UF3

Persistència en SGBD-XML

Isidre Guixà i Miranda
Institut Milà i Fontanals d'Igualada

Accés a dades

12 de febrer de 2013
Isidre Guixà i Miranda
Institut Milà i Fontanals
C/. Emili Vallès, 4
08700 - Igualada

Índex de continguts

Introducció

La informàtica és una branca de la tecnologia que té com objectiu el tractament automàtic de la informació mitjançant dispositius electrònics. Això provoca la necessitat d'emmagatzemar la informació (fer-la persistent) i, en conseqüència, van aparèixer, en un primer moment, els sistemes gestors de fitxers (SGF), els quals van evolucionar fins els anomenats sistemes gestors de bases de dades (SGBD).

En el moment actual, els SGBD són part fonamental en els sistemes informàtics i, segons la tècnica que empren per emmagatzemar la informació, apareixen diversos tipus de bases de dades: jeràrquiques, relacionals, objecte-relacionals, orientades a objectes i XML natives. Per tant, els desenvolupadors de programari han de conèixer com accedir a les dades emmagatzemades en els diversos sistemes existents, objectiu del mòdul professional en el que s'insereix aquesta unitat formativa.

L'objectiu d'aquesta unitat formativa és l'accés a les dades emmagatzemades en SGBD XML natives, és a dir, aprendre a desenvolupar programes que accedeixin a BD XML natives, que són bases de dades que emmagatzemen documents en format XML.

El llenguatge XML permet emmagatzemar informació en un format estructurat, que és més o menys organitzat segons el tipus d'informació (no és el mateix estructurar el contingut d'un llibre, que conté uns pocs capítols molt extensos, que una comanda de venda, que conté unes dades concretes: client, data, productes, quantitats, preus, import,...).

Per aconseguir el nostre objectiu necessitem prendre dues decisions: quin(s) llenguatge(s) de programació emprar i sobre quin(s) SGBD XML natives practicar.

L'accés als SGBD s'acostuma a fer, en l'actualitat, a través d'una interfície de programació d'aplicacions (API) facilitada pels fabricants del SGBD en diversos llenguatges. Així, en el moment actual, pels diversos SGBD podem trobar API pels llenguatges C++, Java, Perl, .NET i d'altres. És clar que el temps no ens permet abastar tots els llenguatges i, per tant, hem hagut de decidir-nos per un: Java.

Un dels motius pels que hem escollit Java resideix en que, a banda de ser un llenguatge actualment molt utilitzat, hi ha en el mercat dues API estàndards, en aquest llenguatge, per accedir a SGBD XML natives.

Ja hem comentat que els fabricants dels SGBD faciliten API d'accés per a diversos llenguatges. Si aquestes API són particulars de cada fabricant, és impossible desenvolupar programes que permetin l'accés a diversos SGBD (doncs cada SGBD facilita una API particular). Per tant, l'existència d'API estàndards és fonamental, doncs els programes basats en una API estàndard haurien de poder accedir a qualsevol SGBD que suporti aquesta API.

I respecte els SGBD XML natives sobre els què practicar... Hi ha molts productes en el

mercat; tots ells amb diferències que feien difícil una elecció. Al final hem optat per utilitzar tres SGBD XML natives diferents (BaseX, Sedna i eXist-db) que cobreixen la majoria de possibilitats que ens podem trobar si hem de treballar, a la vida professional, amb un SGBD XML natives.

Una vegada coneixedors del llenguatge de programació i dels SGBD XML natives que emprarem, ens cal conèixer com està estructurada aquesta unitat. L'hem dividit en dos nuclis formatius.

El primer nucli formatiu "*BD-XML natives. API Java específica del SGBD*" introdueix els conceptes bàsics a conèixer en referència als SGBD XML natives i inicia el desenvolupament de programes en Java que accedeixin a SGBD utilitzant API específiques (no estàndards) del SGBD. El material web incorpora uns annexos que expliquen el procés d'instal·lació dels tres SGBD XML natives i també els llenguatges de gestió de documents XML (XQuery per a consultes i llenguatges per actualització XML) que l'alumne ja hauria de conèixer.

El segon nucli formatiu "*API Java estàndards per BD-XML natives"* ens endinsa, com el seu nom indica, en el desenvolupament de programes Java utilitzant les dues API Java estàndards existents en el moment actual: XQJ i XML-DB.

El seguiment d'aquesta unitat formativa pressuposa que l'alumne és coneixedor de:

- El llenguatge XML i dels llenguatges per a gestionar la informació de documents XML: XPath i XQuery per a fer consultes en documents XML així com alguna de les versions de llenguatges existents per a modificar el contingut de documents XML (XUpdate, Update de P. Lehti, XQUF). Aquests coneixements s'adquireixen a les unitats formatives 1 i 2 del mòdul professional "*Llenguatges de marques i sistemes de gestió de la informació"*. Cal tenir en compte que en els annexos del material web hi ha varis exemples d'utilització d'instrucions dels llenguatges per a gestionar la informació de documents XML.

- El llenguatge Java. Aquests coneixements s'adquireixen en el mòdul professional "*Programació*".

Per tal d'assolir un bon aprenentatge, cal estudiar els continguts en l'ordre indicat, sense saltar-se cap apartat, i quan es fa referència a algun material web, adreçar-s'hi i dur-lo a la pràctica. Els programes d'exemple que incorpora el material, cal analitzar-los amb deteniment i posar-los en execució amb els valors que es proposen i amb altres valors que pugui intuir l'alumne per comprovar-ne el correcte funcionament. Una vegada estudiats a fons els programes d'exemple és necessari desenvolupar les activitats web.

Que us sembla? Comencem!

1. BD-XML natives. API Java específica del SGBD.

L'XML és, segons el seu impulsor 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 seva utilització en la web.

L'XML, avui en dia, és un dels formats més utilitzats per 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'intercanviï en format XML ha implicat l'aparició de mecanismes que permetin enregistrar dita informació en format XML, de manera que no sigui 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 les avantatges i els inconvenients d'utilitzar les implementacions basades en les diverses estratègies.

Les bases de dades natives XML han estat el resultat d'una de les estratègies per a emmagatzemar XML. Ens interessa, d'elles, refrescar les seves principals característiques per poder atacar el nostre objectiu, que no és altre que desenvolupar aplicacions que gestionin la informació emmagatzemada en elles.

1.1. Estratègies d'emmagatzematge d'XML

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

Per enfrontar-nos a la necessitat d'emmagatzematge i poder definir la millor estratègia cal fer una petita reflexió sobre els tipus de documents XML que ens podem trobar:

- **Documents centrats en dades** (*data centric*), pensats per l'intercanvi entre plataformes. Solen ser documents amb estructures regulars i ben definides. Les dades que transmeten són partícules atòmiques ben definides. En moltes ocasions les dades

són actualitzables. Acostumen a tenir com origen una base de dades i, en conseqüència, no és gaire important la seva persistència com a document XML.

Així, per exemple, un document XML centrat en les dades podria ser:

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

- **Documents centrats en el document** (*document centric*), amb una estructura irregular. L'origen i el destí acostuma a estar en les persones i solen estar fets a ma. N'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 ,...

Exemple de document XML centrat en el document

```
<Product>

<Intro>
The <ProductName>Turkey Wrench</ProductName> from
<Developer>Full Fabrication Labs, Inc.</Developer> is
<Summary>like a monkey wrench, but not as big.</Summary>
</Intro>

<Description>

<Para>The turkey wrench, which comes in <i>both right-
and left-handed versions (skyhook optional)</i>, is made
of the <b>finest stainless steel</b>. The Readi-grip
rubberized handle quickly adapts to your hands, even in
the greasiest situations. Adjustment is possible through
a variety of custom dials.</Para>

<Para>You can:</Para>

<List>
<Item><Link URL="Order.html">Order your own turkey
wrench</Link></Item>
<Item><Link URL="Wrenches.htm">Read more about
wrenches</Link></Item>
<Item><Link URL="Catalog.zip">Download the
catalog</Link></Item>
</List>

<Para>The turkey wrench costs <b>just $19.99</b> and, if
you order now, comes with a <b>hand-crafted shrimp
hammer</b> as a bonus gift.</Para>

</Description>

</Product>
```

> Observem que es tracta d'un document descriptiu d'un producte (llibre) en el que l'autor ha inserit diverses marques per a facilitar-ne la consulta i la formatació en un navegador.
> Bourret, R. (2005) XML and Databases. Consultat el 30/01/2012. http://www.rpbourret.com/xml/XMLAndDatabases.htm

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 tingui 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-nos en quina tècnica és més adequada segons la tipologia de documents a emmagatzemar i la gestió que en pretenguem.

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

1. Enregistrament directe 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. Enregistrament en un dels tipus de bases de dades tradicional: relacional, jeràrquic u orientat a objectes

   Aquesta opció obliga a una transformació del document XML cap el model que correspongui (relacional, jeràrquic u orientat a objectes), de manera que les dades s'emmagatzemen segons el model del SGBD.

3. Enregistrament en una base de dades XML nativa.

   Aquesta opció permet enregistrar 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 l'emmagatzematge d'XML.

> Una **base de dades XML** nativa és una base de dades dissenyada especialment per l'emmagatzematge d'XML. L'abreviatura en català és **BD-XML nativa**. L'abreviatura anglesa és **NXD.**

Com a regla general, els documents de tipologia *data-centric* s'emmagatzemen en una base de dades tradicional (relacional, orientada a objectes o jeràrquica). Això es pot fer per mitjà d'eines de tercers o per les capacitats incorporades a la pròpia base de dades. En aquest últim cas, la base de dades es diu que està habilitada per a XML (*XML- enabled*). En canvi, els documents de tipologia *document-centric* s'emmagatzemen en una base de dades XML nativa o en un sistema de gestió de continguts.

Un sistema de gestió de continguts és una aplicació dissenyada per gestionar documents, construïda normalment damunt una base de dades nativa XML.

Aquestes regles no són absolutes. Per una banda, els documents *data-centric* -sobretot si es tracta de dades semi-estructurades- es poden emmagatzemar en bases de dades natives. Per altra, els documents *document-centric* poden ser emmagatzemats en bases de dades tradicionals en aquells casos en que es precisa poques funcionalitats específiques XML.

En el moment en que va aparèixer la necessitat d'emmagatzemar documents XML, l'emmagatzematge de dades estava centrat en els SGBD relacionals, jeràrquics u orientats a objectes i van sorgir dues tendències:

- Per una banda, els grans SGBD existents, van començar a introduir funcionalitats XML en els seus SGBD, donant lloc a l'aparició dels SGBD-XML habilitades.
- Altrament, 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, doncs les versions actuals de molts SGBD tradicionals (relacionals, jeràrquics 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).

1.2. SGBD-XML natives vers SGBD predecessors

Els SGBD predecessors dels SGBD-XML natives a tenir en compte són els relacionals, jeràrquics i orientats a objectes, existents abans de la irrupció, en el món de les bases de dades, dels SGBD-XML natives.

Per entendre l'aparició dels SGBD-XML natives és molt interessant veure quines possibilitats facilitava cadascun dels SGBD predecessors així com els seus inconvenients.

1.2.1. SGBD jeràrquics

Els SGBD jeràrquics (anomenats SGBDJ a partir d'ara) poden semblar una eina vàlida per emmagatzemar dades XML, degut a la naturalesa jeràrquica dels seus elements. Van aparèixer en els anys 60 i emmagatzemen els seus elements en una jerarquia de nodes (arbre). Cada node conté dades d'identificació i un conjunt de subnodes d'un cert tipus. L'accés a les dades és sempre molt predictible i, en conseqüència, els accessos i actualitzacions estan optimitzats. La sintaxis de les consultes és molt similar al llenguatge *XPath.* El principal problema d'intentar emmagatzemar dades XML en un SGBDJ és la

manca de flexibilitat en l'estructura de la base de dades, ja que no es pot modificar l'estructura de la base de dades sobre la marxa, cosa bàsica en documents XML.

1.2.2. SGBD relacionals

Els SGBD relacionals (anomenats SGBDR a partir d'ara) també podrien esdevenir un receptacle per dades XML. Tenim tres possibilitats bàsiques d'emmagatzematge d'XML en aquests SGBD.

1. Emmagatzematge del document XML complet, com a text, en una columna CLOB o BLOB

> Els BLOB (***B**inary **L**arge **Ob**jects*) és una col·lecció de dades binàries emmagatzemades en un SGBD com a una entitat simple. S'utilitzen normalment per emmagatzemar àudio, imatges o objectes multimèdia, tot i que també poden ser utilitzats per emmagatzemar textos. Els CLOB (***C**haracter **L**arge **Ob**ject*) és una col·lecció de dades textuals emmagatzemades en un SGBD com a una entitat simple. Es diferencien dels BLOB per què poden gestionar codificació de caràcters.

Aquesta estratègia és una bona opció quan el document XML conté informació estàtica que només serà modificada quan el document complet sigui reemplaçat. Aquesta és la primera opció que van facilitar els primers SGBDR per començar a donar suport a l'emmagatzematge XML.

L'emmagatzematge de dades XML seguint aquesta opció és simple d'implementar, ja que no es necessita efectuar cap transformació del document cap el model relacional, però presenta enormes deficiències a l'hora de realitzar recerques d'informació dins el document o intentar la indexació de dades.

> Si heu utilitzat algun SGBD ofimàtic, l'emmagatzematge del document XML complet en una columna CLOB o BLOB seria similar a l'opció d'incrustar un objecte, que acostumen a facilitar els SGBD ofimàtics.

2. Gestió (consulta i modificació) del document XML des de les eines del SGBDR, però emmagatzematge dins el sistema d'arxius del sistema operatiu.

Aquesta opció s'acostuma a utilitzar quan el nombre de documents XML és petit i difícilment la informació que conté és actualitzable (tot i que es pot permetre l'actualització). Presenta limitacions en escalabilitat, flexibilitat a l'hora de l'emmagatzematge i recuperació i, evidentment, deficiències de seguretat, ja que les dades resideixen fora de la base de dades com arxius externs.

> Si heu utilitzat algun SGBD ofimàtic, l'emmagatzematge del document XML en el sistema d'arxius seria similar a l'opció de vincular o enllaçar un arxiu del sistema d'arxius del sistema operatiu, que acostumen a facilitar els SGBD ofimàtics.

3. Transformació (mapatge) de l'estructura de dades del document XML cap el model relacional, per emmagatzemar les dades en taules.

> A l'apartat referències bibliogràfiques de la web hi trobareu l'enllaç a articles científics que versen sobre la transformació entre XML i el model relacional.

La idea principal d'aquesta opció és observar l'estructura del document XML, amb els elements i atributs que hi apareixen i efectuar la conversió al model relacional seguint un algorisme. La taula 1-1 presenta una versió simplificada de les conversions a efectuar.

Taula 1-1: Correspondència d'elements XML a elements relacionals

Esquema XML	**Esquema relacional**
Element	Taula
Atribut / Element imbricat	Columna
Atribut `ID`	Clau primària
`IDREF` / Element imbricat	Clau secundària
`#REQUIRED, #IMPLIED`	`NOT NULL, NULL`

Val a dir que, evidentment, l'observació cal efectuar-la sobre l'XSD o el DTD que valida els documents a mapar, per assegurar que el mapatge és correcte davant qualsevol instància de document XML que verifiqui l'XSD o el DTD.

> El DTD (*Document Type Definition*) d'un document XML és un document que defineix tant els elements del document XML com les relacions que es donen entre ells. L'XSD (*XML Schema Definition*) és posterior al DTD, està escrit en XML i permeten noves característiques (definir

tipus de dades, utilitzar espais de noms, definir intervals de valors pels atributs i els elements,...); en definitiva, l'XSD aporta un major potencial semàntic que el DTD.

Per il·lustrar aquest mapatge, observem el següent document XML de tipologia *data-centric,* per procedir a la generació del model relacional que ens permeti emmagatzemar les seves dades. Per simplificar el procés, analitzem directament el document, tot i que en un cas real caldria analitzar l'XSD o el DTD.

```
<?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>
```

Després d'una simple anàlisi del document, observem que ens calen dues taules (`ComandaVenda` i `ComandaVendaLínia`) per poder emmagatzemar la informació:

```
ComandaVenda(Número, Client, Data)
ComandaVendaLínia(Comanda, Número, Article, Quantitat,
Preu) on Comanda referencia ComandaVenda(Número)
```

Com podeu veure, es tracta d'una situació molt simple i no del tot documentada, doncs el document XML no aporta l'XSD ni el DTD. Així, no podem estar segurs dels camps identificadors a les taules, que suposadament serien el `Número` a la taula `ComandaVenda` i la parella `(Comanda,Número)` a la taula `ComandaVendaLínia`.

Tot i que el mecanisme sembla simple, no sempre és senzill fer aquesta conversió ja que el model relacional i l'XML parteixen de conceptes força diferents:

- El model relacional està basat en dades bidimensionals sense jerarquia ni ordre

mentre el model XML està basat en arbres jeràrquics on l'ordre és rellevant

- En un document XML hi pot haver dades repetides mentre el model relacional fuig de les repeticions
- Les relacions i les estructures dins dels documents XML no sempre són obvies
- 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?

Intenteu, per exemple, generar el model relacional per a un document XML de tipologia *document-centric*, com el següent:

```
<Product>

<Intro>
The <ProductName>Turkey Wrench</ProductName> from
<Developer>Full Fabrication Labs, Inc.</Developer> is
<Summary>like a monkey wrench, but not as big.</Summary>
</Intro>

<Description>

<Para>The turkey wrench, which comes in <i>both right-
and left-handed versions (skyhook optional)</i>, is made
of the <b>finest stainless steel</b>. The Readi-grip
rubberized handle quickly adapts to your hands, even in
the greasiest situations. Adjustment is possible through
a variety of custom dials.</Para>

<Para>You can:</Para>

<List>
<Item><Link URL="Order.html">Order your own turkey
wrench</Link></Item>
<Item><Link URL="Wrenches.htm">Read more about
wrenches</Link></Item>
<Item><Link URL="Catalog.zip">Download the
catalog</Link></Item>
</List>

<Para>The turkey wrench costs <b>just $19.99</b> and, if
you order now, comes with a <b>hand-crafted shrimp
hammer</b> as a bonus gift.</Para>

</Description>

</Product>
```

Bourret, R. (2005) XML and Databases. Consultat el 30/01/2012. http://www.rpbourret.com/xml/XMLAndDatabases.htm

El problema no només radica en aconseguir el model relacional, sinó també en poder reconstruir el document XML a partir de la informació emmagatzemada en taules.

1.2.3. SGBD orientats a objectes

La darrera opció d'emmagatzemar documents XML en SGBD predecessors dels SGBD-XML natives, passa per la utilització dels SGBD orientats a objectes (anomenats SGBDOO a partir d'ara).

En aquest cas, es tracta d'analitzar el document (millor dit, l'XSD o el DTD) i efectuar un mapatge XML – OO basat en interrelacions entre classes, tot dissenyant el diagrama de classes adequat.

> A l'apartat referències bibliogràfiques de la web hi trobareu l'enllaç a articles científics que versen sobre la transformació entre XML i el model orientat a objectes.

De forma molt simplificada, el mapatge XML – OO consisteix en que cada tipus d'element XML de més alt nivell es modela com una classe d'objectes i els atributs XML es modelen com propietats de les classes.

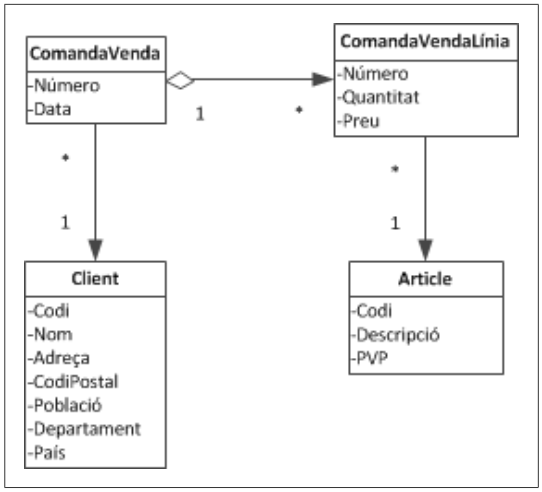
Per il·lustrar aquest mapatge, observem el següent document XML de tipologia *data-centric,* per procedir a la generació del model orientat a objecte que ens permeti emmagatzemar les seves dades. Per simplificar el procés, analitzem directament el document, tot i que en un cas real caldria analitzar l'XSD o el DTD.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Vendes>
   <ComandesVenda>
      <ComandaVenda>
         <Número>1234</Número>
         <Client>100</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>200</Client>
         <Data>30/10/2011</Data>
         <Línia Número="1">
            <Article>XQT301</Article>
            <Quantitat>14</Quantitat>
            <Preu>23.2</Preu>
         </Línia>
      </ComandaVenda>
   </ComandesVenda>
```

