View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by UEF Electronic Publications

Natiivit XML-tietokannat

Antti Mustonen

Pro gradu –tutkielma

ITÄ-SUOMEN YLIOPISTO

Tietojenkäsittelytieteen laitos

Tietojenkäsittelytiede

Toukokuu 2013

ITÄ-SUOMEN YLIOPISTO, Luonnontieteiden ja metsätieteiden tiedekunta, Kuopio
Tietojenkäsittelytieteen laitos
Tietojenkäsittelytiede

Opiskelija, Antti Mustonen: Natiivit XML-tietokannat
Pro gradu –tutkielma, 70 s.
Pro gradu –tutkielman ohjaaja: prof., FT Pekka Kilpeläinen
Toukokuu 2013

Metamerkintäkieli XML:n suosion kasvun myötä on syntynyt tarve tehokkaalle XML-tiedon hallinnalle. Suurten tietomäärien hallintaan perinteisesti käytetyt relaatiotietokantajärjestelmät eivät tarjoa optimaalista ratkaisua XML-tiedon hallintaan, koska relaatiotietokantaan tallennettava XML-data on aina ensin muunnettava relaatiotietokannoille sopivaan muotoon. Tämä muunnosvaatimus johtuu XML-tiedon sekä relaatiotietokantojen tiedon rakenteiden erilaisuudesta: relaatiotietokantojen tiedon rakenne on aina samanlaisessa taulupohjaisessa muodossa, kun taas XML-muotoisen tiedon rakenne vaihtelee sovellusalan mukaan. Epäsäännöllisen rakenteen ansiosta XML on hyvin joustava kieli, jonka avulla voidaan kuvata tietoa käytännössä kaikenlaisilta sovellusaloilta.

Natiivit XML-tietokannat tarjoavat tehokkaan ratkaisun XML-muotoisen tiedon hallintaan. Natiivit XML-tietokannat on suunniteltu mahdollisimman tehokasta XML-datan hallintaa tavoitellen, joten natiiveissa XML-tietokannoissa XML-tieto voidaan tallentaa sellaisenaan suoraan tietokantaan ilman ylimääräisiä muunnoksia. Natiivit XML-tietokannat tarjoavat myös tehokkaat ratkaisut XML-tiedon hakemiseen ja palauttamiseen tietokannasta. XML-tiedolle on olemassa useita erilaisia käsittelykieliä, joista natiivien XML-tietokantojen näkökulmasta merkittävimmät ovat osoituskieli XPath sekä kyselykieli XQuery.

Natiivien XML-tietokantojen kehittäminen on aloitettu 1990-luvun lopussa. Natiivit XML-tietokannat hyödyntävät useita alun perin perinteisiin relaatiotietokantajärjestelmiin kehitettyjä menetelmiä. Relaatiotietokannanhallintajärjestelmissä hyväksi havaitut menetelmät – kuten esimerkiksi indeksointi- ja samanaikaisuudenhallintamenetelmät – eivät kuitenkaan toimi suoraan sellaisinaan natiiveissa XML-tietokannoissa, vaan niiden toiminta on muokattava natiiveille XML-tietokannoille soveltuvaan muotoon. Tämä johtuu siitä, että XML-tiedon joustava rakenne monimutkaistaa natiivien XML-tietokantojen toiminnallisuuksien toteutustapoja verrattuna relaatiotietokantajärjestelmien vastaaviin.

Avainsanat: XML, Tietokanta, Indeksointi, Kyselyjen evaluointi, Transaktioidenhallinta.

ACM-luokat (ACM Computing Classification System, 1998 version): H.2, I.7.2

UNIVERSITY OF EASTERN FINLAND, Faculty of Science and Forestry, Kuopio
School of Computing
Computer Science

Student, Antti Mustonen: Native XML databases
Master's Thesis, 70 p.
Supervisor of the Master's Thesis: prof., PhD Pekka Kilpeläinen
May 2013

The development of meta-markup language XML was started in 1996 by the World Wide Web Consortium. Soon after the development, XML became the most popular language for marking up structured documents. The popularity of XML has led to a need for effective management of XML data. Relational database systems have been the most used tool for managing large amounts of data over the past few decades. However, relational database systems do not provide optimal ways to manage XML data, due to differences between the structure of XML data and the structure of the data in a relational database. This is because the data in a relational database is regular, whereas XML data is in irregular form. The mismatches between XML-structured data and relational data mean that extra transformations are needed if one wants to store XML data in a relational database.

Native XML databases provide an efficient way to manage XML data, since they are designed especially to store and retrieve XML documents. There are many different kinds of processing languages for XML. From a native XML database's perspective the most important ones are the selection language XPath and the query language XQuery.

The development of native XML databases has been going on since the end of the last millennium. Because of the relatively young age of native XML databases, they exploit many methods that were originally designed for relational database systems. However nearly all of those methods – such as indexing and concurrency control protocols – need to be adapted to fit in a native XML database environment. This is because the irregularity of XML data complicates the implementation methods for databases that intend to process XML data effectively.

Keywords: XML, Database, Indexing, Query Evaluation, Transaction Management.

CR Categories (ACM Computing Classification System, 1998 version): H.2, I.7.2

Esipuhe

Tämä tutkielma on tehty Itä-Suomen yliopiston tietojenkäsittelytieteen laitokselle keväällä 2013. Tutkielman ohjaajana toimi Pekka Kilpeläinen, jolle haluan osoittaa erityiskiitoksen.

Kuopiossa 23.5.2013

____

Antti Mustonen

Lyhenneluettelo

Ajax	Asynchronous JavaScript and XML
CSS	Cascading Style Sheets
DOM	Document Object Model
DTD	Document Type Definition
HTML	HyperText Markup Language
I/O	Input/Output
MathML	Mathematical Markup Language
PDF	Portable Document Format
REST	Representational State Transfer
SAX	Simple API for XML
SGML	Standard Generalized Markup Language
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
ViST	Virtual Suffix Tree
W3C	World Wide Web Consortium
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformations

Sisällysluettelo

1 Johdanto ... 1

2 XML:n perusteet ... 3

2.1 Mikä on XML? ... 3
2.2 XML-dokumentin rakenne ... 4
2.3 Rakenteen ja sanaston määrittely ... 7
2.4 XML-käsittelykielet ... 8
2.4.1 XPath ... 8
2.4.2 XQuery ... 11

3 Natiivit XML-tietokannat ... 14

3.1 Miksi natiiveja XML-tietokantoja tarvitaan? ... 14
3.2 Mitä natiivit XML-tietokannat ovat? ... 16
3.3 Natiivien XML-tietokantojen arkkitehtuurit ... 19

4 Natiivien XML-tietokantojen suorituskyvyn optimointi ... 23

4.1 Indeksointi ... 23
4.1.1 Polkuindeksointi ... 24
4.1.2 Solmuindeksointi ... 25
4.1.3 Sekvenssipohjainen indeksointi ... 26
4.2 Kyselyjen evaluointi ... 28
4.2.1 Tiedon fragmentointi ... 28
4.2.2 Solmutunnisteiden hyödyntäminen ... 32
4.2.3 Rakenteellisen liitosoperaation evaluointi ... 36

5 Transaktioidenhallinta natiiveissa XML-tietokannoissa ... 41

5.1 Transaktioiden ACID-ominaisuudet ... 41
5.2 Samanaikaisuudenhallinta ... 42
5.2.1 Lukitusmenetelmät ... 43
5.2.2 Aikaleimapohjaiset menetelmät ... 47
5.3 Häiriöistä toipuminen ... 50

6 Kokeellinen osio ... 52

6.1 Natiivi XML-tietokanta eXist ... 52
6.2 Vaatimukset ... 53
6.3 Toteutustekniikat ... 54
6.4 Tulokset ... 54

7 Yhteenveto ja pohdinta ... 59

Lähteet ... 61

1 Johdanto

Metamerkintäkieli XML:n kehitystyö aloitettiin vuonna 1996. Pian tämän jälkeen XML kasvoi nopeasti suosituimmaksi rakenteisten dokumenttien merkintäkieleksi. XML:n merkittävimpiä vahvuuksia ovat selkeys ja joustavuus: XML-kielen joustavuuden ansiosta uusien XML-pohjaisten kielten määrittely on hyvin helppoa. Tästä johtuen erilaisia XML-pohjaisia kieliä on kehitetty hyvin erilaisten sovellusalojen tarpeisiin. Luvussa 2 esitellään XML:n perusteet. XML-muotoiselle tiedolle on olemassa useita erilaisia käsittelykieliä. Luku 2 sisältää lisäksi natiivien XML-tietokantojen näkökulmasta tärkeimpien XML-käsittelykielten esittelyt. Nämä kielet ovat osoituskieli XPath sekä kyselykieli XQuery.

XML:n suuren suosion vuoksi on syntynyt tarve tehokkaalle XML-tiedon hallinnalle. Tähän tarpeeseen vastaamiseksi on aloitettu natiivien XML-tietokantojen kehittäminen. Natiivit XML-tietokannat tarjoavat tehokkaan XML-muotoisen tiedon hallintaympäristön. Luvussa 3 esitellään natiivien XML-tietokantojen tärkeimmät ominaisuudet.

Natiivien XML-tietokantojen suorituskykyä parantaviin menetelmiin pureudutaan luvussa 4, joka sisältää esittelyt niin indeksoinnista kuin kyselyjen evaluointimenetelmistä. Indeksoinnin tarkoituksena on rajata tietokannan suorittamien kyselyiden hakuavaruutta ja täten nopeuttaa kyselyiden suorittamista. XML-muotoisen tiedon indeksointimenetelmät voidaan luokitella kolmeen luokkaan: polkuindeksointiin, solmuindeksointiin sekä sekvenssipohjaiseen indeksointiin. Kyselyjen evaluoinnissa voidaan puolestaan hyödyntää näitä indeksejä. Kyselyjen evaluoinnin tarkoituksena on suorittaa tietokannan vastaanottamat kyselyt mahdollisimman tehokkaasti. Kyselyiden suoritustehokkuuden kannalta olennainen asia on levyliikenteen määrä: oheismuistin käyttö halutaan pitää mahdollisimman pienenä.

Natiivit XML-tietokannat tarjoavat yleensä palveluitaan useille samanaikaisille käyttäjille. Tämä tarkoittaa sitä, että samanaikaisuudenhallinta on eräs natiiveille XML-tietokannoille tyypillinen vaatimus. Eräs toinen natiiveille XML-tietokannoille tyy-

pillinen vaatimus on häiriöistä toipuminen: tietokantojen pitää pystyä toipumaan tilapäisistä laitteistohäiriöistä. Natiivien XML-tietokantojen samanaikaisuudenhallinta- sekä toipumismenetelmät perustuvat transaktioihin, jotka ovat luku- ja kirjoitusoperaatioiden sarjoja. Luvussa 5 esitellään transaktioidenhallintaa natiiveissa XML-tietokannoissa: luku sisältää katsaukset natiivien XML-tietokantojen samanaikaisuudenhallinta- sekä toipumismenetelmistä.

Luvussa 6 esitellään tutkielman kokeellinen osio, jonka tarkoituksena oli tutustua natiiviin XML-tietokantaan nimeltä eXist. Kokeellisessa osiossa käsiteltiin XML-muodossa olevia Shakespearen näytelmiä. Kokeellisen osion tuloksena syntyi web-palvelu, jonka avulla käyttäjä voi hakea haluamiaan tietoja näistä näytelmistä. Lopuksi luvussa 7 vedetään yhteen tutkielman anti sekä arvioidaan lyhyesti natiivien XML-tietokantojen tulevaisuutta.

2 XML:n perusteet

Metamerkintäkieli XML (Extensible Markup Language) (Bray & al., 2008) on kasvattanut suuren suosion viimeisen 15 vuoden aikana ja sitä käytetään laajasti eri sovellusaloilla, kuten esimerkiksi tiedonvaihdossa verkon välityksellä. Tässä luvussa esitellään XML pääpiirteittäin. XML on jo itsessään varsin laaja aihe, joten sitä ei esitellä tässä tutkielmassa kovinkaan yksityiskohtaisesti, vaan johdantona natiivien XML-tietokantojen esittelyyn.

2.1 Mikä on XML?

XML on W3C:n (World Wide Web Consortium) kehittämä metamerkkauskieli, jonka avulla voidaan määritellä rakenteellisia merkkauskieliä ja kuvata dokumenttien loogisia rakenteita. XML:n kehitystyö aloitettiin vuonna 1996. Pian tämän jälkeen XML-kielestä kasvoi suosituin rakenteisten dokumenttien merkintäkieli. XML:n avulla dokumenttiin voidaan merkata tietoa dokumentin eri osista, ja joustavuutensa ansiosta XML-määritysten avulla voidaan luoda helposti uusia XML-pohjaisia merkintäkieliä. Näiden merkintäkielten sääntöjen määrittelystä kerrotaan tarkemmin luvussa 2.3.

XML pohjautuu SGML-kieleen (Standard Generalized Markup Language) (ISO, 1986), joka on 1980-luvulla kehitetty metakieli dokumenttien merkintäkielten määrittelemiseen. Motivaatio XML:n kehittämiselle syntyi SGML:n tiettyjen rajoitteiden takia. Ensinnäkin SGML ei sovi kovin hyvin datan vaihtoon internetin välityksellä, mikä on puolestaan yksi XML:n tärkeimmistä ominaisuuksista (Walsh, 1998). Lisäksi SGML koettiin liian monimutkaiseksi ja raskaaksi kieleksi dokumenttien merkkaamiseen.

XML on SGML:n yksinkertaistettu alijoukko, koska XML-kielestä haluttiin SGML-kieltä selkeämpi metamerkkauskieli. XML:n merkittäviä vahvuuksia ovat yleiskäyttöisyys, alustariippumattomuus sekä joustavuus. XML-dokumentit ovat sekä ihmisten että koneiden ymmärrettävissä olevassa muodossa, minkä ansiosta XML-kielen

oppiminen sekä sitä käsittelevien ohjelmien kirjoittaminen on helppoa. XML-kielelle onkin olemassa paljon erilaisia työkaluja.

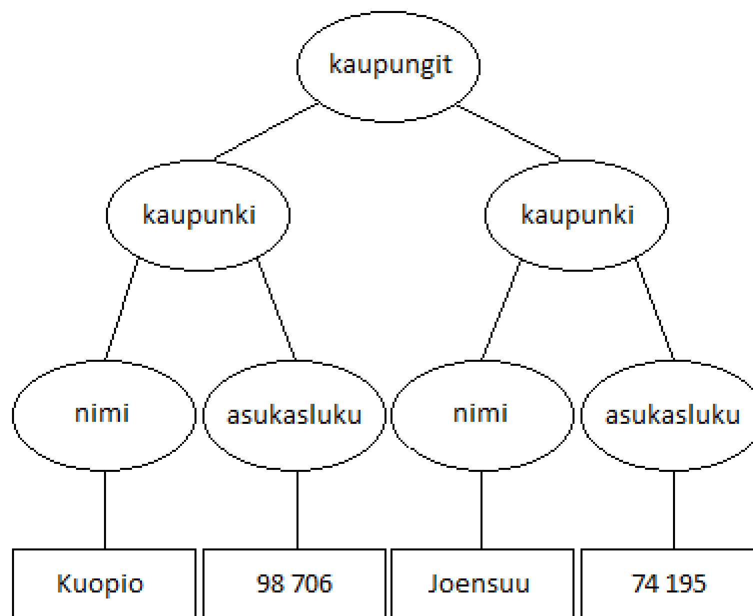
2.2 XML-dokumentin rakenne

XML-dokumentti koostuu dokumentin merkkauksesta sekä dokumentin tekstisisällöstä. XML:n yleisin ja tärkein merkkausmuoto on elementti. Elementin sisältö kuvataan sen aloitus- ja lopetusmerkintöjen välissä. Esimerkiksi kuvan 1 XML-esimerkissä ensimmäisen `nimi`-elementin tekstisisältö on ”Kuopio”. Elementti `kaupunki` puolestaan sisältää `nimi`- sekä `asukasluku`-elementit.

```
<kaupungit>
  <kaupunki>
    <nimi>Kuopio</nimi>
    <asukasluku>98 706</asukasluku>
  </kaupunki>
  <kaupunki>
    <nimi>Joensuu</nimi>
    <asukasluku>74 195</asukasluku>
  </kaupunki>
</kaupungit>
```

Kuva 1. Esimerkki XML-dokumentista.

Elementtejä voi olla rajattomasti sisäkkäin, mutta ne eivät saa mennä ristiin toisten elementtien kanssa, sillä XML-merkkaus perustuu hierarkkiseen elementtirakenteeseen. Jokaisella XML-dokumentilla tulee olla tasan yksi juurielementti, eli elementti, jolla ei ole vanhempaa ja joka sisältää kaikki muut dokumentin elementit. Kuvan 1 esimerkin juurielementti on `kaupungit`-elementti. Kaikilla muilla elementeillä paitsi juurielementillä on tasan yksi vanhempi, joten XML-dokumentti muodostaa puurakenteen. Kuvassa 1 esitetyn XML-dokumentin muodostama puu on esitetty kuvassa 2.



Kuva 2. Esimerkki XML-dokumentin muodostamasta puurakenteesta.

Elementtien lisäksi XML-dokumentin muut mahdolliset merkkausmuodot ovat:

- attribuutti
- kommentti
- prosessointiohje
- merkkidatalohko (CDATA)
- entiteettiviittaus
- dokumenttityypin esittely

Attribuutilla tarkoitetaan ominaisuutta, joka merkataan elementin aloitusmerkinnän yhteyteen. Attribuutti koostuu attribuutin nimestä sekä arvosta. Esimerkiksi `kaupunki`-elementti, jolla on attribuutti `id` arvolla 1, merkattaisiin `<kaupunki id=”1”>`.

Kommentti merkataan puolestaan aloitusmerkinnällä `<!--` ja lopetusmerkinnällä `-->`. XML-kommentin toimintaperiaate on sama kuin ohjelmointikielten kommenteilla, sillä XML-kommentin yleisimpänä käyttötarkoituksena on säilöä tekijän muis-

tiinpanoja dokumenttiin. XML-sovellukset voivat jättää XML-kommentit huomioimatta.

Prosessointiohjeella annetaan tietoa ja ohjeita jollekin sovellukselle ja sen aloitusmerkintä on `<?` ja lopetusmerkintä `?>`. Merkkidatalohkot puolestaan merkataan aloitusmerkinnällä `<![CDATA[` ja lopetusmerkinnällä `]]>`. Merkkidatalohkojen käyttö on tarpeellista sellaisten tekstinpätkien kanssa, jotka sisältävät useita XML-merkkaukseen varattuja merkkejä kuten ”&” ja ”>”, eli esimerkiksi tietokoneohjelmien lähdekoodien kanssa. XML-jäsennin jättää prosessoimatta merkkidatalohkoon asetetun tekstin, joten merkkidatalohkon sisällä olevat XML-merkkauksen varatut merkit eivät sotke XML-dokumentin merkkausta.

Entiteettiviittaukset auttavat varattujen merkkien kirjoittamista XML-dokumentin sisältöön. Esimerkiksi merkki ”<” tarkoittaa normaalisti jonkin aloitusmerkinnän ensimmäistä merkkiä. Kyseisen merkin käyttö normaalissa tekstisisällössä onnistuu lyhyen entiteettiviittauksen `<` avulla. Entiteettiviittaukset voivat myös viitata isompiin kokonaisuuksiin kuin yksittäisiin merkkeihin: tällöin entiteettiviittaukset toimivat fyysisten esitysmuotojen lyhennysmerkintöinä. Dokumenttityypin esittelyn avulla voidaan puolestaan määritellä dokumentin sallittu sanasto sekä rakenne.

XML-dokumentin oikeellisuus määritellään hyvinmuodostuneisuuden sekä validisuuden perusteella. XML-dokumentti on hyvinmuodostettu (well formed), jos se täyttää tietyt ehdot, joita ovat muun muassa

- Dokumentilla on tasan yksi juurielementti.
- Kaikilla elementeillä on lopetusmerkintä.
- Elementtien nimet ovat merkkikokoriippuvaisia.
- Elementit voivat olla sisäkkäin, mutta ne eivät mene ristiin toisten elementtien kanssa.
- Jokainen attribuutin arvo on merkattu lainausmerkkien sisälle.

XML-dokumentti on validi (valid) mikäli se on hyvin muodostettu ja viittaa johonkin DTD-kuvaukseen sekä toteuttaa kyseisen kuvauksen mukaisen rakenteen.

