Títol del Treball Fi de Carrera

Sistemes de BD:XML natius. Estat de l'art

Josep Furriols Perelló
Enginyeria Tècnica Informàtica de Gestió

Consultor: Oscar Celma

18/06/2004

Aquest treball està dedicat a la meva dona per la paciència que ha tingut i el suport que m'ha donat durant els 5 anys que he estat estudiant la carrera

Resum del treball

El projecte es basa en avaluar l'estat actual dels Sistemes de BD:XML natius que permeten desar i recuperar documents XML, alhora que es compara amb les solucions clàssiques adoptades pels SGBD relacionals els quals afegeixen extensions (XML, per exemple) al capdamunt del model relacional.

En aquest treball inicialment el que farem és un resum dels coneixements bàsics que cal tenir per tal de poder fer una comparativa entre les bases de dades XML natives (s'ha escollit la BD:XML eXist) i les bases de dades relacions (el SGBD escollit ha estat Oracle 9i).

En primer lloc explico la sintaxi bàsica XML, així com algunes eines associades al llenguatge (fulles d'estil CSS, etc.)..

Un cop explicat els coneixements bàsics sobre XML faig una petita introducció als llenguatges de consulta XPath i XQuery, coneixements que seran imprescindibles a la part final del treball per comparar els sistemes de bases de dades estudiats.

Tot seguit, explico les característiques principals de eXist i Oracle 9i i explico com es desa la informació en la base de dades, com s'indexa, com es fan les consultes i els avantatges i inconvenients d'usar un sistema gestor de bases de dades o un altre.

En la part final de treball faig una comparativa, intentant posar exemples entenedors, tot mostrant com es realitza una mateixa consulta en eXist i Oracle 9i usant els mitjans disponibles per consultar en cada base de dades.

I per acabar exposaré les conclusions a les quals he arribat després de fer aquest TFC.

Índex

Capítol 1. Introducció 1
1.1. Objectius 1
1.2. Mètode seguit 1
1.3. Planificació 1
1.3.1. Cerca d'informació i estudi del llenguatge XML 1
1.3.2. Cerca d'informació i estudi del llenguatge de consultes XQuery i XPath 1
1.3.3. Cerca d'informació, instal·lació i estudi de eXist i XIndice 2
1.3.4. Cerca d'informació, instal·lació i estudi del funcionament d'Oracle 9i amb XML 2
1.3.5. Comparativa d'Oracle 9i amb eXist o XIndice 2
Capítol 2. El llenguatge de marques XML 3
2.1. Orígens 3
2.2. Sintaxis XML 3
2.2.1. Declaració XML 4
2.2.2. Etiquetes 4
2.2.3. Atributs 4
2.2.4. Comentaris 4
2.3. Definició de tipus de document (DTD) 5
2.3.1. Declaració d'elements 5
2.3.2. Declaració d'atributs 5
2.4. Schema 7
2.4.1. Associar esquemes a documents XML 7
2.4.2. Element simples 7
2.4.3. Atributs 7
2.4.4. Restriccions 8
2.4.5. Elements complexes 9
2.5. Fulls d'estil CSS 10
Capítol 3. Els llenguatges de consulta XML 12
3.1. XPath 12
3.2. XQuery 15
3.2.1. Introducció 15
3.2.2. Consultes tipus XPath 16
3.2.3. Creació de nodes (constructors) 16
3.2.4. Expressions FLWOR (flower expressions) 17
3.2.5. Expressions condicionals 18
3.2.6. Quantificadors 19
3.2.7. Operadors 19
3.2.8. Funcions 20
3.2.9. Datatype expressions 21
Capítol 4. Bases de dades XML natives 22
4.1. Introducció 22
4.2. Característiques comunes: 22
4.3. eXist 23
4.3.1. Descripció 23
4.3.2. Indexat i emmagatzematge 23
4.3.3. Implementació del llenguatge de consultes 26
4.3.4. Consultes a eXist 28
4.4. XIndice 34
Capítol 5. Bases de dades relacionals i XML 36
5.1. Introducció 36
5.1.1. Analitzador XML de Oracle 36
5.1.2. El generador de classes XML de Oracle 36
5.1.3. La SQL XML Utility de Oracle 37
5.2. Oracle 9i: persistència de documents XML 39
5.2.1. Emmagatzematge de documents XML com a CLOB 40
5.2.2. Cerques dins de XMLType 41
5.2.3. Manipulació de documents XML en columnes XMLType 44
Capítol 6. Conclusions i tancament 45

6.1. Resum eXist 45
6.2. Resum Oracle 9i 45
6.3. Conclusions finals 45
6.4. Línies de futur 46
Bibliografia 48

Índex de figures

Figura 1. Primer exemple d'XML 3
Figura 2. Exemple etiquetes XML 4
Figura 3. Exemple document XML 5
Figura 4. Llistat operadors DTD 5
Figura 5. Exemple DTD 6
Figura 6. Document XML amb declaració de DTD 6
Figura 7. Exemple restricció Schema (valors dins un rang) 8
Figura 8. Exemple restricció Schema (valors restringits a un conjunt de valors possibles) 8
Figura 9. Elements per establir restriccions a XML Schema 8
Figura 10. Exemple de definició d'elements complexes a XML Schema 9
Figura 11. Exemple de definició d'elements complexes a XML Schema (reusable) 9
Figura 12. Exemple de l'indicador xsd:all de XML Schema 9
Figura 13. Exemple de l'indicador xsd:choice de XML Schema 10
Figura 14. Exemple de l'indicador maxOccurs de XML Schema 10
Figura 15. Exemple de l'indicador xsd:any de XML Schema 10
Figura 16. Exemple document css 11
Figura 17. Exemple de document XML amb un css associat 11
Figura 18. Document XML pels exemples de XPath 12
Figura 19. Document XML per fer consultes amb XQuery 16
Figura 20. Exemple de creació de nodes amb XQuery 16
Figura 21. Exemple de creació de nodes amb XQuery (amb el constructor explícit) 16
Figura 22. Exemple de creació de contingut dinàmicament a XQuery 16
Figura 23. Resultat de la creació de contingut dinàmicament a XQuery 17
Figura 24. Exemple de consulta FLWOR 17
Figura 25. Exemple del funcionament de les expressions condicionals a XQuery 18
Figura 26. Resultat de la consulta de la figura 24 18
Figura 27. Exemple quantificadors some-in-satisfies de XQuery 19
Figura 28. Exemple quantificadors every-in-satisfies de XQuery 19
Figura 29. Operadors de comparació de XQuery 20
Figura 30. Operador union de XQuery 20
Figura 31. Exemple de funció a XQuery 21
Figura 32. Logo eXist 23
Figura 33. Multi-root B+- Tree 26
Figura 34. B+- Tree keys 26
Figura 35. Document books.xml 29
Figura 36. Document books2.xml 29
Figura 37. Document books.dtd 29
Figura 38. Captura pantalla eXist (on es pot visualitzar la creació de la col·lecció TFC) 30
Figura 39. Captura de la pantalla de consultes eXist (es visualitza el resultat d'una consulta) . 30
Figura 40. Resulta de una consulta feta a eXist amb document 31
Figura 41. Exemple de consulta a eXist (cerca d'un llibre en concret a la base de dades) 31
Figura 42. Exemple de consulta a eXist (obtenir resultats en un ordre determinat) 31
Figura 43. Exemple de consulta a eXist (nou document a partir de les dades consultades) 32
Figura 44. Exemple de consulta a eXist (obtenir dades ordenades a partir de certa lletra) 32
Figura 45. Logo XIndice 34
Figura 46. Col·leccions XIndice 34
Figura 47. Logo Oracle 36
Figura 48. Mode execució de SQL XML Utility 37
Figura 49. Codi java per generar documents XML a partir d'una consulta 38

Figura 50. Codi java per emmagatzemar les dades d'un document XML en una base de dades 38
Figura 51. Instruccions SQL per crear una taula per guardar un document XML 40
Figura 52. Instruccions SQL per extreure el contingut d'un document XML i emmagatzemar-lo dins d'una columna 41
Figura 53. Instruccions per modificar el format de sortida de les dades..... 41
Figura 54. Pantalla de connexió a SQL Plus 41
Figura 55. Consulta SQL*Plus Llibres (getClobVal()) 42
Figura 56. Consulta SQL*Plus Autors (getClobVal()) 42
Figura 57. Consulta fallida SQL*Plus Llibres (getStringVal())..... 43
Figura 58. Consulta fallida SQL*Plus Cerca dades d'un llibre (getStringVal()) 43
Figura 59. Inserció dades usant la comanda SQL insert into 44
Figura 60. Modificació de dades usant la comanda SQL update 44
Figura 61. Eliminació de dades usant la comanda SQL delete 44

Capítol 1. Introducció

El projecte es basa en avaluar l'estat actual de diferents Sistemes de BD:XML natius que permeten desar i recuperar documents XML.

XML està àmpliament adoptat com a format robust i flexible que facilita la portabilitat de dades entre sistemes. Darrerament, vàries iniciatives han sorgit dins l'àmbit de les BD:XML natives, és a dir Sistemes Gestors de Bases de Dades que no es basen en el model relacional clàssic. Aquest nou paradigma ha vingut de la mà amb una sèrie de llenguatges per accedir a les dades contingudes en un document XML (com per exemple; XPath, XQuery, ...).
L'objectiu del projecte, doncs, és avaluar diferents Sistemes de BD:XML natius (com per exemple *eXist* i *XIndice*) tot comparant-los, en la mesura que sigui possible, amb els SGBD tradicionals que incorporen la gestió de dades en format XML (com per exemple *Oracle 9i*).

Caldrà investigar els avantatges i inconvenients dels diferents sistemes gestors de BD així com l'ús dels diferents llenguatges de consulta XML actuals: XPath i XQuery.

1.1. Objectius

Els objectius principals del treball són:

- Conèixer l'estructura i organització de les BD:XML natives així com utilitzar alguns sistemes concrets (eXist i XÍndice)
- Analitzar els llenguatges de consulta actuals que incorporen els Sistemes de BD:XML natius.
- Comparar un sistema de BD:XML natius (eXist) amb un SGBD relacional actual (Oracle 9i)

1.2. Mètode seguit

La metodologia seguida com es pot veure en la planificació que hi ha a continuació. Consisteix, bàsicament, en cercar informació dels diferents temes sobre els que tractarà el treball i anar-ho documentant. Conforme vagi trobant informació interessant l'aniré resumint i estructurant de manera clara i entenedora fins obtenir el document final amb tot el TFC acabat. Intentaré en tot moment posar exemples dels conceptes de manera que quedi ben clar el que es pretén explicar.

1.3. Planificació

1.3.1. Cerca d'informació i estudi del llenguatge XML

Inicialment he de fer una tasca de cerca d'informació incidint especialment amb tot el que fa referència a bases de dades. Aquesta cerca la faré tant a internet, com a la documentació proporcionada per la UOC, com a llibres, i materials diversos que espero obtenir.
Període estimat: 1 setmana

1.3.2. Cerca d'informació i estudi del llenguatge de consultes XQuery i XPath

Un cop ja adquirits els coneixements bàsics d'XML passaré a estudiar els llenguatges de consulta XQuery i XPath intentant en tot moment posar exemples per tal de que quedi clar el seu funcionament.
Període estimat: 1 setmanes

1.3.3. Cerca d'informació, instal·lació i estudi de eXist i XIndice

En aquest cas farà falta més temps degut a la pèrdua de temps que comporta la instal·lació del programari i a que es tracta de BD:XML natives molt noves de les quals penso que em serà difícil trobar informació clara i fàcil d'entendre.
Període estimat: 5 setmanes

1.3.4. Cerca d'informació, instal·lació i estudi del funcionament d'Oracle 9i amb XML

Primer de tot caldrà obtenir el programari, instal·lar el software i a partir d'aquí començar una cerca de documentació al respecte del funcionament d'Oracle 9i.
Cal esmentar que prèviament no he utilitzat aquest gestor de bases de dades
Període estimat: 5 setmanes

1.3.5. Comparativa d'Oracle 9i amb eXist o XIndice

Un cop obtinguts els coneixements bàsics de XML, de les de BD:XML natives i d'Oracle 9i caldrà comparar ambdues aproximacions.
Període estimat: 3 setmanes

Capítol 2. El llenguatge de marques XML

2.1. Orígens

L'origen del XML el trobem en els intents de trobar un estàndard per a l'intercanvi d'informació entre sistemes. Un dels primers intents seriosos el va desenvolupar IBM al crear el GML (Generalized Markup Language) creat amb l'objectiu de compartir informació entre documents. Aquest llenguatges va anar evolucionant i finalment va aparèixer l'SGML (Standardized Generalized Markup Language) el qual va ser reconegut com a estàndard internacional (ISO 8879). SGML, però, tenia una imperfecció que va ser clau en el seu fracàs: era massa complex. No obstant això SGML no ha desaparegut del tot i actualment cada dia usem HTML que no és més que un llenguatge basat en l'SGML.

HTML és un llenguatge de marques molt més fàcil i senzill d'usar que el SGML. És el llenguatge estàndard de la web. Fins i tot hi ha processadors de textos que permeten guardar els seus documents en aquest format. No obstant això aquest llenguatge presenta certs desavantatges:

- Està pensat per mostrar-se en navegadors però no per la impressió o tasques de disseny
- Les etiquetes existents són limitades i això implica que el llenguatge no és suficientment flexible
- No permet mostrar contingut dinàmic i això és especialment important en el nostre cas. Imaginem que volem mostrar la informació de productes emmagatzemada en una base de dades permanentment actualitzada. Això amb HTML és impossible.
- L'estructura i el disseny de la pàgina estan barrejats

XML a diferència de l'HTML és un llenguatge purament estructural (no inclou cap referència al disseny). En aquest sentit queda clar que ens farà falta un llenguatge de disseny. No obstant això XML presenta unes característiques prou importants que l'han portat a l'èxit actual com a llenguatge d'interoperabilitat entre sistemes:

- Permet definir etiquetes pròpies
- Se li poden assignar atribut a les etiquetes
- Les etiquetes i els atributs es defineixen mitjançant un esquema (DTD o XML Schema)
- L'estructura i el disseny estan separats

```
<persona>
        <nom> Josep F </nom>
        <telef> 93 123 45 67 </telef>
</persona>
```

Figura 1. Primer exemple d'XML

2.2. Sintaxis XML

Un document XML es diu que està **ben format** si compleix totes les regles de sintaxi que veurem a continuació. Els documents XML tenen una estructura jeràrquica molt estricta i els documents ben formats l'han de complir. Si a més a més de ser ben formats segueixen una estructura i semàntica determinada per un DTD o XML Schema, és diu que el document és **vàlid.**

Les principals regles que ha de complir un document XML perquè es consideri ben format són:

- Tot document XML té un únic element arrel
- Les etiquetes han d'estar correctament aniuades
- Els noms dels elements han de complir certes convencions de nomenclatura

També cal tenir en compte, tot i que no es requisit perquè el document es consideri ben format, que XML mantindrà els espais en blanc i els salts de línia posats dins de les etiquetes. A més a

més cal anar en compte amb el tema majúscules i minúscules ja que XML interpreta diferent <nom> que <Nom> i per tant <Nom>Josep</nom> no seria correcte.

L'XML té un conjunt de caràcters reservats: <,>,&," i '. No obstant això si els hem d'usar ho podem fer usant les entitats següents: <, >, &, " i '.

