulta i la formatació en un navegador. Podeu comprovar-ho copiant el contingut en un fitxer amb extensió **.xml** (per exemple **Torro.xml**) i obrint-lo amb un navegador.

```

<Producte>

<Descripcio>
El <NomProducte>Torró</NomProducte> és un <Sumari>dolç que es fa generalment d'<Ingredient>ametla</Ingredient>,
<Ingredient>mel</Ingredient> i <Ingredient>ou</Ingredient></Sumari>, generalment cuit.
</Descripcio>

<Origen>

<Paragraf>L'origen del torró no és del tot clar, la teoria més plausible és que siga d'origen <i>àrab</i>, com altres postres
amb ametles, i des de les comarques del Sud del País Valencià, especialment de <b>Xixona</b> i <b>Alacant</b> estant es va
popularitzar a altres zones peninsulars, el sud de França i altres parts del món</Paragraf>

<Paragraf>Potser el nogat occità i el <i>torrone</i> o torró italià van néixer també directament dels àrabs, o no se sap si
per influència del torró català, que ja existia a l'Edat Mitjana i que en aquell moment, i en concret al segle XV, la catalana
era la cuina més prestigiosa d'Europa.</Paragraf>

</Origen>

<Varietats>Varietats
<List>
<Item><i>Torró de Xixona</i></Item>
<Item><i>Torró d'Alacant</i></Item>
<Item><i>Torró de Xerta</i></Item>
<Item><i>Torró de Xerta</i></Item>
<Item><i>Torró de Xerta</i></Item>
</List>
</Varietats>

</Producte>

```

Si li afegírem un full d'estil (stylesheet o css) de manera prou senzilla podríem donar-li un aspecte més agradable, tal i com es va veure en el mòdul de Llenguatge de Marques. I es veu clarament que està centrat en el propi document, sense estructurar les dades massa. Aquest podria ser un exemple de css (el podríem anomenar **Torro.css**):

```

Producte {
    background-color: #FFD7D7;
    font-size: medium;
    display: block;
    margin: 10px;
}

NomProducte {
    font-size: large;
    color: red;
}

Descripcio{
    display: block;
}

Origen {
    display: block;
    margin-top: 20px;
}

```

```

    border-top: 1px;
}
Paragraf {
    display: block;
}
Varietats {
    display: block;
    font-size: x-large;
    color: green;
    margin-top: 20px;
}
List {
}
Item {
    display: list-item;
    margin-left: 20px;
    margin-top: 5px;
    font-size: large;
    color: black;
}
i {
    font-style: italic;
    color: Blue;
}
b {
    font-weight: bold;
}

```

I si l'apliquem al document xml posant la següent línia al principi del document:

```
<?xml-stylesheet type="text/css" href="Torro.css"?>
```

tindríem el següent aspecte:

El **Torró** és un dolç que es fa generalment d'ametla, mel i ou, generalment cuit.

L'origen del torró no és del tot clar, la teoria més plausible és que siga d'origen *àrab*, com altres postres amb ametles, i des de les comarques del Sud del País Valencià, especialment de **Xixona** i **Alacant** estant es va popularitzar a altres zones peninsulars, el sud de França i altres parts del món

Potser el nogat occità i el *torrone* o torró italià van néixer també directament dels àrabs, o no se sap si per influència del torró català, que ja existia a l'Edat Mitjana i que en aquell moment, i en concret al segle XV, la catalana era la cuina més prestigiosa d'Europa.

### Varietats

- *Torró de Xixona*

- *Torró d'Alacant*
- *Torró de Xerta*
- *Torró de Xerta*
- *Torró de Xerta*

La classificació de documents en **data-centric** i **document-centric** no és sempre directa i clara i, en múltiples ocasions, el contingut estarà barrejat o serà difícil de catalogar en un o altre tipus. Així, podem tenir documents centrats en dades (com una comanda o una factura) on alguna de les dades tinga codificació lliure (per exemple, part de la descripció de les línies) i documents centrats en el document (com un manual d'usuari) amb una part regular amb format estricte i ben definit (com el nom de l'autor i la data de revisió). Malgrat això, la caracterització dels documents en **data-centric** i **document-centric** s'utilitza per ajudar a decidir quina és la millor estratègia d'emmagatzematge.

Ens cal, doncs, conèixer les tècniques d'emmagatzematge existents per, una vegada conegudes, decidir quina tècnica és més adequada segons el tipus de documents que hem de guardar i la gestió que en pretenem.

## Estratègies d'emmagatzematge

Tècnicament hi ha tres possibles estratègies per guardar els documents XML:

1. Guardar-los directament en el sistema d'**arxius del sistema operatiu**. Opció molt pobre, ja que les funcionalitats que permet fer sobre el document queden limitades i definides pel Sistema Operatiu. No permet efectuar operacions sobre el contingut i només es permet el moviment del document com a una unitat. Si es tracta d'una petita quantitat de dades guardades en uns pocs documents XML, aquesta opció pot funcionar, però no és gens recomanable per a gestionar un volum elevat de documents XML.
2. Guardar-los en un **Sistema Gestor de Bases de Dades Relacional** o **Orientat a Objectes**. Aquesta opció obliga a una transformació del document XML cap al model que corresponga (relacional o orientat a objectes), per a poder guardar-lo.
3. Guardar-los en una **base de dades XML nativa**. Aquesta opció permet guardar directament el document XML a la base de dades sense cap tipus de transformació. Les bases de dades XML natives han estat dissenyades especialment per a l'emmagatzematge d'XML. L'abreviatura és **BD-XML nativa**. L'abreviatura anglesa és **NXD**.

Deixant de banda la primera opció, podríem dir com a **regla general** que els documents de tipus **data-centric** s'emmagatzemarien en una Base de Dades tradicional (relacional o orientada a objectes). Això es pot fer bé fent un mapatge de les dades, tenint la possibilitat de fer-lo l'usuari, per mitjà d'eines de tercers, o per les capacitats que el SGBD incorpora per a poder tractar XML. En aquest últim cas, es diu que la base de dades està **habilitada per a XML** (*XML-enabled*).

En canvi, els documents de tipus **document-centric** es guardarien en una base de dades **XML nativa**.

Aquestes regles no són absolutes. D'una banda, els documents **data-centric** (sobretot si es tracta de dades semiestructurades) es poden guardar en bases de dades natives. De l'altra, els documents **document-centric** poden ser guardats en bases de dades tradicionals en aquells casos en què es precisen poques funcionalitats específiques XML.

En el moment en què va aparèixer la necessitat de guardar documents XML, l'emmagatzematge de dades estava centrat en els SGBD Relacionals o Orientats a Objectes i van sorgir dues tendències:

- D'una banda, els grans SGBD existents van començar a introduir funcionalitats XML, donant lloc a l'aparició dels SGBD-XML habilitats.

- D'una altra banda, van començar a aparèixer els SGBD-XML natives.

Avui en dia, els límits entre els SGBD-XML habilitades i els SGBD-XML natives s'estan desdibuixant, ja que les versions actuals de molts SGBD tradicionals (Relacionals i Orientats a Objectes) ja incorporen capacitats natives d'XML i alguns SGBD-XML natives faciliten l'emmagatzematge de fragments de documents XML en bases de dades externes (usualment relacionals).

En la pregunta següent ens centrarem en guardar en un SGBD relacional, primer fent un mapatge manual, i després utilitzant les eines que ens proporciona el SGBD per a manipular documents XML. Ho farem en PostgreSQL, que és un **SGBD habilitat per a XML**.

### 3 - Emmagatzematge en SGBD tradicionals

#### ( VOLUNTARI )

Per a il·lustrar l'emmagatzematge en SGBD tradicionals, ens centrarem en un SGBD Relacional, concretament en **PostgreSQL**, que ja l'havíem utilitzat amb anterioritat.

Tenim 3 possibilitats bàsiques de guardar informació XML en un SGBD Relacional:

- Guardar el document XML en un camp BLOB o CLOB (que en PostgreSQL és **TEXT**). Evidentment podrem guardar el document, però realment el SGBD no ens està oferint cap utilitat. La utilització posterior serà molt pesada: no es pot fer cap tipus de consulta dins del document XML, i per tant estariem obligats a fer tota la feina en el programa que accedisca al document (el programa Java).
- Transformació (mapatge) de l'estructura de dades del document XML cap al model relacional, per guardar les dades en taules. La idea principal d'aquesta opció és observar l'estructura del document XML, amb els elements i atributs que apareixen i efectuar la conversió al model relacional. Ens donarà molta feina per a fer el mapatge, encara que després es podran fer consultes molt elaborades. Mirem un exemple, on en principi sembla prou fàcil passar de l'estructura jeràrquica del document XML a taules del model relacional

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ComandesVenda>
  <ComandaVenda>
    <Número>1234</Número>
    <Client>Indústries Fèrriques</Client>
    <Data>29/10/2011</Data>
    <Línia Número="1">
      <Article>PIL001</Article>
      <Quantitat>12</Quantitat>
      <Preu>12.85</Preu>
    </Línia>
    <Línia Número="2">
      <Article>CAT010</Article>
      <Quantitat>30</Quantitat>
      <Preu>5.25</Preu>
    </Línia>
  </ComandaVenda>
  <ComandaVenda>
    <Número>1235</Número>
    <Client>Eines del Berguedà</Client>
    <Data>30/10/2011</Data>
    <Línia Número="1">
      <Article>XQT301</Article>
      <Quantitat>14</Quantitat>
      <Preu>23.2</Preu>
    </Línia>
  </ComandaVenda>
</ComandesVenda>
```



```
</ComandesVenda>
```

Les taules que quedarien en el Model Relacional serien:

```
Comanda (numero, client, data)
```

```
Linia_Comanda (num_com, num_linia, article, quantitat, preu)
```

Com comentàvem, la informació quedarà ben guardada i consultar les dades serà fàcil i potent (amb SQL), però duu l'inconvenient d'una gestió del mapatge total. I encara que el mecanisme sembla simple, no sempre és senzill fer aquesta conversió, ja que el model relacional i el XML parteixen de conceptes molt diferents:

- El model relacional està basat en dades bidimensionals sense jerarquia ni ordre, mentre que el model XML està basat en arbres jeràrquics on l'ordre és rellevant.
  - En un document XML poden haver dades repetides, mentre el model relacional fuig de les repeticions.
  - Les relacions i les estructures dins dels documents XML no sempre són òbvies.
  - I, a més, què passa si necessitem tenir el document XML de nou? Fer el procés invers no sempre és trivial. Un dels conceptes difícils és determinar quines dades eren atributs i quines eren elements.
- Utilitzar eines específiques per al tractament de la informació XML. Només es podrà fer en SGBD XML habilitats. Aquest serà el punt que tractarem.

En els següents subapartats farem un resum de les possibilitats que ens ofereix **PostgreSQL** com a **XML habilitat**.

### 3.1 - Tipus de dades XML

#### Nota inicial

Les proves i exercicis sobre PostgreSQL les realitzarem cadascú en un usuari diferent, per a no interferir entre tots. Podríem instal·lar cadascú el seu SGBD PostgreSQL, administrar-lo per a crear els usuaris pertinents i treballar amb PgAdmin de forma còmoda. Si voleu provar-lo teniu una guia d'Administració PostgreSQL en els annexos.

Tanmateix, com que l'únic que volem és connectar-nos com a clients i tenim un servidor PostgreSQL al qual podem accedir tant des de dins de l'Institut com des de fora, no ens caldrà.

L'usuari i BD serà el mateix que ja es va utilitzar en el Tema 6. Si no tens clar quin és aquest usuari i BD, contacta amb el teu professor.

En altres ocasions ens hem aconformat utilitzant la perspectiva **Database Development**. Lamentablement, ara no ens és suficient, ja que no suporta els tipus especials, com el tipus **XML**. Per tant ens fa falta **PgAdmin** per a poder treballar còmodament. Si no el teniu instal·lat, ho podeu fer fàcilment des de la pàgina [www.pgadmin.org](http://www.pgadmin.org), però cuideu que siga la versió **PgAdmin 3** (la versió PgAdmin 4 potser no siga compatible amb el nostre servidor). Fins i tot existeix una versió **portable** que es pot instal·lar tant en Windows com en Linux (executant-la amb el **Wine**), que si no heu d'utilitzar pgAdmin per a res més, és la que us recomane tilitzar. Us la podeu descarregar de:

<http://sourceforge.net/projects/pgadminportable/files/latest/download>

Tindrem un nou tipus de dades **XML**, i ens permetrà guardar tot un document XML com el valor d'un camp d'aquest tipus. Anem a crear una taula d'exemple, que contindrà diferents documents XML.

```
CREATE TABLE p_xml
(
  num integer PRIMARY KEY,
  doc xml
)
```

Fem un primer exemple d'introducció d'un document. Posarem el valor directament en una sentència INSERT:

```
INSERT INTO P_XML VALUES(1, '<clients>
  <client><codi>10</codi>
    <raoSocial>Components Informàtics</raoSocial>
  </client>
  <client><codi>20</codi>
    <raoSocial>Institut Obert de Catalunya</raoSocial>
  </client>
</clients>')
```

```
</clients>')
```

PostgreSQL farà una conversió automàtica de **VARCHAR** a **XML**, i es guarda bé. Podríem fer la prova d'introduir malament la informació XML, per exemple llevant una etiqueta `</client>`. PostgreSQL detectarà que no està ben format i donarà un error.

## 3.2 - Funcions de conversió

De tota manera disposarem de funcions de conversió entre VARCHAR i XML i a l'inrevès, que ens assegurin els tipus desitjat. Aquestes funcions són:

- **XMLPARSE ( {DOCUMENT | CONTENT} *varchar* )** , que tornarà un valor de tipus **xml**.

Podem posar **document** davant de la cadena, si és un document complet, i si no ho és posarem **content** (totes les etiquetes han d'estar tancades, però no cal que tinga una arrel única). Aquesta seria una manera alternativa d'introduir la mateixa informació XML que abans

```
INSERT INTO P_XML VALUES (2,XMLPARSE(CONTENT '<clients>
      <client>
        <codi>10</codi>
        <raoSocial>Components Informàtics</raoSocial>
      </client>
      <client>
        <codi>20</codi>
        <raoSocial>Institut Obert de Catalunya</raoSocial>
      </client>
    </clients>') )
```

- **XMLSERIALIZE ( {DOCUMENT | CONTENT} *xml AS tipus* )** , que tornarà un valor del tipus especificat (normalment VARCHAR)

Aquest exemple trau dues vegades el mateix valor, però la primera vegada és de tipus XML, i la segona VARCHAR

```
SELECT doc,XMLSERIALIZE(CONTENT doc AS VARCHAR) AS doc2 FROM P_XML
```

Com es pot comprovar en la següent imatge, el primer camp és de tipus XML, i el segon VARCHAR:

Query - r00 sobre r00@89.36.214.106:5432 \*

Editor SQL

Constructor de Consultes Gràfic

Consultes anteriors

Eborra

Eborra-ho tot

SELECT doc,XMLSERIALIZE(CONTENT doc AS VARCHAR) AS doc2 FROM P\_XML

Subfinestra de sortida

Sortida de dades

Explain

Missatges

Historial

	doc xml	doc2 character varying
1	<clients> <client><codi>10</codi> <raoSocial>Components Inf	<clients> <client><codi>10</codi> <raoSocial>Components Informàt
2	<clients>	<clients>

OK.

Unix

Ln 1, Col 1, Ch 1

2 registres, 41 ms

Observeu també que cada resultat pot ocupar més d'una línia.

### 3.3 - Funcions que produeixen contingut XML

Veurem en aquest apartat les funcions que ens proporciona PostgreSQL per a generar contingut XML. El resultat de totes elles és de tipus XML, i serviran per a generar un element, atribut, element arrel, concatenar, ... de manera que podrem generar contingut XML ben format.

Són aquestes:

#### XMLCOMMENT

*Sintaxi:* **xmlcomment (text)**

*Descripció:* genera un comentari XML, amb el contingut del paràmetre. En aquest contingut no pot anar '--', per a no confondre amb l'etiqueta del comentari

*Exemple:*

```
SELECT xmlcomment('Hola');

      xmlcomment
-----
<!--Hola-->
```

#### XMLCONCAT

*Sintaxi:* **xmlconcat (xml [, ... ])**

*Descripció:* concatena una llista de valors XML, sent el resultat també XML

*Exemple:*

```
SELECT xmlconcat('<primer>Hola</primer>', '<segon>Adéu</segon>');

      xmlconcat
-----
<primer>Hola</primer><segon>Adéu</segon>
```

## XMLEMENT

*Sintaxi:* **xmlement** (name *nom* [, **xmlattributes** (valor [**AS** *nomatribu*] [, ...]) [, *contingut*, ...])

*Descripció:* genera un element amb el nom especificat, possibles atributs amb els seus valors, i possible contingut (si no posem contingut serà un element buit)

*Exemples:*

```
SELECT xmlement(name primer);
```

```
xmlement
-----
<primer/>
```

```
SELECT xmlement(name segon, 'Hola');
```

```
xmlement
-----
<segon>Hola</segon>
```

```
SELECT xmlement(name tercer, xmlattributes ('Primer atribut' as a1), 'Hola');
```

```
xmlement
-----
<tercer a1="Primer atribut">Hola</tercer>
```

Anem a posar ara un exemple que combine tot un poc:

```
SELECT xmlement(name quart, xmlattributes(current_date as data),
          xmlement(name quart_1,
          xmlcomment('comentari1'),
          xmlement(name quart_2, 'valor quart 2')));
```

```
xmlement
-----
<quart data="2016-12-21"><quart_1/><!--comentari--><quart_2>valor quart 2</quart_2></quart>
```

O un altre molt interessant que crea un element comarca per a cada comarca de la taula COMARQUES, posant-li a més com a atribut la província:

```
SELECT xmlement(name Comarca,xmlattributes(provincia as Provincia),nom_c)
FROM COMARQUES
ORDER BY provincia,nom_c;
```

El resultat seria aquest:

	<b>xmlelement xml</b>
1	<comarca provincia="Alacant">Alacantí</comarca>
2	<comarca provincia="Alacant">Alcoià</comarca>
3	<comarca provincia="Alacant">Alt Vinalopó</comarca>
4	<comarca provincia="Alacant">Baix Segura</comarca>
5	<comarca provincia="Alacant">Baix Vinalopó</comarca>
6	<comarca provincia="Alacant">Comtat</comarca>
7	<comarca provincia="Alacant">Marina Alta</comarca>
8	<comarca provincia="Alacant">Marina Baixa</comarca>
9	<comarca provincia="Alacant">Vinalopó Mitjà</comarca>
10	<comarca provincia="Castell&#xF3;">Alcalatén</comarca>
11	<comarca provincia="Castell&#xF3;">Alt Maestrat</comarca>
12	<comarca provincia="Castell&#xF3;">Alt Millars</comarca>
13	<comarca provincia="Castell&#xF3;">Alt Palància</comarca>

on podria sorprendre el tractament dels accents i caràcter especials, però no oblidem que és la manera de codificar aquestos caràcters en XML

## XMLFOREST

*Sintaxi:* **xmlforest (contingut [ AS nom] [ , ... ])**

*Descripció:* crea una sèrie d'elements i els seqüència un darrere de l'altre. És especialment útil quan accedim a alguna taula

*Exemple:*

```
SELECT xmlforest('Hola' AS primer,'Adéu' AS segon);

-----
xmlconcat
-----
<primer>Hola</primer><segon>Adéu</segon>
```

Com comentàvem és especialment útil quan accedim a taules. El següent exemple crea els element nom i província amb els valors de cadascuna de les files:

