LAPPEENRANNAN TEKNILLINEN YLIOPISTO
Tietotekniikan osasto

Diplomityö

XML-VERKKOTIETOKANTOJEN SUORITUSKYKYVERTAILU

Diplomityön aihe hyväksytty tietotekniikan osaston osastoneuvostossa 12.10.2005
Työn tarkastajat ovat Professori Jari Porras ja Yliassistentti Pekka Jäppinen
Työn ohjaaja on Professori Jari Porras

Lappeenrannassa 21.12.2005

Henri Laamanen
Huopatehtaankatu 2 D 18
53900 Lappeenranta
Puhelin +358 50 531 4280

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto

Tietotekniikan osasto

Henri Laamanen

XML-verkkotietokantojen suorituskykyvertailu

Diplomityö

2005

95 sivua, 24 kuvaa, 10 taulukkoa, 2 liitettä

Tarkastajat: Professori Jari Porras

Yliassistentti Pekka Jäppinen

Hakusanat: XML-tiedonhallintajärjestelmä, natiivi XML -tietokanta, suorituskyky

XML:n kasvava suosio dokumenttiformaattina sekä sen alati monipuolistuva käyttö ovat lisänneet XML-tiedonhallintajärjestelmien tarvetta. Yksi tapa XML-dokumenttien hallintaan on edelleen tiedostopohjainen järjestelmä. Erilaisiin tietokantoihin perustuvat XML-tiedonhallintajärjestelmät ovat kuitenkin viime vuosina kasvattaneet suosiotaan monipuolisempien ominaisuuksien ja paremman suorituskyvyn takia. Lisäksi XML-dokumenttien hallinta tiedostopohjaisessa järjestelmässä käy lähes mahdottomaksi suurilla datamäärillä. Suhteellisen uutena tulokkaana XML-dokumenttien hallintaan ovat tulleet natiivit XML -tietokannat, jotka ovat suunniteltu juuri XML:ää silmälläpitäen.

Tässä diplomityössä esitellään erilaisia XML-tiedonhallintajärjestelmiä. Erityisesti relaatiotietokantoihin ja natiiveihin XML -tietokantoihin perustuvien ratkaisujen taustoihin ja teknisiin yksityiskohtiin yritetään luoda syvällinen katsaus.

Neljälle XML-tiedonhallintaratkaisulle - Binary Approachille, Edge Approachille, eXistille ja Xindicelle - suoritetaan XMach-1 suorituskykytesti. Lisäksi testattavien ratkaisujen teknistä toimivuutta arvioidaan sekä analyyttisen että käytännön tarkastelun kautta. Suorituskykytestien ja teknisen toimivuuden arvioiden perusteella on tarkoitus valita XML-tiedonhallintaratkaisu Javalla toteutetulle Web-sovellukselle, joka käyttää XML:ää tietojen tallennusformaattina.

ABSTRACT

Lappeenranta University of Technology

Department of Information Technology

Henri Laamanen

Benchmarking of XML-databases

Master's Thesis

2005

95 pages, 24 figures, 10 tables, 2 appendices

Supervisors: Professor Jari Porras

Senior Assistant Pekka Jäppinen

Keywords: XML database management system, Native XML database, benchmarking

The growing popularity of XML as a document format and its various using purposes has increased the need of XML management systems. One simple way to manage XML-documents is to use file system. XML management systems based on different databases have gained more popularity because of versatile features and better performance. Besides, management of large amount of XML-documents is rather difficult in a system based on file system. Native XML databases, which are designed to be XML-compatible, are newcomers in group of XML management systems.

Different kinds of XML management systems are introduced in this thesis. An inspection for solutions and techniques behind those XML management systems is also performed. A special attention is paid for XML management systems which are based on relational databases and native XML databases.

XMach-1 benchmark will be performed for four different XML management systems. The benchmarked solutions will be Binary Approach, Edge Approach, eXist and Xindice. A technical evaluation will be performed for systems under test through analytical and practical inspection. The results of benchmarks and technical evaluations will be used as a criterion when deciding suitable XML management system for Web-based application implemented with Java.

SISÄLLYS

LYHENTEET

A

ACID, Atomicity, Consistency, Isolation, Durability

API, Application Program Interface

B

BLOB, Binary Large Object

C

CLOB, Character Large Object

CORBA, Common Object Request Broker Architecture

D

DBMS, Database Management System

DCOM, Distributed Common Object Model

DOM, Document Object Model

DTD, Document Type Definition

H

HTML, HyperText Mark up Language

I

I/O, Input/ Output

ISAM, Index-Sequential Access Method

J

JDBC, Java Database Connectivity

JDK, Java Development Kit

O

OCBD, Open Database Connectivity

OQL, Object Query Language

P

PRIX, Prüfer sequences for Indexing XML

R

RDBMS, Relational Database Management System

REST, Representational State Transfer

RPC, Remote Procedure Call

S

SAX, Simple API for XML

SGML, Standard Generalized Markup Language

SOAP, Simple Object Access Protocol

SQL, Structured Query Language

T

TCO, Total Cost of Ownership

V

ViST, Virtual Suffix Tree

W

WebDAV, Web-based Distributed Authoring and Versioning

X

XDB, XML Database

XDBMS, XML Database Management System

XISS, XML Indexing and Storage System

XIST, XML Index Selection Tool

XMach-1, XML Data Management Benchmark

XLP, XPath Locking Protocol

XML, eXtensible Markup Language

XML:DB, XML Database Application Program Interface

XML-QL, A Query Language for XML

XMLDBMS, XML Database Management System

XPath, XML Path Language

XQL, XML Query Language

Xqps, XML queries per second

XML-RPC, XML Remote Procedure Call

1. JOHDANTO

XML:n lisääntynyt käyttö dokumentti- sekä dataformaattina on luonut tarpeen erityisille XML-dokumenttien hallintaan soveltuville järjestelmille. Ensimmäisten XML-tiedonhallintajärjestelmien kehittäminen aloitettiin 1990-luvun lopulla. Vuonna 2000 Software AG julkaisi ensimmäisen kaupallisen XML-tiedonhallintajärjestelmän Taminon ja lanseerasi samalla käsitteen natiivi XML -tietokanta. Sittemmin XML-tiedonhallintajärjestelmiä on kehitetty useiden eri yritysten ja kehitysyhteisöjen toimesta. XML-tiedonhallintajärjestelmät voidaan jakaa kahteen kategoriaan: XML-tuellisiin ja natiiveihin XML -tiedonhallintajärjestelmiin.

Tämän diplomityön tarkoituksena on tutkia olemassa olevia XML-tuellisia ja natiiveja XML -tiedonhallintaratkaisuja. Tutkittujen XML-tiedonhallintaratkaisujen joukosta valitaan neljä ehdokasta suorituskykytestiin. Suorituskykytestit suoritetaan käyttäen XMach-1 -testiä, joka pyrkii mallintamaan usean yhtäaikaisen käyttäjän tilannetta verkkokäytössä. Nykyään yksi tyypillinen käyttötarkoitus XML-tiedonhallintajärjestelmälle on toiminta osana verkkosovellusta, joten XMach-1 on sopiva valinta suorituskykytestiksi.

XML-tiedonhallintajärjestelmien suorituskyvyn mittauksesta on olemassa useita edeltäviä tutkimuksia. Natiivien XML -tietokantojen on todettu soveltuvan paremmin suurikokoisille XML-dokumenteille kuin XML-tuellisten tietokantojen. Indeksoinnin on todettu parantavan natiivien XML -tietokantojen suorituskykyä. Relaatiotietokantoihin perustuvien XML-tiedonhallintajärjestelmien on havaittu skaalautuvan hyvin eri tietomäärille sekä tarjoavan korkean luotettavuuden. Skaalautuvuuden lisäksi relaatiotietokantapohjaisia järjestelmiä on mahdollista käyttää myös muuntyyppisen tiedon hallintaan XML:n rinnalla.

Suorituskykytestin sekä sen valmistelusta ja suorittamisesta saatujen kokemusten ja tulosten perusteella valitaan XML-tiedonhallintajärjestelmä verkossa toimivalle testaussovellukselle.

2. JOHDATUS XML:ään

Kun XML-standardista julkaistiin ensimmäinen luonnos marraskuussa vuonna 1996 (Connolly 2003), ei varmaankaan osattu odottaa sen saavuttavan nykyisen kaltaista valta-asemaa. Alun perin XML:ää käytettiin tiedonvaihdon formaattina, mutta tänä päivänä XML on laajasti käytössä kaikkialla missä rakenteista dataa käytetään (Brandin 2003, s. 3).

2.1 Mikä XML on?

W3C määrittelee XML:n seuraavasti: "XML on yksinkertainen, erittäin joustava, SGML:sta johdettu tekstiformaatti." (W3C 2005a). W3C:n määritelmä on pelkistetty ja yksinkertainen mutta pätevä, sillä se sisältää viittauksen SGML:een. SGML on rakenteisten dokumenttien kuvaukseen tarkoitettu kieli, jonka ensimmäinen kansainvälinen standardi julkaistiin vuonna 1986 (Price 1997, s. 172).

Koska SGML on useimmiten liian laaja kokonaisuus jokapäiväistä ja yleistä käyttöä ajatellen, on siitä johdettu useita kieliä joista tunnetuimmat ovat HTML ja XML. XML on merkkauskieli, joka on suunniteltu rakenteisen tiedon esittämiseen. HTML on sen sijaan ulkoasun kuvaamiseen soveltuva merkkauskieli, jonka lähtökohtana on ollut WWW ja hyperteksti (Vanha-Vuori 2005).

2.2 XML:n ominaisuudet

Seuraavissa kolmessa kappaleessa tutustutaan hieman XML:n ominaisuuksiin ja luonteeseen. Rakenteisuus on yksi XML:n ominaisuuksista, johon on hyvä tutustua

hieman lähemmin. Rakenteisuuden lisäksi XML:stä puhuttaessa törmätään usein käsitteeseen XML-dokumentti. Kappaleessa 2.2.2 käsitellään lyhyesti XML:n dokumentti- ja datakeskeisyyttä. Lopuksi tutustutaan XML-dokumentin määrittelytekniikoihin: XML-skeemaan ja DTD:hen.

2.2.1 Rakenteisuus

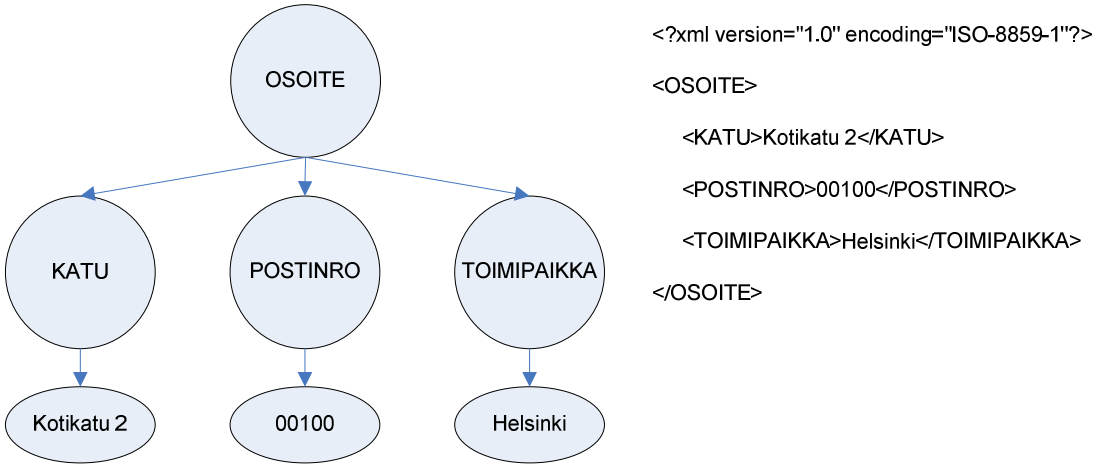
XML on siis tarkoitettu rakenteisen tiedon esittämiseen ja kuvaamiseen. Mutta mitä rakenteinen tieto on? Vanha-Vuoren mukaan rakenteisella tiedolla tarkoitetaan tietoa, joka koostuu informaation lisäksi rakennekuvauksesta (Vanha-Vuori 2005). Esimerkkinä rakenteisesta tiedosta toimii osoite, joka koostuu mm. katu- ja postiosoitteesta. Edellä mainitut kaksi termiä kuvaavat tietoa, joten ne antavat tiedolle merkityksen. Esimerkiksi pelkkä numerosarja 00100 ei kerro lukijalle itsessään mitään, mutta kun ympärille lisätään kuvaus rakenteesta ja merkityksestä, tulee tiedosta käyttökelpoista myös sellaiselle henkilölle, jolle tuo numerosarja ei ole tuttu. XML:ssä tiedolle annetaan merkitys lisäämällä sen ympärille merkitystä kuvaavat tunnisteet eli tagit: <postinumero>00100</postinumero>.

Tietoa voidaan hierarkisoida käyttämällä tunnisteita sisäkkäin:

```
<osoite>
        <katu>Kotikatu 2</katu>
        <postinumero>00100</postinumero>
</osoite>
```

Nyt lukija, on se sitten kone tai ihminen, tietää osoitteen koostuvan Kotikatu 2 nimisestä kadusta sekä postinumerosta 00100. XML-dokumentin sisällä vallitsevan hierarkian johdosto XML-dokumentti voidaankin käsittää puurakenteena, joka koostuu solmuista ja lehdistä. Kuvassa yksi on esitetty esimerkkidokumentti sekä sen

puurakenne. Kuvan yksi puurakenteessa osoite-solmu on myös dokumentin ns. dokumentti- tai juurisolmu. Tavallisia solmuja edustavat solmut katu, postinro ja toimipaikka, joita seuraavat lehtisolmut. Varsinainen tieto on esitetty lehtisolmuissa. Toisin kuin HTML:ssä, XML:n tunnisteiden (tag) nimiä ei ole ennalta määritelty, vaan ne ovat dokumenttien laatijan päätettävissä (Vanha-Vuori 2005).



Kuva 1. XML-dokumentin puurakenne

Brandin esittää kirjoituksessaan rakenteisuuden lisäksi kolme ominaisuutta, jotka tekevät XML:stä erittäin ilmaisuvoimaisen kielen (Brandin 2003, s. 4):

- *Heterogeenisyys*: Reaalimaailman tietoa ei voida organisoida ennalta määriteltyjen, kiinteiden sääntöjen mukaan. XML:n avulla tietoa voidaan määritellä ilman rajoituksia.
- *Laajennettavuus*: Uudentyyppistä tietoa voidaan määritellä ja lisätä tarvittaessa.
- *Joustavuus*: Uudelle ja olemassa olevalle tiedolle ei ole ennalta määritelty kiinteää kokoa ja ominaisuuksia, vaan ne voidaan määritellä aina tapauskohtaisesti.

Heterogeenisyys, laajennettavuus ja joustavuus tekevät XML:stä dynaamisen merkkauskielen. Dynaamisuuden ansiosta XML on selkeästi skaalautuvampi perinteisiin tiedonhallintamenetelmiin kuten relaatiotietokantoihin ja proseduraalisiin dokumentteihin verrattuna. XML:n dynaamisuus tuo mukanaan huomattavia etuja, mutta myös monia haasteita, joista yksi merkittävimmistä on kasvanut tallennuskapasiteetin tarve; rakenteinen tieto vie huomattavasti enemmän tilaa kuin ei-rakenteinen tieto. XML:n haittapuoliin palataan tulevissa kappaleissa.

2.2.2 Dokumentti vai dataa

Kun tietoa tallennetaan XML-muotoon, on lopputuloksena dokumentti. Jopa edellisessä kappaleessa esitetty osoite-esimerkki on XML-dokumentti, kunhan vain sen eteen liitetään tarvittavat tunnistetiedot. Tunnistetietojen tarkoituksena on kertoa, että kyseinen dokumentti on XML:ää ja mitä merkistöä se käyttää. Tunnistetietoihin voidaan liittää myös muuta informaatiota dokumentista. W3C:n alkuperäisenä ajatuksena oli kehittää XML:stä merkkauskieli suurten elektronisten dokumenttien julkaisuun (W3C 2005a), mutta vuosien saatossa XML on otettu käyttöön myös datapohjaisen tiedon kuvauksessa. Chaudri et al. jakavat XML-dokumentit kolmeen tyyppiin: dokumenttikeskeisiin, datakeskeisiin ja hybrididokumentteihin (Chaudri et al. 2003, s. xxi). Dokumenttikeskeisissä XML-dokumenteissa tieto koostuu ei-rakenteisesta tiedosta kuten artikkeleista, sähköposteista tai jopa kirjoista. Datakeskeisissä dokumenteissa tieto on rakenteista ja se voi koostua esimerkiksi katalogeista, tilauksista tai laskuista. Hybrididokumentit koostuvat sekä dokumentti- että datakeskeisestä tiedosta (Chaudri et al. 2003, s. xxi). Kun XML-dokumentteja tallennetaan tietokantoihin, XML-dokumenttien tyypeillä on suuri merkitys valittaessa tietokantaratkaisua.

2.2.3 Skeemat

Brandinin mukaan XML antaa dokumenttien laatijalle vapaat kädet tiedon määrittelyssä (Brandin 2003, s. 4). XML:n suoma vapaus tiedon mallintamisessa aiheuttaa myös ongelmia. Dokumentin laatija voi helposti määritellä tiedon väärin tai tehottomasti. Brandin esittää kuusi yleistä virhettä XML-dokumenttia luotaessa (Brandin 2003, s. 9):

- Puutteellinen konteksti kuvattaessa tietoa (tunnisteiden puutteellinen käyttö)
- Puutteelliset ohjeet tiedon tulkitsemiseksi (attribuuttien puutteellinen käyttö)
- Tiedon kuvaaminen käyttäen attribuutteja (attribuuttien väärä käyttö)
- Tietoelementtien käyttäminen metatietona (tunnisteiden väärä käyttö)
- Dokumentin väärä tai puutteellinen hierarkia
- Väärien tai epäolennaisten attribuuttien käyttäminen

Kuten edellä esitetystä listasta voidaan huomata, dokumentin laatija voi tehdä virheitä monessa kohdassa. Tiedon mallintaminen käyttäen XML:ää voidaan rinnastaa tietokannan suunnitteluun. Tietokantaa suunniteltaessa tulee tieto mallintaa tauluihin siten, että tietokantaa voidaan pitää normalisoituna. Väärien suhteiden ja puutteellisten sarakemäärittelyjen seurauksena tietokanta voi aiheuttaa pitkällä aikavälillä enemmän haittaa kuin hyötyä. Näin on myös datakeskeisten XML-dokumenttien kanssa.

W3C on julkaissut standardit XML-dokumenttien määrittelyä varten. XML-dokumenttien määrittely tehdään laatimalla skeema. Skeema on määritelmä tai kaava, jonka XML-dokumentin tulee täyttää, jotta sitä voidaan pitää kelvollisena. Skeemoja ei yleensä sisällytetä XML-dokumentteihin, vaan itse dokumentti ja skeema ovat kaksi erillistä objektia. Hyvin suunnitellun ja toteutetun skeeman avulla voidaan välttää tiedon mallintamisessa esiintyvät ongelmat. Skeemaa käyttämällä XML-dokumentin rakenne, sisältö ja semantiikka voidaan määritellä kaikille yhteisiksi,

jolloin dokumentin sisältämä tieto voidaan ymmärtää kaikkialla oikein (W3C 2005b). Tällä hetkellä on olemassa kaksi skeemastandardia: DTD ja XML Schema, joista jälkimmäisestä käytetään jatkossa nimitystä skeema. Skeemastandardeista DTD on vanhempi ja on perintöä SGML-kielestä. Vaikka DTD on tällä hetkellä selkeästi kahdesta skeemastandardista käytetympi, on sen asema heikentymässä, sillä skeema on huomattavasti ilmaisuvoimaisempi ja monipuolisempi määrittelytekniikka XML-dokumenteille. Skeemojen käyttämistä lisää myös se, että ne ovat puhtaasti XML-kielellä toteutettuja kun taas DTD pohjautuu SGML-kieleen. Skeeman yleistymistä hillitsee kuitenkin sen monimutkainen rakenne verrattuna DTD:hen.

Skeemojen, oli kyseessä sitten DTD tai XML Schema, käyttö on vapaaehtoista. XML-dokumentteja voidaan laatia ilman skeemoja, mutta niiden käyttäminen on suositeltavaa varsinkin jos dokumentti on luonteeltaan datakeskeinen.

3. TIEDONHALLINTAJÄRJESTELMÄT JA XML

Tiedonhallintajärjestelmällä (DBMS) tarkoitetaan tietokannan lisäksi siihen liittyviä sovelluksia, joilla voidaan tallentaa, muokata ja noutaa tietoa tietokannoista (Lewis et al. 2002, s.4). Tiedonhallintajärjestelmällä ei siis tarkoiteta pelkkää tietokantaa, joskin tiedonhallintajärjestelmä voi koostua pelkästä tietokannasta, sillä useissa tietokannoissa on valmiina toiminnot tiedon tallentamiselle, noutamiselle ja muokkaamiselle. Tiedonhallintajärjestelmän ydin on tietokanta, jonka mukaan tiedonhallintajärjestelmä saa yleensä nimensä. Esimerkiksi relaatiotietokantaan perustuvasta tiedonhallintajärjestelmästä käytetään lyhennettä RDBMS ja natiiviin XML -tietokantaan perustuva tiedonhallintajärjestelmä tunnetaan lyhenteellä XDBMS, XMLDMBS tai XDB.