```
  <Articles>
      <Article Codi="PIL001">
         <Descripció></Descripció>
         <PVP>13.15</PVP>
      </Article>
      <Article Codi="CAT010">
         <Descripció></Descripció>
         <PVP>6.15</PVP>
      </Article>
      <Article Codi="XQT301">
         <Descripció></Descripció>
         <PVP>25.00</PVP>
      </Article>
   </Articles>
   <Clients>
      <Client Codi="100">
         <Nom>Indústries Fèrriques</Nom>
         <Adreça>Carrer Muntanya, 5</Adreça>
         <CodiPostal>08700</CodiPostal>
         <Població>Igualada</Població>
         <Departament>Barcelona</Departament>
         <País>ES</País>
      </Client>
      <Client Codi="200">
         <Nom>Eines del Berguedà</Nom>
         <Adreça>Gran Via, 36</Adreça>
         <CodiPostal>08600</CodiPostal>
         <Població>Berga</Població>
         <Departament>Barcelona</Departament>
         <País>ES</País>
      </Client>
   </Clients>
</Vendes>
```

Després de l'anàlisi del document, podem arribar a la conclusió que el diagrama de classes de la figura 1-1 ens podria servir per emmagatzemar la informació.

Figura 1. Diagrama de classes



Una vegada trobat el diagrama de classes adequat, cal efectuar-ne la implementació que pertoqui en el SGBD orientat a objectes. La mateixa tècnica es pot utilitzar per, a partir del diagrama de classes, efectuar-ne la implementació en un SGBD objecte-relacional.

1.2.4. SGBD-XML habilitades

El model de dades d'XML presenta característiques que l'apropen a qualsevol dels SGBD predecessors dels SGBD-XML natives (SGBDJ, SGBDR i SGBDOO). Les tres tipologies presenten punts a favor i en contra per a ser utilitzades en l'emmagatzematge de documents XML.

- El fet que els elements d'un document XML puguin contenir dades heterogènies, ens apropa als SBGDOO, però el problema apareix quan l'XSD o el DTD no permeten flexibilitat d'ubicació dels elements (és a dir, quan l'ordre és rellevant).
- La rellevància en la ubicació dels elements ens apropa als SGBDJ.
- Els SGBDR, tant estesos en l'actualitat, són adequats per a documents *data-centric* en els que no hi hagi gran complexitat.

Els grans fabricants de SGBD, davant la irrupció de l'XML, van realitzar esforços per incorporar informació XML en els seus productes, donant lloc als SGBD-XML habilitades. Sense voler deixar de banda els altres SGBD, potser els SGBDR-XML habilitades són els que, donada la seva gran estesa, han evolucionat més.

Les extensions XML que han anat incorporant els SGBDR han tingut crítiques que han constituït el punt de partida pel disseny dels SGBD-XML natives. Les crítiques inclouen:

- **Pèrdua d'estructura**. Si la jerarquia de dades XML és complexa, la seva conversió a un conjunt de taules produeix una gran quantitat d'aquestes o de columnes dins de cada taula amb valors nuls. Es poden relacionar les taules resultants per mantenir l'estructura jeràrquica d'XML, però sol ser un procés complicat per estructures de dades XML complexes.
- **Poca flexibilitat de les consultes**. És molt possible que es vulgui fer consultes sobre qualsevol element o propietat d'XML, però podria no ser possible si l'element no està inclòs en cap índex.
- **Poca velocitat d'accés**. Segons la manera en com les dades XML estan emmagatzemades físicament, pot resultar força lent recuperar-les en una estructura relacional.
- **Impossible utilització directa de tecnologies XML**, com les consultes *XPath, XSLT*, *XQL* o *XQuery*.

Tot i que aquestes crítiques han afavorit l'aparició dels SGBD-XML natives, cal tenir en compte que els grans fabricants de SGBD han continuat amb l'evolució de les funcionalitats XML que facilitaven els primers SGBD-XML habilitades, arribant al moment actual en el que faciliten prestacions de SGBD-XML natives. Això alimenta la controvèrsia: cal considerar-los SGBD-XML natives?

El SGBDR Oracle 11gR2 d'Oracle Corporation, en referència a la seva extensió XML, assegura emmagatzematge XML natiu:
Oracle XML DB is a feature of the Oracle Database. It provides a high-performance, native XML storage and retrieval technology. It fully absorbs the W3C XML data model into the Oracle Database, and provides new standard access methods for navigating and querying XML. With Oracle XML DB, you get all the advantages of relational database technology plus the advantages of XML.

Consultat el 06/02/2012.
http://www.oracle.com/technetwork/database/features/xmldb/index.html

El SGBDJ IMS d'International Business Machines Corporation (IBM), en referència a les característiques d'emmagatzematge i recuperació d'informació XML, deixa entreveure que facilita l'emmagatzematge XML natiu:
Because XML and IMS™ databases are both hierarchical, IMS is a natural database management system for managing XML documents. IMS allows you to easily receive and store incoming XML documents as well as compose XML documents from existing, legacy information stored that is in IMS databases. For example, you can:

- *Compose XML documents from all types of existing IMS databases, to support, for example, business-to-business on demand transactions and intra-organizational sharing of data.*
- *Receive incoming XML documents and store them in IMS databases. These databases can be legacy databases or new databases. XML documents are stored decomposed: the document is parsed and element data and attributes are stored in fields in segments as normal IMS data. This is appropriate for data-centric documents.*

You can store XML documents decomposed, intact, or in a combination of decomposed and intact. In decomposed storage mode, the incoming document is parsed and element data and attributes are stored in fields as normal IMS data. Decomposed storage is appropriate for data-centric documents. In intact storage, the incoming document, including its tags, is stored directly in the database without IMS being aware of its structure. Intact storage is appropriate for document-centric documents.
Consultat el 06/02/2012.
http://www-01.ibm.com/software/data/ims/imsjava/xmldb.html

El SGBDR DB2 d'International Business Machines Corporation (IBM), en referència a la seva extensió XML, assegura emmagatzematge XML natiu:
DB2 pureXML® offers sophisticated capabilities to store, process and manage XML data in its native hierarchical format. By integrating XML data intact into a relational database structure, users can take full advantage of DB2's relational data management features.
DB2 is unrivaled in its ability to manage both relational and XML data, enabling strong runtime performance and high levels of development time and cost savings.
Consultat el 06/02/2012.
http://www-01.ibm.com/software/data/db2/xml/

El SGBDR SQLServer de Microsoft Corporation assegura emmagatzematge XML natiu:
The storage options for XML in SQL Server include (...) native storage as xml data type:

> *The data is stored in an internal representation that preserves the XML content of the data. This internal representation includes information about the containment hierarchy, document order, and element and attribute values.*

Consultat el 06/02/2012.
http://msdn.microsoft.com/en-us/library/bb522493.aspx

1.2.5. SGBD-XML natives

Què és un SGBD-XML natives? En principi aquest terme va ser creat per fer referència als SGBD dissenyats especialment i única per a l'emmagatzematge de documents XML. Avui en dia, aquesta definició porta controvèrsia, donat que:

- No serien SGBD-XML natives aquells que, sent predecessors dels SGBD-XML natives (relacionals, jeràrquics i orientats o objectes), incorporen emmagatzematge XML natiu, és a dir, inclouen l'emmagatzematge en format XML.
- Podrien ser considerats SGBD, aquells que tot i facilitar l'emmagatzematge XML natiu, no compleixen les característiques ACID que ha de proporcionar tot SGBD.

El terme ACID en el món de les bases de dades

En el món de les bases de dades, el terme ACID és un acrònim de les característiques necessàries que ha de facilitar el SGBD per a que un conjunt d'instruccions puguin ser considerades una transacció. Aquestes característiques són atomicitat (***a**tomicity*), consistència **(***c**onsistency*), aïllament ***(i**solation)* i durabilitat (***d**urability*), que en llengua anglesa donen lloc a l'acrònim.
Atomicitat: Propietat que assegura que l'operació s'ha realitzat en la seva totalitat o no s'ha realitzat. Per tant, davant un error del sistema, no pot quedar a mitges.
Consistència: També anomenada integritat, és la propietat que assegura que no es perdrà mai la integritat de la base de dades.
Aïllament: Propietat que assegura que una operació no pot afectar a d'altres, és a dir, l'execució de dues transaccions sobre la mateixa informació siguin independents i no provoquin cap error.
Durabilitat: Propietat que assegura que una vegada executada una operació, aquesta serà persistent i no es podrà desfer en cap cas, ni davant una fallada del sistema.

Davant tot això, el concepte de SGBD-XML natives ha evolucionat i va més enllà de quedar definit per una única característica.

Un **SGBD-XML natives** és un sistema de gestió de la informació que ha de:

- Definir un model lògic per un document XML (en contraposició als SGBD 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.

- Incloure un nombre arbitrari de nivells de dades i complexitat.

- Permetre les tecnologies de consulta i transformació pròpies

d'XML: *XPath*, *XSLT*, *XQL*, *XQuery*,... com a vehicle principal de consulta i gestió.

- Permetre la introducció d'informació *data-centric*, *document-centric* i mixta.

El segon punt de la definició porta implícita la idea de 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 o un SGBDOO o un SGBDJ 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.

La realitat és que el camp de les bases de dades està dividit en dos: un, abanderat per les grans empreses de SGBD (Oracle Corporation, IBM Corporation, Microsoft Corporation) que argumenten que els seus productes (inicialment XML habilitats) han solucionat tots els problemes inicials i que faciliten emmagatzematge XML natiu; l'altre, defensat per les empreses que utilitzen un model físic de dades propi, ideat específicament per l'emmagatzematge XML, i que destaquen els problemes dels primers. Ben segur els dos tipus de SGBD conviuran durant molt de temps degut, entre altres raons, per la gran quantitat de dades que es troben emmagatzemades en SGBDR.

Donat, doncs, que en el moment actual poca diferència hi ha entre els inicialment anomenats SGBD-XML natives i les extensions XML que proporcionen els SGBD no XML, el terme “natiu/va” és àmpliament utilitzat per designar els productes especialitzats en solucions de BD-XML.

Funcionalitats usuals de les BD-XML natives

En el mercat trobem, actualment, molts productes SGBD-XML natives, però no tots faciliten les mateixes prestacions. Ens cal conèixer les principals característiques a avaluar en les BD-XML natives, per tal de poder escollir el producte que millor s'adapti a les nostres necessitats:

1. Emmagatzematge dels documents XML en col·leccions

   Les col·leccions juguen, en les BD-XML natives, el paper de les taules en les BDR. Els documents s'acostumen a agrupar, en funció de la informació que contenen, en col·leccions que, a la vegada, haurien de poder contenir altres col·leccions.

   No tots els SGBD-XML natives faciliten aquest mecanisme, de la forma que s'ha presentat i, a més, n'hi ha que confonen el concepte “col·lecció” amb el concepte “bases de dades”. En realitat, un SGBD-XML hauria de possibilitar gestionar diverses BD i cada BD hauria de possibilitar definir col·leccions i, a la vegada, cada col·lecció hauria de poder contenir altres col·leccions.

   A la realitat, ens trobem SGBD-XML que permeten aquesta organització, però n'hi ha que només permeten una única BD, la qual sí permet col·leccions; altres permeten vàries BD però no permeten col·leccions i, en aquest cas, haurem

d'utilitzar diferents BD si ens interessa organitzar la informació en col·leccions.

2. Validació de documents.

Caldria que el SGBD permetés la introducció d'XSD o DTD per validar els documents XML que gestionem a la BD.

3. Tecnologies de consulta i transformació pròpies d'XML: *XPath*, *XSLT*, *XQL*, *XQuery*,... com a vehicle principal de consulta i gestió.

Per definició, els SGBD-XML natives han d'incorporar aquestes tecnologies. Cal veure quines en facilita i la seva versió.

4. Estratègies d'actualització

Evidentment és molt interessant poder actualitzar els documents XML d'una BD-XML nativa. Els actuals SGBD-XML natives permeten, sense cap problema, l'actualització d'un document XML via substitució total. Però cal anar més lluny, i veure la possibilitat d'actualitzar (inserir, modificar, eliminar) dades dels documents XML sense haver de substituir el document sencer.

En aquest punt els SGBD-XML natives estan treballant per assolir millors prestacions, que passen per implementar algun dels llenguatges de modificació de les dades dels documents XML que han anat sorgint:

- Llenguatge *XUpdate,* promogut pel grup XML:DB (any 2000)
- Llenguatge *Update* promogut per Patrick Lehti (any 2001)
- Llenguatge *XQUF* promogut per W3C (any 2011)

5. Indexació XML

La indexació en els SGBD-XML natives és un dels punts febles a tenir en compte a l'hora d'avaluar el SGBD.

En el món de les BDR, a l'hora de definir el model relacional de les dades, es determina quins són els índexs adequats per a que les consultes més emprades siguin eficients. En les BD-XML natives no és tan fàcil, per què qualsevol estructura present en XML pot formar part d'una consulta. Es podria fer una indexació completa de tots els elements presents en una BD-XML nativa per aconseguir les consultes més ràpides, però això porta dos problemes: en primer lloc, la sobrecàrrega necessària per mantenir els índex portaria a actualitzacions molt lentes i, en segon lloc, està el fet que no es coneix quines etiquetes i atributs tindrà un document XML abans d'introduir-lo a la base de dades.

Donat que l'estructura d'un document XML i la naturalesa de les consultes és impredictible, és mot difícil planificar la indexació per endavant. Per tant, les BD-XML natives utilitzen tècniques d'indexació adaptatives (depenent del fabricant del SGBD) com, per exemple, una combinació d'indexació controlada per l'administrador de la BD-XML nativa, junt amb tècniques avançades de recuperació de textos d'alt rendiment.

6. Interfície d'usuari

És molt interessant que el SGBD aporti una (com a mínim) interfície d'usuari agradable i intuïtiva, que faciliti una ràpida utilització. En els moments actuals, és d'agrair l'existència d'una interfície web que no precisi de cap instal·lació en una màquina client.

7. Protocols d'accés

Un altre punt interessant a tenir en compte és conèixer els protocols d'accés que permet el SGBD, ja que ens facilitaren diferents mecanisme de connectivitat amb el SGBD per tal de poder accedir a la informació que emmagatzemen les BD-XML natives.

Entre els protocols d'accés actuals típics de trobar en els SGBD-XML natives, tenim WebDAV, REST, SOAP i XML-RPC.

Protocols d'accés

Web-based Distributed Authoring and Versioning (**WebDAV**), és un conjunt de mètodes basats en el protocol HTTP que proporciona funcionalitats per crear, canviar i moure documents en un servidor remot (típicament un servidor web). S'utilitza sobretot per permetre l'edició dels documents que serveix un servidor web, però també es pot aplicar a sistemes d'emmagatzematge generals basats en web, que puguin ser accedits des de qualsevol lloc.
La majoria de sistemes operatius moderns proporcionen suport per WebDAV, fent que els arxius d'un servidor WebDAV apareguin emmagatzemats en una unitat local. Tot i que va néixer per controlar també les versions dels documents, aquesta funcionalitat no s'ha implementat i, probablement, haurien de canviar-li el nom i deixar-lo com a WebDA.

Representational State Transfer (REST) és un estil d'arquitectura del programari per sistemes hipermèdia distribuïts que permet obtenir continguts (informació) des d'un lloc web mitjançant la lectura d'una pàgina web (protocol HTTP) que conté codi XML que descriu i inclou la informació. REST es basa únicament en la utilització del protocol HTTP i el llenguatge XML i no necessita de cap protocol d'intercanvi de missatges.
Així, per exemple, REST podria ser utilitzat per un editor en línia per posar a disposició dels subscriptors continguts sindicats. Periòdicament, l'editor prepararia i activaria una pàgina web que inclouria els continguts amb l'XML descriptor dels continguts. Els subscriptors només haurien de conèixer l'adreça URL de la pàgina, on hi accedirien amb un navegador web, obtindrien la informació i la podrien formatar i usar adequadament per als seus propòsits.

REST no és un estàndard reconegut pel W3C, però és àmpliament utilitzat.
Simple Object Access Protocol (SOAP) és un protocol estàndard (sota la tutela del W3C) que defineix com dos objectes en diferents processos es poden comunicar via intercanvi de dades XML. És molt habitual la seva utilització per accedir a serveis web (*web services*).
SOAP té diferents tipus de missatges, però els que més es fan servir són els que segueixen el patró de crida remota a aplicacions (RPC - Remote Procedure Call) on el client fa una petició (*request*) al servidor i aquest respon immediatament amb un missatge (*response*) que conté la resposta a la petició del client. SOAP no deixa de ser una evolució del protocol de comunicació XML-RPC.
XML-RPC és un protocol RPC (Remote Procedure Call) que utilitza XML per codificar les crides i el protocol HTTP com a mecanisme de transport de les dades.

---

8. Interfícies de programació d'aplicacions

Un darrer apartat que cal considerar a l'hora de decantar-nos per un SGBD-XML natives és conèixer quines API facilita. Normalment els SGBD faciliten una API pròpia i simple (baix nivell) amb la que podrem interactuar amb llenguatges com Java i C++.

A banda de l'API pròpia de cada SGBD, interessa saber si hi ha implementacions de les API estàndards per accés a BD-XML natives, com són XML:DB i XQJ.

API estàndards per accés a BD-XML natives

XML:DB, també anomenada **XAPI**, és una API ideada pel grup XML:DB, iniciativa apareguda l'any 2000 per intentar aconseguir un mecanisme estàndard d'accés a les BD-XML natives, de manera similar al mecanisme JDBC per a les BDR, davant els mecanismes propietaris que cada SGBD-XML natives es veia obligat a dissenyar. El grup està inactiu des del 2003.

XQuery API for Java (XQJ) és una interfície de programació d'aplicacions Java pensada per utilitzar el llenguatge *XQuery* (i també el llenguatge *XUpdate*) per obtenir informació de BD-XML natives, de manera similar a com JDBC és una API pensada per utilitzar el llenguatge SQL per accedir a BDR.
XQJ va néixer el 2003 i la seva versió definitiva ha estat publicada en el 2009. També és coneguda com JSR 225 doncs 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.

Avantatges i inconvenients dels SGBD-XML natives

Les avantatges que s'adjudiquen les BD-XML natives, en comparació a les BD no XML, són:

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

Per contra, els inconvenients que s'associen als SGBD-XML natives, són:

- La gran quantitat d'espai necessari per emmagatzemar el propi document XML com a format de representació de la informació, degut a que les etiquetes poden suposar el 75% de la informació d'un document XML. I això és a totes llums 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 del temps necessari (d'ordre seqüencial a ordre logarítmic) 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 ben segur que s'anirà avançant en aquest camp fins deixar de ser inconvenients.

Estratègies d'emmagatzematge dels SGBD-XML natives

Els SGBD-XML natives no tenen cap model d'emmagatzematge físic subjacent concret i, en conseqüència, poden ser construïts sobre bases de dades relacionals, jeràrquiques, orientades a objectes o en formats d'emmagatzematge propietaris. Així doncs, podem trobar:

1. Emmagatzematge basat en text (arxius de text)

   Emmagatzemen el document XML sencer en forma de text i proporcionen alguna funcionalitat de bases de dades per accedir a ell.

   Apliquen, com a molt, tècniques de compressió per reduir l'espai d'emmagatzematge i mantenen índexs addicionals per augmentar l'eficiència d'accés a la informació.

   Es poden definir sobre BD o sistemes d'arxius:

   - Possibilitat simple: Emmagatzemar el document com un BLOB en una base de dades relacional o mitjançant un fitxer i proporcionar alguns índexs sobre el document que accelerin l'accés a la informació.
   - Possibilitat sofisticada: Emmagatzemar el document en un magatzem adequat amb índexs, suport per transaccions,...

2. Emmagatzematge basat en un model

   Defineixen un model de dades lògic (com DOM) per l'estructura jeràrquica dels documents XML i emmagatzemen els documents d'acord amb aquest model utilitzant el model d'emmagatzematge físic que es desitgi (mapatge a BDR, mapatge a BDOO, ...). Així tenim:

   - Possibilitat 1: Traduir el DOM al model relacional.
   - Possibilitat 2: Traduir el DOM al model OO
   - Possibilitat 3: Utilitzar un magatzem creat especialment per aquesta finalitat

3. Emmagatzematge desenvolupat específicament per la gestió de documents XML

Arribats a aquest punt, ens adonem que els SGBD-XML natives no deixen d'utilitzar, per al seu emmagatzematge, les estratègies que directament ens podem plantejar per emmagatzemar XML en SGBD no XML (mapatge al model relacional, mapatge al model orientat a objectes,...). L'avantatge d'utilitzar els SGBD-XML natives és que l'estratègia d'emmagatzematge que utilitzen no afecta (és transparent) als usuaris de la BD, els quals es limiten a treballar amb documents XML i amb els llenguatges XML que facilita el SGBD i mai han de pensar en efectuar cap tipus de mapatge.

Productes SGBD-XML natives en el mercat

Els SGBD-XML natives existents en el mercat són molts i presentar-los tots es converteix en una tasca difícil. Les taules 2 i 3, extretes de la web de Ronald Bourret (guru de la temàtica "XML i BD"), mostren un recull força exhaustiu dels principals SGBD que permeten emmagatzematge XML. La taula 1-2 presenta un recull de SGBD-XML natives, mentre que la taula 1-3 correspon al recull de SGBD-XML habilitades. Observeu que els SGBD Oracle, DB2 i SQLServer no són inclosos a la taula 1-2 però a la taula 1-3 hi formen part amb la indicació de que faciliten emmagatzematge XML natiu.

Taula 1-2. Relació de productes SGBD-XML natives

Product	Developer	License	DB Type
4Suite, 4Suite Server	FourThought	Open Source	Object-oriented
BaseX	University of Konstanz	Open Source	Proprietary
Berkeley DB XML	Oracle	Open Source	Key-value
DBDOM	K. Ari Krupnikov	Open Source	Relational
dbXML	dbXML Group	Open Source	Proprietary
Dieselpoint	Dieselpoint, Inc.	Commercial	None (indexes only)
DOMSafeXML	Ellipsis	Commercial	File system(?)
EMC Documentum xDB	X-Hive Corporation	Commercial	Proprietary. Relational through JDBC
eXist	Wolfgang Meier	Open Source	Proprietary
eXtc	M/Gateway Developments Ltd.	Free	Post-relational
Extraway	3D Informatica	Commercial	Files plus indexes
Infonyte DB	Infonyte	Commercial	Proprietary
Ipedo XML Database	Ipedo	Commercial	Proprietary
Lore	Stanford University	Research	Semi-structured
MarkLogic Server	Mark Logic Corp.	Commercial	Proprietary
M/DB:X	M/Gateway Developments Ltd.	Free	Hierarchical
MonetDB/XQuery	CWI Database Group	Open Source	Proprietary
myXMLDB	Mladen Adamovic	Open Source	MySQL
Natix	University of Mannheim	Free / non-commercial	Proprietary
ozone	ozone-db.org	Open Source	Object-oriented
Qizx	XMLMind	Commercial	Proprietary
Sedna XML DBMS	ISP RAS MODIS	Free	Proprietary
Sekaiju / Yggdrasill	Media Fusion	Commercial	Proprietary
SQL/XML-IMDB	QuiLogic	Commercial	Proprietary (native XML and relational)
Sonic XML Server	Sonic Software	Commercial	Object-oriented (ObjectStore). Relational and other data through Data Junction
Tamino	Software AG	Commercial	Proprietary. Relational through ODBC.

TeraText DBS	TeraText Solutions	Commercial	Proprietary
TEXTML Server	IXIASOFT, Inc.	Commercial	Proprietary
TigerLogic XDMS	Raining Data	Commercial	Pick
Timber	University of Michigan	Open Source (non-commercial only)	Shore, Berkeley DB
TOTAL XML	Cincom	Commercial	Object-relational?
Virtuoso	OpenLink Software	Commercial	Proprietary. Relational through ODBC
XediX TeraSolution	AM2 Systems	Commercial	Proprietary
Xindice	Apache Software Foundation	Open Source	Proprietary
xml.gax.com	GAX Technologies	Commercial	Proprietary
Xpriori XMS	Xpriori	Commercial	Proprietary
XQuantum XML Database Server	Cognetic Systems	Commercial	Proprietary
XStreamDB Native XML Database	Bluestream Database Software Corp.	Commercial	Proprietary
Xyleme Zone Server	Xyleme SA	Commercial	Proprietary

Font: http://www.rpbourret.com/xml/XMLDatabaseProds.htm. Consultat el 07/02/2012

Taula 1-3. Relació de productes SGBD-XML habilitades

Product	**Developer**	**License**	**DB Type**
Access 2007	Microsoft	Commercial	Relational
Cache	InterSystems Corp.	Commercial	Post-relational
DB2	IBM	Commercial	Relational, native XML
eXtremeDB	McObject	Commercial	Object-oriented
FileMaker	FileMaker	Commercial	FileMaker
FoxPro	Microsoft	Commercial	Relational
Informix	IBM	Commercial	Relational
Matisse	Matisse Software	Commercial	Object-oriented
MonetDB/SQL	CWI Database Group	Open Source	Relational
MySQL	Sun Microsystems	Open Source	Relational
Objectivity/DB	Objectivity	Commercial	Object-oriented
OpenInsight	Revelation Software	Commercial	Multi-valued
Oracle	Oracle	Commercial	Relational, native XML
Orient ODBMS	Orient Technologies	Open Source	Object-oriented
PostgreSQL	PostgreSQL Global Development Group	Open Source	Relational
RDM Embedded	Raima, Inc.	Commercial	Network, relational
RDM Server	Raima, Inc.	Commercial	Network, relational
Sentences	Lazy Software, Ltd.	Free	Associative
SQL Server	Microsoft	Commercial	Relational, native XML
Sybase ASE	Sybase	Commercial	Relational
UniData	IBM	Commercial	Nested relational
UniVerse	IBM	Commercial	Nested relational

Versant Object Database	Versant Corp.	Commercial	Object-oriented
ViewDS	eB2Bcom	Commercial	Proprietary (LDAP)

Font: http://www.rpbourret.com/xml/XMLDatabaseProds.htm. Consultat el 07/02/2012

A la web de Ronald Bourret d'on han estat extretes les taules 2 i 3, hi podeu trobar informació molt interessant sobre altres productes vinculats amb l'XML.

1.3. Llibreries Java específiques del SGBD-XML natives

Els diversos SGBD-XML natives faciliten llibreries per permetre l'accés al SGBD des dels llenguatges més populars (Java, C++, .NET, Perl, PHP,...), però no tots els SGBD faciliten les llibreries per a tots els llenguatges. Així doncs, per saber quines llibreries facilita cada SGBD, caldrà analitzar la seva documentació. Per altra banda, tenint en compte que Java és un dels llenguatges més utilitzats actualment, és força normal que la majoria de SGBD facilitin llibreries Java per poder desenvolupar aplicacions en Java que accedeixin a les BD-XML natives.

Les BD-XML natives són relativament modernes i, en els darrers anys, des del 2000, hi ha hagut diversos intents de generar un estàndard de connexió Java pels SGBD-XML natives. El primer intent va ser l'API XML:DB (o XAPI) promogut pel grup XML:DB entre els anys 2000 i 2003. Posteriorment, entre el 2003 i el moment actual, ha anat prosperant, com a segon intent, l'API XQJ, sota la tutela de JCP (*Java Community Process*). Molts SGBD-XML natives proporcionen llibreries XML:DB i/o XQJ, de manera que si desenvolupem aplicacions utilitzant aquestes llibreries, aquestes aplicacions podran accedir, indistintament, als diversos SGBD que proporcionen implementacions per aquestes API.

La utilització de les API estàndards facilita reutilització de les aplicacions per diferents SGBD, però constitueixen una capa més de programari, doncs normalment els SGBD faciliten una llibreria pròpia per accedir al SGBD i les API estàndard són una capa intermèdia entre la llibreria pròpia del SGBD i l'aplicació que utilitza les API estàndard. Pot ser interessant, doncs, conèixer l'API pròpia que facilita el SGBD. Concretament veurem les API Java que proporcionen els SGBD-XML *BaseX* i *Sedna*.

1.3.1. API Java del SGBD *BaseX*

> Als annexos de la web trobareu l'apartat "Introducció al SGBD-XML natives *BaseX*" amb les indicacions per instal·lar aquest SGBD i tenir-hi una primera presa de contacte.

El SGBD-XML natives *BaseX* proporciona un protocol client/servidor que permet escriure clients en diversos llenguatges de programació. El desenvolupament de clients per accedir a *BaseX* sobrepassa l'objectiu d'aquest apartat, on simplement volem utilitzar el client Java que se'ns facilita.

> A la pàgina web oficial de *BaseX*, a l'apartat Desenvolupament, es facilita el codi de clients per a diversos llenguatges. En concret, pot ser interessant fer una ullada al client Java (`BaseXClient.java`) que ve acompanyat d'exemples d'utilització.

Entre els clients que *BaseX* facilita per a diversos llenguatges de programació, a nosaltres ens podria interessar el client per a Java (`BaseXClient.java`). *BaseX*, però, també facilita una extensa API Java, la qual incorpora un client equivalent al facilitat per la classe `BaseXClient`.

Nosaltres utilitzarem les classes incorporades a la API Java que facilita la versió 7.1 de *BaseX* i que trobarem en diversos paquets inclosos en el fitxer `basex.jar` ubicat a la carpeta arrel d'on s'hagi efectuat la instal·lació del SGBD.

> La instal·lació de *BaseX* no incorpora la documentació (arxiu *javadoc)* d'aquesta API, la qual es pot trobar a la pàgina web oficial de *BaseX* i també als annexos de la web, a l'apartat "Material per desenvolupament sobre SGBD-XML natives *BaseX*".

L'API Java de *BaseX* facilita molts paquets i classes. En aquest moment ens interessa fer una ullada a les necessàries per desenvolupar programes clients del SGBD. En concret:

- Classe `org.basex.server.ClientSession`, que ens facilita el mecanisme per establir una sessió client amb un servidor *BaseX* i un seguit de mètodes per a executar accions en la sessió establerta:
  - Mètodes constructors, per establir una sessió client amb un servidor.
  - Mètode `close()` per tancar la sessió.
  - Mètode `create()` per a crear bases de dades

- Mètode `add()` per afegir un document a la base de dades oberta
- Mètode `replace()` per substituir un document a la base de dades oberta
- Mètode `query()` per a crear un objecte `ClientQuery` a partir d'una consulta emmagatzemada en una `String`, a punt de ser executada.
- Mètodes `execute()`, heretats de la classe abstracta `Session`, pensats per executar qualsevol de les ordres que admet *BaseX*. En concret, l'utilitzarem per obrir la base de dades amb la que vulguem treballar, doncs no es facilita cap mètode específic per aquesta tasca.
- Mètode `info()`, heretat de la classe abstracta `Session`, per a obtenir informació de la darrera ordre executada.

- Classe `org.basex.server.ClientQuery`, pensada per a executar consultes *XPath*, *XQuery* i *XQUF* a la base de dades activa. Per això ens facilita els mètodes:
  - Mètode `execute()`, que executa la consulta i retorna el resultat complet de la consulta.
  - Mètode `info()`, per a obtenir informació de la darrera consulta executada.
  - Mètode `close()`, per tancar la consulta.
- Classe `org.basex.core.BaseXException` per a gestionar algunes de les excepcions que llença *BaseX*.

> Podeu obtenir un llistat de les ordres que admet un servidor *BaseX*, tot posant en marxa la consola *BaseX Client* que el procés d'instal·lació deixa a l'arbre de programes, entrant-hi amb usuari i contrasenya `admin` i demanant ajuda via l'ordre `help`.

Cal tenir clara la diferència entre l'execució d'una `query` (*XPath*, *XQuery* i *XQUF*) i l'execució d'una ordre de consola de les que proporciona *BaseX*. El mètode `ClientSession.execute()` està pensat per a executar una ordre de consola, mentre que el mètode `ClientQuery.execute()` està ideat per a executar una `query`.

> Per executar qualsevol programa que utilitzi els paquets `org.basex.core` i `org.basex.core.cmd` haureu de tenir el fitxer `basex.jar` inclòs en el `CLASSPATH` actiu.

Amb tota aquesta informació, ja estem en condicions de desenvolupar alguns programes Java que interactuïn amb les bases de dades d'un servidor *BaseX*. Els exemples següents estan basats en el servidor *BaseX* instal·lat seguint les indicacions del material web.

> En el servidor *BaseX* instal·lat seguint les instruccions de l'apartat "Introducció al SGBD-XML natives *BaseX*" dels annexos de la web, hi tenim instal·lada una base de dades de nom `mondial`. En ella basarem els exemples desenvolupats en aquest apartat.

Exemple de connexió a un servidor *BaseX* per executar-hi una consulta senzilla.

El següent programa visualitza els noms dels continents emmagatzemats en el document `mondial.xml` de la base de dades `mondial`.

> Trobareu el fitxer `BX_Client01.java` dins el paquet de codi font a l'apartat "Material per desenvolupament sobre SGBD-XML natives *BaseX*" dels annexos del web.

```
/*
 * Programa: BX_Client01.java
 * Objectiu: Programa que mostri els noms dels continents emmagatzemats en
 *           el document "mondial.xml" de la base de dades "mondial"
 * Autor...: Isidre Guixà
 */
package proves;
import org.basex.server.ClientSession;
import java.io.IOException;

public final class BX_Client01 {
  // Amaguem el constructor per defecte. */
  private BX_Client01() { }

  public static void main(String[] args) {
    ClientSession session=null;
    try {
      // Obrir sessió:
      session = new ClientSession("localhost", 1984, "admin", "admin");
      // Aquest client de BaseX no facilita funcions per indicar en quina
      // BD treballar... Per tant, ens caldrà utilitzar la funció collection()
      // per recollir els documents de la base de dades que ens interessa
      // utilitzar i efectuar les consultes a partir d'ells.
      // A més, és adequat indicar el document sobre el qual consultar, per si
      // la base de dades conté més documents.
      // Preparem la instrucció a executar
      //    for $doc in collection('mondial')
      //    where document-uri($doc)='mondial.xml'
      //    return $doc//continent/@name
      String cad = "for $doc in collection('mondial') ";
      cad = cad + "where document-uri($doc) = \"mondial.xml\" ";
      cad = cad + "return $doc//continent/@name/string()";
      // Executem la consulta
      System.out.println("Executant consulta: "+cad);
      System.out.println(session.query(cad).execute());
    }
    catch (IOException ioe) {
      ioe.printStackTrace();
    }
    finally { /* Tanquem sessió en qualsevol cas */
      try {
          if(session != null) session.close();
      }
      catch(IOException ioe) {
          ioe.printStackTrace();
      }
    }
```

```
  }
}
```

L'execució d'aquest programa obté:

```
G:\>java -Dfile.encoding=cp850 proves.BX_Client01
Executant consulta: for $doc in collection('mondial') where document-uri($doc)
= "mondial.xml" return $doc//continent/@name
 Europe Asia America Australia/Oceania Africa
```

Exemple de connexió a un servidor *BaseX* per procedir a crear-hi una base de dades buida

El següent programa mostra com es pot crear una base de dades buida. Per a fer-ho s'utilitza el mètode `ClientSession.create()` que obliga a introduir un primer document XML a afegir a la base de dades. L'exemple mostra com utilitzar el mètode per crear la base de dades buida.

> Trobareu el fitxer `BX_Client02.java` dins el paquet de codi font a l'apartat "Material per desenvolupament sobre SGBD-XML natives *BaseX*" dels annexos del web.

```
/*
 * Programa: BX_Client02.java
 * Objectiu: Programa que efectua la creació d'una BD buida amb el nom
             indicat com argument en la crida d'execució del programa
 * Autor...: Isidre Guixà
 */
package proves;
import org.basex.server.ClientSession;
import org.basex.core.BaseXException;
import java.io.IOException;
import java.io.ByteArrayInputStream;

public final class BX_Client02 {
  // Amaguem el constructor per defecte. */
  private BX_Client02() { }

  /* El programa principal admet com a paràmetre el nom de la base de
   * dades a crear. Cal comprovar la no existència d'una base de dades
   * amb idèntic nom, doncs en la versió actual (7.1) de BaseX, la
   * creació d'una base de dades no comprova l'existència d'una base de
   * dades amb igual nom i sobreescriu la base de dades que pogués existir.
   */
  public static void main(String[] args) {
    if (args.length!=1) {
      System.out.println("Cal indicar el nom de la BD a crear");
      System.exit(1);
    }
    ClientSession session=null;
    try {
      // Obrir sessió:
      session = new ClientSession("localhost", 1984, "admin", "admin");
      // Procedim a comprovar si la base de dades existeix, tot obrint-la:
      boolean existeix=true;
      try {
        session.execute("open "+args[0]);
      }
      catch (BaseXException bxe) {
```

```
            existeix=false;
        }
        // Actuem segons la BD existeix o no existeix
        if (existeix) {
          System.out.println("Ja existeix una BD amb aquest nom.");
        }
        else {
          System.out.println("No existeix cap BD amb aquest nom. Procedim...");
          session.create(args[0],new ByteArrayInputStream("".getBytes()));
          // El mètode "create" obliga a introduir un primer document,
          // com a segon paràmetre del tipus InputStream.
          // Com que volíem crear una base de dades buida, li hem passat
          // un document buit, construït amb "".getBytes()

          // Per obtenir informació de la creació:
          System.out.println(session.info());
        }
      }
      catch (IOException ioe) {
        ioe.printStackTrace();
      }
      finally { /* Tanquem sessió en qualsevol cas */
        try {
            if(session != null) session.close();
        }
        catch(IOException ioe) {
            ioe.printStackTrace();
        }
      }
    }
}
```

L'execució d'aquest programa introduint el nom d'una base de dades ja existent, obté:

```
G:\>java -Dfile.encoding=cp850 proves.BX_Client02 mondial
Ja existeix una BD amb aquest nom.
```

L'execució d'aquest programa introduint el nom d'una base de dades inexistent, obté:

```
G:\>java -Dfile.encoding=cp850 proves.BX_Client02 novaBD
No existeix cap BD amb aquest nom. Procedim...
Database 'novaBD' created in 299.87 ms.
```

Per crear una base de dades (buida o no), a més del mètode `ClientSession.create()`, disposem de l'ordre `create database` que pot ser invocada pel mètode `ClientSession.execute()`. Només cal substituir el codi:

```
session.create(args[0],new ByteArrayInputStream("".getBytes()));
```

per:

```
session.execute ("create database "+args[0]);
```

Podeu obtenir un llistat de les ordres que admet un servidor *BaseX*, tot posant en marxa la consola *BaseX Client* que el procés d'instal·lació deixa a l'arbre de programes, entrant-hi amb usuari i contrasenya `admin` i demanant ajuda via l'ordre `help`.

Exemple de connexió a un servidor *BaseX* per executar-hi consultes sobre la base de dades `mondial`, a introduir per l'usuari

El següent programa facilita la possibilitat que l'usuari introdueixi qualsevol consulta admesa per *BaseX* sobre la base de dades `mondial`.

> Trobareu el fitxer `BX_Client03.java` dins el paquet de codi font a l'apartat "Material per desenvolupament sobre SGBD-XML natives *BaseX*" dels annexos del web.

```
/*
 * Programa: BX_Client03.java
 * Objectiu: Programa que permet executar qualsevol consulta introduïda
 *           per l'usuari, de la base de dades "mondial"
 * Autor...: Isidre Guixà
 */

package proves;
import org.basex.server.ClientSession;
import org.basex.server.ClientQuery;
import org.basex.core.BaseXException;
import java.io.*;

public final class BX_Client03 {
  // Amaguem el constructor per defecte. */
  private BX_Client03() { }

  private static String introduirInstruccio() {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    String instr="";
    System.out.println("Introdueixi la instrucció a executar...");
    System.out.println("Per finalitzar, introdueixi una línia buida (en
blanc):");
    try {
      String text;
      do {
        text = br.readLine();
        if (!(text.isEmpty())) instr=instr.concat(text+"\n");
      } while (!(text.isEmpty()));
    }
    catch (IOException e) {
       System.out.println("S'ha produït una excepció en la lectura de la
instrucció:");
       System.err.println(e);
    }
    return instr.trim();
  }

  public static void main(String[] args) {
    ClientSession session=null;
    try {
      // Demanem la instrucció a executar
      String cad = introduirInstruccio();
      if ("".equals(cad))
      {
        System.out.println("No ha introduït cap consulta.");
        System.exit(1);
```

```
      }
      // Obrir sessió:
      session = new ClientSession("localhost", 1984, "admin", "admin");
      // Obrim la base de dades "mondial"
      session.execute("open mondial");
      // Preparem la instrucció introduïda com a consulta:
      ClientQuery cq = session.query(cad);
      try
      {
        String resultat = cq.execute();
        System.out.println("Consulta executada:\n"+cad);
        System.out.println("\nResultats:");
        System.out.println(resultat);
        // Informació de la consulta executada
        System.out.println("\nInformació de la consulta executada:");
        System.out.println(cq.info());
      }
      catch (BaseXException bxe)
      {
        System.out.println("La instrucció introduïda o no és executable com"+
        " a consulta, o té algun error sintàctic.");
        System.out.println("Error reportat pel servidor:");
        bxe.printStackTrace();
      }
      // Tancament de la base de dades
      session.execute("close");
    }
    catch (IOException ioe) {
      ioe.printStackTrace();
    }
    finally { /* Tanquem sessió en qualsevol cas */
      try {
          if(session != null)
            session.close();
      }
      catch(IOException ioe) {
          ioe.printStackTrace();
      }
    }
  }
}
```

Comprovem el funcionament d'aquest programa davant diverses instruccions a introduir per l'usuari. L'execució en qualsevol cas és:

```
G:\>java -Dfile.encoding=cp850 proves.BX_Client03
Introdueixi la instrucció a executar...
Per finalitzar, introdueixi una línia buida (en blanc):
```

Vegem quina és la resposta davant les següents instruccions:

- Consulta simple (una sola línia)

```
//continent/@name/string()

Consulta executada:
//continent/@name/string()

Resultats:
 Europe Asia America Australia/Oceania Africa

Informació de la consulta executada:
Hit(s): 5 Items
Updated: 0 Items
Total Time: 122.48 ms
```

- Consulta complexa (instrucció FLWOR)

```
for $i in //continent
return $i/@name/string()

Consulta executada:
for $i in //continent
return $i/@name/string()

Resultats:
 Europe Asia America Australia/Oceania Africa

Informació de la consulta executada:
Hit(s): 5 Items
Updated: 0 Items
Total Time: 14.2 ms
```

- Intent d'execució d'una instrucció que no és consulta `(query)`

```
create database xxx

La instrucció introduïda o no és executable com a consulta, o té algun error
sintàctic.
Error reportat pel servidor:
org.basex.core.BaseXException: Stopped at line 1, column 7:
[XPST0003] Unexpected end of query: 'database xxx'.
        at org.basex.server.ClientQuery.exec(ClientQuery.java:93)
        at org.basex.server.ClientQuery.execute(ClientQuery.java:58)
        at proves.BX_Client03.main(BX_Client03.java:55)
```

- Enumerar els documents existents a la base de dades

> En el SGBD *BaseX*, els conceptes bases de dades i col·lecció són equivalents. *BaseX* ens facilita la funció `collection()` per fer referència a la base de dades oberta.

```
for $doc in collection()
return document-uri($doc)

Consulta executada:
for $doc in collection()
return document-uri($doc)

Resultats:
mondial.xml

Informació de la consulta executada:
Hit(s): 1 Item
Updated: 0 Items
Total Time: 23.9 ms
```

- Enumerar les bases de dades existents

> *BaseX* facilita la funció `db:list()` per obtenir un llistat de les bases de dades.

```
db:list()
```

```
Consulta executada:
db:list()

Resultats:
mondial

Informació de la consulta executada:
Hit(s): 1 Item
Updated: 0 Items
Total Time: 4.2 ms
```

- Llista de totes les bases de dades, amb els seus documents:

```
<collection nom="{$i}">
  {
  for $doc in collection($i)
  return <document>{document-uri($doc)}</document>
  }
</collection>

Consulta executada:
for $i in db:list()
return
<collection nom="{$i}">
  {
  for $doc in collection($i)
  return <document>{document-uri($doc)}</document>
  }
</collection>

Resultats:
<collection nom="mondial">
  <document>mondial.xml</document>
</collection>

Informació de la consulta executada:
Hit(s): 1 Item
Updated: 0 Items
Total Time: 79.79 ms
```

XQUF no facilita instruccions específiques per gestionar transaccions i s'entén que cada execució d'una instrucció XQUF és una transacció. En cas que vulguem executar en una transacció diverses instruccions, cal posar-les en seqüència, separades per una coma.

- Inserir nous nodes (seguint sintaxis XQUF)

```
insert node <institut nom="IOC">IOC</institut>
after //continent[@name="Asia"]

Consulta executada:
insert node <institut nom="IOC">IOC</institut>
after //continent[@name="Asia"]

Resultats:
```

```
Informació de la consulta executada:
Hit(s): 0 Items
Updated: 1 Item
Total Time: 991.29 ms
```

Podem comprovar, via *BaseX GUI*, que la inserció s'ha dut a terme.

- Canviar el valor de nodes (seguint sintaxis XQUF)

```
replace value of node //institut[@nom="IOC"]
with "Institut Obert de Catalunya"

Consulta executada:
replace value of node //institut[@nom="IOC"]
with "Institut Obert de Catalunya"

Resultats:

Informació de la consulta executada:
Hit(s): 0 Items
Updated: 1 Item
Total Time: 156.83 ms
```

Podem comprovar, via *BaseX GUI*, que la substitució s'ha dut a terme.

- Canviar nodes (seguint sintaxis XQUF)

```
replace node //institut[@nom="IOC"]
with
<cicle>
  <codi>DAM</codi>
  <nom>Desenvolupament d'aplicacions multiplataforma</nom>
</cicle>

Consulta executada:
replace node //institut[@nom="IOC"]
with
<cicle>
  <codi>DAM</codi>
  <nom>Desenvolupament d'aplicacions multiplataforma</nom>
</cicle>

Resultats:

Informació de la consulta executada:
Hit(s): 0 Items
Updated: 1 Item
Total Time: 32.58 ms
```

Podem comprovar, via *BaseX GUI*, que la substitució s'ha dut a terme.

- Reanomenar nodes (seguint sintaxis XQUF)

```
rename node //cicle as "cf"

Consulta executada:
rename node //cicle as "cf"

Resultats:

Informació de la consulta executada:
Hit(s): 0 Items
Updated: 1 Item
Total Time: 20.58 ms
```

Podem comprovar, via *BaseX GUI*, que el canvi de nom s'ha dut a terme.

- Eliminar nodes (seguint sintaxis XQUF)

```
delete node //cf

Consulta executada:
delete node //cf

Resultats:

Informació de la consulta executada:
Hit(s): 0 Items
Updated: 1 Item
Total Time: 29.3 ms
```

Podem comprovar, via *BaseX GUI*, que l'eliminació s'ha dut a terme.

Exemple de connexió a un servidor *BaseX* per executar-hi ordres a introduir per l'usuari

El següent programa facilita la possibilitat que l'usuari introdueixi qualsevol ordre admesa per *BaseX*.

> Trobareu el fitxer `BX_Client04.java` dins el paquet de codi font a l'apartat "Material per desenvolupament sobre SGBD-XML natives *BaseX*" dels annexos del web.

El programa permet introduir diverses instruccions multilínia, que han de finalitzar amb una línia buida (en blanc). Per finalitzar el programa cal introduir una instrucció buida (en blanc).

> Podeu obtenir un llistat de les ordres que admet un servidor *BaseX*, tot posant en marxa la consola *BaseX Client* que el procés d'instal·lació deixa a l'arbre de programes, entrant-hi amb usuari i contrasenya `admin` i demanant ajuda via l'ordre `help`.

```
/*
 * Programa: BX_Client04.java
 * Objectiu: Programa que permet executar qualsevol ordre introduïda
 *           per l'usuari
 * Autor...: Isidre Guixà
 */
package proves;
import org.basex.server.ClientSession;
import org.basex.server.ClientQuery;
import org.basex.core.BaseXException;
import java.io.*;

public final class BX_Client04 {
  // Amaguem el constructor per defecte. */
  private BX_Client04() { }

  private static String introduirInstruccio() {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    String instr="";
```

```
    System.out.println("Introdueixi la instrucció a executar...");
    System.out.println("Per finalitzar, introdueixi una línia buida (en
blanc)");
    System.out.println("Per finalitzar el programa, introdueixi instrucció
buida");
    try {
      String text;
      do {
        text = br.readLine();
        if (!(text.isEmpty())) instr=instr.concat(text+"\n");
      } while (!(text.isEmpty()));
    }
    catch (IOException e) {
       System.out.println("S'ha produït una excepció en la lectura de la
instrucció:");
        System.err.println(e);
    }
    return instr.trim();
  }

  public static void main(String[] args) {
    ClientSession session=null;
    try {
      // Obrir sessió:
      session = new ClientSession("localhost", 1984, "admin", "admin");
      while (true) {
        // Demanem la instrucció a executar
        String cad = introduirInstruccio();
        if ("".equals(cad)) break;
        try {
          // Executem la instrucció introduïda per l'usuari
          session.execute(cad);
          // Informació final del servidor
          System.out.println("\nInformació final del servidor:");
          System.out.println(session.info());
        }
        catch (BaseXException bxe)
        {
          System.out.println("La instrucció introduïda no és vàlida.");
          System.out.println("Error reportat pel servidor:");
          bxe.printStackTrace();
        }
      }
    }
    catch (IOException ioe) {ioe.printStackTrace();}
    finally { /* Tanquem sessió en qualsevol cas */
      try { if(session != null) session.close(); }
      catch(IOException ioe) {
          ioe.printStackTrace();
      }
    }
  }
}
```

Comprovem el funcionament d'aquest programa, tot introduint la seqüencia d'instruccions per aconseguir:

- Obrir la base de dades `mondial`
- Exportar la base de dades a un arxiu XML
- Crear una base de dades anomenada `ioc`
- Obrir la base de dades `ioc`
- Afegir a la base de dades l'arxiu abans assolit via exportació

- Tancar la base de dades

La seqüència d'execució és:

```
G:\>java -Dfile.encoding=cp850 proves.BX_Client04
Introdueixi la instrucció a executar...
Per finalitzar, introdueixi una línia buida (en blanc)
Per finalitzar el programa, introdueixi instrucció buida
open mondial

Informació final del servidor:
Database 'mondial' was opened in 3.15 ms.

Introdueixi la instrucció a executar...
Per finalitzar, introdueixi una línia buida (en blanc)
Per finalitzar el programa, introdueixi instrucció buida
export C:\mondial.xml

Informació final del servidor:
Database 'mondial' was exported in 539.49 ms.

Introdueixi la instrucció a executar...
Per finalitzar, introdueixi una línia buida (en blanc)
Per finalitzar el programa, introdueixi instrucció buida
create database ioc

Informació final del servidor:
Database 'ioc' created in 167.53 ms.

Introdueixi la instrucció a executar...
Per finalitzar, introdueixi una línia buida (en blanc)
Per finalitzar el programa, introdueixi instrucció buida
open ioc

Informació final del servidor:
Database 'ioc' was opened in 3.13 ms.

Introdueixi la instrucció a executar...
Per finalitzar, introdueixi una línia buida (en blanc)
Per finalitzar el programa, introdueixi instrucció buida
add C:\mondial.xml

Informació final del servidor:
Path "mondial.xml" added in 1447.69 ms.