```
SELECT xmlforest(nom_c,provincia)
FROM COMARQUES
ORDER BY nom_c;
```



```

                xmlforest
-----
<nom>Alacantí</nom><provincia>Alacant</provincia>
<nom>Alcalatén</nom><provincia>Castelló</provincia>
<nom>Alcoià</nom><provincia>Alacant</provincia>
<nom>Alt Maestrat</nom><provincia>Castelló</provincia>
...

```

Si açò ho enganxem amb la creació de l'element comarca, el tindrem ben format, d'una forma equivalent a una anterior, però sense atributs, tot subelements:

```

SELECT xmlelement (name Comarca,xmlforest (nom_c,provincia))
FROM COMARQUES
ORDER BY nom_c;

                xmlelement
-----
<comarca><nom>Alacantí</nom><provincia>Alacant</provincia></comarca>
<comarca><nom>Alcalatén</nom><provincia>Castelló</provincia></comarca>
<comarca><nom>Alcoià</nom><provincia>Alacant</provincia></comarca>
<comarca><nom>Alt Maestrat</nom><provincia>Castelló</provincia></comarca>
<comarca><nom>Alt Millars</nom><provincia>Castelló</provincia></comarca>
...

```

## XMLAGG

*Sintaxi:* **xmagg (xml)**

*Descripció:* podem considerar-la una funció d'agregat, que concatena tots els element que li arriben. És a dir, que **xmlconcat** concatena els valors dels paràmetres, però de forma aïllada, cada fila és en un resultat diferent. En canvi **xmlagg** els junta tots en un únic resultat.

*Exemples:*

```

SELECT xmlagg(xmlforest('Hola' AS primer,'Adéu' AS segon));

                xmlagg
-----
<primer>Hola</primer><segon>Adéu</segon>

```

Que així queda aproximadament com la concatenació. Però la utilitat gran és quan agreguem els elements d'una taula (recordeu que el paràmetre ha de ser XML)

```

SELECT xmlagg(xmlforest (nom_c,provincia))

```

```

FROM COMARQUES;

-----
                        xmlagg
-----
<nom>Safor</nom><provincia>València</provincia><nom>Horta Sud</nom><provincia>València</provincia> ...

```

Per tant ara ja tenim la manera d'ajuntar totes les comarques en un únic XML i posar-li nom al conjunt:

```

SELECT xmlelement(name comarques, xmlagg(xmlelement(name comarca,xmlforest(nom_c,provincia))))
FROM COMARQUES;

-----
                        xmlconcat
-----
<comarques><comarca><nom>Safor</nom><provincia>València</provincia></comarca><comarca><nom>Horta Sud</nom>
<provincia>València</provincia></comarca>...</comarques>

```

## XMLROOT

*Sintaxi:* **xmlroot(xml, version text | no value [, standalone yes|no|no value])**

*Descripció:* modifica les propietats del node arrel del XML que tenim com a primer paràmetre. Si s'especifica la versió, reemplaçarà la que hi haja. I el mateix amb l'atribut standalone.

*Exemple:*

```

SELECT xmlroot(xmlparse(document '<primer>abc</primer>'), version '1.0', standalone yes);

-----
                        xmlroot
-----
<?xml version="1.0" standalone="yes"?> <primer>abc</primer>

```

Com podem comprovar, amb aquestes funcions podem formar un document sencer, i de forma prou senzilla.

Anem a aprofitar-ho doncs per a inserir una nova fila a la taula P\_XML, on teníem un camp XML. Per a poder introduir totes les comarques en un document, i afegir-lo a aquesll taula podríem fer:

```

INSERT INTO P_XML
SELECT 3, xmlroot(xmlparse(document (SELECT xmlelement(name comarques,
                        xmlagg(xmlelement(name comarca,xmlforest(nom_c,provincia))))

```

```
version '1.0', standalone yes);          FROM COMARQUES)) ,
```

A continuació veurem una forma molt més senzilla d'introduir tota la taula de comarques com un document XML, però aquesta forma que acabem d'utilitzar ens permetria modificar al gust el format (per exemple posant la província com un atribut)

### 3.4 - Convertir taules senceres a XML

En l'últim exemple de l'apartat anterior havíem aconseguit forma un document XML a partir de la taula **Comarques** i inserir-lo en un camp XML (de la taula P\_XML). Aquella manera ens permet confeccionar al nostre gust l'estructura del XML. Per contra, és un poc laboriós.

Anem a veure ara dues funcions que ens permeten convertir una taula i una consulta, respectivament, a XML amb l'estructura que totes les columnes seran elements. No tindrem per tant tanta versatilitat com en l'apartat anterior. Però per contra la senzillesa és extrema.

#### TABLE TO XML

*Sintaxi:* `table_to_xml(taula, tract_nul, format, espai_noms)`

*Descripció:* converteix una taula en un document XML. El primer paràmetre és la taula a convertir. El segon (booleà) indica si s'inclouen els valors nuls com element buits. El tercer (booleà) implica el format que tindran els elements. El quart serveix per definir l'espai de noms; el podem deixar com una cadena nula (").

*Exemple:*

Anem a fixar-nos en el tercer paràmetre. Si està a *false*, el nom de la taula serà l'element arrel, i per a cada fila crearà un element anomenat row, que inclourà tots els camps:

```
SELECT table_to_xml('COMARQUES', false, false, '');
```

ens donarà aquesta eixida:

```
<comarques xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <row>
    <nom>Safor</nom>
    <provincia>València</provincia>
  </row>

  <row>
    <nom>Horta Sud</nom>
    <provincia>València</provincia>
  </row>

  ...
</comarques>
```

En canvi si el tercer paràmetre està a *true*, traurà una successió d'element amb el mateix nom que la taula, amb els camps com a elements:

```
SELECT table_to_xml('COMARQUES', false, true, '');
```

ara l'eixida serà:

```
<comarques xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <nom>Safor</nom>
  <provincia>València</provincia>
</comarques>

<comarques xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <nom>Horta Sud</nom>
  <provincia>València</provincia>
</comarques>

...
```

És a dir, de la primera manera el document està ben format, i de la segona és una successió d'elements.

Un exemple d'utilització (que aprofitarem per a introduir en la taula P\_XML) seria intentar canviar el nom dels elements *row* per *comarca*. La solució podria ser utilitzar la funció **replace()** per a canviar el que volem. Només haurem d'anar en compte de convertir el document XML generat per TABLE\_TO\_XML a text, per poder utilitzar la funció replace, i després tornar a convertir-lo a XML. Introduïrem també les taules POBLACIONS i INSTITUTS convertides en XML (substituint *row* per un més adequat)

```
INSERT INTO P_XML
  SELECT 4, replace(table_to_xml('COMARQUES',false,false,'')::text,'row','comarca')::xml;
```

```
INSERT INTO P_XML
  SELECT 5, replace(table_to_xml('POBLACIONS',false,false,'')::text,'row','poble')::xml;
```

```
INSERT INTO P_XML
  SELECT 6, replace(table_to_xml('INSTITUTS',false,false,'')::text,'row','institut')::xml;
```

En la següent imatge es mostra el contingut de la taula **P\_XML**, on s'han ampliat les noves files per poder veure el contingut:

	num [PK] integer	doc xml
1	1	<clients>
2	2	<clients>
3	3	<?xml version="1.0" standalone="yes"?><comarques><comarca><nom_c>Safor</nom_c><provincia>València</provincia></comarca>
4	4	<comarques xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">  <comarca> <nom_c>Safor</nom_c> <provincia>València</provincia> </comarca>  <comarca> <nom_c>Horta Sud</nom_c>
5	5	<poblacions xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">  <poble> <cod_m>46001</cod_m> <nom>Ademuz</nom> <poblacio>1179</poblacio> <extensio>100.40</extensio> <altura>660</altura> <longitud>1º17'06"W</longitud> <latitud>40º04'15"N</latitud> <llengua>C</llengua> <nom_c>Racó</nom_c> </poble>
6	6	<instituts xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">  <institut> <codi>12005283</codi> <nom>IES VIOLANT de CASALDUCH</nom> <adreca>DE CASTELLÓN</adreca> <numero>S/N</numero> <codpostal>12560</codpostal> <cod_m>12028</cod_m> </institut>
*		

## QUERY TO XML

Sintaxi: `query_to_xml(sent_SQL, tract_nul, format, espai_noms)`

*Descripció:* absolutament equivalent a l'anterior, però la font de les dades no serà una taula, sinó una sentència SQL.

*Exemple:*

```
SELECT query_to_xml('SELECT nom_c, COUNT(*) AS num_p, AVG(altura) AS alt_mitj
                     FROM POBLACIONS
                     GROUP BY nom_c
                     ORDER BY 3 DESC',
                     false, false, '');
```

I l'eixida seria aquesta:

```
<table xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <row>
    <comarca>Ports</comarca>
    <num_p>13</num_p>
    <alt_mitj>834.5384615384615385</alt_mitj>
  </row>
  ...
</table>
```

## 3.5 - Consultes XPATH

També tenim funcions que ens permeten buscar dins d'un document XML. Podrem fer recerques de tipus **XPATH**. No podrem, en canvi, arribar a la potencialitat que proporciona XQUERY (amb les sentències FLWOR).

Els resultats d'una recerca XPATH es retornaran en un **array de xml**.

### XPATH

*Sintaxi:* **xpath(xpath, xml)**

*Descripció:* fa una recerca d'una expressió XPATH (primer paràmetre) en el document o contingut XML especificat en el segon paràmetre. El resultat vindrà en un array de XML. Hi pot haver un tercer paràmetre per a indicar espais de noms.

*Exemples:*

Veurem uns poquets exemples de recordatori. Aprofundirem més en la pregunta 6.2, ja que en eXist podrem fer més coses, el XPath és més complet. Recordeu que la funció **text()** torna el contingut d'un element. Si no la posàrem, ens tornaria tot l'element (amb etiqueta i tot).

- Buscar el nom de totes les comarques (les comarques les tenim en la fila 4 de **P\_XML**)

```
SELECT xpath('//nom_c/text()', (SELECT doc FROM P_XML WHERE num=4));
```

- Buscar les poblacions de la comarca Alcalatén (les poblacions les tenim en la fila 5 de **P\_XML**)

```
SELECT xpath('//poble[nom_c="Alcalatén"]/nom/text()', (SELECT doc FROM P_XML WHERE num=5));
```

- Buscar els instituts de Castelló (codi de municipi 12040) (els instituts les tenim en la fila 6 de **P\_XML**)

```
SELECT xpath('//institut[cod_m=12040]/nom/text()', (SELECT doc FROM P_XML WHERE num=6));
```

- Buscar els pobles que estan a una altura major de 1000 m.

```
SELECT xpath('//poble[altura > 1000]/nom/text()', (SELECT doc FROM P_XML WHERE num=5));
```

- Buscar tots els pobles de la Plana Alta i la Plana Baixa

```
SELECT xpath('//poble[nom_c="Plana Alta" or nom_c="Plana Baixa"]/nom/text()', (SELECT doc FROM P_XML WHERE num=5));
```



- Buscar les poblacions que tenen entre 100.000 i 300.000 habitants

```
SELECT xpath('//poble[poblacio>100000 and poblacio<300000]/nom/text()', (SELECT doc FROM P_XML WHERE num=5));
```

```
SELECT xpath('//poble[poblacio>100000][poblacio<300000]/nom/text()', (SELECT doc FROM P_XML WHERE num=5));
```

### 3.6 - Utilització des de Java

Però no hem de perdre de perspectiva que el que ens interessa és accedir des de Java als documents XML.

En principi el que podem fer és portar-nos tot el document a Java i tractar-lo com ha vam fer en el tema de fitxers: ens creem un document DOM resultat d'analitzar aquest document, i per tant ja tindrem tota l'estructura del document XML en DOM. La manera de dur tot el document és una senzilla instrucció SQL. L'única dificultat és cridar a l'analitzador (**parse**) tenint en compte que ara no ve d'un fitxer sinó d'un String (al que haurem convertit el que ve del resultat de la consulta).

En el següent exemple agafem el document que provenia de les comarques (num=5). Una vegada en Java traurem senzillament els noms. Ho podeu fer en un projecte nou anomenat **Tema8\_1**, en un paquet anomenat **Exemples**. No us oblideu de canviar la Base de Dades, usuari i contrasenya (**rx**) a la vostra:

```
import java.io.IOException;
import java.io.StringReader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;

public class PG_Prova1 {

    public static void main(String[] args) throws ClassNotFoundException, SQLException, ParserConfigurationException,
SAXException, IOException {
        String url = "jdbc:postgresql://89.36.214.106:5432/rxx";
        String usuari = "rxx";
        String password = "rxx";

        Class.forName("org.postgresql.Driver");

        Connection con = DriverManager.getConnection(url, usuari, password);

        ResultSet rs = con.createStatement().executeQuery("SELECT doc FROM P_XML WHERE num=5");

        if (rs.next()) {
            DocumentBuilder db = DocumentBuilderFactory.newInstance().newDocumentBuilder();
            Document doc = db.parse(new InputSource(new StringReader(rs.getString(1))));

            Element arrel = (Element) doc.getDocumentElement();
            NodeList llista = arrel.getElementsByTagName("nom");
```

```

        for (int i = 0; i < llista.getLength(); i++) {
            Element el = (Element) llista.item(i);
            System.out.println(el.getNodeName() + ' ' + el.getFirstChild().getNodeValue());
        }
    }
    con.close();
}

```

Amb l'anterior únicament estem utilitzant la capacitat de PostgreSQL de guardar documents XML. Però podem anar més enllà. Anem a fer que treballi un poc més PostgreSQL i ens done la informació un poc més elaborada. Farem una consulta **XPATH**. Recordem que el resultat ve en un **array de XML**. Però si considerem que la consulta ja ens torna la informació final podem agafar-lo com un **array de strings**. Recordeu que l'array no és el normal de Java, sinó el tipus de SQL. La manera que vam fer de recórrer-lo era passar-lo a un altre ResultSet

```

import java.sql.Array;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;

public class PG_Prova2 {

    public static void main(String[] args) throws SQLException, ClassNotFoundException {
        String url="jdbc:postgresql://89.36.214.106:5432/rxx";
        String usuari="rxx";
        String password="rxx";

        Class.forName("org.postgresql.Driver");

        Connection con = DriverManager.getConnection(url, usuari, password);

        ResultSet rs = con.createStatement()
            .executeQuery("SELECT xpath('//poble[nom_c=\"Plana Alta\"]/nom/text()',doc) FROM P_XML WHERE num=5");

        if (rs.next()) {
            System.out.println(rs.getString(1));
            Array pobles = rs.getArray(1);
            ResultSet rs1 = pobles.getResultSet();
            while (rs1.next())
                System.out.println(rs1.getString(2));
        }
        con.close();
    }
}

```

Hem de ser conscients que el resultat de la funció **XPATH** és un array de **xml**. Si volem només un resultat, hem d'anar en compte, perquè serà un array amb un únic element. Per exemple, si volem l'**altura** de **Vistabella**, encara que només és un resultat, ens ve en un array:

```

import java.sql.Array;

```

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;

public class PG_Prova3_1 {

    public static void main(String[] args) throws SQLException, ClassNotFoundException {
        String url="jdbc:postgresql://89.36.214.106:5432/rxx";
        String usuari="rxx";
        String password="rxx";

        Class.forName("org.postgresql.Driver");

        Connection con = DriverManager.getConnection(url, usuari, password);

        ResultSet rs = con.createStatement().executeQuery(
            "SELECT xpath('//poble[nom=\"Vistabella del Maestrat\"]/altura/text(),'doc) FROM P_XML WHERE num=5");

        if (rs.next()) {
            Array poble = rs.getArray(1);
            ResultSet rsl = poble.getResultSet();
            if (rsl.next())
                System.out.println(rsl.getString(2));
        }
        con.close();
    }
}
```

Tanmateix podem estalviar-nos el segon **ResultSet**, si elaborem un poc millor la consulta. Com que ja sabem que només serà un resultat, en la consulta podem agafar el primer element únicament. Mireu que estem obligats a tancar la funció entre parèntesis per a que funcione bé.

```
import java.sql.Array;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;

public class PG_Prova3_2 {

    public static void main(String[] args) throws SQLException, ClassNotFoundException {
        String url="jdbc:postgresql://89.36.214.106:5432/rxx";
        String usuari="rxx";
        String password="rxx";

        Class.forName("org.postgresql.Driver");

        Connection con = DriverManager.getConnection(url, usuari, password);

        ResultSet rs = con.createStatement().executeQuery(
            "SELECT (xpath('//poble[nom=\"Vistabella del Maestrat\"]/altura/text(),'doc))[1] FROM P_XML WHERE num=5");

        if (rs.next()) {
            System.out.println("Altura de Vistabella: " + rs.getString(1));
        }
    }
}
```

```

        }
        con.close();
    }
}

```

I treballant a la inversa, també podrem inserir les dades provinents de Java (per exemple agafant les dades des d'un fitxer) en la BD de PostgreSQL. Podreu agafar el fitxer **biblioteca.xml** de l'aula virtual. L'haureu de col·locar a l'arrel del projecte.

```

import java.io.FileReader;
import java.io.IOException;
import java.sql.Array;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;

public class PG_Prova4 {
    public static void main(String[] args) throws SQLException, IOException, ClassNotFoundException {
        String url="jdbc:postgresql://89.36.214.106:5432/rxx";
        String usuari="rxx";
        String password="rxx";

        Class.forName("org.postgresql.Driver");

        Connection con = DriverManager.getConnection(url,usuari,password);

        FileReader f = new FileReader("biblioteca.xml");

        String tot="";
        int c = f.read();
        while (c!=-1){
            tot += (char) c;
            c=f.read();
        }

        con.createStatement().executeUpdate("INSERT INTO P_XML VALUES(7,XMLPARSE(CONTENT ' " + tot + "'))");

        //Ja està inserit. Anem a comprovar que ha arribat de forma correcta
        ResultSet rs = con.createStatement().executeQuery("SELECT xpath('/titol/text()',doc) FROM P_XML WHERE num=7");

        while (rs.next()){
            Array llibres = rs.getArray(1);
            ResultSet rs1 = llibres.getResultSet();
            while (rs1.next())
                System.out.println(rs1.getString(2));
        }
        f.close();
        con.close();
    }
}

```



### 3.7 - Ampliació

Amb les nocions anteriors tenim el suficient per a treballar prou còmodament documents XML en PostgreSQL.

Es pot aprofundir molt més per a arribar a resultats més que acceptables. Tanmateix s'escapa de l'objectiu del present curs.

Únicament a mode d'indicador de la possible potència es mostra algun exemple.

En el següent exemple, i per mig d'una única sentència, s'aconsegueix crear un document XML amb les comarques i dins de cada comarca tots els seus pobles. Evidentment, la sentència ha de quedar un poc complicada. Aprofitarem per a inserir en la nostra taula de documents: P\_XML

```
INSERT INTO P_XML VALUES ( 8 ,
    (SELECT xmlelement(name comarques,
        (SELECT xmlagg(xmlelement(name comarca,xmldata(xmlforest(nom_c,provincia)),
            xmlelement(name pobles, (SELECT xmlagg(xmlelement(name poble,xmldata(xmlforest(nom,altura)))
                FROM POBLACIONS
                WHERE nom_c = COMARQUES.nom_c ) ) ) )
        FROM COMARQUES))) );
```

Evidentment és complicada, però en una única sentència tenim la creació del document XML, a partir de les taules COMARQUES i POBLACIONS. El resultat serà aquest:

```
<comarques>
  <comarca>
    <nom>Safor</nom>
    <provincia>València</provincia>
    <pobles>
      <poble>
        <nom>Alfauir</nom>
        <altura>75</altura>
      </poble>
      <poble>
        <nom>Almiserà</nom>
        <altura>75</altura>
      </poble>
      <poble>
        <nom>Beniflà</nom>
        <altura>50</altura>
      </poble>
      ...
    </pobles>
  </comarca>
  ...
</comarques>
```

Des de pgAdmin no el veurem tabulat així. Per poder veure el resultat, podem "exportar" l'execució d'una sentència SQL que traga només aquest document, de manera que l'execució

vaja cap a un fitxer.