3.1 Relaatiotietokantapohjaiset tiedonhallintajärjestelmät

RDBMS on tällä hetkellä suosituin tiedonhallintajärjestelmätyyppi, sillä pääosa IT-infrastruktuurin omaavista yrityksistä omaa jonkinasteisen relaatiotietokantatoteutuksen (Edvards 2003, s. 189). Relaatiotietokantoja ei ole kuitenkaan suunniteltu XML:ää silmälläpitäen, sillä relaatiotietokantatekniikka on kehitetty paljon ennen XML:ää. XML:n käytön voimakas lisääntyminen on luonut relaatiotietokannoille kilpailijoita muun muassa oliotietokannoista ja natiiveista XML-tietokannoista. Relaatiotietokanta on kuitenkin säilyttänyt selkeän valta-aseman vallitsevana tietokantatekniikkana vaikka se ei olekaan välttämättä optimaalinen ratkaisu XML-pohjaisen tiedon hallintaan. Tähän on useita syitä. Edvardsin mukaan relaatiotietokantatoimittajat ovat toimineet nopeasti ja kehittäneet tuotteistaan XML-yhteensopivia (Edvards 2003, s. 190). Hohensteinin mielestä syynä

relaatiotietokantojen vahvaan suosioon ovat laaja levinneisyys, tekniikan kypsyys, tehokkuus ja skaalautuvuus sekä tehokas SQL-kyselykieli (Hohenstein 2003, s. 123).

3.2 Natiivi XML -tietokanta

XML:n leviäminen yhä useammalle tietotekniikan osa-alueelle sai tutkijat ja yritykset pohtimaan olisiko uudelle tietokantatyypille tarvetta, sillä relaatiotietokantojen tuki XML:lle oli 1990-luvun loppupuolella vielä puutteellinen. Vuonna 2000 saksalainen Software AG julkaisi natiivin XML -palvelinohjelmiston Taminon, joka sisälsi myös natiivin XML -tietokannan (Schöning 2003, s. 21). Taminoa voidaan pitää maailman ensimmäisenä natiivina XML -tietokantana, sillä Software AG lanseerasi termin natiivi XML -tietokanta Taminon julkaisun yhteydessä (Bourret 2004). Mutta mikä tarkalleen on natiivi XML -tietokanta? Software AG määrittelee natiivin XML -tietokannan järjestelmäksi, joka ei ole mikä tahansa umpimähkään valittua tietorakennetta tukeva tietokanta varustettuna XML:ää tukevalla kerroksella, vaan se on järjestelmä, joka on suunniteltu ja luotu käsittelemään nimenomaan XML:ää (Schöning 2003, s. 21). Bourretin käsitys natiivista XML -tietokannasta on Software AG:n määritelmää tarkempi ja laajempi. Bourretin määritelmä natiiville XML -tietokannalle koostuu kolmesta argumentista (Bourret 2004):

- Natiivi XML -tietokanta määrittelee loogisen mallin XML-dokumentille ja varastoi ja noutaa dokumentteja tuon mallin mukaan.
- Natiivin XML -tietokannan tietomallin perusyksikkö on XML-dokumentti, aivan kuten relaatiotietokannalla perusyksikkönä on rivi taulussa.
- Natiivi XML -tietokanta ei rakennu minkään tietyn tallennusmallin päälle, vaan se voidaan toteuttaa esimerkiksi relaatiotietokannan, oliotietokannan tai jopa indeksoitujen tiedostojen varaan.

Tässä diplomityössä natiivin XML -tietokannan määritelmä koostuu Bourretin ensimmäisestä ja toisesta argumentista sekä Software AG:n määritelmästä yhdistettynä argumenttiin, joka kuuluu: ”natiivi XML -tietokanta soveltuu vain XML-dokumenttien hallintaan”. Määritelmä tarkoittaa silloin sitä, että relaatio-, olio- tai muut ei-natiivit XML -tietokannat varustettuna XML-tuella eivät kuulu natiivien XML -tietokantojen joukkoon, vaan niistä käytetään nimitystä XML-tuelliset tietokannat.

3.3 XML:n vaatimukset tiedonhallintajärjestelmälle

Valtaosassa tiedonhallintajärjestelmiä tietovarastona toimii relaatiotietokanta, jota kehitettäessä XML oli tuntematon käsite. Relaatiotietokannoissa tieto on tallennettuna tauluihin, jotka koostuvat sarakkeista eli kentistä ja riveistä eli tietueista. Taulut voivat olla suhteessa toisiinsa eli niiden välillä voi vallita relaatio. Taulujen kentillä on tunniste eli nimi, joka yksilöi taulussa olevan sarakkeen ja kertoo mahdollisesti mitä tietoa sarakkeen kentät sisältävät. Tällainen järjestelmä sopii hyvin datakeskeiselle tiedolle sekä tiedolle, jonka rakenne on ennalta määrätty. Tässä yhteydessä voidaan puhua ”tavanomaisesta tiedosta”.

Kuten edellisissä kappaleissa on esitetty, XML-pohjainen tieto poikkeaa tavanomaisesta tiedosta siinä, että se sisältää sekä datan että rakennekuvauksen, kun taas tavanomainen tieto koostuu pelkästään datasta. Relaatiotietokannan näkökulmasta katsottuna XML-pohjainen tieto tarvitsee tallennuskapasiteettia datalle, rakenteelle ja tunnisteille sekä tunnisteiden mahdollisille lisätiedoille eli attribuuteille. Taulukossa yksi on aiemmin kuvassa yksi esitetyn osoite-esimerkin tiedot esitetty relaatiotietokannan taulussa.

Taulukko 1. Osoitetiedot relaatiotietokannassa

OSOITE_ID	KATU	POSTINRO	TOIMIPAIKKA
1	“Kotikatu 2”	“00100”	“Helsinki”

Taulukossa kaksi osoite-esimerkin tiedot on esitetty XML-muodossa ja tallennettuna relaatiotietokantaan käyttäen edge approach -menetelmää (Tian et al. 2001, s. 8), josta kerrotaan lisää kappaleessa 4.1.2.

Taulukko 2. XML-dokumentti tallennettuna relaatiotietokantaan edge approach-menetelmällä

LÄHDE_ID	JÄRJ.NRO	TAGI	TAGI_ID	ARVO
1	1	OSOITE	2	NULL
2	1	KATU	3	“Kotikatu 2”
2	2	POSTINRO	4	“00100”
2	3	TOIMIPAIKKA	5	“Helsinki”

Kuten taulukkoja yksi ja kaksi vertaamalla voidaan huomata, XML-dokumentin tallentaminen relaatiotietokantaan on monimutkaisempaa ja enemmän tallennuskapasiteettia vaativaa kuin tavanomaisen, rakenteettoman tiedon. Jos lisäksi halutaan tallentaa XML-dokumentin skeema tai DTD, tallennuskapasiteettia tarvitaan lisää. Lawrence toteaa artikkelissaan XML:n käyttämisen lisäävän tiedon informatiivisuutta tallennuskapasiteetin kustannuksella (Lawrence 2004, s. 759).

Tian et al. mukaan relaatiotietokantaa käytettäessä XML-tiedon tallentamisesta ja hakemisesta aiheutuu runsaasti ylimääräistä, itse XML-tietoon kuulumatonta datan prosessointia eli niin sanottua overheadia (Tian et al. 2001, s. 3). Riippuen relaatiotietokannassa käytettävästä tallennusstrategiasta, liitosten (Join) suuri määrä aiheuttaa suorituskyvyn alenemista (Tian et al. 2001, s. 21-22). Myös tietojen päivittäminen XML-tuelliseen relaatiotietokantaan on hitaampaa verrattuna normaaliin relaatiotietokantaan. Samankaltaisia suorituskykyyn liittyviä huomioita esittävät mm. Fong et al. ja Lu et al., joskin molemmat ryhmät toteavat XML-tuellisten relaatiotietokantojen suorituskyvyn olevan natiivien XML-tietokantojen tasoa pienillä ja keskisuurilla tietomäärillä (Fong et al. 2003, s. 558- 563 & Lu et al.2005, 174- 189).

Edellä mainitut seikat liittyvät relaatiotietokantoihin, mutta XML-dokumenttien tallentamisessa esimerkiksi oliotietokantaan on myös omat ongelmansa erityisesti suorituskyvyn suhteen, sillä esimerkiksi DOM-objektien käsittely vaatii järjestelmältä runsaasti muistia sekä mahdollisia I/O-toimintoja.

Yhteenvetona voidaan todeta, että XML-dokumenttien rakenteisuus ja tunnistetiedot (tagit) lisäävät tarvittavan tallennuskapasiteetin määrää. Tietomäärien kasvaessa XML:n rakenteisuudesta johtuva ns. overhead kasvaa mikä aiheuttaa ongelmia suorituskykypuolella. XML-dokumenttien säilyttäminen vaatii tiedonhallintajärjestelmältä selvästi enemmän tallennuskapasiteettia ja suorituskykyä verrattuna tavanomaisen tiedon säilyttämiseen.

4. XML-TIEDONHALLINTARATKAISUT

Tässä kappaleessa tutustutaan tarkemmin teoriatasolla relaatiotietokantapohjaisiin XML-tiedonhallintaratkaisuihin sekä natiiveihin XML-tietokantoihin. Lisäksi kappaleen loppupuolella esitellään lyhyesti XML-dokumenttien hallintaa oliorelaatiotietokannassa.

4.1 XML-tiedon säilytysmenetelmät relaatiotietokannoissa

XML-dokumenttien säilyttämiseen relaatiotietokannoissa on kehitetty suuri määrä erilaisia ratkaisuja. XML-dokumentin rakenteella ja käyttötarkoituksella on suuri merkitys valittaessa säilytysratkaisua. Dokumenttikeskeisille XML-dokumenteille yksinkertainen ratkaisu saattaa soveltua hyvin, kun taas datakeskeisille XML-dokumenteille saatetaan joutua toteuttamaan monimutkaisempi ja yksityiskohtaisempi ratkaisu. Seuraavassa esitellään neljä erilaista tallennusratkaisua alkaen yksinkertaisesta XML-to-BLOB -ratkaisusta (XML-to-Binary Large Object) ja päätyen kekojoukkoon (Nested Set) perustuvaan ratkaisuun.

4.1.1 XML-to-BLOB

Tämä ratkaisu on erittäin yksinkertainen ja soveltuu pääasiassa dokumenttikeskeiselle XML-tiedolle. Ideana on tallentaa XML-dokumentti kokonaisuudessaan yhteen relaatiotietokannan tietueen BLOB-kenttään. Mikäli käytettävä tietokanta ei tue BLOB-tietotyyppiä, myös CLOB-tietotyyppiä (Character Large Object) tai vastaavaa voidaan käyttää. Avoimen lähdekoodin relaatiotietokantatoteutuksista MySQL tukee BLOB-tietotyyppiä ja PostgreSQL tukee TEXT-tietotyyppiä (vastaava kuin CLOB), joihin voidaan tallentaa merkkijonotietoa ennalta määrittelemätön määrä (MySQL

AB 2005 & PostgreSQL Global Development Group 2005). BLOB- tai TEXT-kenttään tallennetun XML-dokumentin sisältöä koskevia hakuja ei voida suorittaa suoraan ilman resursseja vaativia tyyppimuunnoksia. Tyyppimuunnokset voidaan välttää lisäämällä tietueeseen XML-dokumenttia kuvaavia kenttiä. Taulukossa kolme on yksinkertainen esimerkki XML-to-BLOB -menetelmästä. Sarakkeiden lukumäärää lisäämällä, dokumentin haettavuus paranee.

Taulukko 3. XML-dokumentti tallennettuna BLOB-tyyppiseen kenttään

ID	PVM	OMISTAJA	TYYPPI	XML
1	27-07-2005	Teppo Tuntematon	Osoitetieto	<?xml version="1.0" encoding="ISO-8859-1"?> <OSOITE> <KATU>Kotikatu 2</KATU> <POSTINRO>00100</POSTINRO> <TOIMIPAIKKA>Helsinki</TOIMIPAIKKA> </OSOITE>

XML-to-BLOB -menetelmän etuina voidaan pitää erittäin yksinkertaista toteutusta ja tietokannan pientä kuormitusta, sillä hauissa ei tarvita liitoksia (Joins). Menetelmällä on kuitenkin useita heikkouksia. Yhtenä heikkoutena on menetelmän soveltuvuus ainoastaan dokumenttipohjaiselle XML-tiedolle. Koska BLOB- tai TEXT-tyyppisen kentän sisältöön ei voida tehdä tehokkaita hakuja, ovat hakujen tulokset kokonaisia dokumentteja. Dokumenttia tietokantaan lisättäessä on taulun muihin sarakkeisiin tallennettava dokumenttia kuvaavia tietoja, jotta edes yksinkertaiset haut olisivat mahdollisia. Suurikokoiset XML-dokumentit voivat aiheuttaa myös suorituskyky- ja tallennuskapasiteettiongelmia. BLOB-kentän maksimikoko on MySQL-tietokannassa asetettu neljään gigatavuun (MySQL AB 2005) ja PostgreSQL-tietokannassa TEXT-kentän enimmäiskokoa ei ole määritetty (PostgreSQL Global Development Group 2005), joten kentän maksimikoko nousee vain erittäin harvoin ongelmaksi. Sen sijaan järjestelmän suorituskyky saattaa joutua koetukselle jo huomattavasti pienempikokoisilla dokumenteilla, jos järjestelmän toteutus on sellainen, että koko dokumentti noudetaan aina tietokannasta hakuja tehtäessä.

4.1.2 Edge Approach

Florescu ja Kossman esittävät tutkimusraportissaan useita menetelmiä XML-tiedon tallentamiseksi relaatiotietokantaan (Florescu & Kossman 1999a, s. 8-15). Yksi Florescun ja Kossmanin esittämistä menetelmistä on Edge Approach, jossa XML-dokumentti tallennetaan yhteen tauluun, mutta XML-to-BLOB -menetelmästä poiketen dokumenttia ei tallenneta kokonaisuudessaan yhteen kenttään.

Edge Approach -menetelmässä käytetään ns. Edge-taulua, joka koostuu viidestä sarakkeesta: lähde, järjestys, nimi, tyyppi ja arvo kuten taulukossa neljä. Jotta jokainen tietue voidaan yksilöidä, lisätään Edge-tauluun yleensä ID-sarake, joka toimii pääavaimena. Edge-taulun lähde-sarake ilmaisee minkä dokumentin elementistä on kyse. Vaihtoehtoisesti lähde-saraketta voidaan käyttää ilmaisemaan elementin hierarkiatasoa. Esimerkiksi taulukon neljä XML-sarakkeessa olevan dokumentin osoite-elementti on tasolla yksi ja katu-, postinumero- ja toimipaikkaelementit ovat tasolla kaksi. Järjestys-sarake kertoo samalla tasolla olevien elementtien järjestyksen. Nimi-sarakkeessa kerrotaan elementin nimi. Tyyppi-sarake, jonka käyttö on vapaaehtoista, ilmaisee elementin arvon tietotyypin. Viimeinen, eli arvo-sarake sisältää elementin arvon tai viittauksen toiseen olioon. Mikäli lähde-saraketta käytetään Florescun ja Kossmanin alkuperäisesti ehdottamalla tavalla ilmaisemaan dokumentin numeroa, täytyy dokumentin sisäinen hierarkia tällöin ilmaista arvo-kentässä siten, että tietueen arvo-kenttään laitetaan viittaus lapsielementtiin käyttäen lähde-saraketta.

Taulukossa neljä on esimerkki Edge-taulusta, jossa lähde-saraketta käytetään ilmaisemaan elementin tasoa. Jos samaan Edge-tauluun halutaan tallentaa useita eri XML-dokumentteja, voitaisiin tällöin dokumenttien yksilöimiseksi käyttää dokumentti_id- tai vastaavaa kenttää.

Taulukko 4. Esimerkki Edge-taulusta

ID	LÄHDE	JÄRJESTYS	NIMI	TYYPPI	ARVO
1	1	1	Osoite	String	NULL
2	2	1	Katu	String	Kotikatu 2
3	2	2	Postinro	String	00100

Florescu ja Kossman esittävät Edge Approachista myös hieman parannetun version, jossa elementtien arvot tallennetaan erillisiin arvo-tauluihin tietotyypin mukaan, ja tällöin edge-taulun arvo-sarake viittaa arvo-tauluun (Florescu & Kossman 1999a, s.12- 13). Tian et al. esittävät omassa tutkimuspaperissaan Edge-tauluun kohdeID-kenttää (TargetID), jolla mahdollistetaan XML-dokumentin hierarkian parempi kuvaaminen (Tian et al. 2001, s.8). KohdeID-kentän arvo 0 ilmaisee kyseessä olevan lehtisolmu tai attribuutti. Attribuutti kuvataan lisäksi järjestyssarakkeen arvolla 0. Mikäli elementillä on useita attribuutteja, on niiden järjestys sama kuin tietokannan taulussa.

Edge Approach -menetelmä on XML-to-BLOB -menetelmää jo huomattavasti kehittyneempi ratkaisu, jossa XML-dokumentti hajautetaan tietokannan taulun sarakkeisiin. Menetelmän etuna voidaan pitää yksinkertaista rakennetta, mutta toteutus on XML-to-BLOB -menetelmään verrattuna jo hieman monimutkaisempi. Tian et al. mukaan Edge Approachin suorituskykyyn vaikuttaa suuresti sarakkeiden indeksointi (Tian et al. 2001, s. 8-9).

4.1.3 Binary Approach

Binary Approach -menetelmässä, jonka Florescu ja Kossman esittelevät, on yhden taulun sijasta käytössä useita tauluja (Florescu & Kossman 1999b, s. 29). Menetelmässä jokainen elementti ja attribuutti tallennetaan omaan tauluunsa. Taulujen määrä kasvaa siten sitä mukaa, kun uudella nimellä varustettuja elementtejä

tai attribuutteja tallennetaan tietokantaan. Tauluja, joihin tieto tallennetaan, kutsutaan binääritauluiksi. Florescun ja Kossmanin mukaan binääritaulu koostuu nimestä sekä neljästä sarakkeesta: lähde, järjestys, tyyppi ja arvo kuten kuvassa kaksi. Sarakkeiden merkitys on sama kuin Edge Approach -menetelmässä. Binääritaulun nimi ilmaisee elementin tai attribuutin nimen. Elementin tai attribuutin nimi voidaan ilmaista taulun nimen sijaan myös erillisessä sarakkeessa siten, että binääritauluun lisätään elementin nimen ilmaiseva nimi-sarake. Kuvassa kaksi on esitetty osoite-esimerkki Binary Approach -menetelmällä.

LÄHDE	JÄRJESTYS	NIMI	TYYPPI	ARVO
1	1	Osoite	STRING	NULL

LÄHDE	JÄRJESTYS	NIMI	TYYPPI	ARVO
2	1	Katu	STRING	Kotikatu 2

LÄHDE	JÄRJESTYS	NIMI	TYYPPI	ARVO
2	2	Postinro	STRING	00100

LÄHDE	JÄRJESTYS	NIMI	TYYPPI	ARVO
2	3	Toimipaikka	STRING	Helsinki

Kuva 2. Esimerkki Binary Approach –menetelmästä

Vaikka Binary Approach -menetelmässä attribuutit tallennetaan omiin tauluihinsa aivan kuten elementit, niin silti attribuuttien käyttämisessä esiintyy ongelmia aivan kuten Edge Approach -menetelmässä. Attribuutti-ongelman ratkaisemiseksi voitaisiin käyttää Edge Approach -menetelmän kohdalla esiteltyä KohdeID-kenttää (Tian et al. 2001, s. 8). Tyyppi-kentän käyttäminen on edelleen vapaaehtoista ja sen tarpeellisuus on kyseenalaista, mikäli käytetään erillisiä tietotyyppikohtaisia arvo-tauluja, kuten Edge Approach -menetelmän kohdalla esitettiin.

Florescu ja Kossman esittävät myös vaihtoehtoisen rakenteen binääritaululle, jossa arvosarakkeita olisi kolmen tyyppisiä: merkkijono, lukuarvo ja viittaus (Florescu & Kossman 1999b, s. 30). Tällä menetelmällä null-arvojen määrä nousee suureksi, sillä

elementin tai attribuutin arvo voi olla vain yksi kolmesta tyypistä kahden muun saadessa null-arvon. Indekseiksi Florescu ja Kossman ehdottavat käytettävän lähde- ja arvo-sarakkeita.

Binary Approach -menetelmä on Edge Approach -menetelmän kaltainen menetelmä sillä erotuksella, että yhden taulun sijasta XML-dokumentit on hajautettu useampaan tauluun elementtien ja attribuuttien nimien perusteella. Lisäämällä sarakkeiden ja liitosten määrää, voidaan Binary Approach -menetelmän ominaisuuksia lisätä.

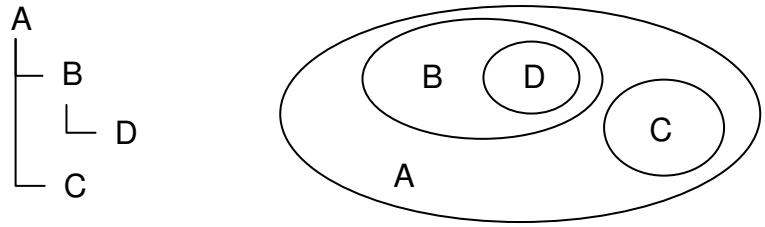
4.1.4 Nested Sets -malli

Viimeisenä relaatiotietokantaan soveltuvana mallina esitellään Nested Sets -malli, josta voidaan käyttää suomennosta koteloitu tai lomitettu joukko. Relaatiotietokantoihin on kehitetty eri menetelmiä puurakenteen tallentamiseksi tietokannan tauluihin. Yksi menetelmä on vieruslistaan perustuva malli, josta on esimerkki taulukossa viisi. Koska relaatiotietokannoissa käytetyin kyselykieli SQL on joukko-orientoitunut, vieruslistaan perustuva malli ei sellaisenaan ole kaikkein tehokkain tapa esittää hierarkkisia rakenteita relaatiotietokannoissa (Celco 2000).

Taulukko 5. Esimerkki vieruslistasta

Solmu	Vanhempi
A	NULL
B	A
C	A
D	B

Joukko alkioita voidaan myös esittää sekä puurakenteena että diagrammina kuten kuvassa kolme.