2.2.1. Declaració XML

Un document XML generalment comença amb una línia com aquesta tot i que no és obligatòria:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

amb aquesta línia el que indiquem és que el que ve a continuació segueix el format de l'XML. Anem a explicar que signifiquen els diferents atributs de la línia:
version: ens indica la versió d'XML que s'usarà en el document
encoding: es un camp opcional que indica la codificació que s'usarà en el document
standalone: és un camp opcional que ens indica si el document va acompanyat d'un DTD o no.

2.2.2. Etiquetes

Tal i com ja hem explicat a la introducció tot document XML té un element (etiqueta) arrel. Aquesta arrel s'obre a l'inici del document i es tanca al final. Tota etiqueta que s'obri cal que sigui tancada respecte el seu respectiu ordre. Tal i com ja s'ha comentat cal que les etiquetes estiguin correctament aniuades. Un petit exemple de tot l'explicat fins el moment podria ser el següent:

```
 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<llistin>
        <persona>
                <nom> Josep F </nom>
                <telef> 93 123 45 67 </telef>
                <codi />
        </persona>
        <persona>
                <nom>Manel</nom>
                <telef> 93 123 45 76 </telef>
                <codi>PM-25</codi>
        </persona>
</llistin>
```

Figura 2. Exemple etiquetes XML

Aquí veiem un document ben format on a la seva primera línia hi ha la declaració i una etiqueta llistin (element arrel) que conté una sèrie d'elements persona, que alhora conté el nom i telef. Podem observar que tots els elements oberts estan tancats i correctament aniuats. Veiem també que hi ha una etiqueta especial (<codi/>). Amb aquesta etiqueta indiquem que l'element codi està buit i ho podem indicar d'aquesta manera tot i que també seria correcte fer: <codi></codi>.

2.2.3. Atributs

Les etiquetes poden contenir atributs que representen informació addicional d'aquests. El valor d'aquests atributs ha d'anar entre cometes dobles o simples.
Un exemple podria ser:
<nom apodo="furri"> Josep Furriols </nom>

2.2.4. Comentaris

Per tal de facilitar la lectura d'un document XML pot ser interessant usar comentaris. Els comentaris comencen pel caràcter <!--i acaben amb els -->

Un exemple amb tot el que hem vist fins ara seria:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- Document d'exemple -->
<llistin>
	<persona>
		<nom> Josep F </nom>
		<telef> 93 123 45 67 </telef>
		<codi />
	</persona>
	<persona>
		<nom>Manel</nom>
		<telef> 93 123 45 76 </telef>
		<codi>PM-25</codi>
	</persona>
</llistin>
```

Figura 3. Exemple document XML

2.3. Definició de tipus de document (DTD)

A l'inici del l'explicació de la sintaxi de XML hem comentat que un document serà vàlid si compleix les normes d'un DTD. En aquest apartat veurem que significa i perquè serveix.

Un DTD és un conjunt de regles que defineix la sintaxi de les etiquetes, el seu nom, el seu ordre, la seva jerarquia, etc. En el DTD podem indicar els atributs que estan permesos i els seus possibles valors. També es pot establir si l'ús d'una etiqueta o atribut és opcional o obligatori.

2.3.1. Declaració d'elements

Usarem la següent sintaxi:
<!ELEMENT nomElement categoria>
on categoria prendrà com a valor EMPTY, #PCDATA o ANY
assignarem EMPTY a aquelles etiquetes que no tenen contingut
assignarem #PCDATA a aquelles etiquetes que contenen conjunts de caràcters

Llista d'operadors:

Símbol	Explicació
,	(partícula i): la coma indica ordenació exacte dels elements en l'ordre establert
|	(partícula o): la barra significa que solament es pot usar un dels elements proposats
+	El signe suma indica que l'element apareix una vegada o més
?	El signe interrogació indica que l'element és opcional
*	El signe asterisc indica que l'element es pot usar tantes vegades com es vulgui però el seu us és obligatori

Figura 4. Llistat operadors DTD

2.3.2. Declaració d'atributs

Usarem la següent sintaxi:
<!ATTLIST nomElement nomAtribut tipusAtribut valorDefecte>

tipusAtribut: ens indica el tipus d'atribut que és. Pot tenir els següents valors:
CDATA indica que es un atribut de tipus String

NMTOKEN és equivalent a CDATA però elimina els espais en blanc
(en1 | en2 | …) indica que es una enumeració
ID indica que l'atribut és un identificador
IDREF indica que l'atribut es de tipus referència a un identificador de un altra element

valorDefecte: ens dona indicacions sobre el valor que ha de prendre l'atribut i de les característiques que ha de tenir. Pot prendre els següents valors:

#REQUIRED indica que l'atribut és obligatori
#IMPLIED indica que el atributo és opcional
#FIXED Indica un valor fix per l'atribut
"valor" predefineix valor com a valor per defecte de l'atribut

Veiem ara un exemple d'una definició de DTD i XML d'exemple:

Document dtd

```
<!ELEMENT llistin (persona+)>
<!ELEMENT persona (nom, telef, codi?)>
<!ELEMENT nom (#PCDATA) >
<!ELEMENT telef (#PCDATA) >
<!ELEMENT codi (#PCDATA) >
<!ATTLIST nom apodo CDATA #REQUIRED>
```

Figura 5. Exemple DTD

Document xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- A la línia de sota declarem el DTD que usarem per aquest document xml-->
<!DOCTYPE llistin SYSTEM "llistin.dtd">
<llistin>
	<persona>
		<nom apodo="furri"> Josep Furriols </nom>
		<telef>a93 123 45 67 </telef>
		<codi/>
	</persona>

	<persona>
		<nom apodo="apocapoc">  Marc Poch </nom>
		<telef>a93 345 67 78 </telef>
		<codi/>
	</persona>
</llistin>
```

Figura 6. Document XML amb declaració de DTD

Per validar el document xml amb el seu corresponen dtd ens cal una eina de validació. Per exemple podem descarregar Internet Explorer Tools for Validating XML and Viewing XSLT Output des del link següent:
http://www.microsoft.com/downloads/details.aspx?familyid=d23c1d2c-1571-4d61-bda8-adf9f6849df9&languageid=f49e8428-7071-4979-8a67-3cffcb0c2524&displaylang=en

Un cop instal·lada aquesta eina en el menú contextual de l'Internet Explorer ens apareixerà una nova funcionalitat Validate XML. Per validar un document xml només l'haurem d'obrir amb l'Internet Explorer i fer clic en el botó dret sobre del document i seleccionar Validate XML. Si el document s'ha validat correctament ens apareixerà un missatge que dirà Validation Successful, en cas contrari ens indicarà els errors trobats.

2.4. Schema

Un schema XML és similar a un DTD, és a dir defineix quins elements pot tenir un document XML, com estan organitzades les dades i quin atributs i de quin tipus poden ser.

Aleshores la pregunta seria perquè usar esquemes si ja tenim els DTD?. Doncs perquè els esquemes tenen una sèrie de avantatges importants sobre els DTD:

- Usen sintaxi XML al contrari que els DTD
- Permeten especificar els tipus de dades
- Són extensibles

Per exemple un esquema ens permet definir el tipus de contingut d'un element o d'un atribut i especificar si ha de ser un enter, una cadena de text, una data, etc. Amb un DTD això no ho podríem fer.

Per entendre el seu funcionament el millor es veure'n un exemple senzill.

2.4.1. Associar esquemes a documents XML

Hi ha varies maneres d'associar esquemes a documents XML:

```
<libros xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamesspaceSchemaLocation="llibres.xsd">

<libros xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi: SchemaLocation="http://www.us.com/schema/note.xsd">
```

Els elements usats en la creació d'un esquema "procedeixen" del espai de noms: http://www.w3.org/2001/XMLSchema

L'element schema és l'arrel del document en el qual es defineix l'esquema:

```
<xsd:schema xmlns:xsd="http:// www.w3.org/2001/SMLSchema">
</xsd:schema>
```

2.4.2. Element simples

Un element simple és un element que només pot contenir text però no altres elements ni atributs. Per definir un element simple usem la següent sintaxi:

```
<xsd:element name="xxx" type="yyy"/>
```

Per exemple:

```
<xsd:element name="cognom" type="string"/>
```

Els tipus de dades (type) més usats són: xsd:string, xsd:decimal, xsd:boolean, xsd:date, xsd:time

Un element simple pot tenir un valor per defecte i un valor fix. Això ho indiquem usant els atributs default i fixed. Per exemple:

```
<xsd:element name="color" type"xsd:string" default="red"
```

2.4.3. Atributs

Els atributs es declaren de forma similar als elements simples i a l'igual que els elements simples són d'un tipus de dada xsd:string, xsd:decimal, xsd:boolean, xsd:date, xsd:time. Si un element pot anar acompanyat per atributs s'ha de declarar com a un element complex.

El format es el següent: <xsd:attribute name="xxx" type="yyy"/>

Per exemple: <xsd:attribute name="idioma" type="xsd:string"/>

Igual que els elements els atributs poden tenir valors per defecte i valors fixes. Per defecte els atributs son opcionals si volem que un atribut sigui obligatori s'ha d'afegir a la seva declaració usant l'atribut use (que pot tenir per valor optional o required).

Per exemple: <xsd:atrribute name="llengua" type="xsd:string" use="required"/>

2.4.4. Restriccions

Les restriccions permeten restringir el valor que es pot donar a un element o atribut XML. Mitjançant les restriccions podem indicar que un valor ha d'estar comprès dins un rang determinar, ha de ser un valor de una llista prèviament establert, ha de ser major o menor que un valor determinat, ...

Tipus de restriccions:

- valor dins un rang
- el valor esta restringit a un conjunt de valors establerts
- restringir el valor de un element a una sèrie de caràcters
- longitud dels valors dels elements

Veiem un parell d'exemples:

Valor dins un rang:

```
<xsd:element name="edat">
        <xsd:simpleType>
                <xsd:restriction base="xsd:integer">
                        <xsd:minInclusive value="0"/>
                        <xsd:maxInclusive value="100"/>
                </xsd:restriction>
        </xsd:simpleType>
</xsd:element>
```

Figura 7. Exemple restricció Schema (valors dins un rang)

Valor restringit a un conjunt de valors possibles:

```
<xsd:element name="car">
        <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="Audi"/>
                        <xsd:enumeration value="Golf"/>
                        <xsd:enumeration value="BMW"/>
                </xsd:restriction>
        </xsd:simpleType>
</xsd:element>
```

Figura 8. Exemple restricció Schema (valors restringits a un conjunt de valors possibles)

Elements per establir restriccions	
Enumeration	Estableix una llista de valors acceptats
FractionDigits	Per establir el nombre de xifres decimals
Length	Per establir el nombre de caràcters obligatoris
MaxExclusive i maxInclusive	Per establir el valor màxim d'un rang
MinExclusive i minInclusive	Per establir el valor mínim d'un rang
MaxLength i minLength	Per establir el nombre màxim i mínim de caràcters permesos
Pattern	Per definir una seqüència de caràcters permesos
TotalDigits	Per establir el nombre exacte de dígits permesos
WhiteSpace	Per indicar com s'han de tractar els espais en blanc

Figura 9. Elements per establir restriccions a XML Schema

2.4.5. Elements complexes

Son elements que tenen altres elements fills o que tenen atributs. Es solen dividir en 4 tipus:

- elements buits
- elements no buits amb atributs
- elements amb elements fills
- elements amb fills i amb text o valor propi

Per definir elements complexes s'utilitza la següent sintaxi:

```
<xsd:element name="empleat">
	<xsd:complexType>
		<xsd:sequence>
			<xsd:element name="nom" type="xsd:string"/>
			<xsd:element name="cognom" type="xsd:string"/>
		</xsd:sequence>
	</xsd:complexType>
</xsd:element>
```

Figura 10. Exemple de definició d'elements complexes a XML Schema

També podem usar aquesta altre sintaxi per reusar la definició dels elements fills a diversos elements:

```
<xsd:element name="empleat" type="personaInfo"/>
<xsd:element name="estudiant" type="personaInfo"/>
<xsd:complexType name="personaInfo">
	<xsd:sequence>
		<xsd:element name="nom" type="xsd:string"/>
		<xsd:element name="cognom" type="xsd:string"/>
	</xsd:sequence>
</xsd:complexType>
```

Figura 11. Exemple de definició d'elements complexes a XML Schema (reusable)

En la declaració d'elements complexes es possible usar un mecanisme d'herència per reusar o estendre elements definits anteriorment.

En els exemples que hem vist fins ara hem usat l'element xsd:sequence indica que els elements aniuats en ell han d'aparèixer en un ordre determinat. Els esquemes XML ofereixen altres alternatives per indicar com s'han de tractar els elements aniuats en un element complex. Aquestes alternatives son: xsd:all i xsd:choice.

L'indicador xsd:all indica que els elements que conté poden aparèixer en qualsevol ordre però com a màxim una sola vegada. Per exemple:

```
<xsd:element name="persona">
	<xsd:complexType>
		<xsd:all>
			<xsd:element name="nom" type="xsd:string"/>
			<xsd:element name="cognom" type="xsd:string"/>
		</xsd:all>
	</xsd:complexType>
</xsd:element>
```

Figura 12. Exemple de l'indicador xsd:all de XML Schema

L'indicador xsd:choice indica que pot aparèixer només un dels elements que conté. Per exemple:

```
<xsd:element name="persona">
        <xsd:complexType>
                <xsd:choice>
                        <xsd:element name="nom" type="xsd:string"/>
                        <xsd:element name="cognom" type="xsd:string"/>
                </xsd:choice>
        </xsd:complexType>
</xsd:element>
```

Figura 13. Exemple de l'indicador xsd:choice de XML Schema

Per controlar el comportament dels elements també podem usar altres indicadors com maxOccurs, minOccurs que s'usen per indicar el màxim i mínim de vegades que pot aparèixer un element fill de un element complex. L'atribut maxOccurs també pot prendre com a valor unbounded que indica que no existeix cap límit.

```
<xsd:element name="persona">
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element name="nom i cognom" type="xsd:string"/>
                        <xsd:element name="nom dels fills" type="xsd:string" maxOccurs="5"/>
                </xsd:sequence>
        </xsd:complexType>
</xsd:element>
```

Figura 14. Exemple de l'indicador maxOccurs de XML Schema

Per afegir elements no declarats inicialment a l'esquema contem amb el model de contingut any que permet fet coses com aquesta:

```
<xsd:element name="persona">
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element name="nom" type="xsd:string"/>
                        <xsd:element name="cognom" type="xsd:string"/>
                        <xsd:any minOccurs="0"/>
                </xsd:sequence>
        </xsd:complexType>
</xsd:element>
```

Figura 15. Exemple de l'indicador xsd:any de XML Schema

També podem amb anyAttribute estendre el nombre d'atributs d'un element

Per tal de validar un document xml contra el xsd no podem usar la mateixa eina que usàvem per validar els dtd i per tant haurem de trobar una altra eina. Per solucionar-ho podem descarregar des de http://msdn.microsoft.com/xml l'eina MSXML Version 4.0 (que modernitza les capacitats xml d'Internet Explorer) i descarregar des de http://www.topologi.com l'eina Schematron per fer la validació dels documents xml i els xsd. Un cop instal·lat Schematron només l'hi hem d'indicar l'origen del document xml i xsd i al clicar a run s'inicia la validació.

2.5. Fulls d'estil CSS

Com ja hem vist XML el que fa és tractar l'estructura de les dades en forma d'arbre. Per donar format a un document XML podem usar CSS (Cascading Style Sheets). CSS es pot usar tant en HTML com en XML. En el segon cas presenta unes petites variacions. En aquest document no obstant ens limitarem a explicar la versió per XML.