Introdueixi la instrucció a executar...
Per finalitzar, introdueixi una línia buida (en blanc)
Per finalitzar el programa, introdueixi instrucció buida
close

Informació final del servidor:
Database 'ioc' was closed.

Introdueixi la instrucció a executar...
Per finalitzar, introdueixi una línia buida (en blanc)
Per finalitzar el programa, introdueixi instrucció buida

G:\>
```

Podem comprovar, via *BaseX GUI*, l'existència de la nova base de dades amb l'arxiu `mondial.xml` com a contingut.

1.3.2. API Java del SGBD Sedna

> Als annexos de la web trobareu l'apartat "Introducció al SGBD-XML natives *Sedna*" amb les indicacions per instal·lar aquest SGBD i tenir-hi una primera presa de contacte.

El SGBD-XML natives *Sedna* proporciona un protocol client/servidor que permet escriure clients en diversos llenguatges de programació. El desenvolupament de clients per accedir a *Sedna* sobrepassa l'objectiu d'aquest apartat, on simplement volem utilitzar el client Java que se'ns facilita.

Centrem-nos, doncs, en el client Java que facilita la versió 3.5 de *Sedna*, que resideix en el paquet `ru.ispras.sedna.driver` que trobareu en el fitxer `sednadriver.jar` ubicat a la carpeta `driver\java\lib` del directori on hagueu instal·lat *Sedna*. Així mateix, la documentació d'aquest paquet es troba a la carpeta `driver\java\javadoc` i només cal obrir l'arxiu `index.html` per accedir-hi des de qualsevol navegador. I si teniu curiositat per conèixer el codi font del paquet, també el teniu disponible a la carpeta `driver\java\src`.

> Per executar qualsevol programa que utilitzi el paquet `ru.ispras.sedna.driver`, haureu de tenir el fitxer `sednadriver.jar` inclòs en el `CLASSPATH` actiu.

Observarem que el paquet `ru.ispras.sedna.driver` proporciona un reduït conjunt de classes i interfícies. Per poder utilitzar un client amb un SGBD ens cal saber com establir i gestionar connexions i com executar-hi instruccions de consulta i actualització.

En primer lloc ens cal establir connexions amb servidors *Sedna.* Per això disposem del mètode `getConnection(...)` de la classe `DatabaseManager`:

```
static SednaConnection getConnection (
      java.lang.String urlString,
      java.lang.String dbName,
      java.lang.String login,
      java.lang.String password)
      throws DriverException
```

El paràmetre `urlString` ha de contenir el nom o la IP de la màquina que conté el servidor *Sedna* al que ens volem connectar. Pot contenir el port pel qual està escoltant el servidor *Sedna* (en format `ip:port` o `nom:port`) . En cas de no indicar cap port, s'intenta establir la connexió pel port 5050 (port utilitzat normalment per *Sedna*).

La resta de paràmetres són obvis: `dbName` per indicar el nom de la base de dades a la que ens volem connectar, `login` per indicar el nom de l'usuari i `password` per indicar la seva contrasenya.

Per a que la connexió s'estableixi sense problemes, cal que el servidor *Sedna* estigui engegat i escoltant pel port pel qual estem intentant establir la connexió i que la base de dades a la que ens volem connectar, estigui també engegada. El client Java de *Sedna* que estem estudiant, no proporciona cap mecanisme d'engegada i/o aturada del servidor ni de les bases de dades existents en el servidor i tampoc permet crear-hi ni eliminar-hi bases de dades.

Una vegada ja tenim una connexió establerta, disposarem d'un objecte `SednaConnection` que ens permetrà anar executant les diverses accions sobre la base de dades.

A nivell de connexió (interfície `SednaConnection`), ens cal saber els diversos mètodes de què disposem:

- Mètode `isClose()`, per saber si la connexió s'ha tancat o es manté oberta.
- Mètode `close()`, per tancar la connexió.
- Mètode `begin()` per obrir una transacció. És obligatori tenir una transacció oberta per poder executar consultes i actualitzacions sobre la base de dades.
- Mètode `rollback()` per tancar una transacció sense enregistrar cap dels canvis que s'hagin pogut efectuar durant la transacció.
- Mètode `commit()` per tancar una transacció enregistrant tots els canvis que s'hagin pogut efectuar durant la transacció.
- Mètode `createStatement()` per crear un objecte `SednaStatement` que ens permet executar consultes i actualitzacions sobre la base de dades.
- Mètode `setTraceOutput()` per activar o desactivar el funcionament de la funció *XQuery* estàndard `trace()`.

  Sedna suporta la funció estàndard `trace()` del llenguatge *XQuery* que permet efectuar un seguiment dels valors que el motor *XQuery* va processant. El client Java de *Sedna* mostra, per la sortida estàndard, els missatges que la funció `trace` genera, en els punts on s'hagi incorporat en el codi. Si es vol mantenir la funció `trace` en el codi però desactivar llur funcionalitat, el client Java de *Sedna* ens proporciona el mètode `setTraceOutput()` de la interfície `SednaConnection`.

- Mètode `setDebugMode()` per activar/desactivar el mode depuració.

  Sedna facilita el mètode `getDebugInfo()` de la classe `DriverException` per a obtenir informació quan una operació falla, sempre i quan la sessió tingui activat el mode depuració, cosa que es pot gestionar amb el mètode `setDebugMode` de la interfície `SednaConnection`.

- Mètode `setReadonlyMode()` per canviar el mode “només lectura” per a la següent transacció.

  Sedna crea les transaccions en mode actualització a no ser que s'indiqui “només lectura” amb el mètode `setReadonlyMode()` de la interfície `SednaConnection`. En cas que es tingui seguretat que en una transacció no s'efectuarà cap instrucció d'actualització de la base de dades, és millor activar el mode “només lectura” ja que es guanyarà efectivitat, doncs les operacions no es queden en espera si hi ha altres transaccions que estan modificant les mateixes dades.

La interfície `SednaStatement` ens proporciona diversos mètodes:

- Mètodes `execute()`, per executar consultes *XQuery* que produeixen un resultat i altres instruccions (actualitzacions, creació/eliminació de col·leccions, càrrega/eliminació de documents). El propi mètode ens retorna un valor booleà que ens indica el tipus d'instrucció executada:
  - `true`, indicador que s'ha executat una consulta i que podrem obtenir els resultats amb el mètode `getSerializedResult()`.
  - `false`, indicador que s'ha executat una instrucció que no és una consulta; en aquest cas obtindrem un error si intentem executar el mètode `getSerializedResult()`.
- Mètode `getSerializedResult()`, per obtenir el resultat (objecte `SednaSerializedResult`) del darrer mètode `execute()` sempre i quan correspongui a una consulta.
- Mètodes `loadDocument()`, per carregar documents a la base de dades.

Els programes Java que executen consultes sobre bases de dades XML han de poder rebre els resultats de les consultes per tal d'efectuar-ne la gestió que correspongui. En el cas dels resultats facilitats per servidor *Sedna*, haurem d'utilitzar els mètodes de la interfície `SednaSerializedResult`. Aquesta interfície únicament proporciona dos mètodes `next()`, que permeten anar iterant sobre l'objecte `SednaSerializedResult` obtingut amb el mètode `SednaStatement.getSerializedResult()`.

En els exemples de consultes que segueixen, donat que interessa visualitzar els resultats per la consola, acostumarem a utilitzar el següent mètode:

```
/* Mètode per mostrar resultats per la sortida estàndard
 * Retorna el nombre d'elements de la consulta
 */
  private static int mostrarResultats(SednaStatement st)
    throws DriverException {
    int comptador = 0;
    String item;
    System.out.println("\nResultats:\n");
    SednaSerializedResult pr = st.getSerializedResult();
    while ((item = pr.next()) != null) {
      comptador++;
      System.out.print(item);
      // No utilitzem "println" per què cada "item", a partir del 2n.
      // comença amb "\n" i això ja provoca el salt de línia
    }
    System.out.println();
    return comptador;
  }
}
```

Amb tota aquesta informació, ja estem en condicions de desenvolupar alguns programes Java que interactuïn amb les bases de dades d'un servidor *Sedna*. Els exemples següents estan basats en el servidor *Sedna* instal·lat seguint les indicacions del material web.

> En el servidor *Sedna* instal·lat seguint les instruccions de l'apartat "Introducció al SGBD-XML natives *Sedna*" dels annexos de la web, hi tenim instal·lada una base de dades de nom `mondial`. En ella basarem els exemples desenvolupats en aquest apartat.

Exemple de connexió a un servidor *Sedna* per executar-hi una consulta senzilla.

El següent programa visualitza els noms dels continents emmagatzemats a la base de dades `mondial`.

> Trobareu el fitxer `SE_Client01.java` dins el paquet de codi font a l'apartat "Material per desenvolupament sobre SGBD-XML natives *Sedna*" dels annexos del web.

```
/*
 * Programa: SE_Client01.java
 * Objectiu: Programa que mostri els noms dels continents emmagatzemats a
 *           la base de dades "mondial"
 * Autor...: Isidre Guixà
 */
package proves;
import ru.ispras.sedna.driver.*;

public final class SE_Client01 {
  // Amaguem el constructor per defecte. */
  private SE_Client01() { }

  public static void main(String[] args) {
    SednaConnection session=null;
    try {
      // Obrir sessió:
      session = DatabaseManager.getConnection
                ("localhost:5050","mondial","SYSTEM","MANAGER");
      // Obrir transacció
      session.begin();
      // Creem objecte SednaStatement per poder executar consultes
      SednaStatement st = session.createStatement();
      // Preparem la instrucció a executar
      //    doc('mondial.xml')//continent/name
      String cad = "doc('mondial.xml')//continent/name/text()";
      // Executem la consulta
      System.out.println("Executant consulta: "+cad);
      st.execute(cad);
      // Mostrem resultats.. No ens cal saber si és una consulta o
      // un altre tipus d'instrucció. Per això no recollim el valor
      // booleà que retorna el mètode execute.
      int nbRes=mostrarResultats(st);
      System.out.println("\nS'ha obtingut "+nbRes+" resultats.");
      // No ens cal fer "commit" doncs es tracta d'una consulta
    }
    catch (DriverException de) {
      de.printStackTrace();
    }
    finally { /* Tanquem sessió en qualsevol cas */
      try {
          if(session != null) session.close();
      }
      catch(DriverException e) {
          e.printStackTrace();
      }
    }
```

```
  }
```

L'execució d'aquest programa obté:

```
G:\>java -Dfile.encoding=cp850 proves.SE_Client01
Executant consulta: doc('mondial.xml')//continent/name/text()
Resultats:

Europe
Asia
Australia/Oceania
Africa
America

S'ha obtingut 5 resultats.
```

Exemple de connexió a un servidor *Sedna* per executar-hi una consulta `FLWOR` i comprovar el funcionament de la funció `trace()`.

El següent programa visualitza els noms dels països existents a la base de dades `mondial`, acompanyats de la seva població. Aprofitem aquest exemple per mostrar el funcionament de la funció `trace()` del llenguatge *XQuery* i la possibilitat d'activar-la o desactivar-la des d'un programa client Java.

> Trobareu el fitxer `SE_Client02.java` dins el paquet de codi font a l'apartat "Material per desenvolupament sobre SGBD-XML natives *Sedna*" dels annexos del web.

```
/*
 * Programa: SE_Client02.java
 * Objectiu: Programa que mostri els noms dels països emmagatzemats a
 *           la base de dades "mondial", junt amb la seva població
 * Autor...: Isidre Guixà
 */
package proves;
import ru.ispras.sedna.driver.*;

public final class SE_Client02 {
  // Amaguem el constructor per defecte. */
  private SE_Client02() { }

  /* El programa principal admet com a paràmetre la cadena "traceOn" o
   * "traceOff" per indicar si es vol activar o no la funcionalitat de
   * la funció trace.
   * En cas de no indicar res, la funcionalitat quedarà desactivada.
   */
  public static void main(String[] args) {
    if (args.length>0 && !("traceOn".equals(args[0])) &&
                         !("traceOff".equals(args[0]))) {
      System.out.println("En cas d'indicar paràmetre cal que sigui "+
                         "\"traceOn\" o \"traceOff\"");
      System.exit(1);
    }
    SednaConnection session=null;
    try {
      // Obrir sessió:
      session = DatabaseManager.getConnection
                ("localhost:5050","mondial","SYSTEM","MANAGER");
```

```
      if (args.length==0 || args[0].equals("traceOff"))
        // Desactivem l'execució de la funció trace
        session.setTraceOutput(false);
      // Obrir transacció
      session.begin();
      // Creem objecte SednaStatement per poder executar consultes
      SednaStatement st = session.createStatement();
      // Preparem la instrucció a executar
      //    for $i in trace(doc("mondial.xml")//country,'###')
      //    return concat($i/name[1],"-",$i/population)
      String cad = "for $i in trace(doc(\"mondial.xml\")//country,'###')";
      cad = cad + "\nreturn concat($i/name[1],\"-\",$i/population)";
      // Executem la consulta
      System.out.println("Executant consulta:\n"+cad);
      st.execute(cad);
      // Mostrem resultats.. No ens cal saber si és una consulta o
      // un altre tipus d'instrucció. Per això no recollim el valor
      // booleà que retorna el mètode execute.
      int nbRes=mostrarResultats(st);
      System.out.println("\nS'ha obtingut "+nbRes+" resultats.");
      // No ens cal fer "commit" doncs es tracta d'una consulta
    }
    catch (DriverException de) {
      de.printStackTrace();
    }
    finally { /* Tanquem sessió en qualsevol cas */
      try {
          if(session != null) session.close();
      }
      catch(DriverException e) {
          e.printStackTrace();
      }
    }
  }
}
```

L'execució d'aquest programa obté:

```
G:\>java -Dfile.encoding=cp850 proves.SE_Client02 traceOff
Executant consulta:
for $i in trace(doc("mondial.xml")//country,'###')
return concat($i/name[1],"-",$i/population)
Resultats:

Albania-3249136
Greece-10538594
Macedonia-2104035
Serbia-7379339
Montenegro-672180
Kosovo-1804838
...
Sao Tome and Principe-144128
Seychelles-77575

S'ha obtingut 238 resultats.
```

En aquest cas la funció `trace()` que forma part de la consulta *XQuery* no s'ha manifestat en el resultat, per què l'execució s'ha indicat amb `traceOff`.

Si repetim l'execució amb `traceOn`, obtenim un resultat similar a:

```
G:\>java -Dfile.encoding=cp850 proves.SE_Client02 traceOn
Executant consulta:
for $i in trace(doc("mondial.xml")//country,'###')
```

```
return concat($i/name[1],"-",$i/population)
Resultats:

### <country car_code="AL" area="28750"
...
</country>
Albania-3249136### <country car_code="GR" area="131940"
...
</country>
Greece-10538594### <country car_code="MK" area="25333"
...
</country>
Macedonia-2104035### <country ...
...
...
</country>
Seychelles-77575

S'ha obtingut 238 resultats.
```

Fixem-nos que com la funció `trace()` està aplicada sobre la consulta `doc("mondial.xml")//country`, amb prefix `"###"`, veiem per la consola el contingut de cada element `country` recuperat pel mètode `SednaSerializedResult.next()`, precedit pel prefix `###`, abans què puguem fer res amb aquest element. La utilització d'aquesta funció pot ajudar a depurar les consultes tot i que és més fàcil fer la depuració executant la consulta directament en una terminal de *Sedna* o des de la GUI *SednaAdmin* que no pas des de dins un programa Java.

Exemple de connexió a un servidor *Sedna* per executar-hi qualsevol instrucció (consulta i gestió de documents i col·leccions) a introduir per l'usuari

El següent programa facilita la possibilitat que l'usuari introdueixi qualsevol instrucció admesa per *Sedna* (consultes, actualitzacions, càrrega i eliminació de documents, creació i eliminació de col·leccions,...) sobre la base de dades `mondial`.

> Trobareu el fitxer `SE_Client03.java` dins el paquet de codi font a l'apartat "Material per desenvolupament sobre SGBD-XML natives *Sedna*" dels annexos del web.

```
/*
 * Programa: SE_Client03.java
 * Objectiu: Programa que permet executar qualsevol instrucció introduïda
 *           per l'usuari, a la base de dades "mondial"
 * Autor...: Isidre Guixà
 */

package proves;
import ru.ispras.sedna.driver.*;
import java.io.*;
import java.util.Vector;

public final class SE_Client03 {
  // Amaguem el constructor per defecte. */
  private SE_Client03() { }
```

```
  private static String introduirInstruccio() {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    String instr="";
    System.out.println("Introdueixi la instrucció a executar...");
    System.out.println("Per finalitzar, introdueixi una línia buida (en
blanc):");
    try {
      String text;
      do {
        text = br.readLine();
        if (!(text.isEmpty())) instr=instr.concat(text+"\n");
      } while (!(text.isEmpty()));
    }
    catch (IOException e) {
       System.out.println("S'ha produït una excepció en la lectura de la
instrucció:");
       System.err.println(e);
    }
    return instr;
  }

  public static void main(String[] args) {
    SednaConnection session=null;
    try {
      // Demanem la instrucció a executar
      String cad = introduirInstruccio();
      if ("".equals(cad))
      {
        System.out.println("No ha introduït cap consulta.");
        System.exit(1);
      }
      // Obrir sessió:
      session = DatabaseManager.getConnection
                ("localhost:5050","mondial","SYSTEM","MANAGER");
      // Obrir transacció
      session.begin();
      // Creem objecte SednaStatement per poder executar consultes
      SednaStatement st = session.createStatement();
      // Executem la consulta
      System.out.println("Executant instrucció:\n"+cad);
      boolean teResultats=st.execute(cad);
      // Mostrem resultats
      if (!teResultats)
        System.out.println("Instrucció executada");
      else {
        int nbRes=mostrarResultats(st);
        System.out.println("\nS'ha obtingut "+nbRes+" resultats.");
      }
      // Enregistrem els canvis produïts durant la transacció
      // Necessari per si s'ha executat instruccions "no consulta"
      session.commit();
    }
    catch (DriverException de) {
      de.printStackTrace();
    }
    finally { /* Tanquem sessió en qualsevol cas */
      try {
          if(session != null) session.close();
      }
      catch(DriverException e) {
```

```
            e.printStackTrace();
        }
      }
    }
}
```

Comprovem el funcionament d'aquest programa davant diverses instruccions a introduir per l'usuari. L'execució en qualsevol cas és:

```
G:\>java -Dfile.encoding=cp850 proves.SE_Client03
Introdueixi la instrucció a executar...
Per finalitzar, introdueixi una línia buida (en blanc):
```

Vegem quina és la resposta davant les següents instruccions:

- Consulta simple (una sola línia)

```
doc('mondial.xml')//continent/name

Executant instrucció:
doc('mondial.xml')//continent/name

Resultats:

<name>Europe</name>
<name>Asia</name>
<name>Australia/Oceania</name>
<name>Africa</name>
<name>America</name>

S'ha obtingut 5 resultats.
```

- Consulta complexa (instrucció FLWOR)

```
for $i in doc('mondial.xml')//continent
return $i/name

Executant instrucció:
for $i in doc('mondial.xml')//continent
return $i/name

Resultats:

<name>Europe</name>
<name>Asia</name>
<name>Australia/Oceania</name>
<name>Africa</name>
<name>America</name>

S'ha obtingut 5 resultats.
```

- Càrrega del document `G:\DOCENCIA\books.xml` amb nom `llibres.xml` dins l'arrel de la base de dades.

```
LOAD 'G:\DOCENCIA\books.xml' 'llibres.xml'

Executant instrucció:
LOAD 'G:\DOCENCIA\books.xml' 'llibres.xml'

Instrucció executada
```

Podem comprovar, via *SednaAdmin*, que la càrrega s'ha dut a terme.

- Intent de càrrega del document `G:\DOCENCIA\books.xml` amb nom `llibres.xml` (ja existent) dins l'arrel de la base de dades.

```
LOAD 'G:\DOCENCIA\books.xml' 'llibres.xml'
```

```
Executant instrucció:
LOAD 'G:\DOCENCIA\books.xml' 'llibres.xml'

ru.ispras.sedna.driver.DriverException: SEDNA Message: ERROR SE2001
Document with the same name already exists.
Details: llibres

        at ru.ispras.sedna.driver.NetOps.bulkLoad(NetOps.java:202)
        at
ru.ispras.sedna.driver.SednaStatementImpl.executeResponseAnalyze(SednaStatement
Impl.java:155)
        at
ru.ispras.sedna.driver.SednaStatementImpl.execute(SednaStatementImpl.java:114)
        at
ru.ispras.sedna.driver.SednaStatementImpl.execute(SednaStatementImpl.java:38)
        at proves.SE_Client03.main(SE_Client03.java:50)
```

- Crear una col·lecció anomenada `xxx`

```
create collection 'xxx'

Executant instrucció:
create collection 'xxx'

Instrucció executada
```

Podem comprovar, via *SednaAdmin*, que la creació s'ha dut a terme.

- Enumerar les col·leccions existents a la base de dades

> *Sedna* facilita el document de sistema `$collections` amb la llista de totes les col·leccions existents a la base de dades

```
doc('$collections')

Executant instrucció:
doc('$collections')

Resultats:

<collections>
  <collection name="$modules"/>
  <collection name="xxx"/>
</collections>

S'ha obtingut 1 resultats.
```

O també:

```
doc('$collections')//collection

Executant instrucció:
doc('$collections')//collection

Resultats:

<collection name="$modules"/>
<collection name="xxx"/>

S'ha obtingut 2 resultats.
```

O també:

```
doc('$collections')//collection/@name
```

```
Executant instrucció:
doc('$collections')//collection/@name

Resultats:

<name="$modules"/>
<name="xxx"/>

S'ha obtingut 2 resultats.
```

- Intent de creació de la col·lecció `xxx` (ja existent).

```
create collection 'xxx'

ru.ispras.sedna.driver.DriverException: SEDNA Message: ERROR SE2002
Collection with the same name already exists.
Details: xxx

        at
ru.ispras.sedna.driver.SednaStatementImpl.executeResponseAnalyze(SednaStatement
Impl.java:145)
        at
ru.ispras.sedna.driver.SednaStatementImpl.execute(SednaStatementImpl.java:114)
        at
ru.ispras.sedna.driver.SednaStatementImpl.execute(SednaStatementImpl.java:38)
        at proves.SE_Client03.main(SE_Client03.java:50)
```

- Reanomenar la col·lecció `xxx` a `ioc`

```
RENAME COLLECTION 'xxx' INTO 'ioc'

Executant instrucció:
RENAME COLLECTION 'xxx' INTO 'ioc'

Instrucció executada
```

Podem comprovar, via *SednaAdmin*, que el canvi de nom s'ha dut a terme.

> *Sedna* no proporciona, en l'actual versió, la possibilitat de reanomenar un document introduït a la base de dades.

- Càrrega del document `mondial` (el mateix que en el procés d'instal·lació i configuració de *Sedna* es va introduir a la base de dades) dins la col·lecció de nom `ioc`., amb nom `mondial.xml`.

```
LOAD '...\mondial.xml' 'mondial.xml' 'ioc'

Executant instrucció:
LOAD '...\mondial.xml' 'mondial.xml' 'ioc'

Instrucció executada
```

Podem comprovar, via *SednaAdmin*, que la càrrega s'ha dut a terme.

- Llistat de tots els documents existents a la base de dades

> *Sedna* facilita el document de sistema `$documents` amb la llista de totes les col·leccions amb els documents de cada col·lecció i

els documents ubicats a l'arrel de la base de dades.

```
doc('$documents')

Executant instrucció:
doc('$documents')

Resultats:

<documents>
  <document name="$db_security_data"/>
  <collection name="$modules"/>
  <collection name="ioc">
    <document name="mondial.xml"/>
  </collection>
  <document name="llibres.xml"/>
  <document name="mondial.xml"/>
</documents>

S'ha obtingut 1 resultats.
```

- Obtenir els documents de la col·lecció `ioc`

```
doc('$documents')//collection[@name='ioc']/document

Executant instrucció:
doc('$documents')//collection[@name='ioc']/document

Resultats:

<document name="mondial.xml"/>

S'ha obtingut 1 resultats.
```

- Eliminar el document `llibres.xml` de l'arrel de la base de dades

```
DROP DOCUMENT 'llibres.xml'

Executant instrucció:
DROP DOCUMENT 'llibres.xml'

Instrucció executada
```

Podem comprovar, via *SednaAdmin*, que l'eliminació s'ha dut a terme.

- Eliminar el document `mondial.xml` de la col·lecció `ioc`

```
DROP DOCUMENT 'mondial.xml' IN COLLECTION 'ioc'

Executant instrucció:
DROP DOCUMENT 'mondial.xml' IN COLLECTION 'ioc'

Instrucció executada
```

Podem comprovar, via *SednaAdmin*, que l'eliminació s'ha dut a terme.

- Eliminar la col·lecció `ioc`

L'ordre `DROP COLLECTION` elimina una col·lecció malgrat tingui documents, que també són eliminats.

```
DROP COLLECTION 'ioc'
```

```
Executant instrucció:
DROP COLLECTION 'ioc'

Instrucció executada
```

Podem comprovar, via *SednaAdmin*, que l'eliminació s'ha dut a terme.

- Inserir nous nodes (seguint sintaxis *Update* de *Sedna*)

```
UPDATE insert <institut nom="IOC">IOC</institut>
following doc("mondial.xml")//continent[name="Asia"]

Executant instrucció:
UPDATE insert <institut nom="IOC">IOC</institut>
following doc("mondial.xml")//continent[name="Asia"]

Instrucció executada
```

Podem comprovar, via *SednaAdmin*, que la inserció s'ha dut a terme.

- Canviar el valor de nodes (seguint sintaxis *Update* de *Sedna,* on cal canviar tot el node)

```
UPDATE replace $n in doc("mondial.xml")//institut[@nom="IOC"]
with
<institut nom="IOC">Institut Obert de Catalunya</institut>

Executant instrucció:
UPDATE replace $n in doc("mondial.xml")//institut[@nom="IOC"]
with
<institut nom="IOC">Institut Obert de Catalunya</institut>

Instrucció executada
```

Podem comprovar, via *SednaAdmin*, que la substitució s'ha dut a terme.

- Canviar nodes (seguint sintaxis *Update* de *Sedna*)

```
UPDATE replace $n in doc("mondial.xml")//institut[@nom="IOC"]
with
<cicle>
  <codi>DAM</codi>
  <nom>Desenvolupament d'aplicacions multiplataforma</nom>
</cicle>

Executant instrucció:
UPDATE replace $n in doc("mondial.xml")//institut[@nom="IOC"]
with
<cicle>
  <codi>DAM</codi>
  <nom>Desenvolupament d'aplicacions multiplataforma</nom>
</cicle>

Instrucció executada
```

Podem comprovar, via *SednaAdmin*, que la substitució s'ha dut a terme.

- Reanomenar nodes (seguint sintaxis *Update* de *Sedna*)

```
UPDATE rename doc("mondial.xml")//cicle on cf

Executant instrucció:
UPDATE rename doc("mondial.xml")//cicle on cf

Instrucció executada
```

Podem comprovar, via *SednaAdmin*, que el canvi de nom s'ha dut a terme.

- Eliminar nodes (seguint sintaxis *Update* de *Sedna*)

```
UPDATE delete doc("mondial.xml")//cf

Executant instrucció:
UPDATE delete doc("mondial.xml")//cf

Instrucció executada
```

Podem comprovar, via *SednaAdmin*, que l'eliminació s'ha dut a terme.

2. API Java estàndards per BD-XML natives

Les dades emmagatzemades a les BD han de poder ser accedides des d'aplicacions desenvolupades en diferents llenguatges i, per aquest motiu, els SGBD es veuen obligats a facilitar interfícies de programació pels llenguatges de programació més emprats. Així, els fabricants de SGBD, coneixedors de la tecnologia emprada 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 porta l'aparició d'un problema: l'API proporcionada per cada SGBD és pròpia i diferent de les API dels altres SGBD i, en conseqüència, les aplicacions desenvolupades queden lligades al SGBD i els programadors han de conèixer un munt d'API diferents, tantes com nombre de SGBD diferents hagin 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 *XQuery* API for Java (XQJ).

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

2.1. 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 similar a com JDBC és una API pensada per utilitzar el llenguatge SQL per accedir a BDR.

XQJ va néixer el 2003 i la seva versió definitiva ha estat publicada en el 2009. També és coneguda com JSR 225 doncs 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 accedeixin a SGBD-XML natives via XQJ amb la 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 en comprovarem la seva correcta execució en tres SGBD-XML natives: *eXist-db*, *BaseX* i *Sedna*. Per això necessitem tenir instal·lats els tres SGBD i disposar de les implementacions XQJ per a cada SGBD.

Als annexos de la web trobareu els apartats "Introducció al SGBD-XML natives ..." per als SGBD *eXist-db*, *BaseX* i *Sedna*, amb les indicacions per instal·lar aquest SGBD i tenir-hi una primera presa de contacte.
Als annexos de la web trobareu els apartats "Material per desenvolupament sobre SGBD-XML natives utilitzant l'API XQJ" on hi ha una llibreria XQJ, desenvolupada per Charles Foster, pels SGBD *eXist-db* i *Sedna*.

El SGBD *BaseX* (versió 7.1) porta inclosa una llibreria XQJ, que presenta algunes anomalies. En el moment de confeccionar aquests materials, els responsables de *BaseX* han informat que Charles Foster està desenvolupant una llibreria XQJ per a *BaseX*, similar a les desenvolupades per *eXist-db* i *Sedna.* Cal doncs, està atent a l'aparició d'aquesta llibreria per a utilitzar-la en lloc de la que incorpora *BaseX*.

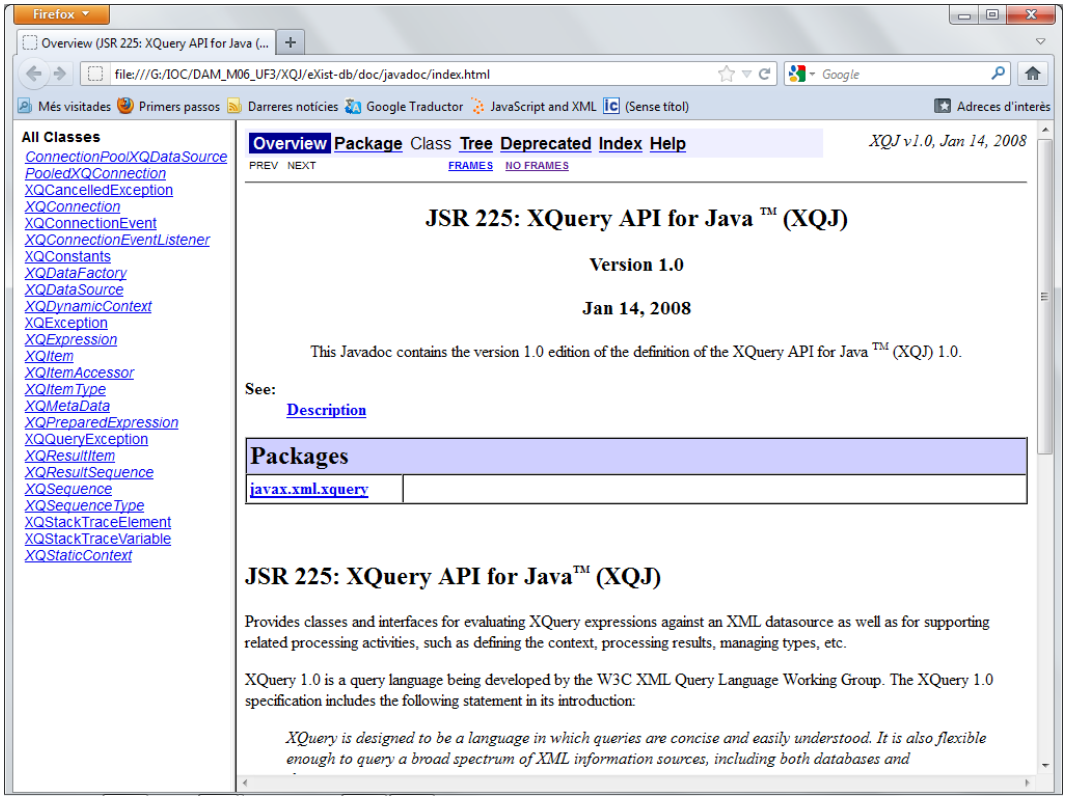
Si fem una ullada a la documentació (`javadoc` de la figura 2-1) de l'API XQJ, veurem que està constituïda per un gran nombre d'interfícies i unes poques classes. Això és així per què cada SGBD ha de proveir les classes que implementen les interfícies que dictamina l'API. En el desenvolupament d'aplicacions amb aquesta API utilitzarem sempre referències a aquestes interfícies i mai utilitzarem directament referències a les classes subministrades pel SGBD. D'aquesta manera aconseguirem dissenyar aplicacions que es puguin utilitzar per gestionar BD en diversos SGBD que implementen l'API XQJ.

Així doncs, ens interessa conèixer les interfícies que aporta l'API XQJ i emprar-les en el disseny d'aplicacions que accedeixin als SGBD-XML natives (concretament: *eXist-db*, *BaseX* i *Sedna).*

La documentació (`javadoc`) de l'API XQJ es pot trobar en els arxius zip corresponents a l'API XQJ per a *eXist-db* i *Sedna* i també

directament a la web del projecte JSR 225.

Figura 2-1. Documentació de l'API XQJ



2.1.1. Establiment de connexió

Les aplicacions que accedeixen a BD necessiten, com a primer pas per poder gestionar les dades de la BD, establir la connexió amb la BD a gestionar. L'API XQJ facilita dues interfícies apropiades per aconseguir aquest objectiu:

- `XQDataSource`, fàbrica per obtenir objectes `XQConnection`.
- `XQConnection`, per referenciar connexions (sessions) amb un SGBD específic.

Tota connexió s'assoleix, forçosament, a través d'un objecte `XQDataSource`.

Així, per obtenir una connexió cal seguir l'esquema:

```
XQDataSource xqs = new ClasseEspecíficaQueGeneraXQDataSource();
XQConnection con = xqs.getConnection(llistaParàmetres);
```

La primera línia del codi anterior mostra com crear un objecte `XQDataSource` específic del SGBD al que ens volem connectar. La segona línia mostra com establir la connexió i, com a llista de paràmetres, haurem d'incloure els adequats al SGBD.

Per cada SGBD-XML natives amb el que vulguem connectar via XQJ, ens cal saber la classe específica facilitada pel SGBD que implementa la interfície `XQDataSource`. La taula 2-1 recull les classes adequades pels SGBD amb els que practiquem.

Taula 2-1. Classes que implementen la interfície XQDataSource en diversos SGBD

SGBD	Classe
eXist-db	`net.xqj.exist.ExistXQDataSource`
BaseX	`org.basex.api.xqj.BXQDataSource`
Sedna	`net.xqj.sedna.SednaXQDataSource`

La classe de *BaseX* indicada a la taula 2-1 és la que porta integrada *BaseX* (versió 7.1) i que presenta algunes anomalies. En el moment de confeccionar aquests materials, els responsables de *BaseX* han informat que Charles Foster està desenvolupant una llibreria XQJ per a *BaseX*, similar a les desenvolupades per *eXist-db* i *Sedna*. Cal doncs, està atent a l'aparició d'aquesta llibreria per a utilitzar-la en lloc de la que incorpora *BaseX*.

Així doncs, si volem assolir un programa per connectar-nos amb un SGBD *eXist-db*, caldria escriure quelcom similar a:

```
XQDataSource xqs = new net.xqj.exist.ExistXQDataSource();
XQConnection con = xqs.getConnection(llistaParàmetres);
```

El codi anterior és codi específic per al SGBD *eXist-db*, i ens interessa, a ser possible, escriure codi que sigui independent, en el major grau possible, del SGBD. Per tant, ens interessa obtenir l'objecte `XQDataSource` de forma genèrica, sense cridar cap mètode específic de l'API subministrada pel SGBD. Per aconseguir això escriurem:

```
String driver = "nomClasseEspecíficaQueGeneraXQDataSource";
XQDataSource xqs = (XQDataSource)Class.forName(driver).newInstance();
XQConnection con = xqs.getConnection(llistaParàmetres);
```

En el codi anterior, la cadena `driver` ha de contenir el nom de la classe específica del SGBD que genera l'objecte `XQDataSource` (taula 2-1), però l'avantatge respecte la situació anterior és que el contingut de la cadena `driver` no té per què residir en el codi font del programa, sinó que es pot llegir d'un fitxer de configuració o es pot recollir via paràmetre, entre d'altres possibilitats.

Una vegada es disposa de l'objecte `XQDataSource` ja estem en condicions de crear un objecte `XQConnection` per establir la sessió de treball amb el SGBD i això ho aconseguim amb el mètode `getConnection()` de la interfície `XQDataSource`. L'API XQJ facilita tres sobrecàrregues del mètode `getConnection()`:

```
XQConnection getConnection(java.sql.Connection con)
                           throws XQException;
XQConnection getConnection() throws XQException;
XQConnection getConnection(java.lang.String username,
                           java.lang.String passwd)
                           throws XQException;
```

Les tres modalitats intenten establir una connexió contra el SGBD, de diferents maneres:

- La primera, utilitzant una connexió JDBC ja existent.
- La segona, sense aportar cap informació (paràmetre).
- La tercera opció, indicant l'usuari i la contrasenya (paràmetres).

Si hem treballat amb SGBD relacionals, estem acostumats a indicar, quan volem establir la connexió, entre d'altres paràmetres: la màquina (IP o nom), el port, la base de dades, l'usuari i la contrasenya. Observant les modalitats anteriors, estarem d'acord en que la segona i tercera opcions semblen incompletes, doncs no indiquem alguns dels paràmetres claus: màquina i port on resideix el SGBD i nom de la base de dades. A la segona opció, fins i tot, no s'indiquen usuari i contrasenya.

No desesperem! La solució passa per conèixer que la interfície `XQDataSource` facilita el mètode `setProperty()` que ens permet definir un seguit de propietats abans d'intentar establir la connexió amb el mètode `getConnection()`:

```
void setProperty(java.lang.String name,
                 java.lang.String value)
                 throws XQException;
```

Entre les propietats que permet establir el mètode `setProperty()`, hi trobem el nom de la màquina, el port, el nom de la base de dades, l'usuari i la contrasenya. No tots els SGBD-XML tenen les mateixes propietats o utilitzen idèntics noms per a una mateixa propietat. Ens cal saber, doncs, les propietats que cada SGBD admet, per poder-les assignar amb el mètode `setProperty()` abans d'establir la connexió amb el mètode `getConnection()` i per aconseguir-ho, la interfície `XQDataSource` facilita el mètode:

```
java.lang.String[] getSupportedPropertyNames();
```

El següent programa ens serveix per detectar les propietats suportades pels SGBD-XML *eXist-db*, *BaseX* i *Sedna*. Observeu, amb atenció, el requeriment per la correcta execució, incorporat a la capçalera del font.

> Trobareu el fitxer `XQJ01.java` dins el paquet de codi font a l'apartat "Material per desenvolupament sobre SGBD-XML natives utilitzant l'API XQL" dels annexos del web.