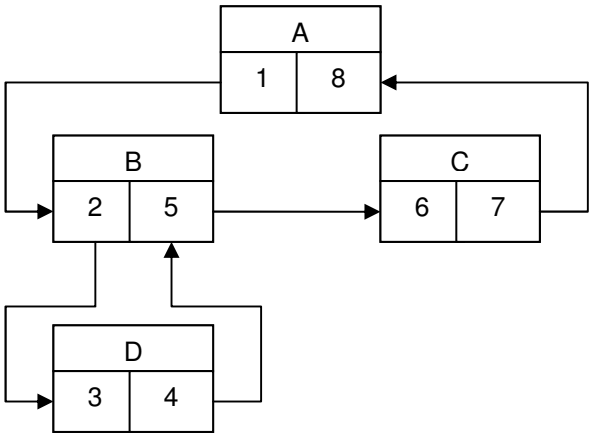


Kuva 3. Taulukon 5 vieruslista puurakenteena ja Nested Set -diagrammina

Kun vieruslista-tyyppistä ratkaisua ei haluta käyttää, puurakenteen hierarkian ilmaisemiseksi tarvitaan vaihtoehtoisia tekniikoita. Yksi tapa on lisätä solmuihin vasen-oikea-tunnistekentät (tai x, y) kuten kuvassa neljä. Tunnistekentät toimivat koordinaatteina ja mahdollistavat solmun paikallistamisen puussa (Develnet.org 2005). Edvards esittelee tämän menetelmän Nested Sets -mallina, jolla puurakenne voidaan esittää tehokkaasti SQL-pohjaisissa relaatiotietokannoissa (Edvards 2003, s. 195- 196). Nested Sets -mallissa puurakenteen solmut ja niiden keskinäiset suhteet määritellään seuraavasti (Edvards 2003, s. 195):

- Solmun (x, y) seuraavalle sisarukselle (x', y') pätee x' = y + 1
- Solmun (x, y) ensimmäiselle lapselle (x', y') pätee x' = x + 1
- Solmun (x, y) jälkeläisille (x', y') löytyvät x, x' ja y sekä x, y' ja y
- Solmun (x, y) esi-isille (x', y') pätevät x' < x ja x > y' sekä x', y.y'
- Solmun (x, y) vanhemmalla on suurin x' joukossa x', x.y'

Nyt kuvan kolme esimerkki voidaan esittää käyttäen Nested Sets- mallia sekä puurakenteena että tietokantatauluna. Kuvassa neljä on esitetty puurakenne käyttäen Nested Sets -mallia.



Kuva 4. Puurakenne esitettynä Nested Sets -mallilla

Sama esimerkki voidaan esittää myös tauluna, kuten taulukossa kuusi. Taulukossa on tässä tapauksessa vain kolme saraketta, mutta niiden määrää voidaan lisätä esimerkiksi arvo-sarakkeella.

Taulukko 6. Kuvan 4 Nested Sets -mallinen puurakenne esitettynä taulussa

Solmu	x	y
A	1	8
B	2	5
C	6	7
D	3	4

Nested Sets -mallia käytettäessä kaikkea tietoa ei tallenneta yleensä yhteen tauluun, vaan tauluja on käytössä useita. Edvards esittää yleisen arkkitehtuurin XML-dokumenttien tallentamiseksi relaatiotietokantaan, joka perustuu Nested Sets -malliin (Edvards 2003, s. 189- 258). Edvardsin esittämässä arkkitehtuurissa on käytössä kaksitoista taulua, mikä ei tosin ole suuri määrä verrattuna esimerkiksi Binary Approach -menetelmään, jossa kaksitoista taulua saavutetaan kahdellatoista erinimisellä elementillä ja attribuutilla. Taulujen suhteellisen suuri määrä johtuu Edvardsin mukaan siitä, että arkkitehtuuri ottaa huomioon kaikki XML-kielen

ominaisuudet, kuten CDATA-tyypin ja nimiavaruudet sekä arkkitehtuurin kyvystä toimia useamman kuin yhden dokumentin tietosäilönä (Edvards 2003, s. 196). Lisäksi tietokannan normalisointi lisää taulujen lukumäärää.

Edge ja Binary Approach -menetelmiin verrattuna myös Nested Set -malli käyttää hierarkian kuvaamiseen koordinaattien kaltaisia tunnisteita. Nested Set -mallin voidaan olettaa suoriutuvan hieman Edge ja Binary Approachia paremmin SQL-kyselyistä, mutta tietojen päivitys on sen sijaan oletettavasti hitaampaa johtuen suuresta koordinaattien päivitystarpeesta.

4.2 Natiivit XML-tietokannat

Siinä missä relaatiotietokannat toimivat tietovarastona monen tyyppiselle tiedolle, ovat natiivit XML-tietokannat kehitetty ja tarkoitettu vain XML-tiedon hallintaan. Tässä kappaleessa tutustutaan natiivien XML-tietokantojen teoriaan sekä niihin läheisesti liittyviin oheisteknologioihin.

4.2.1 Lähtökohdat ja oheisteknologiat

Natiivien XML-tietokantojen lähtökohtana on tarjota tiedonhallintajärjestelmä, joka on optimoitu XML:lle ja sen oheistekniikoille. XML:n oheistekniikoita ovat mm. kyselykielet, muunto- ja ulkoasukielet sekä parserit. Suurimmat eroavaisuudet XML-tuettujen ja natiivien XML-tietokantojen välillä ovat niiden tietomalleissa; relaatiotietokannoissa tietomalli perustuu relaatioihin, kun taas natiivien XML-tietokantojen tietomalli perustuu XML:ään (Bourret 2004). Vaikka natiivit XML-tietokannat ovat tarkoitettu XML-dokumenttien hallintaan, niin niiden perusajatuksena ei ole pelkkä dokumenttikeskeinen toimintamalli, vaan ne soveltuvat aivan yhtä hyvin datakeskeisen XML-tiedon hallintaan. Termi XML-dokumentti

tulee siitä, että yksi XML-tietokokonaisuus - koostuipa se yhdestä tai tuhannesta elementistä tai attribuutista - on aina dokumentti. Termin ei pidä siis antaa johtaa harhaan siinä mielessä, että XML:ssä oli kyse pelkästään dokumenttikeskeisestä tiedosta.

XML:n käyttöä tehostamaan on kehitetty useita oheisteknologioita, joista tietokantojen kannalta esille nousee kaksi teknologiaa: XML-kyselykielet ja XML-rajapinnat. SQL ei sovellu suoraan käytettäväksi XML:n kanssa, johtuen XML:n ja relaatiotietokantojen tietomallien poikkeavuudesta. XML:lle on kehitetty kyselykieliä, joista kaksi suosituinta ovat XPath ja XQuery (Bourret 2004). XML:lle kehitetyt kyselykielet mahdollistavat SQL:ää huomattavasti monipuolisempien kyselyjen toteuttamisen. Lisäksi XML-kyselykielillä tulokset voidaan saada tilanteeseen sopivassa muodossa, kuten oliona (DOM) tai XML-dokumenttina, kun SQL:llä tehtyjen kyselyjen tulokset tulevat aina tauluna (Feinberg 2004, s. 3). XML-kyselykielet ovat tällä hetkellä yksi tutkituimmista XML:n osa-alueista ja tulevat varmasti kehittymään tulevaisuudessa yhä tehokkaammiksi ja monipuolisemmiksi.

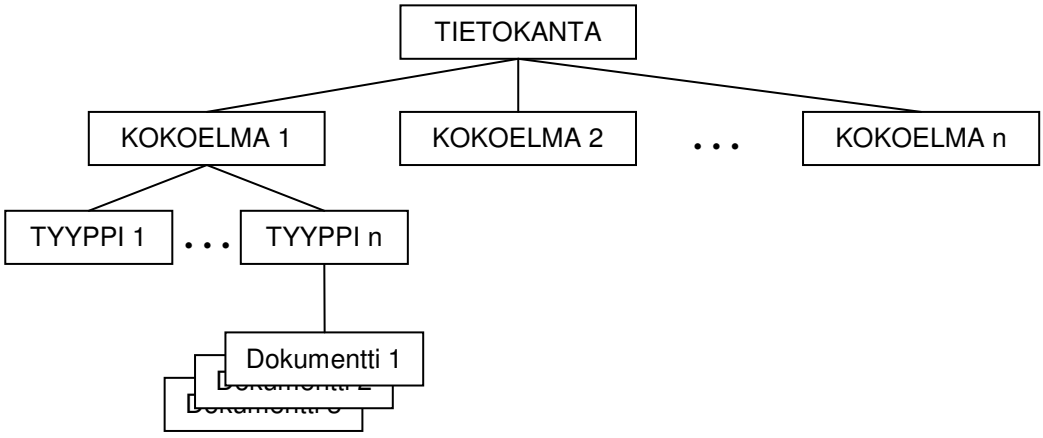
XML-rajapinnat, joista kaksi eniten käytettyä lienee DOM ja SAX, mahdollistavat XML-dokumenttien käyttämisen eri sovelluksissa (W3C 2005c, SAX 2005). DOM:n ja SAX:n toimintaperiaatteet poikkeavat toisistaan, mutta molemmat tarjoavat ohjelmointirajapinnat XML-dokumenttien käsittelyyn. DOM muodostaa koko XML-dokumentista puurakenteen muistiin, joten pääsy kaikkiin dokumentin elementteihin on nopeaa. Haittana DOM:n käytössä on resurssien hukkakäyttö (SAX 2005), sillä ison XML-dokumentin lataaminen tietokoneen muistiin vaatii paljon resursseja ja suurin osa resurssien käytöstä on turhaa jos esimerkiksi tuhannen elementin dokumentista tarvitsee noutaa vain yksi elementti. SAX on yksinkertainen ohjelmointirajapinta XML:lle ja sen toiminta poikkeaa DOM:sta siten, että se ei lataa koko XML-dokumenttia kerrallaan tietokoneen muistiin, vaan ainoastaan ne osat joita tarvitaan (SAX 2005). Tätä lähestymistapaa kutsutaan tapahtuma-pohjaiseksi (event-based) lähestymistavaksi. Vaikka SAX ei lataa XML-dokumenttia

kokonaisuudessaan muistiin, niin sen on läpikäytävä dokumentti löytääkseen halutun tiedon, ja tämä toimenpide vaatii suoritinresursseja. Tietokantakäytössä XML-rajapinnat vastaavat relaatiotietokantojen ohjelmointirajapintoja kuten ODBC ja JDBC.

4.2.2 Natiivin XML-tietokannan perusrakenne

Natiivin XML -tietokannan rakenne vaihtelee luonnollisesti eri toteutuksissa, mutta tässä kappaleessa pyritään hahmottelemaan natiivin XML -tietokannan perusrakenne.

Tyypillisesti natiivin XML -tietokannan alimman, niin sanotun varastointikerroksen (Storage Layer), muodostavat tallennuskone (Storage Engine) ja tietovarasto (Data Storage) (Fiebig et al. 2002, s. 294, Jagadish et al. 2002, s. 276). Fiebig et al. Mukaan varastointikerroksen tehtävänä on tarjota XML-dokumenteille, indekseille ja metatiedolle tehokas ja luotettava säilytysalusta (Fiebig et al. 2002, s. 293). Tietovarasto voi olla kiinteä osa varastointikerrosta, kuten Fiebig et al. esittävät tai se voi olla vaihdettava moduuli kuten esimerkiksi Meier sekä Jagadish et al. esittävät (Fiebig et al. 2002, s. 293-295, Meier 2003, s. 45-46 & Jagadish et al. 2002, s. 276). XML-dokumentit ovat yleensä tallennettuina kokoelmiin (Collections) kuten kuvassa viisi (Schöning 2003, s. 25, Staken 2005, s. 2, Meier 2003, s. 44), mutta myös vaihtoehtoisia rakenteita käytetään. Meier vertaa kokoelmia tiedostojärjestelmiin, joiden välillä ja sisällä vallitsee hierarkia (Meier 2003, s. 44). Schöning määrittelee lisäksi kokoelmien sisälle eri dokumenttityyppejä, jolloin myös muiden kuin XML-dokumenttien säilyttäminen kokoelmassa on mahdollista, kuitenkin siten, että yksi dokumentti voi olla vain yhden kokoelman ja yhden dokumenttityypin jäsen kerrallaan (Schöning 2003, s. 25). Kokoelmien määrää tai kapasiteettia ei ole ennalta määrätty. Kuvassa 5 on esitetty kokoelmista koostuvan tietokannan malli.



Kuva 5. Dokumenttien jakautuminen kokoelmiin

Tietovarastototeutukset ovat tietokantakohtaisia. Esimerkiksi Taminon tietovarasto on jaettu kahteen osaan: data- ja indeksiosioon, jotka sijaitsevat kiintolevyllä tai vastaavalla medialla (Schöning 2003, s. 36). Schöningin mukaan dokumenttityypit ovat jaettu ryppäisiin (Clusters) peräkkäisten hakujen nopeuttamiseksi. Riippuen dokumentin koosta, voidaan sitä tilan säästämiseksi yrittää pakata.

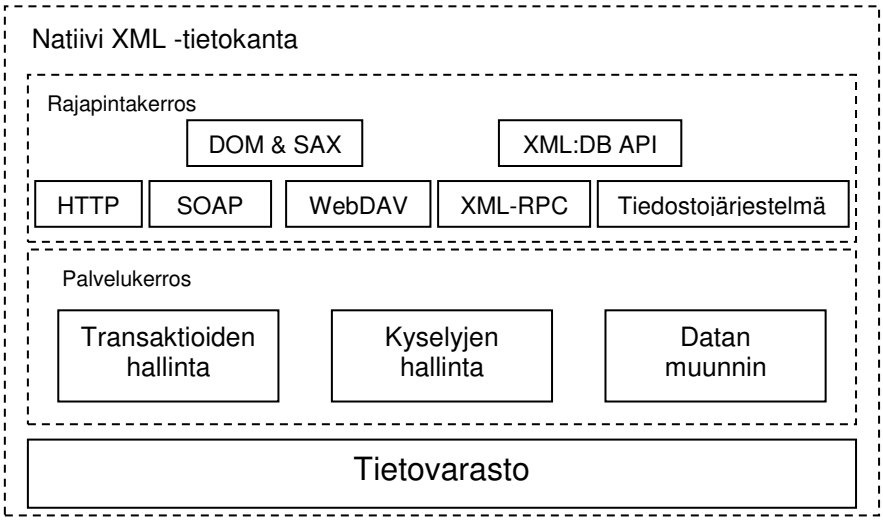
Fiebig et al. esittelevät oman natiivin XML -tietokantansa, Natixin, tietovarastoratkaisun erittäin yksityiskohtaisesti (Fiebig et al. 2002, s. 294-299). Natixin tietovarasto koostuu osioista eli tallennusmedioista, jotka koostuvat joukosta sivuja. Sivut ovat ryhmitelty loogisesti segmenteiksi, jotka edustavat eri dokumenttityyppejä, aivan kuten Schöningin esittelemässä Taminossa. Sivujen ja segmenttien käsittelystä huolehtivat puskurinhallinta (Buffer Manager) ja sivutulkit (Page Interpreters), joista ensimmäinen vastaa sivujen siirtämisestä levy- ja päämuistin välillä. Sivutulkit muuntavat raakadatan ylempien kerrosten ymmärtämään muotoon. Itse XML-dokumentit ovat tallennettuna puumaiseen rakenteeseen, joka on jaettu tietueisiin.

Varastointikerroksen yläpuolella sijaitsee tyypillisesti palvelukerros (Service Layer), kuten Fiebig et al. esittävät, tai Meierin esittämä tietokantamoottori (Database

Engine) (Fiebig et al. 2002, s. 293-294 & Meier 2003, s. 46). Poikkeavista nimistään huolimatta, vastaavat ne pitkälti toisiaan. Palvelukerroksen tärkeimmät tehtävät ovat transaktioiden hallinta, kyselyjen tulkitseminen ja suorittaminen sekä tietovirtojen muuntaminen tietovaraston ja ohjelmointirajapintojen välillä (Fiebig et al. 2002, s. 293-294). Kuvassa kuusi kyselyjen tulkitsemisesta ja suorittamisesta käytetään nimitystä kyselyjen hallinta. Lisäksi joissain toteutuksissa indeksin hallinta on eriytetty tietovarastosta ja on sijoitettuna palvelukerrokseen (Jagadish et al. 2002, s. 276).

Kuvassa kuusi ylin osio on rajapintakerros, joka sijaitsee Natiivin XML -tietokannan huipulla. Rajapintakerros tarjoaa rajapinnat eri sovelluksille (Fiebig et al. s. 293). Rajapinnat voidaan jakaa tässä tapauksessa kahteen kategoriaan: ohjelmointi- ja kommunikointirajapintoihin. Meierin esittelemä eXist tarjoaa sovelluksille neljä kommunikointirajapintaa: HTTP, SOAP, WebDAV ja XML-RPC sekä yhden ohjelmointirajapinnan, XML:DB Apin (Meier 2003, s. 46). Natix tarjoaa hieman rajatumman valikoiman kommunikointirajapintoja: HTTP:n, WebDAV:n ja tuen tiedostojärjestelmälle erillisellä ajurilla. Natixin ohjelmointirajapintoina toimivat DOM ja SAX, molemmat sekä C++:lla että Javalla (Fiebig et al. 2002, s. 293-294). Kuvasta kuusi tulee huomioida, että siinä esiintyvät rajapinnat ovat vain osa natiiveissa XML -tietokannoissa käytettävistä rajapinnoista.

Korkealla tasolla tarkasteltuna natiivien XML -tietokantojen rakenteet ovat melko samankaltaisia, joten kuvan kuusi malli esittää natiivien XML -tietokantojen rakennetta yleisellä tasolla.



Kuva 6. Natiivin XML -tietokannan yleinen malli

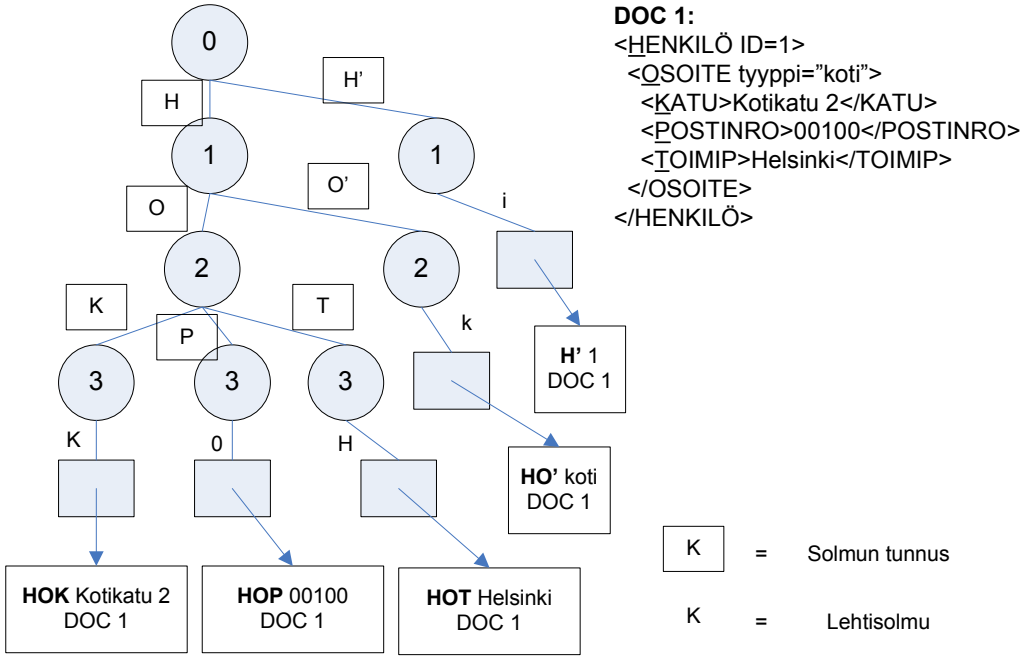
Eri tietokantatoteutusten eroavaisuudet tulevat esille tarkastelemalla erityisesti indeksien ja transaktioiden hallintaa sekä kykyä suorittaa eri kyselykielillä laadittuja kyselyjä. Seuraavissa kappaleissa tarkastellaan hieman lähemmin, mikä merkitys on indeksien ja transaktioiden hallinnalla sekä esitellään kaksi eniten käytettyä XML-kyselykieltä: XPath ja XQuery.

4.2.3 Indeksointimallit XML-dokumenteille

Indeksien tarkoituksena on tehostaa hakuja tietokannasta, aivan kuten kirjan sisällysluettelo nopeuttaa halutun kohdan löytämistä kirjasta. Kuten relaatiotietokannoissa, myös natiiveissa XML -tietokannoissa on suorituskyvyn kannalta oleellista, että käytetään tarkoituksenmukaista indeksointia. Relaatiotietokannoissa käytetyt indeksointimenetelmät, kuten B^+ ja ISAM, eivät toimi tehokkaasti sellaisenaan natiiveissa XML -tietokannoissa johtuen XML-dokumenttien rakenteisuudesta. XML-tiedon indeksointiin onkin kehitetty monia indeksointimenetelmiä. Zou et al. jakavat indeksointimenetelmät kolmeen kategoriaan: polkuindeksiin (Path Indexing), solmuindeksiin (Node Indexing) ja

sarjaindeksiin (Sequence Indexing) perustuviin menetelmiin (Zou et al. 2004, s. 39). Seuraavaksi tässä kappaleessa esitellään esimerkki jokaisesta edellä mainitusta kategoriasta.