El primer que hem de fer és establir un vincle entre el nostre document xml i el document css. Això ho fem de la següent manera:

<?xml-stylesheet href=”elmeucss.css” type=”text/css”?>
href: ens indica la direcció absoluta o relativa del full d’estil
type: ens indica el llenguatge del full d’estil, generalment text/css o text/xsl

També es possible posar el nostre document css dins del document xml però trobem que aquesta opció es poc útil ja que ens limita la reutilització del codi css.

Amb css el que fem és assignar un format a cada etiqueta. Cal tenir en compte però que les característiques de l‘element més alt es transmeten als element inferiors (d’aquí precisament bé el nom de full d’estil en cascada) així doncs si establim com a tipus de lletra Arial a l’etiqueta arrel i no establim el contrari en cap altra etiqueta tot el document estarà en Arial.

Veiem un petit exemple:

<u>document css</u>

```
llistin { font-family: Arial, sans-serif; font-size:12 pt;}
nom { display: block; font-weight: bold}
telef { display:block; color:red;}
```

Figura 16. Exemple document css

<u>document xml</u>

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<?xml-stylesheet href="elmeucss.css" type="text/css"?>
<llistin>
        <nom> Josep Furriols </nom>
        <telef> 93 123 45 67 </telef>
</llistin>
```

Figura 17. Exemple de document XML amb un css associat

El resultat que mostraria el navegador seria:
Josep Furriols
93 123 45 67

Si mirem el document css línia per línia veiem que en la primera línia s’estableix que el tipus de lletra per tot el document serà Arial o Sans-serif sinó està disponible Arial i el tamany 12 pt. En la segona línia amb display: block simulem un salt de línia i posem el text de l’etiqueta en negreta i en la última etiqueta (telef) posem el seu contingut de color vermell.

Per tant com veiem fer un full d’estils css és molt senzill i per fer-lo només ens cal conèixer els atributs i característiques de CSS. Aquí tenim un petit resum:

Font- family (nom de la font) : ha de contenir un nom de font vàlida
Font-size (tamany de la font): poden ser punts (pt) o pixels (px)
Color (color): nombre del color en anglès o el valor hexadecimal
Blackground-color (color de fons): nombre del color en anglès o el valor hexadecimal
Font-weight (pes de la font): bold, light, normal, etc.
Font-style (estil de la font): italic
Line-height (alçada de la línia): 1 o 1.2
Margin (marge): pot ser en pixels (px) o centímetres (cm)
Padding (marge inferior): pot ser en pixels (px) o centímetres (cm)
Width (amplada): pot ser en pixels (px) o centímetres (cm)
Border-style (estil del borde): per exemple solid
Border-width (amplada del borde): pot ser en pixels (px) o centímetres (cm)

Capítol 3. Els llenguatges de consulta XML

3.1. XPath

XPath és un llenguatge d'especificació d'expressions que permet la cerca i consulta d'informació emmagatzemada dins de documents XML.
Quan validem un document amb un analitzador (parser) es construeix un arbre de nodes que representa, jeràrquicament, el document XML. XPath treballa precisament sobre aquest arbre creat, arbre anàleg al d'un sistema de fitxers de qualsevol sistema operatiu. Com a peculiaritat esmentar que en l'arbre creat es pot donar el fet que hi hagin dos nodes que es diguin igual cosa que en l'arbre del sistema de fitxers no podria passar mai.

En l'arbre creat hi ha diferents tipus de nodes: node arrel, node element, node text, node atribut, node comentari i node d'instruccions de procés.

Node Arrel
No hem de confondre l'element arrel del node arrel d'un document. Si un document té per element d'arrel <llistin>, aleshores llistin és el primer element del node arrel.

Node Element
Tots els elements d'un document xml es converteixen en nodes element dins de l'arbre generat. Tot element té un element pare i pot tenir elements fills. També un element pot tenir nodes text, nodes comentari i nodes d'instruccions de procés. Els nodes element poden tenir un identificador únic que permet referenciar-los de manera directa.

Node Text
Tal i com indica el seu nom són aquell conjunt de cadenes de caràcters del document que no pertanyen a cap etiqueta, sinó que formen part del contingut de l'etiqueta.

Nodes atribut
Els nodes atribut els podem veure com etiquetes afegides al node que els conté. Tot node atribut consta d'un nom i un valor.

Node comentari i node d'instrucció de procés
En l'arbre generat també es generen nodes pels comentaris i les instruccions de procés. Podem accedir-hi usant la propietat string-value

A partir d'aquest document xml d'exemple veurem ara com funciona XPath:

```
<llistin>
        <unapersona>
        </unapersona>
        <persona>
                <nom apodo="furri"> Josep Furriols </nom>
                <telef>93 123 45 67 </telef>
                <codi/>
        </persona>
        <persona>
                <nom apodo="apocapoc">  Marc Poch </nom>
                <telef>93 345 67 78 </telef>
                <codi/>
        </persona>
        <unaaltrapersona>
        </unaaltrapersona>
</llistin>
```

Figura 18. Document XML pels exemples de XPath

La sintaxis bàsica de XPath com he dit abans és similar a la del direccionament d'arxius. Un camí que s'inicia amb el caràcter / representa un camí absolut cap a l'element que volem seleccionar:
/llistin/persona → selecciona tots els elements persona que són fills de l'element llistin

Si el camí s'inicia amb els caràcters // tots els elements que compleixen el criteri són seleccionats independentment de la posició que ocupin a l'arbre:
//persona → selecciona tots els elements persona de l'arbre

L'asterisc selecciona tots els elements ubicats en el camí:
/llistin/persona/* →selecciona els elements nom, telef i codi
/*/*/nom → selecciona tots els elements nom que estan a dos "ancestros"
//* → selecciona tots els elements de l'arbre

Una expressió entre claudators permet precisar l'especificació d'un element:
/llistin/persona[1] → selecciona el primer element persona que apareix a llistin
/llistin/persona[last()] → selecciona l'últim element persona que hi ha a llistin

Els atributs s'especifiquen amb el símbol prefix @
//@apodo → selecciona tots els atributs apodo de l'arbre

Els valors dels atributs poden ser usats com a criteri de selecció:
/llistin/persona/nom[@apodo='furri']→ selecciona l'element nom que té com a valor de l'atribut apodo furri en aquest cas el primer element nom.

Es poden combinar diferents camins amb el separador |:
/llistin/persona/nom|llistin/persona/codi → selecciona els elements nom i codi que són fills de llistin/persona

L'eix child conté els fills del node del context. Es selecciona per defecte i es pot ometre
/child::llistin→ és equivalent a /llistin
/child::llistin/child::persona o /child::llistin/persona → és equivalent a /llistin/persona

L'eix 'descendant' conté els descendents del node del context. Els descendents són els fills del node i els seus descendents. Mai conté atributs ni nodes 'namespace':
/llistin/persona/descendant::*→ selecciona tots els descendents de /llistin/persona

L'eix 'parent' conté el pare del node del context si aquest existeix:
//nom/parent::* → selecciona els elements pares de cada element nom

L'eix 'ancestor' conté els ancestres del node del context que són el pare del node i els seus ancestros:
/llistin/persona/nom/ancestor::*→ selecciona tots els elements llistin i persona

L'eix following-sibling conté tots els nodes germans que segueixen el node del context:
/llistin/unapersona/following-sibling::*→ selecciona tots els elements persona i unaaltrapersona

L'eix preceding-sibling conté tots els elements germans que precedeixen el node del context:
/llistin/unaaltrapersona/following-sibling::*→ selecciona tots els elements persona i unapersona

L'eix following conté tots els nodes del document que es troben després del node del context. No inclou ancestres, descendents, nodes atribut ni nodes namespace
/llistin/persona/following::*→ selecciona tots l'element unaaltrapersona

L'eix preceding conté tots els nodes del document que es troben abans dels node del context. No inclou ancestros, descendents, nodes atribut ni nodes namespace
/llistin/persona/preceding::*→ selecciona l'element unapersona

L'eix descendant-or-self conté el node context i els seus descendents:
/llistin/persona/descendant-or-self::*→ conté els elements persona, nom, telf i codi

L’eix ancestor-or-self conté el node del context i els seus ancestros:
/llistin/persona/nom/ancestor-or-self::*→ conté els elements nom, persona, llistin

La sintaxi abreujada dels eixos és la següent
// → descendant
. → self
.. → parent
@ → attribute
→ child

<u>Funcions XPath</u>
La funció name() ens permet seleccionar elements a partir del seu nom:
//*[name()='telef']→ selecciona tots els elements telef de l’arbre
//*[starts-with(name(),'n')]→ selecciona tots els elements que el seu nom comença per n
//*[contains(name(),'n')]→ selecciona tots els elements que el seu nom contingui una n

La funció node() ens retorna el propi node.
La funció position() ens retorna la posició del node dins el conjunt de nodes germans.
La funció last() Retorna el node que ocupa la última posició en el conjunt de nodes germans.

3.2. XQuery

3.2.1. Introducció

El model de dades XQuery 1.0 ha estat definit conjuntament per el XML Query Working Group (Grup de Treball de XML Query) y el XSL Working Group (Grup de treball XSL) del W3C. Podem trobar molta informació d'XQuery a http://www.w3.org/xml7Query.html

XQuery pot ser usat tan amb document XML sense esquema com amb documents organitzats a partir d'un XML Schema o d'un DTD. És un llenguatge funcional en el qual cada consulta és una expressió per ser avaluada i les expressions poden ser combinades de manera flexible cosa que permet crear noves expressions.

XQuery està definit en funció d'un model de dades, no en funció del text XML. Cada document està representat com un arbre de nodes. Com ja s'ha explicat a la secció anterior, aquests nodes poden ser: document, element, atribut, text, namespace, instrucció de procés o comentari. Cada node té un identificador de node únic que el distingeix dels altres.

En aquest treball farem una petita introducció del llenguatge per tal de poder després, quan treballem amb eXist i Oracle 9i, fer consultes a les bases de dades

Mostrarem també alguns exemples a partir d'aquest document XML anomenat books.xml

```
<bib>
        <book year="1994">
                <title>TCP/IP Illustrated</title>
                <isbn>123-1345-44</isbn>
                <author><last>Stevens</last><first>W.</first></author>
                <publisher>Addison-Wesley</publisher>
                price> 65.95</price>
        </book>
        <book year="1992">
                <title>Advanced Programming in the Unix environment</title>
                <isbn>12-1345-XX</isbn>
                <author><last>Stevens</last><first>W.</first></author>
                <publisher>Addison-Wesley</publisher>
                <price>65.95</price>
        </book>
        <book year="2000">
                <title>Data on the Web</title>
                <isbn>34-13445-00</isbn>
                <author><last>Abiteboul</last><first>Serge</first></author>
                <author><last>Buneman</last><first>Peter</first></author>
                <author><last>Suciu</last><first>Dan</first></author>
                <publisher>Morgan Kaufmann Publishers</publisher>
                <price>39.95</price>
        </book>
        <book year="1999">
                <title>The Economics of Technology and Content for Digital TV</title>
                <isbn>01-00001-01</isbn>
                <editor>
                <last>Gerbarg</last><first>Darcy</first>
                <affiliation>CITI</affiliation>
                </editor>
                <publisher>Kluwer Academic Publishers</publisher>
                <price>129.95</price>
                </book>
                <book year="2002">
                <title>Oracle 9i.Desarrollo Web</title>
```

```
                <isbn>84-415-1406-2</isbn>
                <author><last>Brown</last><first>Bradley</first></author>
                <publisher>Anaya Multimedia</publisher>
                <price>54.00</price>
        </book>
</bib>
```

Figura 19. Document XML per fer consultes amb XQuery

3.2.2. Consultes tipus XPath

XQuery permet fer consultes de manera molt simple usant expressions XPath 1.0 i 2.0. A XQuery les expressions XPath s'usen per localitzar nodes dins les dades XML.

A continuació en veurem algun exemple:
doc("books.xml)/bib/book
Aquesta consulta ens retornaria els elements book que hi ha dins de bib. Una expressió que produiria el mateix resultat seria doc("books.xml)//book

També podem usar condicions booleanes per seleccionar subconjunts d'elements:
doc("books.xml)/bib/book/author[last=Stevens]
seleccionaria els elements author en els quals l'element last fos igual a Stevens.

De fet podem usar qualsevol expressió que sigui vàlida a XPath, i fins i tot suporta namespaces.

3.2.3. Creació de nodes (constructors)

Elements, atributs, nodes de text, instruccions de procés i comentaris poden ser creats usant la mateixa sintaxi que XML. Per crear un llibre faríem:

```
<llibre>
        <titol> prova </titol>
</llibre>
```

Figura 20. Exemple de creació de nodes amb XQuery

Però XQuery també proporciona un constructor explícit de nodes. Quan ho vulguem fer haurem d'usar {}. Per exemple si volguessis usar el constructor explícit en l'exemple anterior faríem:

```
document{
<llibre>
        <titol> prova </titol>
</llibre>
}
```

Figura 21. Exemple de creació de nodes amb XQuery (amb el constructor explícit)

Els constructors es poden combinar amb altres expressions XQuery per generar contingut dinàmicament. Un exemple podria ser:

```
<exemple>
        <prova> {doc("books.xml")//book[1]/title} </prova>
</exemple>
```

Figura 22. Exemple de creació de contingut dinàmicament a XQuery

el resultat d'aquesta consulta seria:

```
<exemple>
        <prova> <title>TCP/IP Illustrated</title> </prova>
</exemple>
```

Figura 23. Resultat de la creació de contingut dinàmicament a XQuery

També es poden usar declaracions d'atribut namespaces en els constructors i crear elements com per exemple element title{"prova"} que crearia un element title amb el contingut prova

3.2.4. Expressions FLWOR (flower expressions)

El principal motor de cerca de XQuery són les expressions FLWOR (**F**or-**L**et-**W**here-**O**rder-**R**eturn). Permet fer cerques de l'estil SELECT-FROM-WHERE-ORDER BY

for: associa una o més variables a expressions creant una sèrie de tuples en les quals a cada tupla se li associa el resultat de l'expressió avaluada

let: associa variables al resultat d'una expressió afegint aquestes associacions a les tuples generades pel for o creant noves tuples sinó hi ha for

where: filtra les tuples i reté només aquelles que compleixen les condicions

order: ordena les tuples segons el criteri donat

return: construeix el resultat per una tupla determinada

Ara veurem alguns exemples:

Si volguessin consultar el títol dels llibres que tenen més de dos autors podríem fer:

```
for $b in doc("books.xml)//book
let $c:=$b//author
where count($c)>2
return $b/title
```

Figura 24. Exemple de consulta FLWOR

i el resultat de la consulta seria:

```
<title>Data on the Web</title>
```

Les regles general per construir aquest tipus d'expressions són:

- for i let poden ser usats diverses vegades en qualsevol ordre
- només pot haver un where
- poden existir varis criteris d'ordenació

És important veure la diferència entre for i let. Ho podem veure clarament en un exemple :

```
for $s in (<one/>,<two/>,<three/>
return <resultat>($s)</resultat>
```

Mostraria com a resultat:

```
<resultat> <one/> </resultat>
<resultat> <two/> </resultat>
<resultat> <tree/> </resultat>
```

Mentre que si en el lloc del for hi hagués let el resultat seria:

```
<resultat>
<one/>
<two/>
<tree/>
</resultat>
```

És a dir mentre que el for itera, el let associa cada variable al resultat sense usar la iteració

És important remarcar que també podem usar expressions FLWOR per, a partir d'un o mes orígens, reestructurar i crear nous documents.

També podem usar expressions FLWOR per fer tot tipus de transformacions en documents XML bé combinant dos documents en un mateix o fins crear nous documents modificant l'ordre dels elements que el composen.

3.2.5. Expressions condicionals

Les expressions condicionals (de tipus if-then-else) s'usen a XQuery de manera similar a altres llenguatges.

L'exemple que mostro a continuació llista el nom dels dos primers autors de cada llibre i la cadena de caràcters "et al." per representar que n'hi ha més de dos.