Es deixa com a exercici totalment voluntari l'ampliació de la sentència anterior per a que incorpore també tots els instituts de cada poble.

Amb aquest document que és un poc més complicat, ja que incorpora les comarques i dins d'elles els pobles, podríem fer una consulta XPATH que ens diga les comarques que tenen més de 30 pobles:

```
SELECT xpath('//comarca[count(pobles/poble)>=30]/nom_c/text()', (SELECT doc FROM P_XML WHERE num=8));
```



## 4 - Emmagatzematge en BD-XML Natives

En les **Bases de Dades XML natives**, a diferència de tots els SGBD anteriors (centrats en les dades), no existeixen camps ni guarden dades en tipus de dades senzills. Sempre guarden documents XML i són Bases de Dades centrades en els documents. La unitat mínima d'emmagatzematge és el document XML.

Podríem definir **SGBD-XML natives** com un sistema de gestió de la informació que ha de:

- Definir un model lògic per a un document XML (en contraposició als SGBD habilitats que defineixen el model per les dades) i enregistrar i recuperar els documents segons aquest model. Com a mínim, el model ha d'incloure elements, atributs, PCDATA i l'ordre del document.
- Mantenir una relació transparent amb el mecanisme subjacent d'emmagatzematge, incorporant les característiques ACID de qualsevol SGBD (*Atomicity, Consistency, Isolation and Durability*).
- Incloure un nombre arbitrari de nivells de dades i complexitat.
- Permetre les tecnologies de consulta i transformació pròpies d'XML: **XPath**, **XSLT**, **XQL**, **XQuery**, etc.
- Permetre la introducció d'informació *data-centric*, *document-centric* i mixta.

El segon punt de la definició porta implícita la idea que un SGBD-XML natives no té per què tenir cap model físic d'emmagatzematge subjacent específic, sinó que es podria construir damunt un SGBDR, un SGBDOO, ..., o un sistema propi d'emmagatzematge. Això sembla contradictori amb els problemes i crítiques que presenten per a la gestió de documents XML els SGBD predecessors dels SGBD-XML natives.

Avantatges de les BD-XML natives, en comparació a les BD no XML:

- Faciliten accés i emmagatzematge d'informació en format XML sense necessitat de codi addicional ni cap tipus de mapatge.
- La majoria d'SGBD-XML natives incorporen un motor de recerca d'alt rendiment.
- És molt senzill afegir nous documents XML.
- Permeten emmagatzemar dades heterogènies.
- Conserven la integritat dels documents (es poden recuperar en el seu estat inicial).

Per contra, els inconvenients que tenen normalment els SGBD-XML natives són aquests:

- La gran quantitat d'espai necessari per emmagatzemar el mateix document XML com a format de representació de la informació, a causa del fet que les etiquetes poden suposar el 75% de la informació d'un document XML. I això és, sense cap mena de dubte, innecessari en guardar molts documents validats per un mateix XSD o DTD.
- El fet que les BD-XML natives només puguin guardar i retornar dades en format XML.
- En emmagatzemar la informació en format XML es fa molt complicat poder generar noves estructures a partir de la informació existent com, per exemple, aconseguir càlculs estadístics.
- Les dificultats d'indexació del contingut d'una base de dades, que ha de permetre la reducció dràstica del temps necessari per trobar certs elements clau.
- Les pobres facilitats per modificar el contingut dels documents XML emmagatzemats sense haver de substituir tot el document.

Els dos darrers inconvenients (indexació-actualització) són cavalls de batalla dels SGBD i de ben segur que s'anirà avançant en aquest camp fins que deixen de ser inconvenients.

## 5 - Instal·lació de BD-XML Natives

---

Triarem el Sistema Gestor de Bases de Dades XML Natives **eXist-db**. Com veurem és molt potent i còmode d'utilitzar. A més és Open-Source. Està escrit completament en Java i suporta els estàndards de consulta **XPATH**, **XQuery** i **XSLT**.

Això sí, és un servidor, per tant l'hurem d'instal·lar, posar-lo en marxa quan l'utilitzem, i fer un poc d'administradors.

Afortunadament està construït en Java, i per tant no hem de tenir cap problema per a instal·lar-lo. Ens el podem baixar sense dificultat de la pàgina oficial: [exist-db.org](http://exist-db.org)

El següent vídeo mostra tot el procés d'instal·lació i arrancada inicial. És per a una versió anterior, la 2.1, però la instal·lació és absolutament similar per a la versió 4.5.0 que és l'estable en el moment de fer aquestos apunts.

Els documents es guarden en col·leccions, les quals es poden crear unes dins d'unes altres.

Anem a veure uns exemples d'utilització. Crearem una col·lecció, i en ella inclourem algun documents XML. Posteriorment farem alguna consulta senzilla, a veure com es comporta.

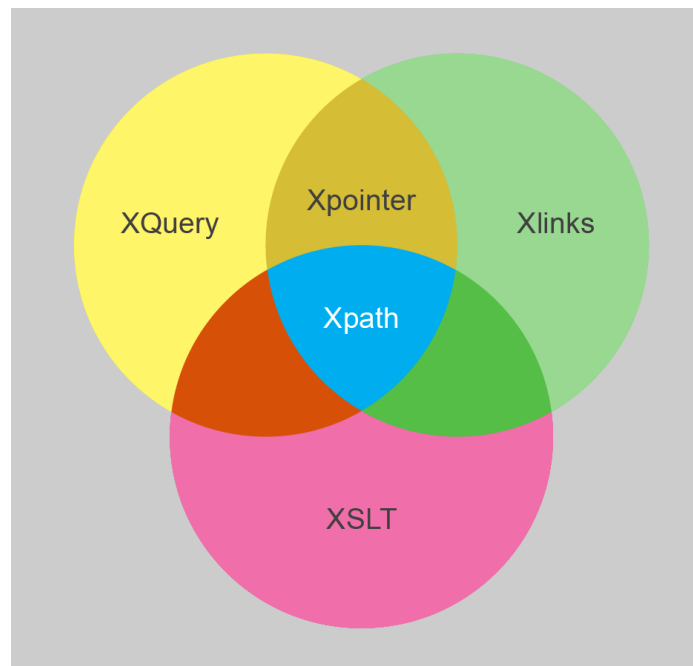
Recordeu que els resultats per defecte es veuen de 10 en 10, és a dir, que poden haver més resultats dels mostrats en principi.



## 6 - Llenguatges de consultes XPATH i XQUERY

Aquesta pregunta pretén ser un repàs del vist en el mòdul de primer **Llenguatges de Marques**. Veurem únicament **XPath** i **XQuery**, d'entre el ventall de llenguatges XML recomanats pel W3C (*World Wide Web Consortium*).

Aquests llenguatges no són separats, sinó que estan interrelacionats entre ells. Concretament la relació entre **XPath** i **XQuery** és que XPath és un subconjunt de XQuery, o millor dit totes les expressions XPath són vàlides en XQuery.



Com veiem en la figura, **XPath** s'ha convertit en un component essencial per a diferents llenguatges XML com **XLink**, **XSLT** i el que més ens interessa, **XQuery**.

## 6.1 - XPATH

---

**XPath** (*XML Path Language*) és un llenguatge capaç d'especificar parts d'un document XML, i té eines per a manipular el contingut de les dades de text, numèriques, etc.

**XPath** és una recomanació del W3C ([www.w3.org/TR/xpath](http://www.w3.org/TR/xpath)), que encara que serveix per treballar amb XML, no és un llenguatge XML. La idea és que en no estar basat en XML es podrà incloure en altres llenguatges XML sense haver de preocupar-se de si el resultat està ben format o no.

La base del funcionament de XPath és l'avaluació d'expressions on constarà el camí (path) del que volem buscar, és a dir, una expressió que s'avaluarà contra un document XML i ens donarà un resultat que pot ser de diferents tipus:

- Un booleà: cert o fals
- Un número
- Una cadena de caràcters
- Un grup d'elements

La versió 2.0 de **XPath** està tan integrada dins de **XQuery** que qualsevol expressió XPath és també automàticament una expressió XQuery correcta.



## 6.1.1 - Vista d'arbre

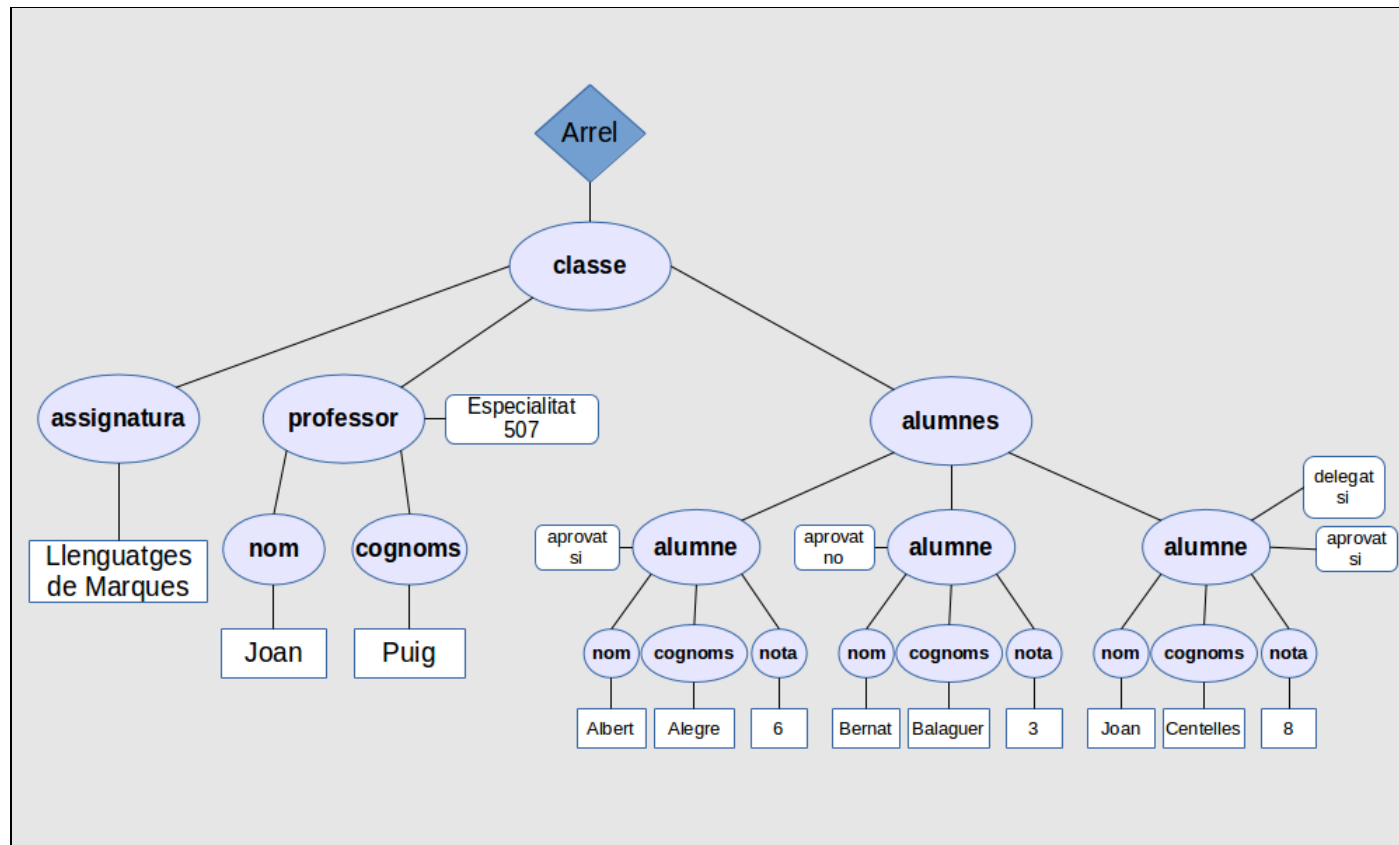
**XPath** tracta tots els documents XML des del punt de vista d'un arbre de nodes en el qual hi ha **una arrel** que no es correspon amb l'arrel del document, sinó que **és el propi document** (i per tant l'immediatament anterior a l'element arrel del document). Es representa amb el símbol **" / "**.

A part de l'arrel també hi ha nodes per representar els **elements**, els **atributs**, els nodes de **dades**, els **comentaris**, les **instruccions de procés** i els **espais de noms**.

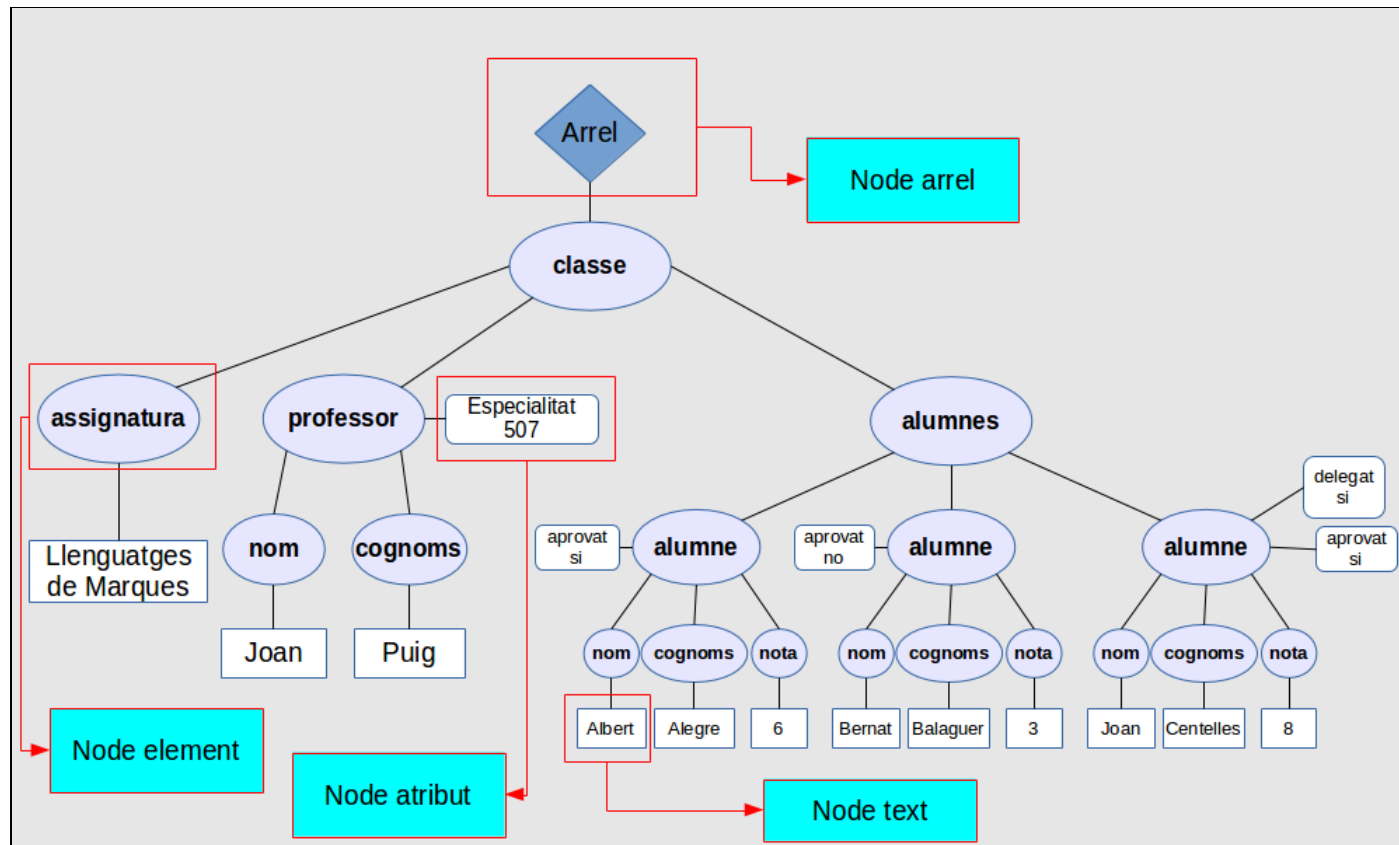
El següent exemple XML (que el podeu copiar en un document **nou** de tipus **XML** anomenat **classe.xml**; cuideu que siga de tipus **xml**):

```
<?xml version="1.0" encoding="UTF-8" ?>
<classe>
  <assignatura>Il·lenguatges de Marques</assignatura>
  <professor Especialitat="507">
    <nom>Joan</nom>
    <cognoms>Puig</cognoms>
  </professor>
  <alumnes>
    <alumne aprovat="si">
      <nom>Albert</nom>
      <cognoms>Alegre</cognoms>
      <nota>6</nota>
    </alumne>
    <alumne aprovat="no">
      <nom>Bernat</nom>
      <cognoms>Balaguer</cognoms>
      <nota>3</nota>
    </alumne>
    <alumne delegat="si" aprovat="si">
      <nom>Joan</nom>
      <cognoms>Centelles</cognoms>
      <nota>8</nota>
    </alumne>
  </alumnes>
</classe>
```

es representarà en XPath amb un arbre com el de la figura:



on els diferents tipus de nodes que trobem són els assenyalats a continuació:



En un arbre XPath els atributs no són considerats nodes fills sinó que són “**propietats**” del node que els conté i els nodes de dades són **nodes sense nom** que només contenen les dades.

## 6.1.2 - Navegació

---

Com que la representació interna del document XML per XPath serà un arbre, es pot navegar per ell especificant camins d'una manera semblant a com es fa en els directoris dels sistemes operatius.

El més important per tenir en compte a l'hora de crear una expressió XPath és saber quin és el node en el qual està situat el procés (**node de context**), ja que és des d'aquest que s'avaluarà l'expressió. El node de context al principi és el node arrel però es va movent a mesura que es van avaluant les expressions, i per tant podem expressar els camins XPath de dues maneres:

- **Camins absoluts**
- **Camins relatius**

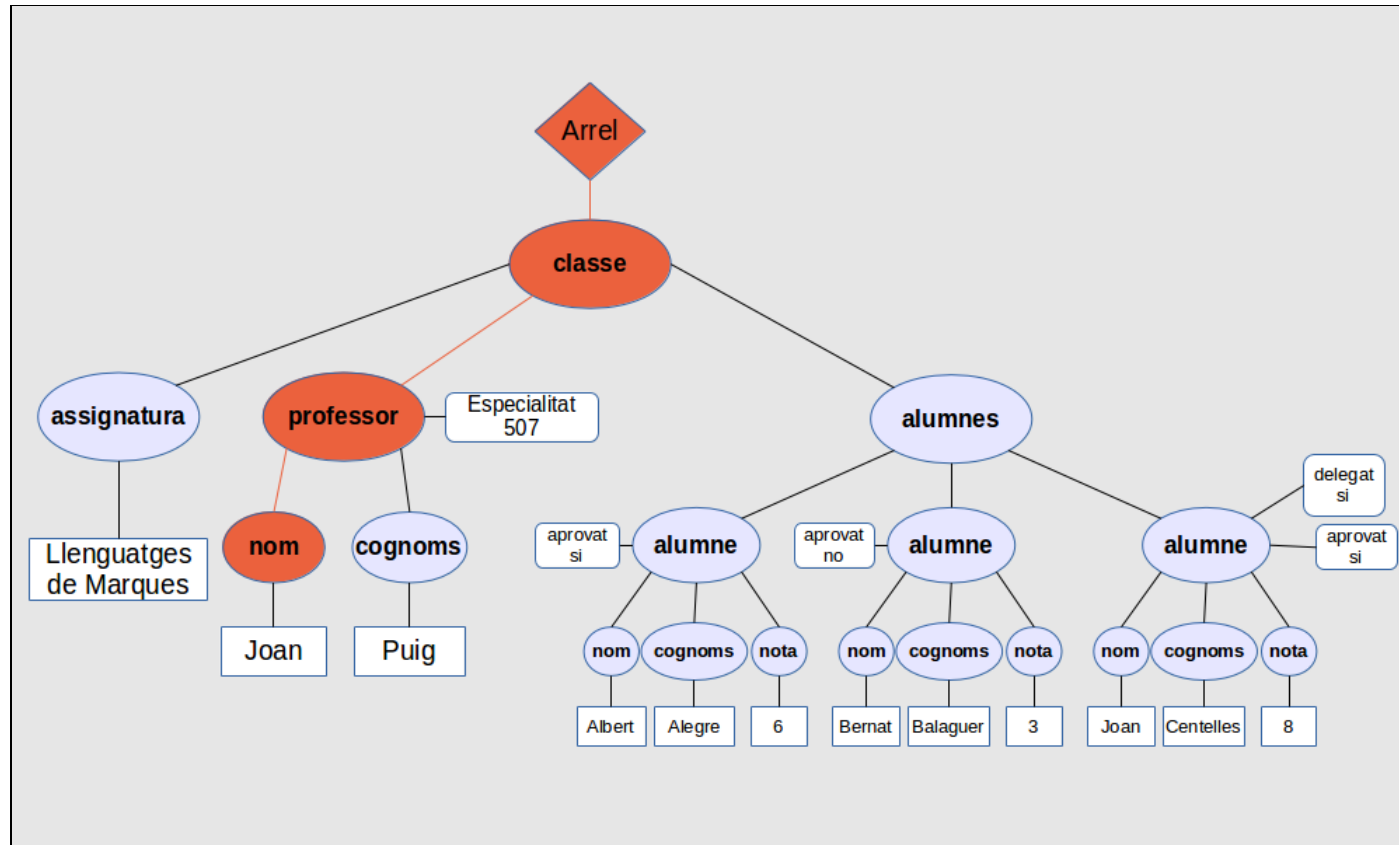
Els **camins absoluts** són camins que sempre comencen en l'arrel de l'arbre. Es poden identificar perquè el primer caràcter de l'expressió sempre serà l'arrel `"/"`. No importa quin siga el node de context si es fan servir camins absoluts, perquè el resultat sempre serà el mateix.