```
/*
 * Programa: XQJ01.java
 * Objectiu: Programa que llista les propietats suportades pel XQDataSource
 *           dels SGBD-XML exist-db, BaseX i Sedna
 * Autor...: Isidre Guixà
 */

/* Requeriment:
   Cal que els paquets on resideixen les llibreries XQJ facilitades pel SGBD
   resideixin en el CLASSPATH actiu. Els paquets a accedir són:
   * eXist-db: Directori "lib" de l'API XQJ per eXist-db, que conté els
               fitxers: xqjapi.jar, xqj2-0.0.1.jar i exist-xqj-1.0.1.jar
   * BaseX: Arxius basex.jar, xqj-api-1.0.jar i basex-api.jar instal·lats en
            procés d'instal·lació del SGBD (no es necessita cap llibreria
            addicional)
   * Sedna: Directori "lib" de l'API XQJ per Sedna, que conté els fitxers:
            xqjapi.jar, xqj2-0.0.1.jar i sedna-xqj-1.0.0.jar
*/
package proves;
```

```
import javax.xml.xquery.*;

public class XQJ01
{
  private static final String sgbd[] = {"eXist-db","BaseX","Sedna"};
  private static final String driver[] = {"net.xqj.exist.ExistXQDataSource",
                                          "org.basex.api.xqj.BXQDataSource",
                                          "net.xqj.sedna.SednaXQDataSource"};
  // Amaguem el constructor per defecte. */
  private XQJ01() { }

  public static void main(String[] args){
    for (int i=0; i<sgbd.length; i++){
      try {
        System.out.println("Propietats de XQDataSource en el SGBD "+sgbd[i]);
        XQDataSource xqs =
                    (XQDataSource)Class.forName(driver[i]).newInstance();
        String propietats[] = xqs.getSupportedPropertyNames();
        for (int j=0; j<propietats.length; j++)
          System.out.println("\t"+propietats[j]);
      }
      catch (ClassNotFoundException cf){
        System.out.println("No es troba la classe: "+cf.getMessage());
      }
      catch (InstantiationException ie){
        System.out.println("No es pot instanciar la classe: "+ie.getMessage());
      }
      catch (IllegalAccessException ia){
        System.out.println("Classe o constructor no accessibles: "+
                           ia.getMessage());
      }
    }
  }
}
```

Cal estar atents a l'aparició de la llibreria XQJ per *BaseX* similar a la ja existent per *eXist-dB* i *Sedna*, per a utilitzar-la enlloc de la que incorpora *BaseX* (versió 7.1).

L'execució del programa ens facilita la següent informació:

```
G:\>java -Dfile.encoding=cp850 proves.XQJ01
Propietats de XQDataSource en el SGBD eXist-db
        description
        logLevel
        loginTimeout
        serverName
        port
        user
        password
Propietats de XQDataSource en el SGBD BaseX
        user
        password
Propietats de XQDataSource en el SGBD Sedna
        description
        logLevel
        loginTimeout
        serverName
        databaseName
```

```
        port
        user
        password
        description
```

Analitzem les propietats suportades per l'objecte `XQDataSource` subministrat per cadascun dels tres SGBD indicats.

Comencem pel SGBD *BaseX*, doncs únicament facilita les propietats usuari i contrasenya. Com és que no ens permet indicar la màquina, el port i el nom de la base de dades? Doncs per què l'API XQJ subministrada per *BaseX* (versió 7.1) no permet la connexió client/servidor i, en conseqüència, només permet la connexió contra el servidor *BaseX* local, motiu pel que no es necessita indicar la màquina ni el port.

> *Note that we recommend everyone to use our own APIs, as they offer better performance and are better supported by our core team. The use of the `XML:DB` and `XQJ` APIs is discouraged: as these APIs do not utilize the client/server architecture of BaseX, their use may lead to conflicting database access operations.*
> *Documentació de BaseX (versió 7.1)*

Fixem-nos que *BaseX* tampoc ens permet indicar la base de dades amb la que ens connectarem. Recordem que BaseX no implementa el concepte de col·lecció i que en aquest SGBD, els conceptes "base de dades" i "col·lecció" són equivalents.

> Cal estar atents a l'aparició de la llibreria XQJ per *BaseX* similar a la ja existent per *eXist-dB* i *Sedna*, per a utilitzar-la enlloc de la que incorpora *BaseX* (versió 7.1). Aquesta llibreria, a banda de millorar el funcionament de l'actual, incorporarà l'arquitectura client-servidor.

Les propietats suportades pels SGBD *eXist-db* i *Sedna* són molt més similars. En ambdós vegem que podem indicar la màquina (IP o nom), el port, l'usuari i la contrasenya. *Sedna* ens permet, també, indicar el nom de la base de dades. El SGBD *eXist-db* permet gestionar connexions però hi ha una única base de dades i, per tant, les propietats per a establir connexió no permeten indicar-ne la base de dades. Dels tres SGBD emprats en aquest material, *Sedna* és l´únic que permet gestionar diverses bases de dades amb diverses col·leccions en cadascuna d'elles.

Una vegada conegudes les propietats suportades pels `XQDataSource` dels diversos SGBD, ja estem en condicions d'intentar establir connexió amb cadascun d'ells, emprant el mètode `XQDataSource.getConnection()`.

Aquest mètode retorna, si tot és correcte, un objecte que implementa la interfície `XQConnection`, el qual utilitzarem per executar qualsevol acció *XQuery* o *Update* sobre la BD. En finalitzar el programa cal recordar sempre de tancar la connexió amb el mètode `XQConnection.close()`, per tal d'alliberar tots els recursos assignats pel sistema operatiu per a mantenir la connexió.

La interfície `XQConnection` facilita mètodes per crear expressions sobre les que executarem sentències *XQuery/Update,* i els mètodes per validar `(commit())` o

retrocedir `(rollback())` els canvis duts a terme durant una transacció.

Per defecte, una connexió opera en mode `auto-commit` , fet que significa que cada instrucció *Update* és executada i validada en una transacció individual Aquesta forma de treballar es pot desactivar, amb el mètode `XQConnection.setAutoCommit().`, i en tal situació, la transacció finalitzarà efectuant una crida al mètode `commit()` o `rollback()`, donant lloc a l'inici d'una nova transacció. No hi ha, doncs, una instrucció específica per indicar l'inici de transacció.

El següent programa mostra com establir connexió amb cadascun dels tres SGBD indicats. Cal tenir present que aquest programa:

- Obliga a passar per paràmetre el nom del SGBD.
- Considera que l'usuari i la contrasenya per a cada SGBD són els que el procés d'instal·lació facilita per defecte (`admin/admin` per a *eXist-db* i *BaseX* i `SYSTEM/MANAGER` per *Sedna).*
- Pels SGBD *eXist-db* i *Sedna,* que són els SGBD que suporten l'arquitectura client/servidor, es considera que el servidor resideix a la mateixa màquina des d'on s'executarà el programa `(localhost)` i que el port pel que s'estableix la comunicació TCP/IP és el que el procés d'instal·lació facilita per defecte `(8080` per *eXist-db* i `5050` per *Sedna).*
- Pel SGBD *Sedna*, intenta establir connexió amb una BD de nom `mondial`. Si no es disposa d'aquesta BD, cal indicar el nom de BD que correspongui.

Trobareu el fitxer `XQJ02.java` dins el paquet de codi font a l'apartat "Material per desenvolupament sobre SGBD-XML natives utilitzant l'API XQL" dels annexos del web.

```
/*
 * Programa: XQJ02.java
 * Objectiu: Programa que mostra com establir connexió amb tres SGBD:
 *           eXist-db, BaseX i Sedna
 * Autor...: Isidre Guixà
 */

package proves;
import javax.xml.xquery.*;
import java.lang.Exception;

public class XQJ02
{
  // Amaguem el constructor per defecte. */
  private XQJ02() { }

  private static XQConnection establirConnexio(String[] args)
                 throws Exception
  /* Intenta establir una connexió contra el SGBD indicat per paràmetre, que
     ha de ser: eXist-db o BaseX o Sedna */
  {
    if (args.length<1){
      throw new Exception("Executeu el programa indicant, primer, el SGBD  "+
                          "al que connectar: eXist-db, BaseX o Sedna.");
    }
    XQDataSource xqs;
    if (args[0].equalsIgnoreCase("eXist-db")) {
```

```
      xqs = (XQDataSource)
          Class.forName("net.xqj.exist.ExistXQDataSource").newInstance();
      xqs.setProperty("serverName","localhost");
      xqs.setProperty("port","8080");
      xqs.setProperty("user","admin");
      xqs.setProperty("password","admin");
    }
    else if (args[0].equalsIgnoreCase("BaseX")) {
      xqs = (XQDataSource)
          Class.forName("org.basex.api.xqj.BXQDataSource").newInstance();
      xqs.setProperty("user","admin");
      xqs.setProperty("password","admin");
    }
    else if (args[0].equalsIgnoreCase("Sedna")) {
      xqs = (XQDataSource)
          Class.forName("net.xqj.sedna.SednaXQDataSource").newInstance();
      xqs.setProperty("serverName","localhost");
      xqs.setProperty("port","5050");
      xqs.setProperty("user","SYSTEM");
      xqs.setProperty("password","MANAGER");
      xqs.setProperty("databaseName","mondial");
    }
    else
      throw new Exception("Executeu el programa indicant, primer, el SGBD  "+
                          "al que connectar: eXist-db, BaseX o Sedna.");
    return xqs.getConnection();
  }

  public static void main(String[] args){
    XQConnection conn = null;
    try {
      conn = establirConnexio(args);
      System.out.println("Connexió establerta amb SGBD " + args[0]);
    }
    catch (Exception e) {
      e.printStackTrace();
    }
    finally { /* Tanquem connexió en qualsevol cas */
      try {
          if(conn != null) conn.close();
      }
      catch(XQException xe) {
          xe.printStackTrace();
      }
    }
  }
}
```

L'execució del programa finalitza amb error (convenientment documentat) si:

- No s'indica, a l'hora d'executar el programa, cap SGBD.
- S'indica, a l'hora d'executar el programa, un nom de SGBD no suportat.
- No es troba, en el `CLASSPATH` actiu, la llibreria XQJ corresponent al SGBD-XML natives amb el què es vol connectar.
- En cas del SGBD *Sedna*, el servidor està aturat o la base de dades indicada està aturada o el port és erroni.
- En cas dels SGBD *BaseX* i *Sedna*, l'usuari o la contrasenya són erronis.

El programa anterior no es queixa, pel SGBD *eXist-db,* si el servidor és aturat o el port o l'usuari o la contrasenya són erronis, i el corresponent error es posposa al moment en que s'intenta executar una instrucció *XQuery* o *Update* a través de la connexió. Aquest

funcionament és degut a que el SGBD *eXist-db* treballa amb connexions no persistents; en paraules de Charles Foster:

> *Sedna XQJ works over a persistent and stateful connection, so the XQDataSource.getConnection() will throw an Exception if a Connection is not established.*
> *However, the eXist XQJ works over none persistent or stateful HTTP.*
> *So yes, if a server's connection details are configured incorrectly, an XQException will only be thrown upon attempting to send an XQuery or XUpdate expression to eXist.*
> *With regards to being normal, or not normal. The eXist XQJ works in a similar fashion to the MarkLogic XQJ and this is also perfectly fine with regards to the XQJ 1.0 Specification.*

Així doncs, haurem de tenir present que en *eXist-db*, el fet que el mètode `XQDataSource.getConnection()` no llenci una excepció, no és garantia de que la connexió s'ha establert correctament.

Observem, també, que l'execució sobre *BaseX* (versió 7.1) ha de ser en local i no precisa que el servidor *BaseX* estigui engegat, doncs no utilitza l'arquitectura client/servidor.

2.1.2. Sentències *XQuery* d'execució immediata

Tots els SGBD acostumen a facilitar dos mecanismes d'execució de sentències contra la BD, utilitzant el llenguatge que correspongui (SQL en SGBDR i SGBDOR, OQL/OML en SGBDOO i *XQuery/Update* en SGBD-XML):

- Execució immediata d'una sentència, utilitzada quan la sentència s'ha d'executar una única vegada.
- Execució de sentències paramètriques o preparades, utilitzada quan la mateixa sentència (o sentència similar amb alguns valors diferents) s'ha d'executar repetidament i consistent en escriure la sentència com una plantilla que conté algunes variables que s'aniran substituint a cada execució.

L'API XQJ facilita els dos mecanismes a través de les interfícies:

- `XQExpression`, per a l'execució immediata de sentències
- `XQPreparedExpression`, per a l'execució de sentències paramètriques

Centrem-nos, en aquest moment, en l'execució immediata de sentències proporcionada per la interfície `XQExpression`.

Per executar una sentència de manera immediata, crearem un objecte `XQExpression` a partir del mètode `XQConnection.createExpression()`. Disposem de dues sobrecàrregues d'aquest mètode:

```
XQExpression createExpression() throws XQException;
XQExpression createExpression(XQStaticContext properties)
                              throws XQException;
```

En ambdues sobrecàrregues, s'obté un objecte `XQExpression` que podrem utilitzar per

executar sentències *XQuery/Update* de manera immediata. La segona sobrecàrrega permet indicar les propietats del context estàtic que es tindran en compte en avaluar l'expressió, mentre que la primera sobrecàrrega pren com a context estàtic l'associat a la connexió.

La interfície `XQStaticContext` proveeix un conjunt de mètodes `get` i `set` per recuperar i establir les propietats d'un context estàtic. La interfície `XQConnection` facilita els mètodes `getStaticContext()` i `setStaticContext()` per recuperar i establir el context estàtic de la connexió.

Context d'una expressió *XPath/XQuery/Update*
El context d'una expressió és tota aquella informació que pot incidir en el resultat de l'avaluació de la mateixa i pot ser de dos tipus: estàtic i dinàmic.
El context estàtic està constituït per tota la informació disponible durant l'anàlisi estàtic, és a dir, abans de l'execució de l'expressió, com per exemple, l'abast de les variables que apareixen dins l'expressió.
El context dinàmic està constituït per tota la informació disponible quan s'està executant l'expressió, per exemple, l'element que s'està avaluant actualment, els valors actuals de les variables, la data i hora actuals, etc.
Es pot establir una analogia dels conceptes "context estàtic i dinàmic" en les expressions *XPath* amb els conceptes "temps de compilació i d'execució" en els llenguatges de programació.

Una vegada tinguem l'objecte `XQExpression`, podrem a través d'ell executar consultes i altres ordres de manera immediata. Recordem que en *XQuery* cal distingir, com en SQL, les sentències "consulta" que poden retornar un conjunt de resultats que caldrà processar, de les sentències "no consulta" que permeten executar una ordre (inserció, eliminació o actualització i fins i tot ordres específiques del SGBD) de la qual se'ns informa, com a molt, de l'èxit o fracàs de la seva execució. Per aquest motiu, la interfície `XQExpression` distingeix els mètodes `executeQuery` per a les "consultes" dels mètodes `executeCommand` per a les "no consultes":

```
void executeCommand(java.lang.String cmd)throws XQException;
void executeCommand(java.io.Reader cmd) throws XQException;
XQResultSequence executeQuery(java.lang.String query)
                              throws XQException;
XQResultSequence executeQuery(java.io.Reader query)
                              throws XQException;
XQResultSequence executeQuery(java.io.InputStream query)
                              throws XQException;
```

A més dels mètodes anteriors, disposem del mètode `close()` per tancar l'expressió, alliberant tots els recursos associats, quan ja no sigui necessària. L'execució del mètode `close()` sobre la connexió, tanca totes les expressions definides sobre la connexió. També disposem del mètode `isClosed()` per poder esbrinar si una expressió està oberta o tancada:

```
void close() throws XQException;
boolean isClosed();
```

Fixem-nos que el mètode `executeQuery()` retorna, si tot va bé, un objecte `XQResultSequence`, interfície que deriva de la interfície `XQSequence`, la qual ens facilita un conjunt de mètodes per avaluar la seqüència de resultats obtinguts amb el mètode `executeQuery()`.

> Mireu la documentació de la interfície `XQSequence` per veure una llista completa dels mètodes que facilita.

La interfície `XQResultSequence` representa una seqüència d'elements seguint la definició de XDM (*XQuery 1.0 and XPath 2.0 Data Model*, de W3C). Una ullada a la documentació d'aquesta interfície ens mostra que disposem de mètodes per:

- Processar els seus elements: `next()`, `previous()`, `getItem()`, `getPosition()`, `count()`, `first()`, `last()`, `isFirst()`, `isLast()`, `isBeforeFirst()`, `isAfeterLast()` i un conjunt de mètodes `get` per obtenir el contingut d'un element en el format adequat (`getBoolean()`, `getByte()`, `getDouble()`,...).
- Obtenir una versió seriada de la seqüència com a un objecte `String` (mètode `getSequenceAsString()`).
- Obtenir una versió seriada de la seqüència (mètode `getSequenceAsStream()`) com a un objecte `XMLStreamReader` (interfície de l'API StAX de Java, que permet la iteració, cap endavant, d'un document XML en mode lectura, utilitzant els mètodes `next()` i `hasNext()`).

Segons sigui la gestió que haguem d'efectuar, utilitzarem uns o altres mètodes. Així, davant un programa de sentència oberta (en el que l'usuari pugui introduir la sentència a executar), només podrem pensar en mostrar el resultat a través de la versió seriada cap `String` o via objecte `Transformer` a partir de la seriació cap a `XMLStreamReader`. I davant un programa de sentència tancada (sentència perfectament coneguda en escriure el programa), donat que es coneix la forma de la resposta, podem pensar en efectuar un tractament específic.

> Per a desenvolupar programes Java que gestionin documents XML és necessari conèixer les API DOM, SAX i StAX facilitades per Java. També és recomanable conèixer l'API JDOM (projecte jdom.org) alternativa a les API facilitades per Java.

A continuació veurem alguns programes de sentència tancada i, finalment, un programa de sentència oberta i els executarem, tots ells, sobre els SGBD *eXist-db*, *Sedna* i *BaseX*. Per a que això sigui possible, cal tenir instal·lat:

- En *eXist-db*, una col·lecció de nom `xqj` amb l'arxiu `mondial.xml`.
- En *BaseX*, una base de dades de nom `xqj` amb l'arxiu `mondial.xml`.
- En *Sedna*, una base de dades de nom `db`, amb una col·lecció de nom `xqj` amb l'arxiu `mondial.xml`.

> Als annexos de la web trobareu l'apartat “Material per desenvolupament sobre SGBD-XML natives utilitzant l'API XQJ” que inclou l'arxiu `mondial.xml` i les instruccions d'instal·lació en

els diversos SGBD per a poder executar els programes aquí desenvolupats.

Exemple de programa amb sentència tancada de tipus "consulta" i d'execució immediata

El següent programa mostra els noms de tots els països de l'arxiu `mondial.xml`.

Trobareu el fitxer `XQJ03.java` dins el paquet de codi font a l'apartat "Material per desenvolupament sobre SGBD-XML natives utilitzant l'API XQL" dels annexos del web.

```
/*
 * Programa: XQJ03.java
 * Objectiu: Programa amb sentència tancada de tipus "consulta" i
 *           d'execució immediata.
 *           Llistar els noms dels països existents a l'arxiu "mondial.xml"
 *           de la col·lecció/base de dades "xqj"
 * Autor...: Isidre Guixà
 */

package proves;
import javax.xml.xquery.*;
import java.lang.Exception;

public class XQJ03
{
  // Amaguem el constructor per defecte. */
  private XQJ03() { }

  private static XQConnection establirConnexio(String[] args)
                throws Exception
  /* Intenta establir una connexió contra el SGBD indicat per paràmetre, que
     ha de ser: eXist-db o BaseX o Sedna */
  { /* Mateix contingut que programa XQJ02.java
       tenint en compte que la base de dades de Sedna és "db" */ }

  public static void main(String[] args){
    XQConnection conn = null;
    try {
      // Obrir sessió:
      conn = establirConnexio(args);
      System.out.println("Connexió establerta amb SGBD " + args[0]);
      // Creem XQExpression:
      XQExpression xqe = conn.createExpression();
      // Preparem instrucció, específica per a cada SGBD
      String cad;
      if (args[0].equalsIgnoreCase("eXist-db"))
        cad = "doc('xqj/mondial.xml')/mondial/country/name";
      else if (args[0].equalsIgnoreCase("BaseX")) {
        cad = "for $doc in collection('xqj')\n";
        cad = cad + "where contains(document-uri($doc),'mondial.xml')\n";
        cad = cad + "return $doc/mondial/country/name";
      }
```

```
      else // Sedna
        cad = "doc('mondial.xml','xqj')/mondial/country/name";
      System.out.println("Executant instrucció:\n"+cad);
      XQResultSequence xqrs = xqe.executeQuery(cad);
      // Mostrem resultats, un a un, convertits a String
      System.out.println("\nResultats:");
      while (xqrs.next())
        System.out.println(xqrs.getItemAsString(null));
    }
    catch (Exception e) {
      e.printStackTrace();
    }
    finally { /* Tanquem connexió en qualsevol cas */
      try {
          if(conn != null) conn.close();
      }
      catch(XQException xe) {
          xe.printStackTrace();
      }
    }
  }
}
```

Comprovem el funcionament d'aquest programa, executant-lo per cadascun dels tres SGBD. L'execució en qualsevol cas és:

```
G:\>java -Dfile.encoding=cp850 proves.XQJ03 nomSGBD
Connexió establerta amb SGBD ...
Executant instrucció:
...
Resultats:
<name>Albania</name>
<name>Greece</name>
<name>Macedonia</name>
...
<name>Seychelles</name>
```

Podem comprovar que el resultat és idèntic en els tres SGBD i que en tots ells obtenim el nom dels països entre les etiquetes `<name>` i `</name>`. En cas que només vulguem obtenir el nom del país sense les etiquetes, podem pensar en utilitzar la funció `text()` al final de la sentència *XQuery:*

```
/mondial/country/name/text()
```

Aquesta sintaxis funciona perfectament en el llenguatge *XQuery* en els tres SGBD, però l'API XQJ no la tracta com esperem i els resultats són molt diferents. Per tant, si ens cal obtenir únicament el nom, podem efectuar dos tractaments:

- Cercar la subcadena existent entre les etiquetes `<name>` i `</name>`.
- Passar cada `XQItem` de la seqüència `XQResultSequence` cap a `XMLStreamReader` (via mètode `getItemAsStream()`) i llavors efectuar-ne un tractament com a objecte `XMLStreamReader`.

El següent tros de programa (incorporat en el programa `XQJ04.java`) mostra les instruccions a executar per mostrar únicament els noms dels països (sense etiquetes) passant per objectes `XMLStreamReader`:

> Trobareu el fitxer `XQJ04.java` dins el paquet de codi font a l'apartat "Material per desenvolupament sobre SGBD-XML natives utilitzant l'API XQL" dels annexos del web.

```
      System.out.println("\nResultats:");
      while (xqrs.next()) {
        XMLStreamReader xsr = xqrs.getItemAsStream();
        for(; xsr.hasNext(); xsr.next()) tractarNom(xsr);
      }
```

on

```
  static void tractarNom(XMLStreamReader reader) {
    if (reader.getEventType() == XMLStreamConstants.CHARACTERS)
        System.out.println(reader.getText());
  }
```

Comprovem el funcionament d'aquest programa, executant-lo per cadascun dels tres SGBD. L'execució en qualsevol cas és:

```
G:\>java -Dfile.encoding=cp850 proves.XQJ04 nomSGBD
Connexió establerta amb SGBD ...
Executant instrucció:
...
Resultats:
Albania
Greece
Macedonia
...
Seychelles
```

Exemple de programa amb sentència tancada de tipus "no consulta" `(insert)` i d'execució immediata

El següent programa insereix un node `country` a l'inici de tots els països (és a dir, immediatament abans del `country[1]`)

> Trobareu el fitxer `XQJ05.java` dins el paquet de codi font a l'apartat "Material per desenvolupament sobre SGBD-XML natives utilitzant l'API XQL" dels annexos del web.

Es tracta d'una instrucció d'actualització de la informació existent en un fitxer XML. Recordem que el llenguatge *XQuery* és, inicialment, de només consulta i que hi ha diverses extensions *Update* d'aquest llenguatge:

- L'extensió promoguda per P. Lehti, implementada a *eXist-db* i *Sedna*. Les instrucció d'aquesta extensió són interpretades, per l'API XQJ, com ordres i, per tant, cal executar-les amb el mètode `XQExpression.executeCommand()`.
- L'extensió XQUF promoguda per W3C, implementada a *BaseX*. Les instruccions d'aquesta extensió són interpretades, per l'API XQJ, com instruccions *XQuery* i, en conseqüència, cal executar-les amb el mètode `XQExpression.executeQuery()`.

> En ocasions, pot ser necessari filtrar la informació a actualitzar (`insert`, `update` o `delete`) amb sentències FLW. En tal cas,, caldrà consultar la sintaxis a emprar en cadascuna de les extensions *Update* existents per *XQuery*: documentació de P. Lehti (extensió *Update* d'*eXist-db* i *Sedna*) o documentació W3Q

(extensió XQUF de *BaseX)*.

El programa, doncs, actua d'una manera o altra segons el SGBD al què es connecta.

```
/*
 * Programa: XQJ05.java
 * Objectiu: Programa amb sentència tancada de tipus "no consulta" (insert)
 *           i d'execució immediata.
 *           Inserir un node <country> a l'inici de l'arxiu "mondial.xml"
 * Autor...: Isidre Guixà
 */

package proves;
import javax.xml.xquery.*;
import java.lang.Exception;

public class XQJ05
{
  // Amaguem el constructor per defecte. */
  private XQJ05() { }

  private static XQConnection establirConnexio(String[] args)
                throws Exception
  /* Intenta establir una connexió contra el SGBD indicat per paràmetre, que
     ha de ser: eXist-db o BaseX o Sedna */
  { /* Mateix contingut que programa XQJ02.java
       tenint en compte que la base de dades de Sedna és "db" */ }

  public static void main(String[] args){
    XQConnection conn = null;
    try {
      // Obrir sessió:
      conn = establirConnexio(args);
      System.out.println("Connexió establerta amb SGBD " + args[0]);
      // Creem XQExpression:
      XQExpression xqe = conn.createExpression();
      // Preparem instrucció, específica per a cada SGBD
      String cad;
      if (args[0].equalsIgnoreCase("eXist-db"))
        cad = "update insert \n"+
              "<country car_code='$$'><name>nouPais</name></country> \n"+
              "preceding doc('xqj/mondial.xml')/mondial/country[1]";
      else if (args[0].equalsIgnoreCase("BaseX"))
        cad = "for $doc in collection('xqj') \n"+
              "where contains(document-uri($doc),'mondial.xml') \n"+
              "return \n"+
              "insert node \n"+
              "<country car_code='$$'><name>nouPais</name></country> \n"+
              "before $doc/mondial/country[1]";
      else // Sedna
        cad = "update insert \n"+
              "<country car_code='$$'><name>nouPais</name></country> \n"+
              "preceding doc('mondial.xml','xqj')/mondial/country[1]";
      System.out.println("Executant instrucció:\n"+cad);
      if (args[0].equalsIgnoreCase("BaseX")) xqe.executeQuery(cad); // XQUF
      else xqe.executeCommand(cad); // Ordres UPDATE segons P. Lehti
      System.out.println("\nInstrucció executada");
    }
    catch (Exception e) {
      e.printStackTrace();
    }
```

```
    finally { /* Tanquem connexió en qualsevol cas */
      try {
          if(conn != null) conn.close();
      }
      catch(XQException xe) {
          xe.printStackTrace();
      }
    }
  }
}
```

Comprovem el funcionament d'aquest programa, executant-lo per cadascun dels tres SGBD. L'execució en qualsevol cas és:

```
G:\>java -Dfile.encoding=cp850 proves.XQJ05 nomSGBD
Connexió establerta amb SGBD ...
Executant instrucció:
...
Instrucció executada
```

Podem comprovar, via la interfície gràfica que correspongui (*eXist Client Shell*, *BaseX GUI* o *SednaAdmin*) que la inserció s'ha dut a terme.

Exemple de programa amb sentència tancada de tipus "no consulta" `(update)` i d'execució immediata

El següent programa canvia el nom dels països que tenen `$$` com a `car_code`.

> Trobareu el fitxer `XQJ06.java` dins el paquet de codi font a l'apartat "Material per desenvolupament sobre SGBD-XML natives utilitzant l'API XQL" dels annexos del web.

```
/*
 * Programa: XQJ06.java
 * Objectiu: Programa amb sentència tancada de tipus "no consulta" (update)
 *           i d'execució immediata.
 *           Canviar un valor d'un node
 * Autor...: Isidre Guixà
 */

package proves;
import javax.xml.xquery.*;
import java.lang.Exception;

public class XQJ06
{
  // Amaguem el constructor per defecte. */
  private XQJ06() { }

  private static XQConnection establirConnexio(String[] args)
                  throws Exception
  /* Intenta establir una connexió contra el SGBD indicat per paràmetre, que
     ha de ser: eXist-db o BaseX o Sedna */
  { /* Mateix contingut que programa XQJ02.java
       tenint en compte que la base de dades de Sedna és "db" */ }

  public static void main(String[] args){
    XQConnection conn = null;
```

```
    try {
      // Obrir sessió:
      conn = establirConnexio(args);
      System.out.println("Connexió establerta amb SGBD " + args[0]);
      // Creem XQExpression:
      XQExpression xqe = conn.createExpression();
      // Preparem instrucció, específica per a cada SGBD
      String cad;
      if (args[0].equalsIgnoreCase("eXist-db"))
        cad = "update value \n"+
              "doc('xqj/mondial.xml')/mondial/country[@car_code='$$']/name \n"+
              "with 'nouNom'";
      else if (args[0].equalsIgnoreCase("BaseX"))
        cad = "for $doc in collection('xqj') \n"+
              "where contains(document-uri($doc),'mondial.xml') \n"+
              "return \n"+
              "replace value of "+
              "node $doc/mondial/country[@car_code='$$']/name \n"+
              "with 'nouNom'";
      else // Sedna
        cad = "update replace $n in \n"+
              "doc('mondial.xml','xqj')/mondial/"+
                   "country[@car_code='$$']/name \n"+
              "with <name>nouNom</name>";
      System.out.println("Executant instrucció:\n"+cad);
      if (args[0].equalsIgnoreCase("BaseX")) xqe.executeQuery(cad); // XQUF
      else xqe.executeCommand(cad); // Ordres UPDATE segons P. Lehti
      System.out.println("\nInstrucció executada");
    }
    catch (Exception e) {
      e.printStackTrace();
    }
    finally { /* Tanquem connexió en qualsevol cas */
      try {
          if(conn != null) conn.close();
      }
      catch(XQException xe) {
          xe.printStackTrace();
      }
    }
  }
}
```

Comprovem el funcionament d'aquest programa, executant-lo per cadascun dels tres SGBD. L'execució en qualsevol cas és:

```
G:\>java -Dfile.encoding=cp850 proves.XQJ06 nomSGBD
Connexió establerta amb SGBD ...
Executant instrucció:
...
Instrucció executada
```

Podem comprovar, via la interfície gràfica que correspongui (*eXist Client Shell*, *BaseX GUI* o *SednaAdmin*) que la modificació s'ha dut a terme.

Exemple de programa amb sentència tancada de tipus "no consulta" `(delete)` i d'execució immediata

El següent programa elimina els països que tenen `$$` com a `car_code`.

> Trobareu el fitxer `XQJ07.java` dins el paquet de codi font a

l'apartat “Material per desenvolupament sobre SGBD-XML natives utilitzant l'API XQL” dels annexos del web.