```
for $b in doc("books.xml")//book
return
<book>
    { $b/title }
    {
      for $a at $i in $b/author
      where $i <= 2
      return <author>{string($a/last), ", ", string($a/first)}</author>
    }
    {
      if (count($b/author) > 2)
      then <author>et al.</author>
      else ()
    }
</book>
```

Figura 25. Exemple del funcionament de les expressions condicionals a XQuery

El resultat d'aquesta consulta seria:

```
<book>
  <title>TCP/IP Illustrated</title>
  <author>Stevens, W.</author>
</book>
<book>
  <title>Advanced Programming in the Unix Environment</title>
  <author>Stevens, W.</author>
</book>
<book>
  <title>Data on the Web</title>
  <author>Abiteboul, Serge</author>
  <author>Buneman, Peter</author>
  <author>et al.</author>
</book>
<book>
  <title>The Economics of Technology and Content for Digital TV</title>
</book>
```

Figura 26. Resultat de la consulta de la figura 24

La seqüència buida, (), pot ser usada per especificar que una clàusula no ha de retornar res.

3.2.6. Quantificadors

Els quantificadors s'usen per determinar si almenys un ítem en una seqüència satisfà una condició (some-in-satisfies) o si cada ítem de una seqüència satisfà una condició (every-in-satisfies).

Veiem algun exemple tot seguit:

```
for $b in doc("books.xml")//book
where some $a in $b/author
satisfies ($a/last="Stevens" and $a/first="W.")
return $b/title
```

Figura 27. Exemple quantificadors some-in-satisfies de XQuery

El quantificador some en la clàusula where comprova si hi ha almenys un autor que satisfà la condició donada dins del parèntesi i el return retorna el resultat que en el cas d'aquesta consulta seria:

```
<title>TCP/IP Illustrated</title>
<title>Advanced Programming in the Unix Environment</title>
```

Un exemple de every-in-satisfies podria ser:

```
for $b in doc("books.xml")//book
where every $a in $b/author
satisfies ($a/last="Stevens" and $a/first="W.")
return $b/title
```

Figura 28. Exemple quantificadors every-in-satisfies de XQuery

Aquesta consulta comprova si cada autor d'un llibre es diu W. Steven. El resultat de la consulta seria:

```
<title>TCP/IP Illustrated</title>
<title>Advanced Programming in the Unix Environment</title>
<title>The Economics of Technology and Content for Digital TV</title>
```

3.2.7. Operadors

Els exemples que hem mostrat fins ara tots contenien operadors tot i que no els havíem explicat detalladament. Com la majoria de llenguatges, XQuery té operadors aritmètics i de comparació i també incorpora operadors de seqüència.

Operadors aritmètics

XQuery suporta els operadors +, -, *, div, idiv i mod.

L'operador div executa la divisió amb qualsevol tipus de dades numèriques mentre que idiv requereix que els arguments siguin enters.

Si un operant de una operació aritmètica és un node s'aplica l'atomització, per exemple 2 + <enter>{2}</enter> retornaria l'enter 4.

Si un operant és una seqüència buida el resultat és una seqüència buida, per exemple 2+()

Si un operant no es de cap tipus de data es fa un cast a double

Operadors de comparació

En primer lloc mostrarem una taula amb els operadors de comparació i després en mostrarem un exemple:

Operador de comparació XQuery	Operador de comparació general
eq	=
ne	!=
lt	<
le	<=
gt	>
ge	>=

Figura 29. Operadors de comparació de XQuery

Un exemple d'us dels operadors on cerquem els llibres que tenen un títol determinat podria ser:

```
for $b in doc("books.xml")//book
where $b/title eq "Data on the Web"
return $b/price
```

En aquest cas la cerca ens retornaria els llibres amb el títol Data on the Web i el seu preu

XQuery proporciona a més a més dos operadors que es poden usar per saber si un element apareix abans que un altra. Són els operadors >> i <<. El operador $a<<$b retorna cert si $a precedeix $b en el document i de manera similar treballa l'altra operador

Operadors de seqüència

XQuery proporciona els operadors **union**, **intersect** i **except** per combinar seqüències de nodes.

L'operador union el podem expressar tant com union com amb una barra vertical (|). Un exemple d'unió podria ser:

```
let $l := distinct-values(doc("books.xml")//(author | editor)/last)
order by $l
return <last>{ $l }</last>
```

Figura 30. Operador union de XQuery

que ens retornaria el contingut de l'element last tant d'autor com d'editor

L'operador d'intersect agafa dos seqüències de nodes com operant i retorna una seqüència que conté tots els nodes que apareixen a ambdós operants mentre que l'operador except agafa dos seqüència de nodes com operats i retorna una seqüència que conté tots els nodes que apareixen en el primer operant però no en el segon.

3.2.8. Funcions

XQuery treballa amb diferents tipus de funcions:

- Funcions built-in
- Funcions definides per l'usuari
- Funcions externes

XQuery implementa moltes funcions built-in típiques de la majoria de llenguatges de programació com min(), max(), coun(), sum() i avg() i també altres funcions molt familiars entre les numèriques podem destacar: round(), floor() i ceiling(), entre les d'string: concat(), string-length(), starts-with(), ends-with(), substring(), upper-case(), lower-case() i casts per diversos tipus de dades.

Però XQuery també implementa funcions pròpies com poden ser distinct-values() o doc() i collection().

A vegades pot ser útil crear funcions pròpies bé perquè una funció s'ha d'usar moltes vegades o bé per aconseguir recursivitat en aquest cas podem definir les nostres pròpies funcions. Un exemple podria ser:

```
define function books-by-author($last, $first) as element()*
{
  for $b in doc("books.xml")/bib/book
  where some $ba in $b/author satisfies ($ba/last=$last and $ba/first=$first)
  order by $b/title
  return $b/title
}
```

Figura 31. Exemple de funció a XQuery

Si el que volem es usar una funció externa (de C o Java per exemple) hem de declarar-la a la capçalera:
define function outtie($v as xs:integer) as xs:integer external
define variable $v as xs:integer external

3.2.9. Datatype expressions

XQuery suporta tots els tipus de XML Schema (primitius i complexes).

Els valors constants poden ser escrits:

- com literals (string, integer, float)
- com functions constructores (true(), date("2001-06-07"))
- com explícits casts (cast as xsd:positiveInteger(47))

Els documents XML Schema poden ser importats a una consulta
Un operador instance of permet runtime validation de qualsevol valor relatiu al tipus de dades o un esquema.

Un operador typeswitch permet branching basat en tipus.

Capítol 4. Bases de dades XML natives

4.1. Introducció

No existeix una definició estàndard de bases de dades nativa però l'organització XML:DB Initiative for XML Databases descriu una base de dades d'aquest tipus com un model lògic per document XML que emmagatzema i recupera documents d'acord amb aquest model.

Una de les diferencies principals que tenen les BD:XML sobre les bases de dades relacionals és que aquestes últimes són centrades en les dades, és a dir que emmagatzemen en els seus camps dades atòmiques, mentre que les BD:XML són centrades en documents ja que el que emmagatzema són documents XML.

Així doncs mentre que en les bases de dades relacionals la unitat bàsica d'emmagatzematge és una tupla (la fila d'una taula), en les BD:XML el document XML és la seva unitat fonamental d'emmagatzematge (lògic).

4.2. Característiques comunes:

Tot i que cada proveïdor té característiques diferents, en general totes les BD:XML tenen aquestes característiques comunes:

Emmagatzematge:
Les bases de dades natives XML emmagatzemen els documents creant models lògics. Aquests models es basen directament en XML o en alguna de les tecnologies relacionades com DOM o Infoset. Aquests models inclouen diferents nivells d'agregació i complexitat així com un suport complet per la gestió de dades semiestructurades. Els models son automàticament mapejats per les bases de dades XML natives al mecanisme d'emmagatzematge corresponent. El mapeig usat per aquestes bases de dades garantirà la correcta utilització del model associat al document original.

Col·leccions:
Aquest tipus de bases de dades treballa sobre col·leccions de documents cosa que permet manipular aquests documents com un sistema. Això és similar al concepte relacional d'una taula tot i que divergeix en el sentit que no requereix que la col·lecció tingui un esquema associat. Això significa que es pot emmagatzemar qualsevol document XML en una col·lecció sense necessitat de cap esquema i tot i això es poden fer consultes a través de tots els documents de la mateixa col·lecció. Aquesta propietat s'anomena esquema-independent.

Cerques
Actualment, XPath és el llenguatge de cerques més estès implementat pel diferents sistemes de BD:XML. XPath, però, no va ser dissenyat com a llenguatge de consultes de bases de dades i té diverses limitacions . Algunes d'elles s'han solucionat ampliant el llenguatge per exemple per permetre consultes a través de col·leccions XML. No obstant les carències més importants com agrupar, ordenar, fer "cross document join" no s'han pogut solucionar i per fer-ho es sol recórrer al llenguatge XSLT o si la BD:XML implementa XQuery llavors permet fer el tipus de cerques explicades a la secció anterior..

Processament de les dades
El processament de dades hauria de ser una cosa realment senzilla però la realitat és que no és així. Moltes bases de dades necessiten recuperar tot el document XML per actualitzar-lo amb qualsevol API i posteriorment tornar a emmagatzemar el document en el repositori. Això és degut a que encara no existeix un llenguatge estàndard que permeti fer l'actualització de manera senzilla i ràpida. Actualment, s'està desenvolupant un llenguatge (XUpdate) per fer aquesta tasca però encara no és un estàndard i moltes BD:XML encara no el suporten.

En aquest treball presentarem a fonts eXist com a exemple d'una BD:XML potent i també algunes nocions sobre XÍndice tot i que no i entrarem tant en detall.

4.3. eXist

4.3.1. Descripció

Figura 32. Logo eXist

eXist és un sistema de BD:XML open source, desenvolupat íntegrament en Java.

El sistema d'emmagatzematge es fa a través de col·leccions jeràrquiques. Usant una versió estesa de XPath els usuaris poden consultar una part de la col·lecció jeràrquica o tot els documents continguts a la base de dades. Un millorat esquema d'indexació proporciona una ràpida identificació de relacions estructurals entre nodes del tipus pare-fill, antepassat-descents o previous-/next-sibling.

La base de dades es ideal tant per petites com grans col·leccions de documents xml que s'actualitzen de tant en tant. eXist proporciona extensions a XPath estàndard i permet fer cerques per paraula clau. Pel programadors permet accés a través d'HTTP, XML-RPC, SOAP i WebDAV.

4.3.2. Indexat i emmagatzematge

En aquesta secció farem una aproximació a la indexació i arquitectura d'emmagatzematge que implementar eXist

Introducció
Els llenguatges de consulta XML com XQuery i XPath usen expressions per navegar a través de les estructures lògiques i jeràrquiques dels documents XML que es modelen com un arbre. Una expressió XPath localitza nodes dins un arbre per exemple book//section/title selecciona tots els elements title que són fills dels elements secció que tenen com a ancestre un element anomenat book.

D'acord amb XPath 2.0 el resultat d'una expressió de camí és una seqüència de nodes diferents en un document ordenat. La seqüència de nodes pot ser filtrada per expressions de predicat. Una expressió de predicat va tancada entre "corchetes". Aquells nodes que no compleixen el predicat són descartats de la seqüència per exemple per trobar aquelles seccions el títol de les quals es XQuery podríem usar la següent expressió: book//section[contains(title, 'XQuery')]

La subexpressió de predicat especifica una selecció basada en valor mentre que la subexpressió book//section denota una selecció estructural. Les seleccions basades en valors poden ser especificades en noms d'element, noms d'atribut/valor o strings de text continguts en un element. Les seleccions estructurals són basades en les relacions estructurals entre nodes tal com ancestor-descendant or parent-child

Bastants sistemes diferents d'implementació de llenguatges de consulta estan disponibles actualment pels desenvolupadors. De totes maneres la majoria d'implementacions Open Source es basen en arbres convencionals top-down o bottom-up per avaluar les expressions path

Tot i les millores de disseny en aquest tipus d'arbres transversals aquests són molt ineficients per grans col·leccions de documents . Per exemple si considerem una expressió XPath que selecciona els títols de totes les figures en una col·lecció de llibres: /book//figure/title. En un

arbre transversal top-down convencional, el processador de consultes ha de seguir cada camí que comença a book per verificar potencials figures descendents perquè no hi ha cap manera de determinar les possibles localitzacions dels descendents figure per endavant. Això implica que un gran nombre de nodes que no són "figure" han de ser accedits per verificar si el node és un element figure

Així doncs les estructures d'índex són necessàries per fer consultes eficients en grans col·leccions. El esquema d'indexació ha de subministrar significat per processar tant seleccions basades en valor com en estructura. Mentre les seleccions basades en valor són generalment correctament suportades pels esquemes extensos d'indexació tradicionals tal com B+-tree, en les seleccions estructurals cal anar més en compte. Per accelerar el procés d'expressions path basades en relacions estructurals els esquemes d'indexació han de suportar una ràpida identificació de les relacions entre nodes tal com ancestor-descendant o parent-child.

Esquema de numeració d'eXist

S'han proposat diversos esquemes de numeració pels documents XML per solucionar aquests problemes. Un esquema de numeració assigna un únic identificador a cada node en l'arbre lògic del document XML. Els identificadors generats són aleshores usats com a referència al node actual. Un esquema de numeració ha de proporcionar mecanismes per determinar ràpidament la relació estructural entre un parell de nodes i identificar totes les aparicions de tal relació en un document o col·lecció de documents.

S'han proposat diversos esquemes de numeració. eXist, però, es basa en l'esquema en sistema de Lee et al.

El sistema de Lee et al. proposa un esquema de numeració el qual modela el document arbre com un arbre complet k-ary on k és igual al màxim nombre de nodes fill de l'element en el document. Un únic node identificador és assignat a cada node per travessar l'arbre en ordre.

Els identificadors únics generats per aquest esquema de numeració té alguns propietats importants: per un identificador donat pot ser fàcilment determinat l'identificador del node del seu pare, germans o fills. Per obtenir-lo només cal aplicar la fórmula següent:

$$Parent_i = (i-2/k)+1$$

No obstant, la completesa de l'arbre imposa una restricció en el tamany màxim del document a ser indexat usant aquest esquema de numeració ja que el nombre d'identificadors necessaris pot créixer molt ràpidament.

Per solucionar el problema de les limitacions del tamany del document eXist decideix perdre parcialment la restricció de completesa a favor de un esquema alternatiu. El document ja no serà vist com un arbre complet k-ary. Enlloc d'això el nombre de fills que un node pot tenir es recomputat per cada nivell del arbre de la següent manera: dos nodes x i y de un arbre tamany(x)=tamany(y) si nivell(x)=nivell(y) on tamany(n) és el nombre de fills de un node n i nivell(m) es la longitud del camí des del node arrel fins m. La informació addicional sobre el nombre de fills que un node pot tenir en cada nivell de l'arbre es guardada amb el document en un simple array.

Típicament els documents tenen un gran nombre de nodes i menys nivells i hi ha menys elements als nivells superiors de la jerarquia. La limitació de tamany ha crescut considerablement per permetre indexar documents molt més llargs. Si ho comparem amb l'esquema de numeració original s'han de usar menys identificadors.

També si inserim un node a un nivell més profund no té efecte sobre els identificadors assignats a nodes dels nivell superiors. És també possible deixar uns pocs identificadors entre nodes existents per evitar una freqüent reordenació dels nodes identificadors en posteriors actualitzacions del document. No obstant això eXist no proporciona encara un sistema avançat de actualització com el definit per l'estàndard XUpdate. Els documents poden ser actualitzats de manera global però no es possible manipular nodes individuals. Aquesta millora esta

prevista per futures versions de eXist. De totes maneres eXist es el que millor s'ajusta per documents més o menys estàtics que rarament són actualitzats.

Les modificacions fetes per eXist sobre el sistema original no afecten a la propietat general de la assignació d'identificadors. Per un identificador únic donat podem encara esbrinar els nodes pare, germans o fill usant la informació addicional sobre el nombre de fills de cada node que pot tenir cada nivell de l'arbre.

eXist proporciona una implementació completa del llenguatge de consulta XPath. Suporta tots els axis del llenguatge. Per exemple si considerem l'expressió que selecciona els element pare de tots els elements paràgraf que contenen la paraula XML: //para[contains(., 'XML')]/..

Usant l'esquema de numeració de eXist podem fàcilment computar l'identificador del node pare per cada node donat per avaluar l'expressió. També podem computar els identificadors dels nodes germans i fills. D'aquesta manera tots axis de la navegació poden ser implementats en el nostre esquema.

Això redueix significativament el tamany d'emmagatzematge de un node simple en el magatzem xml: no es necessari guardar el links als nodes i atributs pare fills i germans. Per accedir al parent del node simplement calculem el seu identificador únic i busquem l'índex. Des del moment que no es necessari guardar links entre nodes un node element ocuparà no més que 4 a 8 bytes en el magatzem xml eXist.

A més a més amb el sistema d'indexació d'eXist cap node en un document xml pot servir com punt d'inici per una expressió XPath.

<u>Índexs i organització de les dades</u>
eXist usa quatre fitxers índex en el cor de la base de dades nativa:
-collections.dbx administra la col·lecció jeràrquicament
-dom.dbx recopila nodes en un arxiu paginat i "associates unique node identifiers to the actual nodes"
-elements.dbx indexa element i atributs
-words.dbx manté un camí a les aparicions de paraules i es usat per les extensions de cerca fulltext.

Tots els índex estan basats en arbres B+-. Un important punt a tenir en compte es que els índex per element atributs i paraules clau són organitzats per col·lecció i no per document, és a dir, un element en una col·lecció serà guardat com un simple índex d'entrada en el índex d'elements. Això manté el nombre de pàgines de l'arbre B+- petit i proporciona un millor funcionament per les consultes en una col·lecció sencera.

El fitxer índex collections.dbx administra la col·lecció jeràrquicament i mapeja noms de col·lecció a objectes de col·lecció. Degut a consideracions de funcionament, les descripcions de documents són sempre guardades amb el objecte col·lecció al qual pertany. Un únic identificador és assignat a cada col·lecció i document durant la indexació.

El magatzem de dades (dom.dbx) representa el component central de l'arquitectura d'emmagatzematge de eXist. Consisteix en un fitxer paginat en el qual tot node document és guardat d'acord amb el model d'objectes (DOM). El magatzem de dades és recolzat per un la B +-Tree multi-arrel en el mateix arxiu per a associar els identificadors de node únic d'alt nivell en un document donat a la direcció d'emmagatzematge del node a la secció de dades.

Només els elements del nivell superior són indexats per un arbre B+-. Els atributs, nodes text I elements a nivells inferiors del node document són solament escrits a les pagines de dades sense afegir una clau en l'arbre B+-. L'accés a aquesta classe de nodes es proporcionada creuant el més proper ancestro disponible trobat a l'arbre. De totes maneres els accessos directes a aquests nodes són molt poc freqüents. La màquina de consulta processa la majoria de tipus de expressions XPath sense accedir a dom.dbx

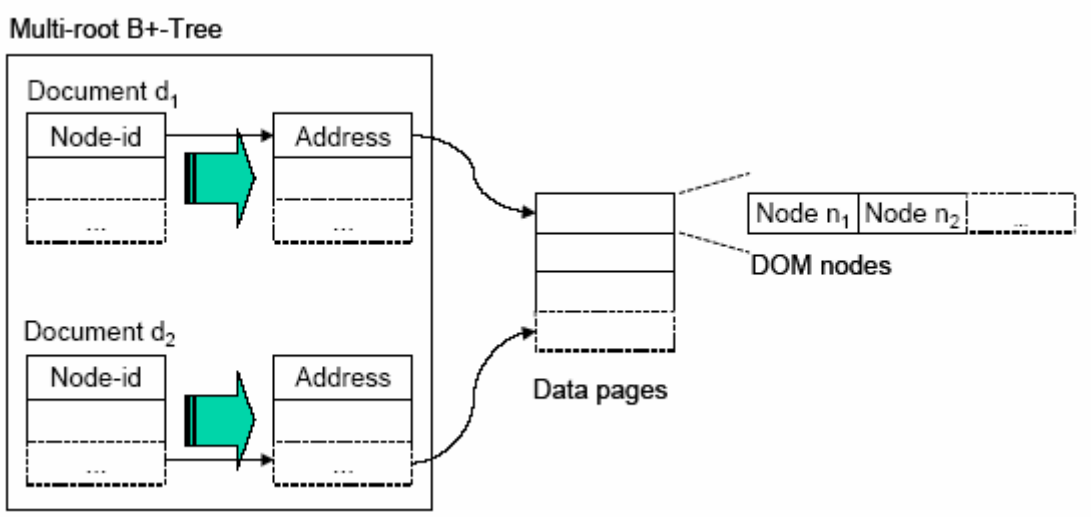


Figura 33. Multi-root B+- Tree

Els elements i noms d'atributs són mapejats amb identificadors únics de nodes en l'arxiu elements.dbx. Cada entrada en el índex consisteix en una clau <collectionid,name-id> i un valor d'array que conté una llista ordenador de identificadors de document I node els quals corresponen a elements i atributs que fan concordar el nom qualificat en una clau. Per trobar tots els capítols en una col·lecció de llibres, l'enginy de consulta requereix un índex de cerca per recuperar la llista completa de nodes identificadors apuntant als elements capítol.

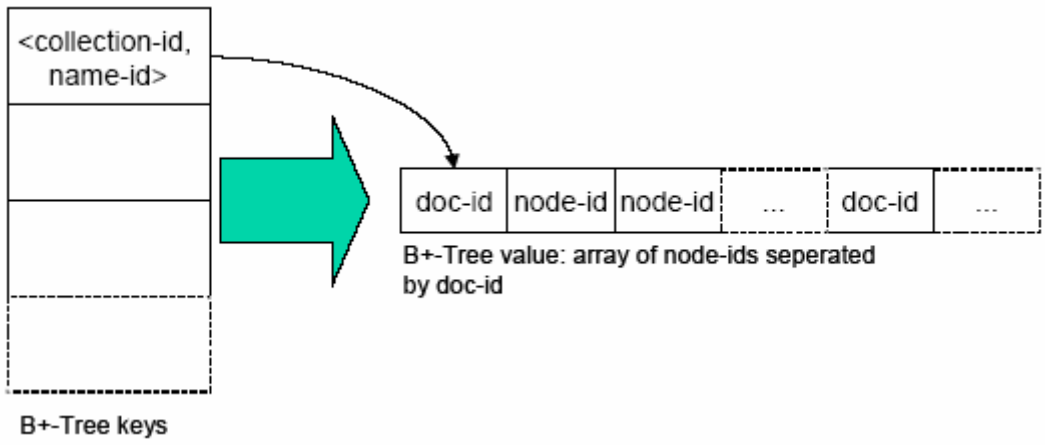


Figura 34. B+- Tree keys

Finalment l'arxiu word.dbx correspon a un índex invertit com el que podem trobar en molts sistemes de representació de dades. El índex invertit representa un estructura de dades comuns i es típicament usada per associar una paraula o frase amb el conjunt de document en el qual ha estat trobat i la posició exacte on ha ocorregut. Els índex invertits d'eXist difereixen dels tradicions en que aquest guarden la posició de la paraula mentre que el sistema d'eXist usa un node únic identificador per seguir les aparicions de paraules. En l'arxiu words.dbx les paraules clau extretes són mapejades a una llista ordenada de document i unique node identifiers. El fitxers segueix la mateixa estructura que elements.dbx Cada entrada en la llista de valors apunta a un node text o atribut on la paraula clan apareix.

4.3.3. Implementació del llenguatge de consultes

eXist conté actualment un processador de llenguatge de consultes XPath així com part de l'estàndard XQuery, ara mateix en una fase experimental. El processador d'eXist XPath implementa la major part del estàndard XPath 1.. No obstant les funcionalitats existents

cobreixen la majoria de les necessitats d'expressions XPath. A més a més, algunes extensions de l'estàndard XPath estan disponibles.

Algoritme d'unió path

També s'han proposat i testejat experimentalment algorismes d'unió path (path join algorithms) basats en el esquema de numeració en preordre d'XISS. Els tres algoritmes són assignats a diferents tipus de subexpressions: Element-Attribute Join, Element- Element Join i Kleene-Closure Algorithm.

El processador de consultes d'eXist descomposarà l'expressió de camí donada en una cadena de comandes bàsiques.
Considerant una expressió XPath com: /PLAY//SPEECH[SPEAKER='HAMLET']
Cada PLAY està dividit en seccions ACT, SCENE I SPEECH. Un element SPEECH inclou elements SPEAKER i LINE . L'expressió XPath anterior es talla en les següents subexpressions:
PLAY//SPEECH →SPEECH[SPEAKER] ←SPEAKER='HAMLET'

La posició exacta dels elements PLAY, SPEECH i SPEAKER es proporcionada pel fitxer d'índexs anomenat *element.dbx*. Per processar la primera subexpressió la maquina de consulta carrega els elements arrel(PLAY) per tots els document en el sistema de document d'entrada. En segon lloc el conjunt d'element SPEECH es recuperat pel document d'entrada fent una cerca de elements.dbx. Ara ja tenim dos conjunts de nodes que contenen els nodes ancestros i descendents per cada document. Cada conjunt de nodes consisteix de un parell <document-id, node-id> ordenat per identificador de document i un identificador de node únic. Els conjunts de nodes estan implementats usant arrays de Java.

Per trobar tots els nodes SPEECH que són descendents dels nodes del conjunt de nodes PLAY s'aplica un algorisme d'unió path ancestor-descendant.

El conjunt de nodes generat per l'algorisme es convertirà en el conjunt de nodes per la següent subexpressió de la cadena. D'aquesta manera el conjunt de nodes resultants per l'expressió PLAY//SPEECH es converteix en el conjunt de nodes ancestre per l'expressió SPEECH[SPEAKER], mentre que els resultats generats per l'avaluació de l'expressió de predicat SPEAKER="HAMLET" es converteix en el conjunt de nodes descendents.

Per avaluar les subexpressions PLAY//SPEECH i SPEECH[SPEAKER], eXist no necessita accedir als nodes DOM actuals en el magatzem XML. Les dues expressions són completament processades en base als identificadors de node únics proporcionats en el fitxer d'índexs. De manera addicional, el algorisme determina les relacions ancestor-descendant per tots els nodes candidats en tots els document en un sol pas.

Per processar en igualtat l'operador en la subexpressió de predicat la màquina de consultes ha de recuperar els nodes DOM actuals per determinar el seu valor I comparar-lo a l'argument string. Des d'aquest moment el valor d'un node pot ser distribuït a molts nodes descents, la màquina té de fer un arbre transversal començant pel node context de la subexpressió (SPEAKER). Això podria ser evitat afegint una altra estructura pels valors dels nodes.

Extensions del llenguatge de consultes

La especificació XPath només defineix unes poques funcions per cercar a partir d'un string dins el contingut d'un node. Aquest és un punt dèbil de XPath si volem buscar a través de document que continguin grans seccions de text. Per molt tipus de documents les funcions estàndard no ens proporcionaran un resultat satisfactori.

eXist ofereix dos operadors addicionals i diverses funcions d'extensió per proporcionar un millor accés. Per exemple per seleccionar la escena en la caverna de Macbeth de Shakespeare:
//SCENE[SPEECH[SPEAKER &= 'witch' and near(LINE, 'fenny snake']]
&= és un operador de cerca de text especial. . Això seleccionarà els nodes de context que continguin tots el termes , separats per espai a la dreta de l'argument. Per trobar nodes que continguin algun dels termes el operador usat és |=. L'ordre dels termes no és important. Ambdós operadors suporten comodins en els termes de cerca. Per imposar un ordre en la

cerca de termes la funció near selecciona els nodes pels quals cada un dels termes del segon argument apareix prop de l'altra en el valor de node i en l'ordre correcte. Per acomodar el model més complex la sintaxi de l'expressió regular serà recolzada per funcions addicionals.

Tota extensió de cerca fulltext usa el arxiu d'índex invertit words.dbx el qual mapeja les paraules clau extretes a una llista ordenada de documents i identificadors de node únics. D'aquesta manera mentre el operador d'igualtat a l'igual que les funcions XPath requereixen eXist per completar l'escaneix sobre els continguts de cada node en el conjunt de nodes del context, les extensions de cerca fulltext confien plenament en la informació guardada en l'índex.

4.3.4. Consultes a eXist

Ara que ja em vist el funcionament de XPath, XQuery i em explicat com s'emmagatzema la informació a eXist anem a explicar-ne el seu funcionament i a veure alguns exemples de com es realitzen les consultes.

Per fer les consultes tant a eXist com a Oracle 9i usarem els arxius books.xml, books2.xml i el dtd books.dtd que exposo a continuació:

books.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE bib SYSTEM "books.dtd">
<bib>
	<book year="1994">
		<title>TCP/IP Illustrated</title>
		<isbn>123-1345-44</isbn>
		<author><last>Stevens</last><first>W.</first></author>
		<publisher>Addison-Wesley</publisher>
		<price> 65.95</price>
	</book>
	<book year="1992">
		<title>Advanced Programming in the Unix environment</title>
		<isbn>12-1345-XX</isbn>
		<author><last>Stevens</last><first>W.</first></author>
		<publisher>Addison-Wesley</publisher>
		<price>65.95</price>
	</book>
	<book year="2000">
		<title>Data on the Web</title>
		<isbn>34-13445-00</isbn>
		<author><last>Abiteboul</last><first>Serge</first></author>
		<author><last>Buneman</last><first>Peter</first></author>
		<author><last>Suciu</last><first>Dan</first></author>
		<publisher>Morgan Kaufmann Publishers</publisher>
		<price>39.95</price>
	</book>
	<book year="1999">
		<title>The Economics of Technology and Content for Digital TV</title>
		<isbn>01-00001-01</isbn>
		<editor>
		<last>Gerbarg</last><first>Darcy</first>
		<affiliation>CITI</affiliation>
		</editor>
		<publisher>Kluwer Academic Publishers</publisher>
		<price>129.95</price>
		</book>
	<book year="2002">
		<title>Oracle 9i.Desarrollo Web</title>
```

```
                <isbn>84-415-1406-2</isbn>
                <author><last>Brown</last><first>Bradley</first></author>
                <publisher>Anaya Multimedia</publisher>
                <price>54.00</price>
        </book>
</bib>
```

Figura 35. Document books.xml

<u>books2.xml</u>

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE bib SYSTEM "books.dtd">
<bib>
        <book year="2222">
                <title>Prova Josep</title>
                <isbn>000-0000-00</isbn>
                <author><last>Josep</last> <first>F.</first> </author>
                <publisher>JFP</publisher>
                 <price>222,22</price>
        </book>
</bib>
```

Figura 36. Document books2.xml

<u>books.dtd</u>

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!ELEMENT bib (book* )>
<!ELEMENT book (title, isbn, (author+ | editor+ ), publisher, price )>
<!ATTLIST book year CDATA #REQUIRED >
<!ELEMENT isbn (#PCDATA )>
<!ELEMENT author (last, first )>
<!ELEMENT editor (last, first, affiliation )>
<!ELEMENT title (#PCDATA )>
<!ELEMENT last (#PCDATA )>
<!ELEMENT first (#PCDATA )>
<!ELEMENT affiliation (#PCDATA )>
<!ELEMENT publisher (#PCDATA )>
<!ELEMENT price (#PCDATA )>
```

Figura 37. Document books.dtd

Primer de tot el que hem de fer es arrancar el programa. Hem de tenir en compte, preo, que si hem instal·lat també Oracle al mateix pc on volem fer anar eXist hem d'aturar primer alguns serveis d'Oracle que s'inicien automàticament al iniciar el pc encara que no haguem engegat Oracle 9i. Per tal de d'aturar aquests serveis hem d'anar a Panel de Control / Herramientas Administrativas / Servicios i aturar tots els serveis que comencin amb el nom Oracle

Un cop feta aquest petit aclariment i un cop ja hem pogut arrancar eXist el primer que hem de fer es crear un projecte a on emmagatzemarem tots els documents xml de la nostra col·lecció (en el nostre cas hem creat una collection anomenada TFC). Podem crear-la des de la opció File/ Create Collection.