2.3 Rakenteen ja sanaston määrittely

XML:n avulla voidaan määritellä säännöt tiedon rakenteelle. Tällaista sääntöjen mukaista rakennetta kutsutaan usein XML-pohjaiseksi kieleksi ja se koostuu sovelluskohtaisesta sanastosta. Esimerkiksi XHTML (eXtensible Hypertext Markup Language) (Pemberton & al., 2002) ja MathML (Mathematical Markup Language) (Ausbrooks & al., 2010) ovat tunnettuja XML-pohjaisia kieliä. XHTML on perinteisen HTML-kielen puhtaaksi XML:ksi siistitty versio. XHTML:n määritykset ovat HTML:ää tarkempia ja sen tarkoituksena on olla mahdollisimman yksiselitteinen, jotta eri verkkoselaimet osaisivat tulkita sitä samalla tavalla. MathML on puolestaan XML-pohjainen kieli matemaattisten symbolien ja kaavojen esittämiseen. MathML:n mukaisten dokumenttien juurielementti on `math`-elementti. MathML-kielen sanastosta löytyy elementit erilaisten matemaattisten symbolien merkintään: esimerkiksi neliöjuuren sisältö merkitään `msqrt`-elementin sisään.

XML ei siis itsessään määrittele sanastoa eli merkkauksessa käytettäviä nimiä, vaan se voidaan määritellä kulloisenkin sovelluskohtaisen sanaston perusteella, kuten edellä mainitut XHTML:n ja MathML:n esimerkit osoittavat. XML on laajennettava merkintäkieli, koska uusia XML-pohjaisia kieliä on helppoa luoda ja määritellä. Laajennettavuuden ansiosta XML on hyvin joustava kieli.

XML-dokumentin rakenteen sääntöjen määrittelyä varten on olemassa erilaisia XML-kuvauskieliä, joista yleisimmät ovat DTD (Document Type Definition) ja XML Schema. DTD on kuvauskielistä perinteisin. Sen avulla dokumentille voidaan määrittää dokumentin sallittu rakenne ja sanasto. XML Schema (Sperberg-McQueen & Thompson, 2000) on puolestaan DTD:tä uudempi ja ilmaisuvoimaisempi XML-kuvauskieli. XML Scheman avulla dokumentin rakenne voidaan kuvata tarkemmin kuin DTD:llä. XML Scheman mukaiset määrittelyt kirjoitetaan XML-kielellä (toisin kuin DTD:t), joten niiden prosessointi onnistuu tavallisilla XML-työkaluilla. Lisäksi XML Schema tukee tietotyyppien (esim. liukuluvut ja päivämäärät) määrittelyä, mikä johtaa huomattavasti tarkempaan dokumentin määrittelyyn DTD:hen verrattuna.

XML-dokumenttiin ei kuitenkaan ole pakko liittyä DTD:n, XML Scheman tai minkään muun kuvauskielen mukaista kuvausta määrittelemään dokumentin rakennetta. XML-kuvauskielten käyttö on kuitenkin monesti hyödyllistä, sillä niiden avulla voidaan varmistaa dokumentin rakenteen oikeellisuus ja täten parantaa XML-tiedon laatua.

2.4 XML-käsittelykielet

XML:lle on olemassa useita erilaisia käsittelykieliä. Seuraavaksi esitellään natiivien XML-tietokantojen näkökulmasta kaksi merkittävintä: osoituskieli XPath ja kyselykieli XQuery, jotka ovat käytössä käytännössä kaikissa nykyaikaisissa natiiveissa XML-tietokannoissa.

2.4.1 XPath

XPath on XML-datan osoituskieli, joka on ollut W3C:n suositus vuodesta 1999 lähtien (Clarke & DeRose, 1999). XPathin avulla voidaan paikantaa XML-dokumentista tietoa. XPath on käytössä mm. myöhemmin esiteltävässä XQuery-kyselykielessä sekä useissa muissa XML-tekniikoissa. Nimensä mukaisesti XPathin notaatiosyntaksi perustuu XML-dokumentin polkuun. XPathissa lausekkeilla tunnistetaan XML-dokumentin muodostaman puun solmuja niiden tyypin, nimen, arvojen sekä dokumentin solmujen keskinäisten suhteiden perusteella.

XPathissa tiedon paikantaminen perustuu sijaintipolkuihin (location paths). Sijaintipolku on lauseke, jonka perusteella haluttu solmujoukko palautetaan. Sijaintipolku voi olla joko absoluuttinen tai suhteellinen. Absoluuttisen sijaintipolun alkuun merkitään ”/”. Absoluuttinen sijaintipolku alkaa aina juurisolmusta, kun taas suhteellinen sijaintipolku alkaa nykyisestä sijaintikohdasta.

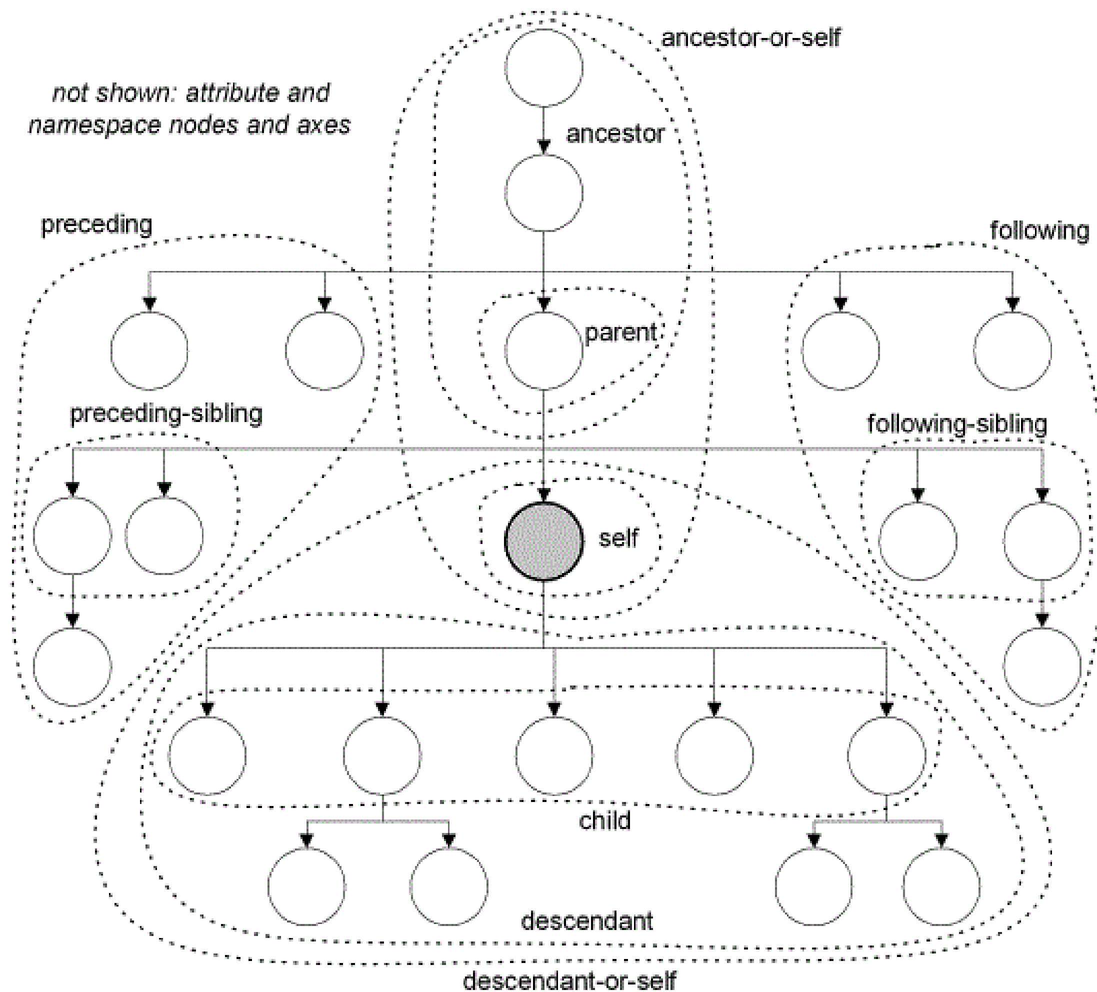
Sijaintipolun askel koostuu akselista, solmutestistä sekä vapaavalintaisista ehdoista eli predikaateista. Sijaintipolkulausekkeessa akselia seuraa merkintä ”::” ja solmutesti, joten lausekkeen sijaintiaskeleen syntaksi on muotoa

```
akseli::solmutesti([predikaatti])*
```

Akselilla tarkoitetaan sijaintiaskeleen (location step) ja kontekstisolmun (eli sillä hetkellä prosessoitavan solmun) välistä suhdetta XML-datan muodostamassa puurakenteessa. Akselin avulla kuvattavat suhteet ovat:

- ancestor (solmun esi-isät)
- ancestor-or-self (esi-isät sekä nykyinen solmu)
- attribute (attribuutit)
- child (lapsisolmut)
- descendant (jälkeläiset)
- descendant-or-self (jälkeläiset sekä nykyinen solmu)
- following (seuraavat solmut)
- following-sibling (seuraavat sisarsolmut)
- namespace (nimiavaruudet)
- parent (vanhempi)
- preceding (edeltävät solmut)
- preceding-sibling (edeltävät sisarsolmut)
- self (nykyinen solmu)

Kuvassa 3 on havainnollistettu harmaalla merkatun kontekstisolmun ja muiden elementtisolmujen väliset suhteet.



Kuva 3. Akselimäärittelyjen avulla kuvattavat suhteet (Kredel 2013).

Solmutestin avulla voidaan määrittää haettavien solmujen tyyppi (esimerkiksi elementti tai attribuutti) sekä nimi. Esimerkiksi kontekstisolmun lapsisolmujen `A` lapsisolmuihin `B` viitattaisiin XPath-lausekkeella `child::A/child::B`. XPath-lausekkeiden yksinkertaistamiseksi on luotu myös vaihtoehtoinen syntaksi, jonka avulla osa akseleista voidaan kirjoittaa lyhyempään muotoon. Akselimäärittelyistä child voidaan haluttaessa jättää kokonaan kirjoittamatta, koska tyhjä akseli vastaa lyhennetyssä syntaksissa child-akselia. Muut akselit, joilla on lyhennetty muoto ovat attribute, descendant-or-self, parent ja self. Edellä mainittuja akseleita vastaavat lyhenteet on kuvattu taulukossa 1.

Taulukko 1. Akseleiden lyhennetyt syntaksit.

Täysi syntaksi	Lyhennetty syntaksi
child	
attribute	@
descendant-or-self	//
parent	..
self	.

Edellä esitetty XPath-lauseke voitaisiin siis kirjoittaa ilman child-akseleita lyhyesti muotoon `A/B`.

XPath-lausekkeita voidaan täydentää predikaateilla, eli ehdoilla. Ehto on hakasulkuihin merkattava lauseke, jonka avulla voidaan rajata haettavaa solmujoukkoa. Ehto merkitään solmutestin jälkeen ja ehtoja voidaan määritellä useita peräkkäin. Esimerkiksi edellisen esimerkin tarkentaminen muotoon, jossa haetaan ne `B`-elementit, joilla on attribuutin `C` arvo ”2013”, merkataan `A/B[@C=”2013”]`. XPathin sijaintipolkulausekkeet muistuttavat ulkoasultaan usein hakemistopolkuja.

2.4.2 XQuery

XQuery on W3C:n kehittämä funktionaalinen XML-kyselykieli, jonka avulla voidaan hakea XML-datasta haluttua tietoa (Boag & al., 2010). XQueryn toiminta ei kuitenkaan rajoitu pelkkään tiedon hakemiseen (Walmsley, 2007, luku 1), vaan haettua tietoa voidaan myös muuntaa ja palauttaa käyttäjän haluamassa rakenteessa, mikä tekee XQuerystä XPathia huomattavasti monipuolisemman kielen. XPath toimiikin XQueryn alijoukkona, joten XQueryllä on mahdollista tehdä kaikki se mitä XPathillakin. XQuery on tällä hetkellä merkittävin XML-kyselykieli, ja se on käytössä kaikissa nykyaikaisissa natiiveissa XML-tietokannoissa. XQuery mielletäänkin helposti natiivien XML-tietokantojen vastineeksi relaatiotietokantojen yleisesti käyttämälle

kyselykielelle SQL (Structured Query Language). XQueryä voidaan kuitenkin käyttää muuallakin kuin vain tietokantaympäristöissä, esimerkiksi suoraan tiedostojärjestelmän XML-dokumentteihin.

XQueryn tärkeimmät toiminnallisuudet ovat

- tiedon hakeminen XML-datasta halutuin kriteerein
- haetun tiedon palauttaminen käyttäjän haluamassa muodossa
- tiedon hakeminen yksittäisistä XML-dokumenteista tai useiden XML-dokumenttien muodostamista kokoelmista
- numeroilla ja merkkijonoilla operointi.

XQueryn tietomalli (data model) perustuu solmuihin (node) ja atomiarvoihin. Tietomallista löytyy kuusi erityyppistä solmua: elementtisolmut kuvaamaan XML-elementtejä, attribuuttisolmut kuvaamaan XML-attribuutteja, tekstisolmut kuvaamaan elementin merkkidatasisältöä, prosessointiohjeet kuvaamaan XML-prosessointiohjeita, kommenttisolmut kuvaamaan XML-kommentteja ja dokumenttisolmut kuvaamaan kokonaisia XML-dokumentteja.

Atomiarvoilla tarkoitetaan data-arvoja kuten kokonaislukuja tai merkkijonoja. Solmut ja atomiarvot muodostavat sekvenssejä (sequence), jotka ovat järjestettyjä joukkoja, jotka puolestaan koostuvat solmuista sekä atomiarvoista. Kaikki XQuery-lausekkeet palauttavat sekvenssejä. XQueryllä ohjelmoitavat lausekkeet noudattavatkin usein hyvin samankaltaista rakennetta, koska XQuery tarjoaa ns. FLWOR-lauserakenteen. Lyhenne FLWOR tulee sanoista for, let, where, order by sekä return. For-lauseella määritellään iteraatio, eli kyselyssä toistettava osa. Let-lauseella voidaan asettaa muuttujalle jokin arvo. Where-lauseella voidaan asettaa jokin ehto hakukriteeriksi, order by -lauseella palautettava sekvenssi voidaan asettaa haluttuun järjestykseen ja lopuksi return-lauseella määritetään se mitä halutaan palauttaa ja minkälaisessa rakenteessa. Esimerkiksi haettaessa XHTML-kielisestä dokumentista

`uutiset.xml` sellaisten `p`-elementtien sisältö, joilla on `class`-attribuutin arvo ”uutinen”, voitaisiin käyttää seuraavaa XQuery-koodia:

```
let $document := doc("uutiset.xml")

for $x in $document//p

where $x[@class="uutinen"]

order by $x/text()

return <uutinen>{$x/text()}</uutinen>
```

Esimerkkikoodi palauttaisi `uutinen`-elementtejä, joiden tekstisisältöinä olisi löydettyjen `p`-elementtien tekstisisällöt. Order by -lausekkeen perusteella `uutinen`-elementit palautettaisiin niiden tekstisisältöjen mukaisessa aakkosjärjestyksessä.

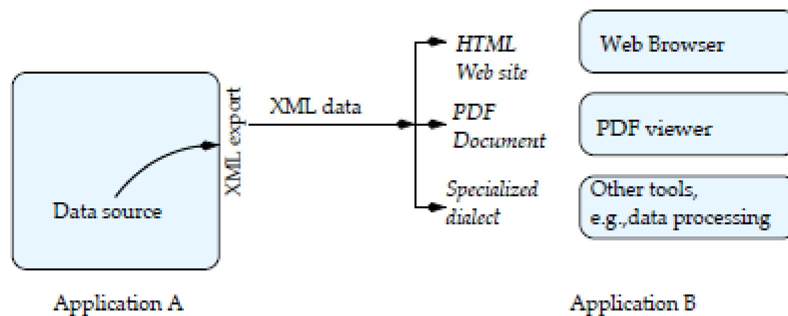
XQuery tukee myös ehtolause- eli if-then-else-rakennetta, jonka avulla XQuerylla ohjelmoitavia toiminnallisuuksia voidaan monipuolistaa. Myös funktioilla voidaan täydentää XQuery-kyselyitä: funktiot voivat olla käyttäjän luomia tai XQueryssa valmiina olevia funktioita. XQuery sisältää yli 100 valmista sisäänrakennettua funktiota, joihin kuuluu esimerkiksi merkkijonojen käsittelyyn liittyvät funktiot.

3 Natiivit XML-tietokannat

XML:n suuren suosion vuoksi on syntynyt tarve tehokkaalle XML-dokumenttien hallinnalle. Yleisesti suurten tietomäärien hallintaan käytetyin ratkaisu on ollut viime vuosikymmeninä relaatiotietokannat. Relaatiotietokannoissa on kuitenkin omat rajoitteensa XML-muotoisen tiedonhallinnan näkökulmasta. Natiivien XML-tietokantojen tarkoituksena on päästä irti näistä rajoituksista ja tarjota tehokas XML-datan hallintajärjestelmä. Tässä luvussa perustellaan natiivien XML-tietokantojen tarve sekä kuvataan niille tyypilliset ominaispiirteet.

3.1 Miksi natiiveja XML-tietokantoja tarvitaan?

XML on luonteeltaan tiedonvaihtoformaatti, joka sopii hyvin kutakuinkin kaikille erilaisille sovellusaloille kuvaamaan tietoa (Abiteboul & al., 2011, s.15). Kuvassa 4 on havainnollistettu kuinka XML-datasta saadaan jalostettua tietoa erilaisia tarkoituksia varten: sovellus A sisältää XML-dataa, jota hallinnoidaan jollain tiedonhallintaohjelmistolla. Sovelluksen A sisältämästä datasta voidaan muuntaa tietoa haluttuun muotoon sovellukseen B.



Kuva 4. Tiedonkulku XML-pohjaisessa tiedonvaihdossa (Abiteboul & al., 2011, s. 15).

Sovelluksen B käyttämä muoto voi olla esimerkiksi verkkoselaimella katseltavissa oleva HTML-sivu.

Kuvan 4 sovellus A voisi olla esimerkiksi relaatiotietokantajärjestelmä, sillä useat relaatiotietokantajärjestelmät tukevat XML-dokumenttien tallentamista tietokan-

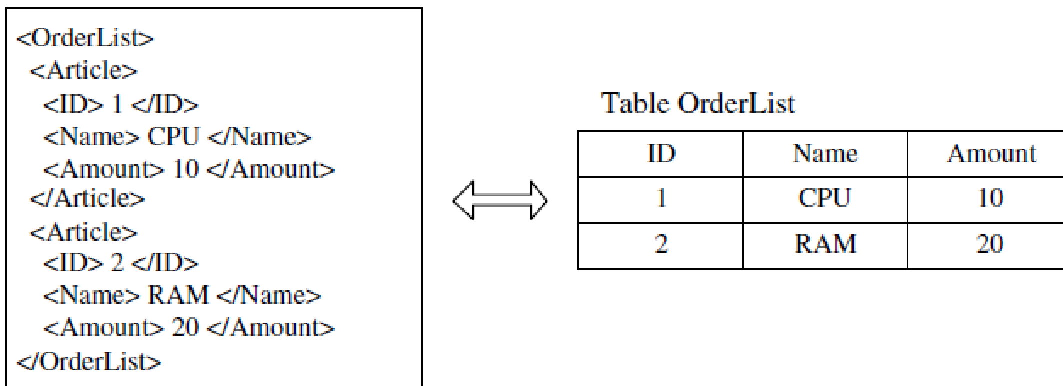
toihinsa. XML-dokumentin tallentaminen relaatiotietokantaan vaatii kuitenkin aina lisätyötä XML-dokumentin ja relaatiotietokannan rakenteen erilaisuudesta johtuen: XML-dokumentti muodostaa puurakenteen, kun taas relaatiotietokanta perustuu taulujen riveihin. Tästä erilaisuudesta johtuen XML-dokumentti on aina ensin muokattava relaatiotietokannalle sopivaan muotoon (Chuang & al., 2006). XML-muotoisen ja relaatiotietokannan käyttämän rakenteen erilaisuus perustuu siihen, että relaatiotietokantoja kehitettäessä ei XML-kieltä ollut vielä olemassakaan.

Mitä monimutkaisempi XML-dokumentti, sitä vaikeampaa on sen rakenteen siirtäminen alkuperäisessä muodossaan relaatiotietokantaan. Sama ongelma on läsnä myös tiedonsiirrossa toiseen suuntaan, eli alkuperäisen XML-datan palauttaminen alkuperäisessä muodossaan relaatiotietokannasta voi olla haastavaa: XML-dokumentista voi hävitä (muunnostekniikoista riippuen) muunnoksen yhteydessä erilaisia tietoja, kuten prosessointiohjeet, kommentit ja etenkin elementtien keskinäinen järjestys.