Käytetyimmät XML-kyselykielet Xpath ja Xquery käyttävät nimettyjä polkuja (Label Paths) navigointiin XML-dokumenteissa (Chung et al. 2002, s. 121 & Chen et al. 2003, s. 134). Tästä syystä onkin luonnollista, että indeksit perustuvat polkuihin. Reittiä XML-dokumentin juuresta johonkin solmuun kutsutaan poluksi. Normaalit polkuindeksit pitävät kaikkia nimettyjä polkuja juuresta kohdesolmuun muistissaan ja ovat siten tehokkaita sellaisissa hauissa, jotka lähtevät liikkeelle dokumentin juurisolmusta (Chung et al. 2002, s. 122). Kohdesolmu löydetään vertaamalla kyselyn polkua indeksiin. Chen et al. mukaan tällaisessa indeksointimenetelmässä indeksin koko kasvaa hyvin suureksi ja osin tehottomaksi (Chen et al. 2003, s. 134). Lisäksi osittaisten hakujen (Partial Matching Queries) tekeminen täydellisten polkujen indeksiin on hankalaa, sillä osittaiset haut on muunnettava täydellisen indeksiin soveltuvaan muotoon (Chung et al. 2002, s. 122). Cooper et al. esittelevät Index Fabrig -polkuindeksimenetelmän, jossa indeksien kokoa on pystytty lyhentämään käyttämällä solmun nimen ensimmäistä kirjainta indeksinä. Menetelmän tehostamiseksi monimutkaisia kyselyjä on mahdollista määritellä ennalta (Cooper et al. 2001, s. 2). Index Fabrig -menetelmää on havainnollistettu kuvassa seitsemän. Kuvassa seitsemän suorakulmion sisällä olevat kirjaimet ovat samat kuin kyseessä olevan elementin nimen ensimmäinen kirjain. Esimerkiksi siirryttäessä juurisolmusta (solmu 0) sen ensimmäiseen lapsisolmuun eli dokumentin Henkilö-elementtiin, saadaan indeksiksi H-kirjain. Mikäli suorakulmion sisällä olevan kirjaimen perässä on heittomerkki (’), on tällöin kyseessä elementin attribuutti, kuten esimerkiksi O’, joka on Osoite-elementin attribuutin indeksi. Lehtisolmut ovat ilmaistu tyhjillä suorakulmioilla, ja niistä lähtee osoitin dokumentin tunnukseen (DOC 1 kuvassa seitsemän) sekä elementin tai attribuutin arvoon.



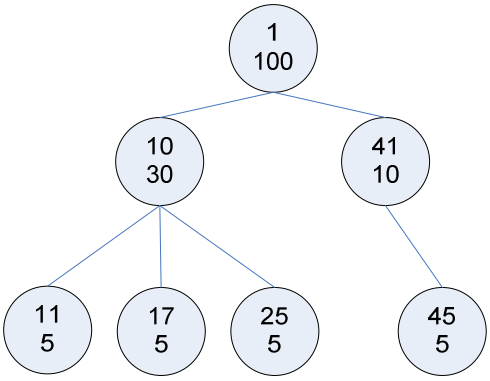
Kuva 7. Esimerkki Index Fabric -indeksointimenetelmästä

Solmuindeksiin perustuvissa indeksointimenetelmissä, kuten Lin ja Moonin kehittämässä XISS:ssä (XML Indexing and Storage System), indeksi perustuu numerointiskeemaan (Li & Moon 2001, s. 362-363 & Zou et al. 2004, s. 39). Numerointiskeemoissa XML-dokumentin solmuille annetaan kaksi kokonaislukutunnistetta, jotka voivat olla esi- ja jälkijärjestys tai kuten XISS:ssä järjestys ja koko. Kuvassa kahdeksan solmun ylempi lukuarvo on järjestysnumero ja alempi lukuarvo koko. Solmun järjestysnumero ja koko määräytyvät seuraavien kahden säännön mukaan (Li & Moon 2001, s. 362):

- Solmulle *y* ja sen vanhemmalle *x* pätevät: *järjestys(x)* < *järjestys(y)* ja *järjestys(y)* + *koko(y)* ≤ *järjestys(x)* + *koko(x)*
- Jos *x* on *y*:n edeltäjä esijärjestyksessä ja *x* ja *y* ovat sisaruksia, niin tällöin pätee: *järjestys(x)* + *koko(x)* < *järjestys(y)*

Edellä mainitut säännöt määräävät järjestysnumerolle sekä koolle vaihteluvälit, ja itse tunnusluvut voidaan määrätä käyttämällä kaavaa tai satunnaisesti, kuten kuvassa

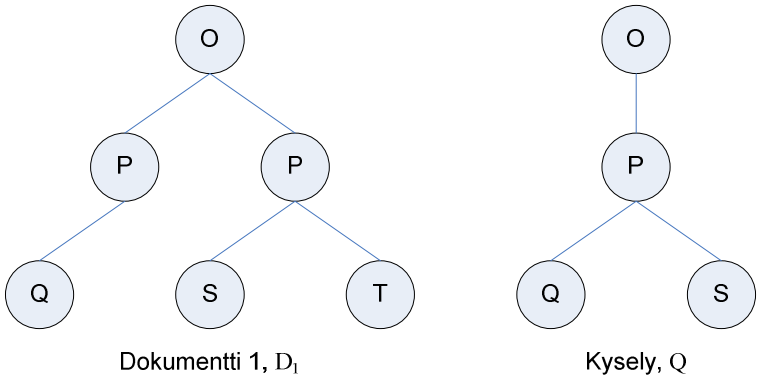
kahdeksan on tehty. Numerointiskeemaa käytetään solmujen yksilöllisten tunnusten määräämiseen. Hakuja varten elementtien ja attribuuttien nimistä luodaan B^+ -indeksipuut, joiden perusteella varsinainen haku tapahtuu (Li & Moon 2001, s.363-364). Lisäksi hakuja voidaan tehdä myös dokumentin rakenteen perusteella, joka on myös indeksoitu.



Kuva 8. Esimerkki XISS:n numerointiskeemasta (Li & Moon 2001)

Sarjaindeksiin (Sequence-based indexing) perustuvissa indeksointimenetelmissä, kuten ViST:ssä (Virtual Suffix Tree) ja PRIX:ssä (Prüfer sequences for Indexing XML), sekä dokumentit että kyselyt muunnetaan sarjoiksi, joita verrataan keskenään tuloksen löytämiseksi kyselyyn (Zou et al. 2004, s. 39). Sarjaindeksimenetelmistä ViST julkaistiin ensin Wang et al. toimesta ja se on hieman PRIXiä yksinkertaisempi (Wang et al. 2003). ViST:ssä XML-dokumentit ja niihin kohdistuvat kyselyt muunnetaan rakenne-enkoodatuiksi sarjoiksi (structure-encoded sequences) (Wang et al. 2003, s. 111-112). Wang et al. mukaan tällä pyritään välttämään turhien liitosoperaatioiden (Join operations) suorittaminen kyselyvaiheessa. Kuvassa yhdeksän on esimerkki ViST:stä. Rakenne-enkoodatut sarjat muodostetaan solmu-polku -pareista. Kuvassa yhdeksän dokumentista yksi muodostetun rakenne-enkoodatun sarjan ensimmäinen jäsen on solmu-polku -pari: solmu O, tyhjä. Koska solmu O on dokumentin juurisolmu, ei siihen johda polkua. Kuvan yhdeksän dokumentin 1 S-solmuun johtaa polku solmujen O ja P kautta, joten siihen viitataan

solmu-polku -parilla: S, PO. Kun sekä dokumentista että kyselystä on muodostettu rakenne-enkoodatut sarjat, kysely suoritetaan vertaamalla sarjoja. ViSTissä indeksointia on tehostettu tilastotieteen ja todennäköisyyslaskennan keinoin luomalla indeksoitavasta aineistosta tilastoja, joita ViST:n algoritmit käyttävät hyväksi (Wang et al. 2003, s. 116-119). Lisäksi sarjat ovat indeksoitu B^+-puuhun hakujen nopeuttamiseksi. Kuvassa yhdeksän alleviivatut solmu-polku -parit täsmäävät kyselyn Q solmu-polku -parien kanssa.



D_1 = (O, Ø) (P, O) (Q, PO) (P, O) (S, PO) (T, PO)
Q = (O, Ø) (P, O) (Q, PO) (S, PO)

Kuva 9. Esimerkki ViST-indeksointimenetelmästä

Raon ja Moonin kehittämässä PRIX-indeksointimenetelmässä käytetään rakenne-enkoodattujen sarjojen sijasta Prüferin sarjoja (Rao & Moon 2004, s. 291), jotka esitteli saksalainen matemaatikko Heinz Prüfer vuonna 1918. Prüferin menetelmällä puurakenne voidaan esittää sarjana. PRIX:ssä sekä XML-dokumentit että niihin kohdistuvat kyselyt muunnetaan Prüferin sarjoiksi, minkä jälkeen sarjoille suoritetaan suodatus- (filtering) ja jalostusvaiheita (refinement) (Rao & Moon 2004, s. 292). Myös PRIX indeksoi sarjansa B^+-puuhun, jonka avulla haut suoritetaan.

Edellä esitettyjen indeksointimenetelmien lisäksi on olemassa suuri joukko muita menetelmiä, joiden lisäksi uusia menetelmiä kehitetään jatkuvasti lisää. Indeksien

valinnalla on tietokannan suorituskyvyn kannalta erittäin merkittävä rooli, joten tiedonhallintajärjestelmän käyttöönottovaiheessa indeksien valinta kannattaa suorittaa huolellisesti. Tietokannan kehitysprosessin helpottamiseksi Runapongsa et al. ovat kehittäneet työkalun nimeltä XIST (An XML Index Selection Tool), jonka avulla XML-tietokannan indeksien valinta helpottuu (Runapongsa et al. 2004). XIST laatii annettujen tietojen, kuten skeemojen, kyselymäärän ja datan tilastotietojen perusteella suosituksen käytettävistä indekseistä. XIST:n suosittelemat indeksit ovat tyypiltään polkuindeksejä. Runanpongsa et al. mukaan hyödyntämällä skeemojen rakennekuvauksia, XIST:n suosittelemien kandidaatti-indeksien määrää voidaan olennaisesti vähentää (Runapongsa et al. 2004, s. 232).

4.2.4 Transaktioiden hallinta natiiveissa XML -tietokannoissa

Kun tiedonhallintajärjestelmällä on useampia käyttäjiä, on mahdollista, että käyttäjät tekevät toisistaan tietämättä yhtaikaisia päivityksiä samoihin tietueisiin. Tällöin pahimmassa tapauksessa tiedonhallintajärjestelmän tieto ei ole enää paikkansa pitävää. Korvaamatonta tietoa voi jopa tuhoutua. Relaatiotietokannoissa yhtaikaisten käyttäjien hallintaan on ollut olemassa ratkaisu jo 1980-luvulta lähtien (Härder & Reuter 1983). Ratkaisua kutsutaan transaktionhallinnaksi (transaction management). Transaktion määritelmään liittyy olennaisesti ACID-ehto, joka tarkoittaa, että jokaisen transaktion on toteutettava seuraavat ehdot (Lewis et al. 2002, s. 25-26):

- *Atomisuus* (Atomicity): Jokainen transaktio suoritetaan joko kokonaan tai ei ollenkaan.
- *Oikeellisuus* (Consistency): Jokainen transaktio säilyttää tietokannan oikeellisuuden.
- *Eristys* (Isolation): Transaktiot eivät saa vaikuttaa toisiinsa.
- *Kestävyys* (Durability): Onnistuneet transaktiot tallennetaan pysyvästi tietokantaan.

Transaktioiden hallinta on olennainen osa myös natiiveja XML -tietokantoja. Relaatiotietokannoissa käytetyt menetelmät eivät kuitenkaan sovellu tehokkaasti käytettäviksi natiiveissa XML -tietokannoissa, sillä ne ovat usein liian rajoittavia (Dekeyser & Hidders 2004). Dekeyserin ja Hiddersin mukaan tämä tarkoittaa, että relaatiotietokantoihin tarkoitetut transaktionhallintamenetelmät lukitsevat usein jopa koko dokumentin päivityksen ajaksi vaikka päivitys kohdistuisi vain yhteen dokumentin elementtiin (Dekeyset & Hidders 2004, s. 93-94). Tämä johtuu relaatiotietokantojen ja natiivien XML -tietokantojen eriävistä tavoista käsitellä ja mallintaa XML-dokumenteissa esiintyviä vanhempi-lapsi -suhteita. Natiiveihin XML -tietokantoihin onkin kehitetty erilaisia transaktionhallintamenetelmiä, joista osaan tutustutaan tässä kappaleessa.

Transaktioiden hallinnan kannalta olennainen tekijä on yhtäaikaisten transaktioiden hallinta. Yhtäaikaisten transaktioiden hallinta on yleisesti toteutettu käyttämällä erilaisia lukitusmekanismeja. Lukitsemisella tarkoitetaan transaktion käsittelyn alla olevan tiedon varaamista vain kyseisen transaktion käyttöön. Useimmat lukitusmenetelmät perustuvat Eswaran et al. esittelemään kaksivaiheiseen lukitusprotokollaan (Two-Phase Locking Protocol) (Eswaran et al. 1976, Lewis et al. 2002, s. 454). Kaksivaiheisessa lukitusprotokollassa on käytössä kahdenlaisia lukkoja: luku- ja kirjoituslukkoja. Luku- ja kirjoituslukoille on olemassa seuraavat ominaisuudet (Lewis et al. 2002, s. 454-455):

- Yhdelle tietoelementille voidaan myöntää eri transaktioiden pyynnöstä yksi tai useita yhtäaikaisia lukulukkoja. Tällöin kyseistä tietoelementtiä voidaan ainoastaan lukea.
- Jos tietoelementillä on jonkin yksittäisen transaktion pyynnöstä päällä kirjoituslukko, niin tällöin yksikään muu transaktio ei voi saada kyseiselle elementille luku- tai kirjoituslukkoa.
- Jos lukulukon omaavalle elementille haetaan kirjoituslukkoa jonkin toisen transaktion, T, toimesta, tällöin T:n on odotettava kunnes elementin lukulukko

aukeaa. Vastaavasti jos jollain elementillä on voimassa kirjoituslukko, on muiden transaktioiden odotettava lukon aukeamista saadakseen luku- tai kirjoituslukon kyseiselle elementille.

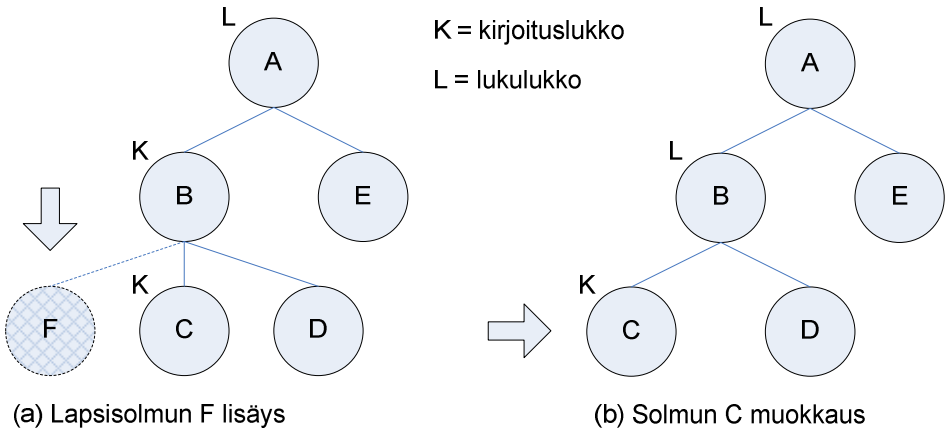
- Transaktio avaa kaikki sille myönnetyt lukot, kun se on saatu suoritettua loppuun. Suorassa kaksivaiheisessa lukitusprotokollassa transaktio lukitsee kaikki tarvitsemansa elementit ja suorittaa tämän jälkeen kaikki tarvittavat luku- ja kirjoitusoperaatiot, minkä jälkeen se vapauttaa kaikki lukot. Epäsuorassa kaksivaiheisessa lukitusprotokollassa transaktio varaan niin ikään kaikki tarvitsemansa lukot, minkä jälkeen se siirtyy suorittamaan luku- ja kirjoitusoperaatioita kuitenkin siten, että se voi vapauttaa lukon missä tahansa vaiheessa.

Kaksivaiheinen lukitusprotokolla ei vielä itsessään riitä toimivaksi lukitusmekanismiksi natiivissa XML -tietokannassa, vaan se on vain osa lukitusmekanismia. Seuraavaksi esitellään kolme lukitusmenetelmää XML-dokumenteille.

Yksinkertaisin lukitusmenetelmä on lukitus dokumenttitasolla (Helmer et al. 2004, s. 59). Dokumenttitason lukituksessa kyseessä oleva dokumentti lukitaan kokonaisuudessaan yhden transaktion käyttöön. Menetelmä on yksinkertainen, mutta varaa koko dokumentin yhdelle transaktiolle kerrallaan, ja on siksi epäkäytännöllinen järjestelmissä, joissa on suuri määrä käyttäjiä.

Toinen ja huomattavasti kehittyneempi tapa toteuttaa transaktionhallinta on käyttää solmutason lukitusta (Node level lock). Solmutason lukituksessa solmu ja sen edeltäjäsolmut sekä joissain tapauksissa myös lapsisolmut lukitaan luku- tai kirjoitusoperaation ajaksi (Haustein & Härder 2004, s. 233-234 & Helmer et al. 2004, s. 60). Solmutason lukitusmekanismeissa on käytössä erityyppisiä lukkoja. Haustein & Härder esittävät peräti seitsemän eri lukitustyyppiä (Haustein & Härder 2004, s.

234). Kuvassa 10 on kaksi esimerkkiä solmutason lukituksesta. Esimerkkien yksinkertaistamiseksi on käytetty vain kahdentyyppisiä lukituksia. Kohdassa 10a suoritetaan lapsisolmun lisäys solmulle B. B-solmu lukitaan kirjoituslukolla ja sitä edeltävä solmu A lukulukolla. Myös B:n lapsisolmu C lukitaan kirjoituslukolla, koska sen vasen sisarosoitin (Sibling pointer) muuttuu (Helmer et al. 2004, s. 60). Solmulle F ei tarvita lukitusta, koska sitä ei ole olemassa ennen lisäystä. Kohdassa 10b solmun C arvoa muokataan, jolloin se lukitaan kirjoituslukolla. Solmun C edeltäjille A ja B annetaan lukulukot. Haustein & Härder esittävät kolmea kirjoituslukkotyyppiä, IX (Intention Exclusive) ja CX (Child Exclusive) ja X (Exclusive) (Haustein & Härder 2004, 234). IX ilmaisee, että jossain alipuun solmussa suoritetaan kirjoitusoperaatio. CX tarkoittaa, että solmun lapsisolmulle suoritetaan kirjoitusoperaatio ja se on siten lukittu X-lukolla. Täten esimerkin 10b solmu A saisi IX-lukon, B CX-lukon ja C X-lukon. Koska solmu A olisi IX-lukittu, ei sen lapsisolmulle E voitaisi myöntää toisen transaktion toimesta esimerkiksi X-lukkoa ennen kuin A:n IX-lukitus olisi purettu.

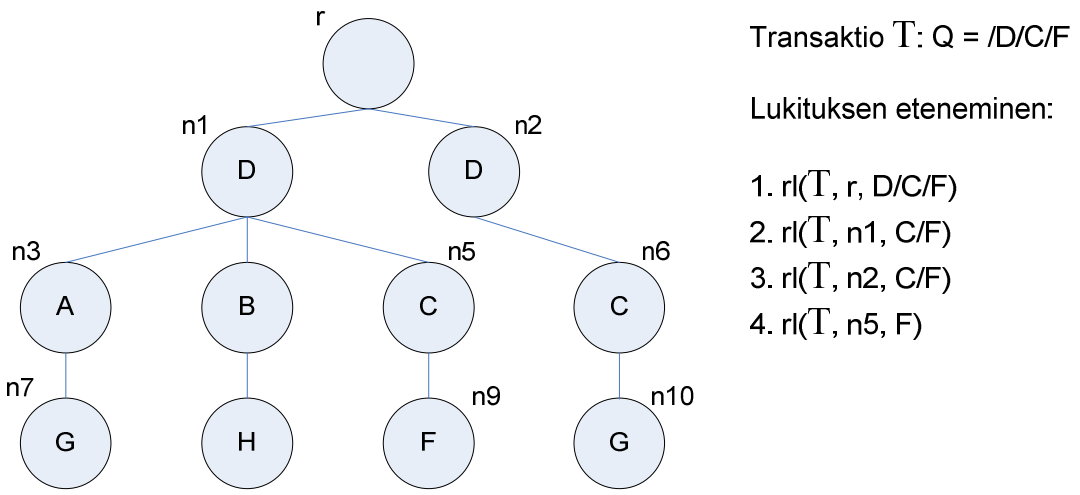


Kuva 10. Esimerkit solmutason lukituksesta

Viimeisenä lukitusmekanismina esitellään polkulukitus (Path lock). Useat polkulukitusmekanismit (Choi & Kanai 2003, Dekeyser & Hidders 2004 & Jea et al. 2002) perustuvat XPath-kyselykieleen. XPath-kyselykielen tietomalli mallintaa XML-dokumentin puuksi, jonka solmuina ovat XML-dokumenttien elementit,

attribuutit ja kommentit sekä niiden arvot, jotka ovat tekstisolmuja (Lewis et al. 2002, s. 582). Lisäksi jokaisella puulla on niin sanottu juurisolmu, joka ei ole sama kuin XML-dokumentin juurielementti. Polkulukituksen ideana on lukita vain tarvittavat polut joko luku- tai kirjoituslukoin, jolloin yksittäinen transaktio rajoittaa mahdollisimman vähän muiden transaktioiden toimintaa. Seuraavaksi esitellään kaksi esimerkkiä polkulukituksen toteuttamisesta.