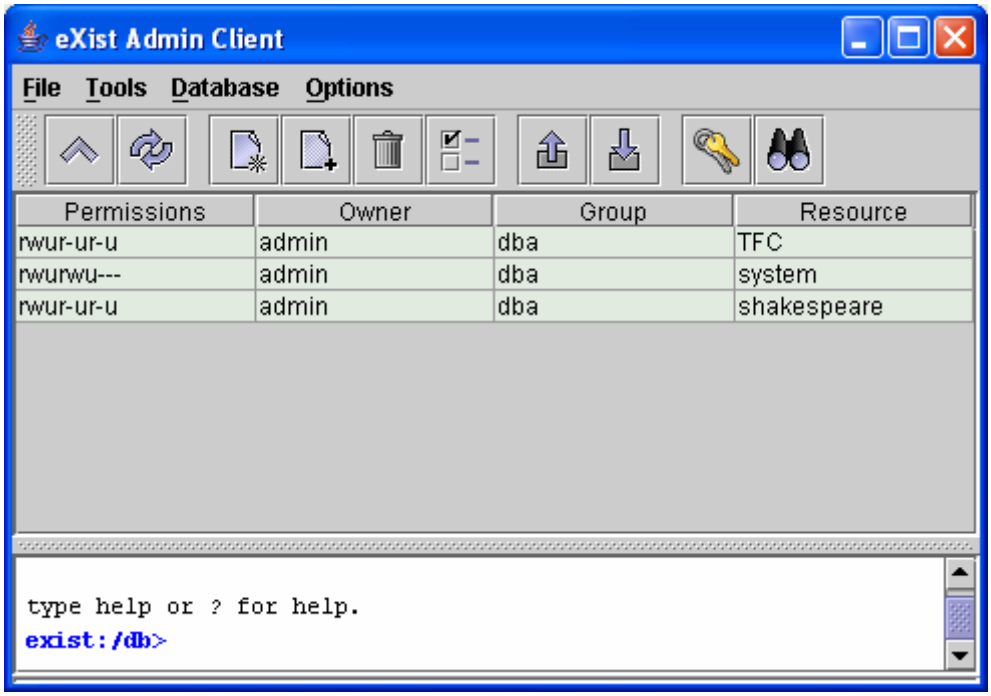


Figura 38. Captura pantalla eXist (on es pot visualitzar la creació de la col·lecció TFC)

Un creada la nostra col·lecció li afegim els documents xml que ens facin falta usant la opció File/store files/directories i una vegada fet això ja podem començar a fer cerques de tipus XPath i XQuery si cliquem al botó de cerques (els binocles que veiem a la pantalla anterior).

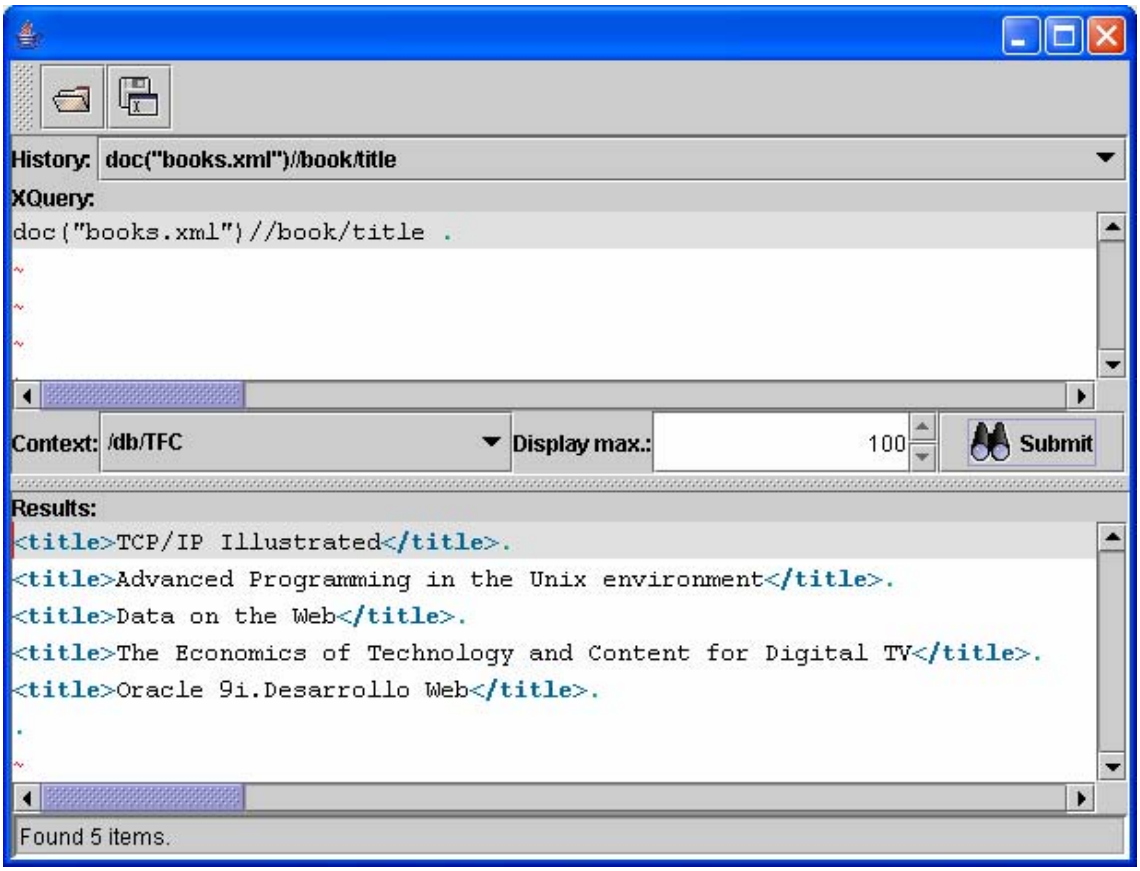


Figura 39. Captura de la pantalla de consultes eXist (es visualitza el resultat d'una consulta)

A la pantalla que veiem a sobre podem veure una consulta senzilla que ens retorna el nom dels llibres del document books.xml. També hauríem pogut obtenir el mateix resultat fem la consulta de la manera següent: doc("books.xml")/bib/book/title

Una mètode molt interessant que ofereix eXist és document que ens permet fer la cerca a tota la base de dades independentment del document xml a on estigui. Per exemple la mateixa consulta d'abans amb el mètode document enlloc de doc ens retornaria:

```
<title>TCP/IP Illustrated</title>
<title>Advanced Programming in the Unix environment</title>
<title>Data on the Web</title>
<title>The Economics of Technology and Content for Digital TV</title>
<title>Oracle 9i.Desarrollo Web</title>
<title>Prova Josep</title>
```

Figura 40. Resulta de una consulta feta a eXist amb document

ja que també s'ha consultat books2.xml i s'ha obtingut el títol de llibre que hi ha.

En certes ocasions pot ser útil eliminar duplicats per exemple quan volem obtenir una llista de tots els autors sense duplicats. Això ho podríem fer de la següent manera:

```
distinct-values (doc("books.xml")//author/last)&(doc("books.xml")//author/first)
```

Una cerca ja més interessant seria la següent:

```
for $b in document()//book
where $b/title="Prova Josep"
return $b//book
```

Figura 41. Exemple de consulta a eXist (cerca d'un llibre en concret a la base de dades)

Aquesta cerca busca a la base de dades si hi ha algun llibre amb el títol Prova Josep i en cas de ser-hi mostra totes les seves dades. El resultat d'aquesta consulta seria:

```
<book year="2222">
   <title>Prova Josep</title>
   <isbn>000-0000-00</isbn>
   <author>
      <last>Josep</last>
      <first>F.</first>
   </author>
   <publisher>JFP</publisher>
   <price>222,22</price>
</book>
```

També pot ser interessant en un moment donat poder tenir els resultats amb un ordre determinat. A continuació Veurem un exemple de com extreure el títol dels llibres per ordre:

```
for $b in document()//book
order by($b/title) ascending
return $b//book/title
```

Figura 42. Exemple de consulta a eXist (obtenir resultats en un ordre determinat)

i el resultat obtingut serà:

```
<title>Advanced Programming in the Unix environment</title>
<title>Data on the Web</title>
<title>Oracle 9i.Desarrollo Web</title>
```

```
<title>Prova Josep</title>
<title>TCP/IP Illustrated</title>
<title>The Economics of Technology and Content for Digital TV</title>
```

Una altra característica interessant de XQuery és que permet crear nous documents a partir de les dades existents. Per exemple si volguéssim crear un nou document xml amb el nom dels autors dels llibres del document books.xml podríem fer el següent:

```
for $b in doc(books.xml)//author
return
<autor>
{$b/last ,$b/first}
</autor>
```

Figura 43. Exemple de consulta a eXist (nou document a partir de les dades consultades)

i obtindríem el següent resultat:

```
<autor>
   <last>Stevens</last>
   <first>W.</first>
</autor>
<autor>
   <last>Stevens</last>
   <first>W.</first>
</autor>
<autor>
   <last>Abiteboul</last>
   <first>Serge</first>
</autor>
<autor>
   <last>Buneman</last>
   <first>Peter</first>
</autor>
<autor>
   <last>Suciu</last>
   <first>Dan</first>
</autor>
<autor>
   <last>Brown</last>
   <first>Bradley</first>
</autor>
```

Aquesta mateixa consulta la podem millorar per exemple ordenant-la en ordre ascendent i agafant només els autors que el seu cognom sigui a partir de la lletra J

```
for $b in doc("books.xml")//author
where $b/last>"J"
order by($b/last) ascending
return
<autor>
{$b/last ,$b/first}
</autor>
```

Figura 44. Exemple de consulta a eXist (obtenir dades ordenades a partir de certa lletra)

i aleshores obtindríem el següent resultat:

```
<autor>
   <last>Stevens</last>
```

```
    <first>W.</first>
</autor>
<autor>
    <last>Stevens</last>
    <first>W.</first>
</autor>
<autor>
    <last>Suciu</last>
    <first>Dan</first>
</autor>
```

4.4. XIndice

Figura 45. Logo XIndice

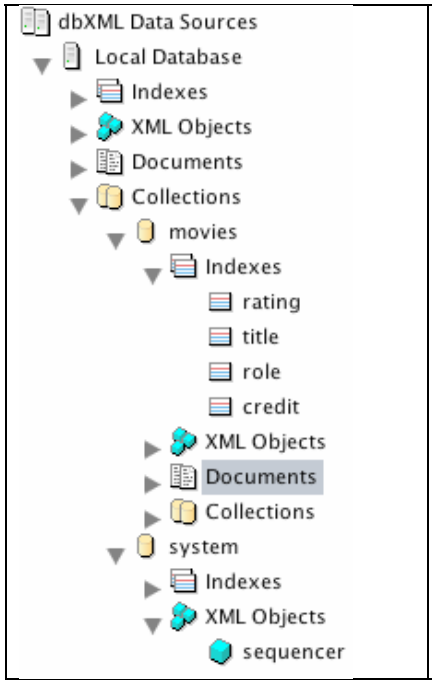
Degut a problemes de temps només farem una petita pinzellada de XIndice per veure'n les característiques generals sense entrar gaire en detall.

XÍndice (inicialment anomenada dbXML Core) és a l'igual que eXist una base de dades XML nativa. Va ser la primera BD XML nativa open source i és una de les BD XML natives més usades.

Es caracteritza per:

- Ser més ràpida que altres BD XML natives
- No necessita mapejar a relacional
- Guarda document "in tokenized form"
- Usa un esquema d'indexació específic XML
- Suporta consultes XML optimitzades

Es basa en un sistema de col·leccions de documents.



- L'objecte del nivell superior és una base de dades (Database)
- Database és una col·lecció especial
- Database manté taules de símbols
- "SystemCollection" guarda metadades DB
- La unitat bàsica de dades és un Document
- Un conjunt de document són una col·lecció
- Una col·lecció pot contenir col·leccions
- S'ha de pensar en ella com un sistema de fitxers per XML
- Les col·leccions poden ser indexades
- Les col·leccions poden mantenir objectes XML
- Els objectes XML són com "Procs" guardats

Figura 46. Col·leccions XIndice

XIndice permet fer consultes usant el llenguatge XPath. I a més a més:

- Pot seleccionar nodes de manera múltiple o individual
- Pot utilitzar i retornar els resultats de funcions
- Extend XPath a col·leccions

XIndice també treballa amb l'XUpdate especificat per XML:DB Initiative.

- És una sintaxi XML per modificacions de lots contra conjunts de documents XML

- Diverses operacions poden ser executades contra una col·lecció a la vegada
- Modifica les capacitats de filtrar de XPath permetent usar selectors
- Els selectors operen contra col·leccions
- Tot i que té limitacions ja el podem usar

XIndice treballa amb la API XML:DB especificada per XML:DB Initiative i estandarditza la API per XML Database acces. Té un llenguatge i protocol independent (OS) similar en alguns aspectes a JDBC i ODBC. Suporta diversos mètodes d'accés XML i és senzill d'instal·lar. A més a més ofereix una versió comercial i una altra de gratuïta

L'arquitectura oberta d'XIndice proporciona molts mètodes per la connectivitat del client. Suporta CORBA, XML-RPC i recuperació directa dels documents usant un servidor HTTP intern. També incorpora eina de línia de comandes per fer tasques administratives i una interfície gràfica que actualment està en desenvolupament

Capítol 5. Bases de dades relacionals i XML

Figura 47. Logo Oracle

5.1. Introducció

Tot i que inicialment Oracle no formava part dels grups de treball de XML del W3C, la gran acceptació que ha tingut XML ha fet que s'hagi hagut d'implicar en els organismes de normalització així com en les tasques per impulsar l'estàndard.