Relaatiotietokantaan luodaan yleensä useita eri tauluja XML-datan tallentamisen yhteydessä. Relaatiotietokannasta XML-datan hakeminen ja palauttaminen alkuperäiseen muotoon voi vaatia useita liitosoperaatioita näiden luotujen taulujen välillä, mikä puolestaan voi johtaa hitaaseen suorituskykyyn. Myöskään XML-kyselykieliä ei voi käyttää relaatiotietokannassa suoraan, vaan niiden käskyt on aina ensin muokattava relaatiotietokannoille ominaiseen SQL-muotoon.

Tiedon rakenteen muuntamiseen XML-muotoisen datan ja relaatiotietokannan välillä on olemassa useita erilaisia tapoja. Yksi tällainen tapa on tauluihin pohjautuva kuvaus (table-based mapping), jonka muuntamismenetelmää havainnollistaa kuva 4. Tauluihin pohjautuvassa kuvauksessa tietokantaan luodaan taulu, joka saa nimensä XML-dokumentin juurielementin mukaan. Tähän tauluun tehdään rivi kaikista juurielementin lapsielementeistä siten, että rivin attribuuteiksi tulee näiden lapsielementtien lapsielementit. Esimerkiksi kuvassa 5 luodaan taulu `OrderList`, jonka kukin rivi kuvaa `Article`-elementin sisällön. Kunkin rivin attribuutteina ovat `Article`-elementin lapsielementit eli `ID`, `Name` ja `Amount`. Tauluihin pohjautuva kuvaus toimii kuitenkin vain silloin kun XML-tiedon rakenne muistuttaa relaatiotaulua. Mo-

nimutkaisempien rakenteiden omaavien XML-dokumenttien esittäminen ei täten onnistu taulupohjaisen kuvauksen avulla.



Kuva 5. Tauluihin pohjautuvan kuvauksen mukainen muutos XML-datan ja relaatiotietokannan taulun välillä (Chuang & al., 2006).

Myös XML-elementtien hakeminen sijainnin perusteella on hankalaa relaatiotietokantaympäristössä: Esimerkiksi neljännen sellaisen `kappale`-elementin, joka sisältää tietyn avainsanan, hakeminen SQL-kielellä `kappale`-elementeistä koostuvasta XML-datasta voi olla hankalaa. Vastaavan haun suorittaminen puolestaan natiivissa XML-tietokantaympäristössä onnistuisi kätevästi XQueryllä.

Lisäksi myös XML-kuvauskielten huomioonottaminen on haasteellista relaatiotietokannoissa. Sen sijaan natiivit XML-tietokannat pystyvät hallitsemaan kätevästi myös XML-kuvauskieliä ja niiden muutoksia, kuten myös XML-dokumentteja, joiden rakennetta ei ole määritelty millään XML-kuvauskielellä. Relaatiotietokantojen käyttö XML-dokumenttien hallinnassa aiheuttaa siis aina enemmän tai vähemmän ongelmia, eikä täten ole tehokkuuden ja käytännöllisyyden puutteiden takia optimaalista (Chaudhri & al., 2003, s. 21).

3.2 Mitä natiivit XML-tietokannat ovat?

Natiivit XML-tietokannat tarjoavat tehokkaan ratkaisun XML-tiedon hallintaan. Natiivit XML-tietokannat ovat tietokantoja, joiden tarkoituksena on tarjota mahdollisimman tehokas XML-muotoisen tiedon hallintaympäristö. Natiivissa XML-tietokannassa XML-dataa voidaan tallentaa tai hakea alkuperäisessä XML-muodossa

ilman ylimääräisiä muunnoksia (Chaudhri & al., 2003, s. 21). XML:ään pohjautumista lukuun ottamatta natiivit XML-tietokannat kuitenkin muistuttavat relaatiotietokantoja, sillä myös natiiveilla XML-tietokannoilla on relaatiotietokannoille tyypillisiä ominaisuuksia kuten indeksointi, kyselyjen evaluointi, ohjelmointirajapinnat ja transaktiot (Bourret, 2005).

XML-dokumenttien rakenteen monimutkaisuus riippuu kulloinkin tarvittavasta käyttötarkoituksesta. XML:ää käytetäänkin sen joustavuuden ansiosta useilla erilaisilla sovellusaloilla. XML:n avulla kuvataan esimerkiksi bioinformatiikassa proteiinisekvenssejä, jotka voivat muodostaa erittäin suuria, useiden gigatavujen kokoisia yksittäisiä dokumentteja. Toisen ääripään esimerkkinä mainittakoon XML:n käyttö tiedonvaihdossa Internetin välityksellä esimerkiksi SOAP-teknologian avulla, jossa käytetään puolestaan satoja tuhansia pieniä erillisiä XML-dokumentteja. Näiden esimerkkien väliin mahtuu valtavasti erilaisia käyttötarkoituksia XML:lle, mikä tarkoittaa sitä, että XML:ää käytetään hyvin erilaisten rakenteiden omaavien dokumenttien yhteydessä (Mathis, 2009). Natiivien XML-tietokantojen tulisikin tarjota yhtä hyvää suorituskykyä ja toiminnallisuutta kaiken tyyppisille XML-dokumenteille. XML-tietokantahallintajärjestelmän tulisi lähtökohtaisesti tarjota kahdeksan seuraavaa ominaisuutta:

- Tehokas tiedon varastointi ja palauttaminen tietokannasta.
- Navigointioperaatiot, joita tarvitaan halutun tiedon paikantamiseen tietokannasta.
- Kyselyjen evaluointi.
- Modifiointimahdollisuus tietokantaan talletetun XML-datan muuttamiseksi.
- Ns. kiertomatka-ominaisuus (Round-trip property), joka takaa, että tietokantaan säilötty XML-dokumentti voidaan palauttaa tietokannasta ilman minkäänlaisia tiedon menetyksiä.

- Tuki sekä yksittäisten XML-dokumenttien että useiden dokumenttien muodostamien kokoelmien hallinnalle.

- Tiivis ja tilatehokas dokumenttien varastointi, jonka tarkoitus on säästää ulkoisia muistikuluja sekä vähentää oheismuistin käyttöä ja täten johtaa parempaan suorituskykyyn.

- Tuki indeksoinnille.

Tehokas tiedon tallentaminen ja palauttaminen on tärkein lähtökohta natiivin XML-tietokannan toiminnalle, sillä tietokannat joutuvat jatkuvasti vastaanottamaan ja lähettämään dataa. Pullonkaulailmiön välttämiseksi datan tallennuksen tietokantaan sekä datan palautuksen tietokannasta on siis oltava tehokasta.

Tuki sekä dokumenttien että kokoelmien hallintaan viittaa siihen, että tietokannan XML-data voi olla tietokannassa joko yksittäisinä XML-dokumentteina tai usean dokumentin muodostamina kokoelmina. Tietokannan on pystyttävä hallitsemaan dataa yhtä tehokkaasti, olipa se kummassa muodossa tahansa.

Indeksoinnilla halutaan puolestaan tehostaa tietokannan toimintaa. Indeksointi on hyvin oleellinen osa natiiveja XML-tietokantoja (kuten myös relaatiotietokantoja), sillä ilman indeksointia suurten datamäärien prosessointi olisi huomattavasti hitaampaa. Indeksointi auttaa rajaamaan hakuavaruutta XML-kyselyjen suorittamisessa. Natiiveille XML-tietokannoille tyypillisistä indeksointitavoista kerrotaan luvussa 4.1.

Indeksointi on eräs keino parantaa tietokannan suorituskykyä. Toinen hyvin oleellinen asia tietokannan suorituskyvyn tehokkuuden kannalta on oheismuistiin kohdistuvien I/O-operaatioiden (eli syöttö- ja tulostus-operaatioiden) lukumäärän minimointi. Hyvin suuret XML-dokumentit eivät välttämättä mahdu keskusmuistiin ladattaviksi, jolloin joudutaan käyttämään I/O-operaatioita. Oheismuistiin kohdistuvat I/O-operaatiot ovat hyvin paljon hitaampia kuin datan operointi keskusmuistissa. Tästä syystä levyoperaatioiden määrä halutaan minimoida (Abiteboul & al., 2011, luku 4).

Tavoista, joilla tähän tavoitteeseen päästään, kuin myös yleisesti natiivien XML-tietokantojen suorituskyvyn parantamisesta, kerrotaan tarkemmin luvussa 4.

Natiivin XML-tietokannan ominaispiirteenä on vaatimus tarjota toiminnallisuutta useilla eri käsittelykielillä, koska XML:lle on olemassa useita yleisesti käytettyjä erilaisia kieliä (Haustein & Härder 2007). Luvussa 2.4 esitellyt XPath ja XQuery ovat tärkeimmät kielet natiivissa XML-tietokantaympäristössä. Natiivit XML-tietokannat tarjoavat kuitenkin usein näiden lisäksi myös DOM- ja SAX-rajapinnat. DOM (Document Object Model) on W3C:n kehittämä kieliriippumaton sovellusrajapinta XML-dokumenttien käsittelyyn (W3C, 2005). DOM kuvaa XML-dokumentin puurakenteena, josta haluttujen dokumentin osien käsittely on kätevää, mutta resursseja kuluttavaa, sillä XML-dokumentti täytyy yleensä ladata kokonaisuudessaan muistiin ennen kuin sitä voidaan käsitellä. SAX (Simple API for XML) (Megginson, 2004) on puolestaan tapahtumapohjainen rajapinta XML-datan käsittelyyn. Tapahtumapohjaisuus tarkoittaa sitä, että SAX lukee XML-dokumentin tapahtumien virtana, josta saadaan ”vedettyä” halutut dokumentin osat käsittelyä varten. SAX ei siis lataa koko XML-dokumenttia kerralla muistiin, mikä johtaa DOMia parempaan tehokkuuteen tiedostopohjaisissa ratkaisuissa.

Transaktioidenhallinta on puolestaan kaikenlaisille tietokannoille ominainen vaatimus. Transaktioiden avulla tietokanta pystyy hallitsemaan useita samanaikaisia käyttäjiä sekä toipumaan häiriöistä. Transaktioidenhallinnasta natiiveissa XML-tietokannoissa kerrotaan tarkemmin luvussa 5.

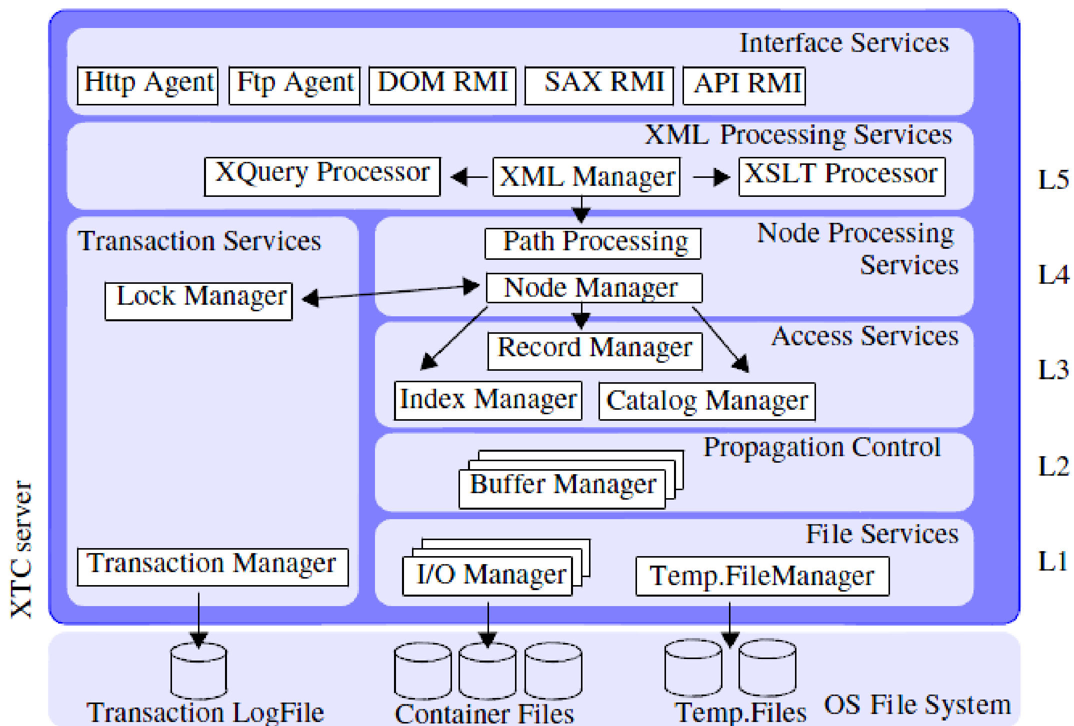
3.3 Natiivien XML-tietokantojen arkkitehtuurit

Natiivin XML-tietokannan rakenne perustuu yleensä karkeasti ottaen kolmeen kerrokseen, jotka ovat varastointikerros, palvelukerros ja rajapintakerros. Varastointikerros on kerroksista alin ja sen tehtävänä on tarjota tehokas ja luotettava säilytys tietokannan sisältämälle datalle. Varastointikerroksen yläpuolella on palvelukerros, joka vastaa tietokannan toiminnallisuuksista, kuten kyselyjen evaluoinnista sekä transak-

tioidenhallinnasta. Kerroksista ylimpänä on rajapintakerros, joka toimii tietokannan ja erilaisten sovellusten kommunikointikanavana (Fiebig & al., 2002).

Seuraavaksi esitellään XTCserverin arkkitehtuuri, joka näkyy kuvassa 6. XTCserver kuuluu Kaiserslauternin yliopiston tutkimusprojektiin XTC (XML Transaction Coordinator), jonka tarkoituksena on tutkia natiivien XML-tietokannanhallintajärjestelmien toimintaa. XTCserver on natiivin XML-tietokannan prototyyppi, joka on kehitetty natiivin XML-tietokannan toiminnan, kuten esimerkiksi transaktioiden prosessoinnin tehokkuuden testaamista varten. XTCserverin arkkitehtuuri esitellään tässä tutkielmassa, koska se on selkeä ja hyvä esimerkki siitä minkä tyyppistä arkkitehtuuria tehokas natiivi XML-tietokanta vaatii.

XTCserverin arkkitehtuuri koostuu viidestä tasosta, mutta siitä huolimatta sen idea on samankaltainen edellä esitetyn kolmen kerroksen arkkitehtuurimallin kanssa, koska myös XTCserverin arkkitehtuurissa alimpana toimii varastointikerros, jonka jälkeen tulevat kerrokset vastaavat tietokannan toiminnallisuuksista (palvelukerros) ja ylimpänä arkkitehtuurissa on rajapintapalvelut. XTCserverissä ja useissa muissa tietokantojen arkkitehtuurimalleissa pohjalla on siis pysyvään levymuistiin tallennettu suuri määrä bittejä, jotka tietokannanhallintajärjestelmä muuntaa mielekkääseen muotoon, jota puolestaan tietokannan käyttäjä voi käsitellä. Toisin sanoen, mitä ylemmäksi arkkitehtuurissa mennään, sitä monipuolisemmiksi arkkitehtuurin tason tarjoamat toiminnallisuudet muuttuvat (eli fyysisestä tallennuksesta aina XML-prosessointi- sekä rajapintapalveluihin asti) (Haustein & Härder, 2007).



Kuva 6. XTCserverin arkkitehtuuri (Haustein & Härder, 2007).

XTCserverin kerros L1 vastaa tiedon varastoinnista, eli sen avulla tietokannasta voidaan lukea tai kirjoittaa raakaa dataa, jonka varsinainen tulkinta toteutetaan ylemmissä kerroksissa. Kerros L2 vastaa puskuroinnista (buffering): tietokantaympäristössä puskuri tarkoittaa datajoukkoa, joka koostuu pysyvässä muistissa olevasta datasta sekä keskusmuistiin ladatusta datasta. Puskurinhallintaa tarvitaan, koska tietokannanhallintajärjestelmän pitää tietää mitkä osat datasta on pysyvässä muistissa ja mitkä puolestaan vain keskusmuistissa (Sciore, 2008, luku 13). Tietokannan pitää pystyä varautumaan siihen, että keskusmuistissa oleva data häviää, mikäli koneeseen tulee jokin häiriö. Tämä varautuminen voidaan toteuttaa transaktioiden lokitiedoston avulla. Transaktioiden lukkojen hallinnalla (Lock Manager -komponentti kuvassa 6) voidaan puolestaan hallita useiden samanaikaisten käyttäjien toimintaa. Transaktioiden toimintaan paneudutaan tarkemmin luvussa 5.

Kerroksissa L1 ja L2 voidaan hyödyntää jossain määrin relaatiotietokannoissa tehokkaiksi osoittautuneita tekniikoita, toisin kuin ylemmissä kerroksissa, jotka vaativat vain XML:lle sopivia toteutustekniikoita. Kerroksen L3 tarkoituksena on tarjota indeksointia hyödyntävä nopea XML-solmujen haku, jota tarvitaan esimerkiksi navi-

goitaessa tiettyyn XML-dokumentin solmuun. Solmujen tunnistamista helpottavista yksikäsitteisistä solmujen nimeämiskäytännöistä kuin myös indeksoinnista kerrotaan tarkemmin seuraavassa luvussa.

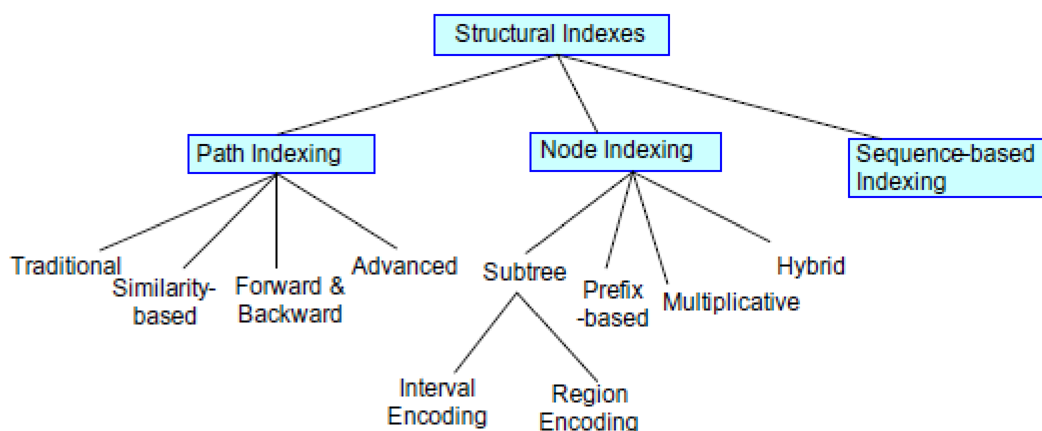
Kerroksen L4 tärkein tehtävä on muuntaa alemmilta tasoilta tulevaa raakadataa XML-muotoisen datan esittämiseen. Lopuksi ylin kerros L5 tarjoaa mahdollisuuden prosessoida tätä XML-dataa XML-käsittelykielillä. Kerrokseen L5 kuuluu oleellisena osana kyselyjen evaluointi, jonka tarkoituksena on valita mahdollisimman tehokas tapa toteuttaa käyttäjän pyytämä kysely. Kyselyjen evaluoinnista kerrotaan lisää seuraavassa luvussa.

4 Natiivien XML-tietokantojen suorituskyvyn optimointi

Yleisesti tietokannat toimivat suurten tietomäärien hallintatyökaluina. Tästä johtuen tietokannoissa tarvitaan tekniikoita, joiden avulla säilötystä tiedosta saadaan haettua halutut tiedot mahdollisimman tehokkaasti. Yksi merkittävimmistä tietokannan suorituskykyä parantavista tekniikoista on indeksointi, jonka avulla tietokanta pystyy rajaamaan hakuavaruutta tiedonhaussa. XML-muotoisen tiedon yleiset indeksointimenetelmät esitellään luvussa 4.1. Luvussa 4.2 käydään puolestaan läpi kyselyjen evaluoinnin teoriaa natiiveissa XML-tietokannoissa, hyödyntäen osaa luvun 4.1 indeksointimenetelmistä.

4.1 Indeksointi

Indeksoinnin tarkoituksena on tehostaa tietokannan toimintaa rajaamalla kyselyjen hakuavaruutta. Indeksointia on tutkittu laajalti relaatiotietokantajärjestelmissä jo kauan ennen XML-kielen kehittämistä. Relaatiotietokantojen käyttämiä indeksointimalleja ei voida kuitenkaan suoraan kopioida natiiveihin XML-tietokantoihin XML-datan rakenteisuudesta johtuen. XML-muotoiselle tiedolle ominaiset indeksointimallit voidaan luokitella kolmeen pääryhmään: polkuindeksointiin, solmuindeksointiin sekä sekvenssipohjaiseen indeksointiin. Nämä pääluokat sisältävät useita erilaisia tekniikoita, jotka Haw & Lee (2011) luokittelevat kuvassa 7 esitettyihin aliluokkiin.