En canvi, els **camins relatius** parteixen des del node en el qual estem situats.

Per exemple, es pot obtenir el node **<nom>** del professor de l'exemple anterior fent servir l'expressió XPath següent:

```
/classe/professor/nom
```

Podem veure com s'avalua l'expressió en l'arbre XPath en la següent figura, marcat el camí en taronja:



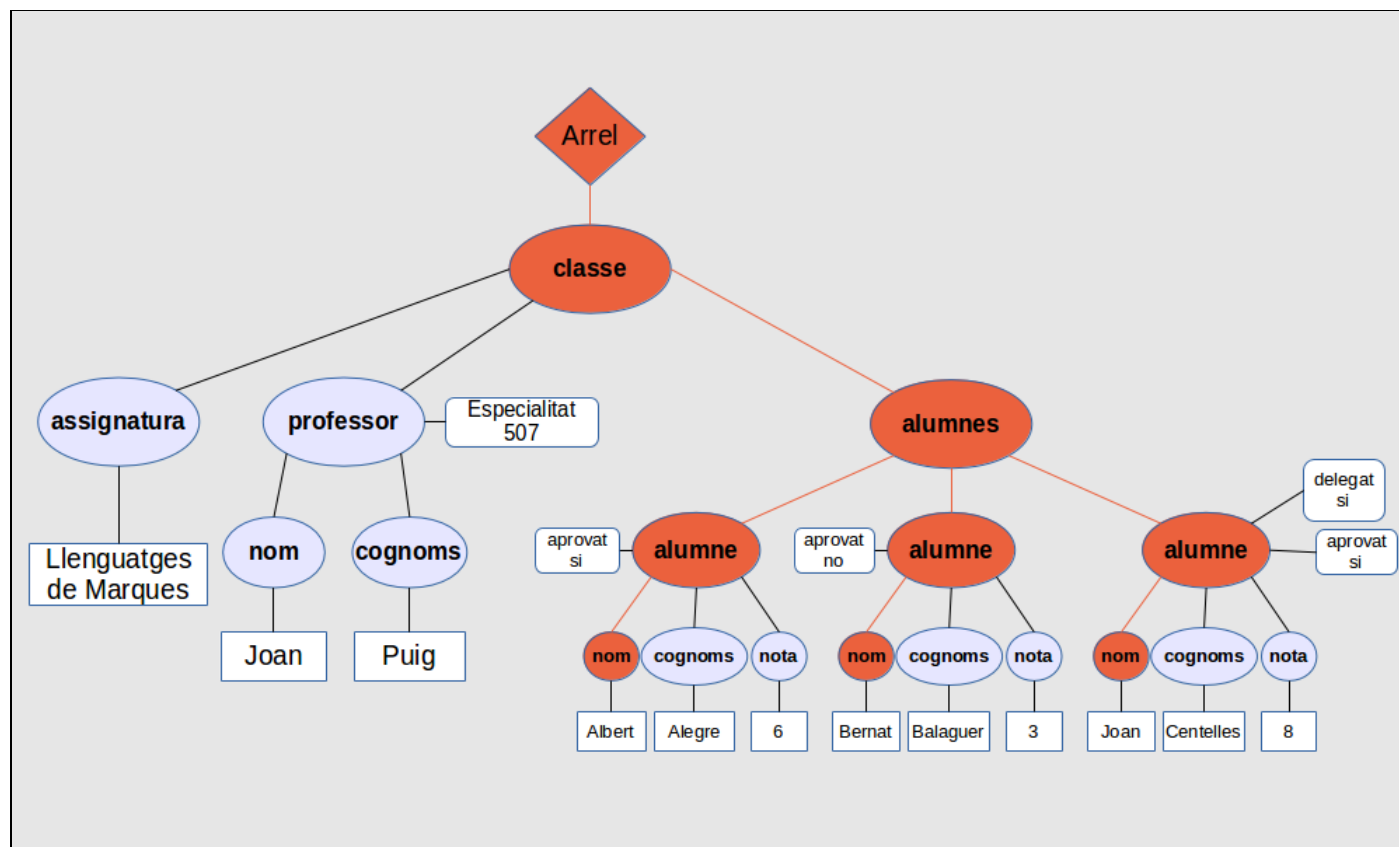
Observeu que el resultat d'aquesta expressió no és només el contingut de l'element sinó tot l'element **<nom>**.

```
<nom>Joan</nom>
```

Mai no s'ha d'oblidar que l'expressió sempre intenta aconseguir el nombre màxim de camins correctes i, per tant, no necessàriament ha de retornar només un únic valor. Per exemple, si l'expressió per avaluar és la següent:

```
/classe/alumnes/alumne/nom
```

XPath l'avaluaria intentant aconseguir tots els camins que quadren amb l'expressió, tal com podeu veure visualment en la següent figura:



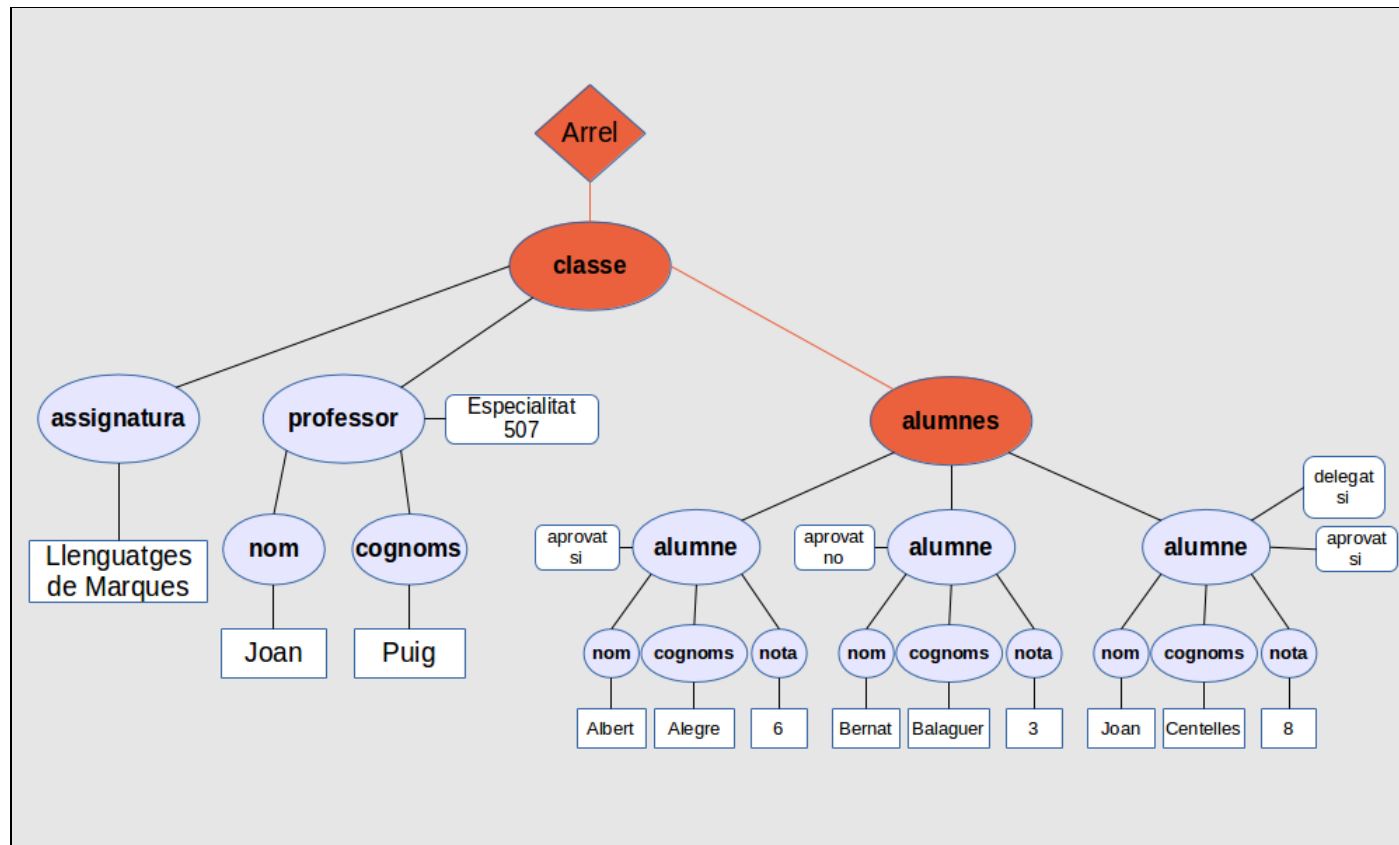
i per tant els resultats són (observeu que Joan es refereix al nom del tercer alumne, no del professor):

```
<nom>Albert</nom>
<nom>Bernat</nom>
<nom>Joan</nom>
```

A pesar que en els exemples anteriors sempre s'han retornat nodes finals això no necessàriament ha de ser així, ja que XPath pot retornar qualsevol tipus d'element com a resultat.

```
/classe/alumnes
```

La navegació per l'arbre no difereix massa dels altres casos:



En aquest cas el resultat no és un element simple sinó que és tot un subarbre d'elements.

```
<alumnes>
  <alumne aprovat="si">
    <nom>Albert</nom>
    <cognoms>Alegre</cognoms>
    <nota>6</nota>
  </alumne>
  <alumne aprovat="no">
    <nom>Bernat</nom>
    <cognoms>Balaguer</cognoms>
    <nota>3</nota>
  </alumne>
  <alumne delegat="si" aprovat="si">
    <nom>Joan</nom>
    <cognoms>Centelles</cognoms>
  </alumne>
</alumnes>
```

```
<nota>8</nota>
</alumne>
</alumnes>
```

Si se sap que una expressió retornarà més d'un resultat però només se'n vol un d'específic, es pot fer servir un número envoltat per claudàtors quadrats "[ ]" per indicar quin és el que es vol aconseguir. Per retornar només el primer alumne podeu fer el següent:

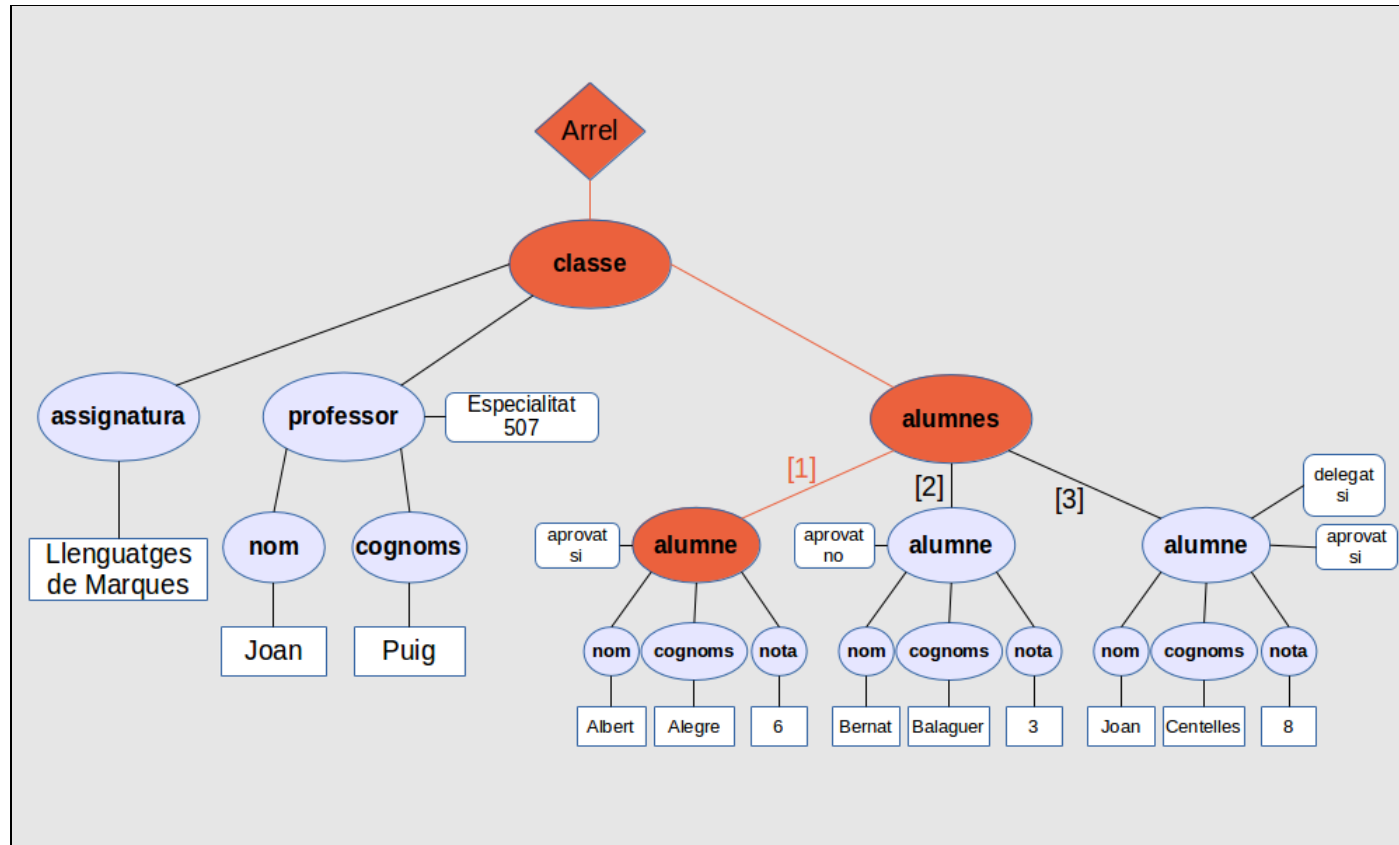
```
/classe/alumnes/alumne[1]
```

que tornarà el següent:

```
<alumne aprovat="si">
  <nom>Albert</nom>
  <cognoms>Alegre</cognoms>
  <nota>6</nota>
</alumne>
```

ja que dels tres nodes disponibles com a fills de **<alumnes>**, només se seleccionarà el primer:





Es poden fer servir els claudàtors en qualsevol lloc de l'expressió per fer determinar quina de les branques es triarà. Per exemple, es pot obtenir només el **nom** del segon alumne amb una expressió com aquesta:

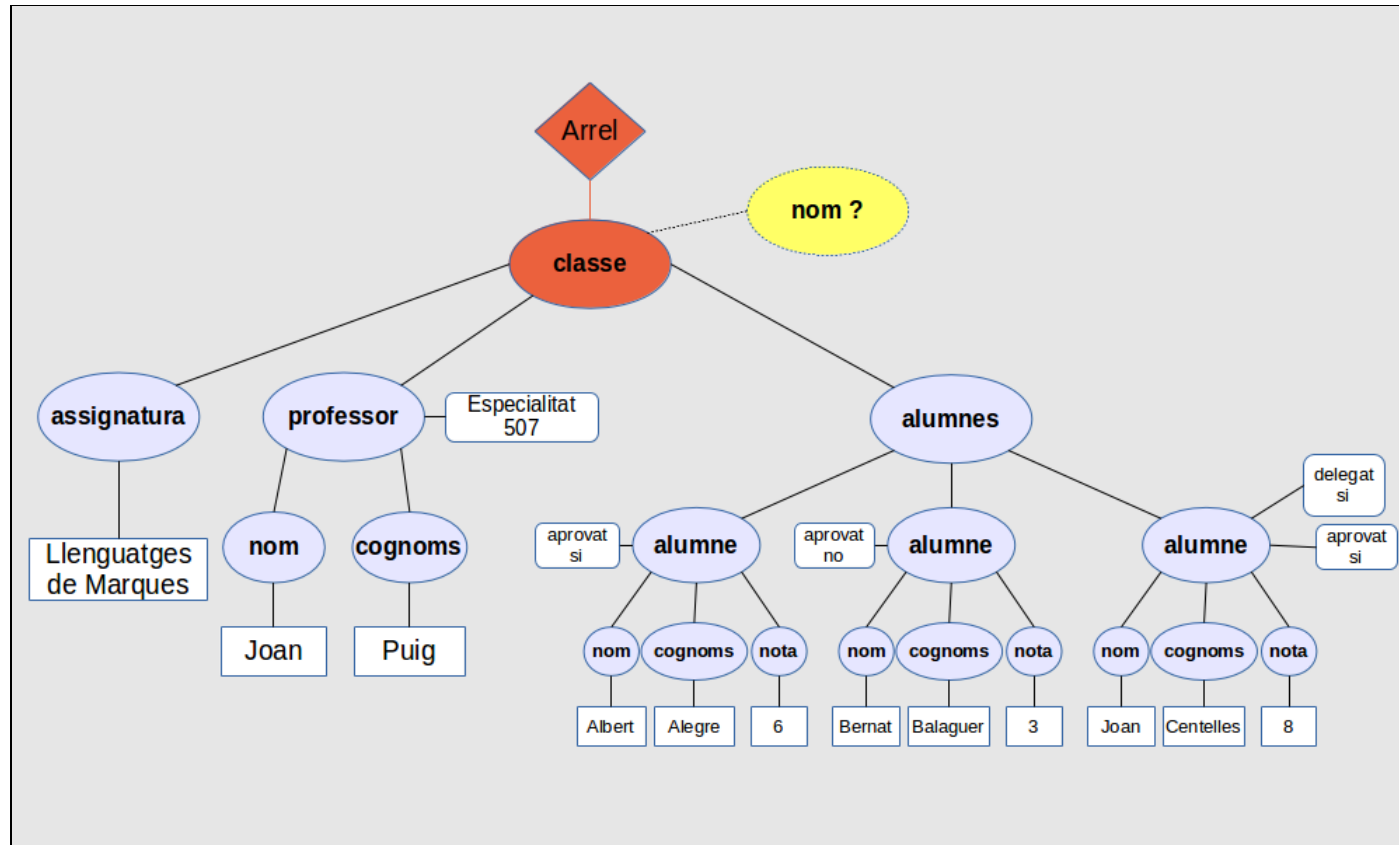
```
/classe/alumnes/alumne[2]/nom
```

```
<nom>Bernat</nom>
```