```
/*
 * Programa: XQJ07.java
 * Objectiu: Programa amb sentència tancada de tipus “no consulta” (delete)
 *           i d'execució immediata.
 *           Eliminar un node
 * Autor...: Isidre Guixà
 */

package proves;
import javax.xml.xquery.*;
import java.lang.Exception;

public class XQJ07
{
  // Amaguem el constructor per defecte. */
  private XQJ07() { }

  private static XQConnection establirConnexio(String[] args)
                throws Exception
  /* Intenta establir una connexió contra el SGBD indicat per paràmetre, que
     ha de ser: eXist-db o BaseX o Sedna */
  { /* Mateix contingut que programa XQJ02.java
       tenint en compte que la base de dades de Sedna és "db" */ }

  public static void main(String[] args){
    XQConnection conn = null;
    try {
      // Obrir sessió:
      conn = establirConnexio(args);
      System.out.println("Connexió establerta amb SGBD " + args[0]);
      // Creem XQExpression:
      XQExpression xqe = conn.createExpression();
      // Preparem instrucció, específica per a cada SGBD
      String cad;
      if (args[0].equalsIgnoreCase("eXist-db"))
        cad = "update delete \n"+
              "doc('xqj/mondial.xml')/mondial/country[@car_code='$$']";
      else if (args[0].equalsIgnoreCase("BaseX"))
        cad = "for $doc in collection('xqj') \n"+
              "where contains(document-uri($doc),'mondial.xml') \n"+
              "return \n"+
              "delete node $doc/mondial/country[@car_code='$$']";
      else // Sedna
        cad = "update delete \n"+
              "doc('mondial.xml','xqj')/mondial/country[@car_code='$$']";
      System.out.println("Executant instrucció:\n"+cad);
      if (args[0].equalsIgnoreCase("BaseX")) xqe.executeQuery(cad); // XQUF
      else xqe.executeCommand(cad); // Ordres UPDATE segons P. Lehti
      System.out.println("\nInstrucció executada");
    }
    catch (Exception e) {
      e.printStackTrace();
    }
    finally { /* Tanquem connexió en qualsevol cas */
      try {
          if(conn != null) conn.close();
```

```
        }
        catch(XQException xe) {
            xe.printStackTrace();
        }
    }
  }
}
```

Comprovem el funcionament d'aquest programa, executant-lo per cadascun dels tres SGBD. L'execució en qualsevol cas és:

```
G:\>java -Dfile.encoding=cp850 proves.XQJ07 nomSGBD
Connexió establerta amb SGBD ...
Executant instrucció:
...
Instrucció executada
```

Podem comprovar, via la interfície gràfica que correspongui (*eXist Client Shell*, *BaseX GUI* o *SednaAdmin*) que l'eliminació s'ha dut a terme.

Els tres exemples anteriors han consistit en l'execució d'una sentència "no consulta" corresponent a actualització de les base de dades (`insert`, `update` i `delete`, respectivament). Hi ha, però, altres tipus d'ordres, segons el SGBD, que també es poden executar des de l'API XQJ (gestió de col·leccions, gestió de documents,...). Donat que no tots els SGBD les permeten i que la sintaxis és molt diferent d'uns a altres, no té cap interès desenvolupar programes que puguin servir per a diversos SGBD. És convenient, però, que el lector n'efectuï proves en els diversos SGBD.

Observem, també, que en els tres exemples anteriors s'ha executat instruccions d'actualització de la base de dades sense validar (`commit`) en cap cas els canvis i els canvis han esdevingut permanents. Això ha estat així per què, si no s'indica el contrari, les connexions de l'API XQL neixen amb la propietat `auto-commit` activada.

Per finalitzar l'execució de sentències immediates, presentem dos programes de sentència oberta (l'usuari introdueix la sentència a executar): un per executar sentències "consulta" i l'altre per executar sentències "no consulta". L'usuari ha de tenir consciència de que està executant el programa adequat, segons el tipus de consulta.

> Recordem que les instruccions `UPDATE` que aporten els SGBD *eXist-db* i *Sedna* són considerades "no consulta", mentre que les que aporta el SGBD *BaseX* (XQUF) són considerades "consulta".

Exemple de programa amb sentència oberta de tipus "consulta" i d'execució immediata

> Les API client específiques d'alguns SGBD-XML natives faciliten mecanismes per saber, abans de procedir a l'execució, si una sentència correspon a una "consulta" (retorna un conjunt de resultats a processar) o a una "no consulta" (executa una ordre). El SGBD *Sedna* n'és un exemple.

El programa demana a l'usuari, per la consola, la instrucció a executar i procedeix a executar la instrucció com a "consulta". Recordem que, malauradament, la interfície `XQExpression` no facilita cap mètode per deduir si una sentència és "consulta" o "no consulta". Si no es produeix error, el programa intenta mostrar la seqüència de resultats de dues maneres:

- Com a arxiu XML, a partir de la seriació de la seqüència de resultats via el mètode `XQResultSequence.getSequenceAsStream()` i amb l'ajut de la classe `Transformer`.
- Com a objecte String, a partir de la seriació de la seqüència de resultats via el mètode `XQResultSequence.getSequenceAsString()`.

Per tal de poder veure el resultat de dues maneres, cal tornar a executar la sentència abans de la segona visualització.

En cas que la seqüència de resultats (`XQResultSequence`) contingui més d'un valor, per a que la seriació cap a `XMLStreamReader` funcioni correctament, cal que l'API XQJ emboliqui tota la seqüència dins un node de més alt nivell. Aquest funcionament és correcte en les interfícies desenvolupades per Charles Foster per als SGBD *eXist-db* i *Sedna*, però no és així en l'API XQJ integrada en *BaseX* (versió 7.1), de manera que la visualització XML dels resultats de les consultes amb més d'un resultat, en *BaseX,* només mostra el primer resultat. Per això també mostrem la solució en format `String`, per constatar que la instrucció s'executa correctament i que el que falla és la seriació cap a `XMLStreamReader`.

El SGBD *BaseX* (versió 7.1) porta inclosa una llibreria XQJ, que presenta algunes anomalies. En el moment de confeccionar aquests materials, els responsables de *BaseX* han informat que Charles Foster està desenvolupant una llibreria XQJ per a *BaseX*, similar a les desenvolupades per *eXist-db* i *Sedna*. Cal doncs, està atent a l'aparició d'aquesta llibreria per a utilitzar-la en lloc de la que incorpora *BaseX*.

Trobareu el fitxer `XQJ08.java` dins el paquet de codi font a l'apartat "Material per desenvolupament sobre SGBD-XML natives utilitzant l'API XQL" dels annexos del web.

```
/*
 * Programa: XQJ08.java
 * Objectiu: Programa amb sentència oberta de tipus "consulta" i d'execució
 *           immediata, en SGBD-XML eXist-db, BaseX i Sedna
 * Autor...: Isidre Guixà
 */

package proves;
import javax.xml.xquery.*;
```

```
import java.lang.Exception;
import java.io.*;
import javax.xml.stream.*;
import javax.xml.transform.*;
import javax.xml.transform.stax.*;
import javax.xml.transform.stream.*;

public class XQJ08
{
  // Amaguem el constructor per defecte. */
  private XQJ08() { }

  private static XQConnection establirConnexio(String[] args)
                throws Exception
  /* Intenta establir una connexió contra el SGBD indicat per paràmetre, que
     ha de ser: eXist-db o BaseX o Sedna */
  { /* Mateix contingut que programa XQJ02.java
       tenint en compte que la base de dades de Sedna és "db" */ }

  private static String introduirInstruccio() {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    String instr="";
    System.out.println("Introdueixi la instrucció a executar...");
    System.out.println("Per finalitzar, introdueixi línia buida (en blanc):");
    try {
      String text;
      do {
        text = br.readLine();
        if (!(text.isEmpty())) instr=instr.concat(text+"\n");
      } while (!(text.isEmpty()));
    }
    catch (IOException e) {
       System.out.println("Excepció en la lectura de la instrucció:");
       System.err.println(e);
    }
    return instr.trim();
  }

  public static void main(String[] args){
    XQConnection conn = null;
    try {
      // Demanem la instrucció a executar
      String cad = introduirInstruccio();
      if ("".equals(cad))
      {
        System.out.println("No ha introduït cap consulta.");
        System.exit(1);
      }
      // Obrir sessió:
      conn = establirConnexio(args);
      System.out.println("Connexió establerta amb SGBD " + args[0]);
      // Creem XQExpression:
      XQExpression xqe = conn.createExpression();
      // Executem la instrucció
      System.out.println("Executant instrucció \"consulta\":\n"+cad);
      XQResultSequence xqrs = xqe.executeQuery(cad);
      // Mostrem resultats
      System.out.println("\nVisualització de resultats com a XML:");
      try {
        Source resultXML = new StAXSource(xqrs.getSequenceAsStream());
        StreamResult resultSR = new StreamResult(System.out);
```

```
        TransformerFactory factory = TransformerFactory.newInstance();
        Transformer tr = factory.newTransformer();
        // Les següents línies són per fer "llegible" la sortida
        tr.setOutputProperty(OutputKeys.INDENT, "yes");
        tr.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION, "no");
        tr.setOutputProperty("{http://xml.apache.org/xslt}indent-amount",
                             "2");
        tr.transform(resultXML, resultSR);
      }
      catch (Exception e) { // Mostrar el resultat com a String
        System.out.println("\nNo es pot visualitzar com a XML! Error:");
        e.printStackTrace();
      }
      System.out.println("\nRepetim execució:\n"+cad);
      xqrs = xqe.executeQuery(cad);
      // Mostrem resultats
      System.out.println("\nVisualització de resultats com a String:");
      try {
        System.out.println(xqrs.getSequenceAsString(null));
      }
      catch (Exception e) { // Mostrar el resultat com a String
        System.out.println("\nNo es pot visualitzar com a String! Error:");
        e.printStackTrace();
      }
    }
    catch (Exception e) {
      e.printStackTrace();
    }
    finally { /* Tanquem connexió en qualsevol cas */
      try {
          if(conn != null) conn.close();
      }
      catch(XQException xe) {
          xe.printStackTrace();
      }
    }
  }
}
```

Comprovem el funcionament d'aquest programa, executant una mateixa instrucció de consulta (amb la sintaxis adequada a cada SGBD) per cadascun dels tres SGBD. Demanem, per exemple, els noms de les autonomies de l'estat espanyol.

Execució en *eXist-db*:

```
G:\>java -Dfile.encoding=cp850 proves.XQJ08 eXist-db
Introdueixi la instrucció a executar...
Per finalitzar, introdueixi línia buida (en blanc):
doc("xqj/mondial.xml")/mondial/country[@car_code='E']/province/name

Connexió establerta amb SGBD exist-db
Executant instrucció "consulta":
doc("xqj/mondial.xml")/mondial/country[@car_code='E']/province/name

Visualització de resultats com a XML:
<?xml version="1.0" encoding="UTF-8"?>
<r:result xmlns:r="http://www.xqj.net/">
  <name>Andalusia</name>
  <name>Aragon</name>
  <name>Asturias</name>
  <name>Balearic Islands</name>
  <name>Basque Country</name>
  <name>Canary Islands</name>
```

```
  <name>Cantabria</name>
  <name>Castile and Leon</name>
  <name>Castile La Mancha</name>
  <name>Catalonia</name>
  <name>Estremadura</name>
  <name>Galicia</name>
  <name>Madrid</name>
  <name>Murcia</name>
  <name>Navarre</name>
  <name>Rioja</name>
  <name>Valencia</name>
</r:result>

Repetim execució:
doc("xqj/mondial.xml")/mondial/country[@car_code='E']/province/name

Visualització de resultats com a String:
<name>Andalusia</name> <name>Aragon</name> <name>Asturias</name> <name>Balearic
Islands</name> <name>Basque Country</name> <name>Canary Islands</name><name>Can
tabria</name> <name>Castile and Leon</name> <name>Castile La Mancha</name><name
>Catalonia</name> <name>Estremadura</name> <name>Galicia</name> <name>Madrid</n
ame> <name>Murcia</name> <name>Navarre</name><name>Rioja</name><name>Valencia</
name>
```

Execució en *Sedna*:

```
G:\>java -Dfile.encoding=cp850 proves.XQJ08 Sedna
Introdueixi la instrucció a executar...
Per finalitzar, introdueixi línia buida (en blanc):
doc("mondial.xml","xqj")/mondial/country[@car_code='E']/province/name

Connexió establerta amb SGBD Sedna
Executant instrucció "consulta":
doc("mondial.xml","xqj")/mondial/country[@car_code='E']/province/name

Visualització de resultats com a XML:
<?xml version="1.0" encoding="UTF-8"?>
<r:result xmlns:r="http://www.xqj.net/">
  <name>Andalusia</name>
  <name>Aragon</name>
  <name>Asturias</name>
  <name>Balearic Islands</name>
  <name>Basque Country</name>
  <name>Canary Islands</name>
  <name>Cantabria</name>
  <name>Castile and Leon</name>
  <name>Castile La Mancha</name>
  <name>Catalonia</name>
  <name>Estremadura</name>
  <name>Galicia</name>
  <name>Madrid</name>
  <name>Murcia</name>
  <name>Navarre</name>
  <name>Rioja</name>
  <name>Valencia</name>
</r:result>

Repetim execució:
doc("mondial.xml","xqj")/mondial/country[@car_code='E']/province/name

Visualització de resultats com a String:
<name>Andalusia</name> <name>Aragon</name> <name>Asturias</name> <name>Balearic
Islands</name> <name>Basque Country</name> <name>Canary Islands</name><name>Can
```

```
tabria</name> <name>Castile and Leon</name> <name>Castile La Mancha</name><name
>Catalonia</name> <name>Estremadura</name> <name>Galicia</name> <name>Madrid</n
ame> <name>Murcia</name> <name>Navarre</name><name>Rioja</name><name>Valencia</
name>
```

Execució en *BaseX*:

```
G:\>java -Dfile.encoding=cp850 proves.XQJ08 BaseX
Introdueixi la instrucció a executar...
Per finalitzar, introdueixi línia buida (en blanc):
for $doc in collection("xqj")
where contains(document-uri($doc),"mondial.xml")
return $doc/mondial/country[@car_code='E']/province/name

Connexió establerta amb SGBD Basex
Executant instrucció "consulta":
for $doc in collection("xqj")
where contains(document-uri($doc),"mondial.xml")
return $doc/mondial/country[@car_code='E']/province/name

Visualització de resultats com a XML:
<?xml version="1.0" encoding="UTF-8"?>
<name>Andalusia</name>

Repetim execució:
for $doc in collection("xqj")
where contains(document-uri($doc),"mondial.xml")
return $doc/mondial/country[@car_code='E']/province/name

Visualització de resultats com a String:
<name>Andalusia</name>
<name>Aragon</name>
<name>Asturias</name>
<name>Balearic Islands</name>
<name>Basque Country</name>
<name>Canary Islands</name>
<name>Cantabria</name>
<name>Castile and Leon</name>
<name>Castile La Mancha</name>
<name>Catalonia</name>
<name>Estremadura</name>
<name>Galicia</name>
<name>Madrid</name>
<name>Murcia</name>
<name>Navarre</name>
<name>Rioja</name>
<name>Valencia</name>
```

Podem comprovar que, la visualització XML en *BaseX* (versió 7.1) no és correcta.

Exemple de programa amb sentència oberta de tipus “no consulta” i d'execució immediata

El programa demana a l'usuari, per la consola, la instrucció a executar i procedeix a executar la instrucció com a “no consulta”. Recordem que, malauradament, la interfície `XQExpression` no facilita cap mètode per deduir si una sentència és “consulta” o “no consulta”. Si no es produeix error, el programa informarà que la sentència ha estat executada.

Trobareu el fitxer `XQJ09.java` dins el paquet de codi font a

l'apartat “Material per desenvolupament sobre SGBD-XML natives utilitzant l'API XQL” dels annexos del web.

```
/*
 * Programa: XQJ09.java
 * Objectiu: Programa amb sentència oberta de tipus "no consulta" i
 *           d'execució immediata, en SGBD-XML eXist-db, BaseX i Sedna
 * Autor...: Isidre Guixà
 */

package proves;
import javax.xml.xquery.*;
import java.lang.Exception;
import java.io.*;

public class XQJ09
{
  // Amaguem el constructor per defecte. */
  private XQJ09() { }

  private static XQConnection establirConnexio(String[] args)
                throws Exception
  /* Intenta establir una connexió contra el SGBD indicat per paràmetre, que
     ha de ser: eXist-db o BaseX o Sedna */
  { /* Mateix contingut que programa XQJ02.java
       tenint en compte que la base de dades de Sedna és "db" */ }

  private static String introduirInstruccio() {
    /* Mateix contingut que programa XQJ08.java */
  }

  public static void main(String[] args){
    XQConnection conn = null;
    try {
      // Demanem la instrucció a executar
      String cad = introduirInstruccio();
      if ("".equals(cad))
      {
        System.out.println("No ha introduït cap instrució.");
        System.exit(1);
      }
      // Obrir sessió:
      conn = establirConnexio(args);
      System.out.println("Connexió establerta amb SGBD " + args[0]);
      // Creem XQExpression:
      XQExpression xqe = conn.createExpression();
      // Executem la instrucció
      System.out.println("Executant instrucció \"no consulta\":\n"+cad);
      xqe.executeCommand(cad);
      System.out.println("\nInstrucció executada");
    }
    catch (Exception e) {
      e.printStackTrace();
    }
    finally { /* Tanquem connexió en qualsevol cas */
      try {
          if(conn != null) conn.close();
      }
      catch(XQException xe) {
```

```
            xe.printStackTrace();
        }
      }
    }
}
```

Comprovem el funcionament d'aquest programa, executant una mateixa instrucció (amb la sintaxis adequada a cada SGBD) per cadascun dels tres SGBD. Procedim, per exemple, a crear una col·lecció (base de dades a *BaseX)* a l'arrel de la base de dades activa. Per conèixer la sintaxis adequada per aconseguir el nostre objectiu, en cadascun dels tres SGBD, ens cal consultar la corresponent documentació:

Execució en *eXist-db*:

```
G:\>java -Dfile.encoding=cp850 proves.XQJ09 eXist-db
Introdueixi la instrucció a executar...
Per finalitzar, introdueixi línia buida (en blanc):
xmldb:create-collection("/","xqjbis")

Connexió establerta amb SGBD exist-db
Executant instrucció "no consulta":
xmldb:create-collection("/","xqjbis")

Instrucció executada
```

Podem comprovar, via la interfície gràfica *eXist Client Shell* que la instrucció s'ha dut a terme.

Execució en *Sedna*:

```
G:\>java -Dfile.encoding=cp850 proves.XQJ09 Sedna
Introdueixi la instrucció a executar...
Per finalitzar, introdueixi línia buida (en blanc):
create collection"xqjbis"

Connexió establerta amb SGBD Sedna
Executant instrucció "no consulta":
create collection"xqjbis"

Instrucció executada
```

Podem comprovar, via la interfície gràfica *SednaAdmin* (prèvia desconnexió-connexió a la base de dades) que la instrucció s'ha dut a terme.

Execució en *BaseX*:

```
G:\>java -Dfile.encoding=cp850 proves.XQJ09 BaseX
Introdueixi la instrucció a executar...
Per finalitzar, introdueixi línia buida (en blanc):
create database xqjbis

Connexió establerta amb SGBD BaseX
Executant instrucció "no consulta":
create database xqjbis

Instrucció executada
```

Podem comprovar, via la interfície gràfica *BaseX GUI* que la instrucció s'ha dut a terme.

2.1.3. Variables lligades

En les aplicacions Java que accedeixen a bases de dades, ja sigui via SQL o via *XQuery,*

una de les pitjors implementacions que pot efectuar un programador, és utilitzar el contingut d'una variable emplenada per l'usuari directament en una expressió SQL o *XQuery*, doncs hi ha usuaris finals maliciosos que poden injectar codi SQL o *XQuery* maliciós per aconseguir informació privilegiada o per efectuar actualitzacions no desitjades.

Exemple de programa que admet injecció *XQuery*

El següent programa mostra els noms dels països democràtics que han aconseguit la independència a partir d'una determinada data a introduir per l'usuari.

> Trobareu el fitxer `XQJ10.java` dins el paquet de codi font a l'apartat "Material per desenvolupament sobre SGBD-XML natives utilitzant l'API XQL" dels annexos del web.

```
/*
 * Programa: XQJ10.java
 * Objectiu: Exemple de programa que permet injecció de codi XQuery
 * Autor...: Isidre Guixà
 */

package proves;
import javax.xml.xquery.*;
import java.lang.Exception;
import java.io.*;

public class XQJ10
{
  // Amaguem el constructor per defecte. */
  private XQJ10() { }

  private static XQConnection establirConnexio(String[] args)
                throws Exception
  /* Intenta establir una connexió contra el SGBD indicat per paràmetre, que
     ha de ser: eXist-db o BaseX o Sedna */
  { /* Mateix contingut que programa XQJ02.java
       tenint en compte que la base de dades de Sedna és "db" */ }

  private static String introduirDada() {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    String text=null;
    System.out.println("Països democràtics independents a partir de data...");
    System.out.println("Introdueixi data [dddd-mm-yy] o res per finalitzar:");
    try {
      text = br.readLine();
    }
    catch (IOException e) {
       System.out.println("Excepció en la lectura de la dada:");
       System.err.println(e);
    }
    return text;
  }

  public static void main(String[] args){
    XQConnection conn = null;
    try {
```

```
      // Demanem dada a l'usuari
      String desdeData = introduirDada();
      if ("".equals(desdeData))
      {
        System.out.println("No ha introduït cap dada.");
        System.exit(1);
      }
    // Obrir sessió:
      conn = establirConnexio(args);
      System.out.println("Connexió establerta amb SGBD " + args[0]);
      // Creem XQExpression:
      XQExpression xqe = conn.createExpression();
      // Preparem instrucció, específica per a cada SGBD
      String cad;
      if (args[0].equalsIgnoreCase("eXist-db"))
        cad = "for $i in doc('xqj/mondial.xml')/mondial/"+
              "country[contains(government,'democracy') and "+
              "indep_date >= '"+desdeData+"'] \n"+
              "return concat($i/name,' - ',$i/government,' - ',$i/indep_date)";
      else if (args[0].equalsIgnoreCase("BaseX")) {
        cad = "for $doc in collection('xqj') \n"+
              "for $i in $doc/mondial/"+
              "country[contains(government,'democracy') and "+
              "indep_date >= '"+desdeData+"'] \n"+
              "where contains(document-uri($doc),'mondial.xml') \n"+
              "return concat($i/name,' - ',$i/government,' - ',$i/indep_date)";
      }
      else // Sedna
        cad = "for $i in doc('mondial.xml','xqj')/mondial/"+
              "country[contains(government,'democracy') and "+
              "indep_date >= '"+desdeData+"'] \n"+
              "return concat($i/name,' - ',$i/government,' - ',$i/indep_date)";
      System.out.println("Executant instrucció:\n"+cad);
      XQResultSequence xqrs = xqe.executeQuery(cad);
      // Mostrem resultats, un a un, convertits a String
      System.out.println("\nResultats:");
      while (xqrs.next())
        System.out.println(xqrs.getItemAsString(null));
    }
    catch (Exception e) {
      e.printStackTrace();
    }
    finally { /* Tanquem connexió en qualsevol cas */
      try {
          if(conn != null) conn.close();
      }
      catch(XQException xe) {
          xe.printStackTrace();
      }
    }
  }
}
```

Comprovem el funcionament d'aquest programa, executant-lo per cadascun dels tres SGBD i suposem que l'usuari introdueix una data, com s'indica.
L'execució en qualsevol cas és:

```
G:\>java -Dfile.encoding=cp850 proves.XQJ10 nomSGBD
Països democràtics independents a partir de data...
Introdueixi data [dddd-mm-yy] o res per finalitzar:
1992-01-01
Connexió establerta amb SGBD ...
Executant instrucció:
```

```
...
Resultats:
Serbia - parliamentary democracy - 2006-06-05
Montenegro - parliamentary democracy - 2006-06-03
Czech Republic - parliamentary democracy - 1993-01-01
Slovakia - parliamentary democracy - 1993-01-01
Bosnia and Herzegovina - emerging democracy - 1992-04-01
Timor-Leste - parliamentary democracy - 2002-05-20
```

Observem, però, com l'usuari pot injectar codi *XQuery* per aconseguir tots els països, democràtics i no democràtics, i a partir de qualsevol data...

```
G:\>java -Dfile.encoding=cp850 proves.XQJ09 nomSGBD
Països democràtics amb independència a partir de data...
Introdueixi data [dddd-mm-yy] o res per finalitzar:
1992-01-01' or '1'='1
Connexió establerta amb SGBD ...
Executant instrucció:
...
Resultats:
Albania - emerging democracy - 1912-11-28
Greece - parliamentary republic - 1829-01-01
Macedonia - emerging democracy - 1991-09-17
...
Seychelles - republic - 1976-06-29
```

Per tant, l'usuari ha aconseguit informació que no tenia per què obtenir i el problema es produeix per què el programador ha utilitzat el contingut de la variable `desdeData` directament dins la sentència *XQuery*, sense cap tipus de comprovació del seu contingut.
El fet d'actuar així també provoca errors en cas que l'usuari no introdueixi una data en el format indicat. Comproveu, què succeeix quan l'usuari introdueix 01/01/1992 o qualsevol cadena alfanumèrica.

Per evitar els problemes d'injecció de codi i per assegurar que les variables que s'utilitzen en les expressions *XQuery* corresponguin a tipus adequats, *XQuery* permet declarar variables externes (sentència `declare` prèvia) i en el programa haurem de lligar les variables externes definides a la sentència *XQuery* amb els valors que corresponguin, normalment introduïts per l'usuari, mitjançant els mètodes `bind` que facilita la interfície `XQDynamicContext`.

Exemple de programa que utilitza variables externes en *XQuery* prevenint la injecció de codi i errors de tipus

El següent programa mostra els noms dels països democràtics que han aconseguit la independència a partir d'una determinada data a introduir per l'usuari. El programa controla que la data sigui correcta i no permet injecció de codi.

> Trobareu el fitxer `XQJ11.java` dins el paquet de codi font a l'apartat "Material per desenvolupament sobre SGBD-XML natives utilitzant l'API XQL" dels annexos del web.

```
/*
 * Programa: XQJ11.java
 * Objectiu: Exemple de programa que utilitza variables externes en XQuery
 *           prevenint la injecció de codi XQuery i errors de tipus
 * Autor...: Isidre Guixà
```

```
 */

package proves;
import javax.xml.xquery.*;
import java.lang.Exception;
import java.io.*;
import javax.xml.namespace.QName;

public class XQJ11
{
  // Amaguem el constructor per defecte. */
  private XQJ11() { }

  private static XQConnection establirConnexio(String[] args)
                throws Exception
  /* Intenta establir una connexió contra el SGBD indicat per paràmetre, que
     ha de ser: eXist-db o BaseX o Sedna */
  { /* Mateix contingut que programa XQJ02.java
       tenint en compte que la base de dades de Sedna és "db" */ }

  private static String introduirDada() {
  { /* Mateix contingut que programa XQJ10.java */ }

  public static void main(String[] args){
    XQConnection conn = null;
    try {
      // Obrir sessió:
      conn = establirConnexio(args);
      System.out.println("Connexió establerta amb SGBD " + args[0]);
      // Demanem dada a l'usuari
      String desdeData = introduirDada();
      if ("".equals(desdeData))
        throw new Exception("No ha introduït cap dada.");
      // Creem XQExpression:
      XQExpression xqe = conn.createExpression();
      // Preparem variable d'enllaç "fromDate" a partir del valor
      // contingut a "desdeData". A la instrucció XQuery utilitzarem
      // la variable d'enllaç "fromDate"
      try {
        xqe.bindAtomicValue(new QName("fromDate"),
                            desdeData,
                            conn.createAtomicType(XQItemType.XQBASETYPE_DATE)
                           );
      }
      catch(XQException e) {
        throw new Exception("No ha introduït una data correcta.");
      }
      // Preparem instrucció, específica per a cada SGBD
      String cad;
      if (args[0].equalsIgnoreCase("eXist-db"))
        cad = "declare variable $fromDate external; "+
              "for $i in doc('xqj/mondial.xml')/mondial/"+
              "country[contains(government,'democracy') and "+
              "indep_date >= $fromDate] \n"+
              "return concat($i/name,' - ',$i/government,' - ',$i/indep_date)";
      else if (args[0].equalsIgnoreCase("BaseX")) {
        cad = "declare variable $fromDate external; "+
              "for $doc in collection('xqj') \n"+
              "for $i in $doc/mondial/"+
              "country[contains(government,'democracy') and "+
              "indep_date >= $fromDate] \n"+
```

```
                "where contains(document-uri($doc),'mondial.xml') \n"+
                "return concat($i/name,' - ',$i/government,' - ',$i/indep_date)";
        }
        else // Sedna
          cad = "declare variable $fromDate external; "+
                "for $i in doc('mondial.xml','xqj')/mondial/"+
                "country[contains(government,'democracy') and "+
                "indep_date >= $fromDate] \n"+
                "return concat($i/name,' - ',$i/government,' - ',$i/indep_date)";
        System.out.println("Executant instrucció:\n"+cad);
        XQResultSequence xqrs = xqe.executeQuery(cad);
        // Mostrem resultats, un a un, convertits a String
        System.out.println("\nResultats:");
        while (xqrs.next())
          System.out.println(xqrs.getItemAsString(null));
      }
      catch (Exception e) {
        e.printStackTrace();
      }
      finally { /* Tanquem connexió en qualsevol cas */
        try {
            if(conn != null) conn.close();
        }
        catch(XQException xe) {
            xe.printStackTrace();
        }
      }
    }
}
```

En aquest cas, és impossible l'execució del programa introduint valors erronis com a data o intentant injecció de codi *XQuery*.

A l'exemple anterior s'utilitza el mètode `bindAtomicValue()` de la interfície `XQDynamicContext` per assignar el contingut de la variable `String desdeData` a la variable externa `fromDate` de la sentència *XQuery*:

```
xqe.bindAtomicValue(new QName("fromDate"),
                    desdeData,
                    conn.createAtomicType(XQItemType.XQBASETYPE_DATE)
                   );
```

on `xqe` correspon a l'expressió que executarem i que ha de contenir la variable externa `$fromDate` declarada en el prolog de la instrucció. En eXist-db, per exemple:

```
xqe.executeQuery(
   "declare variable $fromDate external; "+
   "for $i in doc('xqj/mondial.xml')/mondial/"+
   "country[contains(government,'democracy') and "+
   "indep_date >= $fromDate] "+
   "return concat($i/name,' - ',$i/government,' - ',$i/indep_date))";
```

Fixem-nos que:

- El primer paràmetre és el nom de la variable externa de l'objecte `XQExpression.`
- El segon paràmetre és el valor `String` que es vol assignar a la variable externa.
- El tercer paràmetre ha de ser un objecte `XQItemType`. corresponent al tipus amb el què s'ha d'interpretar el valor del segon paràmetre per assignar-lo a la variable del primer paràmetre.

La interfície `XQItemType` defineix el conjunt de tipus pels que es pot tenir objectes

`XQItemType`, els quals s'han de crear amb el mètode `createAtomicType()` de la interfície `XQDataFactory`.

La interfície `XQDynamicContext` conté mètodes similars a `bindAtomicValue()`, com `bindInt()`, `bindByte()`, `bindDouble()`, `bindDocument()`, `bindBoolean()`... que es diferencien de `bindAtomicValue()` en el segon paràmetre, que ha de ser del tipus que indica el nom del mètode.

> Mireu la documentació de les interfícies `XQDynamicContext` i `XQItemType`.

2.1.4. Sentències *XQuery* preparades

Tots els SGBD acostumen a facilitar dos mecanismes d'execució de sentències contra la BD, utilitzant el llenguatge que correspongui (SQL en SGBDR i SGBDOR, OQL en SGBDOO i *XQuery*/*Update* en SGBD-XML):

- Execució immediata d'una sentència, utilitzada quan la sentència s'ha d'executar una única vegada.
- Execució de sentències paramètriques o preparades, utilitzada quan la mateixa sentència (o sentència similar amb alguns valors diferents) s'ha d'executar repetidament i consistent en escriure la sentència com una plantilla que conté algunes variables que s'aniran substituint a cada execució.

L'API XQJ facilita els dos mecanismes a través de les interfícies:

- `XQExpression`, per a l'execució immediata de sentències
- `XQPreparedExpression`, per a l'execució de sentències paramètriques

Centrem-nos, en aquest moment, en l'execució de sentències paramètriques proporcionada per la interfície `XQPreparedExpression`.

Per executar una sentència preparada, crearem un objecte `XQPreparedExpression` a partir del mètode `XQConnection.prepareExpression()`. Disposem de sis sobrecàrregues d'aquest mètode:

```
XQPreparedExpression prepareExpression(java.lang.String consulta)
                     throws XQException;
XQPreparedExpression prepareExpression(java.lang.String consulta,
                     XQStaticContext properties) throws XQException;
XQPreparedExpression prepareExpression(java.io.InputStream consulta)
                     throws XQException;
XQPreparedExpression prepareExpression(java.io.InputStream consulta,
                     XQStaticContext properties) throws XQException;
XQPreparedExpression prepareExpression(java.io.Reader consulta)
                     throws XQException;
XQPreparedExpression prepareExpression(java.io.Reader consulta,
                     XQStaticContext properties) throws XQException;
```

En totes aquestes sobrecàrregues, s'obté un objecte `XQPreparedExpression` que portarem a execució amb el mètode `XQPreparedExpression.executeQuery()`. Tres de les sis sobrecàrregues permeten indicar les propietats del context estàtic que es

tindran en compte en avaluar l'expressió, mentre que la resta de sobrecàrregues prenen com a context estàtic l'associat a la connexió.

Les propietats del context estàtic es poden gestionar amb mètodes `get` i `set` de la interfície `XQStaticContext`. La interfície `XQConnection` facilita els mètodes `getStaticContext()` i `setStaticContext()` per recuperar i establir el context estàtic de la connexió.

Els objectes `XQPreparedException` estan pensats per definir una plantilla de sentència "consulta" ideada per a ser executada múltiples vegades, canviant a cada execució, el valor de les variables que incorpora. Així doncs, no té cap sentit crear un objecte `XQPreparedException` sense variables.

Les variables que s'utilitzen en les sentències preparades són sempre variables enllaçades, per prevenir injecció de codi i per garantir la correctesa dels tipus dels valors assignats a les variables. Aquestes variables han d'estar declarades en el prolog de la instrucció *XQuery* a executar i han d'haver estat emplenades abans de procedir a l'execució de la sentència.

La interfície `XQPreparedExpression` disposa de diversos mètodes `get` per obtenir informació sobre el context estàtic i el context dinàmic (variables enllaçades). Disposa, també, del mètode `close()` per tancar l'expressió, alliberant tots els recursos associats, quan ja no sigui necessària. L'execució del mètode `close()` sobre la connexió, tanca totes les expressions definides sobre la connexió. També disposem del mètode `isClosed()` per poder esbrinar si una expressió està oberta o tancada:

```
void close() throws XQException;
boolean isClosed();
```

Fixem-nos que el mètode `executeQuery()` retorna, si tot va bé, un objecte `XQResultSequence`, interfície que deriva de la interfície `XQSequence`, la qual ens facilita un conjunt de mètodes per avaluar la seqüència de resultats obtinguts amb el mètode `executeQuery()`, com el mètode `XQExpression.executeQuery()`.

Exemple de programa amb sentència *XQuery* preparada

El següent programa mostra el rànquing dels països segons la mortalitat infantil (nombre de nens menors d'un any morts per cada mil naixements), en intervals d'amplada vint ([0,20[, [20,40[, [40,60[,...)
Per aconseguir el resultat desitjat, el programa prepara una sentència *XQuery* que cerca els països amb mortalitat infantil en un determinat interval i executa aquesta instrucció tantes vegades com intervals d'amplada 20 hi ha entre 0 i 1000, és a dir, 50 vegades.

Trobareu el fitxer `XQJ12.java` dins el paquet de codi font a l'apartat "Material per desenvolupament sobre SGBD-XML natives utilitzant l'API XQL" dels annexos del web.