Kuva 7. Rakenteisen tiedon indeksointimenetelmien luokittelu (Haw & Lee, 2011).

Tässä luvussa esitellään pääpiirteittän kuvan 7 indeksointiluokat. Polku- ja solmuindeksointia hyödyntävät käytännön esimerkit esitellään luvussa 4.2.

4.1.1 Polkuindeksointi

Polkuindeksoinnin (*path indexing*) perusideana on ylläpitää erikseen tietoa XML-datan poluista. Haw & Lee (2011) ryhmittelevät polkuindeksointitekniikat perinteisiin, samankaltaisuuksia hyödyntäviin, eteen- ja taaksepäin linkittäviin sekä kehittyneisiin polkuindeksointitekniikoihin. Perinteisen polkuindeksoinnin avulla voidaan oleellisesti tehostaa sellaisten kyselyiden toimintaa, joiden tarvitsee käsitellä vain yksittäistä XML-puun haaraa. Useita XML-puun haaroja yhdistävät kyselyt sen sijaan vaativat tehokkuutta heikentäviä liitosoperaatioita (Zou & al., 2004). Perinteiset polkuindeksointitekniikat pitävät yllä eksplisiittisesti tietoa jokaisesta juurisolmusta lähtevästä polusta. Tämä johtaa suureen indeksin kokoon, koska indeksissä on oltava viittaus jokaiseen tietokannan solmuun. Suuri tilavaatimus on tyypillinen ongelma perinteisten polkuindeksointitekniikoiden lisäksi myös muille polkuindeksointimenetelmille.

Samankaltaisuuksiin pohjautuvat polkuindeksointitekniikat jakavat XML-puun solmut ryhmiin jonkin ominaisuuden perusteella. Ryhmittelyn avulla indeksi saadaan jaettua pienempiin osiin. Tämä ryhmittely johtaa mahdollisesti siihen, että kyselyn suorittamisessa riittää käydä läpi ainoastaan jokin näistä osista, eikä koko indeksiä. Ryhmittelyä hyödyntävästä polkuindeksoinnista käydään läpi tarkempi esimerkki luvussa 4.2.1. Samankaltaisuuksiin perustuvissa polkuindeksointitekniikoissa on lisäksi mahdollista ottaa huomioon yksittäisten polkujen kuormitus. Tämän tiedon avulla indeksiä voidaan sopeuttaa tukemaan paremmin sellaisia kyselyitä, jotka käyttävät toistuvasti samoja polkuja.

Eteen- ja taaksepäin linkittävän indeksoinnin tarkoituksena on solmujen molemminsuuntaisten linkitysten avulla tarjota tehokkaampaa ratkaisua useita XML-puun haaroja käyttäviin kyselyihin. Ideana on pienentää kyselyjen prosessointiaikaa ulkoisten muistikulujen kustannuksella. Eteen- ja taaksepäin linkittävässä indeksoinnissa in-

deksiin on siis tallennettava muita polkuindeksointimenetelmiä enemmän tietoa, mikä johtaa pahimmassa tapauksessa valtavan suureen tilavaatimukseen.

Kehittyneillä polkuindeksointitekniikoilla tarkoitetaan tässä yhteydessä viime vuosien aikana kehitettyjä, tai yhä kehitteillä olevia, polkuindeksointimenetelmiä, jotka eivät ole vakiintuneet laajaan käyttöön.

4.1.2 Solmuindeksointi

Solmuindeksoinnissa (*node indexing*) jokainen solmu indeksoidaan jollain ennalta määrätyllä numerointitavalla. Solmuindeksoinnin idea perustuu siihen, että kahden solmun välinen suhde XML-puussa on mahdollista selvittää vakioajassa vertaamalla solmujen indeksejä. Solmujen indeksejä vertaamalla voidaan vakioajassa päätellä esimerkiksi onko kahden solmun välillä vanhempi-lapsi-suhde. Kahden solmun vertailuun pohjautuvat menetelmät eivät kuitenkaan välttämättä tarjoa tehokasta ratkaisua kaikenlaisten kyselyjen suorittamiseen (Zou & al., 2004). Haw & Lee (2011) jakavat solmuindeksointitekniikat neljään aliluokkaan, jotka ovat alipuu-perustaiset, etuliite-perustaiset, multiplikatiiviset sekä hybridi-solmuindeksointimenetelmät.

Alipuu-perustaisissa solmuindeksointitekniikoissa kutakin XML-dokumentin solmua vastaavaan indeksiin tallennetun tiedon pohjalta on mahdollista päätellä solmun sijainti sekä solmun alipuun laajuus. Käytännössä tämä tarkoittaa sitä, että kahden solmun välinen esi-isä-jälkeläinen- tai vanhempi-lapsi-suhde voidaan selvittää vakioajassa. Tähän solmuindeksointitekniikkaan palataan tarkemmin luvussa 4.2.2 sijaintiin perustuvien solmutunnisteiden esittelyn yhteydessä.

Etuliite-perustaisissa solmuindeksointitekniikoissa solmun `v` tunnisteen etuliite nimetään juurisolmusta solmuun `v` kulkevan polun mukaan. Toisin sanoen solmun `v` tunniste saa etuliitteekseen kaikkien esi-isäsolmujensa tunnisteet. Tämän etuliitteen loppuun kirjataan solmun oma tunnus `v`. Etuliite-perustaisessa solmuindeksoinnissa solmu `u` voidaan päätellä solmun `v` esi-isäksi, mikäli solmun `u` solmutunniste on solmun `v` solmutunnisteen etuliite. Kahden solmun välisestä suhteesta on mahdollista päätellä myös muita tietoja etuliite-perustaisten solmutunnisteiden avulla. Näihin

palataan tarkemmin luvussa 4.2.2, jossa esitellään Dewey-luokitteluun perustuvat solmutunnisteet. Etuliite-perustaisten tekniikoiden haasteena on suuri tilavaatimus, koska solmun `v` tunnisteen koko kasvaa sitä suuremmaksi mitä kauempana `v` on juurisolmusta. Pahimman tapauksen tilavaativuus solmun `v` tunnisteelle on täten O(`n`), kun se esimerkiksi alipuu-perustaisissa olisi ainoastaan O(log `n`).

Multiplikatiivisissa solmuindeksointimenetelmissä solmujen välisten suhteiden selvittäminen perustuu solmutunnisteiden aritmeettisten ominaisuuksien laskemiseen. Ideana on löytää kuvaus jostain epäsäännöllisen rakenteisuuden omaavasta dokumentista D johonkin säännölliseen puurakenteeseen D_0 siten, että jotkut D_0:n aritmeettiset ominaisuudet ovat voimassa myös puussa D. Multiplikatiivisten solmutunnisteiden laskeminen on kuitenkin hidasta, joten multiplikatiivisten solmutunnisteiden käyttö ei ole suotavaa suurten tietomäärien hallinnassa.

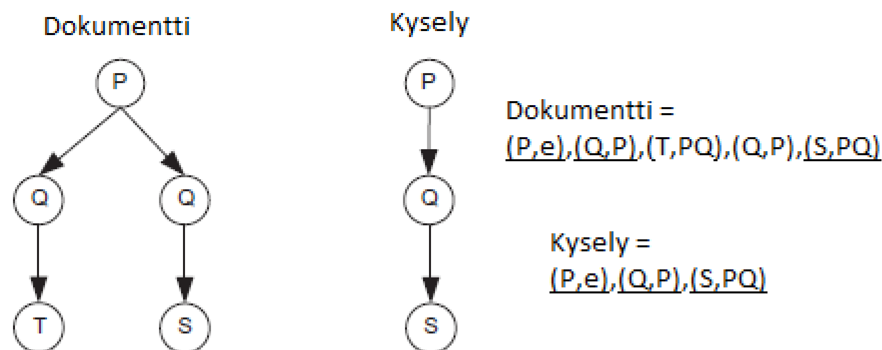
Hybridi-solmuindeksointimenetelmillä tarkoitetaan viime vuosien aikana kehitettyjä solmuindeksointimenetelmiä, joiden tarkoituksena on yhdistää muiden solmuindeksointimenetelmien hyviä puolia.

4.1.3 Sekvenssipohjainen indeksointi

Sekvenssipohjainen indeksointi (*sequence-based indexing*) muuntaa sekä XML-dokumentin että kyselyt sekvensseiksi. Sekvenssipohjaisen indeksoinnin avulla on mahdollista välttää kalliita liitosoperaatioita, koska kyselyjen evaluoimiseen riittää ainoastaan sekvenssien keskinäinen vertailu. Toisaalta virheettömien ja vertailukelpoisten sekvenssien muodostaminen XML:n puurakenteesta on haasteellista: sekvenssipohjaisen indeksoinnin tyypillisenä riskinä on antaa kyselyihin virheellisiä tuloksia, koska kahden sekvenssin välinen täsmääminen ei ole välttämättä sama asia kuin kahden alipuun täsmääminen XML-puussa (Zou & al., 2004).

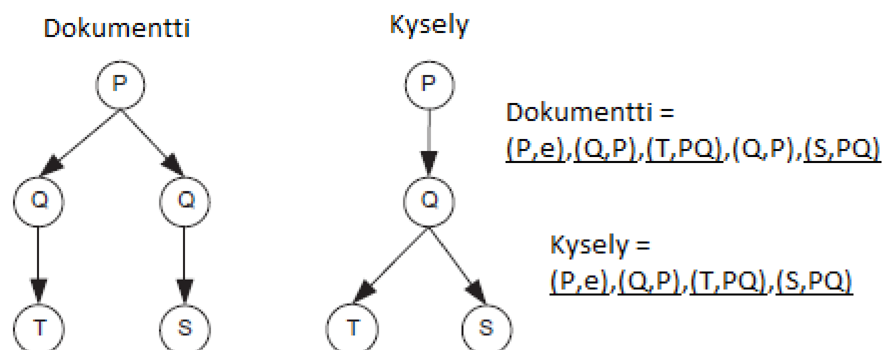
ViST (Virtual Suffix Tree) (Wang & al., 2003) on eräs sekvenssipohjainen XML-muotoisen tiedon indeksointimenetelmä. ViST-menetelmä muuntaa sekä dokumentin että kyselyn sekvenssiksi, joka koostuu solmu-polku-pareista. Kukin solmu-polku-pari kertoo solmun nimen sekä polun juurisolmusta kyseiseen solmuun. Kuvassa 8

on esitetty esimerkkidokumentti ja -kysely puurakenteina. Dokumenttia ja kyselyä vastaavat sekvenssit on kuvattu kuvan oikeassa reunassa. Kysely voidaan nyt suorittaa vertaamalla dokumentin ja kyselyn sekvenssejä: tavoitteena on löytää kyselyn sekvenssiä täsmäävät solmu-polku-parit dokumentin sekvenssistä. Kuvassa 8 on alleviivattu ne solmut, jotka löytyvät molemmista sekvensseistä. Tässä tapauksessa jokaiselle kyselyn solmulle löytyy vastine dokumentista, joten dokumentti sisältää kyselyn mukaiset solmut.



Kuva 8. Esimerkki ViST-menetelmästä (Haw & Lee, 2011).

Kuvassa 9 on havainnollistettu esimerkki virheellisestä tuloksesta.



Kuva 9. ViST-menetelmän virheellinen tulos (Haw & Lee, 2011).

ViST-menetelmän mukaan kuvan mukainen kysely tuottaisi hakutuloksen, vaikka todellisuudessa kysely ei täsmää dokumenttiin.

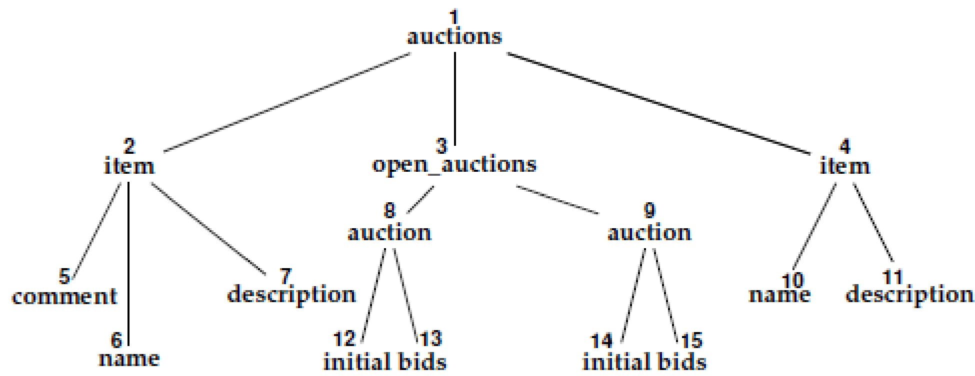
4.2 Kyselyjen evaluointi

Kyselyjen evaluoinnin tarkoituksena on suorittaa tietokannan vastaanottamat kyselyt mahdollisimman tehokkaasti. Kyselyiden suoritustehokkuuden kannalta olennainen asia on I/O-operaatioiden lukumäärä, joka halutaan minimoida (kuten luvussa 3.2 perusteltiin). Toisin sanoen tietokannan toiminta halutaan toteuttaa siten, että tietokannasta luetusta mahdollisimman pienestä datamäärästä saataisiin irti mahdollisimman paljon tietoa, jotta I/O-operaatioiden määrä pysyisi mahdollisimman pienenä. Tämän takia natiivin XML-tietokannan pitää pystyä säilömään XML-data sellaisessa järkevässä muodossa, joka minimoi levyliikenteen määrän. Mikäli XML-tieto on säilötty tietokantaan naiivilla tavalla huomioimatta evaluoinnin merkitystä, niin tietokanta joutuu suorittamaan kyselyitä navigoimalla oheismuistissa toistuvien I/O-operaatioiden merkeissä. Esimerkiksi yksittäisen XPath-jälkeläisten hakuoperaation `//` suorittaminen vaatisi tällöin XML-datan jokaisen solmun läpikäymistä, mikä olisi hyvin hidasta.

Abiteboul & al. (2011, luku 4) jakaa tehokkaan prosessoinnin mahdollistavat XML-datan säilömismenetelmät kahteen pääluokkaan: tiedon fragmentointiin ja monipuolisten solmutunnisteiden hyödyntämiseen.

4.2.1 Tiedon fragmentointi

XML-solmuja käsitellään tyypillisesti jonkin ominaisuuden (kuten nimen tai sijainnin) perusteella. Fragmentoinnin tarkoituksena on jakaa XML-data erilaisiin ryhmiin jonkin ominaisuuden perusteella. Nyt sellaiset solmut, joita haetaan usein samanaikaisesti, löytyvät ideaalitilanteessa samasta ryhmästä. Tämä johtaa siihen, että ideaalitilanteessa riittää prosessoida ainoastaan yksi tällainen ryhmä, eikä kaikkia XML-dokumentin solmuja. Toisin sanoen tiedon fragmentoinnin avulla voidaan pienentää hakuavaruutta ja siten myös vähentää oheismuistin käyttöä.



Kuva 10. XML-dokumentti, jonka solmut on numeroitu leveysjärjestyksessä (Abiteboul & al., 2011, s. 93).

Kuvassa 10 on esimerkki XML-dokumentista, jonka solmut on numeroitu leveysjärjestyksessä. Tällaisesta dokumentista voidaan muodostaa kokoelma kaarista esimerkiksi siten, että kullakin kaarella on ominaisuuksina vanhempi-solmun tunniste (`pid`), lapsisolmun tunniste (`cid`) sekä lapsisolmun nimi (`clabel`). Tämä kokoelma voidaan kuvata relaatioksi `Edge(pid, cid, clabel)`, jonka sisällöstä osa on havainnollistettu kuvassa 11.

pid	cid	clabel
-	1	auctions
1	2	item
2	5	comment
2	6	name
2	7	description
1	3	open_auctions
3	8	auction
...	...	...

Kuva 11. Osa Edge-relaation sisällöstä (Abiteboul & al., 2011, s. 93).

Käytettäessä relaatiotauluja apuna, voidaan XML-kyselyjen evaluointi toteuttaa muuntamalla ensin XML-kysely relaatiokyselyksi ja tämän jälkeen evaluoimalla ko. relaatiokysely. Kuvassa 11 havainnollistetun kaariryhmittelyn myötä relaation `Edge` avulla voidaan nyt kätevästi toteuttaa kysely, joka sisältää yksittäisen jälkeläisten hakuoperaation (eli XPathin `//`-operaation). Esimerkiksi kysely `//initial` voitaisiin nyt toteuttaa yksinkertaisesti hakemalla `Edge`-relaatiosta ne rivit, joissa `clabel=initial`, eli evaluoimalla relaatioalgebran lauseke $\pi_{cid}(\sigma_{clabel=initial}(Edge))$. Nyt `Edge`-relaation `clabel`-attribuutin indeksin

avulla relaatiosta voitaisiin hakea suhteellisen tehokkaasti kaikki `initial`-rivit käymättä jokaista relaation riviä läpi.

Kaariryhmittely on kuitenkin tehoton toteuttamaan sellaisia kyselyitä, jotka sisältävät jälkeläisten hakuoperaation `//` muualla kuin kyselyn alussa. Esimerkiksi kyselyssä `//auction//bid` haastelliseksi muodostuisi `bid`-jälkeläisten haku, koska haettavien `auction`-solmujen ja näiden `bid`-jälkeläissolmujen etäisyys (eli `auction`- ja `bid`-solmujen välisten solmujen lukumäärä) voi olla XML-puussa kuinka suuri tahansa. Tämä johtaisi siihen, että tietokanta joutuisi evaluoimaan rajoittamattoman suuren määrän kyselyjen yhdisteitä.

Seuraavista esimerkeistä huomataan, että kaariryhmittely on tehokas ainoastaan yksittäisten `//`-kyselyjen toteutuksessa. Esimerkiksi kysely `/auctions/item` kääntyisi kaariryhmittelyn pohjalta relaatioalgebramuotoon $\pi_{cid}((\sigma_{clabel=auctions}(Edge)) \bowtie_{cid=pid} (\sigma_{clabel=item}(Edge)))$.

Vastaavasti kysely `/auctions/item/description` kääntyisi muotoon $\pi_{cid}((\sigma_{clabel=auctions}(Edge)) \bowtie_{cid=pid} (\sigma_{clabel=item}(Edge)) \bowtie_{cid=pid} (\sigma_{clabel=description}(Edge)))$. Huomionarvoista näissä esimerkkikyselyissä on kalliit liitosoperaatiot `Edge`-relaatioiden välillä. Ensin mainitussa esimerkissä on liitettävä kaksi `Edge`-relaation esiintymää. Vastaavasti jälkimmäisessä esimerkissä liitoksia tarvitaan kolmen `Edge`-relaation välillä. Yleisesti tämän tyyppisissä kyselyissä tarvittaisiin siis useita tehokkuuden kannalta kalliita liitosoperaatioita `Edge`-relaatioon. Kalliiden liitosoperaatioiden myötä kyselyjen suorittamisesta tulisi käytännössä liian hidasta. Edellä esitettyyn kaairyhmittelyyn tarvitaan siis parannusta, jotta monenlaisia kyselyitä voitaisiin evaluoida tehokkaasti.

Tarkennetaan seuraavaksi kaariin perustuvaa kuvausta jakamalla kaarirelaatio XML-datassa käytetyn sanaston perusteella, eli käyttämällä nimiryhmittelyä (tag-partitioning). Nyt kukin relaatio sisältää attribuutteinaan ainoastaan kaaren vanhempi-solmun tunnisteen sekä kaaren lapsisolmun tunnisteen. Lapsisolmun nimeä ei siis tarvitse enää tallentaa erikseen relaatiotaulun joka riville, koska kunkin relaation jo-

kaisella kaarella on sama lapsisolmun nimi. Nimiryhmittelyä havainnollistaa kuva 12.

auctionsEdge

pid	cid
-	1

itemEdge

pid	cid
1	2
1	4

open_auctionsEdge

pid	cid
1	3

auctionEdge

pid	cid
3	8
3	9

Kuva 12. Osa nimiryhmittelyn relaatioista (Abiteboul & al., 2011, s. 94).

Nimiryhmittelyn käyttö vähentää levyliikennettä haettaessa tietyn nimisiä solmuja, koska nyt solmut on jaettu nimikohtaisiin ryhmiin. Tietyn yksittäisen ryhmän läpikäyminen on paljon nopeampaa verrattuna alkuperäiseen kuvan 11 mukaiseen kaarirelaatioon. Tämäkään tapa ei kuitenkaan tarjoa tehokasta ratkaisua sellaisen kyselyn suorittamiseen, jossa //-operaatio sijaitsee muualla kuin lausekkeen alussa. Tällaisten kyselyjen evaluointia voidaan parantaa jakamalla ryhmät siten, että jokaiselle erilliselle vanhempi-lapsi-suhteelle muodostetaan oma relaatio. Näiden lisäksi tarvitaan vielä yksi ylimääräinen relaatiotaulu `path`, joka sisältää kaikki XML-dokumentin erilliset polut. Tätä polkuryhmittelyyn (path-partitioned) perustuvaa menetelmää havainnollistaa kuva 13.