Sempre s'ha d'anar amb compte en escriure les expressions XPath, ja que si el camí especificat no es correspon amb un camí **real** dins de l'arbre no es retornarà cap resultat.

```
/classe/nom
```

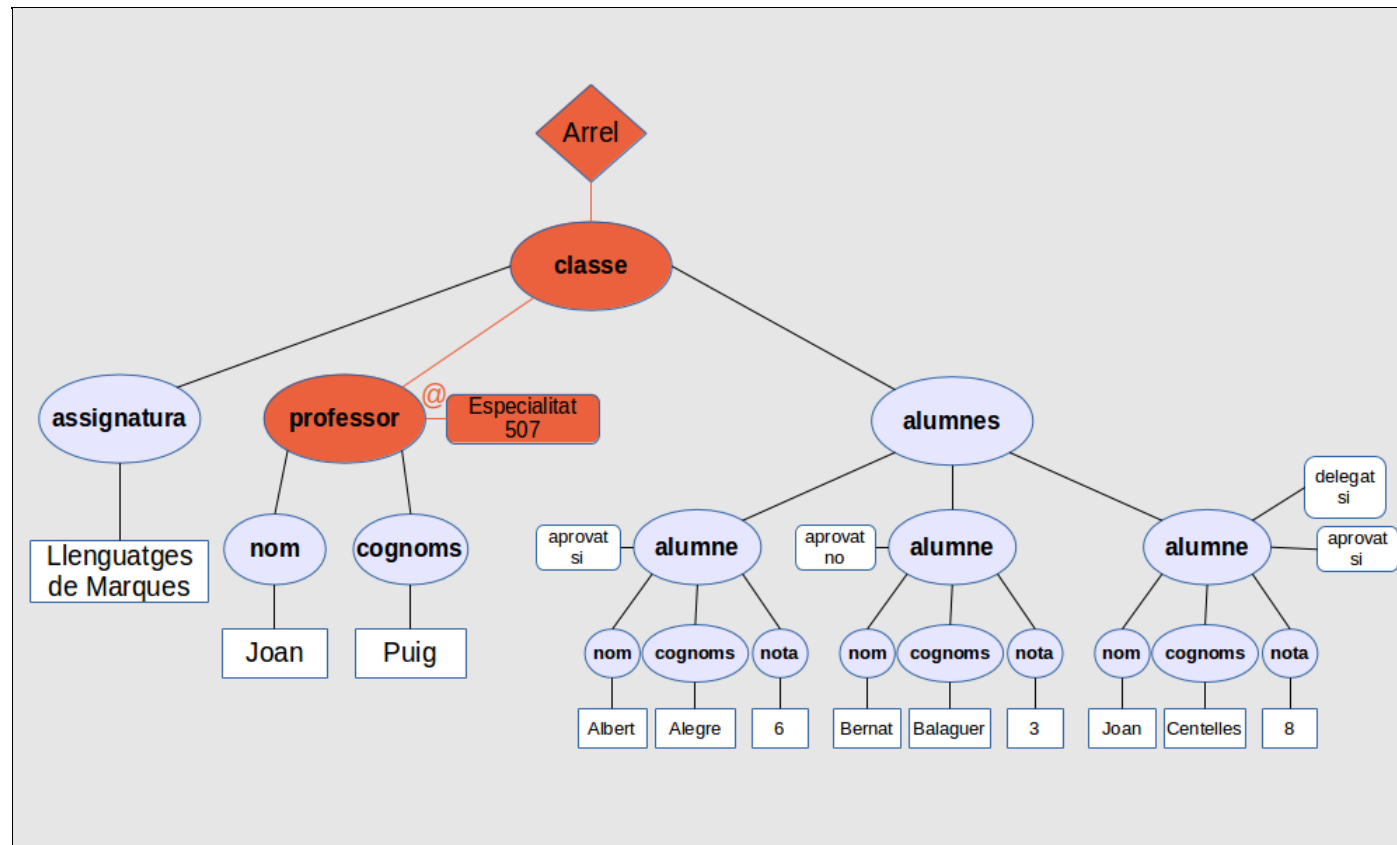
Com que en arribar al node **classe** no n'hi trobarà cap d'anomenat **<nom>**, no retornarà cap resultat, com podeu veure si seguiu l'arbre:



### Obtenir els atributs d'un element

Els valors dels atributs es poden aconseguir especificant el símbol **@** davant del nom de l'atribut, quan s'haja arribat a l'element que el conté:

```
/classe/professor/@Especialitat
```



S'ha de tenir en compte que a diferència del que passa amb els elements, en obtenir un atribut no tindrem un element sinó només el seu valor:

507

**Nota**

Depenent de l'entorn en què ens trobem, podria ser que l'anterior no ens mostre res. La raó és perquè els atributs no es poden serialitzar, i per tant no es poden mostrar en un entorn de XML. No tindríem aquest problema si accedim des de Java. La manera de solucionar-lo des de l'entorn de eXist és utilitzar la funció **data()**:

```
/classe/professor/data (@Especialitat)
```

## Obtenir el contingut d'un element

Per a aquells casos en què només vulguem el contingut de l'element, s'ha definit la funció **text()** per obtenir aquest contingut. Això s'ha fet així perquè d'altra manera, com que els nodes de text no tenen nom, no s'hi podria accedir.

De manera que si a un element que tinga contingut de dades se li afegeix **text()**:

```
/classe/professor/nom/text()
```

...retornarà el contingut del node sense les etiquetes:

```
Joan
```

## Comodins

De la mateixa manera que en els sistemes operatius, es poden fer servir comodins diversos en les expressions XPath. Es poden veure els comodins en la taula següent.

Comodi	Significat
*	L'asterisc es fa servir per indicar tots els elements d'un determinat nivell.
.	Com en els directoris dels sistemes operatius el punt serveix per indicar el node actual.
..	Es fa servir per indicar el pare del node en el qual estem.
//	La doble barra indica que quadrarà amb qualsevol cosa des del node en el qual estem, es a dir que buscarà en tots els seus descendents. Pot ser un sol element o un arbre de nodes.

Amb l'asterisc es poden obtenir tots els elements d'un determinat nivell. Amb aquesta expressió es poden obtenir tots els elements de dins del node **professor**.

```
/classe/professor/*
```

Aquesta expressió retornarà per separat els dos nodes fills de **<professor>** , **<nom>** i **<cognom>**.

```
<nom>Joan</nom>  
<cognoms>Puig</cognoms>
```

O bé fer servir les dobles barres (//) per obtenir tots els elements **<nom>** del fitxer independentment del lloc on estiguen dins del document XML.

```
//nom
```

El resultat serà:

```
<nom>Joan</nom>  
<nom>Albert</nom>  
<nom>Bernat</nom>  
<nom>Joan</nom>
```

Observeu que apareix el nom Joan 2 vegades, ja que és tant el nom del professor com d'un dels alumnes

#### **Nota**

En realitat aquesta sentència XPath tornarà molts més resultats, ja que buscarà en tots els documents de la col·lecció, i en **Rutes.xml** hi ha molts elements **nom**: els noms de les rutes i els noms dels punts, a banda dels noms de professors i d'alumnes.

Es poden posar les dobles barres en qualsevol lloc dins de l'expressió per indicar que pot haver qualsevol cosa enmig a partir del lloc on apareguen.

```
/classe/alumnes//nom
```

Donarà els dos noms dels alumnes (ara sí només alumnes, ni professors ni noms de ruta ni de punts):

```
<nom>Albert</nom>  
<nom>Bernat</nom>  
<nom>Joan</nom>
```

Tot i que facilita molt la creació d'expressions no és molt recomanable abusar del comodí // per motius d'eficiència. Les expressions amb aquest comodí requeriran molts més càlculs per ser avaluades, i per tant les expressions tardaran més a donar resultats.

## **Eixos XPATH**

Per avaluar les expressions XPath s'explora un arbre, de manera que també es proporcionen una sèrie d'elements per fer referència a parts de l'arbre. Aquests elements s'anomenen **eixos XPath**

Eix	Significat
<b>self::</b>	El node en el qual està el procés (fa el mateix que el punt)
<b>child::</b>	Fill de l'element actual
<b>parent::</b>	El pare de l'element actual (idèntic a fer servir ..)
<b>attribute::</b>	Es fa servir per obtenir un atribut de l'element actual (@)

Alguns d'aquestos eixos pràcticament no es fan servir perquè generalment és més còmode i curt definir les expressions a partir del símbol. Tothom prefereix fer servir una expressió com aquesta:

[/classe/professor/nom](#)

Que no la seua versió equivalent fent servir els eixos:

[/child::classe/child::professor/child::nom](#)

A part dels vistos en la taula anterior n'hi ha d'altres, que en aquest cas no tenen cap símbol que els simplifiqui:

Eix	Significat
<b>descendant::</b>	Tots els descendents del node actual
<b>desdendant-or-self::</b>	El node actual i els seus descendents
<b>ancestor::</b>	Els ascendents del node
<b>ancestor-or-self::</b>	El node actual i els seus ascendents
<b>preceding::</b>	Tots els elements precedents al node actual
<b>preceding-sibling::</b>	Tots els germans precedents
<b>following::</b>	Elements que segueixen el node actual
<b>following-sibling::</b>	Germans posteriors al node actual
<b>namespace::</b>	Conté l'espai de noms del node actual

## Condicions

Un apartat interessant de les expressions XPath és poder afegir condicions per a la selecció de nodes. A qualsevol expressió XPath se li poden afegir condicions per obtenir només els nodes que compleixen la condició especificada.

Per exemple, aquesta expressió selecciona només els professors que tinguen un element **<nom>** com a fill de **<professor>**:

```
/classe/professor[nom]
```

Si s'aplica l'expressió a l'exemple que hem fet servir per fer la vista d'arbre, el resultat serà el node **<professor>** que té dins seu **<nom>** (i ens apareixerà l'únic professor que tenim, ja que sí que té un fill nom)

```
<professor Especialitat="507">
  <nom>Joan</nom>
  <cognoms>Puig</cognoms>
</professor>
```

En el valor de l'expressió s'hi especifiquen camins relatius des del node que tinga la condició. Fent servir condicions es pot fer una expressió que només retorne el professor si té alumnes (i ens tornaria el mateix resultat que abans, ja que sí que té alumnes)

```
/classe/professor[../alumnes/alumne]
```

Normalment la complexitat de les condicions va més enllà de comprovar si el node existeix, i es fan servir per comprovar si un node té un valor determinat. Per exemple, per obtenir els alumnes que es diguen "Bernat":

```
/classe/alumnes/alumne[nom="Bernat"]
```

que donarà:

```
<alumne aprovat="no">
  <nom>Bernat</nom>
  <cognoms>Balaguer</cognoms>
  <nota>3</nota>
</alumne>
```

Les condicions es poden posar en qualsevol lloc del camí i n'hi pot haver tantes com calga. Per obtenir el cognom del professor que es diu "Bernat" es pot fer servir una expressió com aquesta.

```
/classe//alumne[nom="Bernat"]/cognoms
```

Que donarà de resultat:

```
<cognoms>Balaguer</cognoms>
```

De la mateixa manera que per obtenir-ne els valors, es poden fer comparacions amb els valors dels atributs especificant el seu nom després del símbol @. Per saber si un element té l'atribut 'delegat':

```
/classe//alumne[@delegat]
```

Retornarà:

```
<alumne delegat="si" aprovat="si">
  <nom>Joan</nom>
  <cognoms>Centelles</cognoms>
  <nota>8</nota>
</alumne>
```

mentre que en la mateixa sentència ens haguérem preguntat per l'atribut aprovat, ens apareixerien els 3 alumnes, ja que tots tres el tenen.

De la mateixa manera que amb els elements, es poden posar condicions als atributs per saber si el seu valor té un determinat valor, etc.

```
/classe//alumne[@aprovat="si"]
```

que ens donaria els dos alumnes que estan aprovats

```
<alumne aprovat="si">
  <nom>Albert</nom>
  <cognoms>Alegre</cognoms>
  <nota>6</nota>
</alumne>
<alumne delegat="si" aprovat="si">
  <nom>Joan</nom>
  <cognoms>Centelles</cognoms>
  <nota>8</nota>
</alumne>
```

Les condicions també poden ser de desigualtat:

```
/classe/professor[@Especialitat>=507]
```

La funció **not()** es fa servir per negar les condicions:

```
/classe//alumne[not(@delegat)]
```

```
<alumne aprovat="si">
  <nom>Albert</nom>
  <cognoms>Alegre</cognoms>
  <nota>6</nota>
</alumne>
<alumne aprovat="no">
  <nom>Bernat</nom>
  <cognoms>Balaguer</cognoms>
  <nota>3</nota>
</alumne>
```

Es poden mesclar les expressions amb condicions sobre atributs i sobre elements per aconseguir expressions més complexes especificant-les una al costat de l'altra. Per exemple, podem obtenir l'alumne que té no té l'atribut delegat i que està aprovat:



```
/classe//alumne[@delegat][@aprovat="si"]
```

```
<alumne delegat="si" aprovat="si">
  <nom>Joan</nom>
  <cognoms>Centelles</cognoms>
  <nota>8</nota>
</alumne>
```

L'expressió pot ser tan complexa com calga. Per exemple es pot obtenir la llista dels cognoms dels alumnes del professor de l'especialitat "507" que es diu "Joan":

```
/classe/professor[@Especialitat="507"][nom="Joan"]/../alumnes/alumne/cognoms
```

Que retornarà els tres cognoms dels alumnes:

```
<cognoms>Alegre</cognoms>
<cognoms>Balaguer</cognoms>
<cognoms>Centelles</cognoms>
```

## Documents

Si no especifiquem res, a banda de la ruta, **eXist** buscarà en **tots** els documents de la col·lecció en la qual estiguem situats.

Per a buscar únicament en un document, l'hauréu d'especificar al principi de la ruta. I la manera d'especificar el document serà: **doc(ruta\_del\_document)**. Si posem únicament el nom del document no el trobarà, i estarem obligats a posar tota la seua ruta que serà: **/db/nom\_col/nom\_doc**. És a dir, el següent no troba res:

```
doc("classe.xml")
```

La manera d'accedir serà:

```
doc("/db/Tema8/classe.xml")
```

En altres entorns potser sí que funcione posar únicament el nom del document, però en el que estem treballant nosaltres, hauréu de posar tota la ruta del document en la Base de Dades.

D'aquesta manera podrem solucionar el problema de la següent sentència, que agafa tots els noms d'alumnes i de professors, però també de rutes i de punts:

```
//nom
```

que donarà un total de 32 resultats

Si només els volem d'un document:

```
doc("db/Tema8/classe.xml")//nom
```

```
<nom>Joan</nom>  
<nom>Albert</nom>  
<nom>Bernat</nom>  
<nom>Joan</nom>
```

## 6.1.3 - Seqüències

Una seqüència és una expressió XPath que retorna més d'un element. S'assemblen prou a les llistes d'altres llenguatges:

- Tenen ordre
- Permeten duplicats
- Poden contenir valors de tipus diferent en cada terme

És fàcil crear seqüències, ja que només cal tancar-les entre parèntesis i separar cadascun dels termes amb comes. L'expressió següent aplicada a qualsevol document:

```
(1,2,3,4)
```

Retorna la seqüència de nombres d'un en un:

```
1  
2  
3  
4
```

També es poden crear seqüències a partir d'expressions XPath. En aquest cas s'avaluarà primer la primera expressió, després la segona, etc.

```
(//alumnes//nom/text() , //cognoms/text())
```

Aplicat al nostre exemple retornarà primer tots els noms dels alumnes i després tots els cognoms (de professor i alumnes):

```
Albert  
Bernat  
Joan  
Puig  
Alegre  
Balaguer  
Centelles
```

### Unió, Intersecció i disjunció

També es pot operar amb les seqüències d'elements amb els operadors d'unió (**union**), intersecció (**intersec**) o disjunció (**except**). El resultat serà una altra seqüència on **no** hi haurà elements duplicats.

Anem a utilitzar com a exemple dues seqüències relativament fàcils de construir:

- el nom dels alumnes aprovats (Albert i Joan): //alumne[@aprovat="si"]/nom
- el nom dels alumnes que són delegats (Joan): //alumne[@delegat]/nom

La intersecció ens tornaria els elements que estan en les dues llistes, és a dir

```
//alumne[@aprovat="si"]/nom intersect //alumne[@delegat]/nom
```

```
<nom>Joan</nom>
```

Amb la unió es poden unir les llistes de manera que quede una sola sense duplicats:

```
//alumne[@aprovat="si"]/nom union //alumne[@delegat]/nom
```

```
<nom>Albert</nom>
```

```
<nom>Joan</nom>
```

I amb la disjunció obtenim els noms de la primera seqüència que no apareixen en la segona:

```
//alumne[@aprovat="si"]/nom except //alumne[@delegat]/nom
```

```
<nom>Albert</nom>
```

### **Nota important**

En les seqüències apareixen els **elements** que són **distints**, encara que els seu contingut siga el mateix. Per exemple, el nom del professor és Joan, igual que un dels alumnes. A pesar de tenir el mateix valor, són elements diferents, nodes diferents. Així, en contra del que cabria esperar en principi, si fem la intersecció entre els noms dels alumnes i dels professors, no eixirà cap element .

```
//alumne/nom intersect //professor/nom
```

I si fem la unió de les seqüències anteriors, el nom Joan ens apareixarà 2 vegades, senyal que són elements diferents:

```
//alumne/nom union //professor/nom
```

```
<nom>Joan</nom>
<nom>Albert</nom>
<nom>Bernat</nom>
<nom>Joan</nom>
```

## 6.1.4 - Funcions

XPath ofereix una gran quantitat de funcions destinades a manipular els resultats obtinguts.

Aquestes funcions es classifiquen en diferents grups:

- **Per manipular conjunts de nodes:** es pot obtenir el nom dels nodes, treballar amb les posicions, comptar-los, etc.
- **Per treballar amb cadenes de caràcters:** permeten extreure caràcters, concatenar, comparar... les cadenes de caràcters.
- **Per fer operacions numèriques:** es poden convertir els resultats a valors numèrics, comptar els nodes, fer operacions, etc.
- **Funcions booleanes:** permeten fer operacions booleanes amb els nodes.
- **Funcions per dates:** permeten fer operacions diverses amb dates i hores.

És impossible especificar-les totes en aquestos apunts, de manera que el millor és consultar l'especificació XPath (<http://www.w3.org/TR/xpath-functions>) per veure quines funcions es poden fer servir.