Dekeyserin ja Hiddersin etenevä polkulukitus -skeemassa (Path lock propagation scheme) luku- ja kirjoituslukot esitetään kolmen alkion listoina rl(*t, n, p*) ja wl(*t, n, p*), missä *t* on transaktion tunnus, *n* on solmun tunnus ja *p* on polkuilmaus (path expression) (Dekeyser & Hidders 2004, s. 96-97). Polkulukitus lähtee liikkeelle niin sanotusta alkulukosta (initial lock), joka määräytyy annetusta Xpath-kyselystä. Kun alkulukko on määritelty, voidaan siitä johtaa seuraava lukko, joka toimii lähtökohtana seuraavalle lukolle. Tätä jatketaan kunnes tarvittavat polut ovat lukittu. Kuvan 11 esimerkki havainnollistaa polkulukituksen etenemistä. Esimerkissä ainoastaan solmu n9 saa kirjoituslukon, joka estää muiden transaktioiden operaatiot kyseiseen solmuun. Muut polulla sijaitsevat solmut saavat lukulukot, jotka estävät kirjoitusoperaatiot kyseisiin solmuihin, mutta sallivat lukuoperaatiot mahdollistaen muiden transaktioiden osittaisen toiminnan. Transaktion lopuksi lukot avataan.



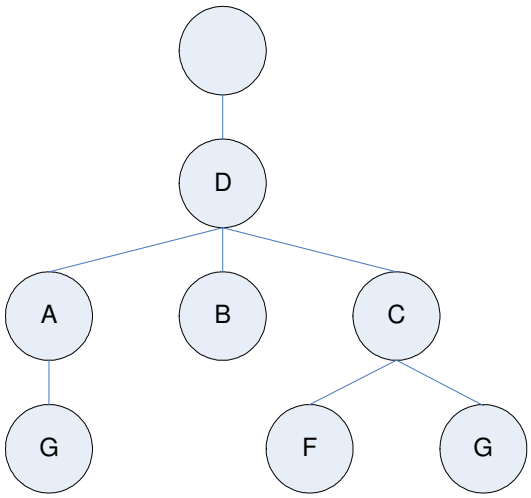
Kuva 11. Polkulukituksen eteneminen

Jea et al. esittelemässä XLP-lukitusmenetelmässä (XPath Locking Protocol) on käytössä viisi erilaista lukitusta: luku-, kirjoitus, lisäys-, poisto- ja läpäisylukitus (pass lock) (Jea et al. 2002, s. 2). Lukituksista viimeinen, läpäisylukitus, lukitsee solmun siten, ettei sitä voida poistaa, mutta sen arvoa voidaan muuttaa. Eli läpäisylukittu solmu voi saada lisäksi luku-, kirjoitus tai lisäyslukituksen. XLP poikkeaa Dekeyserin ja Hiddersin etenevästä polkulukituksesta juuri läpäisylukituksen erityisen toiminnallisuuden vuoksi. XLP:ssä läpäisylukitus vapautetaan heti, kun se on mahdollista. Tällöin yhtäaikaisia lukituksia on voimassa lukumääräisesti vähemmän kuin Dekeyserin ja Hiddersin menetelmässä. Kuvan 12 esimerkki havainnollistaa XLP:n toimintaa kahden yhtäaikaisen transaktion tapauksessa. Esimerkissä merkintä P(x) tarkoittaa läpäisylukitusta, R(x) lukulukitusta ja W(x) kirjoituslukitusta solmulle x.

Transaktio T_1: Q = /D/C/F

Transaktio T_2: Q = /D/C/F(Write)

Askel	T_1	T_2
1	Lukitse P(D)	
2		Lukitse P(D)
3	Lukitse P(C)	
4		Lukitse P(C)
5	Vapauta P(D)	
6		Vapauta P(D)
7	Lukitse R(F)	
8	Lue F:n arvo	
9	Vapauta R(F)	
10		Lukitse W(F)
12		Vapauta P(D)
13		Kirjoita F:n arvo
14		Vapauta W(F)



Kuva 12. Esimerkki XLP-menetelmästä

Transaktionhallinnan osa-alueista lukitusmekanismi on näkyvin käyttäjälle. Riippuen lukitusmekanismin toiminnasta, saattaa käyttäjä joutua turhaan odottamaan muiden käyttäjien transaktioiden suorittamista.

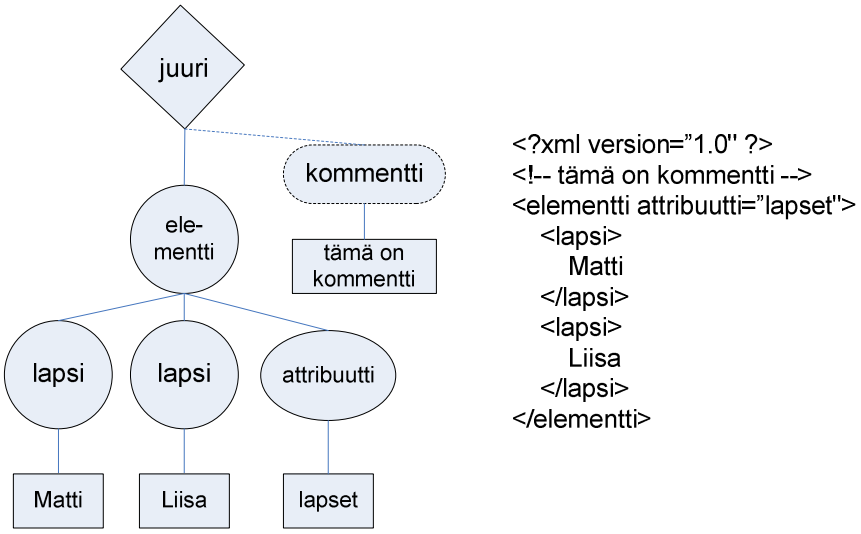
4.2.5 XML-kyselykielet

Edellisissä kappaleissa on esitelty natiivin XML -tietokannan perusrakenne, erilaisia indeksointimalleja sekä transaktiohallintamenetelmiä, jotka ovat tärkeitä osia XML-tiedonhallintajärjestelmässä. Mikään tiedonhallintajärjestelmä ei kuitenkaan ole käyttökelpoinen, mikäli sen toiminnallisuus rajoittuu pelkästään tiedon tallentamiseen ja säilyttämiseen. Tietoa pitää pystyä myös etsimään ja noutamaan tehokkaasti. Relaatiotietokannoista tuttu SQL on juuri kehitetty tiedon etsimiseen ja hakemiseen. Vaikka SQL on erittäin tehokas kyselykieli, se on kuitenkin suunniteltu relaatiomallia silmälläpitäen. SQL:n toiminta perustuu relaatioalgebraan sekä predikaattilogiikkaan (Lewis et al. 2002), mikä tekee siitä matalatasoisen kyselykielen. XML-tietomalli

poikkeaa relaatiomallista pääasiassa sen puurakenteen vuoksi. Puurakenne aiheuttaa sen, että SQL:n käyttö XML-dokumentteihin vaatii suuren määrän liitosoperaatioiden (Joins) käyttämistä, mikä on sekä tietohallintajärjestelmän että sen käyttäjän kannalta ongelmallista. Liitosoperaatiot vaativat paljon resursseja, joten kysely, jossa on esimerkiksi 10 liitosta, vaatii paljon resursseja suorittimelta. Lisäksi kysely on monimutkainen kirjoittaa ja vaikeaa saada toimimaan oikein. Tämän takia XML-dokumentteja varten on kehitetty useita korkean tason kyselykieliä. Tässä kappaleessa tutustutaan kahteen käytetyimpään XML-kyselykieleen, XPathiin ja XQueryyn.

XPathista julkaistiin ensimmäinen suositus W3C:n (W3C Recommendation) toimesta vuonna 1999. Aikainen julkaisu on edesauttanut XPathin suosion kasvua. Toinen suosittu kyselykieli, XQuery, on vielä luonnosvaiheessa (W3C Working Draft).

XPath on kevyt, polkuilmauksiin perustuva kyselykieli, kuten edellisessä kappaleessa XLP:n yhteydessä esitettiin, XPathin käyttämässä tietomallissa XML-dokumentit mallinnetaan XML-puina. XML-puiden solmut vastaavat XML-dokumenttien elementtejä, attribuutteja ja kommentteja. XML-dokumenttien arvot esitetään lehtisolmuina. Lisäksi jokaisella XML-puulla on erityinen juurisolmunsa, jonka alle ovat koottuna kaikki XML-dokumentin osat, myös juurielementti. Esimerkki XPathin tietomallista on kuvan 13 XML-puu. (Lewis et al. 2002, s. 581-583 & Schwentick 2004)



Kuva 13. XPathin tietomallin mukainen XML-puu

XPath käyttää XML-puissa navigointiin laajennettuja polkuilmauksia. Termi laajennettu polkuilmaus voidaan jakaa kahteen osaan: polkuun ja laajennettuun ilmaukseen. Polulla tarkoitetaan tässä yhteydessä reittiä solmusta A solmuun B. Polkuja on kahdenlaisia, absoluuttisia ja suhteellisia. Absoluuttinen polku on reitti juurisolmusta kohdesolmuun, kun taas suhteellinen polku on reitti kahden solmun välillä. Polku voidaan esittää muodossa */solmu/lapsisolmu*. Tällöin puhutaan polkuilmauksesta. Laajennetussa polkuilmauksessa on käytössä ehto-operaattorit, joilla kyselyjä voidaan monipuolistaa. Ehto-operaattorit mahdollistavat kuitenkin vain suhteellisen yksinkertaisten kyselyjen tekemisen, kuten haun kokonaisella tai osittaisella hakusanalla tai haun ilmentymien lukumäärän perusteella. XPathin etuina voidaan pitää kyselyjen syntaksin yksinkertaista rakennetta sekä käytettävyyttä ja keveyttä yksinkertaisiinkin sovelluksiin. (W3C 1999, Lewis et al. 2002, s. 581-585)

Vaikka XPath on riittävä kyselykieli useimpiin tarkoituksiin, siinä on ilmaisuvoiman puutteen lisäksi yksi merkittävä puute: XPathilla voi tehdä kyselyjä ainoastaan XML-dokumentteihin. Tämä rajoittaa XPathin käyttöä esimerkiksi relaatiotietokannoissa, sillä relaatiomalli piilottaa XML-dokumenttien rakenteen XPathilta. W3C:n toimesta

perustettiinkin työryhmä vuonna 1999 kehittämään uutta kyselykieltä nimenomaan XML-tietolähteitä varten. (Chamberlin 2002, s. 597)

XQuery on XPathia huomattavasti laajempi kokonaisuus, ja sen juuret ovat enemmän SQL:ssä sekä oliomaailmassa. XQuery perustuu Quilt-nimiseen kieleen, jossa on vaikutteita OQL oliokyselykielestä, aiemmista XML-kyselykielistä Lorelista, XQL:stä ja XML-QL:stä sekä relaatiotietokantojen kyselykielestä SQL:stä (Chamberlin 2002, s. 598 & Lewis et al. 2002, s. 599). XQuery käyttää lisäksi XPathin syntaksia polkuilmauksissa.

XQueryn tietomalli perustuu käsitteellisiin sekvensseihin. Yksi sekvenssi on järjestetty kokoelma, jolla voi olla nolla tai useampia alkioita. Alkio voi olla joko solmu tai atominen arvo (Atomic Value), mutta ei toinen sekvenssi. Yhdessä sekvenssissä voi olla sekä solmuja että atomisia arvoja. Atomisilla arvoilla tarkoitetaan XML-skeeman tietotyyppien, kuten kokonaisluku, merkkijono tai päivämäärä, ilmentymiä. Solmuja on seitsemän eri tyyppiä: elementti, attribuutti, teksti, dokumentti, kommentti, nimiavaruus tai prosessointiohje. Solmulla voi olla lapsisolmuja, jolloin ne muodostavat hierarkioita. Joillakin solmuilla, kuten elementti ja attribuutti voi olla nimi ja arvo. Solmun arvolla tarkoitetaan nollasta tai useammasta atomisesta arvosta koostuvaa sekvenssiä. Lisäksi jokaisella solmulla on yksilöllinen tunniste. Solmujen ja niiden hierarkioiden keskinäinen järjestys muodostaa dokumenttijärjestyksen, jossa vallitsee vastaava järjestys kuin varsinaisessa XML-dokumentissa. (Chamberlin 2002, s. 598-599)

XPathista poiketen XQuery on funktionaalinen kieli. Tämä tarkoittaa, että XQueryssa on käytössä joukko funktioita, aivan kuten SQL:ssä, jotka palauttavat arvon. XQueryssa lisäksi on käytössä muuttujat funktioiden käytön mahdollistamiseksi. Mahdollisuus iteraatioden ja ehtolauseiden käyttöön tekee XQuerystä tehokkaan hakukielen. Iteroinnin ansiosta XQuerylla voidaan ohjelmoida suhteellisen helposti

skriptejä, joiden avulla monimutkaisenkin kyselyraportin laatiminen nopeutuu huomattavasti. Luonnollisesti XQueryssa on käytettävissä Boolen algebra ja yleisimmät aritmeettiset operaattorit. Polkuilmaukset esitetään käyttämällä XPathin syntaksia. (Chamberlin 2002, s. 600-613 & Lewis et al. 2002, s. 616)

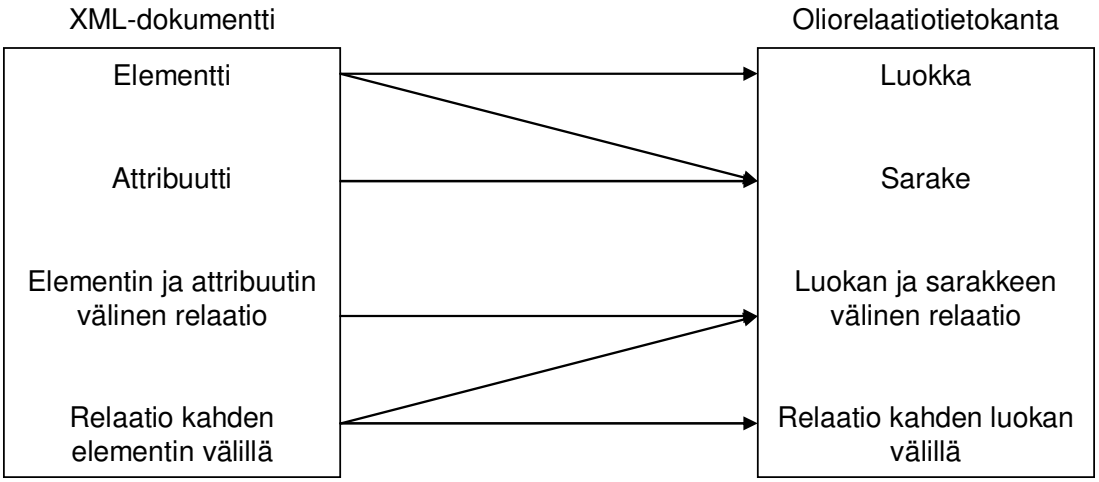
XQueryyn on sisällytetty mittava määrä ominaisuuksia, joilla siitä on saatu erittäin kattava ja tehokas kyselykieli. Mittavien ominaisuuksien ja osittain vielä keskeneräisen kehitystyön takia XQuery on kuitenkin monimutkainen ja raskas kieli peruskäyttöä ajatellen. Kevyeen käyttöön ajateltuna XPath on selkeä valinta, mutta jouduttaessa työskentelemään suurien tietomäärien ja eri tietolähteiden parissa, on XQuery järkevä valinta. Sekä XQueryn että XPathin kehitystyö etenee W3C:n työryhmissä. W3C:n kunnianhimoisena tavoitteena on kehittää XQuerysta standardikyselykieli niin XML:lle, kuin kaikille verkkodokumenteille (W3C 2005d). XQuerysta on myös kehitetty niin sanottuja alikieliä, jotka omaavat vain osan sen ominaisuuksista. Eräs tällainen XQueryn kevennetty versio on Hidders et al. kehittämä LiXQuery (Light XQuery) (Hidders et al. 2004).

4.3 XML-dokumenttien hallinta oliorelaatiotietokannassa

Relaatiotietokantojen puutteet ja rajoitukset sekä sopimattomuus ei-perinteisille sovelluksille ovat olleet syynä oliorelaatiotietokantojen kehittämiselle. Oliorelaatiotietokannoissa tietomalli koostuu edelleen relaatioista, jotka voidaan esittää listoina. Listojen alkiot voivat kuitenkin olla pelkkien atomisten arvojen sijasta myös olioita. Kyselykieli SQL 1999 on tärkeä osa oliorelaatiotietokantoja, sillä siinä on tuki oliorelaatiotietomallille. (Lewis et al. 2002, s. 479, 486-487)

XML-dokumenttien tallentamiseksi oliorelaatiotietokantoihin on esitetty useita ratkaisuja, joista osassa XML-tietomalli kytketään oliorelaatiotietomalliin (Partede et al. 2004, s. 704). Partede et al. esittelemä malli perustuu SQL 1999 kokoelmatietotyyppien käyttöön (Partede et al. 2004). Seuraavaksi tässä kappaleessa tutustutaan Han et al. esittelemään ratkaisuun sekä Partede et al. malliin (Han et al. 2003 & Partede et al. 2004). Han et al. mallissa XML-dokumentit kytketään oliorelaatiomalliin käyttämällä XML-skeemoja, kun Partede et al. ratkaisu perustuu kokoelmatyyppien käyttöön.

Han et al. esittelevät artikkelissaan mallin, joka perustuu W3C:n XML-skeeman appinfo-elementin sekä erityisen XML-mallin ja oliorelaatiomallin välisen kytköksen käyttöön (Han et al. 2003). W3C:n XML-skeeman määrityksessä on esitetty annotation-elementti sekä sen alaelementit appinfo ja documentation. Documentation-elementin avulla voidaan skeemaan lisätä selventäviä kommentteja ja appinfo-elementillä voidaan määritellä ohjeistuksia ja lisätietoja skeemaa käyttäville sovelluksille (Walkama & Laakkonen 2004, s. 287-288). Han et al. mukaan appinfo-elementissä voidaan esittää tarvittavat kytkökset XML-dokumentin ja oliorelaatiomallin välillä (Han et al. 2003). Kuva 14 havainnollistaa Han et al. esittelemiä kytkösvaihtoehtoja.



Kuva 14. Mahdolliset kytkennät XML- ja oliorelaatiomallin välillä

Han et al. käyttävät menetelmässään XML-skeemoja kytkösten määrittelemiseksi (Hat et al. 2003, s. 120). Kytkösten määrittelyssä käytetään kolmea elementtiä: luokka (Class), sarake (Column) ja relaatio (Relationship). Luokkaelementti määrittelee luokan ja sen nimen. Sarake-elementillä määritellään luokan jäsenet. Luokan jäsenillä eli sarakkeilla on kaksi attribuuttia: nimi ja tyyppi. Luokka- ja sarake-elementtien avulla määritellään XML-dokumentin elementtien ja attribuuttien kytkemiset oliorelaatiomalliin. Relaatioelementillä, jolla on neljä attribuuttia, määritellään elementtien välisten relaatioiden kytkökset oliorelaatiomalliin. Relaatioelementin attribuuteilla määritellään relaation osapuolet, tyyppi sekä järjestys-ominaisuus. Relaation tyyppi voi olla yksi-yhteen tai yksi-moneen. Relaation järjestyksellä tarkoitetaan tässä tapauksessa XML-dokumentin jäsenten (elementit, attribuutit) keskinäisen järjestyksen säilyttämistä. Käyttämällä edellä esitettyä kolmea elementtiä XML-skeeman appinfo-elementtien sisällä, voidaan XML-mallin ja oliorelaatiomallin väliset kytkökset määritellä, kuten kuvan 15 esimerkissä on tehty. Kuvassa 15 kytkentöjen määritykset ovat esitetty lihavoidulla ja kursivoidulla kirjasimella.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:annotation>
    <xs:appinfo>
      <class name="henkilo">
        <column name="henkilo.puhelin" type="list(ref(puhelin))"/>
        <column name="henkilo.id" type="integer"/>
        <column name="henkilo.ammatti" type="varchar(100)"/>
      </class>
      <class name="puhelin">
        <column name="puhelin.henkilo" type="ref(henkilo)"/>
        <column name="puhelin.numero" type="varchar(100)"/>
        <column name="puhelin.tyyppi" type="varchar(100)"/>
      </class>
      <relationship parent="henkilo.puhelin"
      child="puhelin.henkilo" cardinality="oneToMany"
      isOrdered="yes"/>
    </xs:appinfo>
  </xs:annotation>
  <xs:element name="henkilo">
    <xs:annotation>
      <xs:appinfo>
        <class name="henkilo"/>
      </xs:appinfo>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="id" type="xs:integer">
          <xs:annotation>
            <xs:appinfo>
              <column name="henkilo.id"/>
            </xs:appinfo>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Kuva 15. Esimerkki XML-mallin ja oliorelaatiomallin välisestä kytkennästä

XML-dokumentit tallennetaan oliorelaatiotietokannan luokkiin. Han et al. esittävät käytettäväksi vain kolmea luokkaa liitosten minimoimiseksi (Han et al. 2003, s. 122). Elementtiluokkaan tallennetaan elementit ja niiden kytkösmäärittelyt. Attribuuttiluokkaan tallennetaan vastaavasti attribuutit määrittelyineen. Viimeinen luokka, eli relaatioluokka, vastaa relaatioiden ja niiden kytkösten säilyttämisestä. Kun oliorelaatiotietokannan luokat ovat määritelty, voidaan XML-dokumentit tallentaa

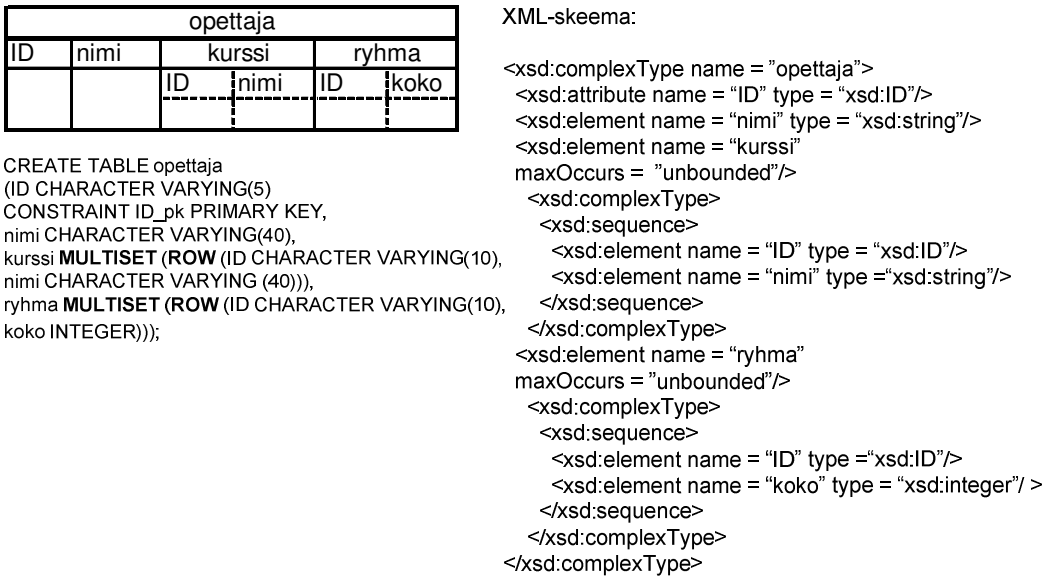
tietokantaan käyttämällä Han et al. esittämää algoritmia, joka tulkitsee skeemassa määritellyt kytkökset (Han et al. 2003, s. 123-124).