```
/*
 * Programa: XQJ12.java
 * Objectiu: Exemple de programa amb sentència XQuery preparada
 *           prevenint la injecció de codi XQuery i errors de tipus
 * Autor...: Isidre Guixà
```

```
 */

package proves;
import javax.xml.xquery.*;
import java.lang.Exception;
import java.io.*;
import javax.xml.namespace.QName;

public class XQJ12
{
  // Amaguem el constructor per defecte. */
  private XQJ12() { }

  private static XQConnection establirConnexio(String[] args)
                throws Exception
  /* Intenta establir una connexió contra el SGBD indicat per paràmetre, que
     ha de ser: eXist-db o BaseX o Sedna */
  { /* Mateix contingut que programa XQJ02.java
       tenint en compte que la base de dades de Sedna és "db" */ }

  public static void main(String[] args){
    XQConnection conn = null;
    try {
      // Obrir sessió:
      conn = establirConnexio(args);
      System.out.println("Connexió establerta amb SGBD " + args[0]);
      // Preparem instrucció, específica per a cada SGBD
      String cad;
      if (args[0].equalsIgnoreCase("eXist-db"))
        cad = "declare variable $finsMortInf external; "+
              "for $i in doc('xqj/mondial.xml')/mondial/"+
              "country[infant_mortality >= $finsMortInf - 20 and "+
              "infant_mortality < $finsMortInf] "+
              "order by number($i/infant_mortality) "+
              "return concat($i/infant_mortality,' - ',$i/name)";
      else if (args[0].equalsIgnoreCase("BaseX"))
        cad = "declare variable $finsMortInf external; "+
              "for $doc in collection('xqj') "+
              "for $i in $doc/mondial/country "+
              "where contains(document-uri($doc),'mondial.xml') and "+
              "$i/infant_mortality >= $finsMortInf - 20 and "+
              "$i/infant_mortality < $finsMortInf "+
              "order by number($i/infant_mortality) "+
              "return concat($i/infant_mortality,' - ',$i/name)";
      else // Sedna
        cad = "declare variable $finsMortInf external; "+
              "for $i in doc('mondial.xml','xqj')/mondial/"+
              "country[infant_mortality >= $finsMortInf - 20 and "+
              "infant_mortality < $finsMortInf] \n"+
              "order by number($i/infant_mortality) "+
              "return concat($i/infant_mortality,' - ',$i/name)";
      // Creem XQPreparedExpression:
      double x;
      XQItemType xqit = conn.createAtomicType(XQItemType.XQBASETYPE_DOUBLE);
      XQPreparedExpression xqpe = conn.prepareExpression(cad);
      for (x = 20.; x<=1000; x=x+20.) {
        // En BaseX (versió 7.1), cal tornar a preparar la sentència
            if (args[0].equalsIgnoreCase("BaseX"))
              xqpe = conn.prepareExpression(cad);
        // Preparem variable d'enllaç "finsMortInf" per definir
        // els extrems dels intervals
```

```
        xqpe.bindDouble(new QName("finsMortInf"), x, xqit);
        XQResultSequence xqrs = xqpe.executeQuery();
        // Mostrem resultats, un a un, convertits a String
        if (xqrs.next()) {
          System.out.println("Interval [" + (x-20) + "," + x + "[:");
          System.out.println("\t"+xqrs.getItemAsString(null));
          while (xqrs.next())
            System.out.println("\t"+xqrs.getItemAsString(null));
        }
      }
    }
    catch (Exception e) {
      e.printStackTrace();
    }
    finally { /* Tanquem connexió en qualsevol cas */
      try {
          if(conn != null) conn.close();
      }
      catch(XQException xe) {
          xe.printStackTrace();
      }
    }
  }
}
```

Comprovem el funcionament d'aquest programa, executant-lo per cadascun dels tres SGBD. L'execució en qualsevol cas és:

```
G:\>java -Dfile.encoding=cp850 proves.XQJ12 nomSGBD
Connexió establerta amb SGBD ...
Interval [0.0,20.0[:
        2.2 - Andorra
        2.4 - Man
        ...
        6.3 - Spain
        ...
        19.6 - Qatar
Interval [20.0,40.0[:
        20 - Saint Lucia
        ...
...
Interval [140.0,160.0[:
        145.82 - Western Sahara
        149.7 - Afghanistan
```

El codi presentat conté la sentència:

```
// En BaseX (versió 7.1), cal tornar a preparar la sentència
   if (args[0].equalsIgnoreCase("BaseX"))
      xqpe = conn.prepareExpression(cad);
```

> Cal estar atents a l'aparició de la llibreria XQJ per *BaseX* similar a la ja existent per *eXist-dB* i *Sedna*, per a utilitzar-la enlloc de la que incorpora *BaseX* (versió 7.1).

És a dir, a la versió 7.1 de *BaseX* cal repetir la preparació de la sentència abans de cada execució. Això és, evidentment, un mal funcionament del mecanisme en aquest SGBD. Comproveu el funcionament si comenteu aquesta sentència condicional.

2.1.5. XQJ per processar documents XML

L'API XQJ permet processar documents XML que no tenen per què estar emmagatzemats a la base de dades, mitjançant els mètodes `bind` adequats per a documents que facilita la interfície `XQDynamicContext`.

Concretament ens interessa conèixer els mètodes:

```
void bindDocument(javax.xml.namespace.QName varName,
                  java.io.Reader value,
                  java.lang.String baseURI,
                  XQItemType type) throws XQException;
void bindDocument(javax.xml.namespace.QName varName,
                  java.io.InputStream value,
                  java.lang.String baseURI,
                  XQItemType type) throws XQException;
void bindDocument(javax.xml.namespace.QName varName,
                  javax.xml.stream.XMLStreamReader value,
                  XQItemType type) throws XQException;
void bindDocument(javax.xml.namespace.QName varName,
                  javax.xml.transform.Source value,
                  XQItemType type) throws XQException;
```

Els dos primers mètodes esperen, en el segon paràmetre `value`, la font XML a gestionar, ja sigui com objecte `Reader` o com objecte `InputStream`, la qual serà analitzada i recuperada com un node document. El tercer paràmetre `baseURI` és opcional (per tant, pot ser `null`) i pot ser utilitzat per resoldre URI relatives o inclòs en missatges d'error. El quart paràmetre ha de ser `null` o `XQITEMKIND_DOCUMENT_ELEMENT` o `XQITEMKIND_DOCUMENT_SCHEMA_ELEMENT`.

Els dos darrers mètodes esperen, en el segon paràmetre `value`, un document XML, ja sigui com objecte `XMLStreamReader` o com objecte `Source`. Recordem que `XQResultSequence.getSequenceAsStream()` obté el resultat d'una consulta com a objecte `XMLStreamReader`.

Exemple de programa que utilitza els motors *XQuery* dels SGBD per processar un fitxer XML resident al sistema de fitxers

Volem executar una instrucció *XQuery* sobre el fitxer `mondial.xml` existent en el sistema de fitxers del sistema operatiu, és a dir, fora de cap BD-XML. Concretament, volem cercar els noms de les autonomies d'Espanya.

> Als annexos de la web trobareu l'apartat "Material per desenvolupament sobre SGBD-XML natives utilitzant l'API XQJ" que inclou l'arxiu `mondial.zip` que conté els fitxers `mondial.xml` i `mondial.dtd`

Per poder executar el programa, necessitem tenir el fitxer `mondial.xml` en una carpeta del nostre sistema d'arxius. Donat que el fitxer incorpora, en la seva segona línia, una referència al fitxer `mondial.dtd` corresponent,

ens caldrà disposar d'aquest fitxer en la mateixa carpeta.

> Trobareu el fitxer `XQJ13.java` dins el paquet de codi font a l'apartat "Material per desenvolupament sobre SGBD-XML natives utilitzant l'API XQL" dels annexos del web.

```
/*
 * Programa: XQJ13.java
 * Objectiu: Exemple de programa que processa, mitjançant l'API XQJ, el
 *           document "mondial.xml" no emmagatzemat a la BD, que cal passar
 *           com a segon argument en la línia d'execució.
 * Autor...: Isidre Guixà
 */

package proves;
import javax.xml.xquery.*;
import java.lang.Exception;
import java.io.*;
import javax.xml.namespace.QName;

public class XQJ13
{
  // Amaguem el constructor per defecte. */
  private XQJ13() { }

  private static XQConnection establirConnexio(String[] args)
                 throws Exception
  /* Intenta establir una connexió contra el SGBD indicat per paràmetre, que
     ha de ser: eXist-db o BaseX o Sedna */
  { /* Mateix contingut que programa XQJ02.java
       tenint en compte que la base de dades de Sedna és "db" */ }

  public static void main(String[] args){
    XQConnection conn = null;
    try {
      // Recuperar el nom del fitxer XML a processar (args[1])
      if (args.length < 2)
        throw new Exception("Introduir la carpeta on resideix \"mondial.xml\""+
                            "com a 2n. paràmetre en la línia d'execució");
      // Creem objecte Reader per obrir l'arxiu XML a processar
      Reader reader;
      try {
        reader = new FileReader(args[1] + File.separator + "mondial.xml");
      }
      catch (IOException e) {
        throw new Exception("Error en intentar accedir al fitxer "+args[1]);
      }
      // Obrir sessió:
      conn = establirConnexio(args);
      System.out.println("Connexió establerta amb SGBD " + args[0]);
      // Preparem instrucció de procés
      String cad = "declare variable $doc external; "+
                   "$doc/mondial/country[name='Spain']/province/name";
      // Creem expressió sobre la que executarem la instrucció
      XQExpression xqe = conn.createExpression();
      // Preparem variable d'enllaç cap un document extern (sistema fitxers)
      xqe.bindDocument(new QName("doc"), reader, null, null);
      XQResultSequence xqrs = xqe.executeQuery(cad);
```

```
      // Mostrem resultats, un a un, convertits a String
      while (xqrs.next())
        System.out.println(xqrs.getItemAsString(null));
    }
    catch (Exception e) {
      e.printStackTrace();
    }
    finally { /* Tanquem connexió en qualsevol cas */
      try {
          if(conn != null) conn.close();
      }
      catch(XQException xe) {
          xe.printStackTrace();
      }
    }
  }
}
```

Comprovem el funcionament d'aquest programa, executant-lo per cadascun dels tres SGBD. L'execució en qualsevol cas hauria de ser:

```
G:\>java -Dfile.encoding=cp850 proves.XQJ12 nomSGBD
Connexió establerta amb SGBD ...
<name>Andalusia</name>
<name>Aragon</name>
<name>Asturias</name>
<name>Balearic Islands</name>
<name>Basque Country</name>
<name>Canary Islands</name>
<name>Cantabria</name>
<name>Castile and Leon</name>
<name>Castile La Mancha</name>
<name>Catalonia</name>
<name>Estremadura</name>
<name>Galicia</name>
<name>Madrid</name>
<name>Murcia</name>
<name>Navarre</name>
<name>Rioja</name>
<name>Valencia</name>
```

El resultat és així en *BaseX*, però en *eXist-db* i *Sedna* apareixen errors diversos.

En *eXist-db*, l'error fa referència a que no troba el fitxer `mondial.dtd` indicat a la segona línia del fitxer `mondial.xml`. Si es comenta aquesta línia, l'error canvia però *eXist-db* continua donant un error diferent, que desapareix en eliminar la segona línia del document XML. Llavors s'obté el resultat esperat. Sembla que es tracta d'un error de l'API XQJ per a *eXist-db*.

En *Sedna* es genera l'excepció `ArrayOutOfBoundsException`, independentment que la segona línia del document `mondial.xml` hi sigui o estigui comentada o no hi sigui.

En conseqüència, observem que els motors *XQuery* dels SGBD ens poden ajudar a gestionar arxius XML externs a les BD via l'API XQJ.

Però, cal estar alerta, doncs per defecte, l'API XQJ intenta emmagatzemar tot el docuemtn XML en memòria, com fa el DOM (Document Object Model), i això pot provocar problemes de memòria quan el document XML a processar té una grandària considerable (de l'ordre de gigabytes).

Per evitar això i assegurar-se que XQJ no actuï d'aquesta manera, XQJ proporciona

l'enllaç diferit, que permet no carregar l'arxiu en memòria i accedir a l'arxiu, en el moment de procés, segons les necessitats de la sentència a executar.

> La documentació de la interfície `XQDynamicContext` detalla el funcionament de l'enllaç immediat i diferit.

Per evitar l'enllaç immediat (opció per defecte) cal establir l'enllaç diferit, cosa que s'assoleix fent:

```
// 1r. Crear un nou objecte XQStaticContext basat en el context estàtic actual:
XQStaticContext propietats = conn.getStaticContext();

// 2n. Establir la propietat "mode d'enllaç" a diferit:
propietats.setBindingMode(XQConstants.BINDING_MODE_DEFERRED);

// 3r. Establir les noves propietats com a context a emprar, fet que es pot
// activar en tres punts diferents:

// a) Per a una determinada sentència d'execució immediata:
XQExpression xqe = conn.createExpression (propietats);

// b) Per a una determinada sentència preparada:
String sentencia = "declare...";
XQPreparedExpression xqpe = conn.prepareExpression(sentencia, propietats);

// c) Per a totes les totes les noves expressions (canvi global):
conn.setStaticContext(propietats);
```

Exemple de programa que utilitza els motors *XQuery* dels SGBD per processar un fitxer XML resident al sistema de fitxers, amb enllaç diferit.

Aquest exemple és la repetició de l'exemple anterior, per obtenir els noms de les autonomies d'Espanya a partir del fitxer `mondial.xml` resident en el sistema d'arxius, utilitzant enllaç diferit.

> Trobareu el fitxer `XQJ14.java` dins el paquet de codi font a l'apartat “Material per desenvolupament sobre SGBD-XML natives utilitzant l'API XQL” dels annexos del web.

Mostrem només els canvis respecte l'anterior solució sense enllaç diferit:

```
...
XQStaticContext properties = conn.getStaticContext();
properties.setBindingMode(XQConstants.BINDING_MODE_DEFERRED);
// Creem expressió sobre la que executarem la instrucció
XQExpression xqe = conn.createExpression(properties);
...
```

L'execució en *BaseX* continua funcionant correctament. I en *eXist-db* i *Sedna,* prèvia eliminació de la segona línia de l'arxiu `mondial.xml`, també funciona. És a dir, en *Sedna* ha desaparegut l'excepció `ArrayIndexOutOfBoundsException` existent quan no estava activat l'enllaç diferit.

2.2. API XML:DB

XML:DB, també anomenada **XAPI**, és una API ideada pel grup XML:DB, iniciativa apareguda l'any 2000 per intentar aconseguir un mecanisme estàndard d'accés a les BD-XML natives, de manera similar al mecanisme JDBC per a les BDR, davant els mecanismes propietaris que cada SGBD-XML natives es veia obligat a dissenyar. El grup està inactiu des del 2003, la qual cosa fa pensar que el recorregut d'aquesta API ja ha finalitzat.

L'any 2003, precisament any en que va cessar l'activitat del grup XML:DB, va néixer l'API XQJ, dissenyada com un projecte JCP (*Java Community Process*), de la qual l'any 2009 se n'ha publicat la versió definitiva. Sembla, doncs, que el futur està en la utilització de l'API XQJ, però això no desmereix una ullada a l'API XML:DB, implementada en diversos SGBD-XML natives.

Nombrosos SGBD-XML natives faciliten la connectivitat des de Java mitjançant aquesta API, de manera que podem desenvolupar aplicacions que accedeixin a SGBD-XML natives via XML:DB amb la única particularitat d'haver d'utilitzar el connector que facilita cada SGBD.

A continuació desenvoluparem aplicacions Java que connecten contra SGBD-XML natives via XML:DB i en comprovarem la seva correcta execució en tres SGBD-XML natives: *eXist-db*, *BaseX* i *Sedna*. Per això necessitem tenir instal·lats els tres SGBD i disposar dels connectors adequats per a cada SGBD.

> Als annexos de la web trobareu els apartats “Introducció al SGBD-XML natives ...” per als SGBD *eXist-db*, *BaseX* i *Sedna*, amb les indicacions per instal·lar aquest SGBD i tenir-hi una primera presa de contacte.
> Als annexos de la web trobareu els apartats “Material per desenvolupament sobre SGBD-XML natives utilitzant l'API XML:DB” on hi ha una llibreria XML:DB, desenvolupada per Charles Foster, pel SGBD *Sedna*.

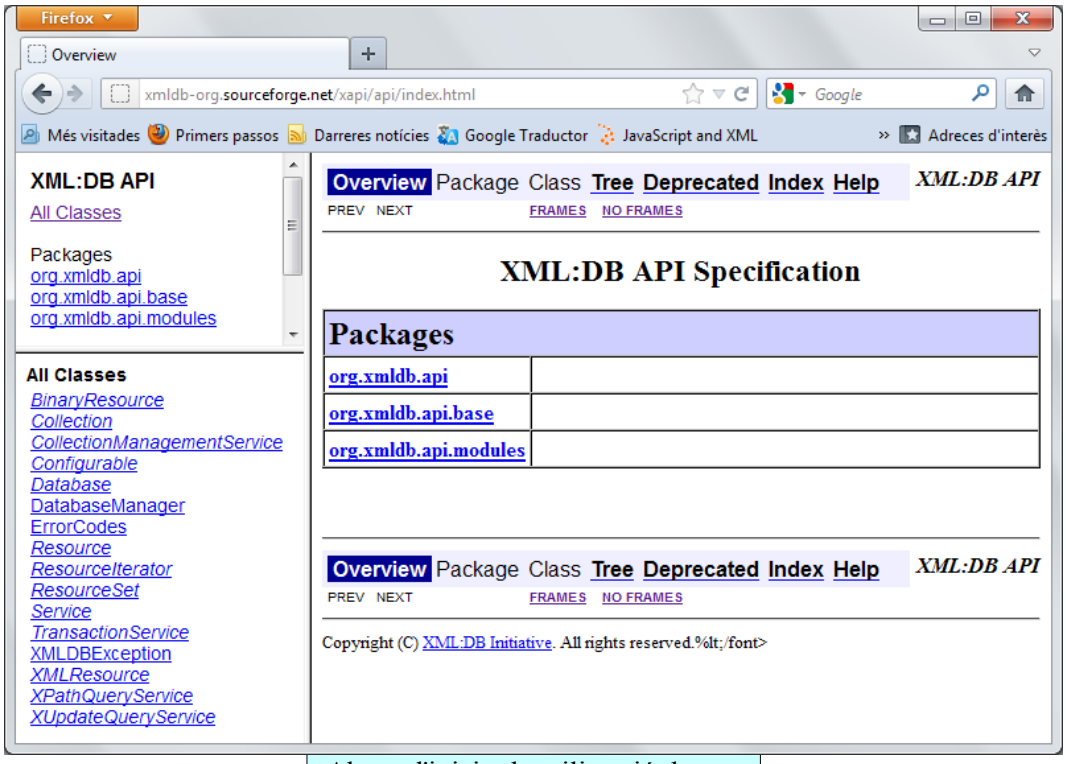
Si fem una ullada a la documentació (`javadoc` de la figura 2-1) de l'API XML:DB, veurem que està constituïda per un gran nombre d'interfícies i unes poques classes. Això és així per què cada SGBD ha de proveir les classes que implementen les interfícies que dictamina l'API. En el desenvolupament d'aplicacions amb aquesta API utilitzarem sempre referències a aquestes interfícies i mai utilitzarem directament referències a les classes subministrades pel SGBD. D'aquesta manera aconseguirem dissenyar aplicacions que es puguin utilitzar per gestionar BD en diversos SGBD que implementen l'API XML:DB.

> La documentació oficial (javadoc)

de l'API XML:DB al portal web oficial de l'API XML:DB.
Els SGBD amb què estem practicant, han evolucionat l'API oficial i, en conseqüència, convé també disposar de la corresponent documentació:
Per *Sedna*, la podem trobar a l'arxiu zip que conté l'API XML:DB, desenvolupada per Charles Foster.
Per *eXist-db*, la podem trobar a la documentació local del SGBD: http://localhost:8080/exist/devguide_xmldb.xml
BaseX no facilita cap documentació específica.

Així doncs, ens interessa conèixer les interfícies que aporta l'API XML:DB i emprar-les en el disseny d'aplicacions que accedeixin als SGBD-XML natives (concretament: *eXist-db*, *BaseX* i *Sedna).*

Figura2. Documentació oficial de l'API XML:DB



Abans d'iniciar la utilització de l'API XML:DB, comentar que en el portal web oficial de l'API XML:DB hi ha l'espai *API Use Cases* que incorpora els escenaris més freqüents d'utilització de l'API amb solucions proposades.

2.2.1. Establiment de connexió

Les aplicacions que accedeixen a BD necessiten, com a primer pas per poder gestionar les dades de la BD, establir la connexió amb la BD a gestionar. En el cas de l'API XML:DB, per poder intentar establir la connexió a una base de dades, es necessita el pas previ d'enregistrar el SGBD-XML dins la classe `DatabaseManager`, cosa que es fa executant les dues instruccions següents:

```
Database dbDriver = new ClasseEspecíficaQueGeneraDatabase();
DatabaseManager.registerDatabase(dbDriver);
```

La primera línia del codi anterior mostra com crear un objecte `Database` específic del SGBD al que ens volem connectar. A la segona línia, el mètode `DatabaseManager.registerDatabase()` registra l'objecte `Database` proporcionat pel SGBD-XML a l'API XML:DB.

Per cada SGBD-XML natives amb el que vulguem connectar via XML:DB, ens cal saber la classe específica facilitada pel SGBD que implementa la interfície `Database`. La taula 2-2 recull les classes adequades pels SGBD amb els que practiquem.

Taula 2-2. Classes que implementen la interfície Database en diversos SGBD

SGBD	Classe
eXist-db	`org.exist.xmldb.DatabaseImpl`
BaseX	`org.basex.api.xmldb.BXDatabase`
Sedna	`net.cfoster.sedna.DatabaseImpl`

Així doncs, si volem assolir un programa per connectar-nos amb un SGBD *eXist-db*, per enregistrar el SGBD *eXist-db*, caldria escriure quelcom similar a:

```
Database dbDriver = new org.exist.xmldb.DatabaseImpl();
DatabaseManager.registerDatabase(dbDriver);
```

El codi anterior és codi específic per al SGBD *eXist-db*, i ens interessa, a ser possible, escriure codi que sigui independent, en el major grau possible, del SGBD. Per tant, ens interessa obtenir l'objecte `Database` de forma genèrica, sense cridar cap mètode específic de l'API subministrada pel SGBD. Per aconseguir això escriurem:

```
String driver = "nomClasseEspecíficaQueGeneraXQDataSource";
Database dbDriver = (XQDataSource)Class.forName(driver).newInstance();
DatabaseManager.registerDatabase(dbDriver);
```

En el codi anterior, la cadena `driver` ha de contenir el nom de la classe específica del SGBD que genera l'objecte `Database` (taula 2-2), però l'avantatge respecte la situació anterior és que el contingut de la cadena `driver` no té per què residir en el codi font del programa, sinó que es pot llegir d'un fitxer de configuració o es pot recollir via paràmetre, entre d'altres possibilitats.

El següent programa ens serveix per comprovar que disposem de les classes adequades per treballar, via XML:DB amb els SGBD-XML *eXist-db*, *BaseX* i *Sedna*. Observeu, amb atenció, el requeriment per la correcta execució, incorporat a la capçalera del font.

Trobareu el fitxer `XMLDB01.java` dins el paquet de codi font a l'apartat "Material per desenvolupament sobre SGBD- XML natives utilitzant l'API " dels annexos del web.

```
/*
 * Programa: XMLDB01.java
 * Objectiu: Programa que enregistra els SGBD-XML exist-db, BaseX i Sedna
 *           dins la classe DatabaseManager de l'API XML:DB
 * Autor...: Isidre Guixà
 */

/* Requeriment:
   Cal que els paquets on resideixen les llibreries XMLDB facilitades pel SGBD
   resideixin en el CLASSPATH actiu. Els paquets a accedir són:
   * eXist-db: Directori "lib\core" i arxiu exist.jar instal·lats en el
               procés d'instal·lació del SGBD (no es necessita cap llibreria
               addicional)
   * BaseX: Arxius basex.jar, xmldb-api-1.0.jar i basex-api.jar instal·lats
            en el procés d'instal·lació del SGBD (no es necessita cap
            llibreria addicional)
   * Sedna: Directori arrel del fitxer que conté l'API XML:DB per Sedna, que
            conté els fitxers xmldb.jar i sedna-xmldb.jar
   Hi ha alguna incompatibilitat entre els .jar subministrats pels diferents
   SGBD, de manera que si es vol executar aquest programa tenint carregat, en
   el CLASSPATH, els paquets dels 3 SGBD, cal seguir l'ordre següent:
   1r. Paquets de BaseX / 2n. Paquets de Sedna / 3r. Paquets d'eXist-db
   Si només es tenen carregats els paquets d'un SGBD, per la resta de SGBD el
   programa informarà que no es troba la corresponent classe
*/
package proves;
import org.xmldb.api.base.*;
import org.xmldb.api.DatabaseManager;

public class XMLDB01
{
  private static final String sgbd[] = {"eXist-db","BaseX","Sedna"};
  private static final String driver[] = {"org.exist.xmldb.DatabaseImpl",
                                          "org.basex.api.xmldb.BXDatabase",
                                          "net.cfoster.sedna.DatabaseImpl"};
  // Amaguem el constructor per defecte. */
  private XMLDB01() { }

  public static void main(String[] args){
    for (int i=0; i<sgbd.length; i++) {
      try {
        System.out.println("Intentem registrar el SGBD "+sgbd[i]);
        Database dbDriver =(Database) Class.forName(driver[i]).newInstance();
        DatabaseManager.registerDatabase(dbDriver);
        System.out.println("Registre efectuat");
        System.out.println("Nivell de conformitat: " +
                           dbDriver.getConformanceLevel() + "\n");
      }
      catch (ClassNotFoundException cf){
        System.out.println("No es troba la classe: "+cf.getMessage());
      }
      catch (InstantiationException ie){
        System.out.println("No es pot instanciar la classe: "+ie.getMessage());
```

```
      }
      catch (IllegalAccessException ia){
        System.out.println("Classe o constructor no accessibles: "+
                           ia.getMessage());
      }
      catch (XMLDBException xdbe){
        System.out.println("Error en registrar el driver: "+xdbe.getMessage());
      }
    }
    Database dbDrivers[] = DatabaseManager.getDatabases();
    System.out.println("\nSGBD registrats: "+dbDrivers.length);
    if (dbDrivers.length>0) {
      System.out.print("Noms:");
      for (int i=0; i<dbDrivers.length; i++) {
        try {
          System.out.print (" "+dbDrivers[i].getName());
        }
        catch (XMLDBException xdbe){
          xdbe.printStackTrace();
        }
      }
      System.out.println();
    }
  }
}
```

Vegem-ne l'execució:

```
G:\XMLDB>java -Dfile.encoding=cp850 proves.XMLDB01
Intentem registrar el SGBD eXist-db
Registre efectuat
Nivell de conformitat: 0

Intentem registrar el SGBD BaseX
Registre efectuat
Nivell de conformitat: 0

Intentem registrar el SGBD Sedna
-------------------------------------------------------
Sedna XML:DB API Client started, Version 1.2.4 08/Feb/11
Copyright (C) 2007 Charles Foster, www.cfoster.net.
-------------------------------------------------------
Registre efectuat
Nivell de conformitat: 1

SGBD registrats: 3
Noms: exist basex sedna
```

El programa intenta registrar cadascun dels tres SGBD *eXist-db*, *BaseX* i *Sedna* i, en cas d'aconseguir-ho, informa sobre el nivell de conformitat de cada SGBD amb el mètode `Database.getConformanceLevel()`. Finalment, utilitza el mètode `Database.getDatabases()` per obtenir la llista dels SGBD registrats i, per cadascun, utilitza el mètode `Database.getName()` per mostrar-ne el nom. La compilació d'aquest programa informa que s'està utilitzant el mètode obsolet `getName()`. Aquest mètode ha estat substituït pel mètode `getNames()`, implementat per *eXist-db* i *Sedna*, però *BaseX* no en proporciona la nova versió i, per tant, cal continuar utilitzant `getName()` per tenir un programa que pugui ser executat en els tres SGBD.

Trobareu informació completa sobre els nivells de conformitat de l'API XML:DB en el portal web oficial de l'API XML:DB

Nivell de conformitat de l'API XML:DB

El grup XML:DB va definir uns nivells de conformitat per l'API XML:DB, de manera que una implementació de l'API per a un SGBD verifica un determinat nivell de conformitat si implementa un conjunt determinat de funcionalitats. Es va arribar a definir dos nivells de conformitat:

- *Core Level 0*, en implementar els mòduls `API Base` i `XMLResource`
- *Core Level 1*, en implementar tots els mòduls del *Core Level 0* i, a més, `XPathQueryService`.

El programa anterior ens mostra que *eXist-db* i *BaseX* compleixen el nivell 0 i *Sedna* el nivell 1.

Fixem-nos que per a executar el programa anterior, no cal que els SGBD estiguin en marxa, doncs únicament cerca les classes corresponents per poder connectar amb els SGBD. Cal, però, que les classes estiguin en el `CLASSPATH` actiu.

La classe `DatabaseManager`, a banda dels mètodes ja utilitzats, també proporciona altres mètodes interessants:

- `setProperty()`, per establir una propietat pel `DatabaseManager`.
- `getProperty()`, per recuperar una propietat del `DatabaseManager`.
- `getDatabase()`, per recuperar l'objecte `Database` (SGBD) associat a una `uri` indicada per paràmetre.
- `deRegisterDatabase()`, per eliminar el registre d'una `Database` (SGBD) registrada, indicada per paràmetre.
- `getCollection()`, per establir connexió amb una base de dades.

Una vegada el SGBD és registrat, ja estem en condicions de crear un objecte `Collection` per establir la sessió de treball amb el SGBD i això ho aconseguim amb el mètode `getCollection()` de la classe `DatabaseManager`. L'API XML:DB facilita dues sobrecàrregues del mètode `getCollection()`:

```
Collection getCollection(java.lang.String URI)
                                throws XMLDBException;
Collection getCollection(java.lang.String URI,
                         java.lang.String username,
                         java.lang.String password)
                                throws XMLDBException;
```

Les dues modalitats intenten establir una connexió contra el SGBD, tot obtenint una instància d'una col·lecció que ha d'existir a la base de dades.

En el primer paràmetre, anomenat `URI`, existent en ambdós mètodes, cal indicar la base de dades i col·lecció a connectar. La sintaxis d'aquesta cadena és específica de cada SGBD, però ha de començar per la cadena `xmldb:` i acostuma a anar seguida d'un identificador `vendor:` del fabricant del SGBD i posteriorment la identificació de la màquina, el port, la base de dades i/o la col·lecció. És a dir, el primer paràmetre té la

forma `xmldb:vendor://host:port/path/to/collection`. La taula 2-3 recull la URI a utilitzar pels SGBD amb els què practiquem.

Taula 2-3. URI a utilitzar en el mètode getCollection(), segons el SGBD

SGBD	URI
eXist-db	`xmldb:exist://host:8080/exist/xmlrpc/db/...`
BaseX	`xmldb:basex://localhost:1984/nomBD`
Sedna	`xmldb:sedna://host:5050/nomBD/...`

A la taula 2-3 s'observa que la `URI` per *BaseX* ha d'indicar obligatòriament la màquina local. Això és degut a que l'API XML:DB de *BaseX* (versió 7.1) no implementa l'arquitectura client-servidor i s'ha d'indicar obligatòriament la cadena `localhost:1984` (qualsevol altre valor produirà una `XMLDBException: Invalid URI`).

> *Note that we recommend everyone to use our own APIs, as they offer better performance and are better supported by our core team. The use of the `XML:DB` and `XQJ` APIs is discouraged: as these APIs do not utilize the client/server architecture of BaseX, their use may lead to conflicting database access operations.*
> *Documentació de BaseX (versió 7.1)*

La taula 2-3 també mostra que cada `URI` finalitza amb el nom de la base de dades, que pot anar acompanyat del camí a la col·lecció amb la què treballar; cal indicar, com a mínim, el nom de la base de dades. En el cas d'*eXist-db*, com que només pot tenir una base de dades, s'indica el nom de la base de dades que crea el procés d'instal·lació: `db`. En el cas de *BaseX*, com que no admet col·leccions, només es pot indicar el nom de la base de dades. *Sedna* és l´únic SGBD (dels tres que estem utilitzant) que permet treballar amb bases de dades i col·leccions.

La primera modalitat de mètode `getCollection()`, amb només el paràmetre `URI`, només es pot utilitzar quan la base de dades a accedir no precisa autenticació. En cas d'haver d'indicar usuari i contrasenya, cal utilitzar la segona modalitat.

El mètode `getCollection()` retorna, si tot és correcte, un objecte que implementa la interfície `Collection`, el qual utilitzarem per gestionar els continguts de la col·lecció a la que s'ha connectat. Quan la connexió ja no es necessita, cal recordar sempre de tancar-la amb el mètode `Collection.close()`, per tal d'alliberar tots els recursos assignats pel sistema operatiu per a mantenir la connexió.

El següent programa mostra com establir connexió amb cadascun dels tres SGBD indicats. Cal tenir present que aquest programa (i tots els que segueixin):

- Obliga a passar per paràmetre el nom del SGBD.
- Considera que l'usuari i la contrasenya per a cada SGBD són els que el procés d'instal·lació facilita per defecte (`admin/admin` per a *eXist-db* i *BaseX* i `SYSTEM/MANAGER` per *Sedna).*
- Pels SGBD *eXist-db* i *Sedna,* que són els SGBD que suporten l'arquitectura client/servidor, es considera que el servidor resideix a la mateixa màquina des d'on s'executarà el programa `(localhost)` i que el port pel que s'estableix la comunicació TCP/IP és el que el procés d'instal·lació facilita per defecte `(8080` per *eXist-db* i `5050` per *Sedna).*

- Intenta establir connexió a la base de dades/col·lecció indicada a continuació, segons el SGBD:

> Als annexos de la web trobareu l'apartat "Material per desenvolupament sobre SGBD-XML natives utilitzant l'API XML:DB" que inclou l'arxiu `mondial.xml` i les instruccions d'instal·lació en els diversos SGBD per a poder executar els programes aquí desenvolupats.

  - En *eXist-db*, que només permet una base de dades que pot contenir múltiples col·leccions a diferents nivells, a una col·lecció de nom `xmldb`, que conté l'arxiu `mondial.xml`.
  - En *BaseX*, que no permet col·leccions però sí múltiples bases de dades, a una base de dades de nom `xmldb`, que conté l'arxiu `mondial.xml`.
  - En *Sedna*, que permet múltiples bases de dades i dins d'elles, múltiples col·leccions d'un únic nivell, a una base de dades de nom `db`, amb una col·lecció de nom `xmldb`, que conté l'arxiu `mondial.xml`.

> Trobareu el fitxer `XMLDB02.java` dins el paquet de codi font a l'apartat "Material per desenvolupament sobre SGBD- XML natives utilitzant l'API XQL" dels annexos del web.