Entre totes les funcions podem destacar les de la següent taula:

Funció	Explicació
<b>name()</b>	Retorna el nom del node
<b>sum()</b>	Retorna la suma d'una seqüència de nodes o de valors
<b>count()</b>	Retorna el nombre de nodes que hi ha en una seqüència
<b>avg()</b>	Fa la mitjana dels valors de la seqüència
<b>max()</b>	Retorna el valor màxim de la seqüència
<b>min()</b>	Dóna el valor mínim de la seqüència
<b>position()</b>	Diu en quina posició es troba el node actual
<b>last()</b>	Retorna si el node actual és l'últim
<b>distinct-values()</b>	Retorna els elements de la seqüència sense duplicats
<b>concat()</b>	Uneix dues cadenes de caràcters
<b>starts-with()</b>	Retorna si la cadena comença amb els caràcters marcats
<b>contains()</b>	Ens diu si el resultat conté el valor
<b>string-length()</b>	Retorna la llargària de la cadena
<b>substring()</b>	Permet extraure una subcadena del resultat

<b>string-join()</b>	Uneix la seqüència amb el separador especificat
<b>current-date()</b>	Ens retornarà l'hora actual
<b>not()</b>	Inverteix el valors booleans

Amb les funcions es podran fer peticions que retornen valors numèrics. Per exemple, "quants alumnes tenim?":

```
count(/classe/alumnes/alumne)
```

que tornarà el nombre d'alumnes del fitxer

```
3
```

O bé retornar cadenes de caràcters. Per exemple, unir tots els noms, separant-los per una coma:

```
string-join(/classe//nom, ",")
```

que tornarà en un únic resultat els noms separats per una coma:

```
"Joan,Albert,Bernat,Joan"
```

També es poden posar les funcions en les condicions. Per exemple, aquesta expressió ens tornarà els alumnes amb el cognom que comence per B:

```
/classe/alumnes/alumne[starts-with(cognoms, "B") ]  
<alumne aprovat="no">  
<nom>Bernat</nom>  
<cognoms>Balaguer</cognoms>  
<nota>3</nota>  
</alumne>
```

En aquesta expressió volem obtenir el tercer alumne de la llista:

```
/classe/alumnes/alumne[position()=3]  
<alumne delegat="si" aprovat="si">  
<nom>Joan</nom>  
<cognoms>Centelles</cognoms>  
<nota>8</nota>  
</alumne>
```

## 6.2 - XQUERY

---

**XQuery** és un llenguatge de consultes pensat per convertir-se en la manera estàndard de recuperar dades de col·leccions de documents XML.

Es tracta d'un llenguatge molt potent i que és funcional, de manera que en compte de dir-li quins són els passos per fer una tasca, el que es fa és **avaluar** les expressions contra el fitxer XML i **generar un resultat**. A diferència dels llenguatges de programació habituals, en XQuery s'especifica què és el que es vol i no la manera com ho ha de fer per obtenir-ho. Per tant el podríem catalogar com a llenguatge de quarta generació (com el SQL).

Entre les característiques més interessants d'XQuery, aquest permet:

- Seleccionar la informació segons criteris. Ordenar, agrupar, afegir dades.
- Filtrar la informació.
- Buscar informació en un document o en un grup de documents.
- Unir dades de múltiples documents.
- Transformar i reestructurar XML.
- No està limitat a la recerca, ja que pot fer operacions numèriques i de manipulació de caràcters.
- Pot treballar amb espais de noms i amb documents definits per mitjà de DTD o XSD.

Una part important de XQuery 1.0 és el llenguatge XPath 2.0, que és la part que li permet fer les seleccions d'informació i la navegació pel document.

Com anirem veient en els següents punts, la sentència que més utilitzarem en XQuery és la sentència **FLWOR**, acrònim de **For**, **Let**, **Where**, **Order by** i **Return**, que són les clàusules que es poden posar en la sentència .

L'última versió reconeguda de XQuery és la 3.1, que inclou clàusules com el **group by**.

## 6.2.1 - Consultes XQuery

Generalment una consulta XQuery consistirà en aconseguir les dades amb les quals es vol treballar, filtrar-les i retornar el resultat com a XML. L'expressió més simple que es pot fer en XQuery és escriure un element buit:

```
<Hola />
```

En executar aquesta ordre el resultat serà:

```
<?xml version="1.0" ?>  
<Hola/>
```

Encara que a nosaltres en l'entorn de **eXist** ens eixirà únicament l'etiqueta. A pesar d'això serà un document ben format.

Aquesta és una de les característiques importants de XQuery. Pot escriure literalment un text. Però si en el resultat es defineixen etiquetes, l'expressió només serà correcta si el resultat final està ben format. Per exemple, executar el següent en XQuery donarà un error, ja que el resultat final no està ben format:

```
<a>
```

```
Cannot compile xquery: exerr:ERROR err:XPST0003: No closing end tag found for element constructor: a [at line 2, column 2]
```

(observeu com en l'error ens avisa que no ha trobat el tancament de l'etiqueta, és a dir, que falta `</a>`)

Però a pesar de poder escriure text, el més corrent és recuperar algun tipus de dades, avaluar-les i retornar el resultat amb un **return**, que és la instrucció que s'encarrega de generar les sortides.

Una altra característica important és que **fa servir variables**. Per exemple la següent és una expressió XQuery correcta que assigna el valor 5 a **\$a** i després en retorna el valor:

```
let $a:=5  
return $a
```

El resultat d'executar-ho amb XQuery serà:

```
5
```

A diferència del que passa amb els literals, el **return** s'executa una vegada per cada valor que agafa del **for**, de manera que si hi ha una expressió que es fa més d'una vegada:

```
for $a in (1,2)  
return <Hola/>
```

El return s'executaria més d'una vegada:

```
<Hola/>  
<Hola/>
```





6.2.2 - Variables

XQuery és un llenguatge pensat per buscar en documents XML, però no solament es pot fer servir per a això, ja que té suport per fer operacions aritmètiques i per treballar amb cadenes de caràcters.

En aquest exemple fem servir XQuery per fer una operació matemàtica senzilla:

```
let $x := 5
let $y := 4
return $x + $y
```

Una característica d’XQuery que el diferencia d’altres llenguatges de consulta és que **disposa de variables**:

- Les variables en XQuery s’identifiquen perquè comencen sempre amb el símbol \$.
- Poden contenir qualsevol valor: literals numèrics o caràcters, seqüències de nodes, ...
- La instrucció per assignar valors a una variable és **let**.

Una de les utilitzacions fonamentals de les variables és guardar elements per poder-los fer servir posteriorment. En l’exemple següent es guarden els elements **<nom>** en la variable **\$alumnes** i després torna la quantitat d’alumnes:

```
let $alumnes := //alumne/nom
return count($alumnes)

3
```

Es poden aplicar filtres XPath a les variables:

```
let $tot := /classe
let $profe := $tot//professor
let $alum := $tot//alumne
return string-join(count($profe), "professors.", count($alum), "alumnes"), " ")

1 professors. 3 alumnes
```

Els valors de les variables els podem comparar amb els operadors de comparació habituals que podem veure a la següent taula:

Operador	Operador2	Ús
=	eq	Dos valors són iguals
!=	ne	Dos valors són diferents
>	gt	Major
>=	ge	Major o igual

<	lt	Menor
<=	le	Menor o igual

Els operadors **eq**, **ne**, **gt**, **ge**, **lt** i **le** només es poden fer servir per a comparar valors individuals.

## 6.2.3 - Expressions avaluables

XQuery està pensat per poder mesclar les consultes amb qualsevol altre tipus de contingut. Si es mescla amb contingut, les expressions que s'hagen d'avaluar s'han de col·locar entre claus "{...}".

Per exemple, podem mesclar HTML i XQuery per tal que el processador XQuery genere una pàgina web amb les dades d'un document XML.

```
<html>
<head><title>Llista de classe</title></head>
<body>
  <h1>
  {
    //assignatura
  }
  </h1>
</body>
</html>
```

```
<html>
<head>
  <title>Llista de classe</title>
</head>
<body>
  <h1>
    <assignatura>Llenguatges de Marques</assignatura>
  </h1>
</body>
</html>
```

En mesclar contingut, el processador XQuery només avaluarà les instruccions que estiguen dins de les claus, i per tant ha deixat les etiquetes HTML tal com estan.

### Creació d'elements i atributs

Fent servir les expressions avaluables és fàcil crear nous elements.

```
<modul>
{ //assignatura/text() }
</modul>
```

```
<modul>Llenguatges de Marques</modul>
```

També es poden crear atributs (s'ha d'anar en compte de no deixar-se les cometes al voltant del valor que obtindran els atributs).

```
<modul nom="{ //assignatura/text() }"/>
```

```
<modul nom="Llenguatges de Marques"/>
```

Una manera alternativa i més potent de definir elements i atributs és fer servir les paraules clau **element** i **attribute**. Aquestes instruccions permeten crear elements i definir-ne el contingut especificant-lo dins de les claus.

```
element modul {  
}
```

Crearia l'element buit **<modul/>**

Si volem crear nous elements o atributs dins d'un element definit d'aquesta manera s'han d'especificar dins de les claus. Per exemple, el codi següent:

```
element modul {  
  attribute nom { //assignatura/text() },  
  element alumnes { //alumne }  
}
```

Que ens donarà el següent resultat:

```
<modul nom="Llenguatges de Marques">  
  <alumnes>  
    <alumne aprovat="si">  
      <nom>Albert</nom>  
      <cognoms>Alegre</cognoms>  
      <nota>6</nota>  
    </alumne>  
    <alumne aprovat="no">  
      <nom>Bernat</nom>  
      <cognoms>Balaguer</cognoms>  
      <nota>3</nota>  
    </alumne>  
    <alumne delegat="si" aprovat="si">  
      <nom>Joan</nom>  
      <cognoms>Centelles</cognoms>  
      <nota>8</nota>  
    </alumne>  
  </alumnes>  
</modul>
```

## 6.2.4 - Expressions FLWOR

Les expressions FLWOR són la part més potent d'XQuery. Estan formades per cinc instruccions opcionals que permeten fer les consultes i alhora processar-les per seleccionar els ítems que interessen d'una seqüència.

	Clàusula	Ús
f	<b>for</b>	Permet recórrer una seqüència, agafant cada vegada un element d'aquesta
l	<b>let</b>	Serveix per assignar valors a una variable
w	<b>where</b>	Permet filtrar els resultats segons condicions
o	<b>order by</b>	Ordena els resultats abans de mostrar-los
r	<b>return</b>	Retorna el resultat de tota l'expressió

### "FOR ... RETURN"

La instrucció **for** es fa servir per avaluar individualment cadascun dels resultats d'una seqüència de nodes. En un **for** es defineixen dues coses:

- Una variable, que serà la que anirà agafant els elements de la seqüència un per un.
- La paraula clau **in**, en la qual es defineix quina és la seqüència d'elements per processar.

La manera més senzilla d'expressió FLWOR és combinar el **for** amb el **return** per retornar els valors obtinguts de manera seqüencial:

```
for $i in ('Hola' , 'Adéu' )  
return $i
```

generarà:

```
Hola  
Adéu
```

Les seqüències de valors poden ser de qualsevol tipus. Per exemple, poden ser números. Així, el següent exemple:

```
for $i in (1,2,3)  
return $i * $i
```

generarà:

```
1  
4  
9
```

O bé una seqüència de nodes, per mig d'una expressió XPath. Per exemple amb l'expressió següent capturem una seqüència de nodes i els fem servir per obtenir-ne el cognom:

```
for $alumne in //alumne
return <alumne> { $alumne/cognoms } </alumne>
```

generarà:

```
<alumne>
  <cognoms>Alegre</cognoms>
</alumne>
<alumne>
  <cognoms>Balaguer</cognoms>
</alumne>
<alumne>
  <cognoms>Centelles</cognoms>
</alumne>
```

Es pot veure que els valors del **return** s'han anat processant un per un i per aquest motiu es repeteixen les etiquetes **<alumne>**.

Un aspecte important és que no cal que el **for** siga la primera expressió de la consulta. També es pot posar com a paràmetre en una funció XPath.

Per exemple, l'expressió següent ens tornarà el nombre d'alumnes:

```
count (
  for $alumne in //alumne
  return $alumne
)
```

3

En el **for** es pot incloure l'operador **at**, que permet obtenir la posició del node que s'està processant. Amb aquest operador podem fer que aparega el número d'ordre en els resultats.

```
for $alumne at $pos in //alumne
return <alumne numero="{ $pos }">
  { $alumne/cognoms/text() }
</alumne>
```

que donarà

```
<alumne numero="1">Alegre</alumne>
<alumne numero="2">Balaguer</alumne>
<alumne numero="3">Centelles</alumne>
```

## "FOR NIUAT" ("anidat")

Les ordres **for** es poden posar unes dins d'unes altres, d'una manera similar a com ho fa SQL amb les subconsultes. Així es poden aconseguir resultats més complexos que es basen en els resultats obtinguts anteriorment.

```
for $selec in //alumne
return
  <persona>
    { $selec/nom }
    {
      for $selec2 in $selec
      return $selec2//cognom
    }
  </persona>
```

```
<persona>
  <nom>Albert</nom>
</persona>
<persona>
  <nom>Bernat</nom>
</persona>
<persona>
  <nom>Joan</nom>
</persona>
```

Aquesta sentència anterior queda un poc pobra, ja que en el document hi ha molt poques dades.

Un exemple un poc més complet (i comprensiu) seria el següent:

```
for $selec in //ruta
return
  <ruta>
    <nomRuta>
      { $selec/nom/text() }
    </nomRuta>
    <punts>
      {
        for $selec2 in $selec//punt
        return <punt> {$selec2/nom/text()} </punt>
      }
    </punts>
  </ruta>
```

que donaria el sagüent resultat:

```
<ruta>
  <nomRuta>Pujada a Penyagolosa</nomRuta>
  <punts>
    <punt>Sant Joan</punt>
    <punt>Encreuament</punt>
    <punt>Barranc de la Pregunta</punt>
    <punt>El Corralico</punt>
    <punt>Penyagolosa</punt>
  </punts>
```



```

</ruta>
<ruta>
  <nomRuta>La Magdalena</nomRuta>
  <punts>
    <punt>Primer Molí</punt>
    <punt>Segon Molí</punt>
    <punt>Caminàs</punt>
    <punt>Riu Sec</punt>
    <punt>Sant Roc</punt>
    <punt>Explanada</punt>
    <punt>La Magdalena</punt>
  </punts>
</ruta>
...

```

## LET

Ens permet declarar variables i assignar-los un valor. Sobretot es fa servir per guardar valors que s'han de fer servir més tard.

```

let $num := 1
let $nom := "Pere"
let $i := (1 to 3)

```

En les expressions FLWOR hi pot haver tants **let** com calga.

```

for $alumne in doc("db/Tema8/classe.xml")//alumne
let $nom := $alumne/nom
let $nota := $alumne/nota
return <nom>{ concat($nom," : ", $nota) }</nom>

```

```

<nom>Albert: 6</nom>
<nom>Bernat: 3</nom>
<nom>Joan: 8</nom>

```

S'ha d'anar amb compte amb **let** perquè el seu funcionament és molt diferent del de **for**:

- **for** s'executa per a cada membre d'una seqüència.
- **let** fa referència al seu valor en conjunt. Si és una seqüència, el que es processa són tots els valors de cop.

Podem veure la diferència amb un exemple. Aquesta instrucció amb **for**:

```

for $selec in //alumne
return <persona>{$selec/nom}</persona>

```

Dóna:

```

persona>
  <nom>Albert</nom>
</persona>
<persona>
  <nom>Bernat</nom>
</persona>
<persona>
  <nom>Joan</nom>
</persona>

```

I en canvi amb **let**:

```

let $selec := //alumne
return <persona>{$selec/nom}</persona>

```

Donarà:

```

<persona>
  <nom>Albert</nom>
  <nom>Bernat</nom>
  <nom>Joan</nom>
</persona>

```

Els resultats són bastant diferents! Es pot veure clarament que **let** ha tractat tots els valors de cop.

## WHERE

La instrucció **where** és la part que indicarà quin filtre s'ha de posar a les dades rebudes abans d'enviar-les a la sortida del **return**. Normalment en el filtre es fa servir algun tipus de predicat XPath:

```

for $alumne in //alumne
where $alumne/@aprovat="si"
return $alumne/nom

```

```

<nom>Albert</nom>
<nom>Joan</nom>

```

Com que XPath forma part d'XQuery moltes vegades el mateix filtre es pot definir de diverses maneres. Per exemple, aquesta expressió és equivalent a l'anterior:

```

for $alumne in //alumne[@aprovat="si"]
return $alumne/nom

```

Això sí, en un **where** hi pot haver tantes condicions com faça falta simplement encadenant-les amb les operacions lògiques **and**, **or** o **not()**.

Aquesta expressió retorna el cognom dels alumnes que han aprovat i que es diuen *Joan*:

```
for $alumne in //alumne
where $alumne/@aprovat="si" and $alumne/nom="Joan"
return $alumne/cognoms
```

```
<cognoms>Centelles</cognoms>
```

El **where** afegeix més potència del que sembla, ja que ens permet fer els **inner joins** d'SQL en fitxers XML. Per exemple, a partir de dos fitxers amb les dades de dues classes diferents es poden obtenir només els alumnes que es repeteixen entre les dues classes amb:

```
for $alum1 in doc("db/Tema7/classe.xml")//alumne
let $alum2 in doc("db/Tema7/classe2.xml")//alumne
where $alum1 = $alum2
return $alum1
```

### Atenció

L'anterior només és un exemple que no funcionarà, ja que no disposem del document **classe2.xml**

## ORDER BY

Una de les operacions habituals en les cerques és representar els resultats en un ordre determinat. Aquesta és la funció que fa **order by**.

Es pot ordenar de manera ascendent amb **ascending** (que és el que fa per defecte) o bé descendent amb **descending**.

```
for $alumne in //alumne
order by $alumne/cognoms descending
return $alumne
```

```
<alumne delegat="si" aprovat="si">
  <nom>Joan</nom>
  <cognoms>Centelles</cognoms>
  <nota>8</nota>
</alumne>
<alumne aprovat="no">
  <nom>Bernat</nom>
  <cognoms>Balaguer</cognoms>
  <nota>3</nota>
</alumne>
<alumne aprovat="si">
  <nom>Albert</nom>
  <cognoms>Alegre</cognoms>
  <nota>6</nota>
</alumne>
```

També es poden especificar diferents conceptes en l'ordenació. En aquest exemple els resultats s'ordenaran primer per cognom i després per nom:

```
for $alumne in //alumne
order by $alumne/cognoms, $alumne/nom
```

```
return $alumne
```

En aquest altre el cognom s'ordena de manera descendent i el nom de manera ascendent.

```
for $alumne in //alumne  
order by $alumne/cognoms descending, $alumne/nom ascending  
return $alumne
```

Per defecte ordena com si el contingut de les etiquetes fóra text. Si hem d'obtenir una ordenació numèrica, caldrà convertir els valors a números amb la funció d'XPath **number()**.

```
for $ruta in doc("db/Tema8/Rutes.xml")//ruta  
order by number($ruta/desnivellAcumulat)  
return concat($ruta/nom, " --> ", $ruta/desnivellAcumulat)
```

```
La Magdalena --> 84  
Pujada a Penyagolosa --> 530  
Cati - Sant Pere de Castellfort --> 1286  
Pelegrins de Les Useres --> 1738
```

## 6.2.5 - Alternatives

La instrucció **if** ens permetrà fer una cosa o una altra segons si es compleix la condició especificada o no. Per exemple, podem fer una llista d'alumnes en què s'especifiqui si han aprovat o no a partir del valor que hi ha en l'element **<nota>**.

```
let $doc := doc("/db/Tema8/classe.xml")
let $aprovat := 5
for $b in $doc//alumne
return
  if ($b/nota >= $aprovat) then
    <aval> { $b/nom/text() } està aprovat</aval>
  else
    <aval> { $b/nom/text() } no està aprovat</aval>
```

Això generarà el resultat següent:

```
<aval>Albert està aprovat</aval>
<aval>Bernat no està aprovat</aval>
<aval>Joan està aprovat</aval>
```

La instrucció **else** és habitual en llenguatges de programació, però a diferència del que passa normalment, en XQuery **sempre s'ha d'especificar el else**, encara que no es vulga fer res (en aquest cas posaríem uns parèntesis sense res dins). Si no està l'**else**, donarà error.

```
for $alu in doc("/db/Tema8/classe.xml")//alumne
return <alumne>
  { $alu/nom }
  { if ($alu/nota > 5) then
    <aprovat/>
    else () }
  </alumne>
```

```
<alumne>
  <nom>Albert</nom>
  <aprovat/>
</alumne>
<alumne>
  <nom>Bernat</nom>
</alumne>
<alumne>
  <nom>Joan</nom>
  <aprovat/>
</alumne>
```

## 6.2.6 - Exemple de consulta "elaborada"

Partint d'aquest XML d'exemple que representa comandes de llibres (guardau-lo com un document anomenat **comanda\_llibres.xml**):

```
<?xml version="1.0" encoding="UTF-8"?>
<petició llibreria="Fantastica SL">
  <data>15-11-2011</data>
  <autor>
    <nom>Frédèrik McCloud</nom>
    <llibres>
      <llibre>
        <titol>Les empentes</titol>
        <quantitat>2</quantitat>
      </llibre>
    </llibres>
  </autor>
  <autor>
    <nom>Corsaro Levy</nom>
    <llibres>
      <llibre>
        <titol>Marxant de la font del gat</titol>
        <quantitat>2</quantitat>
      </llibre>
      <llibre>
        <titol>Bèstia!</titol>
        <quantitat>2</quantitat>
      </llibre>
    </llibres>
  </autor>
  <autor>
    <nom>Marc Blairet</nom>
    <llibres>
      <llibre>
        <titol>Tres de tres</titol>
        <quantitat>3</quantitat>
      </llibre>
    </llibres>
  </autor>
</petició>
```

Es vol respondre la pregunta següent: **de quins autors s'han demanat tres o més llibres?**

Anem a fer l'exercici a poc a poc, de manera constructiva.