/auctions

pid	cid
-	1

/auctions/item

pid	cid
1	2
1	4

/auctions/item/name

pid	cid
2	6
4	14

Paths

path
/auctions
/auctions/item
/auctions/item/comment
/auctions/item/name
...

Kuva 13. Osa polkuryhmittelyn relaatioista (Abiteboul & al., 2011, s. 95).

Nyt kyselyt, jotka sisältävät //-operaation muualla kuin lausekkeen alussa, voidaan evaluoida suhteellisen tehokkaasti seuraavalla tavalla:

1. Haetaan `path`-relaatiosta ne polut, jotka sisältävät kyselyyn täsmäävät esi-isä-jälkeläinen-suhteet.

2. Käydään läpi ensimmäisen kohdan polkuja vastaavat relaatiotaulut.

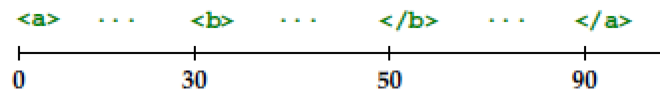
Ensimmäisen kohdan ansiosta `//`-operaatioiden evaluointi on nyt huomattavasti yksinkertaisempaa. Tällainen polkuryhmittelyä hyödyntävä menetelmä on sovellus luvussa 4.1.1 esitellystä samankaltaisuuksiin pohjautuvasta polkuindeksointitekniikasta. Kuten polkuindeksointimenetelmissä yleensä, myös tässä toteutustavassa sellaisten kyselyjen evaluointi, jotka vaativat useita XML-puun haaroja, vaatii kalliita liitosoperaatioita.

4.2.2 Solmutunnisteiden hyödyntäminen

Solmutunnisteita tarvitaan, jotta kukin XML-datan solmu voidaan yksilöidä ja täten erottaa muista solmuista. Solmutunnisteiden avulla solmuista saadaan lisäksi selville lisätietoja niiden keskinäisistä suhteista. Fragmentoitu XML-dokumentti on mahdollista rekonstruoida eli palauttaa alkuperäiseksi XML-dokumentiksi näiden suhteiden perusteella. Tämän takia onkin erittäin hyödyllistä tallettaa solmutunnisteisiin tietoa elementin sijainnista alkuperäisessä XML-dokumentissa. Seuraavaksi esiteltävät asiat perustuvat luvussa 4.1.2 esiteltyyn solmuindeksointiin.

Yleensä tunnisteiden nimeämiskäytännöt perustuvat XML:n puurakenteen hyödyntämiseen. Abiteboul & al. (2011, luku 4) jakaa solmutunnisteet kahteen eri päätyyppiin: sijaintiin perustuviin tunnisteisiin (region-based identifiers) ja Dewey-luokitteluun perustuviin tunnisteisiin. Sijaintiin perustuvat tunnisteet kuuluvat luvun 4.1.2 solmuindeksointiluokituksessa alipuu-perustaisiin solmuindeksointitekniikoihin. Dewey-luokittelun mukaiset solmutunnisteet kuuluvat puolestaan etuliiteperustaisten solmuindeksointitekniikoiden ryhmään.

Seuraavaksi esitellään sijaintiin perustuvat tunnisteet. Sijaintiin perustuvat tunnisteet käyttävät merkintöjä, jotka kertovat XML-elementin alku- sekä päättymiskohdan. Kuva 14 havainnollistaa sijaintiin perustuvien tunnisteiden toimintaa: kyseessä on kuvaus XML-dokumentista, joka koostuu `a`- ja `b`- elementeistä.



Kuva 14. Elementtien aloitus- ja lopetusmerkintöjen sijainti XML-tiedostossa (Abiteboul & al., 2011, s. 96).

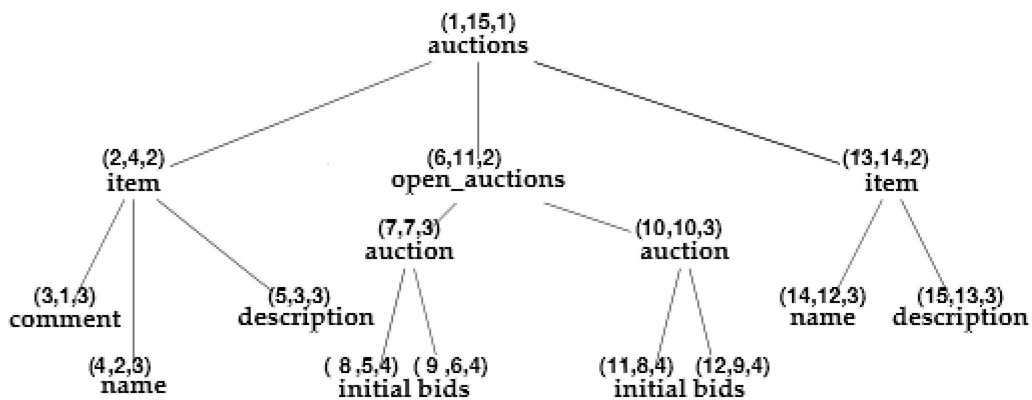
Kuvan numerot kertovat XML-tiedostosta kohdan, josta kukin elementin aloitus- tai lopetusmerkintä löytyy. Kuvassa 14 juurielementti a alkaa kohdasta 0 ja se sisältää aluksi tekstiä, kunnes kohdassa 30 on elementin b aloitusmerkintä. Elementti b on elementin a lapsielementti, joka sisältää tekstiä ja päättyy kohdassa 50. b-elementin jälkeen XML-dokumentti sisältää tekstiä kunnes se päättyy a-elementin lopetusmerkintään kohdassa 90. XML-dokumentin hyvinmuodostuneisuuden perusteella elementin aloitusmerkintä seuraa aina esi-isiensä aloitusmerkintöjä. Vastaavasti elementin lopetusmerkintä edeltää aina esi-isiensä lopetusmerkintöjä. Sijaintiin perustuvien tunnisteiden ideana on siis liittää jokaiseen XML-solmuun tieto siitä, missä kohdassa kyseinen solmu alkaa ja päättyy. Sijaintiin perustuvien tunnisteiden avulla nähdään helposti onko solmu n_1 solmun n_2 esi-isä vertaamalla somujen solmutunnisteita. Olkoon x.start solmun x aloituskohta ja vastaavasti x.end solmun x päättymiskohta. Nyt solmu n_1 on solmun n_2 esi-isä, jos n_1.start < n_2.start ja n_2.end < n_1.end. Kuvassa 14 solmun a solmutunniste on (0,90). Solmun b solmutunniste on puolestaan (30,50). Solmu a voidaan nyt siis päätellä vakioajassa solmun b esi-isäksi, koska a.start < b.start sekä b.end < a.end.

Tehokas kyselyjen evaluointi ei kuitenkaan vaadi XML-tiedoston tekstisisällön suuruuden laskemista kuten kuvassa 14 on tehty. Sen sijaan sijaintiin perustuvissa solmutunnisteissa riittää tieto siitä, miten elementit ovat suhteessa toisiinsa XML-puussa. Tämän johdosta sijaintiin perustuvina solmutunnisteina riittää käyttää yksinkertaisempaa (aloitusmerkintä, lopetusmerkintä)-notaatiota, jonka tarkoituksena on laskea kullekin elementille pelkistetymmät aloitus- ja lopetusmerkinnät siten, että elementtien teksisisällön merkkien määrää ei huomioida. Esimerkiksi kuvan 14 a-elementin solmutunniste olisi tällä tavalla määriteltynä (1,4) ja b-elementin vastaavasti (2,3). Näitäkin solmutunnisteita vertaamalla havaitaan, että solmu a on solmun b esi-isä, koska edelleen a.start < b.start ja b.end < a.end.

Sijaintiin perustuvien solmutunnisteiden merkintätavoista on olemassa erilaisia hieman toisistaan eroavia sovelluksia. Eräs näistä on (pre, post, depth)-merkintä, jonka avulla voidaan tarkistaa kätevästi mm. onko kahden solmun välillä vanhempi-lapsi-suhde. Merkintätavan (pre, post, depth) mukaiset solmutunnisteet on määritelty seuraavalla tavalla:

- pre: numeroidaan XML-puun solmut esijärjestyksessä
- post: numeroidaan XML-puun solmut jälkijärjestyksessä
- depth: lasketaan solmun syvyys (eli etäisyys) juurisolmuun nähden (juurielementin syvyys on 1)

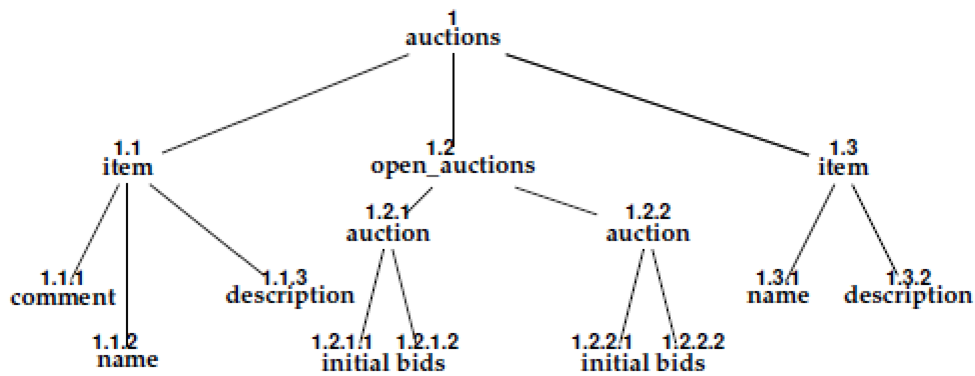
Esimerkki (pre, post, depth)-solmutunnisteista on esitetty kuvassa 15.



Kuva 15. Esimerkki (pre, post, depth)-solmutunnisteista (Abiteboul & al., 2011, s. 98).

Nyt solmu n_1 on solmun n_2 esi-isä mikäli $n_1.pre < n_2.pre$ ja $n_2.post < n_1.post$. Solmu n_1 on puolestaan solmun n_2 vanhempi jos n_1 on n_2:n esi-isä ja $n_1.depth = n_2.depth-1$.

Edellä esitellyt solmutunnisteiden nimeämiskäytännöt kuuluivat sijaintiin perustuviin solmutunnisteisiin. Seuraavaksi esitellään Dewey-pohjaisen luokittelun mukaiset solmutunnisteet. Dewey-pohjaisissa solmutunnisteissa XML-solmu v saa solmutunnisteensa etuliitteeksi vanhempi-solmun solmutunnisteen. Tämän etuliitteen loppuun lisätään solmun v oma tunnus. Kuvassa 16 on havainnollistettu Dewey-luokitteluun perustuva solmujen nimeämiskäytäntö.



Kuva 16. Esimerkki Dewey-solmutunnisteista (Abiteboul & al., 2011, s. 98).

Dewey-solmutunnisteet sisältävät enemmän informaatiota kuin sijaintiin perustuvat solmutunnisteet. Olkoon Dewey-solmutunnisteet n_1 ja n_2 muotoa $n_1 = x_1.x_2....x_m$ ja $n_2 = y_1.y_2....y_n$. Tällaisten solmutunnisteiden perusteella voidaan päätellä seuraavat asiat:

- Solmu n_1 on solmun n_2 esi-isä, jos n_1 on n_2:n alkuosa. Tällöin n_1 on n_2:n vanhempi, mikäli $n = m + 1$.
- Solmu n_1 on solmua n_2 edeltävä sisarsolmu, jos $x_1.x_2....x_{m-1} = y_1.y_2....y_{n-1}$ ja $x_m < y_n$ (vastaavaan tapaan voidaan päätellä onko solmu n_1 solmua n_2 seuraava sisarsolmu).
- Solmujen n_1 ja n_2 alin yhteinen esi-isä saadaan laskemalla solmujen n_1 ja n_2 pisin yhteinen etuliite.

Dewey-solmutunnisteiden avulla saadaan siis vakioajassa paljon tietoa solmujen välisistä suhteista. Toisaalta Dewey-solmutunnisteet vaativat sijaintiin perustuvia solmutunnisteita enemmän tilaa: Dewey-solmutunnisteiden tilavaativuus on O(h), missä h on puun syvyys. Sijaintiin perustuvien solmutunnisteiden tilavaatimus on puolestaan O(log n), missä n on puun solmujen lukumäärä.

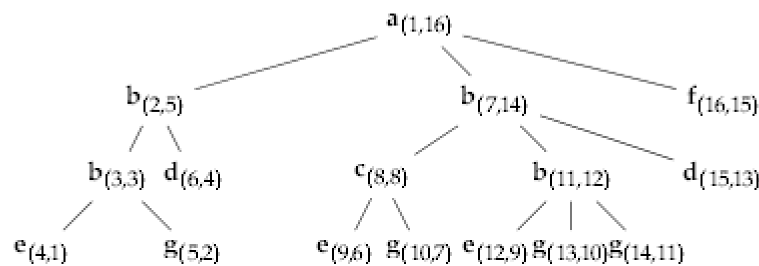
Olipa solmutunnisteina sijaintiin perustuvat solmutunnisteet tai Dewey-solmutunnisteet, niin solmutunnisteita joudutaan joka tapauksesssa päivittämään, jos XML-dataan lisätään uusi solmu. Mikäli tietokantaan lisätään uusi solmu n_{new}, niin

tietokannan täytyy päivittää vähintään kaikkien niiden solmujen tunniste, jotka esiintyvät XML-puussa solmun n_{new} jälkeen esijärjestyksen mukaisessa läpikäynnissä. Sen sijaan tietokannan solmujen tekstisisällön muokkaaminen ei vaadi päivityksiä solmutunnisteisiin.

Solmutunnisteiden päivittämisen kannalta solmujen poistaminen on paljon yksinkertaisempaa kuin uusien solmujen lisääminen. Tämä perustuu siihen, että XML-puun solmutunnisteisiin ei ole välttämätöntä tehdä muutoksia, vaikka puusta poistettaisiin solmuja. Tämä johtuu siitä, että jäljellä olevien solmujen väliset keskinäiset suhteet käyvät edelleen ilmi niiden solmutunnisteista.

4.2.3 Rakenteellisen liitosoperaation evaluointi

Edellä esitellyt tiedon fragmentointi ja solmutunnisteiden hyödyntäminen tarjoavat pohjan tehokkaalle kyselyjen evaluoinnille. Tarkemmista evaluointitekniikoista tässä tutkielmassa esitellään rakenteellisen liitosoperaation evaluointi. *Rakenteelliset liitosoperaatiot* voidaan ajatella natiivien XML-tietokantojen vastineeksi relaatiotietokantojen liitosoperaatioille: rakenteelliset liitosoperaatiot yhdistävät monikoita kahdesta syötteestä jonkin rakenteellisen ehdon perusteella.



Kuva 17. Esimerkki XML-dokumentista (Abiteboul & al., 2011, s. 102).

Kuvassa 17 on esimerkki XML-dokumentista, jonka solmutunnisteina on käytetty sijaintiin perustuvia solmutunnisteita. Olkoon tehtävänä liittää sellaiset solmujen `b` ja `g` parit, joissa solmu `b` on solmun `g` esi-isä. Olkoot nyt `X` ja `Y` solmutunnisteita sisältäviä listoja siten, että `X` sisältää `b`-solmujen solmutunnisteet ja `Y` vastaavasti `g`-solmujen solmutunnisteet. Tarkoituksena on nyt siis yhdistää sellaiset joukkojen `X` ja `Y` sisältämien solmutunnisteiden parit, jotka muodostavat esi-isä-jälkeläinen-suhteen.

Yksinkertaisin ja naiivein tapa toteuttaa liitos on käyttää sisäkkäisiä silmukoita: jokaisesta listan X solmutunnisteesta käydään läpi jokainen listan Y solmutunniste. Tämä johtaa aikavaativuuteen O(|X|*|Y|), koska jokaista listan X solmutunnistetta verrataan jokaiseen listan Y solmutunnisteeseen. Tämä on hyvin hidas ratkaisu eikä toimi käytännössä riittävän nopeasti. Sen sijaan eräs tehokas tapa toteuttaa rakenteellinen liitosoperaatio on pinoihin perustuva liitos (stack-based join). Seuraavaksi esitellään lyhyesti pinoihin perustuvan liitoksen yleinen toimintaperiaate.

Pinoihin perustuvassa liitoksessa solmutunnisteiden on pystyttävä vastaamaan seuraaviin kysymyksiin mahdollisimman tehokkaasti:

1. Onko solmu id_1 solmun id_2 vanhempi tai esi-isä?

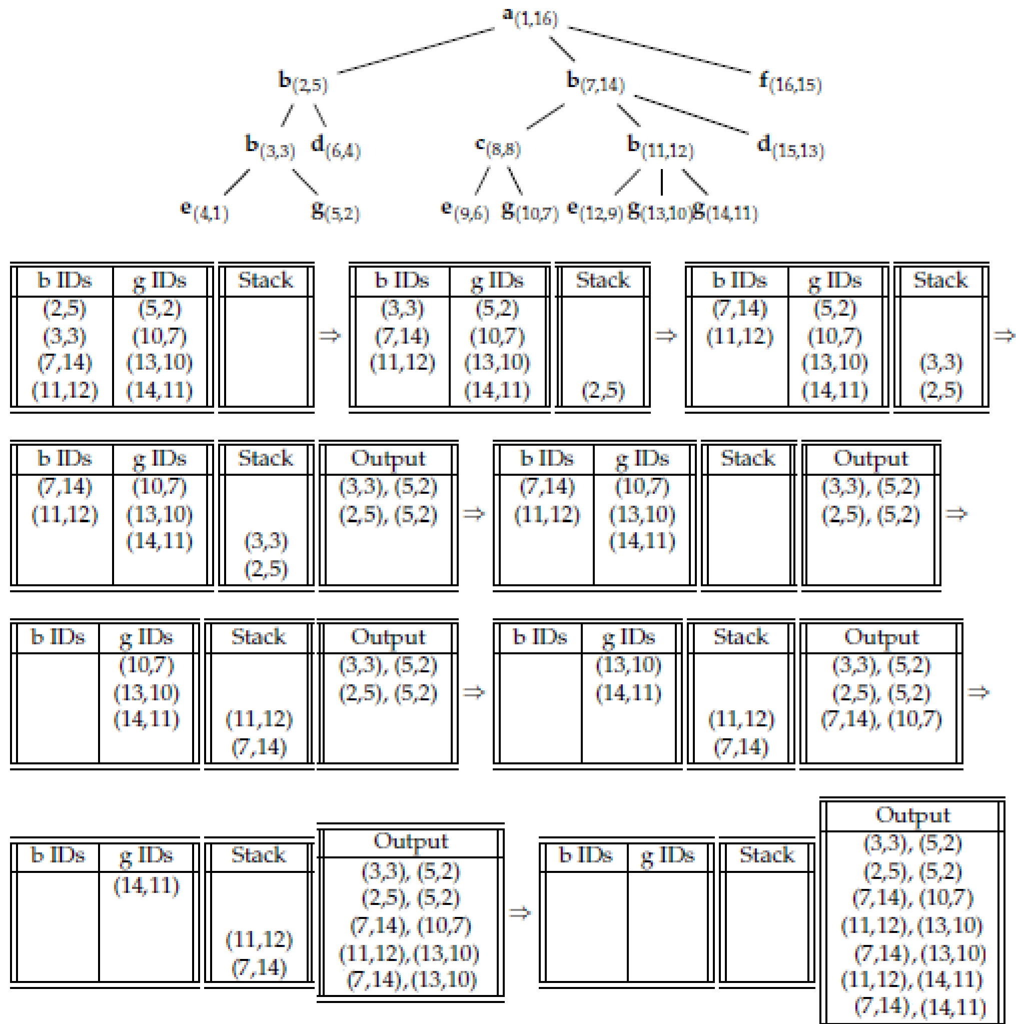
2. Alkaako solmu id_1 solmun id_2 jälkeen käytäessä puu läpi esijärjestyksessä (eli onko id_1:n aloitusmerkintä id_2:n aloitusmerkinnän jälkeen)

3. Päättyykö solmu id_1 solmun id_2 jälkeen? (eli onko id_1:n lopetusmerkintä id_2:n lopetusmerkinnän jälkeen)

Mikäli kuhunkin näistä kolmesta kysymyksestä voidaan vastata vakioajassa (kuten esimerkiksi luvussa 4.2.2 esiteltyjen sijaintiin perustuvien solmutunnisteiden avulla voidaan), niin pinoihin perustuvat liitosoperaatiot voidaan evaluoida tehokkaasti ajassa $O(|X|+d_x*|Y|)$, missä d_x on suurin määrä listassa X toisiinsa jälkeläissuhteessa olevia solmuja. Pinoihin perustuvat liitosoperaatiot käyttävät kahta listaa, jotka sisältävät solmutunnisteita. Kysymyksen 1 avulla voidaan päättää onko pari (i,j) ratkaisu, kun i on solmutunniste ensimmäisestä listasta ja j puolestaan toisesta. Kysymysten 2 ja 3 perusteella voidaan pinoja hyödyntämällä käydä solmutunnistelistat läpi siten, että ratkaisun selvittämiseksi ei tarvitse vertailla jokaista mahdollista (i,j)-paria.