```
/*
 * Programa: XMLDB02.java
 * Objectiu: Programa que mostra com establir connexió amb tres SGBD:
 *           eXist-db, BaseX i Sedna
 * Autor...: Isidre Guixà
 */

package proves;
import org.xmldb.api.base.*;
import org.xmldb.api.DatabaseManager;

public class XMLDB02
{
  // Amaguem el constructor per defecte. */
  private XMLDB02() { }

  private static Collection establirConnexio(String[] args)
                throws Exception
  /* Intenta establir una connexió contra el SGBD indicat per paràmetre, que
     ha de ser: eXist-db o BaseX o Sedna */
  {
    if (args.length<1){
      throw new Exception("Executeu el programa indicant, primer, el SGBD "+
                          "al que connectar: eXist-db, BaseX o Sedna.");
    }
    Database dbDriver;
    Collection coll;
    if (args[0].equalsIgnoreCase("eXist-db")) {
      dbDriver = (Database)
          Class.forName("org.exist.xmldb.DatabaseImpl").newInstance();
```

```
      DatabaseManager.registerDatabase(dbDriver);
      coll = DatabaseManager.getCollection(
          "xmldb:exist://localhost:8080/exist/xmlrpc/db/xmldb",
          "admin", "admin");
    }
    else if (args[0].equalsIgnoreCase("BaseX")) {
      dbDriver = (Database)
          Class.forName("org.basex.api.xmldb.BXDatabase").newInstance();
      DatabaseManager.registerDatabase(dbDriver);
      coll = DatabaseManager.getCollection(
          "xmldb:basex://localhost:1984/xmldb", "admin", "admin");
    }
    else if (args[0].equalsIgnoreCase("Sedna")) {
      dbDriver = (Database)
          Class.forName("net.cfoster.sedna.DatabaseImpl").newInstance();
      DatabaseManager.registerDatabase(dbDriver);
      coll = DatabaseManager.getCollection(
          "xmldb:sedna://localhost:5050/db/xmldb", "SYSTEM", "MANAGER");
    }
    else
      throw new Exception("Executeu el programa indicant, primer, el SGBD "+
                          "al que connectar: eXist-db, BaseX o Sedna.");
    return coll;
  }

  public static void main(String[] args){
    Collection coll = null;
    try {
      coll = establirConnexio(args);
      System.out.println("Connexió establerta amb SGBD " + args[0]);
    }
    catch (Exception e) {
      e.printStackTrace();
    }
    finally { /* Tanquem connexió en qualsevol cas */
      try {
          if(coll != null) {
            coll.close();
            System.out.println("Connexió tancada");
          }
      }
      catch(XMLDBException xe) {
          xe.printStackTrace();
      }
    }
  }
}
```

L'execució del programa `XMLDB02` ha de tenir actiu, com a `CLASSPATH`, únicament els paquets corresponents al SGBD amb el què treballar, ja que hi ha incompatibilitat entre els arxius `JAR` de *BaseX, eXist-db* i *Sedna*.

L'execució del programa finalitza amb error (convenientment documentat) si:

- No s'indica, a l'hora d'executar el programa, cap SGBD.
- S'indica, a l'hora d'executar el programa, un nom de SGBD no suportat.

- No es troba, en el `CLASSPATH` actiu, la llibreria XMLDB corresponent al SGBD-XML natives amb el què es vol connectar.
- En cas del SGBD *Sedna*, el servidor està aturat o la base de dades indicada està aturada o el port és erroni o l'usuari o la contrasenya són erronis.

Particularitats de les connexions XML:DB segons el SGBD

En *eXist-db*, l'establiment de connexió del programa anterior no es queixa, si el servidor és aturat o el port o l'usuari o la contrasenya són erronis, i el corresponent error es posposa al moment en què s'intenta tancar la connexió. Aquest funcionament és degut a que el SGBD *eXist-db* treballa amb connexions no persistents i l'establiment de la connexió (`getCollection()`) prepara la connexió, que de veritat no s'utilitza fins que es fa alguna cosa amb ella, com per exemple, en el programa anterior, en tancar la connexió. Així doncs, haurem de tenir present que en *eXist-db*, el fet que el mètode `DatabaseManager.getCollection()` no llenci una excepció, no és garantia de que la connexió s'hagi establert correctament.
En *BaseX* (versió 7.1), l'execució ha de ser obligatòriament local (doncs no utilitza l'arquitectura client/servidor) i, en conseqüència, no precisa que el servidor *BaseX* estigui engegat,

---

2.2.2. Gestió de col·leccions

L'API XML:DB permet establir connexió amb un SGBD-XML natives amb el mètode `DatabaseManager.getCollection()`, en el qual cal indicar la base de dades amb la que s'estableix la connexió i, si cal, una col·lecció específica.

> Recordem que no tots el SGBD-XML natives faciliten la gestió de col·leccions.

La interfície `Collection` facilita diversos mètodes per gestionar col·leccions:

- `getName()`, per recuperar el nom de la col·lecció actual.
- `getChildCollectionCount()`, per conèixer el nombre de col·leccions incloses dins la col·lecció actual.
- `listChildCollections()`, per obtenir una llista dels noms de les col·leccions existents dins la col·lecció actual.
- `getParentCollection()`, per obtenir una instància a la col·lecció pare de la col·lecció actual o `null` si estem a l'arrel de la base de dades.
- `getChildCollection()`, per obtenir una instància a la col·lecció filla indicada per paràmetre o `null` si no existeix la col·lecció amb el nom indicat.

La creació i eliminació de col·leccions el proporciona la interfície `CollectionManagementService` amb els mètodes:

- `createCollection()`, per crear una nova col·lecció dins la base de dades.
- `removeCollection()`, per eliminar una col·lecció.

En ambdós mètodes, el nom de la col·lecció a crear o a eliminar és relatiu a la col·lecció

des de la qual s'ha obtingut l'objecte `CollectionManagementService` que s'utilitza per invocar els mètodes, el qual s'obté amb algun dels dos mètodes següents de la interfície `Collection:`

- `getServices()`, per obtenir una llista de tots els serveis proporcionats per la col·lecció
- `getService()`, per obtenir un objecte `Service` d'entre els serveis que proporciona la col·lecció, com un objecte `CollectionManagementService`.

El mètode `createCollection()` no genera cap error en cas que ja existeixi una col·lecció de nom igual a la que es pretén crear; es manté la col·lecció existent.

El mètode `removeCollection()` elimina la col·lecció, amb tot el seu contingut.

Particularitats de la gestió de col·leccions via XML:DB segons el SGBD

En *Sedna,* les úniques formes per establir connexió a una col·lecció són els mètodes `getCollection()`, `getChildCollection()` i `getParengCollection()`. El primer mètode crea un sòcol contra el servidor i, els altres mètodes, utilitzen el mateix sòcol. Així doncs, en executar el mètode `close()` sobre una col·lecció, hem de tenir en compte que es tancarà el sòcol pel que tenim establerta la connexió amb la col·lecció i, de retruc, es tancaran tota la resta de col·leccions que utilitzaven el mateix sòcol.
En *Sedna* sabem, per una banda, que no es permet col·leccions jeràrquiques (és a dir, una col·lecció no pot contenir altres col·leccions i només hi pot haver col·leccions a l'arrel de la base de dades). Per altra, XML:DB diu, textualment, que una base de dades XML pot exposar col·leccions com un conjunt jeràrquic de col·leccions pares i fills. Davant aquesta contraposició, l'API XML:DB de *Sedna* emula el muntatge de col·leccions jeràrquiques, de manera que: si estem connectats a una col·lecció anomenada `mevaCol:`

- Si estem connectats a una col·lecció anomenada `mevaCol`, (que obligatòriament resideix a l'arrel de la base de dades), el mètode `createCollection(filla)`, crea una nova col·lecció a l'arrel, anomenada `mevaCol/filla`. Per a *Sedna* és una nova col·lecció, totalment independent de `mevaCol,` però per l'API XML:DB n'és una col·lecció filla.

- Si tenim diverses col·leccions anomenades `mevaCol/xxx` i també la col·lecció `mevaCol` (per Sedna, totes elles són col·leccions independents que pengen de l'arrel), i estem connectats a l'arrel de la base de dades, el mètode `removeCollection(mevaCol)` elimina la col·lecció `mevaCol` i totes les col·leccions anomenades `mevaCol/xxx`, doncs per l'API XML:DB de *Sedna,* aquestes últimes eren filles de la primera.

En *BaseX*, on no hi ha col·leccions i sempre s'està connectat a una base de dades, el mètode `createCollection()` crea una nova base de dades que passa a ser la base de dades on estem connectats i el mètode `removeCollection()` elimina una base de dades, a la qual no podem estar connectats.

Exemple de programa que gestiona col·leccions i mostra els serveis

Trobareu el fitxer `XMLDB03.java` dins el paquet de codi font a l'apartat "Material per desenvolupament sobre SGBD- XML natives utilitzant l'API XMLDB" dels annexos del web.

```
/*
 * Programa: XMLDB03.java
 * Objectiu: Exemple de gestió de col·leccions i recuperació de serveis
 * Autor...: Isidre Guixà
 */

package proves;
import org.xmldb.api.base.*;
import org.xmldb.api.DatabaseManager;
import org.xmldb.api.modules.*;

public class XMLDB03
{
  // Amaguem el constructor per defecte. */
  private XMLDB03() { }

  private static Collection establirConnexio(String[] args)
                throws Exception
  /* Intenta establir una connexió contra el SGBD indicat per paràmetre, que
     ha de ser: eXist-db o BaseX o Sedna */
  { /* Mateix contingut que programa XMLDB02.java */ }

  public static void main(String[] args){
    Collection coll = null;
    try {
      coll = establirConnexio(args);
      System.out.println("Connexió establerta amb SGBD " + args[0]);
      // Recuperem el nom de la col·lecció actual
      System.out.println("Col·lecció actual: "+coll.getName());
      // Obtenim la col·lecció pare, si en té
      Collection pare = coll.getParentCollection();
      if (pare==null)
        System.out.println("Estem a l'arrel. No hi ha pare.");
      else {
        System.out.println("Col·lecció pare: "+pare.getName());
        int fills = pare.getChildCollectionCount();
        System.out.println("Nombre de fills del pare: "+fills);
        if (fills>0) {
          System.out.println("Noms dels fills:");
          String nomFills[] = pare.listChildCollections();
          for (int i=0; i<fills; i++)
            System.out.println("\t"+nomFills[i]);
        }
        if (!(args[0].equalsIgnoreCase("Sedna"))) pare.close();
      }
      // Vegem la llista de serveis que facilita el SGBD
      Service serveis[] = coll.getServices();
      System.out.println("Serveis proporcionats:");
      int cMS=-1;
      for (int i=0; i<serveis.length; i++) {
        System.out.println("\t"+serveis[i].getName()+
                           " - Versió: "+serveis[i].getVersion());
```

```
        if (serveis[i].getName().equals("CollectionManagementService"))
          cMS = i;
      }
      // cMS conté la posició de la taula "serveis" que conté el servei
      // que permet crear i eliminar col·lecions
      // Crear cinc col·leccions dins la col·lecció actual
      if (cMS>=0) {
        System.out.println("Intentarem crear 5 col·leccions filles!");
        CollectionManagementService cms =
            (CollectionManagementService)serveis[cMS];
        for (int i=0; i<5; i++) cms.createCollection("aux"+i);
        // Comprovem llista de col·leccions filles de la col·lecció actual
        int fills = coll.getChildCollectionCount();
        System.out.println("Nombre de fills: "+fills);
        if (fills>0) {
          System.out.println("Noms dels fills:");
          String nomFills[] = coll.listChildCollections();
          for (int i=0; i<fills; i++)
            System.out.println("\t"+nomFills[i]);
          System.out.println("Després d'haver creat les col·leccions, "+
                             "estem situats a la col·lecció: "+coll.getName());
          // Procedim a eliminar les col·leccions creades
          System.out.println("Eliminem les 5 col·leccions creades!");
          for (int i=0; i<5; i++) cms.removeCollection("aux"+i);
          System.out.println("Nre. de fills: "+coll.getChildCollectionCount());
        }
        if (args[0].equalsIgnoreCase("BaseX"))
          System.out.println("BaseX. Estem situats a la col·lecció: "+
                             coll.getName());      }
    }
    catch (Exception e) {
      e.printStackTrace();
    }
    finally { /* Tanquem connexió en qualsevol cas */
      try {
          if(coll != null) {
            coll.close();
            System.out.println("Connexió tancada");
          }
      }
      catch(XMLDBException xe) {
          xe.printStackTrace();
      }
    }
  }
}
```

L'execució en *eXist-db*:

```
G:\>java -Dfile.encoding=cp850 proves.XMLDB03 exist-db
Connexió establerta amb SGBD exist-db
Col·lecció actual: /db/xmldb
Col·lecció pare: /db
Nombre de fills del pare: 12
Noms dels fills:
        twitter
        xinclude
        library
        xqdocs
        mondial
        system
        shakespeare
        mods
```

```
        xproc
        xmlad
        xmldb
        betterform
Serveis proporcionats:
        XPathQueryService - Versió: 1.0
        CollectionManagementService - Versió: 1.0
        UserManagementService - Versió: 1.0
        DatabaseInstanceManager - Versió: 1.0
        IndexQueryService - Versió: 1.0
        XUpdateQueryService - Versió: 1.0
        ValidationService - Versió: 1.0
Intentarem crear 5 col·leccions filles!
Nombre de fills: 5
Noms dels fills:
        aux3
        aux2
        aux1
        aux0
        aux4
Després d'haver creat les col·leccions, estem situats a la col·lecció:
/db/xmldb
Eliminem les 5 col·leccions creades!
Nre. de fills: 0
Connexió tancada
```

Execució en *BaseX*:

```
G:\>java -Dfile.encoding=cp850 proves.XMLDB03 basex
Connexió establerta amb SGBD basex
Col·lecció actual: xmldb
Estem a l'arrel. No hi ha pare.
Serveis proporcionats:
        XPathQueryService - Versió: 1.0
        XQueryQueryService - Versió: 1.0
        CollectionManagementService - Versió: 1.0
Intentarem crear 5 col·leccions filles!
Nombre de fills: 0
BaseX. Estem situats a la col·lecció: aux4
Connexió tancada
```

Observem que, després d'intentar crear les col·leccions, el nombre de col·leccions filles continua sent zero, doncs *BaseX* no admet col·leccions Ara bé, si anem a consultar el SGBD (via la interfície gràfica), veurem que s'ha creat 5 bases de dades (tantes com col·leccions hem creat). A més, comprovem que la connexió ha canviat a la darrera base de dades creada.

Execució en *Sedna*:

```
G:\>java -Dfile.encoding=cp850 proves.XMLDB03 sedna
-------------------------------------------------------
Sedna XML:DB API Client started, Version 1.2.4 08/Feb/11
Copyright (C) 2007 Charles Foster, www.cfoster.net.
-------------------------------------------------------
Connexió establerta amb SGBD sedna
Col·lecció actual: xmldb
Col·lecció pare:
Nombre de fills del pare: 1
Noms dels fills:
        xmldb
Serveis proporcionats:
        XPathQueryService - Versió: 1.0
        XUpdateQueryService - Versió: 1.0
        TransactionService - Versió: 1.0
        CollectionManagementService - Versió: 1.0
```

```
        SednaUpdateService - Versió: 1.0
        XQueryService - Versió: 1.0
        IndexManagementService - Versió: 1.0
        UserManagementService - Versió: 1.0
        ModuleManagementService - Versió: 1.0
        RoleManagementService - Versió: 1.0
Intentarem crear 5 col·leccions filles!
Nombre de fills: 5
Noms dels fills:
        aux0
        aux1
        aux2
        aux3
        aux4
Després d'haver creat les col·leccions, estem situats a la col·lecció: xmldb
Eliminem les 5 col·leccions creades!
Nre. de fills: 0
Connexió tancada
```

Si no haguéssim eliminat les col·leccions creades, podríem comprovar, via la interfície gràfica de *Sedna,* que les col·leccions creades són, realment: `xmldb/aux0`, `xmldb/aux1`, `xmldb/aux2`, `xmldb/aux3` i `xmldb/aux4`.

2.2.3. Afegir, recuperar i eliminar recursos

L'API XML:DB defineix la possibilitat de gestionar dos tipus de recursos: `XMLResource` (per a dades XML emmagatzemades a la base de dades) i `BinaryResource` (per a dades binàries encapsulades i emmagatzemades a la base de dades). Per a cada tipus de recurs hi ha una interfície d'igual nom.

L'API XML:DB incorpora, en el nivell de conformitat zero, l'obligatorietat d'implementar el mòdul `XMLResources` que implica proveir accés a les dades XML, ja sigui com a text XML o com un node `DOM` (W3C) o via un `ContentHandler SAX`. En canvi, la implementació del mòdul `BinaryResource` és optativa i, en conseqüència, no totes les API XML:DB la incorporen.

> Per a desenvolupar programes Java que gestionin documents XML és necessari conèixer les API DOM, SAX i StAX facilitades per Java. També és recomanable conèixer l'API JDOM (projecte jdom.org) alternativa a les API facilitades per Java.

Per poder gestionar recursos (siguin XML o binaris), la interfície `Collection` ens facilita els següents mètodes:

- `getResourceCount()`, per conèixer el nombre de recursos existents a la col·lecció actual.
- `listResources()`, per recuperar la llista d'identificadors (`String`) dels recursos continguts a la col·lecció actual; cada recurs emmagatzemat en una BD-XML

té assignat un identificador.
- `getResource(id)`, per recuperar l'objecte `Resource` corresponent a l'identificador indicat per paràmetre o `null` en cas de no existir.
- `createId()`, per obtenir un nou identificador únic dins la col·lecció actual.
- `createResource(id, tipus)`, per obtenir un objecte `Resource` buit, amb identificador i tipus indicats. Si `id` és `null` o buit, automàticament crida el mètode `createId()` per obtenir un identificador. El tipus de recurs ha de ser qualsevol dels tipus proveïts pel SGBD; XML:DB defineix els tipus `XMLResource` i `BinaryResource` com a tipus de recurs vàlids. El recurs així creat no s'emmagatzema a la base de dades fins que s'invoca el mètode `storeResource()`.
- `storeResource(res)`, per emmagatzemar a la base de dades un objecte `Resource` indicat per paràmetre.
- `removeResource(res)`, per eliminar de la base de dades un objecte Resource indicat per paràmetre.

Els mètodes anteriors ens permeten accedir als recursos, crear-ne i emmagatzemar-los a la BD i eliminar-los de la BD. No ens permeten accedir a la seva gestió, la qual dependrà de si es tracta d'un recurs XML o d'un recurs binari i, per aquesta gestió, disposem de les interfícies `XMLResource` i `BinaryResource.`

Les dues interfícies `XMLResource` i `BinaryResource` implementen la interfície `Resource` i d'ella hereten els mètodes:

- `getId()`, per obtenir l'identificador del recurs.
- `getContent()`, per recuperar el contingut del recurs com a `Object.`
- `getParentCollection()`, per obtenir la col·lecció a la que pertany el recurs.
- `getResourceType()`, per recuperar el tipus del recurs (binari o XML).
- `setContent(valor)`, per omplir de contingut el recurs.

La interfície `BinaryResource` no facilita cap mètode més, doncs no és necessari. En el cas de recursos binaris només cal poder assignar i recuperar el contingut del recurs i això ja ho faciliten els mètodes heretats `setContent()` i `getContent()`.

La interfície `XMLResource` sí facilita més mètodes, doncs en els recursos XML necessitem més funcionalitats, en haver de proveir accés com a text, com a DOM i com a SAX. Hi trobem, doncs, els següents mètodes:

- `getContentAsDOM()`, per recuperar el contingut del recurs com un node DOM.
- `getContentAsSAX(manegador)`, per poder gestionar el contingut del recurs amb l'objecte `ContentHandler` indicat per paràmetre.
- `setContentAsDOM(node)`, per establir com a contingut del recurs el node DOM indicar per paràmetre.
- `setContentAsSAX()`, per generar l'objecte `ContentHandler` que caldrà associar al document XML a enregistrar abans de procedir al seu anàlisi.

Per gestionar el contingut del recurs XML com a text utilitzarem directament els mètodes heretats `setContent()` i `getContent()`.

Exemple de programa que recupera informació XML via DOM i SAX

Es desitja enumerar els noms de les autonomies de l'estat espanyol, fet que podem assolir, a partir del fitxer `mondial.xml`, situant-nos en el node `country` corresponent a Espanya (node que tingui `"E"` com a valor de

l'atribut `car_code`) i una vegada allí, cercar tots els nodes `province` per mostrar-ne el seu `name`. Aquesta recerca tant la podem efectuar de tres maneres:

- Bolcant el contingut del document en una cadena (mètode `getContent()`) i posteriorment analitzant la cadena... Molt complicat...

- Obtenint el contingut del document com un objecte DOM (mètode `getContentAsDOM()`) i posteriorment analitzant aquest document. Recordem que la gestió de documents XML via objectes DOM pot provocar problemes en documents de gran volum doncs DOM carrega tot el document en memòria.

- Analitzant el contingut del document amb un analitzador SAX.

El programa següent mostra com aplicar la via DOM i la via SAX.

> Trobareu el fitxer `XMLDB04.java` dins el paquet de codi font a l'apartat "Material per desenvolupament sobre SGBD- XML natives utilitzant l'API XMLDB" dels annexos del web.

```
/*
 * Programa: XMLDB04.java
 * Objectiu: Exemple de gestió de recursos:
 *           >> Obtenció de llista de recursos
 *           >> Cerca del recurs XML: mondial.xml
 *           >> Recuperació del recurs com a SAX i cerca d'autonomies
 *           >> Recuperació del recurs com a DOM i cerca d'autonomies
 * Autor...: Isidre Guixà
 */

package proves;
import org.xmldb.api.base.*;
import org.xmldb.api.DatabaseManager;
import org.xmldb.api.modules.*;
// Classes necessàries per la gestió DOM
import org.w3c.dom.Node;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import org.w3c.dom.NamedNodeMap;
// Classes necessàries per la gestió SAX
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.SAXParseException;

public class XMLDB04
{
  // Amaguem el constructor per defecte. */
  private XMLDB04() { }

  private static Collection establirConnexio(String[] args)
                  throws Exception
  /* Intenta establir una connexió contra el SGBD indicat per paràmetre, que
     ha de ser: eXist-db o BaseX o Sedna */
```

```
{ /* Mateix contingut que programa XMLDB02.java */ }

public static void main(String[] args){
  Collection coll = null;
  try {
    coll = establirConnexio(args);
    System.out.println("Connexió establerta amb SGBD " + args[0]);
    // Recuperem el nom de la col·lecció actual
    System.out.println("Col·lecció actual: "+coll.getName());
    // Recuperem nombre de recursos dins la col·lecció
    int recursos = coll.getResourceCount();
    System.out.println("Nre. de recursos: "+recursos);
    if (recursos>0) {
      String idRecursos[] = coll.listResources();
      for (int i=0; i<recursos; i++)
        System.out.println("\t"+idRecursos[i]);
    }
    // Cerquem el recurs anomenat "mondial.xml"
    Resource r = coll.getResource("mondial.xml");
    if (r == null)
      System.out.println("No existeix el recurs 'mondial.xml'");
    else {
      System.out.println("Recurs 'mondial.xml' trobat.");
      System.out.println("Tipus de recurs: "+r.getResourceType());
      if (r instanceof XMLResource) {
        XMLResource xml = (XMLResource)r;
        // Cerquem les autonomies d'Espanya via SAX:
        xml.getContentAsSAX (new llistatAutonomiesSAX());
        // Cerquem les autonomies d'Espanya via DOM:
        llistaAutonomiesDOM (xml.getContentAsDOM());
      }
    }
  }
  catch (Exception e) {
    e.printStackTrace();
  }
  finally { /* Tanquem connexió en qualsevol cas */
    try {
        if(coll != null) {
          coll.close();
          System.out.println("Connexió tancada");
        }
    }
    catch(XMLDBException xe) {
        xe.printStackTrace();
    }
  }
}

private static void llistaAutonomiesDOM (Node contingut) {
  Node mondial = contingut.getFirstChild(); // Node <mondial>
  Node nCountry = mondial.getFirstChild();  // Nodes <country>
  boolean trobat = false;
  while (nCountry!=null && !trobat) {
    if (nCountry.getNodeType()==Node.ELEMENT_NODE) {
      NamedNodeMap attr = nCountry.getAttributes();
      Node nodeAttr = attr.getNamedItem("car_code");
      if (nodeAttr!=null && "E".equals(nodeAttr.getNodeValue()))
        { trobat=true; break; }
    }
    nCountry=nCountry.getNextSibling();
```

```
    }
    if (!trobat) {
      System.out.println ("No es troba el país Espanya");
      return;
    }
    // nCountry correspon a "Espanya"
    // Recorrem els fills de nCountry que tinguin per etiqueta "province"
    System.out.println("Llistat d'autonomies via DOM:");
    NodeList nProvince = ((Element)nCountry).getElementsByTagName("province");
    int q = nProvince.getLength();
    for (int i=0; i<q; i++) {
      NodeList nNoms=((Element)nProvince.item(i)).getElementsByTagName("name");
      System.out.println("\t" + (i+1) + ": " +
                         nNoms.item(0).getChildNodes().item(0).getNodeValue());
    }
  }

  private static class llistatAutonomiesSAX extends DefaultHandler {
    boolean countryEspanya = false;
    boolean autonomia = false;
    boolean nom = false;
    int nivell = -1;
    int i = 0;

    public void startDocument () {
      System.out.println("Llistat d'autonomies via SAX:");
    }

    public void startElement(String uri, String localName, String qName,
                             Attributes attributes) throws SAXException {
      if (qName.equalsIgnoreCase("country")) {
        for (int i=0; i<attributes.getLength(); i++) {
          if (attributes.getQName(i).equals("car_code") &&
              attributes.getValue(i).equals("E")) {
            countryEspanya=true;
            break;
          }
        }
      }
      if (countryEspanya && qName.equalsIgnoreCase("province"))
        autonomia = true;
      if (autonomia) nivell++;
      if (autonomia && qName.equalsIgnoreCase("name") && nivell==1)
        nom = true;
    }

    public void endElement(String uri, String localName, String qName)
                throws SAXException {
      if (autonomia) nivell--;
      if (autonomia && qName.equalsIgnoreCase("province")) autonomia = false;
      if (countryEspanya && qName.equalsIgnoreCase("country"))
        countryEspanya=false;
//  En realitat, quan posem countryEspanya=false és per què ja hem processat
//  el país que ens interessava cercar... No caldria continuar amb l'anàlisi
//  del fitxer XML... Per avortar el procés, caldria llençar una SAXException!
    }

    public void characters(char ch[], int start, int length)
                throws SAXException {
      if (nom) {
        System.out.println("\t"+(++i)+": "+new String(ch, start, length));
```

```
            nom = false;
        }
      }
    }
}
```

Comprovem l'execució en qualsevol dels tres SGBD:

```
G:\>java -Dfile.encoding=cp850 proves.XMLDB04 nomSGBD
Connexió establerta amb SGBD ...
Col·lecció actual: ...(depèn del SGBD: xmldb o db/xmldb)
Nre. de recursos: 1
        mondial.xml
Recurs 'mondial.xml' trobat.
Tipus de recurs: XMLResource
Llistat d'autonomies via SAX:
        1: Andalusia
        2: Aragon
        3: Asturias
        4: Balearic Islands
        5: Basque Country
        6: Canary Islands
        7: Cantabria
        8: Castile and Leon
        9: Castile La Mancha
        10: Catalonia
        11: Estremadura
        12: Galicia
        13: Madrid
        14: Murcia
        15: Navarre
        16: Rioja
        17: Valencia
Llistat d'autonomies via DOM:
        1: Andalusia
        2: Aragon
        3: Asturias
        4: Balearic Islands
        5: Basque Country
        6: Canary Islands
        7: Cantabria
        8: Castile and Leon
        9: Castile La Mancha
        10: Catalonia
        11: Estremadura
        12: Galicia
        13: Madrid
        14: Murcia
        15: Navarre
        16: Rioja
        17: Valencia
Connexió tancada
```

Exemple de programa que enregistra informació XML via text, DOM i SAX

Es desitja enregistrar a la BD el següent document XML:

```
<arrel>Hola Món!</arrel>
```

El programa següent mostra tres formes de fer-ho: enviant un objecte `String`, enviant un node DOM o via un analitzador SAX.

Trobareu el fitxer

`XMLDB05.java` dins el paquet de codi font a l'apartat "Material per desenvolupament sobre SGBD- XML natives utilitzant l'API XMLDB" dels annexos del web.

```
/*
 * Programa: XMLDB05.java
 * Objectiu: Enregistrament de document XML
 *           via String, DOM i SAX.
 * Autor...: Isidre Guixà
 */

package proves;
import org.xmldb.api.base.*;
import org.xmldb.api.DatabaseManager;
import org.xmldb.api.modules.*;
// Classes necessàries per la gestió DOM
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
// Classes necessàries per la gestió SAX
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.XMLReaderFactory;
import org.xml.sax.InputSource;
import org.xml.sax.ContentHandler;
import javax.xml.stream.XMLOutputFactory;
import javax.xml.stream.XMLStreamWriter;
import java.io.StringReader;
import java.io.StringWriter;

public class XMLDB05
{
  // Amaguem el constructor per defecte. */
  private XMLDB05() { }

  private static Collection establirConnexio(String[] args)
                 throws Exception
  /* Intenta establir una connexió contra el SGBD indicat per paràmetre, que
     ha de ser: eXist-db o BaseX o Sedna */
  { /* Mateix contingut que programa XMLDB02.java */ }

  public static void main(String[] args){
    Collection coll = null;
    try {
      coll = establirConnexio(args);
      System.out.println("Connexió establerta amb SGBD " + args[0]);
      // Recuperem el nom de la col·lecció actual
      System.out.println("Col·lecció actual: "+coll.getName());
      XMLResource recurs;

      // Enregistrem document XML a partir d'objecte String
      recurs = (XMLResource)coll.createResource("HolaMon_String",
                                                XMLResource.RESOURCE_TYPE);
      recurs.setContent("<arrel>Hola Món!</arrel>");
      coll.storeResource(recurs);
```

```
      // Enregistrem document XML a partir d'objecte SAX
      StringWriter out = new StringWriter();
      XMLOutputFactory xof = XMLOutputFactory.newInstance();
      XMLStreamWriter xtw;
      xtw = xof.createXMLStreamWriter(out);
      xtw.writeStartDocument();
      xtw.writeStartElement("arrel");
      xtw.writeCharacters("Hola Món!");
      xtw.writeEndElement();
      xtw.writeEndDocument();
      xtw.flush();
      xtw.close();
      recurs = (XMLResource)coll.createResource("HolaMon_SAX",
                                                XMLResource.RESOURCE_TYPE);
      ContentHandler ch = recurs.setContentAsSAX();
      XMLReader reader = XMLReaderFactory.createXMLReader();
      reader.setContentHandler(ch);
      reader.parse(new InputSource(new StringReader(out.toString())));
      coll.storeResource(recurs);

      // Enregistrem document XML a partir d'objecte DOM
      recurs = (XMLResource)coll.createResource("HolaMon_DOM",
                                                XMLResource.RESOURCE_TYPE);
      DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();
      DocumentBuilder docBuilder = docFactory.newDocumentBuilder();
      Document doc = docBuilder.newDocument();
      Element rootElement = doc.createElement("arrel");
      doc.appendChild(rootElement);
      rootElement.appendChild(doc.createTextNode("Hola Món!"));
      recurs.setContentAsDOM(rootElement);
      coll.storeResource(recurs);
    }
    catch (Exception e) {
      e.printStackTrace();
    }
    finally { /* Tanquem connexió en qualsevol cas */
      try {
          if(coll != null) {
            coll.close();
            System.out.println("Connexió tancada");
          }
      }
      catch(XMLDBException xe) {
          xe.printStackTrace();
      }
    }
  }
}
```

Comprovem l'execució en qualsevol dels tres SGBD:

```
G:\>java -Dfile.encoding=cp850 proves.XMLDB05 nomSGBD
Connexió establerta amb SGBD ...
Connexió tancada
```

L'execució en *BaseX* (versió 7.1) finalitza amb una excepció en intentar enregistrar el node DOM. Sembla que és un error del SGBD.

Podem comprovar la creació dels fitxers en els diversos SGBD, amb la utilització de les corresponents interfícies gràfiques (prèvia reinicialització).

Respecte l'enregistrament com a SAX, comentar que XML:DB facilita el mètode `setContentAsSAX()` sobre el recurs, per obtenir un objecte `ContentHandler` (`ch` a l'exemple) que s'ha d'associar a un document `XMLReader` amb el què analitzarem el document a enregistrar a la BD, de

manera que el seu anàlisi omple el recurs sobre el que s'havia obtingut l'objecte `ch`. En motes ocasions, el document XML residirà a disc i, per tant, la posta en marxa de l'anàlisi serà amb una instrucció similar a:

```
reader.parse(new InputSource(fitxerPerEnregistrar));
```

El nostre exemple és atípic, doncs no hem d'enregistrar un fitxer XML resident a disc, sinó que hm de procedir a crear el fitxer per enregistrar-lo i, evidentment, no volem passar per disc. Per això la resolució del problema és una mica embolicada, ja que:

- Per una banda, hem creat un document XML en memòria amb la classe `XMLStreamWriter` (Java 6) enviant la sortida a un objecte `StringWriter` (`out` en el nostre exemple).

- Per altra banda, a l'hora d'efectuar l'anàlisi del document amb l'objecte `XMLReader`, hem de passar-li el document, que l'hem enviat a un objecte `StringWriter` (`out`) però que només es pot llegir d'un objecte `InputSource` i, per aquest motiu fem:

```
reader.parse(new InputSource(new StringReader(out.toString())));
```

En aquesta darrera instrucció observem com passem d'un objecte `Writer` (`out`) a un objecte `Reader` utilitzant una zona de memòria intermèdia *(buffer)*.
El problema de passar d'un objecte Writer a un objecte Reader amb memòria intermèdia és que cal tenir suficient memòria per encabir la quantitat total de dades. En el nostre exemple no hi havia problema, doncs es tracta d'uns pocs bytes, però en cas d'arxius grans, el mecanisme emprat no seria adequat i la solució seria crear un fil (`thread`) per traspassar dades d'un `PipedWriter` a un `PipedReader`.

Exemple de programa que elimina recursos

Es vol eliminar els documents creats en el programa anterior (tots els documents que el seu nom comenci per `HolaMon`):

> Trobareu el fitxer `XMLDB06.java` dins el paquet de codi font a l'apartat "Material per desenvolupament sobre SGBD- XML natives utilitzant l'API XMLDB" dels annexos del web.