Partede et al. esittelemässä menetelmässä XML-dokumenttien konseptuaalinen taso (Conseptual level) mallinnetaan käyttäen semanttisen verkon mallia (Semantic Network Model), ja varsinainen kytkentä oliorelaatiotietokantaan tapahtuu käyttäen eri sääntöjä sekä SQL4:n kokoelmia (Partede et al. 2004, s. 705).

Semanttinen verkko on suunnattu graafi, joka koostuu joukosta solmuja sekä niitä yhdistävistä suunnatuista poluista (Feng et al. 2002, s. 395). Semanttisessa verkossa on neljä peruskomponenttia: solmujoukko, polkujoukko, nimiöjoukko ja rajoitejoukko. Solmu, joka voi olla perus- tai kompleksityyppiä, kuvaa reaalimaailman objektia. Solmujen väliset suhteet kuvataan käyttämällä polkuja, joiden merkitys ja tyyppi ilmaistaan käyttämällä nimiötä. Perussolmulla tarkoitetaan lehtisolmua, josta ei lähde eteenpäin vieviä polkuja. Kompleksisesta solmusta taas lähtee yksi tai useampia nimettyjä polkuja eteenpäin toisiin solmuihin.

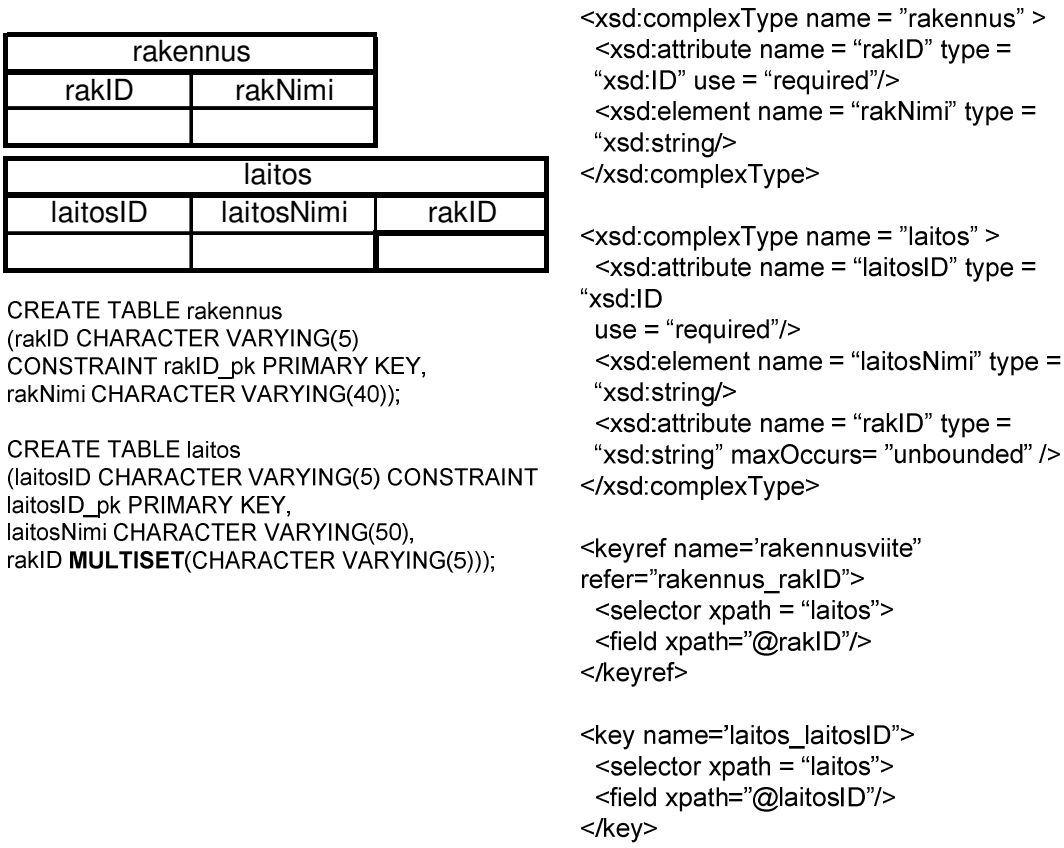
Semanttinen verkko voidaan muuntaa XML-skeemaksi Feng et al. esittelemillä menetelmillä (Feng et al. 2002, s. 404-416). Partede et al. kehittämän menetelmän ideana on laajentaa semanttisen verkon muunnoksia niin, että XML-dokumenttien konseptuaalisen tason semantiikkaa ei hukata tallennettaessa dokumentteja oliotietokannan tauluihin (Partede et al. 2004, s. 705-710). Menetelmän keskeinen kohta on XML-dokumenttien erityyppisten aggregaatioiden ja assosiaatioiden siirtäminen oliotietokannan tauluihin kokoelmatietotyyppiä hyväksikäyttämällä, joita SQL4:ssä ovat Array ja Multiset. Partede et al. esittävät toteutukset kahden eri aggregaatiotyypin ja niin ikään kahden eri assosiaatiotyypin siirtämiseksi oliorelaatiotietokannan tauluihin (Partede et al. 2004, s. 705-709).

Aggregaatiolla tarkoitetaan koosteisen objektin muodostamista komponenteistaan. Aggregaatioita on kahden tyyppisiä. Ensimmäisessä tyypissä komponentit ovat itsenäisiä ja suojaamattomia, eikä niiden olemassaolo ole riippuvainen objektin olemassaolosta. Toisessa aggregaatiotyypissä, jota kutsutaan myös kompositioksi (Composition), on tilanne päinvastainen; komponentit ovat suojattuja ja riippuvaisia objektista. Partede et al. esittävät kummallekin aggregaatiotyypille kaksivaiheisen muunnoksen (Partede et al. 2004, s. 705-707). Muunnoksen ensimmäisessä vaiheessa suoritetaan XML-skeeman koostaminen eli siirros semanttisen verkon mallista XML-skeemaan. Komponentin ja objektin mallinnus riippuu aggregaatiotyypistä. Kompositiotyypin tapauksessa komponentti on elementtityyppinen ja objekti on kompleksityyppinen. Toinen vaihtoehto on mallintaa sekä komponentti että objekti XML-skeeman kompleksisena tyyppinä (complexType). Toisessa vaiheessa XML-skeema mallinnetaan oliorelaatiotietokannan tauluksi käyttäen apuna kokoelmia. Kompleksityyppiä oleva komponentti mallinnetaan tietokannan tauluun käyttäen joko array tai multiset -tietotyyppiä. Komposition tapauksessa riittää yksi taulu, kun taas toisessa aggregatiotyypissä tarvitaan kaksi taulua, koska komponentit eivät ole suojattuja, vaan ovat myös muiden objektien käytettävissä. Kuva 16 havainnollistaa kompositiotyyppistä aggregaatiota tietokantataulussa. Kuvassa 16 on skeeman lisäksi esitetty tietokantataulu ja sen luomiseen tarvittava lause. Lauseessa on lihavoituna komennot, joilla multiset-kokoelma luodaan. Tässä tapauksessa kurssi- ja ryhma-elementit, jotka molemmat ovat määritelty skeemassa kompleksityyppiseksi, mallinnetaan tietokantataulussa kokoelmina. (Partede et al. 2004, s. 705-707)



Kuva 16. Aggregaatio tietokannassa ja skeemassa

Aggregaation lisäksi toisistaan riippumattomat objektit voivat olla suhteessa toisiinsa aivan kuten relaatiomallissa taulujen välillä voi olla relaatioita. Tällaista objektien välistä relaatiota kutsutaan assosiaatioksi. Partede et al. esittävät toteutukset kahden eri assosiaatiotyypin, yksi-moneen ja moni-moneen, siirtämiseksi oliorelaatiotietokantaan (Partede et al. 2004, s. 707-709). Toteutukset ovat myös kaksivaiheisia aivan kuten aggregaation tapauksessa. XML-skeemassa assosiaatioita esiintyy esimerkiksi sellaisissa tapauksissa, kun käytetään W3C XML-skeeman viiteavain-ominaisuutta (KeyRef). Kuvassa 17 on esimerkki yksi-moneen -assosiaation siirtämisestä XML-skeemasta tietokantatauluun. Moni-moneen -tyyppisen assosiaation siirtäminen tietokantatauluun tapahtuu kuten yksi-moneen -assosiaation sekä lisäämällä multiset-kenttään kaksi viitesaraketta, joilla kytkökset määritellään.



rakennus	
rakID	rakNimi

laitos		
laitosID	laitosNimi	rakID

```
CREATE TABLE rakennus
(rakID CHARACTER VARYING(5)
CONSTRAINT rakID_pk PRIMARY KEY,
rakNimi CHARACTER VARYING(40));

CREATE TABLE laitos
(laitosID CHARACTER VARYING(5) CONSTRAINT
laitosID_pk PRIMARY KEY,
laitosNimi CHARACTER VARYING(50),
rakID MULTISET(CHARACTER VARYING(5)));
```

```
<xsd:complexType name = ”rakennus” >
  <xsd:attribute name = “rakID” type =
  “xsd:ID” use = “required”/>
  <xsd:element name = “rakNimi” type =
  “xsd:string/>
</xsd:complexType>

<xsd:complexType name = ”laitos” >
  <xsd:attribute name = “laitosID” type =
“xsd:ID
  use = “required”/>
  <xsd:element name = “laitosNimi” type =
  “xsd:string/>
  <xsd:attribute name = “rakID” type =
  “xsd:string” maxOccurs= ”unbounded” />
</xsd:complexType>

<keyref name=’rakennusviite”
refer=”rakennus_rakID”>
  <selector xpath = “laitos”>
  <field xpath=”@rakID”/>
</keyref>

<key name=’laitos_laitosID”>
  <selector xpath = “laitos”>
  <field xpath=”@laitosID”/>
</key>
```

Kuva 17. Yksi-moneen -assosiaatio tietokantataulussa ja XML-skeemassa

Tässä kappaleessa on esitelty kaksi tapaa, joilla XML-dokumenttien tallentaminen oliorelaatiotietokantaan on mahdollista. Kumpikin menetelmistä käyttää XML-skeemoja hyväkseen, joten menetelmät sopivat vain sellaisten XML-dokumenttien tallentamiseen, joilla on olemassa XML-skeema.

5. XML-TIEDONHALLINTARATKAISUJEN VERTAILU

Tässä kappaleessa vertaillaan keskenään neljää XML-tiedonhallintaratkaisua. Vertailun tulosten avulla valitaan sopiva XML-tiedonhallintaratkaisu Java-sovelluspalvelimella toimivalle testausjärjestelmälle, joka käyttää XML:ää tulosten formaattina.

Vertailu jakautuu kahteen osa-alueeseen: suorituskykyyn ja teknisiin ominaisuuksiin. Suorituskykyvertailun tarkoituksena on saada selville toteutusten väliset suorituskykyerot käyttämällä erityistä XML-tiedonhallintajärjestelmille tarkoitettua suorituskykytestiä. Siinä missä suorituskykytestaus tuottaa tuloksina numeerisia arvoja, ovat tiedonhallintaratkaisujen teknisten ominaisuuksien arvioinnin tulokset sanallisia ja ei-diskreettejä.

Vertailu rakentuu siten, että kappaleessa 5.1 esitellään testattavat XML-tiedonhallintaratkaisut sekä testiympäristö. Suorituskykymittauksen menetelmät ja kriteerit esitellään kappaleessa 5.2. Teknisten ominaisuuksien arviointikriteereihin sekä osa-alueisiin tutustutaan kappaleessa 5.3. Lopuksi suorituskykytestin tulokset esitellään kappaleessa 5.4 ja teknisten ominaisuuksien arviointia on kappaleessa 5.5.

5.1 Testiympäristö ja testattavat tiedonhallintaratkaisut

Testilaitteistona käytetään 1,60 MHz suorittimella, 256 Mt keskusmuistilla ja 40 Gt ATA-133 kiintolevyllä varustettua PC-konetta. Käyttöjärjestelmänä toimii Windows 2000 Advanced Server Service Pack 4:lla varustettuna. Koska vertailun tarkoituksena on hakea sopiva tiedonhallintajärjestelmä Javalla toteutetulle järjestelmälle, testataan relaatiotietokantapohjaiset tiedonhallintajärjestelmät käyttäen JDBC-rajapintaa ja

natiivit XML tiedonhallintajärjestelmät käyttäen XML:DB-rajapintaa. Testeissä käytettävä Java-alusta on JDK 1.4.2_08. Testissä käytettävät relaatiotietokantapohjaiset XML-tiedonhallintamenetelmät toimivat MySQL 4.1 -relaatiotietokannassa. Testattavat natiivit XML-tiedonhallintajärjestelmät toimivat Tomcat 4.1.31 -sovelluspalvelimella. Kaikki testeissä käytettävät ohjelmistot toimivat paikallisesti samalla tietokoneella. Suorituskykytestit suoritetaan käyttäen XMach-1:n referenssitoteutusta (Xmach-1 2005). Kullekin testattavalle tiedonhallintajärjestelmälle toteutetaan oma liitännäismoduuli, joka mahdollistaa tiedonhallintajärjestelmän ja XMach-1:n referenssitoteutuksen välisen kommunikaation. Testattavien tiedonhallintajärjestelmien asetuksia hienosäädetään pelkästään puskurimuistin osalta, jonka kooksi asetetaan 64 Mt.

Vertailun kohteena on neljä XML-tiedonhallintaratkaisua, joista kaksi perustuu relaatiotietokantaan ja kaksi natiiviin XML-tietokantaan. Relaatiotietokantaan perustuvat ratkaisut ovat Binary Approach ja Edge Approach (Florescu & Kossman 1999b). Testattavat järjestelmät ovat valittu sillä perusteella, että alustavien tutkimusten perusteella ne vaikuttavat sopivilta verkkokäyttöön. Natiiveista XML-tietokannoista testiin ovat valittu Apache Software Foudationin Xindice (Apache Software Foundation 2005a) sekä avoimen kehittäjäyhteisön eXist (eXist 2005). Sekä eXist että Xindice ovat Javalla toteutettu avoimen lähdekoodin ohjelmisto. Valintaperusteet eXistille ja Xindicelle ovat avoin lähdekoodi, Java-pohjaisuus ja mahdollisuus ajaa ohjelmistoa Java-sovelluspalvelimella. Mittaukset ja arvioinnit suoritetaan Xindicen versiolle 1.1,4b sekä eXistin 1.02b versiolla, jotka molemmat ovat tämän diplomityön kirjoituksen hetkellä viimeisimmät julkaistut versiot.

Ohjelmointirajapintoina käytetään Binary Approachille ja Edge Approachille JDBC:tä ja eXistille sekä Xindicelle XML:DB:tä. MySQL:n JDBC-ajurin versionumero on 3.1.11. Existin ja Xindicen XMLDB-ajureina käytetään julkaisujen mukana tulleita ajureita.

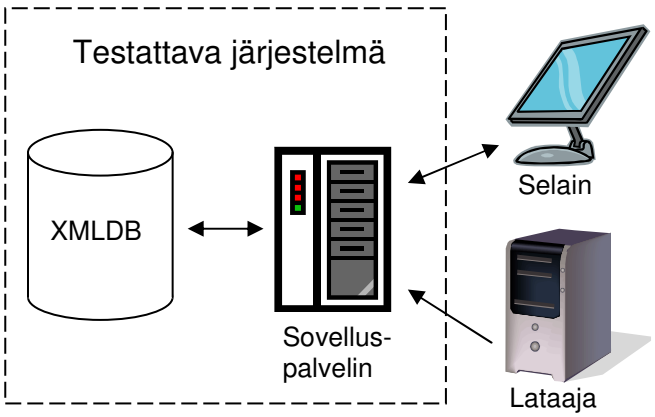
5.2 Suorituskykymittaus

XML-tietokantojen suorituskyvyn mittauksiin on kehitetty useita eri mittausmenetelmiä (Bressan et al. 2003, s. 482-497 & Lu et al. 2005, s. 155-156). Olemassa olevia mittausmenetelmiä ovat ainakin The Michigan Benchmark (Runapongsa et al. 2003), The XOO7 Benchmark (Bressan et al. 2002), Xbench (Yeo et al. 2004), XMach-1 (Böhme & Rahm 2001) ja XMark (Schmidt et al. 2001a).

Edellä mainituista mittausmenetelmistä Böhmen & Rahmin esittelemä XMach-1 soveltuu ominaisuuksiltaan erityisen hyvin tässä suorituskykytestissä käytettäväksi (Böhme & Rahm 2001). XMach-1 on mittausmenetelmä, joka on suunniteltu erityisesti XML-dataa käyttävien B2B- ja verkkosovelluksien suorituskyvyn mittaamiseen (Bressan et al. 2003, s. 488). XMach-1:llä voidaan asettaa testiajoon useita yhtäaikaisia virtuaaliasiakkaita, jolloin saavutetaan verkkosovelluksille ominainen usean yhtäaikaisen käyttäjän aiheuttama kuormitus (Böhme & Rahm 2001, s. 9). Menetelmän testidata on määritelty DTD:llä, ja se koostuu pienikokoisista XML-dokumenteista, joiden koko on keskimäärin 10 kt. Testidatan rakenne on ominaisuuksiltaan hyvin samankaltainen testausjärjestelmän testiraporttien kanssa. XMach-1 koostuu yhdestätoista erilaisesta kyselystä, jotka ovat jaettu kahteen ryhmään sen mukaan, onko kyseessä hakukysely vai datan manipulointiin liittyvä kysely. XMach-1:lle on sen kehittäjien toimesta julkaistu referenssitoteutus ohjelmista, joilla testiajot voidaan suorittaa (Böhme & Rahm 2005). Edellä mainittujen ominaisuuksien perusteella, suorituskykymittaukset suoritetaan XMach-1:llä. Seuraavaksi esitellään XMach-1:n toimintamalli ja rakenne.

XMach-1:n toteutus on sovelluspalvelimella ajettava verkkosovellus. Tällä ratkaisulla pyritään mallintamaan XML-tiedonhallintajärjestelmän tyypillistä käyttöä (Böhme & Rahm 2001, s. 3). XMach-1:n arkkitehtuuri koostuu kolmesta komponentista: selaimesta, lataajasta ja testattavasta järjestelmästä, kuten kuvassa 18 on esitetty.

XMach-1 toimii palvelinohjelmiston päällä, joka on yhteydessä testattavaan järjestelmään, lataajaan sekä asiakassovellukseen. XMach-1:n toteutusta voidaan ajaa joko erillisellä sovelluspalvelimella tai samalla sovelluspalvelimilla testattavan järjestelmän kanssa, kuten kuvan 18 tapauksessa. Asiakassovelluksena toimii HTTP:tä käyttävä Java-sovellus, joka generoi ja lähettää kyselyt testattavalle tiedonhallintajärjestelmälle. Lataajan tehtävänä on generoida testidataa ja ladata sitä testattavaan järjestelmään.



Kuva 18. XMach-1:n arkkitehtuuri.

Testidata koostuu sekä data- että dokumenttikeskeisistä XML-dokumenteista (Böhme & Rahm 2001, s. 4). XMach-1:ssä datakeskeisenä XML-dokumenttina toimii hakemistodokumentti, joka sisältää muun muassa dokumentin id:n, lisäys- ja päivitysajat sekä URL:n osiin jaettuna. Kuvassa 19 on esimerkki hakemistodokumentista, jossa on määritelty dokumentin testi1.xml URL-osoite (www.testiurl.fi/testit/testi1.xml), doc_id, lataaja sekä lisäysaika. Hakemistodokumentista käytetään nimitystä hakemisto. Hakemistodokumenttien tarkoituksena on tarjota hakemisto dokumenttikeskeisille XML-dokumenteille. Tällaista ratkaisua käytetään muun muassa useissa sähköisissä tuotekatalogeissa. Toinen dokumenttityyppi, eli dokumenttikeskeiset XML-dokumentit ovat enemmistön asemassa XMach-1:n testidatassa. Näistä dokumenteista käytetään nimitystä hallittavat dokumentit (Managed documents). Hakemisto- ja hallittavat

dokumentit ovat suhteessa toisiinsa doc_id-elementin kautta. Molempien dokumenttityyppien DTD:t ovat listattuina liitteessä yksi. Koska on epätodennäköistä, että tuotantokäytössä olevassa XML-tietokannassa olisi vain kaksi erilaista DTD:tä, käytetään hallittavien dokumenttien elementtien nimissä lisäksi kokonaislukuja, jolloin erilaisten DTD-määritysten lukumäärää voidaan kasvattaa. Esimerkiksi viidennessä DTD:ssä käytetään elementtejä kuten ”chapter5” ja ”section5”. Hallittavat dokumentit voivat olla myös ilman DTD:tä. XMach-1:n kyselyt kohdistuvat molempiin dokumenttityyppeihin.

```
<directory>
 <host name=“fi">
  <host name=“testiurl">
   <host name="www">
    <path name=“testit">
     <path name=“testi1.xml”>
      <doc_info doc_id="2“ loader=“robot1"
      insert_time=“20050921093001"/>
     </path>
    </path>
   </host>
  </host>
 </host>
</directory>
```

Kuva 19. Esimerkki XMach-1:n hakemistodokumentista.