Seuraavaksi esitellään välivaiheittain STD-algoritmin (StackTreeDescendant) toiminta. STD on eräs toteutustapa pinoihin perustuvalle rakenteelliselle liitosoperaatiolle. Seuraavaksi palataan aiempaan esimerkkiin ja käydään välivaiheittain läpi lii-

tosoperaatio kuvan 17 XML-dokumentin `b`- ja `g`-solmujen välillä, ehdolla solmu `b` on solmun `g` esi-isä. STD-algoritmilla toteutetun liitosoperaation alussa on liitosoperaation syötteiden (eli tässä tapauksessa solmujen `b` ja `g` tunnisteiden) oltava kasvavassa järjestyksessä (kuten kuvassa 18 ensimmäisissä `b Ids` - ja `g Ids` -sarakkeissa). Kuvassa 18 on havainnollistettu STD-algoritmin toiminta välivaiheineen.



Kuva 18. Esimerkki STD-algoritmin toiminnasta (Abiteboul & al., 2011, s. 102).

Ensimmäiseksi solmutunnistelistan `b` ensimmäinen tunniste, eli (2,5), asetetaan pinoon. Seuraavaksi käydään läpi lista lopuista solmun `b` tunnisteista: tarkasteltava `b`-solmutunniste lisätään nyt pinoon mikäli kyseinen solmutunniste on pinon päällimmäisen solmutunnisteen (eli solmutunnisteen (2,5)) jälkeläinen. Kuvan 18 toisessa

välivaiheessa siirretään b-solmun tunniste (3,3) pinoon, koska kyseessä on pinon päällimmäisen solmun jälkeläissolmu. Solmujen b solmutunnistelistan seuraava solmutunniste on (7,14), joka ei ole solmun (2,5) jälkeläissolmu. Tästä johtuen algoritmi lopettaa b-solmujen solmutunnistelistan läpikäymisen ja siirtyy g-solmujen solmutunnistelistaan. Tarkoituksena on nyt hakea mahdollisia esi-isä-jälkeläinen-suhteita pinon ja g-solmutunnistelistan välillä. Havaitaan, että ensimmäinen solmutunniste g (5,2) on jälkeläinen suhteessa molempiin pinojen solmutunnisteisiin. Tästä seuraa kaksi ensimmäistä löydettyä tulosta: solmuparit ((3,3),(5,2)) ja ((2,5),(5,2)). Solmutunnistelistan g solmutunniste (5,2) voidaan nyt poistaa, koska sitä on verrattu pinon kaikkiin solmutunnisteisiin. Toisin sanoen solmutunnisteelle (5,2) on haettu kaikki mahdolliset b-nimiset esi-isäsolmut, joten solmutunnistetta (5,2) ei tarvita enää.

Tämän jälkeen käsittelyyn otetaan g-solmutunnistelistan seuraava solmutunniste, eli (10,7). Nyt havaitaan että solmutunnisteen (10,7) ja pinossa olevien solmutunnisteiden välillä ei ole esi-isä-jälkeläinen –suhdetta. Tämä tarkoittaa sitä, että solmu (10,7) esiintyy alkuperäisessä XML-dokumentissa pinossa olevien solmujen jälkeen. Tästä voidaan puolestaan päätellä, että solmutunnistelistassa g ei voi enää olla solmuja, jotka olisivat pinon solmujen jälkeläisiä. Tämä perustuu siihen, että solmutunnistelistan g solmutunnisteet ovat dokumenttijärjestyksessä. Koska käsiteltävä solmutunnistelistan g solmutunniste ei ole pinon solmujen jälkeläinen, niin listan dokumenttijärjestykseen perustuen myöskään solmutunnistelistan g loput solmutunnisteet eivät voi olla pinon solmujen jälkeläisiä. Näin ollen pino voidaan tyhjentää.

Seuraavaksi pinoon siirretään jälleen solmutunnistelistan b ensimmäinen solmutunniste, eli (7,14). Tämän jälkeen pinoon siirretään myös solmutunniste (11,12), koska se on solmutunnisteen (7,14) jälkeläinen. Seuraavaksi verrataan jälleen solmutunnistelistan g solmutunnisteita pinossa oleviin solmutunnisteisiin. Havaitaan, että solmutunniste (10,7) on nyt jälkeläinen pinon solmulle (7,14). Tästä saadaan siis kolmanneksi tulokseksi solmupari ((7,14),(10,7)). Solmutunniste (10,7) on nyt käsitelty ja voidaan poistaa. Seuraavaksi havaitaan, että solmutunniste (13,10) on jälkeläinen pinon solmuille (11,12) ja (7,14). Joten lisätään tuloksiin solmuparit ((11,12),(13,10)) sekä ((7,14),(13,10)) ja poistetaan solmutunniste (13,10). Lopuksi huomataan, että

solmutunnistelistan g viimeinen solmutunniste (14,11) on jälkeläinen pinon solmuille (11,12) ja (7,14), joten lisätään tuloksiin solmuparit ((11,12),(14,11)) sekä ((7,14),(14,11)) ja poistetaan solmutunniste (14,11). Kaikki STD-algoritmin löytämät tulokset on havainnollistettu kuvan 18 viimeisessä output-sarakkeessa. Nimensä mukaisesti STD:n antamat tulokset ovat jälkeläissolmujen mukaisessa järjestyksessä.

STD toteuttaa siis rakenteellisen liitosoperaation hyvin tehokkaasti käyttämällä apuna järjestettyjä solmutunnistelistoja sekä pinoa. STD:n lisäksi toinen tunnettu pinoihin perustuvaan liitokseen pohjautuva algoritmi on STA (StackTreeAncestor). Algoritmin STA merkittävin ero algoritmiin STD nähden on tuloksen antaminen esi-isäsolmujen mukaisessa järjestyksessä. STD- ja STA-algoritmit antavat siis tulokset joko jälkeläissolmujen tai esi-isä-solmujen mukaisessa järjestyksessä. Järjestetty tulos on hyvin kätevä jatkoa ajatellen, koska mahdollisissa jatkoliitoksissa tuloksen on oltava sopivassa järjestyksessä. STD- ja STA-algoritmeista voidaan valita kumpaa sovelletaan, jotta jatkoliitokset onnistuvat tarvitsematta lajitella välituloksia.

Yleisesti XML-kyselyjen evaluointi on hyvin laajalti tutkittu aihe. Tässä luvussa esitelty pinoihin perustuva rakenteellisen liitosoperaation toteuttava STD-algoritmi on hyvin tehokas tapa toteuttaa rakenteellinen liitosoperaatio. On kuitenkin huomionarvoista, että rakenteellisten liitosoperaatioiden toteuttamiseksi on kehitetty myös muunlaisia kuin pinoihin perustuvia algoritmeja (Al-Khalifa & al., 2002).

5 Transaktioidenhallinta natiiveissa XML-tietokannoissa

Tietokannat toimivat monesti usean samanaikaisen käyttäjän tiedonhallintatyökaluina. Tällöin on mahdollista, että käyttäjät päivittävät toisistaan tietämättä samoja tietokannan tietueita samanaikaisesti. Tämä johtaa siihen, että tietokannanhallintajärjestelmän pitää pystyä hallitsemaan samanaikaisia operaatioita.

Samanaikaisuudenhallinnan tärkeimpänä tehtävänä on pitää tietokannan sisältämä tieto eheänä useiden samanaikaisten käyttäjien suorittamien operaatioiden ristitulessa. Tietokanta pystyy vastaamaan samanaikaisuudenhallinnan vaatimukseen transaktioiden avulla. Transaktiot ovat luku- ja kirjoitusoperaatioiden sarjoja, joiden tulee toteuttaa ACID-ominaisuudet. Nämä transaktioiden ACID-ominaisuudet esitellään luvussa 5.1. Natiivien XML-tietokantojen erilaisia samanaikaisuudenhallintamenetelmiä käydään läpi puolestaan luvussa 5.2. Lopuksi luvussa 5.3 kerrotaan pääpiirteittäin kuinka natiivit XML-tietokannat voivat toipua häiriöistä. Myös toipumismenetelmät perustuvat transaktioidenhallintaan.

5.1 Transaktioiden ACID-ominaisuudet

Tietokantojen toiminta pohjautuu käyttäjän pyytämiin kyselyihin, joiden toteuttaminen perustuu tietokannan sisäisiin luku- ja kirjoitusoperaatioihin. Transaktioilla tarkoitetaan tällaisten operaatioiden sarjoja, jotka toteuttavat *ACID-ominaisuudet* (Sciore, 2008, luku 14). Lyhenne ACID tulee sanoista

- Atomicity (atomisuus)
- Correctness (oikeellisuus)
- Isolation (eristyvyys)
- Durability (pysyvyys).

Atomisuuden mukaan transaktiosta suoritetaan joko jokainen operaatio tai vaihtoehtoisesti ei ainuttakaan operaatiota. Näin jokainen transaktio päättyy siis joko vahvistukseen (commit) tai peruutukseen (abort). Vahvistus tarkoittaa sitä, että kaikki transaktion operaatiot on suoritettu onnistuneesti. Peruutuksessa puolestaan kumotaan kaikki ne operaatiot, jotka transaktio on jo ehtinyt suorittamaan.

Oikeellisuuden mukaan transaktion suorittaminen siirtää tietokannan sallitusta tilasta sallittuun tilaan. Tämä tarkoittaa sitä, että transaktion on noudatettava tietokannan oikeellisuuden määrityksiä.

Eristyvyydellä tarkoitetaan sitä, että transaktiot on eristetty toisistaan siten, että yksittäinen transaktio ei näe eikä vaikuta muihin transaktioihin. Oikeellisuus- ja eristyvyys-ominaisuudet vastaavat samanaikaisuudenhallinnasta.

Pysyvyys tarkoittaa puolestaan sitä, että transaktion tietokannan datasisältöön tekemät muutokset tallennetaan pysyvästi tietokantaan siten, että koneiden tilapäiset häiriöt eivät voi hävittää tietoja. Ominaisuudet pysyvyys ja atomisuus tarjoavat ratkaisun häiriöistä toipumisiin.

Tietokantojen transaktiot voivat perustua ACID-ominaisuuksien sijasta BASE-ominaisuuksiin (Basically Available, Soft state, Eventually consistent) (Pritchett, 2008). Transaktioiden BASE-ominaisuuksien tarkoituksena on tarjota tietokannoille parempaa skaalautuvuutta ACID-transaktioita käyttäviin tietokantoihin nähden. BASE-transaktioita ei kuitenkaan esitellä tässä tutkielmassa tarkemmin, koska selkeästi suurin osa nykyisistä tietokannanhallintajärjestelmistä käyttää perinteisiä ACID-transaktioita.

5.2 Samanaikaisuudenhallinta

Shan & al. (2012) jakaa XML-tiedon samanaikaisuudenhallintamenetelmät kahteen luokkaan: lukituksia käyttäviin menetelmiin ja muihin kuin lukituksia käyttäviin menetelmiin. Luvussa 5.2.1 esitellään lukituspohjaisia samanaikaisuudenhallintaratkaisuja natiiveissa XML-tietokannoissa. Ei-lukituspohjaisista menetelmistä yleisin pe-

rustuu aikaleimojen käyttöön. Aikaleimapohjaisia samanaikaisuudenhallintaratkaisuja esitellään luvussa 5.2.2.

5.2.1 Lukitusmenetelmät

Yleisin käytäntö varautua samanaikaisuudenhallintaan tietokantaympäristöissä on hyödyntää lukitusmenetelmiä. Useimmat lukitusmenetelmät perustuvat Eswaran & al. (1976) esittelemään kaksivaiheiseen lukitusprotokollaan (Two-Phase Locking Protocol). Kaksivaiheinen lukitusprotokolla käyttää kahdenlaisia lukkoja: luku- ja kirjoituslukkoja. Lukulukko on jaettu lukko. Tämä tarkoittaa sitä, että saman tietueen voi lukita samanaikaisesti useita eri lukulukkoja. Kirjoituslukot ovat puolestaan eksklusiivisia lukkoja: ainoastaan yksi transaktio kerrallaan voi lukita tietyn tietueen eksklusiivisella lukolla. Lukitusprotokollan kaksivaiheisuudella tarkoitetaan sitä, että transaktio ei saa lukita tietoa enää sen jälkeen kun kyseinen transaktio on vapauttanut jonkin lukon. Tämä kaksivaiheisuus mahdollistaa transaktioiden *sarjallistuvuuden* (*serializability*). Sarjallistuvuuden avulla transaktioita voidaan suorittaa sarjallisesti. Sarjallinen transaktioiden suorittaminen tarkoittaa sitä, että eri transaktiot suoritetaan kokonaisuudessaan alusta loppuun yksi toisensa jälkeen. Sarjallistuvuus takaa sen, että tietokannan sisältö pysyy virheettömänä eivätkä samanaikaiset transaktiot aiheuta ongelmia tiedon eheydelle.

Lukituspohjaisissa samanaikaisuudenhallintamenetelmissä transaktiot voivat lukkiutua umpikujaan (deadlock). Esimerkiksi transaktiot T_1 ja T_2 ajautuvat umpikujaan, jos T_1:n tarvitsee lukita kirjoituslukolla sellainen tietue, jonka T_2 on jo lukinnut. Samanaikaisesti T_2 haluaa lukita kirjoituslukolla tietueen, jonka T_1 on jo lukinnut. Nyt molemmat transaktiot ovat juuttuneet jonottamaan toistensa lukitsemia tietueita. Tällöin tietokannanhallintajärjestelmän pitää osata peruuttaa toisen transaktion suorittaminen. Suoraviivainen ja naiivi ratkaisu umpikujien purkamiseksi on tutkia transaktioiden suorittamiseen kulunutta aikaa: jos jonkin transaktion suorittaminen on kestänyt liian kauan, voidaan sen todeta olevan umpikujassa. Tällöin kyseisen transaktion suorittaminen perutaan. Kehittyneempiäkin ratkaisuja umpikujien purkamiseen on olemassa.

Kaksivaiheinen lukitusprotokolla ei itsessään riitä toimivaksi lukitusmenetelmäksi natiivissa XML-tietokannassa. Tästä johtuen natiivit XML-tietokannat joutuvat soveltamaan kaksivaiheisesta lukitusprotokollasta XML-muotoiselle tiedolle sopivan lukitusmenetelmän. XML-tiedon lukitsemiseen on kehitetty useita erilaisia lukitusmenetelmiä.

Yksinkertaisin tapa lukita XML-muotoista tietoa on käyttää dokumenttitason lukitusta, jonka ideana on lukita transaktion käsittelemä kokonainen XML-dokumentti. Tämä on kuitenkin hyvin alkeellinen samanaikaisuudenhallintatapa, koska yhtä XML-dokumenttia saa käsitellä vain yksi transaktio kerrallaan (Helmer & al., 2003). Lukituspohjaiset samanaikaisuudenhallintamenetelmät ovat yleisesti sitä parempia mitä pienempiä tietoyksiköitä transaktiot lukitsevat. Tästä johtuen transaktioiden pitää välttää lukitsemasta sellaista tietoa, jota ne eivät tarvitse.

Dokumenttitason lukitusta kehittyneempi tapa on käyttää solmulukitusta. Solmulukituksella lukitaan yksittäinen käsiteltävä XML-solmu sekä tilanteen mukaan myös käsiteltävän solmun esi-isä- tai jälkeläissolmuja. Haustein & Härderin (2004) esittelemässä solmulukitusmenetelmässä on seitsemän erilaista lukitustapaa:

- NR-lukko (node read), jota käytetään kontekstisolmun lukemisen yhteydessä. NR-lukko asetetaan kontekstisolmun lisäksi myös kontekstisolmun esi-isäsolmuihin.

- IX-lukko (intention exclusive), jonka avulla ilmaistaan tarve kirjoitusoperaatioille lukittavan solmun alipuussa. Tämä kirjoitustarve ei kuitenkaan koske lukittavan solmun lapsisolmuja (jota varten on CX-lukko).

- LR-lukko (level read), jolla lukitaan käsiteltävä solmu lapsisolmuineen. Esimerkiksi DOM-metodi getChildNodes() vaatisi ainoastaan LR-lukon käsiteltävään solmuun, eikä yksittäisiä NR-lukkoja jokaiseen käsiteltävän solmun lapsisolmuun.

- SR-lukko (subtree read), jolla lukitaan käsiteltävä solmu silloin kun tarkoituksena on suorittaa lukuoperaatioita kaikkiin käsiteltävän solmun alipuun solmuihin.

- CX-lukko (child exclusive), jonka avulla käsiteltävä solmu lukitsee lapsisolmun, johon on tarkoitus suorittaa kirjoitusoperaatioita.

- U-lukko (update option), joka tarjoaa mahdollisuuden sekä luku- että kirjoitusoperaatioiden suorittamiselle: U-lukko tukee lukuoperaatioita, mutta tarpeen vaatiessa U-lukko voidaan muuntaa X-lukoksi kirjoitusoperaatioita varten.

- X-lukko (exclusive), jota tarvitaan käsiteltävän solmun lukitsemiseen, jos käsiteltävään solmuun halutaan tehdä muutoksia (eli kirjoitusoperaatioita). X-lukko asettaa CX-lukon käsiteltävän solmun vanhempi-solmulle ja IX-lukot sen kaikille muille esi-isä-solmuille.

Kuvassa 19 on esitetty lukkojen yhteensopivuusmatriisi.

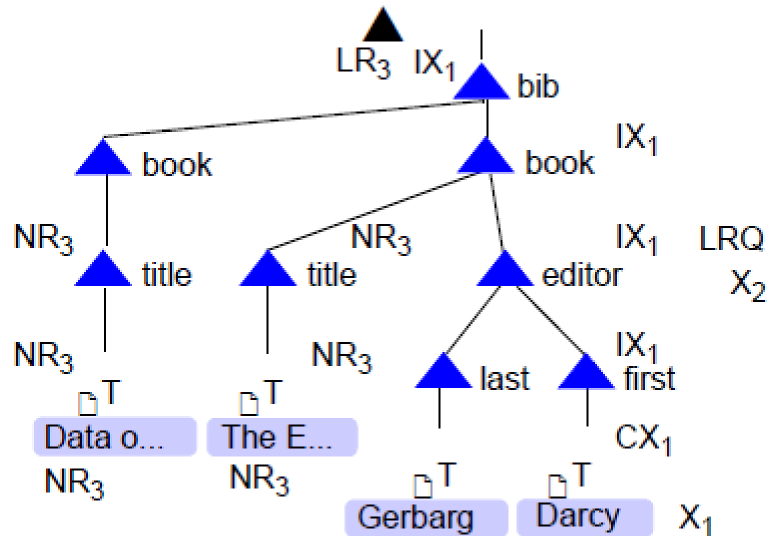
	-	NR	IX	LR	SR	CX	U	X
NR	+	+	+	+	+	+	-	-
IX	+	+	+	+	-	+	-	-
LR	+	+	+	+	+	-	-	-
SR	+	+	-	+	+	-	-	-
CX	+	+	+	-	-	+	-	-
U	+	+	+	+	+	+	-	-
X	+	-	-	-	-	-	-	-

Kuva 19. Solmulukkojen yhteensopivuusmatriisi (Haustein & Härder, 2004).

Kuvasta nähdään milloin mikäkin lukko voidaan asettaa. Esimerkiksi lukottomaan solmuun voidaan asettaa mikä tahansa lukko (ensimmäinen sarake). Vastaavasti sel-

laiseen solmuun, joka on lukittu X-lukolla, ei voida asettaa minkäänlaista lukkoa (viimeinen sarake).

Kuvassa 20 on havainnollistettu esimerkkitilanne solmulukkojen toiminnasta.



Kuva 20. Esimerkki XML-datan solmulukituksesta (Haustein & Härder, 2004).