Per respondre aquesta pregunta el primer que cal és separar el document en blocs d'autor i processar-ne cada un de manera diferent. Per tant, es defineix un bucle amb un **for** per cada autor i s'escriu el nom per pantalla

```
for $autor in doc("/db/Tema8/comanda_llibres.xml")//autor
return $autor/nom
```

El resultat serà una llista de tots els autors:

```
<nom>Frédéric McCloud</nom>
<nom>Corsaro Levy</nom>
<nom>Marc Blairet</nom>
```

Es vol definir una restricció amb la quantitat de llibres que s'han demanat de cada autor; per tant, ho podríem intentar fer amb el **where**

```
for $autor in doc("/db/Tema8/comanda_llibres.xml")//autor
where $autor/llibres/llibre/quantitat >= 3
return $autor/nom
```

Aquesta és la llista dels autors que tenen alguna petició de 3 o més llibres

```
<nom>Marc Blairet</nom>
```

Però es pot veure que encara no és el demanat, ja que no apareix l'autor Corsaro Levy, del qual s'han demanat dues unitats llibres diferents. Per tant, agrupem les quantitats per obtenir el resultat correcte

```
for $autor in doc("/db/Tema8/comanda_llibres.xml")//autor
where sum($autor/llibres/llibre/quantitat) >= 3
return $autor/nom
```

Ara el resultat sí que és el correcte:

```
<nom>Corsaro Levy</nom>
<nom>Marc Blairet</nom>
```

Podem fer que el resultat siga més "bonic" mostrant les quantitats venudes al costat de cada autor i ordenant de més a menys els resultats.

```
for $autor in doc("/db/Tema8/comanda_llibres.xml")//autor
let $suma := sum($autor/llibres/llibre/quantitat)
where $suma >= 3
order by $suma descending
return concat($autor/nom, " : ", $suma)
```

Es pot veure que per no haver d'escriure diverses vegades l'expressió de suma s'ha definit una variable **\$suma** i d'aquesta manera l'expressió ha quedat més fàcil d'escriure

```
"Corsaro Levy : 4"
"Marc Blairet : 3"
```

La funció **concat()** ha eliminat les etiquetes XML i ha generat un resultat de text. Si es vol mantenir una estructura XML es pot crear manualment amb l'ajuda de les expressions avaluable:

```
for $autor in doc("/db/Tema8/comandallibres.xml")//autor
let $suma := sum($autor/libres/libre/quantitat)
where $suma >=3
order by $suma descending
return <autor> { concat($autor/nom," : ", $suma) } </autor>
```

Que generarà:

```
<autor>Corsaro Levy : 4</autor>
<autor>Marc Blairet : 3</autor>
```



## 7 - API Java estàndards per a BD-XML natives

---

Les dades emmagatzemades a les BD han de poder ser accessibles des d'aplicacions desenvolupades en diferents llenguatges i, per aquest motiu, els SGBD es veuen obligats a facilitar interfícies de programació per als llenguatges de programació més comuns. Així, els fabricants d'SGBD, coneixedors de la tecnologia utilitzada en el seu producte, faciliten una API per permetre-hi l'accés, desenvolupada de la manera més eficient possible per al seu producte, fet que comporta l'aparició d'un problema: l'API proporcionada per a cada SGBD és pròpia i diferent de les API dels altres SGBD i, en conseqüència, les aplicacions desenvolupades queden lligades a l'SGBD i els programadors han de conèixer un munt d'API diferents, tantes com SGBD diferents als quals han d'enllaçar.

Davant l'anarquia d'API existent per atacar els SGBD d'un determinat tipus, acostumen a aparèixer intents per estandarditzar el mecanisme i proporcionar una API estàndard. En el cas de les BD-XML natives hi ha hagut dos processos d'estandardització per al llenguatge Java, que han donat lloc a dues API estàndards: **XML:DB** (també anomenada XAPI) i **XQueryAPI for Java** (XQJ).

Actualment hi ha molts SGBD-XML natives que faciliten la implementació de les dues API. Sembla, però, que l'API XQJ és la que està destinada a evolucionar, ja que està promoguda pel W3C, mentre que l'API XML:DB, anterior en el temps, està aturada des del 2003. Encara que seria recomanable conèixer les dues API, nosaltres només veurem **XQJ**, ja que en el moment actual sembla que té molt més futur que XML:DB.

## 7.1 - API XQJ

---

**XQuery API for Java (XQJ)** és una interfície de programació d'aplicacions Java pensada per utilitzar el llenguatge XQuery per obtenir informació de BD-XML natives, de manera semblant a com JDBC és una API pensada per utilitzar el llenguatge SQL per accedir a BD Relacionals.

XQJ va néixer el 2003 i la seua versió definitiva ha estat publicada el 2009. També és coneguda com a JSR 225 ja que ha estat dissenyada com un projecte JCP (Java Community Process). De moment (versió 1.7 de Java), no forma part de la llibreria estàndard de classes Java.

Nombrosos SGBD-XML natives faciliten la connectivitat des de Java mitjançant aquesta XQJ, de manera que podem desenvolupar aplicacions que accedesquen a SGBD-XML natives via XQJ amb l'única particularitat d'haver d'utilitzar la implementació de l'API que facilita cada SGBD.

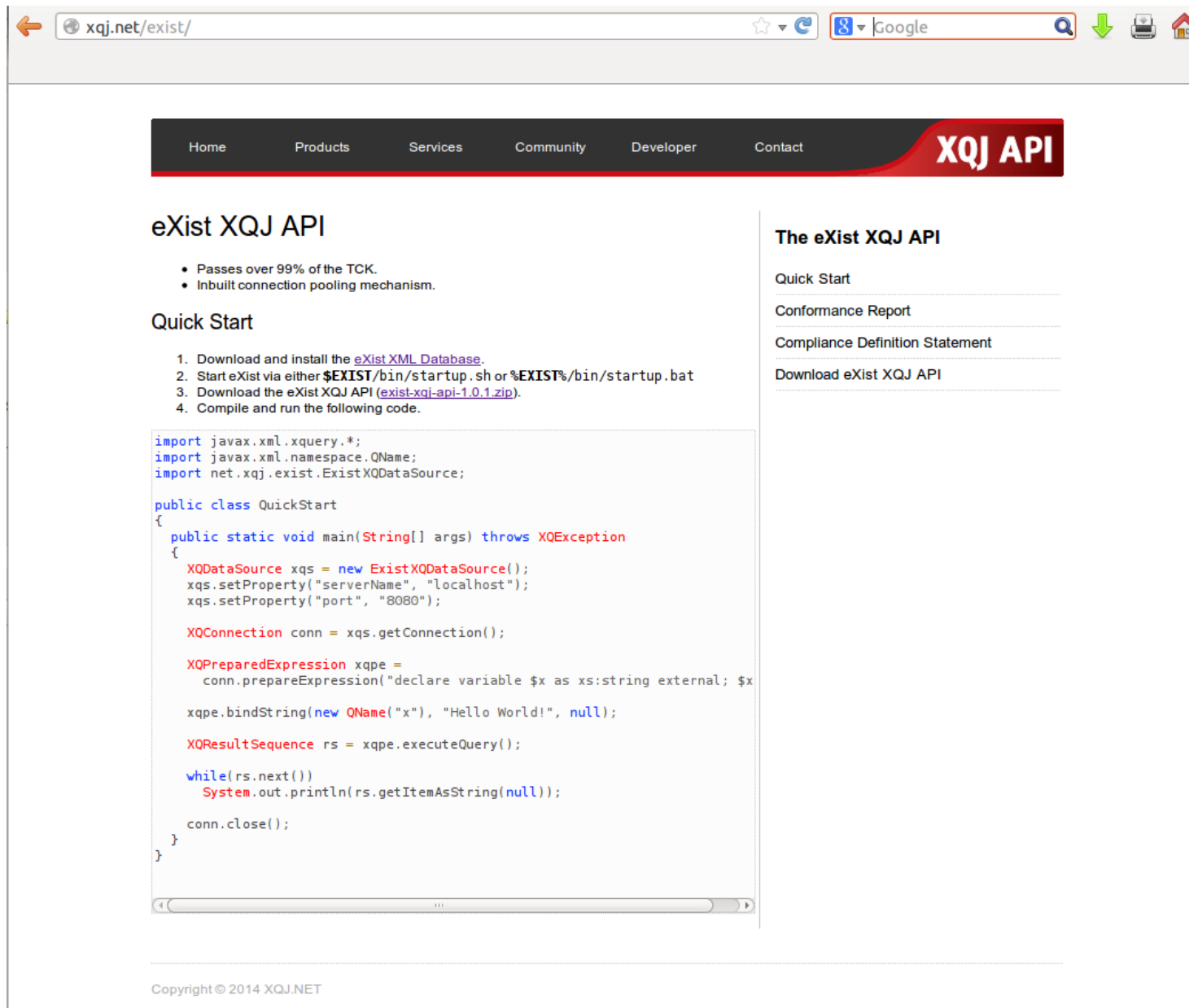
A continuació desenvoluparem aplicacions Java que connecten contra SGBD-XML natives via XQJ i comprovarem que s'executen correctament en el SGBD-XML natives que estem utilitzant: **eXist-db**.

## 7.1.1 - Descàrrega

---

Podem descarregar-nos la API XQJ per a eXist des de la pàgina <http://xqj.net>, on estan les API per a les 3 Bases de Dades XML més utilitzades: eXist, BaseX i Sedna, a banda d'alguna coseta més.

Concretament, la de eXist està en <http://xqj.net/exist>:



The screenshot shows a web browser window with the address bar displaying 'xqj.net/exist/'. The page features a navigation bar with links: Home, Products, Services, Community, Developer, and Contact. A prominent red banner on the right side of the navigation bar reads 'XQJ API'. The main content area is titled 'eXist XQJ API' and includes a bulleted list of features: 'Passes over 99% of the TCK' and 'Inbuilt connection pooling mechanism.' Below this is a 'Quick Start' section with a numbered list of four steps: 1. Download and install the eXist XML Database. 2. Start eXist via either \$EXIST/bin/startup.sh or %EXIST%/bin/startup.bat. 3. Download the eXist XQJ API (exist-xqj-api-1.0.1.zip). 4. Compile and run the following code. A code block follows, containing Java code for a 'QuickStart' class that connects to the eXist database and executes an XQuery. The code is as follows:

```
import javax.xml.xpath.*;
import javax.xml.namespace.QName;
import net.xqj.exist.ExistXQDataSource;

public class QuickStart
{
    public static void main(String[] args) throws XQException
    {
        XQDataSource xqs = new ExistXQDataSource();
        xqs.setProperty("serverName", "localhost");
        xqs.setProperty("port", "8080");

        XQConnection conn = xqs.getConnection();

        XQPreparedExpression xqpe =
            conn.prepareExpression("declare variable $x as xs:string external; $x");

        xqpe.bindString(new QName("x"), "Hello World!", null);

        XQResultSequence rs = xqpe.executeQuery();

        while(rs.next())
            System.out.println(rs.getItemAsString(null));

        conn.close();
    }
}
```

On the right side of the page, there is a sidebar titled 'The eXist XQJ API' containing links for 'Quick Start', 'Conformance Report', 'Compliance Definition Statement', and 'Download eXist XQJ API'. The footer of the page states 'Copyright © 2014 XQJ.NET'.

Com veiem, a banda de suggerir-nos instal·lar eXist i posar-lo en marxa, ens dóna un enllaç per a la descàrrega de l'API XQJ, a més d'un exemple d'utilització. L'enllaç és:

<http://xqj.net/exist/exist-xqj-api-1.0.1.zip>

En aquest fitxer comprimit tenim tant les llibreries (.jar) com documentació. Com únicament són 3 els jar (no caldria incorporar el d'exemples), els podem afegir al projecte sense més, o bé podem col·locar-los en una llibreria d'usuari d'Eclipse (anomenada per exemple XQJ), per a més comoditat

## 7.1.2 - Connexió

Per a poder establir una connexió des de l'aplicació Java fins la Base de Dades (en el nostre cas eXist) necessitem dos objectes:

### XQDataSource

Servirà per a identificar l'origen de dades. A partir d'ell crearem les connexions (similars a sessions). De forma genèrica seria:

```
XQDataSource xqs = new ClasseEspecíficaQueGeneraXQDataSource();
```

que en el cas concret dels SGBD-XML natives més comuns s'haurà de particularitzar en:

- **eXist** serà **ExistXQDataSource()**.
- **BaseX** seria **BaseXQDataSource()**
- **Sedna** seria **SednoXQDataSource()**

I aquesta seria l'única diferència entre utilitzar un o un altre SGBD-XML natives, a banda de les dades de connexió.

Com que en aquestos apunts únicament utilitzarem **eXist-db**, la nostra classe serà **ExistXQDataSource()**

La manera d'indicar el servidor, port, usuari i contrasenya és per mig del mètode **setProperty()**. Aquest seria un exemple, on connectem al servidor situat en la mateixa màquina, al port **8080** (que és el port per defecte), i com a usuari **admin** (**admin**):

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

### XQConnection

Representa una connexió, una sessió. Mantindrà informació d'estat, transaccions, consultes executades i resultats.

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

També es podria passar l'usuari i contrasenya en el moment de fer la connexió:

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

I en finalitzar, hem de recordar tancar aquesta connexió:

```
conn.close();
```

Aquest seria un exemple on únicament avisem que hem pogut connectar. El podem fer en un projecte nou, anomenat **Tema8\_2**, dins del paquet **Exemples**

```
import javax.xml.xquery.XQConnection;
import javax.xml.xquery.XQDataSource;
import javax.xml.xquery.XQException;

import net.xqj.exist.ExistXQDataSource;

public class Prova0_XQJ {

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

        XQConnection conn = s.getConnection();
        System.out.println("Connexió feta");
        conn.close();
    }
}
```

## 7.1.3 - Resultats de sentències XQuery

Aquestes són les classes que utilitzarem per accedir a les dades:

### XQExpression o XQPreparedExpression

Són objectes creats a partir de la connexió per a l'execució d'una expressió (sentència) **XQuery**. Ja sabem que XQuery inclou XPath, per tant també podem posar senzillament una expressió XPath.

**XQExpression** serveix per a executar sentències de forma immediata, si volem una darrere de l'altra. **XQPreparedExpression** serveix més per a preparar-les, exactament igual que amb els **Statement** de **JDBC** per a BD Relacionals.

Mentre que per a XQExpression no declarem la sentència en el moment de crear-la, sinó posteriorment amb **executeQuery(sentència)** passant-li la sentència, en XQExpression declarem la sentència en el moment de crear-la, i després només restarà executar-la (sense passar-li-la com a paràmetre amb **executeQuery()**).

En ambdós casos el resultat vindrà en un **XQResultSequence**.

### XQResultSequence

Arreplega el resultat de l'execució d'una sentència per qualsevol dels mètodes anteriors. Contindrà un conjunt de 0 o més **XQResultItem**.

És un objecte que es pot recórrer. Veja'm un exemple, on per a obtenir l'element del **ResultSequence** utilitzem el mètode **getItemAsString(null)** per a convertir-lo tot sencer en string. Primer utilitzem **XQPreparedExpression**:

```
import javax.xml.xquery.XQConnection;
import javax.xml.xquery.XQDataSource;
import javax.xml.xquery.XQException;
import javax.xml.xquery.XQPreparedExpression;
import javax.xml.xquery.XQResultSequence;

import net.xqj.exist.ExistXQDataSource;

public class Proval_XQJ {

    public static void main(String[] args) throws XQException {
        XQDataSource s = new ExistXQDataSource();
```



```

s.setProperty("serverName", "localhost");
s.setProperty("port", "8080");
s.setProperty("user", "admin");
s.setProperty("password", "admin");

XQConnection conn = s.getConnection();
System.out.println("Connexió feta");
String sent ="for $alumne in doc(\"/db/Tema8/classe.xml\")//alumne order by $alumne/cognoms return $alumne";

XQPreparedExpression cons = conn.prepareExpression(sent);
XQResultSequence rs = cons.executeQuery();

while(rs.next())
    System.out.println(rs.getItemAsString(null));

conn.close();
}

```

I ara anem a canviar la consulta per veure com queda amb **XQExpression**:

```

...

XQExpression cons = conn.createExpression();
XQResultSequence rs = cons.executeQuery(sent);
while(rs.next())
    System.out.println(rs.getItemAsString(null));

...

```

En els programes anteriors hem buscat en un únic document, **classe.xml**, posant-li la ruta. Com que per a posar el nom i ruta del document fan falta les cometes "", ens ha tocat "escapar-les" amb la contra-barra. Si no posàrem el document, ja sabem que eXist buscaria en totes les col·leccions.

## XQResultItem

Representa un element d'un resultat. Ja hem vist que el XQResultSequence el podem recórrer, però si volem treballar millor o de forma més completa amb cada element, podem passar-lo al **XQResultItem**. El mateix exemple d'abans quedaria:

```

import javax.xml.xquery.XQConnection;
import javax.xml.xquery.XQDataSource;
import javax.xml.xquery.XQException;
import javax.xml.xquery.XQPreparedExpression;
import javax.xml.xquery.XQResultItem;
import javax.xml.xquery.XQResultSequence;

import net.xqj.exist.ExistXQDataSource;

```

```
public class Prova2_XQJ {  
    public static void main(String[] args) throws XQException {  
        XQDataSource s = new ExistXQDataSource();  
        s.setProperty("serverName", "localhost");  
        s.setProperty("port", "8080");  
        s.setProperty("user", "admin");  
        s.setProperty("password", "admin");  
  
        XQConnection conn = s.getConnection();  
        String sent ="for $alumne in doc(\"/db/Tema8/classe.xml\")//alumne order by $alumne/cognoms return $alumne";  
  
        XQPreparedExpression cons = conn.prepareExpression(sent);  
        XQResultSequence rs = cons.executeQuery();  
  
        XQResultItem r_item=null;  
        while(rs.next()){  
            r_item = (XQResultItem) rs.getItem();  
            System.out.println(r_item.getItemAsString(null));  
        }  
        conn.close();  
    }  
}
```

## 7.1.4 - Processar el resultat

En els exemples anteriors hem processat de forma molt bàsica el resultat. Hem recorregut el `XQResultSequence`, per a traure tot el contingut de cada element, passant-lo a `String`

En realitat XQJ ens ofereix més possibilitats, que ara veurem de forma resumida.