```
/*
 * Programa: XMLDB06.java
 * Objectiu: Exemple d'eliminació de recursos: eliminem tots els recursos
 *           XML que el seu nom comenci amb "HolaMon"
 * Autor...: Isidre Guixà
 */

package proves;
import org.xmldb.api.base.*;
import org.xmldb.api.DatabaseManager;
import org.xmldb.api.modules.*;

public class XMLDB06
{
  // Amaguem el constructor per defecte. */
```

```
  private XMLDB06() { }

  private static Collection establirConnexio(String[] args)
                throws Exception
  /* Intenta establir una connexió contra el SGBD indicat per paràmetre, que
     ha de ser: eXist-db o BaseX o Sedna */
  { /* Mateix contingut que programa XMLDB02.java */ }

  public static void main(String[] args){
    Collection coll = null;
    try {
      coll = establirConnexio(args);
      System.out.println("Connexió establerta amb SGBD " + args[0]);
      // Recuperem el nom de la col·lecció actual
      System.out.println("Col·lecció actual: "+coll.getName());
      // Recuperem nombre de recursos dins la col·lecció
      int recursos = coll.getResourceCount();
      System.out.println("Nre. de recursos: "+recursos);
      if (recursos>0) {
        String idRecursos[] = coll.listResources();
        for (int i=0; i<recursos; i++) {
          System.out.print("\t"+idRecursos[i]);
          if (idRecursos[i].startsWith("HolaMon")) {
            Resource r = coll.getResource(idRecursos[i]);
            coll.removeResource(r);
            System.out.println(" >> Eliminat!");
          } else System.out.println();
        }
      }
    }
    catch (Exception e) {
      e.printStackTrace();
    }
    finally { /* Tanquem connexió en qualsevol cas */
      try {
          if(coll != null) {
            coll.close();
            System.out.println("Connexió tancada");
          }
      }
      catch(XMLDBException xe) {
          xe.printStackTrace();
      }
    }
  }
}
```

Comprovem l'execució en qualsevol dels tres SGBD:

```
G:\>java -Dfile.encoding=cp850 proves.XMLDB06 nomSGBD
Connexió establerta amb SGBD ...
Col·lecció actual:...(depèn del SGBD: xmldb o db/xmldb)
Nre. de recursos: 4
        mondial.xml
        HolaMon_DOM >> Eliminat!
        HolaMon_String >> Eliminat!
        HolaMon_SAX >> Eliminat!
Connexió tancada
```

Podem comprovar l'eliminació dels fitxers en els diversos SGBD, amb la utilització de les corresponents interfícies gràfiques (prèvia reinicialització).

En els exemples anteriors hem après a afegir, recuperar i eliminar recursos XML d'una

BD-XML amb l'API XML:DB. A més, en el cas d'afegir i recuperar, hem anat més lluny del propi fet d'afegir o recuperar, doncs hem fet una gestió pròpia de DOM o de SAX, tant per afegir un recurs com per trobar una determinada informació.

L'API XML:DB també permet afegir, recuperar i eliminar recursos binaris. No hi ha cap dificultat per executar aquestes accions una vegada sabem afegir, recuperar i eliminar recursos XML.

Ens manca indicar com actualitzar recursos d'una BD-XML. La manera més senzilla és:

4. Recuperar el document del recurs (ja sigui XML o binari) amb el mètode que correspongui (`getContent()`, `getContentAsDOM()`, `getContentAsSAX()`)
5. Obtenir el document modificat, ja sigui substituint-lo tot o modificant-lo amb els mètodes que correspongui.
6. Assignar el document modificat o nou en el recurs, amb el mètode que correspongui (`setContent()`, `setContentAsDOM()`, `setContentAsSAX()`)
7. Enregistrar el recurs amb el mètode `storeResource`, que substitueix el recurs amb idèntic identificador existent dins la col·lecció activa.

2.2.4. Consulta i actualització en documents XML

Les interfícies `Collection`, `Resource` i `XMLResource` de l'API XML:DB faciliten la funcionalitat per gestionar recursos XML a nivell de document, doncs permeten:

- Recuperar un document, ja sigui per enregistrar-lo a disc o, si es tracta de document XML, per efectuar-hi alguna consulta; en aquest cas, si es fa via DOM, hem de tenir tot el document carregat en memòria, mentre que si es fa via SAX, l'anàlisi es pot aturar una vegada la consulta ha estat efectuada.
- Afegir un document.
- Eliminar un document.
- Modificar un document.

Aquestes accions són important però no són suficients. Interessa, en el cas de documents XML, efectuar consultes i modificacions sobre els documents emmagatzemats a la BD-XML, sense haver de recuperar tot el document.

Consultes en documents XML

L'API XML:DB defineix una interfície destinada a l'execució de consultes en documents XML emmagatzemats a la BD. Es tracta de la interfície `XPathQueryService`, que activa l'execució d'instruccions *XPath/XQuery* dins una col·lecció o dins un recurs XML emmagatzemat a la BD.

Com el seu nom indica, aquesta funcionalitat es facilita com un servei i és interessant saber quins serveis facilita el SGBD-XML natives amb el què vulguem treballar. El mètode `Collection.getServices()` permet recuperar la llista de serveis facilitats pel SGBD. La taula 2-4 mostra els serveis facilitats pels SGBD amb els què estem

practicant i observem que els tres SGBD implementen el servei `XPathQueryService`.

Taula 2-4. Llista de serveis de l'API XML:DB proporcionats per diversos SGBD

SGBD-XML natives	Serveis implementats de l'API XML:DB
eXist-db (versió 1.4.1)	XPathQueryService - Versió: 1.0 CollectionManagementService - Versió: 1.0 UserManagementService - Versió: 1.0 DatabaseInstanceManager - Versió: 1.0 IndexQueryService - Versió: 1.0 XUpdateQueryService - Versió: 1.0 ValidationService - Versió: 1.0
BaseX (versió 7.1)	XPathQueryService - Versió: 1.0 XQueryQueryService - Versió: 1.0 CollectionManagementService - Versió: 1.0
Sedna (versió 3.5)	XPathQueryService - Versió: 1.0 XUpdateQueryService - Versió: 1.0 TransactionService - Versió: 1.0 CollectionManagementService - Versió: 1.0 SednaUpdateService - Versió: 1.0 XQueryService - Versió: 1.0 IndexManagementService - Versió: 1.0 UserManagementService - Versió: 1.0 ModuleManagementService - Versió: 1.0 RoleManagementService - Versió: 1.0

Per poder utilitzar el servei, en primer lloc l'haurem d'aconseguir amb el mètode `Collection.getService()`, al que haurem d'indicar el nom del servei i la seva versió.

Per iniciar-nos en el coneixement de la interfície `XPathQueryService`, ens interessa, principalment, conèixer els dos mètodes següents:

- `query (consulta)`, per executar una consulta *XPath*/*XQuery* contra la col·lecció activa.
- `queryResource (recurs, consulta)`, per executar una consulta *XPath*/*XQuery* contra un recurs de la col·lecció activa.

Ambdós mètodes retornen un objecte `ResourceSet` que conté els resultats de la consulta. La interfície `ResourceSet` correspon a un contenidor de recursos i facilita diferents mètodes per a gestionar-ne els continguts:

- `addAll (contenidor)`, per afegir totes les instàncies del contenidor passat per paràmetre, en el contenidor sobre el que s'executa el mètode.
- `addResource (recurs)`, per afegir el recurs indicat per parèmetre, en el contenidor.
- `clear()`, per buidar el contenidor.
- `getIterador()`, per obtenir un objecte `ResourceIterador` que permet efectuar un recorregut sobre els recursos emmagatzemats en el contenidor.
- `getMembersAsResource()`, per obtenir un recurs contenint una representació XML de tots els recursos emmagatzemats en el contenidor.
- `getResource (índex)`, per obtenir el recurs emmagatzemat a la posició indicada per l'índex.

- `getSize ()`, per obtenir el nombre de recursos emmagatzemats en el contenidor.
- `removeResource(índex)`, per eliminar el recurs emmagatzemat a la posició indicada per l'índex.

La interfície `ResourceIterator` facilita els dos mètodes típics de les interfícies `Iterator`:

- `hasMoreResources()`, per saber si hi ha més recursos per a ser accedits.
- `nextResource()`, per passar al següent recurs.

Donat que la interfície facilita els mètodes `getSize()` i `getResource(índex)`, sembla innecessària la utilització dels mètodes proporcionats per la interfície `ResourceIterator`. Res més lluny de la realitat, doncs la utilització de `getSize()`, segons documenta XML:DB, pot provocar la descàrrega, des de la BD, de tots els recursos per esbrinar-ne el nombre, i això és de totes llums ineficient si el què ens interessa és fer un recorregut fins a trobar un determinat recurs.

Exemple d'utilització del servei `XpathQueryService` amb *XPath*

Es desitja enumerar els noms de les autonomies de l'estat espanyol:

> Trobareu el fitxer `XMLDB07.java` dins el paquet de codi font a l'apartat "Material per desenvolupament sobre SGBD- XML natives utilitzant l'API XMLDB" dels annexos del web.

```
/*
 * Programa: XMLDB07.java
 * Objectiu: Exemple d'utilització del servei XPathQueryService (XML:DB)
 *           per executar una consulta Xpath
 * Autor...: Isidre Guixà
 */

package proves;
import org.xmldb.api.base.*;
import org.xmldb.api.DatabaseManager;
import org.xmldb.api.modules.*;

public class XMLDB07
{
  // Amaguem el constructor per defecte. */
  private XMLDB07() { }

  private static Collection establirConnexio(String[] args)
                 throws Exception
  /* Intenta establir una connexió contra el SGBD indicat per paràmetre, que
     ha de ser: eXist-db o BaseX o Sedna */
  { /* Mateix contingut que programa XMLDB02.java */ }

  public static void main(String[] args){
    String servei = "XPathQueryService";
    String versio = "1.0";
    Collection coll = null;
    try {
      coll = establirConnexio(args);
```

```
      System.out.println("Connexió establerta amb SGBD " + args[0]);
      // Recuperem el nom de la col·lecció actual
      System.out.println("Col·lecció actual: "+coll.getName());
      // Recuperem nombre de recursos dins la col·lecció
      XPathQueryService xpqs =
        (XPathQueryService) coll.getService(servei,versio);
      if (xpqs == null)
        System.out.println("El SGBD no té el servei "+ servei +
                           " Versió : " + versio);
      else {
        // Executem la instrucció XPath sobre tota la col·lecció:
        ResourceSet rs =
          xpqs.query("/mondial/country[@car_code='E']/province/name/text()");
        ResourceIterator ri = rs.getIterator();
        if (!ri.hasMoreResources())
          System.out.println ("No s'ha trobat cap autonomia.");
        else {
          System.out.println ("Noms de les autonomies de l'estat espanyol:");
          Resource r;
          int i=0;
          while (ri.hasMoreResources()) {
            r = ri.nextResource();
            System.out.println((++i)+": "+(String)r.getContent());
          }
        }
      }
    }
    catch (Exception e) {
      e.printStackTrace();
    }
    finally { /* Tanquem connexió en qualsevol cas */
      try {
          if(coll != null) {
            coll.close();
            System.out.println("Connexió tancada");
          }
      }
      catch(XMLDBException xe) {
          xe.printStackTrace();
      }
    }
  }
}
```

Comprovem l'execució en qualsevol dels tres SGBD:

```
G:\>java -Dfile.encoding=cp850 proves.XMLDB07 nomSGBD
Connexió establerta amb SGBD ...
Col·lecció actual: ...(depèn del SGBD: xmldb o db/xmldb)
Noms de les autonomies de l'estat espanyol:
1: Andalusia
2: Aragon
3: Asturias
4: Balearic Islands
5: Basque Country
6: Canary Islands
7: Cantabria
8: Castile and Leon
9: Castile La Mancha
10: Catalonia
11: Estremadura
12: Galicia
13: Madrid
```

```
14: Murcia
15: Navarre
16: Rioja
17: Valencia
Connexió tancada
```

La consulta s'ha efectuat sobre tots els documents XML de la col·lecció activa, de manera que si hi hagués hagut dos documents amb les autonomies, la llista hagués sortit duplicada. Si es vol restringir la cerca a un document en concret, cal substituir:

```
ResourceSet rs =
  xpqs.query("/mondial/country[@car_code='E']/province/name/text()");
```

per:

```
ResourceSet rs =
  xpqs.queryResource("mondial.xml",
                     "/mondial/country[@car_code='E']/province/name/text()");
```

Exemple d'utilització del servei `XPathQueryService` amb *XQuery*

El següent programa mostra el rànquing dels països segons la mortalitat infantil (nombre de nens menors d'un any morts per cada mil naixements), en intervals d'amplada vint ([0,20[, [20,40[, [40,60[,...)
Per aconseguir el resultat desitjat, el programa prepara una cadena que conté una sentència *XQuery* que cerca els països amb mortalitat infantil en un determinat interval i executa aquesta sentència tantes vegades com intervals d'amplada 20 hi ha entre 0 i 1000, és a dir, 50 vegades.

> Trobareu el fitxer `XMLDB08.java` dins el paquet de codi font a l'apartat "Material per desenvolupament sobre SGBD- XML natives utilitzant l'API XML:DB" dels annexos del web.

```
/*
 * Programa: XMLDB08.java
 * Objectiu: Exemple d'utilització del servei XPathQueryService (XML:DB)
 *           per executar una consulta XQuery
 * Autor...: Isidre Guixà
 */

package proves;
import org.xmldb.api.base.*;
import org.xmldb.api.DatabaseManager;
import org.xmldb.api.modules.*;

public class XMLDB08
{
  // Amaguem el constructor per defecte. */
  private XMLDB08() { }

  private static Collection establirConnexio(String[] args)
                  throws Exception
  /* Intenta establir una connexió contra el SGBD indicat per paràmetre, que
     ha de ser: eXist-db o BaseX o Sedna */
  { /* Mateix contingut que programa XMLDB02.java */ }

  public static void main(String[] args){
    String servei = "XPathQueryService";
```

```
    String versio = "1.0";
    Collection coll = null;
    try {
      coll = establirConnexio(args);
      System.out.println("Connexió establerta amb SGBD " + args[0]);
      // Recuperem el nom de la col·lecció actual
      System.out.println("Col·lecció actual: "+coll.getName());
      // Recuperem nombre de recursos dins la col·lecció
      XPathQueryService xpqs =
        (XPathQueryService) coll.getService(servei,versio);
      if (xpqs == null)
        System.out.println("El SGBD no té el servei "+ servei +
                           " Versió : " + versio);
      else {
        String cad;
        for (double x=20.; x<=1000; x=x+20.) {
          // Construim la sentència XQuery per l'interval [x-20, x[:
          cad = "for $i in /mondial/"+
              "country[infant_mortality >= " + (x-20) +
              " and infant_mortality < " + x + "] "+
              "order by number($i/infant_mortality) "+
              "return concat($i/infant_mortality,' - ',$i/name)";
          // Executem la sentència Xquery
          ResourceSet rs = xpqs.query (cad);
          ResourceIterator ri = rs.getIterator();
          if (ri.hasMoreResources()) {
            System.out.println("Interval [" + (x-20) + "," + x + "[:");
            Resource r;
            while (ri.hasMoreResources()) {
              r = ri.nextResource();
              System.out.println("\t"+(String)r.getContent());
            }
          }
        }
      }
    }
    catch (Exception e) {
      e.printStackTrace();
    }
    finally { /* Tanquem connexió en qualsevol cas */
      try {
          if(coll != null) {
            coll.close();
            System.out.println("Connexió tancada");
          }
      }
      catch(XMLDBException xe) {
          xe.printStackTrace();
      }
    }
  }
}
```

Comprovem el funcionament d'aquest programa, executant-lo per cadascun dels tres SGBD. L'execució en qualsevol cas és:

```
G:\>java -Dfile.encoding=cp850 proves.XMLDB08 nomSGBD
Connexió establerta amb SGBD ...
Col·lecció actual: ...(depèn del SGBD: xmldb o db/xmldb)
Interval [0.0,20.0[:
        2.2 - Andorra
        2.4 - Man
        ...
```

```
        6.3 - Spain
        ...
        19.6 - Qatar
Interval [20.0,40.0[:
        20 - Saint Lucia
        ...
...
Interval [140.0,160.0[:
        145.82 - Western Sahara
        149.7 - Afghanistan
```

L'anterior exemple ens presenta un cas de consulta repetitiva, és a dir, de consulta que té una determinada estructura i que cal executar-la repetides vegades canviant algun valor intern (en el nostre exemple, els límits de l'interval de mortalitat infantil). Els programadors acostumats a treballar amb consultes repetitives en altres llenguatges de consulta sobre SGBD (SQL o XQJ) haurien dissenyat aquest problema utilitzant una consulta preparada o compilada.

Una consulta preparada o compilada és una sentència escrita com una plantilla que conté algunes variables que s'aniran substituint a cada execució. El SGBD analitza la sentència una única vegada, obtenint el pla d'execució, el qual utilitza en les repetides execucions canviant, únicament, el valor de les variables existents a la consulta.

L'API XML:DB no defineix la possibilitat de consultes preparades o compilades. Però alguns SGBD han evolucionat l'API XML:DB proporcionant aquesta funcionalitat. Així, ens trobem que *Sedna* i *eXist-db* proporcionen la interfície *XQueryService* que facilita poder treballar am consultes compilades.

> El SGBD *eXist-db* facilita la interfície `XQueryService`, lligada al servei d'igual nom, i en canvi, l'execució del mètode `getServices()` no informa d'aquest servei.

El mecanisme de funcionament és molt simple i es basa en utilitzar convenientment els mètodes:

- `declareVariable (nom, valor)`, que permet declarar una variable global, externa a la consulta *XQuery*.
- `compile (consulta)`, que permet compilar una consulta *XQuery* que pot contenir variables i facilita un objecte `CompiledExpression`.
- `execute (expressió compilada)`, que executa l'expressió compilada prèvia substitució dels valors de les variables incloses a la consulta pels valors actuals definits pel mètode `declareVariable()`.

El servei `XQueryService` només funciona correctament a *eXist-db*. En *Sedna* (versió 3.5), per obtenir un bon funcionament, cal tornar a compilar la consulta cada vegada que es modifica el valor de la variable amb el mètode `declareVariable()`, fet que anul·la l'interès d'aquest servei (una única compilació per a múltiples execucions).

Exemple d'utilització del servei `XQueryService` a *eXist-db*

El següent programa mostra el rànquing dels països segons la mortalitat infantil (nombre de nens menors d'un any morts per cada mil naixements),

en intervals d'amplada vint ([0,20[, [20,40[, [40,60[,...)
Per aconseguir el resultat desitjat, el programa prepara una consulta *XQuery* compilada que cerca els països amb mortalitat infantil en un determinat interval i executa aquesta consulta compilada tantes vegades com intervals d'amplada 20 hi ha entre 0 i 1000, és a dir, 50 vegades, definint a cada execució l'interval sobre la que s'ha d'executar la sentència.

> Trobareu el fitxer `XMLDB09.java` dins el paquet de codi font a l'apartat "Material per desenvolupament sobre SGBD- XML natives utilitzant l'API XML:DB" dels annexos del web.

```
/*
 * Programa: XMLDB09.java
 * Objectiu: Exemple d'utilització del servei XQueryService (eXist-db)
 *           En Sedna es compila però l'execució no és l'esperada...
 *           i per solucionar-ho caldria compilació prèvia a cada execució
 * Autor...: Isidre Guixà
 */

package proves;
import org.xmldb.api.base.*;
import org.xmldb.api.DatabaseManager;
import org.xmldb.api.modules.*;
//import org.exist.xmldb.*;

public class XMLDB09
{
  // Amaguem el constructor per defecte. */
  private XMLDB09() { }

  private static Collection establirConnexio(String[] args)
                 throws Exception
  /* Intenta establir una connexió contra el SGBD indicat per paràmetre, que
     ha de ser: eXist-db o BaseX o Sedna */
  { /* Mateix contingut que programa XMLDB02.java */ }

  public static void main(String[] args){
    String servei = "XQueryService";
    String versio = "1.0";
    Collection coll = null;
    try {
      coll = establirConnexio(args);
      System.out.println("Connexió establerta amb SGBD " + args[0]);
      // Recuperem el nom de la col·lecció actual
      System.out.println("Col·lecció actual: "+coll.getName());
      // Cerquem servei XPathQueryService per la col·lecció activa
      XQueryService xqs =
        (XQueryService) coll.getService(servei,versio);
      if (xqs == null)
        System.out.println("El SGBD no té el servei "+ servei +
                           " Versió : " + versio);
      else {
        String cad = "for $i in /mondial/"+
              "country[infant_mortality >= $finsMortInf - 20 and "+
              "infant_mortality < $finsMortInf] \n"+
              "order by number($i/infant_mortality) "+
```

```
            "return concat($i/infant_mortality,' - ',$i/name)";
        CompiledExpression ce = xqs.compile(cad);
        for (double x = 20.; x<=1000; x=x+20.) {
          // Preparem variable d'enllaç "finsMortInf" per definir
          // els extrems dels intervals
          xqs.declareVariable ("finsMortInf",x);
          ResourceSet rs = xqs.execute (ce);
          ResourceIterator ri = rs.getIterator();
          // Mostrem resultats, un a un, convertits a String
          if (ri.hasMoreResources()) {
            Resource r;
            System.out.println("Interval [" + (x-20) + "," + x + "[:");
            while (ri.hasMoreResources()) {
              r = ri.nextResource();
              System.out.println("\t"+(String)r.getContent());
            }
          }
        }
      }
      catch (Exception e) {
        e.printStackTrace();
      }
      finally { /* Tanquem connexió en qualsevol cas */
        try {
            if(coll != null) {
              coll.close();
              System.out.println("Connexió tancada");
            }
        }
        catch(XMLDBException xe) {
            xe.printStackTrace();
        }
      }
    }
}
```

Comprovem el funcionament d'aquest programa a *eXist-db*:

```
G:\>java -Dfile.encoding=cp850 proves.XMLDB09 eXist-db
Connexió establerta amb SGBD eXist-db
Col·lecció actual: db/xmldb
Interval [0.0,20.0[:
        2.2 - Andorra
        2.4 - Man
        ...
        6.3 - Spain
        ...
        19.6 - Qatar
Interval [20.0,40.0[:
        20 - Saint Lucia
        ...
...
Interval [140.0,160.0[:
        145.82 - Western Sahara
        149.7 - Afghanistan
```

Modificacions en documents XML. Transaccions.

El llenguatge *XQuery* és un llenguatge de només consulta i, en conseqüència, no facilita instruccions per efectuar modificacions en els documents XML (inserir nodes, modificar valors de nodes, modificar nodes i eliminar nodes).

Hi ha hagut diversos intents (taula 2-5) de crear un llenguatge estàndard per efectuar modificacions en documents XML, a banda de les extensions que cada fabricant hagi pogut incorporar en el seu SGBD.

Taula 2-5. Llenguatges per modificar documents XML

Llenguatge	Promotor	Any	SGBD que l'implementen API que l'incorporen
XUpdate	Grup XML:DB	2000	API XML:DB teòrica API XML:DB de *Sedna* API XML:DB d'*eXist-db*
Extensió *Update* per *XQuery*	Patrick Lehti	2001	SGBD *Sedna* SGBD *eXist-db*
XQuery Update Facility (XQUF) 1.0	W3C	2011	SGBD *BaseX*

L'API XML:DB que estem presentant, contempla, en la seva definició, la modificació de documents XML amb el llenguatge *XUpdate,* però no totes les implementacions de l'API XML:DB ho faciliten. Així, dels SGBD amb els què estem practicant, *Sedna* i *eXist-db* implementen el servei *XUpdateQueryService* que permet executar instruccions del llenguatge *XUpdate.*

Modificació de documents XML en el SGBD *Sedna*

Els SGBD que implementen l'API XML:DB amb el servei *XUpdateQueryService*, pot ser que ampliïn l'API amb algun servei que faciliti l'execució d'instruccions d'un altre llenguatge d'actualització de dades de documents XML diferent de *XUpdate* (com per exemple l'extensió *Update* de P. Lehti o l'*XQUF*). Aquest és el cas del SGBD *Sedna*, que a més de facilitar el servei *XUpdateQueryService*, amplia l'API XML:DB amb el servei *SednaUpdateService*.
Per altra banda, el servei *XUpdateQueryService* de *Sedna* no incorpora totes les instruccions del llenguatge *XUpdate*. En concret, no implementa les instruccions `xupdate:update`, `xupdate:cdata` i `xupdate:if`.

Centrem-nos en la utilització del servei *XUpdateQueryService* tal i com marca l'API XML:DB. És clar que, per utilitzar-lo, es suposa que es coneix el llenguatge *XUpdate*.

La documentació del llenguatge *XUpdate* es pot trobar al portal web d'aquest llenguatge: http://http://xmldb-org.sourceforge.net/xupdate/

La gestió del servei *XUpdateQueryService* bé marcada pels mètodes que proporciona la

interfície `XUpdateQueryService`:

- `update (instruccionsXUpdate)`, per executar un conjunt d'instruccions *XUpdate* contra la col·lecció activa.
- `updateResource (recurs, instruccionsXUpdate)`, per executar un conjunt d'instruccions *XUpdate* contra un recurs de la col·lecció activa.

Ambdós mètodes retornen el nombre de nodes modificats.

Exemple de programa que insereix un node

El següent programa insereix un node `country` a l'inici de tots els països (és a dir, immediatament abans del `country[1]`)
Recordem, per si el lector no està familiaritzat amb el llenguatge *XUpdate*, la sentència *XUpdate* per aconseguir l'objectiu desitjat:

```
<xupdate:modifications version="1.0"
  xmlns:xupdate="http://www.xmldb.org/xupdate">
  <xupdate:insert-before select="/mondial/country[1]">
    <xupdate:element name="country">
      <xupdate:attribute name="car_code">$$</xupdate:attribute>
      <name>nouPais</name>
    </xupdate:element>
  </xupdate:insert-before>
</xupdate:modifications>
```

Trobareu el fitxer `XMLDB10.java` dins el paquet de codi font a l'apartat "Material per desenvolupament sobre SGBD- XML natives utilitzant l'API XMLDB" dels annexos del web.

Vegem el codi del programa:

```
/*
 * Programa: XMLDB10.java
 * Objectiu: Exemple d'utilització del servei XUpdateQueryService (XML:DB)
 *           per executar una inserció de node via XUpdate:
 *           Inserir un node <country> a l'inici de l'arxiu "mondial.xml"
 * Autor...: Isidre Guixà
 */

package proves;
import org.xmldb.api.base.*;
import org.xmldb.api.DatabaseManager;
import org.xmldb.api.modules.*;

public class XMLDB10
{
  // Amaguem el constructor per defecte. */
  private XMLDB10() { }

  private static Collection establirConnexio(String[] args)
                throws Exception
  /* Intenta establir una connexió contra el SGBD indicat per paràmetre, que
     ha de ser: eXist-db o BaseX o Sedna */
  { /* Mateix contingut que programa XMLDB02.java */ }

  public static void main(String[] args){
```

```
    String servei = "XUpdateQueryService";
    String versio = "1.0";
    Collection coll = null;
    try {
      coll = establirConnexio(args);
      System.out.println("Connexió establerta amb SGBD " + args[0]);
      // Recuperem el nom de la col·lecció actual
      System.out.println("Col·lecció actual: "+coll.getName());
      // Cerquem servei XUpdateQueryService per la col·lecció activa
      XUpdateQueryService xuqs =
        (XUpdateQueryService) coll.getService(servei,versio);
      if (xuqs == null)
        System.out.println("El SGBD no té el servei "+ servei +
                           " Versió : " + versio);
      else {
        String cad = "<xupdate:modifications version=\"1.0\" "+
                     "xmlns:xupdate=\"http://www.xmldb.org/xupdate\">"+
                     "<xupdate:insert-before "+
                     "select=\"/mondial/country[1]\">"+
                     "<xupdate:element name=\"country\">"+
                     "<xupdate:attribute name=\"car_code\">"+
                     "$$"+
                     "</xupdate:attribute>"+
                     "<name>nouPais</name>"+
                     "</xupdate:element>"+
                     "</xupdate:insert-before>"+
                     "</xupdate:modifications>";
        long nre = xuqs.update(cad);
        System.out.println("\nInstrucció executada.");
        System.out.println("S'ha inserit "+nre+" nodes.");
      }
    }
    catch (Exception e) {
      e.printStackTrace();
    }
    finally { /* Tanquem connexió en qualsevol cas */
      try {
          if(coll != null) {
            coll.close();
            System.out.println("Connexió tancada");
          }
      }
      catch(XMLDBException xe) {
          xe.printStackTrace();
      }
    }
  }
}
```

Comprovem el funcionament d'aquest programa, executant-lo per cadascun dels tres SGBD. L'execució en *Sedna* i *eXist-db* provoca la sortida:

```
G:\>java -Dfile.encoding=cp850 proves.XMLDB10 nomSGBD
Connexió establerta amb SGBD ...
Col·lecció actual: ...(depèn del SGBD: xmldb o db/xmldb)
Instrucció executada.
S'ha modificat 1 nodes.
Connexió tancada
```

Podem comprovar, via la interfície gràfica que correspongui (*eXist Client Shell* o *SednaAdmin*) que la inserció s'ha dut a terme. Si en comprovem el funcionament en *BaseX,* veurem que ens informa que no existeix el servei *XUpdateQueryService*, tot i que el procés de compilació no dona cap error.

La inserció s'ha efectuat sobre tots els documents XML de la col·lecció activa, de manera que si hi hagués hagut dos documents amb node arrel `mondial` i nodes fill `country`, la modificació s'hagués dut a terme en tots els documents. Si es vol restringir la modificació a un document en concret, cal substituir:

```
long nre = xuqs.update(cad);
```

per:

```
long nre = xuqs.updateResource("mondial.xml",cad);
```

Exemple de programa que modifica un node

El següent programa canvia el nom dels països que tenen `$$` com a `car_code`.

Recordem, per si el lector no està familiaritzat amb el llenguatge *XUpdate*, la sentència *XUpdate* per aconseguir l'objectiu desitjat:

```
<xupdate:modifications version="1.0"
  xmlns:xupdate="http://www.xmldb.org/xupdate">
  <xupdate:update select="/mondial/country[@car_code='$$']/name">
  nouNom
  </xupdate:update>
</xupdate:modifications>
```

Trobareu el fitxer `XMLDB11.java` dins el paquet de codi font a l'apartat "Material per desenvolupament sobre SGBD- XML natives utilitzant l'API XMLDB" dels annexos del web.

Vegem el codi del programa:

```
/*
 * Programa: XMLDB11.java
 * Objectiu: Exemple d'utilització del servei XUpdateQueryService (XML:DB)
 *           per executar una modificació de node via XUpdate:
 *           Modificar el valor d'un determinat node <country>
 * Autor...: Isidre Guixà
 */

package proves;
import org.xmldb.api.base.*;
import org.xmldb.api.DatabaseManager;
import org.xmldb.api.modules.*;

public class XMLDB11
{
  // Amaguem el constructor per defecte. */
  private XMLDB11() { }

  private static Collection establirConnexio(String[] args)
                  throws Exception
  /* Intenta establir una connexió contra el SGBD indicat per paràmetre, que
     ha de ser: eXist-db o BaseX o Sedna */
  { /* Mateix contingut que programa XMLDB02.java */ }

  public static void main(String[] args){
    String servei = "XUpdateQueryService";
    String versio = "1.0";
    Collection coll = null;
```

```
    try {
      coll = establirConnexio(args);
      System.out.println("Connexió establerta amb SGBD " + args[0]);
      // Recuperem el nom de la col·lecció actual
      System.out.println("Col·lecció actual: "+coll.getName());
      // Cerquem servei XUpdateQueryService per la col·lecció activa
      XUpdateQueryService xuqs =
        (XUpdateQueryService) coll.getService(servei,versio);
      if (xuqs == null)
        System.out.println("El SGBD no té el servei "+ servei +
                           " Versió : " + versio);
      else {
        String cad = "<xupdate:modifications version=\"1.0\" "+
                     "xmlns:xupdate=\"http://www.xmldb.org/xupdate\">"+
                     "<xupdate:update "+
                     "select=\"/mondial/country[@car_code='$$']/name\">"+
                     "nouNom"+
                     "</xupdate:update>"+
                     "</xupdate:modifications>";
        long nre = xuqs.update(cad);
        System.out.println("\nInstrucció executada.");
        System.out.println("S'ha modificat "+nre+" nodes.");
      }
    }
    catch (Exception e) {
      e.printStackTrace();
    }
    finally { /* Tanquem connexió en qualsevol cas */
      try {
          if(coll != null) {
            coll.close();
            System.out.println("Connexió tancada");
          }
      }
      catch(XMLDBException xe) {
          xe.printStackTrace();
      }
    }
  }
}
```

Comprovem el funcionament d'aquest programa, executant-lo per cadascun dels tres SGBD. L'execució en *eXist-db* provoca la sortida:

```
G:\>java -Dfile.encoding=cp850 proves.XMLDB10 eXist-db
Connexió establerta amb SGBD exist-db
Col·lecció actual: /db/xmldb
Instrucció executada.
S'ha modificat 1 nodes.
Connexió tancada
```

Podem comprovar, via la interfície gràfica *eXist Client Shell* que la modificació s'ha dut a terme. Si en comprovem el funcionament en *BaseX,* veurem que ens informa que no existeix el servei *XUpdateQueryService*, tot i que el procés de compilació no dona cap error. Si en comprovem el funcionament en *Sedna*, ens informa que aquesta instrucció no està permesa (és una de les tres instruccions *XUpdate* no suportades per *Sedna).*

Exemple de programa que elimina un node

El següent programa elimina els països que tenen `$$` com a `car_code`. Recordem, per si el lector no està familiaritzat amb el llenguatge *XUpdate*, la

sentència *XUpdate* per aconseguir l'objectiu desitjat:

```
<xupdate:modifications version="1.0"
  xmlns:xupdate="http://www.xmldb.org/xupdate">
  <xupdate:remove select="/mondial/country[@car_code='$$']">
  </xupdate:remove>
</xupdate:modifications>
```

> Trobareu el fitxer `XMLDB12.java` dins el paquet de codi font a l'apartat "Material per desenvolupament sobre SGBD- XML natives utilitzant l'API XMLDB" dels annexos del web.

Vegem el codi del programa:

```
/*
 * Programa: XMLDB12.java
 * Objectiu: Exemple d'utilització del servei XUpdateQueryService (XML:DB)
 *           per executar una eliminació de node via XUpdate:
 *           Eliminar un determinat node <country>
 * Autor...: Isidre Guixà
 */

package proves;
import org.xmldb.api.base.*;
import org.xmldb.api.DatabaseManager;
import org.xmldb.api.modules.*;

public class XMLDB12
{
  // Amaguem el constructor per defecte. */
  private XMLDB12() { }

  private static Collection establirConnexio(String[] args)
                throws Exception
  /* Intenta establir una connexió contra el SGBD indicat per paràmetre, que
     ha de ser: eXist-db o BaseX o Sedna */
  {
    if (args.length<1){
      throw new Exception("Executeu el programa indicant, primer, el SGBD "+
                          "al que connectar: eXist-db, BaseX o Sedna.");
    }
    Database dbDriver;
    Collection coll;
    if (args[0].equalsIgnoreCase("eXist-db")) {
      dbDriver = (Database)
          Class.forName("org.exist.xmldb.DatabaseImpl").newInstance();
      DatabaseManager.registerDatabase(dbDriver);
      coll = DatabaseManager.getCollection(
          "xmldb:exist://localhost:8080/exist/xmlrpc/db/xmldb",
          "admin", "admin");
    }
    else if (args[0].equalsIgnoreCase("BaseX")) {
      dbDriver = (Database)
          Class.forName("org.basex.api.xmldb.BXDatabase").newInstance();
      DatabaseManager.registerDatabase(dbDriver);
      coll = DatabaseManager.getCollection(
          "xmldb:basex://localhost:1984/xmldb", "admin", "admin");
    }
    else if (args[0].equalsIgnoreCase("Sedna")) {
```

```
      dbDriver = (Database)
          Class.forName("net.cfoster.sedna.DatabaseImpl").newInstance();
      DatabaseManager.registerDatabase(dbDriver);
      coll = DatabaseManager.getCollection(
          "xmldb:sedna://localhost:5050/db/xmldb", "SYSTEM", "MANAGER");
    }
    else
      throw new Exception("Executeu el programa indicant, primer, el SGBD "+
                          "al que connectar: eXist-db, BaseX o Sedna.");
    return coll;
  }

  public static void main(String[] args){
    String servei = "XUpdateQueryService";
    String versio = "1.0";
    Collection coll = null;
    try {
      coll = establirConnexio(args);
      System.out.println("Connexió establerta amb SGBD " + args[0]);
      // Recuperem el nom de la col·lecció actual
      System.out.println("Col·lecció actual: "+coll.getName());
      // Cerquem servei XUpdateQueryService per la col·lecció activa
      XUpdateQueryService xuqs =
        (XUpdateQueryService) coll.getService(servei,versio);
      if (xuqs == null)
        System.out.println("El SGBD no té el servei "+ servei +
                           " Versió : " + versio);
      else {
        String cad = "<xupdate:modifications version=\"1.0\" "+
                     "xmlns:xupdate=\"http://www.xmldb.org/xupdate\">"+
                     "<xupdate:remove "+
                     "select=\"/mondial/country[@car_code='$$']\">"+
                     "</xupdate:remove>"+
                     "</xupdate:modifications>";
        long nre = xuqs.update(cad);
        System.out.println("\nInstrucció executada.");
        System.out.println("S'ha eliminat "+nre+" nodes.");
      }
    }
    catch (Exception e) {
      e.printStackTrace();
    }
    finally { /* Tanquem connexió en qualsevol cas */
      try {
          if(coll != null) {
            coll.close();
            System.out.println("Connexió tancada");
          }
      }
      catch(XMLDBException xe) {
          xe.printStackTrace();
      }
    }
  }
}
```

Comprovem el funcionament d'aquest programa, executant-lo per cadascun dels tres SGBD. L'execució en *Sedna* i *eXist-db* provoca la sortida:

```
G:\>java -Dfile.encoding=cp850 proves.XMLDB12 nomSGBD
Connexió establerta amb SGBD ...
Col·lecció actual: ...(depèn del SGBD: xmldb o db/xmldb)
Instrucció executada.
```

```
S'ha modificat 1 nodes.
Connexió tancada
```

Podem comprovar, via la interfície gràfica que correspongui (*eXist Client Shell* o *SednaAdmin*) que la inserció s'ha dut a terme. Si en comprovem el funcionament en *BaseX,* veurem que ens informa que no existeix el servei *XUpdateQueryService*, tot i que el procés de compilació no dona cap error.

En els tres exemples anteriors s'ha executat instruccions d'actualització de la base de dades sense validar (`commit`) en cap cas els canvis i els canvis han esdevingut permanents. Això ha estat així per què no hem obert cap transacció i, en conseqüència, cada instrucció *XUpdate* va acompanyada de la corresponent validació.

L'API XML:DB contempla, en la seva definició, la gestió de transaccions, però no totes les implementacions de l'API XML:DB ho faciliten. Així, dels SGBD amb els què estem practicant, només *Sedna* implementa el servei *TransactionService* que permet executar instruccions del llenguatge *XUpdate* dins una transacció, amb la possibilitat de validar (`commit`) o desfer (`rollback`) els canvis efectuats dins la transacció.

La gestió del servei *TransactionService* bé marcada pels mètodes que proporciona la interfície `TransactionService`:

- `begin()`, per indicar l'inici d'una transacció.
- `commit()`, per finalitzar una transacció validant els canvis efectuats dins la transacció.
- `rollback()`, per finalitzar una transacció desfent els canvis efectuats dins la transacció.