Kuvassa 20 transaktion T_1 tarkoituksena on muuttaa ”`Darcy`”-tekstisolmun sisältöä, joten T_1 lukitsee kyseisen tekstisolmun X-lukolla (X_1). Seuraavaksi lukitaan (X-lukon sääntöjen mukaisesti) ”`Darcy`”-tekstisolmun vanhempisolmu CX-lukolla (CX_1) sekä ”`Darcy`”-tekstisolmun muut esi-isäsolmut IX-lukoilla. Transaktion T_1 suorittamisen aikana transaktio T_2 haluaa poistaa `editor`-solmun (sekä samalla myös tämän alipuun). T_2 tarvitsee nyt X-lukon editor-solmuun. Tätä lukitusta ei kuitenkaan voida vielä suorittaa, koska editor-solmu on jo lukittu IX-lukolla T_1:n toimesta. Transaktion T_2 täytyy nyt odottaa kyseisen IX_1-lukon vapautusta, joten T_2:n lukko X_2 asetetaan jonoon (lock request queue, kuvassa LRQ X_2). Samanaikaisesti transaktio T_3 hakee listausta kaikista `title`-elementeistä. T_3 pyytää LR-lukon `bib`-solmuun. Seuraavaksi T_3 lukitsee NR-lukoilla solmut `book`, `title` ja `title`-elementin tekstisolmun. Kun T_2 saa lopulta lukittua `editor`-solmun X-lukolla, niin T_2 lukitsee CX-lukolla `book`-solmun sekä IX-lukoilla kyseisen `book`-solmun esi-isäsolmut.

Haustein & Härder (2004) esittävät edellä esiteltyjen seitsemän lukon lisäksi vielä kolme muuta lukkoa, joiden avulla transaktioidenhallintaa voidaan tehostaa. Näiden kolmen täydentävän lukon avulla lukittavien tietoyksiköiden koko pienenee. Näitä lukkoja ei kuitenkaan esitellä tässä tutkielmassa, koska alkuperäiset seitsemän lukkoa riittävät toimivan transaktionhallinnan havainnollistamiseksi.

5.2.2 Aikaleimapohjaiset menetelmät

Aikaleimapohjaisissa samanaikaisuudenhallintamenetelmissä tietuetta ei päivitetä kirjoittamalla vanhan tietueen päälle, vaan tietokantaan luodaan tietueesta uusi päivitetty versio. Tällöin tietokanta sisältää samasta tietueesta useita eri versioita, jotka eroavat toisistaan aikaleiman perusteella. Aikaleimapohjaisissa menetelmissä käyttäjälle tarjotaan tietokannan tila jonain tiettynä ajan hetkenä.

Useita tietueiden eri versioita hyödyntävän samanaikaisuudenhallintamenetelmän ajatus on peräisin 1970-luvulta (Reed, 1978). Tämän ajatuksen pohjalta kehitettiin MTOP-protokolla (Multiversion Timestamps Ordering Protocol). MTOP-protokollassa kukin transaktio T saa aloituksensa yhteydessä yksikäsitteisen aikaleiman TS(T). Lisäksi kustakin tietokannan tietueesta ylläpidetään useita eri versioita: tietokannan tietueesta X luodaan uusi versio aina kun tietueen X tietoa päivitetään. Kullakin tietueen X versiolla V on luku-aikaleima RTS(V), jonka arvo on suurin aikaleima niiden transaktioiden aikaleimoista, jotka ovat lukeneet kyseisen version. Vastaavasti kullakin tietueen X versiolla V on kirjoitusaikaleima WTS(V), jonka arvo on suurin aikaleima niiden transaktioiden aikaleimoista, jotka ovat kirjoittaneet kyseiseen versioon.

Transaktio T voi nyt lukea tietueen X, mikäli seuraavat ehdot ovat voimassa:

- on olemassa versio V´, jolle on voimassa WTS(V´) < TS(T).
- ei ole olemassa toista versiota V´´, jolle olisi voimassa WTS(V´) < WTS(V´´) < TS(T).

Transaktio T voi puolestaan kirjoittaa tietueeseen X, mikäli seuraavat ehdot ovat voimassa:

- WTS(V) < TS(T), missä V on X:n uusin versio.
- RTS(V) ≤ TS(T), missä V on X:n uusin versio.

Jos ehdot eivät täyty, niin transaktio T peruutetaan ja käynnistetään uudelleen uudella aikaleimalla.

Seuraavaksi esitellään MTOP-protokollaan perustuva XStamps-protokolla (Win & al., 2003). XStamps on aikaleimapohjainen samanaikaisuudenhallintamenetelmä XML-muotoisen tiedon hallintaan. XStamps luokittelee transaktiot kahteen eri luokkaan: lukutransaktiohin ja kirjoitustransaktioihin. Transaktio on kirjoitustransaktio, jos se sisältää vähintään yhden seuraavista operaatioista:

- XML-solmun lisääminen tietokantaan.
- tietokannan XML-solmun muokkaaminen.
- XML-solmun poistaminen tietokannasta.

Muussa tapauksessa transaktio luokitellaan lukutransaktioksi. XStamps käyttää parametreja SCO (safety coefficient) ja STH (safety threshold). Jokaisella tietokannan XML-solmun versiolla on ominaisuutenaan SCO. SCO-ominaisuuden avulla ilmaistaan todennäköisyysarvio sille kuinka todennäköisesti viimeisin kirjoitustransaktio päättyy vahvistukseen (peruutuksen sijaan). SCO:n arvon laskemisessa käytetään apuna mm. transaktion kirjoitusoperaatioiden lukumäärää, tietokannan transaktoiden kokonaislukumäärää sekä tietokantaan säilötyn tiedon kokoa.

Jokaisella transaktiolla on puolestaan ominaisuutena STH, joka ilmaisee transaktion täsmällisyyttä. STH:n arvo on sitä suurempi mitä uudempaa ja virheettömämpää tietoa transaktio vaatii. STH:n avulla samanaikaisuudenhallinnasta saadaan joustavampaa, koska pienen STH-arvon omaavia transaktioita voidaan suorittaa nyt useammissa tilanteissa vahvistukseen asti.

Seuraaavaksi esitellään XStamps-protokollan mukaiset säännöt transaktioiden suorittamiselle. Nämä säännöt takaavat transaktioiden sarjallistuvuuden. Kuvassa 21 on kerrottu säännöt transaktiolle T, joka yrittää suorittaa lukuoperaation R(N) tietokannan solmuun N.

a) Kun $TS(T) \geq WTS(N)$

i) Jos $SCO(N) \geq STH(T)$ ja solmua N ei ole merkitty poistetuksi, niin transaktio T saa suorittaa operaation R(N) ja RTS(N) saa arvon TS(T), jos $TS(T) > RTS(T)$.

Jos $SCO(N) \geq STH(T)$ ja solmu N on merkitty poistetuksi, niin transaktion T lukuoperaatio perutaan, minkä jälkeen T jatkaa jäljellä olevien operaatioidensa suorittamista.

ii) Jos $SCO(N) < STH(T)$, niin transaktion T täytyy odottaa siihen asti kunnes $SCO(N) \geq STH(T)$.

b) Kun $TS(T) < WTS(N)$

i) Jos $SCO(N) \geq STH(T)$ ja N on merkitty poistetuksi, niin tarkistetaan onko solmulle N olemassa vanhempaa aikaleimaa WTS´(N), jolle pätee $WTS´(N) \leq TS(T)$. Mikäli tällainen aikaleima löytyy, niin transaktio T saa suorittaa operaation R(N).

Jos $SCO(N) \geq STH(T)$ ja N on merkitty lisätyksi, niin transaktion T lukuoperaatio perutaan, minkä jälkeen T jatkaa jäljellä olevien operaatioidensa suorittamista. Tämä lukuoperaation peruminen perustuu siihen, että solmu N on lisätty tietokantaan transaktion T käynnistymisen jälkeen, joten T ei voi lukea solmua N.

ii) Muussa tapauksessa etsi solmun N uusin sellainen versio N´, joka täyttää seuraavat ehdot:

- ei ole olemassa sellaista versiota N``, joka olisi versioiden N ja N´ välissä
- versiolle N` pätee $TS(T) \geq WTS(N`)$

Mikäli tällaista versiota ei löydy, niin transaktio T perutaan. Muussa tapauksessa toistetaan operaatio R(N) löydettyyn versioon N`.

Kuva 21. XStamps-protokollan säännöt transaktion lukuoperaatiolle (Win & al., 2003).

Kuvassa 22 on kerrottu säännöt transaktiolle, joka yrittää suorittaa kirjoitusoperaation solmuun N. Mahdolliset kirjoitusoperaatiot ovat I(N) (solmun lisääminen tietokantaan), D(N) (solmun poistaminen tietokannasta) ja U(N) (solmun päivittäminen).

a) Kun $TS(T) \geq WTS(N)$ ja $TS(T) \geq RTS(N)$

i) Jos kyseessä on U(N)-operaatio, niin luodaan solmuun N perustuen uusi solmu N′, joka saa arvonsa U(N)-operaation perusteella. Asetetaan nyt uudelle solmulle N\` seuraavat arvot: $RTS(N`) = 0$ ja $WTS(N`) = TS(T)$. Lopuksi lasketaan SCO(N\`).

ii) Jos kyseessä on I(N)-operaatio, niin luodaan uusi solmu N\` ja asetetaan tälle aikaleimat $RTS(N`) = 0$ ja $WTS(N`) = TS(T)$. Lasketaan solmulle N\` arvo SCO(N\`). Lisätään solmu N\` solmun N lapsisolmuksi.

iii) Jos kyseessä on D(N)-operaatio, niin merkataan solmu N poistetuksi (mikäli solmua N ei ole aiemmin merkattu poistetuksi). Asetetaan solmun N kirjoitusaikaleima: $WTS(N) = TS(T)$ ja päivitetään SCO(N).

b) Kun $TS(T) \geq RTS(N)$ ja $TS(T) < WTS(N)$

i) Jos $SCO(N) \geq STH(T)$, niin perutaan transaktion T käsittelyssä oleva kirjoitusoperaatio, minkä jälkeen T jatkaa jäljellä olevien operaatioidensa suorittamista.

ii) Jos $SCO(N) < STH(T)$, niin odotetaan kunnes $SCO(N) \geq STH(T)$.

c) Kun $TS(T) < RTS(N)$, niin transaktio T perutaan.

Kuva 22. XStamps-protokollan säännöt transaktion kirjoitusoperaatiolle (Win & al., 2003).

Aikaleimapohjaiset samanaikaisuudenhallintamenetelmät mahdollistavat lukitusmenetelmiä nopeampaa samanaikaisuudenhallintaa tietokannan tietuieden useiden eri versioiden ansiosta. Tämän lisäksi aikaleimapohjaisissa samanaikaisuudenhallintamenetelmissä voidaan välttää kokonaan umpikujat transaktioidenhallinnassa. Aikaleimapohjaisten samanaikaisuudenhallintamenetelmien merkittävimpänä heikkoutena on kuitenkin suuri tilatarve useiden eri tietueiden versioiden ylläpitämisen johdosta.

5.3 Häiriöistä toipuminen

Tietokannan pitää pystyä toipumaan tilapäisistä laitteistohäiriöistä. Häiriöistä toipumisessa tietokanta palautuu johonkin sallittuun tilaan, jossa osa transaktioista on suoritettu loppuun asti ja toisten transaktioiden prosessointia ei ole vielä aloitettu. Natiivien XML-tietokantojen (kuten myös relaatiotietokantojen) toipuminen perustuu tietokannan lokitiedoston hyödyntämiseen (Fiebig & al., 2002). Tietokannanhallintajärjestelmä merkitsee lokitiedostoon lokimerkintöinä kaikki transaktioiden suorittamat operaatiot. Toipumisessa tarvittavan lokitiedoston tiedon määrää voidaan pie-

nentää tarkistuspisteiden (checkpoints) avulla. Tarkistuspisteessä puskureista siirretään tietokannan muistiin tietokantatietuiden päivitykset. Toipumisessa tietokannan tarvitsee lukea lokitiedostosta yleensä ainoastaan viimeisimmän tarkistuspisteen jälkeen tulevat lokimerkinnät.

Lokitiedoston informaation perusteella tietokanta pystyy eheyttämään sisältämänsä tiedot takaisin johonkin sallittuun tilaan. Jos lokitiedostosta löytyy transaktiolle T esimerkiksi aloituskirjaus, mutta ei lainkaan vahvistus- tai peruutuskirjausta, voidaan tästä päätellä, että transaktio T on ollut käynnissä järjestelmän sammumisen aikana. Transaktion T tekemät muutokset joudutaan tällöin jälkikäteen kumoamaan. Lopuksi tietokannanhallintajärjestelmä voi halutessaan suorittaa uudelleen transaktion T.

Yleisesti kumoamisessa tietokannanhallintajärjestelmä kumoaa esimerkiksi sellaisten transaktioiden tekemät muutokset, jotka eivät ehtineet suorittaa kaikkia operaatioitaan ennen tietokantajärjestelmän sammumista. Vastaavasti uudelleensuorittamisessa tietokannanhallintajärjestelmä tekee uudelleen esimerkiksi sellaisten transaktioiden tekemät muutokset, jotka vahvistettiin suoritetuiksi ennen järjestelmän sammumista, mutta joiden puskureiden sisältöä ei oltu välttämättä vielä ehditty kirjoittamaan tietokannan muistiin (Sciore, 2008, luku 14).

6 Kokeellinen osio

Tässä luvussa esitellään tutkielman kokeellinen osio, jonka tarkoituksena oli tutustua eXist-tietokannan toimintaan. eXist on natiivi XML-tietokanta, jonka tarjoamat toiminnallisuudet esitellään lyhyesti luvussa 6.1. Kokeellisessa osiossa toteututun projektin vaatimukset esitellään luvussa 6.2. Käytetyt toteutustekniikat sekä kokeellisen osion tulokset esitellään puolestaan luvussa 6.3.

6.1 Natiivi XML-tietokanta eXist

eXist (Meier, 2013) on avoimeen lähdekoodiin perustuva natiivi XML-tietokanta, jonka ensimmäinen versio julkaistiin vuonna 2000. Tietokantapalveluiden lisäksi eXist tarjoaa selainpohjaisen integroidun ohjelmointiympäristön. eXist tukee mm. seuraavia XML-tekniikoita:

- XPath (kts. luku 2.4.1).
- XQuery (kts. luku 2.4.2) sekä XQuery Update, jonka avulla tietokannan tietueisiin voidaan tehdä päivityksiä (Robie & al., 2011).
- XSLT (Extensible Stylesheet Language Transformations), jonka avulla XML-dataa voidaan muuntaa (Clark, 1999).
- XForms, joka on vuorovaikutteisten lomakkeiden kuvailukieli (Boyer, 2009).
- XInclude, joka tarjoaa geneerisen ratkaisun XML-dokumenttien kokoamiseen useista osista (Marsh & al., 2006).
- XUpdate, joka on kevyt XML-kyselykieli XML-tiedon päivittämiseen (Laux & Martin, 2000).
- XML-kuvauskielet DTD ja XML Schema (kts. luku 2.3).

eXist tarjoaa lisäksi useita sovellusrajapintoja, kuten esimerkiksi

- hajautettujen järjestelmien arkkitehtuurimalli REST (Representational State Transfer) (Fielding, 2000)

- XML-RPC (XML Remote Procedure Call), jonka avulla tietokantaan voidaan lähettää etäkutsuja useilla eri ohjelmointikielillä (Winer, 1999)

- SOAP (Simple Object Access Protocol), joka on protokolla tiedon vaihtoon verkon välityksellä (Gudgin & al., 2007).

6.2 Vaatimukset

Tutkielman kokeellinen osio perustui eXist-tietokannan mukana tulevaan Shakespeare-kokoelmaan, joka sisältää XML-muodossa olevia Shakespearen näytelmiä. Tarkoituksena oli tehdä web-palvelu, jonka avulla käyttäjä voi navigoida helposti näissä näytelmissä ja hakea näytelmistä haluamiaan tietoja. Web-palvelun tuli tarjota käyttäjälle vähintään seuraavat ominaisuudet:

- Halutun henkilöhahmon vuorosanojen näyttäminen halutusta näytöksestä tai kohtauksesta.

- Näytelmän sisällysluettelon näyttäminen siten, että siitä näkyy järkevästi näytelmän näytökset kohtauksineen sekä kunkin kohtauksen henkilöhahmot.

- Näytelmän eri tietojen koostaminen yhteenvedoksi ja yhteenvedon näyttäminen.

- Palvelun on oltava helposti selattavissa: sisällysluettelosta on päästävä haluttuun kohtaukseen, kohtauksesta halutun henkilöhahmon tietoihin, mistä tahansa näytelmän yhteenvetoon ja yhteenvedosta suoraan toisen näytelmän yhteenvetoon.

- Web-palvelun on oltava käytettävissä perinteisessä internet-selaimessa sekä myös älypuhelimen selaimessa.

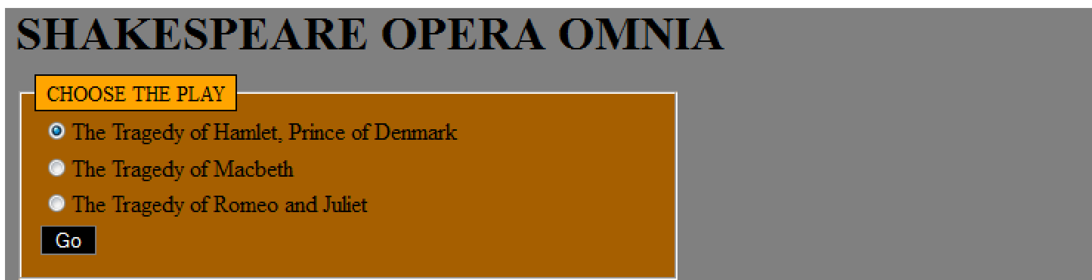
6.3 Toteutustekniikat

Web-palvelu toteutettiin eXist-tietokannan avulla. Suurin osa web-palvelun toiminnallisuuksista ohjelmoitiin XQuery-kielellä. Tarkoituksena oli tuottaa XQueryn avulla XHTML-sivuja, joiden ulkoasu on määritelty erillisessä CSS-tiedostossa (Cascading Style Sheets) (Bos & al., 2011). Apuna käytettiin lisäksi sekä Javascript-kieltä että Ajax-tekniikkaa käytettävyyden parantamiseksi. Web-palvelu jaettiin seitsemään eri XQuery-tiedostoon, joista jokainen toteutti jonkin toiminnallisuuden. Näiden tiedostojen välisessä kommunikoinnissa käytettiin REST-palvelun get-pyyntöjä. Web-palvelun toiminnallisuudet koostuivat seuraavista XQuery-tiedostoista:

- `index.xql` toteutti palvelun etusivun.
- `intro.xql` toteutti valitun näytelmän päävalikkosivun.
- `character.xql` toteutti valitun henkilöhahmon infosivun.
- `lines.xql` toteutti valitun henkilöhahmon vuorosanojen näyttämisen halutussa näytöksessä tai kohtauksessa.
- `toc.xql` toteutti valitun näytelmän sisällysluettelosivun.
- `contents.xql` toteutti valitun näytelmän osion esittämisen.
- `summary.xql` toteutti valitun näytelmän yhteenvetosivun.

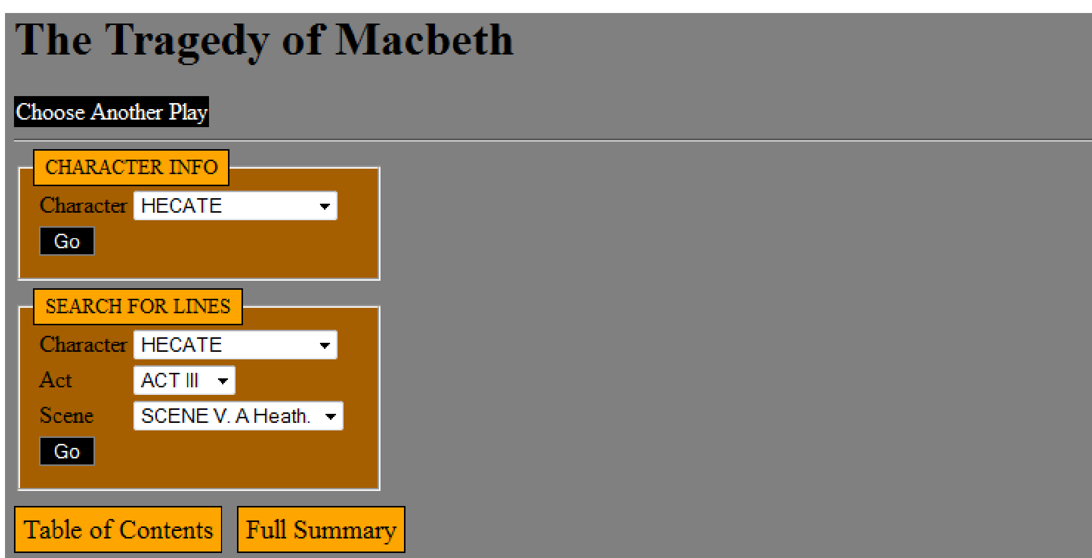
6.4 Tulokset

Seuraavaksi esitellään tutkielman kokeellisessa osiossa toteutettu web-palvelu. Palvelun aloitussivulla listataan kaikki tietokannasta löytyvät näytelmät. Kuvassa 23 on esimerkki aloitussivun näytelmälistauksesta.