Oracle 9i conté un paquet d'eines de desenvolupament XML conegut com XDK (XML Developer's Kit). El XDK usa aplicacions Java, aplicacions desenvolupades en C i mòduls desenvolupats en PL/SQL.

L'XDK suporta els següents estàndards:
• XML 1.0
• XML Schema 1.0
• XML Namespace 1.0
• XSLT 1.0
• XPath 1.0
• DOM 1.0 y DOM 2.0
• SAX 1.0 y SAX 2.0

Mentre que el XDK per Java compte amb els següents components:
• XML Parser for Java V2
• XML Class Generator for Java
• XML SQL Utility
• XML Schema Processor for Java
• XSQL Servlet

5.1.1. Analitzador XML de Oracle

De la mateixa manera que quan treballem directament amb documents XML tenim una eina (parser) que s'encarrega de comprovar si el document creat és vàlid o no i compleix les regles establertes en un determinat DTD, Oracle també ha de tenir una eina per fer-ho.

Aquestes operacions les porta a terme l'analitzador XML d'Oracle. Té la característica de poder usar-se tant des de fora de la base de dades (usant Java) com des de dins usant la Java Virtual Machine d'Oracle 9i

5.1.2. El generador de classes XML de Oracle

Un altra dels components d'Oracle és el generador de classes que permet crear un conjunt d'arxius Java a partir d'un DTD determinat. Els arxius Java obtinguts permeten construir, validar i imprimir documents XML que s'ajustin al DTD donat.

Per poder usar el generador de classes hem de disposar d'un arxiu XML que inclogui al principi del document la declaració del DTD que usarà.

La importància del generador de classes XML radica en què l'entrada de l'aplicació pot ser a través de fonts tant diverses com un formulari web, un analitzador SAX, una aplicació JDBC, etc… i a partir d'aquí generar un document XML ben format i en consonància amb l'arxiu DTD indicat.

5.1.3. La SQL XML Utility de Oracle

El paquet SQL XML Utility són un conjunt de classes Java que permeten:

- Generar documents XML a partir d'una consulta SQL o generar un document XML de les dades obtingudes a través de JDBC
- Recuperar dades d'un document XML emmagatzemats en una taula de la base de dades

La SQL XML Utility de Oracle consta de cinc components, quatre d'ells classes Java, OracleXML, OracleXMLStore, OracleXMLSave y OracleXMLQuery i el cinquè és un paquet per PL/SQL similar al OracleXMLStore anomenat xmlgen.

El mode d'execució de SQL XML Utility en una base de funciona de la següent manera:

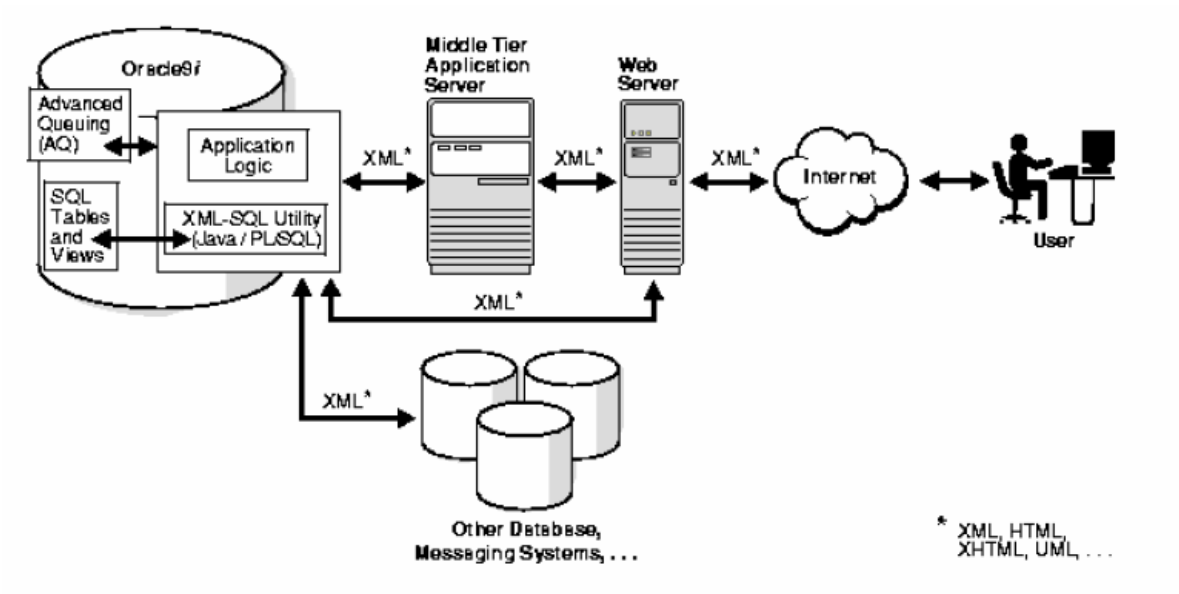


Figura 48. Mode execució de SQL XML Utility

El component OracleXML a partir dels dos components getXML i putXML que conté permet a través d'ordres introduïdes per línia de comandes extreure o inserir informació d'una taula en format XML o bé inserir dades a partir d'informació XML.
c.\oracle\ora90\bin\>java OracleXML .user xml/xml "select * from emp"

OracleXMLQuery de manera molt similar a getXML d'OracleXML és una API que permet generar documents XML a partir d'una consulta o dels resultats procedents de JDBC. La diferència principal amb OracleXML és que permet incloure's en aplicacions Java a diferència de getXML que només pot executar-se a través de la línia de comandes.

Veiem un exemple d'un codi java que ens ho permet:

```
// QueryXML.java
import oracle.xml.sql.query.OracleXMLQuery;
import oracle.jdbc.driver.*;
import oracle.sql.*;
import java.sql.*;

public class QueryXML{

public static void main(String args[]){
try{
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
```

```
      Connection conn =
      DriverManager.getConnection("jdbc:oracle:oci8:@","xml","xml");
      OracleXMLQuery query = new OracleXMLQuery(conn,
      "select * from emp where deptno='20'");
      System.out.println(query.getXMLString());
      query.close();
      }
catch(Exception e){
      System.out.println("Error:"+e.getMessage());
      }
   }
}
```

Figura 49. Codi java per generar documents XML a partir d'una consulta

Per la seva banda el component OracleXMLSave és també una API la utilitat de la qual radica en emmagatzemar les dades d'un document XML en una base de dades. Mitjançant la associació de noms d'etiquetes amb els noms de columnes de la taula de la base de dades permet emmagatzemar les dades per columnes. Veiem-ne un exemple:

```
import oracle.xml.sql.dml.OracleXMLSave;
import oracle.jdbc.driver.*;
import oracle.sql.*;
import java.sql.*;

public class InsertXMLEmp{

public static String xmlData = new String(
"<?xml version = '1.0'?>"+
"<ROWSET>"+
"<ROW num='1'>"+
"<EMPNO>5555</EMPNO>"+
"<ENAME>MARIN</ENAME>"+
"<JOB>BOSS</JOB>"+
"<MGR>5555</MGR>"+
"<HIREDATE>29/11/2003 0:0:0</HIREDATE>"+
"<SAL>8000</SAL>"+
"<DEPTNO>30</DEPTNO>"+
"</ROW>"+
"</ROWSET>");
public static void main(String args[]){
try{
      DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
      Connection conn = DriverManager.getConnection("jdbc:oracle:oci8:@","xml","xml");
      OracleXMLSave sav = new OracleXMLSave(conn,"EMP");
      sav.insertXML(xmlData);
      sav.close();
      }
catch(Exception e){
      System.out.println("Error:"+e.getMessage());
      }
   }
}
```

Figura 50. Codi java per emmagatzemar les dades d'un document XML en una base de dades

Per acabar, tenim el paquet PL/SQL xmlgen. Aquest paquet és idèntic a la classe OracleXMLStore però integrat dins l'entorn PL/SQL. Per poder usar aquest paquet s'ha d'executar l'script SQL xmlgen.sql. Aquest paquet ens permet obtenir en format XML el resultat d'una consulta SQL.

Un exemple en seria:
SQL> select xmlgen.getXML('select from emp where deptno=10')
2 from dual;

5.2. Oracle 9i: persistència de documents XML

Amb aquesta petita introducció hem vist com a Oracle 9i a partir de funcionalitat implementades en Java permet generar o extreure documents XML a partir de consultes sobre la base de dades, com a partir de APIs podem extreure informació de taules en format XML o com a partir d'un document XML amb el seu DTD podem extreure totes les classes necessàries per poder generar documents a partir d'aquestes classes.

Una altra cosa que permet fer Oracle 9i és emmagatzemar documents XML en taules. Podem fer-ho de tres maneres diferents:

- Assignant el document com un únic objecte intacte en una taula com un CLOB (Character Large Object)
- Assignant elements XML a taules i columnes relacionals dins de la base de dades
- Assignant fragments de documents XML com CLOB i la resta com a taules

En funció del que vulguem fer ens haurem de decantar per una opció o per una altra.

Per tal de poder implementar aquestes opcions Oracle 9i suporta un nou tipus d'objecte (XMLType) amb unes funcions incorporades que permeten crear, extreure i indexar dades XML. L'usuari a més a més podrà generar documents XML mitjançant funcions SQL, funcions SYS_XMLGEN i SYS_XMLAGG i mitjançant el paquet DBMS_XMLGEN de PL/SQL

Les principals avantatges d'usar XMLType en una base de dades són:

- La unió de dos móns com XML i SQL que possibilita executar operacions SQL sobre XML i operacions XML sobre SQL
- Aporta funcions de creació, indexació, ajuda, navegació…

XMLType incorpora els següents mètodes:

createXML(xmlval IN varchar2)
Descripció: Funció estàtica que crea un tipus XMLType a partir d'una cadena de caràcters. També comprova que el nou tipus de dada correspon a un contingut XML ben format
Sintaxis: static function
createXML(xmlval IN varchar2)
RETURN sys.XMLTYPE deterministic

createXML(xmlval IN clob)
Descripció: Funció que genera un XMLType a partir d'una dada CLOB. També comprova que el nou valor correspon a un contingut XML ben format
Sintaxis: static function
createXML(xmlval IN clob)
RETURN sys.XMLTYPE deterministic

existNode(xpath IN varchar2)
Descripció: Donada una expressió XPath comprova si aquesta expressió es pot aplicar sobre el document i pot retornar algun node
Sintaxi: member function
existsNode(xpath IN varchar2)
RETURN number deterministic
extract(xpath IN varchar2)
Descripció: Funció que donada una expressió XPath la aplica al document i retorna el fragment sobre el que accedim com un tipus de dada XMLType
Sintaxi: member function
Extract(xpath IN varchar2)

RETURN sys.XMLType deterministic

isFragment()

Descripció: Comprova si el document és un fragment. Tindrem un fragment sempre que haguem aplicat la funció extract o qualsevol altra operació que ens hagi retornat una seqüència de nodes.

Sintaxi: member function
isFragment()
RETURN number

getClobval()

Descripció: Recull el XMLType com un CLOB

Sintaxi: member function
getClobVal()
RETURN clob deterministic

getStringVal()

Descripció: Recull el XML com una cadena de caràcters

Sintaxi: member function
getStringVal ()
RETURN varchar2 deterministic

getNumberVal()

Descripció: Obté el valor numèric apuntat per el XMLType com una dada numèrica

Sintaxi: member function
getNumberVal()
RETURN number deterministic

5.2.1. Emmagatzematge de documents XML com a CLOB

Lo primer que farem es crear una taula per guardar tot el document xml (en el nostre cas books.xml). Ho farem des de SQLPlus:

```
SQL> create table bib
2 (
3 contenido sys.XMLType
4 )
5 XMLType column contenido store as CLOB
6 (
7 storage (initial 12000 next 12000)
8 chunk 12000 cache
9 )
10 ;
```

Figura 51. Instruccions SQL per crear una taula per guardar un document XML

Ara el que farem serà extreure el contingut del document XML books.xml i emmagatzemar-lo dins de la columna contenido de la taula bib que acabem de crear:

```
SQL> create directory XMLFILES as 'C:\Documents and Settings\josep\Escritorio\TFC' ;
SQL> declare
2 CONTENT CLOB := ' ';
3 SOURCE bfile := bfilename('XMLFILES','books.xml');
4 begin
5 DBMS_LOB.fileOpen(SOURCE,DBMS_LOB.file_readonly);
6 DBMS_LOB.loadFromFile(CONTENT,SOURCE,DBMS_LOB.getLength(SOURCE),1,1);
7 DBMS_LOB.fileClose(SOURCE);
8 insert into bib (contenido)
9 values(sys.xmltype.createXML(CONTENT));
10 commit;
```

```
11 end;
12 /
```

Figura 52. Instruccions SQL per extreure el contingut d'un document XML i emmagatzemar-lo dins d'una columna

Ara podrem comprovar el contingut de la taula bib mitjançant la funció getClobVal(). Per poder veure el contingut complet haurem de modificar la variable d'entorn long i la variable pagesize per modificar el format de sortida de les dades ja que sinó no les podrem visualitzar correctament.

```
SQL> set long 10000
SQL> set pagesize 0
```

Figura 53. Instruccions per modificar el format de sortida de les dades

```
SQL> select b.contenido.getclobval()
2 from bib b;
```

Un cop executat això obtindríem les dades contingudes dins de contenido (tot el document books.xml).

5.2.2. Cerques dins de XMLType

Tal i com ja hem comentat abans un dels estàndards per navegar pels documents XML és el XPath. Oracle 9i permet usar les expressions XPath per navegar en una instància de tipus XMLType i per buscar en diverses instància del mateix tipus. Tal i com hem comentat quan em vist els mètodes dels objectes XMLType, aquest tipus d'objectes permeten l'accés als nodes dels documents XML amb els mètodes extract() i existNode().

Per fer les consultes sobre Oracle 9i usarem el programa SQL*Plus i ens connectem amb l'usuari system (manager)

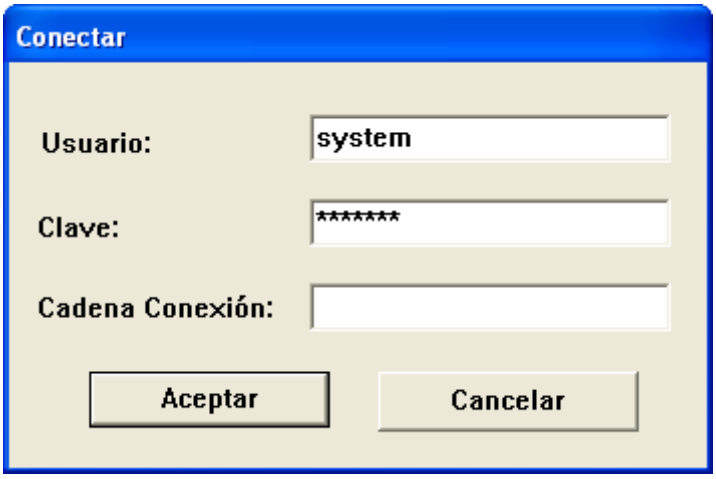


Figura 54. Pantalla de connexió a SQL Plus

a continuació creem una taula per guardar el document books.xml tal i com he explicat a l'apartat Emmagatzematge de documents XML com a CLOB i a partir d'aquí ja podem començar a fer consultes.