XMach-1:n yhdestätoista kyselystä kahdeksan on hakukyselyjä ja loput kolme ovat datan manipulointiin liittyviä kyselyjä (Böhme & Rahm 2001, s. 8). Hakukyselyt ovat esitetty taulukossa seitsemän.

Taulukko 7. XMach-1:den hakukyselyt.

Kysely	Kuvaus	Selitys
Q1	Hae tietyn URL:n omaava dokumentti.	Palauttaa kokonaisen dokumentin.
Q2	Hae tietyn fraasin omaava doc_id ja URL.	Tekstin noutava kysely. Fraasi valitaan listasta.
Q3	Aloita ensimmäisestä elementistä, jonka nimi sisältää ”chapter”-sanan ja rekursiivisesti seuraa polkua, jonka elementtien nimet alkavat ”section”-sanalla. Palauta viimeinen ”section”-elementti.	Kysely simuloi navigointia dokumenttipuussa.
Q4	Palauta lista pääelementeistä, jotka ovat lapsielementtejä elementeille, joiden nimi alkaa”section”-sanalla.	Kysely simuloi sisällysluettelon luomista.
Q5	Hae dokumenttien nimet kaikista niistä dokumenteista, jotka ovat tietyn URL-fragmentin alla.	Selaa hakemistorakennetta. Operaatio rakenteelliselle järjestelemättömälle datalle.
Q6	Hae tietyn sisällön omaavan ”author”-elementin doc_id ja sen vanhemman id.	Tämä kysely testaa indeksoinnin tehokkuutta.
Q7	Hae niiden elementtien doc_id:t, joihin viitataan vähintään neljässä muussa dokumentissa.	Haetaan tärkeitä dokumentteja. Haun suorittamiseksi tarvitaan joukko- ja laskuoperaatioita.
Q8	Hae niiden 100 viimeksi päivitetyn dokumentin doc_id:t, joissa esiintyy ”author”-attribuutti.	Kyselyssä tarvitaan lasku-, lajittelu- ja liitosoperaatioita sekä pääsyä metadataan.

Datan manipulointiin liittyviä kyselyjä XMach-1:ssä on kolme. Datan manipuloinnilla tarkoitetaan tässä yhteydessä sitä, että manipulointikyselyn suorittamisen jälkeen tietokannan data on muuttunut verrattuna kyselyä edeltävään tilanteeseen. Manipulointikyselyt ovat esitetty taulukossa kahdeksan.

Taulukko 8. XMach-1:n manipulointikyselyt.

Kysely	Kuvaus	Selitys
M1	Lisää annetulla URL:lla varustettu dokumentti tietokantaan.	Generoidaan annetulla URL:lla varustettu dokumentti ja ladataan se tietokantaan.
M2	Poista annetulla doc_id:llä varustettu dokumentti.	Poistetaan dokumentti tietokannasta.
M3	Päivitetään annetulla doc_id:llä varustetun dokumentin URL ja update_time.	Päivitetään hakemiston merkintä.

Vertailukelpoisten ja todellista suorituskykyä kuvaavien mittaustulosten saavuttamiseksi eri kyselyjen (Q1-Q8 ja M1-M3) lukumäärien painotuksille on annettu suositukset (Böhme & Rahm 2001, s. 8-9). Böhme & Rahm suosittelevat kyselyjen lukumäärien painotusten mukailevan Transaction Processing Performance Councilin TPC-W-testin määrityksiä (Böhme & Rahm 2001, s. 9 & Smith 2001). Taulukossa yhdeksän on esitetty suositeltavat painotukset eri kyselyille. Taulukosta voidaan huomata, että kyselyjen painotuksia voidaan hienosäätää halutun mittausjärjestelyn saavuttamiseksi.

XMach-1:n mittausprosessissa simuloidaan monen yhtäaikaisen käyttäjän toimintaa ajamalla useita asiakkaita yhtäaikaisesti, kuitenkin siten, että asiakas tekee vain yhden kyselyn kerrallaan (Böhme & Rahm 2001, s. 8). Tällä saavutetaan todellisen kaltainen kuormitus järjestelmälle. Koska mittauksissa simuloidaan useaa yhtäaikaista käyttäjää, ei mittaustulosten yksikkönä voida käyttää sekuntia tai muuta ajan yksikköä, kuten yhtä käyttäjää simuloivissa mittausmenetelmissä käytetään. XMach-1:n mittaustulosten perusyksikkönä käytetään suoritettuja XML-kyselyjä/ sekunti (Xqps). Koska testidatassa voi olla DTD-määriteltyjä ja ilman määritystä olevia XML-dokumentteja, on käytössä kaksi mittayksikköä: Xqps (XML queries per second) ja $Xqps_{sl}$ (Xqps schema less). Molempien mittayksiköiden tapauksessa mittaustulos määräytyy suoritettujen Q1-kyselyjen mukaan. Tämä on Böhmen ja Rahmin mukaan riittävä menetelmä, sillä kyselyjen painotukset takaavat sen, että kokonaiskuormitus on kaikille järjestelmille sama (Böhme & Rahm 2001, s. 9).

Taulukko 9. Kyselyjen painotukset XMach-1:ssä.

Kysely	osuus
Q1	≤ 30 %
Q2	≥ 20 %
Q3, Q4, Q5, Q6	≥ 10 % kukin
Q7, Q8	≥ 4 % kukin
M1	≥ 0,75 %
M2	≥ 0,25 %
M3	≥ 1 %
Yhteensä	100 %

5.3 Teknisten ominaisuuksien arviointikriteerit

Siinä missä suorituskykymittauksiin on olemassa useita valmiita ratkaisuja, teknisten ominaisuuksien arviointiin ei löydy kuin suosituksia. Yksi suositus teknisten ominaisuuksien arvioinnissa käytettävistä laatukriteereistä on julkaistu Schmidt et al. toimesta (Schmidt et al. 2001b). Schmidt et al. jakavat kriteerit kahteen osa-alueeseen: infrastruktuuriin ja kokonaiskustannuksiin (TCO, Total Cost of Ownership) (Schmidt et al. 2001, s. 31).

Infrastruktuuriin liittyviä kriteereitä on kolme: rajapinnat, tulosformaatit ja kyselykielet. Rajapinnoilla tarkoitetaan sekä kommunikointi- että ohjelmointirajapintoja. Käytännössä raja ohjelmointi- ja kommunikointirajapinnan välillä on häilyvä, joten yksittäisen rajapinnan voidaan katsoa kuuluvaksi kumpaankin kategoriaan. Esimerkkeinä rajapintoina ovat CORBA, DCOM, XML-RPC, XML:DB, JDBC ja ODBC. Tulosformaatilla on merkitystä sekä suorituskyvyn että ohjelmoijan kannalta. Hakujen tuloksien muuntaminen formaatista toiseen vaatii järjestelmältä suorituskykyä sekä mahdollisten lisämoduulien toteuttamista. Tietokantahakujen tulokset tulevat relaatiotietokannoissa aina tauluna, joka muunnetaan ohjelmointirajapinnan käskyillä esimerkiksi tekstimuotoon. Natiiveissa

XML-tietokannoissa hakujen tulokset voidaan esittää DOM:n tai SAX:n avulla, tai ne voidaan sarjallistaa suoraan esimerkiksi tiedostoksi. DOM ja SAX ovat osoittaneet hyödyllisyytensä XML-dokumenttien käsittelyssä, ja ovat tällä hetkellä kaksi vallitsevaa XML-teknologiaa. Kyselykielellä on merkitystä erityisesti monimutkaisia hakuja tehdessä. Relaatiotietokannoissa kyselykielenä käytetään SQL:ää, ja natiiveissa XML-tietokannoissa käytössä on usein joko XPath tai XQuery.

Sitä mukaa kun teknisiä ominaisuuksia lisätään, infrastruktuuri monimutkaistuu ja sen hallinnan kokonaiskustannukset lisääntyvät. Kokonaiskustannusten eli TCO:n arviointiin Schmidt et al. esittävät seitsemän kriteeriä (Schmidt et al. 2001b, s. 31):

- *Aloituskustannukset* (Installation Effort): Kuinka paljon resursseja ja toimenpiteitä tuotteen käyttöönotto vaatii?
- *Eheys* (Consistency Support): Tarjoaako tuote toiminnallisuuden uusien dokumenttien validointiin skeemoja tai muita rajoituksia vasten?
- *Infrastruktuurikustannukset* (Interaction Paradigm): Omaako tuote kattavat ominaisuudet dokumenttien käsittelyyn, vai onko tuote vain yksi osa tiedonhallintajärjestelmää?
- *Kattavuus* (Generality Support): Voidaanko tuotteella hallita sekä skeemallisia että skeemattomia XML-dokumentteja? Entä tuki muille kuin XML-dokumenteille?
- *Koulutuskustannukset* (Training): Tarvitseeko tuotteen käyttöä kouluttaa tekniselle henkilökunnalle, vai sulautuuko tuote saumattomasti jo olemassa olevaan infrastruktuuriin?
- *Tallennuskustannukset* (Preparation Effort): Mitä toimenpiteitä uuden dokumentin tallentaminen vaatii?
- *Päivitys* (Updates): Tarjoaako tuote toiminnallisuudet dokumenttien päivittämiseksi yksityiskohtaisella (fine-grained) tasolla, vai pitääkö koko dokumentti korvata uudella päivitystarpeen ilmaantuessa?

Edellä esitellyistä kriteereistä voidaan huomata, että teknisten ominaisuuksien esittäminen absoluuttisina numeerisina arvoina suorituskyvyn tapaan on lähes mahdotonta. Myös objektiivisuuden saavuttaminen teknisiä ominaisuuksia vertailtaessa on hankalaa. Tästä syystä teknisten ominaisuuksien vertailuun ei kiinnitetä yhtä suurta huomiota kuin suorituskyvyn mittaamiseen. Teknisten ominaisuuksien vertailutulokset ovat enemmänkin suuntaa-antavia toteamuksia, joiden perusteella lukija voi tehdä omat johtopäätöksensä.

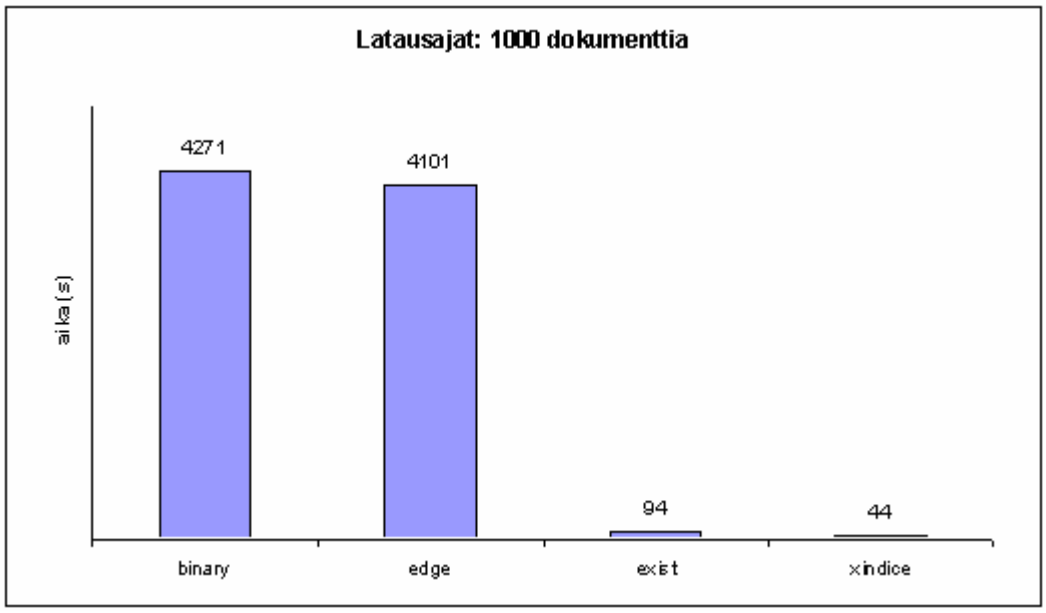
5.4 Suorituskykytestin asettelu ja testitulokset

Tässä kappaleessa esitellään ja analysoidaan testidatan lataukseen käytetyt ajat, valitut indeksoinnit ja testiasetukset sekä varsinaiset Xmach-1 -testin tulokset.

5.4.1 Dokumenttien lataus tietokantoihin

Suorituskykytestejä varten jokaiseen testattavaan tiedonhallintajärjestelmiin ladattiin tuhat (1000) XML-dokumenttia käyttäen Xmach-1:n referenssitoteutuksen lataajaa. Latausta varten jokaiselle testattavalle tiedonhallintajärjestelmälle toteutettiin yhteysmoduuli. Dokumenttien DTD-määrityksiä ei ladattu tietokantoihin. Latauksen aikana ei myöskään suoritettu tietojen indeksointia. Kuvan 20 kuvaajasta nähdään, että relaatiotietokantapohjaisten ratkaisujen latausajat ovat pisimmillään jopa lähes satakertaiset verrattuna natiiveihin XML -tietokantoihin. Tämä merkittävä ero latausajoissa johtuu pääasiassa siitä, että Binary Approach ja Edge Approach menetelmien latausmoduuleihin liittyy paljon DOM-objektien käsittelyä sekä runsas määrä SQL-kyselyitä. Testikoneen pienen keskusmuistin, 256 Mt, takia paljon muistia vaativat DOM-objektit jouduttiin siirtämään väliaikaisesti levyvälimuistiin, mikä aiheutti paljon I/O -operaatioita. I/O -operaatioiden suuren määrän takia järjestelmän suorituskyky heikentyi merkittävästi. Edge Approach -menetelmän

hieman lyhyempi latausaika on selitettävissä relaatiotietokantataulujen pienemmällä lukumäärällä. Sekä Binary Approach että Edge Approach -menetelmissä käytettyjen relaatiotietokantataulujen rakenteet on esitetty liitteessä kaksi. Natiiveista XML -tietokannoista Xindicen latausaika oli vain vajaa puolet eXistin latausajasta. Xindice ei luo oletusarvoisesti mitään indeksejä siihen ladatuista dokumenteista, kun taas eXist luo ladatuista dokumenteista rakenne- ja teksti-indeksit automaattisesti. Indeksien luomista voidaan pitää yhtenä syynä eXistin pidempään latausaikaan.



Kuva 20. Latausajat eri XML-tiedonhallintajärjestelmille (pienempi arvo parempi)

5.4.2 Tietokantojen indeksointi

XML-dokumenttien latausten jälkeen kukin tietokanta kahdennettiin, jotta testit voitaisiin ajaa sekä indeksoimattomalle että indeksoidulle datalle tietokantojen ollessa muuten identtiset.

Binary Approachille indeksit valittiin Florescun ja Kossmanin suosituksen perusteella sekä Xmach-1:n kyselyjen pohjalta (Florescu & Kossman 1999b, s. 29). Florescu ja Kossman suosittelevat käytettäviksi indekseiksi lähde- ja kohdesaraketta, jotka

vastaavat tässä toteutuksessa doc_id ja value -sarakkeita. Florescun ja Kossmanin suositusten lisäksi indekseiksi valittiin solmujen keskinäisen järjestyksen ilmaiseva node_order -sarake sekä solmun vanhemman ilmaisevat parent_element ja parent -sarakkeet, koska hakukyselyissä on paljon viittauksia kyseisiin sarakkeisiin. Parent_element- ja parent- sarakkeista luotiin yhdistetty indeksi, koska niitä ei käytetä missään kyselyssä erikseen. Xmach-1:n testidatassa on myös erillinen hakemistodokumentti (directory), joka sijoitettiin Binary Approachissa omaan erilliseen tauluunsa dokumentin poikkeavan rakenteen vuoksi. Directory-taululle annettiin kolme indeksiä: polkuindeksi, nimi-indeksi ja id-indeksi. Polkuindeksi koostuu kuudesta sarakkeesta (ahost, bhost, chost, apath, bpath ja cpath), jotka muodostavat kullekin dokumentille URL:n. Koska dokumentin polun hakeminen lähtee aina URL:n alusta, voitiin polun osat yhdistää saman indeksin alle. Nimi-indeksi sisältää pelkän dokumentin nimen. Koska dokumentin nimeä voidaan kysellä polusta irrallaan, täytyi dokumentin nimi irrottaa omaksi erilliseksi indeksiksi. Id-indeksi rakentuu pelkän doc_id -sarakkeen pohjalta.

Edge Approachin indeksit poikkeavat hieman Binary Approachista erilaisen tietokantarakenteen vuoksi. Florescu ja Kossman suosittelevat Edge Approachin indekseiksi nimi-arvo ja id -indeksejä (Florescu & Kossman 1999b, s. 29). Florescun ja Kossmanin suositusten lisäksi Tian et al. mukaan on erittäin tärkeää käyttää vanhempi-järjestys ja dokumentti -indeksejä (Tian et al. 2001, s. 8). Florescun ja Kossmanin ja Tian et al. esittämien perusteella sekä analysoimalla Xmach-1:n kyselyjä Edge Approachin indekseiksi valittiin doc_id, name-value, parent ja node_order sarakkeet. Hakemisto-dokumentti (directory) on tallennettu Edge Approachissa erilliseen tauluun ja sen indeksointi on sama kuin Binary Approachissa.

Lu et al. toteavat tutkimuksessaan natiivien XML -tietokantojen suorituskyvyn olevan riippuvainen indeksien käytöstä ja he suosittelevat käytettäväksi natiivien XML -tietokantojen kaikkia indeksejä (Lu et al. 2005). Xindicen kehittäjät suosittelevat

indeksien käyttöä ja toteavat Xindicen suorituskyvyn olevan merkittävästi heikompi indeksoimattomalla tiedolla (Apache Software Foundation 2005b). Lu et al. toteavat indeksien puuttumisen näkyvän huonona suorituskykynä erityisesti osittaisia polkuilmauksia käyttävissä kyselyissä (Lu et al. 2005).

Natiiveista XML -tietokannoista eXist tarjoaa hieman monipuolisemmat indeksointivaihtoehdot. Existissä on tarjolla kolme erilaista indeksointia: rakenne, teksti (fulltext) ja vaihteluväli -indeksit (eXist 2005). Rakenneindeksi sisältää tiedot kokoelmassa (collection) esiintyvistä elementeistä ja attribuuteista. Teksti-indeksi mahdollistaa eXistin tekstihakulaajennusten käytön, ja se pitää tietoa merkkien ja merkkijonojen esiintymisistä tekstisolmuissa. Rakenne- ja teksti-indeksit luodaan oletusarvoisesti automaattisesti. Vaihteluväli-indeksi tehostaa hakuja sellaisiin solmuihin, joihin voidaan tehdä vertailuja, kuten esimerkiksi numeerisen arvon omaaviin solmuihin. Existille indekseiksi valittiin kaikki kolme indeksointia. Teksti-indeksiin valittiin otettavaksi mukaan kaikki solmut sekä myös alphanumeeriset arvot. Vaihteluväli-indeksiin valittiin kahdeksan solmua. Hakemisto-dokumentista vaihteluväli-indeksiin valittiin solmut host@name, path@name, doc_info@doc_id ja doc_info@update_time. Hallittaville dokumenteille vaihteluväli-indeksiin valittiin solmut @doc_id, author, @author ja link@xlink:href. Kaikkien vaihteluväli-indeksien tietotyypiksi valittiin merkkijono (xs:string). Rakenneindeksistä on huomattava, että sitä ei voida poistaa käytöstä, joten se on aina automaattisesti käytössä.

Xindicen indeksointi ei ole aivan yhtä monipuolinen kuin eXistin indeksointi. Xindicessä voidaan indeksoida sekä elementit että attribuutit käyttämällä täsmällisiä polkuja, nimiä tai jokerimerkkiä (*) (Apache Software Foudation 2005b). Xindicelle määriteltiin indeksit Xmach-1:n kyselyjen perusteella. Indeksejä määritettiin seitsemän kappaletta. Indekseistä kolme oli hakemistodokumentin indeksejä (host@name, path@name ja doc_info@update_time), yksi (*@doc_id) kattoi sekä

hakemistodokumentin että hallittavat dokumentit ja loput kolme indeksiä koskivat hallittavia dokumentteja (author, *@author ja link@xlink:href).

5.4.3 Testiajon parametrit

Xmach-1:n referenssitoteutuksessa on mahdollista vaihdella testin asetuksia usean eri parametrin kautta. Tärkeimmät testiajoon vaikuttavat parametrit ovat esitelty taulukossa 10.

Taulukko 10. Testin parametrit

Parametri	Arvo testissä	Selitys
clientCount	5	Asiakassäikeiden määrä: kuinka monta yhtäaikaista asiakaskonetta lähettää kyselyjä.
clientSleep	10	Kuinka monta millisekuntia asiakas pitää taukoa vastauksen saatuaan ennen uuden kyselyn lähettämistä.
documentCount	1000	Dokumenttien lukumäärä.
operationToPerform	400 ja 800	Suoritettavien kyselyiden kokonaismäärä. Huom. Ei alkuperäisessä referenssitoteutuksessa.