Kuva 23. Web-palvelun etusivun näytelmälistaus kun tietokanta sisältää näytelmät Hamlet, Macbeth sekä Romeo ja Julia.

Käyttäjä valitsee näistä näytelmistä yhden ja siirtyy eteenpäin kyseisen näytelmän päävalikkosivulle. Tältä päävalikkosivulta löytyy neljä toimintoa: henkilöhahmon tietojen hakeminen, henkilöhahmon vuorosanojen hakeminen, näytelmän sisällysluettelon näyttäminen sekä näytelmän yhteenvedon näyttäminen. Kuvassa 24 on esimerkki näytelmän ”The Tragedy of Macbeth” päävalikkosivusta.



Kuva 24. Näytelmän Macbeth päävalikkosivu.

Sovellus listaa näytelmän päävalikkosivulle alasvetovalikkoon kaikki näytelmässä esiintyvät henkilöhahmot (Character Info –lomake kuvassa 24), joista käyttäjä valitsee yhden ja siirtyy eteenpäin henkilöhahmon infosivulle, joka sisältää kolme seuraavaa kohtaa:

- Niiden näytösten, kohtausten, epilogien ja prologien lukumäärä, joissa valittu henkilöhahmo esiintyy.

- Valitun henkilöhahmon puheenvuorojen lukumäärä.
- Lista niistä kohtauksista, epilogeista ja prologeista, joissa valittu henkilöhahmo esiintyy.

Edellä mainittu lista sisältää kunkin kohtauksen, epilogin tai prologin yhteydessä lukumäärän niistä valitun henkilöhahmon puheenvuoroista, jotka kuuluvat kyseiseen kohtaukseen, epilogiin tai prologiin. Lisäksi kukin kohtaus, epilogi ja prologi toimii linkkinä kyseiseen näytelmän kohtaan. Puheenvuorojen lukumäärä toimii puolestaan linkkinä valitun henkilöhahmon vuorosanoihin kyseisessä kohdassa näytelmää. Kuvassa 25 on esimerkki henkilöhahmon infosivusta.

The Tragedy of Hamlet, Prince of Denmark

Choose Another Play   Start Page of HAMLET

PRINCE FORTINBRAS

- Appears in 2 acts and in 2 scenes
- Has 6 speeches with 27 lines in total

Present in:

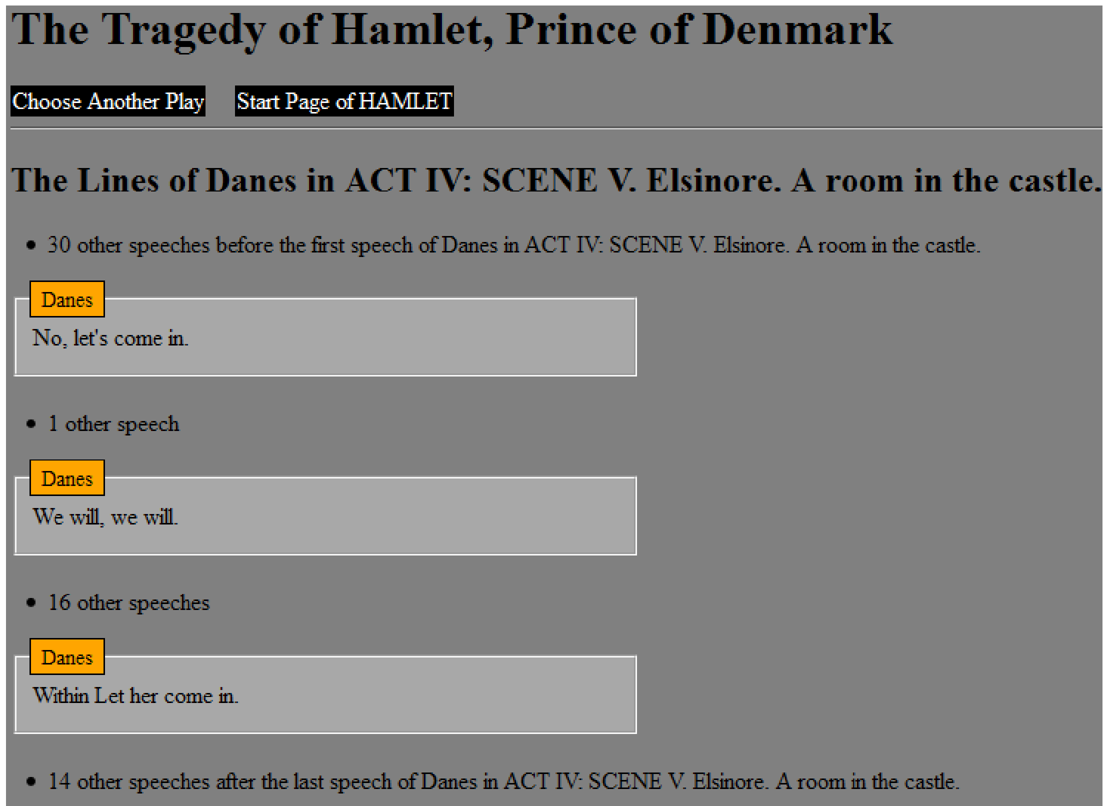
ACT IV: SCENE IV. A plain in Denmark.
Speeches: 2
Lines: 8

ACT V: SCENE II. A hall in the castle.
Speeches: 4
Lines: 19

Kuva 25. Henkilöhahmon Prince Fortinbras infosivu

Näytelmän päävalikkosivulla on vuorosanojen hakutoiminnon yhteydessä kolme eri alasvetovalikkoa (Search for Lines -lomake kuvassa 24). Ensimmäiseen listataan näytelmän henkilöhahmot. Toiseen alasvetovalikkoon haetaan ne näytökset, joissa ensimmäiseen alasvetovalikkoon valittu henkilöhahmo esiintyy. Kolmanteen alasvetovalikkoon haetaan puolestaan ne kohtaukset, prologit tai epilogit, jotka kuuluvat toisessa alasvetovalikossa valittuun näytökseen ja joissa ensimmäiseen alasvetovalikkoon valittu henkilöhahmo esiintyy. Käyttäjä valitsee näihin valikkoihin haluamansa kriteerit ja siirtyy eteenpäin sivulle, jossa näkyy valitun henkilöhahmon vuo-

rosanat valitussa kohdassa näytelmää. Lisäksi näytettävien puheiden (eli peräkkäisten vuorosanojen joukon) väleistä löytyy muiden kuin valitun henkilöhahmon puheiden lukumäärä kussakin kohdassa. Kuvassa 26 on esimerkki valitut vuorosanat näyttävästä sivusta.



Kuva 26. Henkilöhahmon Danes vuorosanat neljännen näytöksen viidennessä kohtauksessa.

Näytelmän päävalikkosivulta löytyy linkki ”Table of Contents”, jota klikkaamalla käyttäjä pääsee näytelmän sisällysluettelosivulle. Tällä sivulla on listattuna näytelmän näytösten, kohtausten, prologien ja epilogien otsikot siten, että kukin otsikko toimii linkkinä, jonka kautta käyttäjä pääsee siirtymään kyseiseen kohtaan näytelmää. Lisäksi kunkin näytöksen, prologin ja epilogin yhteyteen listataan kaikki ne henkilöhahmot, jotka esiintyvät kyseisessä osiossa. Myös listatut henkilöhahmot toimivat linkkeinä, joiden kautta käyttäjä pääsee kyseisen henkilöhahmon infosivulle.

Näytelmän päävalikkosivun linkin ”Full Summary” kautta käyttäjä pääsee puolestaan yhteenvetosivulle, jossa näkyy

- lukumäärät näytelmän näytöksistä, kohtauksista, henkilöhahmoista ja puheista
- listaus näytelmän kaikista henkilöhahmoista
- taulukko, jonka sarakkeina ovat henkilöhahmo, puheet ja vuorosanat.

Käyttäjä voi järjestää em. taulukon joko henkilöhahmojen mukaiseen aakkosjärjestykseen tai suuruusjärjestykseen joko puheiden tai vuorosanojen lukumäärän perusteella. Kuvassa 27 on ote näytelmän Macbeth yhteenvetosivusta.

Play Statistics

- 5 acts
- 28 scenes
- 41 characters
- 649 speeches

Dramatis Personae

- DUNCAN, king of Scotland.

his sons.

- MALCOLM
- DONALBAIN

generals of the king's army.

- MACBETH
- BANQUO

Kuva 27. Ote Macbeth-näytelmän yhteenvetosivusta. Ote sisältää statistiikkaa näytelmästä sekä näytelmän henkilöhahmojen listausta.

Kaiken kaikkiaan eXist-tietokantajärjestelmä osoittautui hyvin käteväksi työkaluksi kokeellisen osion toteuttamisessa. XQuery on työkaluna hyvin käytännöllinen, tehokas sekä suhteellisen helposti opittavissa oleva XML-kyselykieli.

7 Yhteenveto ja pohdinta

Tämän tutkielman tarkoituksena oli tarjota lukijalle yleiskatsaus natiiveista XML-tietokannoista. Tutkielmassa esiteltiin pääpiirteittäin XML-kielen perusteet, XML-käsittelykielet XPath sekä XQuery, natiivien XML-tietokantojen tyypilliset piirteet, XML-muotoisen tiedon indeksointimenetelmät, XML-kyselyjen evaluointimenetelmät sekä transaktioidenhallinta natiiveissa XML-tietokannoissa. Tutkielman aihe oli laaja, minkä johdosta eri osa-alueista esiteltiin lähinnä yleiskatsaukset, menemättä erityisen syvälle näihin osa-alueisiin. Tutkielma tarjoaa natiiveista XML-tietokannoista laajan kokonaiskuvan, jonka pohjalta aiheen eri osa-alueisiin on mahdollista perehtyä entistä syvällisemmin.

XML-kielestä kasvoi 1990-luvun lopulla suosituin rakenteisten dokumenttien merkintäkieli, joka on vielä tänäkin päivänä suuressa käytössä hyvin monilla erilaisilla sovellusaloilla. Tulevaisuuden kannalta kenties tärkein perusta XML-kielen suuren suosion säilymisessä vaikuttaisi olevan Internetin alati kasvavassa käytössä. Eräs XML-kielen alkuperäisistä tavoitteista oli helpottaa kahden yhteensopimattoman järjestelmän välistä tiedonvaihtoa Internetin välityksellä. Internetin tietoliikenne on jatkuvassa kasvussa vuodesta toiseen, joten tältä osin myöskään XML:n suosion hiipumiselle ei ole merkkejä näköpiirissä.

Natiivien XML-tietokantojen kehittäminen on aloitettu 1990-luvun lopussa. Useat natiivien XML-tietokantojen käyttämät prosessointimenetelmät ovat edelleen tutkimusten kohteena natiivien XML-tietokantojen suhteellisen nuoren iän johdosta. Natiivien XML-tietokantojen perusteet ovat kuitenkin jo vakiintuneet, mutta tietokannat tulevat tästä huolimatta edelleen kehittymään lähivuosina. Natiivit XML-tietokannat eivät ole erityisen tunnettuja tiedonhallintatyökaluja nykyään: perinteiset relaatiotietokantajärjestelmät ovat monille tuttu ja turvallinen valinta myös XML-muotoisen tiedon hallintaan. Kuitenkin XML-tiedon hallinta – kuten esimerkiksi luvussa 6 esitellyn web-palvelun toteutus – on paljon kätevämpää ja ennen kaikkea tehokkaampaa natiivin XML-tietokannan kuin relaatiotietokantajärjestelmän avulla.

Tässä tutkielmassa natiiveja XML-tietokantoja on vertailtu jossain määrin relaatiotietokantajärjestelmiin. Natiivit XML-tietokannat ja relaatiotietokantajärjestelmät eivät kuitenkaan varsinaisesti kilpaile keskenään, sillä molemmille tietokannoille on oma tilauksensa. Tästä johtuen natiivien XML-tietokantojen tarkoituksena ei ole koskaan syrjäyttää relaatiotietokantajärjestelmiä. Relaatiotietokantajärjestelmät tarjoavat tehokkaimman ratkaisun sellaisen tiedon hallintaan, joka voidaan esittää relaatiotauluina. Sen sijaan XML-muotoinen tieto mahdollistaa relaatiotauluja suuremman ilmaisuvoiman. Tämä tarkoittaa sitä, että natiivien XML-tietokantojen avulla voidaan puolestaan hallita tehokkaasti sellaista monimutkaisen rakenteen omaavaa tietoa, jota ei voida esittää järkevästi relaatiotaulumuodossa.

Lähteet

Abiteboul, S., Manolescu, I., Rigaux, P., Rousset, M., Senellart, P. (2011) *Web Data Management*. Cambridge University Press, Iso-Britannia.

Al-Khalifa, S., Jagadish, H., Koudas, N., Patel, J., Srivastava, D., Wu, Y. (2002) Structural Joins: A Primitive for Efficient XML Query Pattern Matching. *18th International Conference on Data Engineering,* IEEE Computer Society Press, Yhdysvallat, 141-152.

Ausbrooks, R., Buswell, S., Carlisle, D., Chavchanidze, G., Dalmas, S., Devitt, S., Diaz, A., Dooley, S., Hunter, R., Ion, P., Kohlhase, M., Lazrek, A., Libbrecht, P., Miller, B., Miner, R., Rowley, C., Sargent, M., Smith, B., Soiffer, N., Sutor, R., Watt, S. (2010) *Mathematical Markup Language (MathML) Version 3.0.* W3C Recommendation `http://www.w3.org/TR/MathML3` (15.3.2013).

Boag, S., Chamberlin, D., Fernándes, M., Florescu, D., Robie, J., Siméon, J. (2010) *XQuery 1.0: An XML Query Language (Second Edition).* W3C Recommendation `http://www.w3.org/TR/xquery` (31.5.2013).

Bos, B., Celik, T., Hickson, I., Lie, H. (2011) *Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification.* W3C Recommendation `http://www.w3.org/TR/CSS21` (20.5.2013).

Bourret, R. (2005) *XML and Databases.* Nettiartikkeli XML:n käytöstä tietokannoissa `http://www.rpbourret.com/xml/XMLAndDatabases.htm` (20.1.2013).

Boyer, J. (2009) *XForms 1.1.* W3C Recommendation `http://www.w3.org/TR/xforms` (20.5.2013).

Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., Yergeau, F. (2008) *Extensible Markup Language (XML) 1.0 (Fifth Edition).* W3C Recommendation `http://www.w3.org/TR/2008/REC-xml-20081126` (15.3.2013).

Chaudhri, A., Rashid, A., Zicari, R. (2003) *XML Data Management*. Addison-Wesley, Yhdysvallat.

Chuang, P., Lu, E., Wu, P. (2006) An empirical study of XML data management in business information systems. *The Journal of Systems and Software* **79**, 984-1000.

Clark, J. (1999) *XSL Transformations (XSLT).* W3C Recommendation `http://www.w3.org/TR/xslt` (20.5.2013).

Clark, J., DeRose, S. (1999) *XML Path Language (XPath).* W3C Recommendation `http://www.w3.org/TR/xpath/` (22.1.2013).

Eswaran, K., Gray, J., Lorie, R., Traiger, I. (1976) The Notions of Consistency and Predicate Locks in a Database System. *Communications of the ACM* **19**(11), 624-633.

Fiebig, T., Helmer, S., Kanne, C., Moerkotte, G., Neumann, J., Schiele, R., Westmann, T. (2002) Anatomy of a native XML base management system. *The VLDB Journal* **11**, 292-314.

Fielding, R., (2000) *Architectural Styles and the Design of Network-based Software Architectures.* Väitöskirja. Kalifornian yliopisto, Irvine.

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J., Nielsen, H., Karmarkar, A., Lafon, Y. (2007) *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition).* W3C Recommendation `http://www.w3.org/TR/soap12-part1` (20.5.2013).

Haustein, M., Härder, T. (2004) A Lock Manager for Collaborative Processing of Natively Stored XML Documents. *Proceedings of 19th Brazilian Symposium on Databases*, SBBD, Brasilia, 230-244.

Haustein, M., Härder, T. (2007) An efficient infrastructure for native transactional XML processing. *Data & Knowledge Engineering* **61**, 500-523.

Haw, S., Lee, C. (2011) Data storage practices and query processing in XML databases: A survey. *Knowledge-Based Systems* **24**(8), 1317-1340.

Helmer, S., Kanne, C., Moerkotte, G. (2003) Lock-based Protocols for Cooperation on XML Documents. *Proceedings of the 14th International Workshop on Database and Expert Systems Applications*, IEEE, Yhdysvallat, 230-234.

ISO (1986) *ISO 8879: Standard Generalized Markup Language (SGML).* International Organization for Standardization, Geneve, Sveitsi.

Kredel, H. (2013) *XPath und XPointer*. Kalvosarja osoitteessa `http://krum.rz.uni-mannheim.de/inet-2004/sess-302.html` (20.1.2013).

Laux, A., Martin, L. (2000) *XML:DB Initiative: XUpdate – XML Update Language*. XUpdate-kielen spesifikaatio `http://xmldb-org.sourceforge.net/xupdate/xupdate-wd.html` (20.5.2013).

Marsh, J., Orchard, D., Veillard, D. (2006) *XML Inclusions (XInclude) Version 1.0 (Second Edition).* W3C Recommendation `http://www.w3.org/TR/xinclude` (20.5.2013).

Mathis, C. (2009) *Storing, Indexing, and Querying XML Documents in Native XML Database Management Systems.* Väitöskirja. Kaiserslauternin yliopisto, tietojenkäsittelytieteen laitos.

Megginson, D. (2004) *SAX*. SAX-projektin kotisivu `http://www.saxproject.org/` (15.3.2013).

Meier, W. (2013) *eXist-db Open Source Native XML Database.* eXist-tietokannan kotisivu `http://exist-db.org` (23.5.2013).

Pemberton, S., Austin, D., Axelsson, J., Çelik, T., Dominiak, D., Elenbaas, H., Epperson, B., Ishikawa, M., Matsui, S., McCarron, S., Navarro, A., Peruvemba, S., Relyea, R., Schnitzenbaumer, S., Stark, P. (2002) *XHLTM TM 1,0 The Extensible Hy-*

perText Markup Language (Second Edition). W3C Recommendation `http://www.w3.org/TR/xhtml1` (15.3.2013).

Pritchett, D. (2008) BASE: An Acid Alternative. *ACM Queue* **6**(3), 48-55.

Reed, D. (1978) *Naming and Synchronization in a Decentralized Computer System*. Väitöskirja. Massachusetts Institute of Technology.

Robie, J., Chamberlin, D., Dyck, M., Florescu, D., Melton, J., Siméon, J. (2011) *XQuery Update Facility 1.0*. W3C Recommendation `http://www.w3.org/TR/xquery-update-10` (20.5.2013).

Sciore, E. (2008) *Database Design and Implementation.* Wiley, Yhdysvallat.

Shan, W., Liao, H., Jin, X. (2012) XML Concurrency Control Protocols: A Survey. *WAIM 2012: The 13th International Conference on Web-Age Information Management*, Springer, Kiina, 299-308.

Sperberg-McQueen, C., Thompson, H. (2000) *XML Schema*. W3C Recommendation `http://www.w3.org/XML/Schema` (15.3.2013).

W3C (2005) *Document Object Model (DOM).* W3C Recommendation `http://www.w3.org/DOM/` (15.3.2013).

Walmsley, P. (2007) *XQuery*. O'Reilly, Yhdysvallat.

Walsh, N. (1998) *A Technical Introduction to XML.* Nettiartikkeli XML-kielestä `http://www.xml.com/pub/a/98/10/guide0.html` (20.1.2013).

Wang, H., Park, S., Fan, W., Yu, P. (2003) ViST: a dynamic index method for querying XML data by tree structures, *Proceedings of the 2003 ACM SIGMOD international conference on Management of data,* ACM, Yhdysvallat, 110-121.

Win, K., Ng, W., Liu, Q., Lim, E. (2003) XStamps: A Multiversion Timestamps Concurrency Control Protocol for XML Data. *Proceedings of the 2003 Joint*

Conference of the Fourth International Conference on Information, Communications and Signal Processing, IEEE, Singapore, 1650-1654.

Winer, D. (1999) *XML-RPC Specification.* XML-RPC-protokollan spesifikaatio `http://xmlrpc.scripting.com/spec.html` (20.5.2013).

Zou, Q., Liu, S., Chu, W. (2004) Ctree: A Compact Tree for Indexing XML Data. *Proceedings of the 6th annual ACM international workshop on Web information and data management*, ACM, Yhdysvallat, 39-46.