A continuació podem veure algunes consultes fetes a través de SQLPlus

SQL*Plus: Release 9.0.1.0.1 - Production on Jue May 27 21:02:00 2004

(c) Copyright 2001 Oracle Corporation. All rights reserved.

```
Conectado a:
Oracle9i Enterprise Edition Release 9.0.1.1.1 - Production
With the Partitioning option
JServer Release 9.0.1.1.1 - Production

SQL> select extract(b.contenido,'/bib/book/title').getClobVal()
  2  "titulo"
  3  from bib b;

<title>TCP/IP Illustrated</title>
<title>Advanced Programming in the Unix environment</title>
<title>Data on the Web</title>
<title>The Economics of Technology and Content for Digital TV</title>
<title>Oracle 9i.Desarrollo Web</title>
```

Figura 55. Consulta SQL*Plus Llibres (getClobVal())

```
SQL> select extract(b.contenido,'/bib/book/author').getClobVal()
  2  "Autors"
  3  from bib b;
<author>
  <last>Stevens</last>
  <first>W.</first>
</author>
<author>
  <last>Stevens</last>
  <first>W.</first>
</author>
<author>
  <last>Abiteboul</last>
  <first>Serge</first>
</author>
<author>
  <last>Buneman</last>
  <first>Peter</first>
</author>
<author>
  <last>Suciu</last>
  <first>Dan</first>
</author>
<author>
  <last>Brown</last>
  <first>Bradley</first>
</author>
```

Figura 56. Consulta SQL*Plus Autors (getClobVal())

Fins aquí el funcionament de Oracle treballant amb XPath sembla òptim. No obstant això amb les següents consultes he tingut certes dificultats:

Quan executo la següent consulta:

```
SQL> select extract(b.contenido,'/bib/book/title/text()').getStringVal()
  2  "Llibres"
  3  from bib b;
```

enlloc de retornar-me:

```
Llibres
--------------------------------------------------------------------------------
TCP/IP Illustrated
Advanced Programming in the Unix environment
Data on the Web
The Economics of Technology and Content for Digital TV
Oracle 9i.Desarrollo Web
```

Em retorna:

```
Llibres
--------------------------------------------------------------------------------
TCP/IP IllustratedAdvanced Programming in the Unix environmentData on the Web
The Economics of Technology and Content for Digital TVOracle 9i.Desarrollo Web
```

Figura 57. Consulta fallida SQL*Plus Llibres (getStringVal())

Com podem veure em retorna tots els llibres en una sola línea sense fer el corresponent salt de pàgina entre llibre i llibre.

L'altra consulta que m'ha creat dificultats ha estat la següent:

```
SQL> select extract(b.contenido,'/bib/book').getClobVal()
  2  "Informació sobre un llibre"
  3  from bib b
  4  where extract(b.contenido,'/bib/book/title/text()').getStringVal()='Oracle 9i.Desarrollo Web';
```

aquí el resultat esperat era:

```
Informació sobre un llibre
--------------------------------------------------------------------------------
<title>Oracle 9i.Desarrollo Web</title>
<isbn>84-415-1406-2</isbn>
<author><last>Brown</last><first>Bradley</first></author>
<publisher>Anaya Multimedia</publisher>
<price>54.00</price>
```

i en canvi el resultat obtingut ha estat:

```
ninguna fila seleccionada
```

Figura 58. Consulta fallida SQL*Plus Cerca dades d'un llibre (getStringVal())

A partir d'aquestes dues consultes fallides m'he posat a investigar i he pogut detectar que el problema està provocat per la manera com es guarden les dades dins d'Oracle.

Si recordem quan hem emmagatzemat el document books.xml a a taula bib ho hem fet en una columna de la taula. Sembla ser que degut a aquesta manera de guardar els documents XML quan usem el mètode getStringVal() ens retorna tot el contingut de la columna i per aquest motiu en la primera consulta fallida em retorna tot el text seguit i en la segona no em troba cap valor ja que el getStringVal() retorna tot el text de la columna. Si en la segona columna enlloc de l'igual usessim un like %Oracle 9i.Desarrollo Web% la condició segur que es compliria pero aleshores el que ens retornaria serien tots els llibres i no només les dades del llibre que busquem. Per aconseguir fer funcionar aquestes dues consultes hauria de tractar la informació obtinguda.

5.2.3. Manipulació de documents XML en columnes XMLType

Inserció
XMLType permet l'ús d'operacions de manipulació de dades com són la inserció, modificació i eliminació de dades XML sobre el tipus de dades XMLType.

Pel que fa a la inserció de dades abans ja hem vist un exemple de com inserir un document XML en un element d'una taula (usant el mòdul PL/SQL). També hi ha la possibilitat de inserir uns pocs elements XML sobre una taula usant la comanda SQL INSERT INTO. Veiem-ne un exemple:

Anem a veure un exemple d'inserció de dades XML sobre la taula biblio

```
SQL> insert into biblio (biblio_id, biblio_dat)
2 values (2, sys.XMLType.createXML(
3 '<bib><book year="2003">
4 <title>Desarrollo de aplicacion con PL/SQL</title>
5 <isbn>84-456-98-93</isbn>
6 <author><last>Scardina</last><first>Mark</first></author>
7 <publisher>Oracle Education</publisher>
8 <price> 39.50</price>
9 </book></bib>'));
```

Figura 59. Inserció dades usant la comanda SQL insert into

Modificació de dades XML
La modificació es realitza a través de la comanda UPDATE de SQL. A continuació veurem un exemple que modifica les dades de la taula inicial bib creada a l'apartat Emmagatzematge de documents XML com a CLOB

```
SQL> update bib b
2 set b.contenido=sys.XMLType.createXML(
3 '<book year="2001">
4 <title>Camino de la Atlantida</title>
5 <isbn>84-1400-00-87</isbn>
6 <author><last>King</last><first>Stephen</first></author>
7 <publisher>Plaza Janes</publisher>
8 <price>19.00</price>
9 </book>')
10 where extract(b.contenido,'/bib/book/title/text()').getStringVal()='TCP/IP Illustrated';
```

Figura 60. Modificació de dades usant la comanda SQL update

Eliminació de dades XML
Per esborrar dades XML d'una columna de tipus XMLType usarem la comanda SQL DELETE. Per exemple per eliminar del catàleg el llibre de Stephen King que acabem de donar d'alta a l'apartat anterior faríem:

```
SQL> delete from bib b
2 where extract(b.contenido,'/bib/book/author/last/text()').getStringVal()='King';
```

Figura 61. Eliminació de dades usant la comanda SQL delete

Capítol 6. Conclusions i tancament

6.1. Resum eXist

eXist és una base de dades nativa XML que proporciona una solució d'emmagatzematge pròpia i a més a més permet connectar aplicacions de tipus relacional per emmagatzemen auxiliar. És distribueix en forma de codi lliure.

La solució d'emmagatzematge pròpia està escrita en Java i utilitza les seves pròpies estructures de dades internes per guardar i indexar els documents XML a disc.

Independentment del sistema utilitzat –la solució nativa o la relacional- el processador XPath integrat en el producte permet realitzar consultes amb total abstracció de la capa d'emmagatzematge inferior.

El motor cerques a estat dissenyat per l'execució de queries XPath de forma eficient usant índex de tots els tipus. El servidor és accessible a través de interfícies http o XML/RPC i suporta la API de programació Java XML:DB

6.2. Resum Oracle 9i

Oracle 9i és una de les bases de dades líders dins el mon de les bases de dades relacionals. A partir de la versió 8i incorpora suport per XML aportant eines per l'ús de dades XML en les bases de dades.

Com hem vist Oracle permet que un document XML es pugui emmagatzemar com una única columna en la base de dades amb certes limitacions a l'hora de fer les consultes i indexar les dades. A més a més Oracle 9i també ofereix eines per particionar documents XML en columnes i taules de la base de dades i permet emmagatzemar documents XML com arxius externs a la base de dades.

També aporta una utilitat SQL XML per Java que aporta una sèrie de classes Java que permeten la inserció de dades XML en taules o vistes. Aquestes classes Java també permeten generar documents XML a partir de consultes SQL sobre la base de dades.

La utilitat Servlet Java XSQL també incorporada en el producte permet que la execució de consultes SQL ens retorni el resultat com un document XML i fins i tot a partir d'aquest document XML realitzar una transformació usant fulls d'estil que ens doni com resultat un document HTML.

Aquestes utilitats Java que Oracle incorpora son interessants però cal tenir grans coneixements de Java i tenir coneixements avançats del sistema Oracle per treure'n profit.

Oracle 9i també proporciona un tipus de dades XMLType com a tipus de dada element de les bases de dades que permet emmagatzemar en una mateixa taula del model relacional dades relacional i dades XML. Com hem vist Oracle 9i incorpora funcions i mètodes SQL per poder accedir i modificar aquests tipus de dades.

6.3. Conclusions finals

La principal pregunta que ens podem plantejar després d'haver estudiat Oracle 9i i eXist seria decidir quina de les dues alternatives presentades és millor per treballar amb documents XML. D'una banda tenim una base de dades relacional com Oracle 9i amb gran prestigi que ha fet un gran esforç per anar incorporant funcionalitats XML i de l'altra un producte destinat exclusivament a bases de dades natives XML.

Com ja s'ha comentat durant el desenvolupament del treball Oracle des de la versió 8i ha anat introduint funcionalitats per permetre treballar amb documents XML. De totes maneres totes

aquestes funcionalitats, bàsicament classes Java, que ha anat incorporant no són més que "parches" per tal de dir que Oracle suporta XML.

D'altra banda tenim una sèrie de limitacions que venen donades per la manera com s'emmagatzemen els documents XML dins Oracle (dins d'un sol camp d'una taula). Aquestes limitacions es fan paleses en algunes cerques que he intentat fer. Com per exemple:

```
SQL> select extract(b.contenido,'/bib/book').getClobVal()
  2  "Informació sobre un llibre"
  3  from bib b
  4  where extract(b.contenido,'/bib/book/title/text()').getStringVal()='Oracle 9i.Desarrollo Web';
```

A partir de la versió 9i s'incorpora un nou tipus elemental (XMLType) que no fa més que seguir en la mateixa línia apuntada en el paràgraf anterior és a dir seguir emmagatzemant els documents en un camp de tipus CLOB. El que realment seria útil seria poder emmagatzemar els documents XML com el que realment son dades XML.

Aquestes dificultats que he explicat fins aquí queda clar que amb eXist, i amb altres bases de dades natives com XIndice, no hi son ja que eXist és una base de dades XML nativa que tracta les dades com el que realment son: dades XML. Permet a més a més una cosa molt útil i interessant que es tractar col·leccions de documents XML, cosa que Oracle 9i no pot fer. eXist a més a més per fer les consultes pot usar XQuery. Això fa que els resultats obtinguts en les cerques que he realitzat sobre aquesta base de dades siguin molt més interessants.

Un cop fetes aquestes consultes a simple cop d'ull ja podem veure que les consultes realitzades sobre eXist son molt més potents i evidentment útils. Això és degut a que eXist permet usar XQuery com a llenguatge de consultes combinat amb XPath mentre que Oracle no pot treballar amb XQuery. D'aquí bé que les consultes realitzades sobre Oracle 9i són menys complexes i com a conseqüència els resultats obtinguts menys útils.

Si ens fixem en altres aspectes com la instal·lació i la facilitat d'us, des del meu punt de vista, no hi ha color. eXist és molt senzill d'instal·lar i en molt poca estona ja estàs treballant i fen cerques útils. Només es precisa tenir uns mínims coneixements de XPath i XQuery.

Per altra banda la instal·lació d'Oracle 9i és molt llarga (més d'una hora i quart) i requereix seguir les instruccions d'un manual. Un cop ja tenim instal·lat Oracle 9i al pc, per una persona com ja que no havia mai treballat amb aquesta base de dades, es fa difícil començar a treballar-hi i per realitzar una prova senzilla cal tenir forces coneixements tant d'SQL com d'Oracle.

Penso que si es tracta de treballar amb una base de dades on la principal font d'informació siguin documents XML cal orientar-se sempre cap a una base de dades de tipus natiu i en aquest cas eXist podria ser una bona solució donat que resulta senzill treballar-hi i que a més a més tal i com he comentat la instal·lació és força senzilla.

D'altra banda penso que si Oracle 9i té intenció de captar clients que tinguin interès en treballar amb bases de dades natives haurà de fer un gran esforç per permetre que el treball directament amb documents i col·leccions de documents XML i oferir eines i funcionalitats natives d'emmagatzematge i manipulació de les dades XML sense deixar de banda les capacitats SQL i relacions que actualment ja ofereix.

6.4. Línies de futur

No obstant el que he comentat a l'apartat de conclusions, val la pena comentar que s'hauria pogut plantejar la comparativa des d'altres punts de vista. Per exemple Oracle hauria sortit més ben valorat si per exemple haguéssim plantejat la comparativa des del punt de vista dels backups i restore, inserció massiva de dades, transaccions, etc.. Per exemple, Oracle té un sistema de backup i restore robust que ha anat evolucionant conjuntament amb el seu sistema de gestió de base de dades relacional mentre que eXist la única opció es fer backups i restores manualment i molt limitats (bàsicament, copiant l'estructura i contingut dels fitxers binaris on recauen les col.leccions XML del sistema).

També s'hauria pogut tenir en compte l'escalabilitat del sistema, és a dir, comparar què hauria passat si enlloc de treballar amb pocs documents xml de prova, haguéssim treballat amb un gran volum de documents xml, i a partir d'aquí fer consultes per veure el rendiment d'una base de dades i de l'altra. Pel que fa Oracle és sabut que suporta grans volums de dades i per lògica també suportaria el treball amb un gran volum de documents xml. Caldria veure com es comporta eXist treballant amb un volum important de dades.

.

Bibliografia

- World Wide Web Consortium
  http://www.w3.org/
- W3C Extensible Markup Language
  http://www.w3.org/XML/
- XML Path Language (XPath) 2.0
  http://www.w3.org/TR/xpath20/
- W3C XQuery 1.0: An XML Query Language
  http://www.w3.org/TR/xquery/
- XML.com
  http://www.xml.com.ve/que_es_XQuery.htm
- The XML Revolution
  http://www.brics.dk/~amoeller/XML/
- Introducció a XML
  http://www.forpas.us.es/aula/xml/Introducción%20a%20XML.pdf
- XML:DB Initiative for XML Databases:
  http://www.xmldb.org
- XML and Databases:
  http://www.rpbourret.com/xml/XMLAndDatabases.htm
- XML Database Products:
  http://www.rpbourret.com/xml/XMLDatabaseProds.htm
- Inspirational Technology
  http://xmldatabases.org
- eXist. Open Source XML Database
  http://www.exist-db.org
- Apache XIndice
  http://xml.apache.org/xindice/
- Introducción a XIndice
  http://www-106.ibm.com/developerworks/web/library/wa-xindice.html
- Pàgina principal d'Oracle
  http://otn.oracle.com/index.html
- Using XML in Oracle Database Applications
  http://otn.oracle.com/tech/xml/htdocs/about_oracle_xml_products.htm
- Internet Explorer Tools for Validating XML and Viewing XSLT Output
  http://www.microsoft.com/downloads/details.aspx?familyid=d23c1d2c-1571-4d61-bda8-adf9f6849df9&languageid=f49e8428-7071-4979-8a67-3cffcb0c2524&displaylang=en