### Moviment

XQJ ens permet els següents mètodes de moviment o relacionats amb el moviment:

**`next()`, `previous()`, `first()`, `last()`, `beforeFirst()`, `afterLast()`, `getPosition()`, `isLast()`, `isFirst()`, `isAfterLast()`, `isBeforeFirst()`, `count()`**

Tots ells són de fàcil comprensió. L'únic problema és que la connexió ha de permetre el moviment en les dues direccions. Per defecte només es pot anar cap avant (***forward only***). Es pot canviar modificant les propietats de la connexió per a que siga **`SCROLLTYPE_SCROLLABLE`**, com es veu en el següent exemple, que és una modificació dels exercicis anteriors per a recórrer de forma inversa:

```
import javax.xml.xquery.XQConnection;
import javax.xml.xquery.XQConstants;
import javax.xml.xquery.XQDataSource;
import javax.xml.xquery.XQException;
import javax.xml.xquery.XQPreparedExpression;
import javax.xml.xquery.XQResultSequence;
import javax.xml.xquery.XQStaticContext;

import net.xqj.exist.ExistXQDataSource;

public class Prova3_XQJ {

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

        XQConnection conn = s.getConnection();
        String sent = "for $alumne in //alumne order by $alumne/cognoms return $alumne";

        XQStaticContext cntxt = conn.getStaticContext();
        cntxt.setScrollability(XQConstants.SCROLLTYPE_SCROLLABLE);
        conn.setStaticContext(cntxt);
        XQPreparedExpression cons = conn.prepareExpression(sent);
```

```

        XQResultSequence rs = cons.executeQuery();
        rs.afterLast();
        while(rs.previous())
            System.out.println(rs.getItemAsString(null));

        conn.close();
    }
}

```

## Obtenció del contingut

Si volem traure ja directament contingut de text, per exemple els cognoms dels alumnes, podem endur-nos alguna sorpresa. Per exemple, el següent exemple per a traure els cognoms dels alumnes per ordre alfabètic:

```

import javax.xml.xquery.XQConnection;
import javax.xml.xquery.XQDataSource;
import javax.xml.xquery.XQException;
import javax.xml.xquery.XQResultSequence;

import net.xqj.exist.ExistXQDataSource;

public class Prova4_XQJ {

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

        XQConnection conn = s.getConnection();
        String sent = "//alumne/cognoms/text()";

        XQResultSequence rs = conn.createExpression().executeQuery(sent);

        while (rs.next())
            System.out.println(rs.getItemAsString(null));

        conn.close();
    }
}

```

No ens traurà el text tan senzill com ens pensàvem, sinó que l'eixida serà:

```

text { "Alegre" }
text { "Balaguer" }
text { "Centelles" }

```

indicant-nos que el resultat són nodes de text. Ho podem solucionar d'una forma molt senzilla utilitzant la funció **xs:string()**, que ho converteix a string normal. Únicament canviant la sentència per aquesta:

```
String sent = "//alumne/cognoms/xs:string(text())";
```

el resultat ja serà el que preteníem:

```
Alegre  
Balaguer  
Centelles
```

També tenim una sèrie de mètodes per a obtenir el contingut que ve en un determinat format (numèric, ...) de forma adequada:

**getInt()**, **getBoolean()**, **getBytes()**, **getDouble()**, ...

Però hem de fer que des de la consulta vinga la informació ja en el format adequat. El següent exemple agafa únicament les notes dels alumnes. I aquesta informació ve de forma numèrica (amb la funció **xs:int()**), aleshores podem utilitzar **getInt()** :

```
import javax.xml.xquery.XQConnection;  
import javax.xml.xquery.XQDataSource;  
import javax.xml.xquery.XQException;  
import javax.xml.xquery.XQPreparedExpression;  
import javax.xml.xquery.XQResultSequence;  
import net.xqj.exist.ExistXQDataSource;  
  
public class Prova5_XQJ {  
    public static void main(String[] args) throws XQException {  
        XQDataSource s = new ExistXQDataSource();  
        s.setProperty("serverName", "localhost");  
        s.setProperty("port", "8080");  
        s.setProperty("user", "admin");  
        s.setProperty("password", "admin");  
  
        XQConnection conn = s.getConnection();  
        String sent = "for $alumne in //alumne return xs:int($alumne/nota/text())";  
  
        XQPreparedExpression cons = conn.prepareExpression(sent);  
  
        XQResultSequence rs = cons.executeQuery();  
        while(rs.next())  
            System.out.println(rs.getInt());  
  
        conn.close();  
    }  
}
```

```
6  
3  
8
```

Però encara serà més interessant que quan ens vinga un element, poder analitzar i extraure els seus subelements, atributs, ...

Per a poder fer açò ens valdrem dels documents DOM, vistos en el tema 3. Per a poder passar la informació utilitzarem el mètode **getObject()**

En el següent exemple, de la consulta obtenim els element, que passem a un element DOM. Posteriorment extraurem la informació del nom, cognoms i nota:

```
import javax.xml.xquery.XQConnection;
import javax.xml.xquery.XQDataSource;
import javax.xml.xquery.XQException;
import javax.xml.xquery.XQPreparedExpression;
import javax.xml.xquery.XQResultSequence;

import net.xqj.exist.ExistXQDataSource;

import org.w3c.dom.Element;

public class Prova6_XQJ {
    public static void main(String[] args) throws XQException {
        XQDataSource s = new ExistXQDataSource();
        s.setProperty("serverName", "localhost");
        s.setProperty("port", "8080");
        s.setProperty("user", "admin");
        s.setProperty("password", "admin");

        XQConnection conn = s.getConnection();
        String sent ="for $alumne in //alumne order by $alumne/cognoms return $alumne";

        XQPreparedExpression cons = conn.prepareExpression(sent);

        XQResultSequence rs = cons.executeQuery();
        while(rs.next()) {
            Element el = (Element) rs.getObject();
            System.out.print(el.getElementsByTagName("nom").item(0).getFirstChild().getNodeValue() + " ");
            System.out.print(el.getElementsByTagName("cognoms").item(0).getFirstChild().getNodeValue() + ": ");
            System.out.println(el.getElementsByTagName("nota").item(0).getFirstChild().getNodeValue());
        }
        conn.close();
    }
}
```

```
Albert Alegre: 6
Bernat Balaguer: 3
Joan Centelles: 8
```

Un altre exemple interessant pot ser guardar un fitxer amb una determinada informació (transformada) d'un document guardat en la Base de Dades.

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
```

```

import javax.xml.xquery.XQConnection;
import javax.xml.xquery.XQDataSource;
import javax.xml.xquery.XQException;
import javax.xml.xquery.XQPreparedExpression;
import javax.xml.xquery.XQResultSequence;

import net.xqj.exist.ExistXQDataSource;

public class Prova7_XQJ {
    public static void main(String[] args) throws XQException {
        File f = new File("notes.xml");

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

        XQConnection conn = s.getConnection();
        String sent = "for $classe in /classe " +
            "return <notes> <modul nom=\"{"$classe/assignatura/text()}\">> " +
            "    {for $alumne in $classe//alumne " +
            "        order by $alumne/cognoms " +
            "        return <alumne nota=\"{"$alumne/nota/text()}\">> " +
            "            {concat($alumne/nom/text(), \" \", $alumne/cognoms)}"+
            "    } </alumne> } </modul> </notes> ";

        XQPreparedExpression cons = conn.prepareExpression(sent);

        XQResultSequence rs = cons.executeQuery();
        try {
            BufferedWriter bw = new BufferedWriter(new FileWriter(f));
            bw.write("<?xml version='1.0' ?>" + " ");
            while(rs.next()) {
                String linia = rs.getItemAsString(null);
                System.out.println(linia);
                bw.write(linia + " ");
            }
            bw.close();
        } catch (IOException e) {
            {e.printStackTrace();}
        }
        conn.close();
    }
}

```

El resultat serà el següent fitxer **notes.xml** (a banda que també eixirà per l'eixida estàndar):

```

<?xml version='1.0' ?> <notes>
  <modul nom="Llenguatges de Marques" xmlns="">
    <alumne nota="6" xmlns="">Albert Alegre</alumne>
    <alumne nota="3" xmlns="">Bernat Balaguer</alumne>
    <alumne nota="8" xmlns="">Joan Centelles</alumne>
  </modul>

```

</notes>

que com es veu, està ben format. Aquesta seria la visualització des d'un navegador:

```
-<notes>
  -<modul nom="Llenguatges de Marques">
    <alumne nota="6">Albert Alegre</alumne>
    <alumne nota="3">Bernat Balaguer</alumne>
    <alumne nota="8">Joan Centelles</alumne>
  </modul>
</notes>
```



## Exercicis

---



### Exercici 8.0 (voluntari) (PostgreSQL)

Modifica (ampliant-la) la sentència de l'apartat 3.7 per a guardar en un únic document XML les comarques de la taula COMARQUES. En cada comarca hauran d'aparèixer tots els seu pobles (taula POBLACIONS), i en cada poble tots els seus instituts (taula INSTITUTS).



### Exercici 8.1 (voluntari) (PostgreSQL)

Dins de la BD pròpia de cadascú (la mateixa ja utilitzada en el Tema 6; si no tens clar quina és, contacta amb el teu professor) crea una taula anomenada **DOC\_XML** amb 3 camps:

- **num** (de tipus **serial**, que és l'autonumèric) que contindrà el número del document. Ha de ser **clau principal**.
- **nom** (de tipus cadena) que contindrà un nom per al document.
- **doc** (de tipus xml) que contindrà els documents XML

Puja la sentència de creació de la taula.



### Exercici 8.2 (voluntari) (PostgreSQL)

Fes un programa en Java (en el projecte **Tema8\_1**, en un paquet nou anomenat **Exercicis**) anomenat **Ruta\_XML\_PostgreSQL** que guardi en la taula creada en el punt anterior el contingut del document **Rutes.xml** ja fet en anteriors exercicis (i que et proporcionarà el professor). El nom del document podria ser **Rutes**, i el número no li l'has de proporcionar, ja que és un autonumèric.

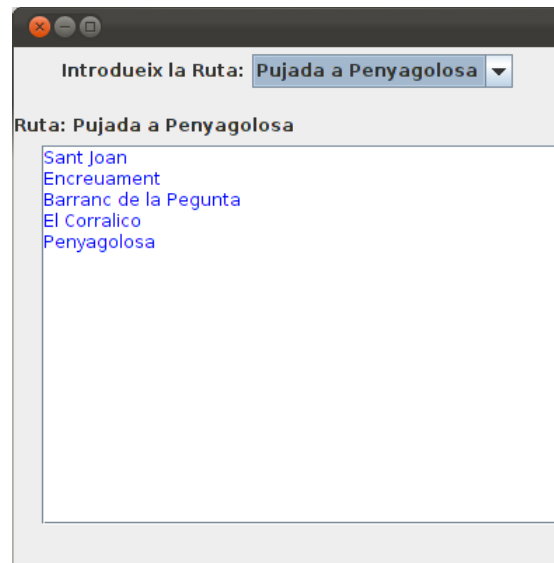


### Exercici 8.3 (voluntari) (PostgreSQL)

Fes un programa dins del mateix projecte i paquet anomenat **Vis\_Rutes\_XML** (i el JFrame: **Vis\_Rutes\_XML\_Pantalla**) que aprofite les llibreries gràfiques Awt i Swing per a mostrar les rutes i els seus punts guardades en la taula DOC\_XML del punt anterior, d'una forma molt similar a l'exercici 3.4 del tema 3.

Observa que es tractarà d'accedir al document XML guardat en la Base de Dades PostgreSQL, però no cal guardar tota l'estructura, sinó que és suficient amb la recerca de determinades coses:

- Per al JComboBox s'han de buscar els noms de les rutes.
- Per al JArea el nom dels punts de la ruta seleccionada.



### Exercici 8.4 (voluntari) (PostgreSQL)

Intenta fer l'exercici 6.5 , però ara accedint als document XML de la Base de Dades PostgreSQL. Anomena'l **Vis\_Rutes\_XML\_PostgreSQL\_Complet**

Ruta:	La Magdalena
Desnivell:	51
Desnivell acumulat:	84

Distància Ruta:			5.33 km
Punts:			
Nom punt	Latitud	Longitud	
Primer Molí	39.99385	-0.032941	
Segon Molí	39.99628	-0.029427	
Caminàs	40.00513	-0.022569	
Riu Sec	40.006765	-0.02237	
Sant Roc	40.017906	-0.02289	
Explanada	40.034048	-0.00633	
La Magdalena	40.034519	-0.005856	

<<
<
>
>>

Intenta traure conclusions sobre quina manera és més còmoda.



### Exercici 8.5 (eXist)

Realitzar les expressions XPath que tornen les següents qüestions (totes elles sobre **Rutes.xml**):

1. Traure el nom de totes les rutes.
2. Traure el nom de les rutes amb un desnivell major que 600 m.
3. Traure les rutes en les quals el desnivell acumulat duplica el desnivell. (**Observació:** quan en una condició combinem alguna operació amb una comparació, és millor posar l'operació abans de l'operador de comparació)
4. Traure el nom del primer punt de cada ruta.
5. Traure el nom de l'últim punt de cada ruta

6. Traure els punts de la tercera ruta.
7. Traure el número de punts de la tercera ruta
8. Traure el nom de les rutes que tenen estrictament més de 5 punts.
9. Traure la mitjana de desnivell de les rutes
10. Traure les rutes en les quals hi ha algun punt més avall del paral·lel 40



## Exercici 8.6 (eXist)

Fer les expressions XQuery per a aconseguir:

1. El mòdul amb el nom del professor com a atribut:

```
<modul professor="Joan Puig">Llenguatges de Marques</modul>
```

2. El nom de cada ruta amb el número de punts com a argument:

```
<rutes>
  <ruta numPunts="5">Pujada a Penyagolosa</ruta>
  <ruta numPunts="7">La Magdalena</ruta>
  <ruta numPunts="6">Pelegrins de Les Useres</ruta>
  <ruta numPunts="6">Catí - Sant Pere de Castellfort</ruta>
</rutes>
```

3. Cada ruta amb el nom com a atribut, la latitud mitjana dels seus punts i la longitud mitjana

```
<ruta nom="Pujada a Penyagolosa">
  <lat_mitj>40.2408508</lat_mitj>
  <long_mitj>-0.3515282</long_mitj>
</ruta>
<ruta nom="La Magdalena">
  <lat_mitj>40.01264257142857</lat_mitj>
  <long_mitj>-0.02034042857142857</long_mitj>
</ruta>
...
```

4. La mitjana de punts de les rutes

```
6
```

5. Quants punts de cada ruta estan en l'hemisferi est (és a dir, longitud positiva)

```
Pujada a Penyagolosa: 0
```

```
La Magdalena: 0
Pelegrins de Les Useres: 0
Catí - Sant Pere de Castellfort: 2
```



### Exercici 8.7 (eXist)

En el projecte **Tema8\_2**, i en un paquet nou anomenat **Exercicis**, fer un programa similar a l'exercici 6.4, però aquesta vegada accedint a la Base de Dades XML-Nativa eXist:

Nom punt	Latitud	Longitud
Catí	40.47095	0.022277778
L'Avellà	40.5019916667	7.75E-4
Salvassòria	40.5124638889	-0.0293944444
La Llâcua	40.5009138889	-0.0552722222
Venta de la Ratlla	40.5103111111	-0.1438055556
Sant Pere	40.4978472222	-0.173075

En aquesta ocasió la connexió serà permanent, fins apretar el botó de tancar. Observeu com els objectes **s** (XQDataSource), **conn** (XQConnection) i **rs** (XResultSetSequence)

estan declarats al principi per no perdre la connexió.

Ens aprofitarem dels mètodes de moviment (first(), previous(), next(), last()) per a manejar-nos d'una ruta a una altra, i per tant s'ha de poder permetre el moviment cap avant i cap arrere.

Observeu com el mètode **plenar\_taula(element)** ha canviat. Us pot servir de model per a emplenar les altres coses, i observeu com se li ha de passar un element DOM (l'element **punts**).

Aquest classe es pot dir **Vis\_Rutes\_XML\_eXist\_Pantalla.java**. I recordeu que heu de tenir una altra classe amb el mètode main que invoque a **Vis\_Rutes\_XML\_eXist\_Pantalla.iniciar()**

```
import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextField;
import javax.swing.JTable;
import javax.xml.xquery.XQConnection;
import javax.xml.xquery.XQConstants;
import javax.xml.xquery.XQDataSource;
import javax.xml.xquery.XQException;
import javax.xml.xquery.XQPreparedExpression;
import javax.xml.xquery.XQResultSequence;
import javax.xml.xquery.XQStaticContext;

import org.w3c.dom.Element;
import org.w3c.dom.NodeList;

import net.xqj.exist.ExistXQDataSource;

public class Vis_Rutes_XML_eXist_Pantalla extends JFrame implements ActionListener {

    private static final long serialVersionUID = 1L;

    XQDataSource s;
    XQConnection conn;
    XQResultSequence rs;

    JLabel etiqueta = new JLabel("");
    JLabel etNom = new JLabel("Ruta:");
    JTextField qNom = new JTextField(15);
    JLabel etDesn = new JLabel("Desnivell:");
    JTextField qDesn = new JTextField(5);
    JLabel etDesnAcum = new JLabel("Desnivell acumulat:");
```

```
JTextField qDesnAcum = new JTextField(5);
JLabel etPunts = new JLabel("Punts:");
JTable punts = new JTable(1,3);
JButton primer = new JButton("<<");
JButton anterior = new JButton("<");
JButton seguent = new JButton(">");
JButton ultim = new JButton(">>");
JButton tancar = new JButton("Tancar");

// en iniciar posem un contenidor per als elements anteriors
public void iniciar() throws XQException {
    getContentPane().setLayout(new GridLayout(0,1));
    JPanel p_prin = new JPanel();
    p_prin.setLayout(new BoxLayout(p_prin, BoxLayout.Y_AXIS));
    // contenidor per als elements
    JPanel panell1 = new JPanel(new GridLayout(0,2));
    panell1.add(etNom);
    qNom.setEditable(false);
    panell1.add(qNom);
    panell1.add(etDesn);
    qDesn.setEditable(false);
    panell1.add(qDesn);
    panell1.add(etDesnAcum);
    qDesnAcum.setEditable(false);
    panell1.add(qDesnAcum);
    panell1.add(etPunts);

    JPanel panell2 = new JPanel(new GridLayout(0,1));
    punts.setEnabled(false);
    JScrollPane scroll = new JScrollPane(punts);
    panell2.add(scroll, null);

    JPanel panell5 = new JPanel(new FlowLayout());
    panell5.add(primer);
    panell5.add(anterior);
    panell5.add(seguent);
    panell5.add(ultim);
    panell5.add(tancar);

    getContentPane().add(p_prin);
    p_prin.add(panell1);
    p_prin.add(panell2);
    p_prin.add(panell5);

    setVisible(true);
    pack();
    primer.addActionListener(this);
    anterior.addActionListener(this);
    seguent.addActionListener(this);
    ultim.addActionListener(this);
    tancar.addActionListener(this);

    inicialitzar();
    VisRuta();
}
```

```
private void plenar_taula(Element e_punts){
    NodeList ll_punts = e_punts.getElementsByTagName("punt");

    Object[][] ll= new Object[ll_punts.getLength()][3];
    for (int i=0;i<ll_punts.getLength();i++){
        Element p=(Element) ll_punts.item(i);
        ll[i][0]=p.getElementsByTagName("nom").item(0).getFirstChild().getNodeValue();
        ll[i][1]=p.getElementsByTagName("latitud").item(0).getFirstChild().getNodeValue();
        ll[i][2]=p.getElementsByTagName("longitud").item(0).getFirstChild().getNodeValue();
    }
    String[] caps = {"Nom punt","Latitud","Longitud"};
    punts.setModel(new javax.swing.table.DefaultTableModel(ll,caps));
}

@Override
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == primer) {
    }

    if (e.getSource() == anterior) {
    }

    if (e.getSource() == seguent ) {
    }

    if (e.getSource() == ultim) {
    }

    if (e.getSource() == tancar) {
    }
}

private void inicialitzar() throws XQException {
}

private void VisRuta() throws XQException{
}
}
```



Llicenciat sota la [Llicència Creative Commons Reconeixement NoComercial Compartir Igual 2.5](#)