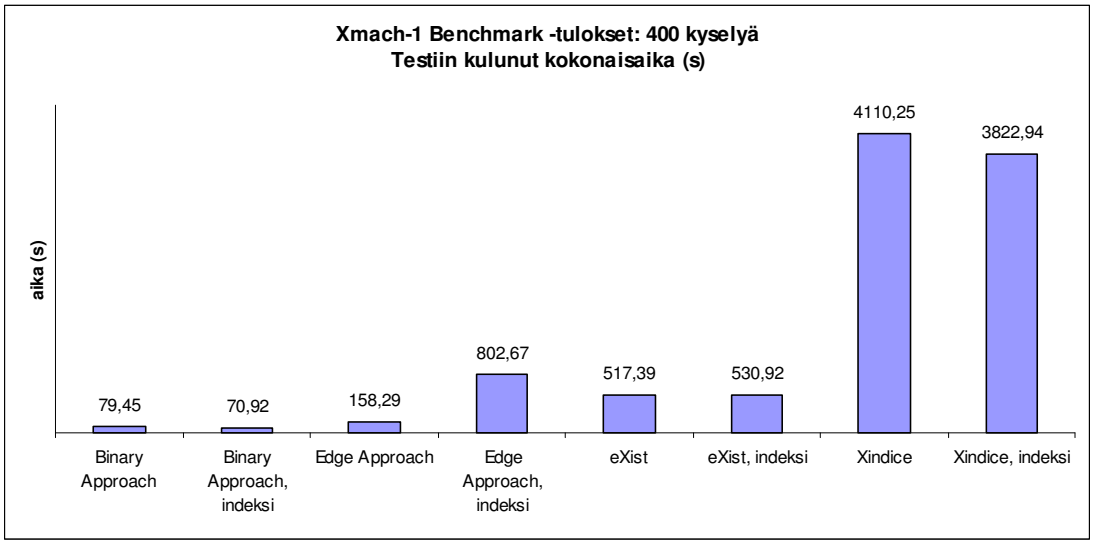
Testejä varten päätettiin muokata referenssitoteutuksen lähdekoodia siten, että operationToPerform -parametrin arvoa voitiin muuttaa asetustiedoston kautta. Testiajoja ajettiin kahdella eri operationToPerform -parametrin arvolla, kuten taulukossa 10 on esitetty. Kullekin testattavalle järjestelmälle, sekä indeksoimattomalle että indeksoidulle tietokannalle, ajettiin kaksi peräkkäistä testiajoa, joista parempi tulos otettiin huomioon. Ensimmäinen testikierros ajettiin operationToPerform -parametrin arvolla 400. Toiseen testikierrokseen operationToPerform -parametrin arvoksi muutettiin 800. Tietokantoja ei tässä

vaiheessa palautettu varmuuskopioista, vaan toinen testiajo ajettiin tietokantoihin, joita ensimmäinen testikierros oli päivittänyt. Ensimmäisen 800 kyselyn testiajon jälkeen selvisi, että referenssitoteutus suorittaa ensimmäiset 400 kyselyä taulukon yhdeksän jakauman mukaan ja loput kyselyt ovat tyyppiä M3, joka on hakemistodokumentin päivityksen tekevä kysely. Tällöin päivitysoperaatioiden (M3) määrä kaikista kyselyistä nousi hieman yli 50 %. Päivityskyselyiden suuren määrän vuoksi ensimmäisen testikierroksen tuloksilla on suurempi painoarvo.

5.4.4 Xmach-1 -testin tulokset

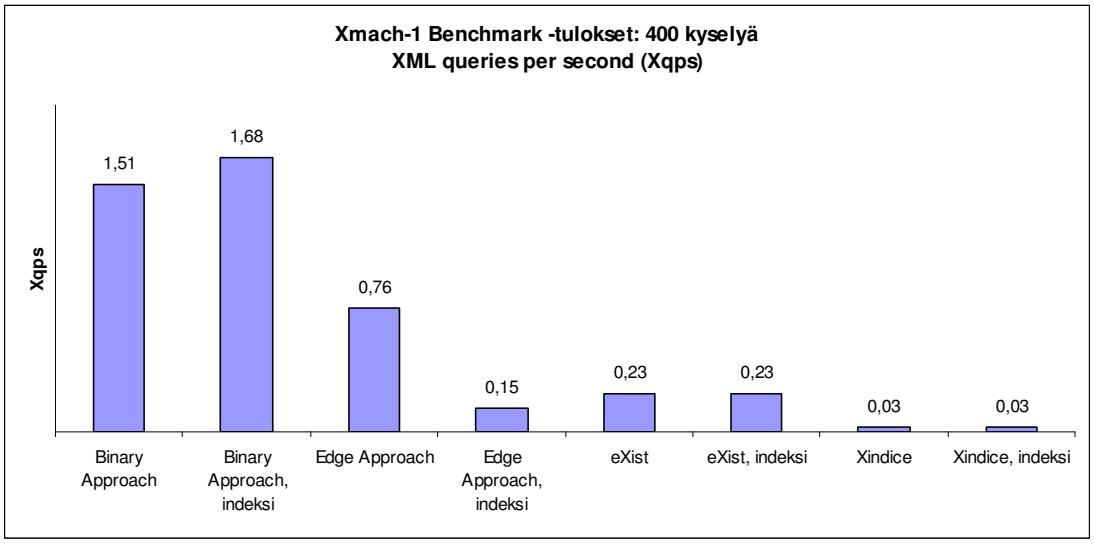
Testiajoja kertyi yhteensä 32 kappaletta. Kullekin testattavana olevalle XML tiedonhallintajärjestelmälle tehtiin siis kahdeksan testiajoa, joista neljässä ensimmäisessä käytettiin 400 kyselyä ja jälkimmäisessä neljässä 800 kyselyä. Sekä indeksoimattomalle että indeksoiduille tietokannoille ajettiin kaksi testiä sekä 400 että 800 kyselyllä. Parempi tulos valittiin testituloksiin. Seuraavaksi esitellään testitulokset. Testitulokset ovat analysoitu kappaleessa 5.4.5.

Ensimmäisellä testikierroksella ajettiin XMach-1 -testi 400 kyselyllä. Referenssitoteutus listasi testiajoista tilastolliset tiedot, joista tärkeimmät yksittäiset kohdat olivat testiajoon käytetty aika ja suoritettuja XML-kyselyjä sekuntia kohden (Xqps). Kuvassa 21 on esitettynä 400 kyselyyn käytetyt kokonaisajat. Kuvasta voidaan selvästi huomata, että relaatiotietokantapohjainen Binary Approach on suoritusajaltaan joukon paras. Myös Edge Approch suoriutui testistä hyvin, joskin indeksoidun tietokannan huono suorituskyky hämmästyttää. Exist on molemmilla tietokannoilla tasaisen varma suoriutuja. Xindicen todella pitkät suoritusajat yllättivät. Merkillepantavaa on todella suuri ero eXistiin, sillä sekä Xindicen että eXistin kyselyt olivat XPathilla toteutettuja ja keskenään lähes identtisiä.



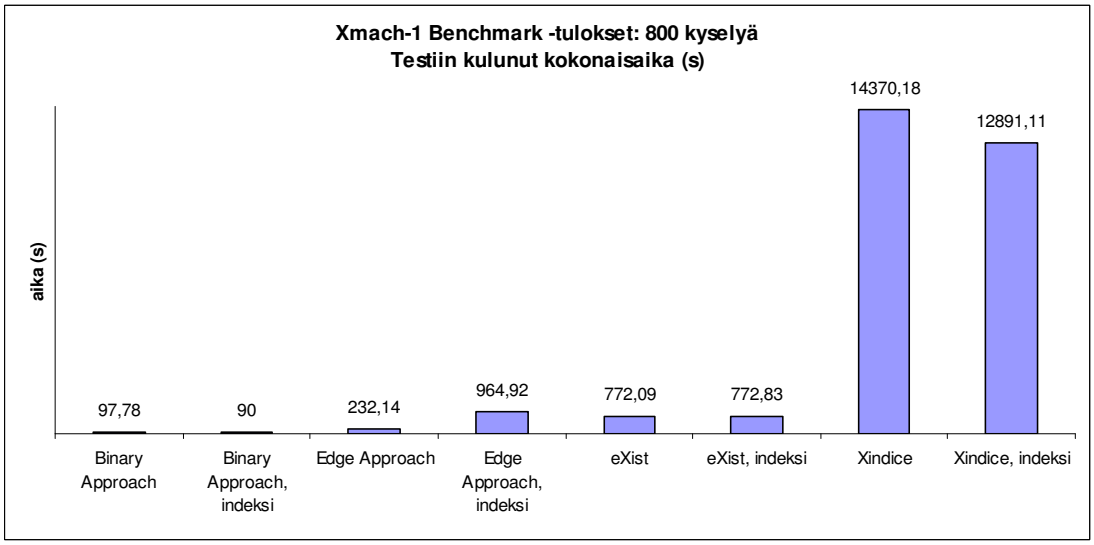
Kuva 21. Xmach-1 testin suoritusajat 400 kyselyllä (pienempi arvo parempi)

Referenssitoteutus laskee testitulosten pohjalta myös suorituskykyä kuvaavan tunnusluvun suoritettuja XML kyselyjä sekunnissa (Xqps). Kuten aiemmin kappaleessa 5.2 mainittiin, määräytyy Xqps:n arvo ensimmäisen kyselyn, Q1, suorituskyvyn perusteella. Kuvassa 22 ovat esiteltynä ensimmäisen testikierroksen Xqps -arvot. Kuvan 22 tulokset vastaavat hyvin kuvan 21 tuloksia, joten Böhmen ja Rahmin väittämää Xqps:n arvon määräytymisestä kyselyn Q1 perusteella voidaan pitää oikeana (Böhme & Rahm 2001, s. 9). Testituloksista Binary Approachin Xqps -arvoa voidaan pitää kohtuullisen hyvän testilaitteiston resursseihin nähden. Existin Xqps -arvoista voidaan päätellä, että teksti- ja vaihteluväli-indekseistä ei ole hyötyä tällä testidatalla ja kyselyillä.



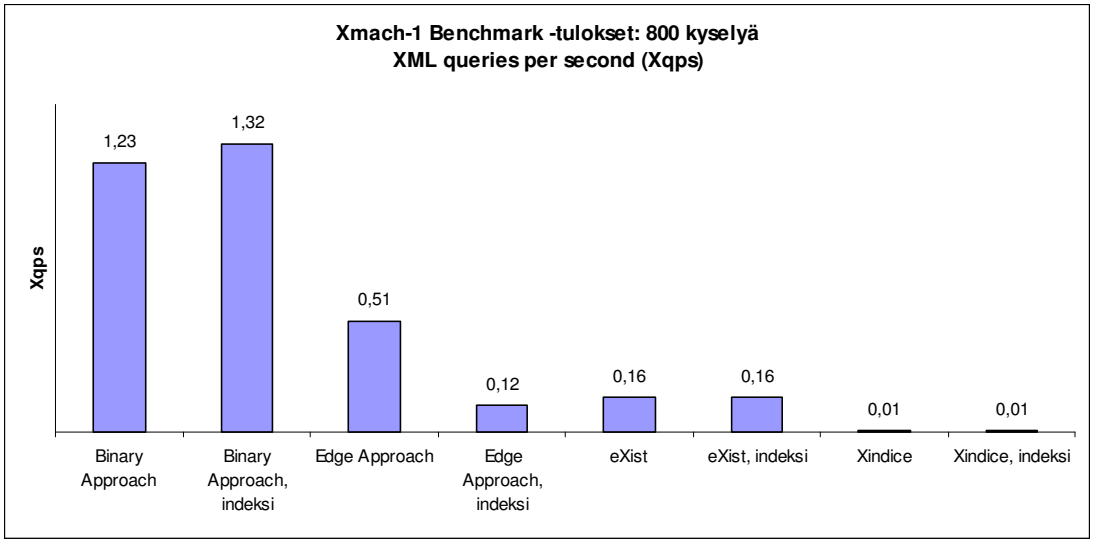
Kuva 22. Xmach-1 -testin Xqps -arvot 400 kyselyllä (suurempi arvo parempi)

Toisella testikierroksella käytettiin 400 kyselyn sijasta 800 kyselyä. Kuten edellisessä kappaleessa mainittiin, 800 kyselyn testin koostumus poikkesi huomattavasti 400 kyselyn testistä päivityskyselyjen (M3) huomattavan suuren määrän takia. Tuloksissa tämä näkyy siten, että erot suorituskyvyssä korostuvat entisestään. Kuvassa 23 on esitettynä 800 kyselyn testiin käytetyt suoritusajat.



Kuva 23. Xmach-1 testin suoritusajat 800 kyselyllä (pienempi arvo parempi)

Verrattaessa kuvien 21 ja 23 arvoja voidaan huomata, että Xindicen suoritusajan ja eXistin suoritusajan välinen suhde on melkein kolminkertaistunut. Muuten testattujen järjestelmien keskinäiset erot ovat pysyneet suhteellisen samoina kuin 400 kyselyn testissä. Xqps -arvoissa (kuva 24) on sen sijaan tapahtunut noin 20-30 % aleneminen verrattuna 400 kyselyn arvoihin (kuva 22). Suorituskyvyn heikkeneminen on tapahtunut kaikille testattaville järjestelmille ja siten sijoitukset keskinäisessä suorituskykyvertailussa eivät ole muuttuneet.



Kuva 24. Xmach-1 -testin Xqps -arvot 800 kyselyllä (suurempi arvo parempi)

5.4.5 Testitulosten analysointia

Tässä kappaleessa käydään läpi jokaisen testatun järjestelmän testitulokset ja pyritään esittämään hyvään tai huonoon testitulokseen johtaneita syitä.

Relaatiotietokantaan perustuvat XML-tiedonhallintajärjestelmät ovat yhtä tapausta lukuun ottamatta selvästi natiiveja XML -tietokantoja suorituskykyisempiä. Tulos ei ole suuri yllätys, sillä vastaavanlaisia tuloksia ovat saavuttaneet omissa tutkimuksissaan muun muassa Lu et al. ja Fong et al. (Lu et al. 2005, Fong et al. 2003). Yhtenä selittävänä tekijänä relaatiotietokantojen tehokkuuteen voidaan Fong et

al. mukaan pitää suhteellisen pientä dokumenttien lukumäärää, joka oli tässä testissä tuhat (Fong et al. 2003, s. 563).

Binary Approach oli molemmissa testeissä ja molemmilla mittareilla mitattuna selvästi suorituskykyisin ratkaisu. Etukäteen Binary Approachin tietokantataulujen suuri lukumäärä, 213, ja siitä johtuva liitosten suuri lukumäärä aiheutti epäilyksiä suorituskyvyn suhteen. Liitosten määrä ei kuitenkaan kasva liian suureksi yhdessäkään kyselyssä. Kun XMach-1:n kyselyt määrittelevät tarkasti halutun dokumentin ja elementin jälkiliitteen (suffix), voidaan kyselyt suorittaa ilman turhia tietokantahakuja. Binary Approachin vahvuudet tulevat ilmi kyselyissä Q2-Q8, joissa se on muita järjestelmiä selvästi nopeampi. Etukäteen vaikeissa kyselyissä Q1, M1, M2 ja M3, Binary Approach käyttää Edge Approachiin verrattuna vain 30 % - 100 % enemmän aikaa. Kyselyn Q1 arveltiin vievän relaatiotietokantapohjaisissa ratkaisuissa paljon aikaa, koska haettu solmujoukko järjestetään DOM-objektiksi, joka sarjallistetaan XML-merkkijonoksi. Uuden dokumentin lisääminen (kysely M1) on Binary Approachissa aikaa vievin operaatio, koska XML-dokumentin jokainen solmu täytyy tallentaa erikseen tietokantaan.

Edge Approachin huono suorituskyky indeksoidulla tietokannalla oli yllätys. Edge Approachin Binary Approachia huonompi suorituskyky on selitettävissä vähäisellä, mutta sitäkin suurempien tietokantataulujen määrällä. Edge Approachissa tietokantatauluja on 42 kappaletta, joissa jokaisessa on keskimäärin 4006 tietuetta, kun Binary Approachin 213 tietokantataulussa on keskimäärin 793 tietuetta. Edge Approachissa joudutaan läpikäymään paljon turhaa tietoa, koska kaikki dokumentin elementit ovat tallennettuna samaan tauluun. Esimerkiksi haku, jossa pitää hakea kaikki author-attribuutit aiheuttaa sen, että kaikki dokumenttitaulut on läpikäytävä, kun Binary Approachissa author-attribuutit löytyvät kaikki yhdestä taulusta. Indeksoidun tietokannan huono suorituskyky voidaan selittää indeksien suurella koolla. Binary Approachissa indeksin keskimääräinen koko on 172 kilotavua ja

joukossa on vain kaksi yli megatavun indeksiä. Edge Approachin indeksin keskimääräinen koko 702 kilotavua ja yli megatavun kokoisia indeksejä on peräti yksitoista kappaletta. Pienellä muistikapasiteetilla suurten indeksien käsittely laskee järjestelmän suorituskykyä.

Natiiveista XML -tietokannoista eXist oli suorituskykyisempi. Syy Binary Approachia heikompaan suorituskykyyn voi olla yksinkertaisesti järjestelmien toteutuksessa. Natiivi XML -tietokanta on rakennettu siten, että se käsittelee suurikokoisempia XML-dokumentteja nopeammin, kun taas relaatiotietokanta on nopeampi pienten XML-dokumenttien käsittelyssä. Toisaalta kyselyjen tyypillä on suuri merkitys suorituskykytestauksessa. Testeissä huomattiin, että eXist on Binary Approachia selkeästi nopeampi kyselyissä Q1, M1 ja M2. Esimerkiksi kokonaisen dokumentin hakeminen on Binary Approachilla vie lähes kaksinkertaisen ajan verrattuna eXistiin. Relaatiotietokantojen heikompi suorituskyky kyselyssä Q1 oli ennakoitavissa, sillä sekä Florescu ja Kossman että Lu et al. ovat saaneet tutkimuksissaan tämän kaltaisia tuloksia (Florescu & Kossman 1999b, Lu et al. 2005).

Existissä on mahdollista käyttää sekä XPathia että XQueryä kyselykielenä. Testit ajettiin XPathilla, koska XQueryllä tehdyt testikyselyt olivat ainakin XML:DB -rajapinnan kautta ajettuna todella hitaita. Lisäksi käyttämällä XPathia, voitiin tehdä vertailu eXistin ja Xindicen välillä, sillä Xindice tukee vain XPathia.

Existissä teksti-indeksin käyttö vaatii XQueryn ja erikoisfunktioiden, kuten near ja match käyttöä (eXist 2005). Kaikki kyselyt olivat toteutettu käyttäen XPathia, jossa ei ole near- tai match-funktioita, joten teksti-indeksille ei tullut käyttöä. Vaihteluväli-indeksi on hyödyllisin ei-merkkijono -tyyppiselle tiedolle, mutta sen pitäisi nopeuttaa hakuja myös merkkijonoista (eXist 2005). Testitulosten mukaan vaihteluväli-indeksin käyttö ei kuitenkaan parantanut järjestelmän suorituskykyä.

Xindicen erittäin huonoja testituloksia voidaan pitää yllätyksenä, sillä verkko-selaimessa toimiva Xindicen kokoelmaselain hakee dokumentit erittäin nopeasti. Kokoelman selaaja ei kuitenkaan käytä XML:DB:tä vaan suoraan paikallisia luokkia (Core Server API). Xindicen suorituskykyä testattiin myös XML-RPC -rajapinnan kautta. Tulokset XML-RPC -rajapintaa käyttäen olivat yhtä huonoja, kun XML:DB -rajapinnan käytettäessä. Syy Xindicen huonoon suorituskykyyn voi olla sen XML:DB ja XML-RPC -ajureissa. Xindicen kehitysyhteisön virallisesti suosittelema ohjelmointirajapinta Xindicea käyttäville Java-sovelluksille on XML:DB (Apache Software Foundation 2005a).

5.5 Teknisten ominaisuuksien analysointia

Tässä kappaleessa analysoidaan testattujen järjestelmien teknisiä ominaisuuksia yleisellä tasolla sekä tuodaan esille suorituskykytestin valmistelussa ilmenneitä seikkoja. Kappaleessa 5.3 esitettyä kokonaiskustannusten (TCO) arviointia ei testatuille järjestelmille suoritettu, sillä olemassa olevilla resursseilla saavutetut tulokset eivät olisi olleet kelvollisia.

5.5.1 Infrastruktuurien arviointi

Infrastruktuurien arviointikriteereinä käytetään kappaleessa 5.3 esiteltyjä, Schmidt et al. laatimia kriteereitä (Schmidt et al. 2001, s. 31). Infrastruktuurin arvioinnissa keskitytään kolmeen osa-alueeseen: rajapintoihin, tuloformaatteihin ja kyselykieliin.

Relaatiotietokantapohjaiset XML-tiedonhallintajärjestelmät, Binary ja Edge Approach, ovat toteutettu käyttäen MySQL-tietokantaa. MySQL tarjoaa neljä yhteysrajapintaa (connector): ODBC, JDBC, MySQL Connector/ MXJ ja MySQL Connector/ NET (MySQL AB 2005). Suorituskykytestit ajettiin JDBC-rajapinnan

kautta. Testien aikana JDBC-rajapinnassa ei havaittu ongelmia tai epävakautta. Lisäksi rajapinnan selkeä ja kattava dokumentaatio helpotti toteutustyötä. MySQL:n yhteysrajapinnoista voidaan todeta, että ne ovat hyvin dokumentoituja, toimivia ja suorituskykyisiä.

Suorituskykytesteissä käytetyt natiivit XML -tietokannat, eXist ja Xindice tarjoavat myös kumpikin useamman kuin yhden yhteysrajapinnan. Exist tukee neljää yhteysrajapintaa: XML:DB API:a, XML-RPC:tä, REST:ia ja SOAP:ia (eXist 2005). Existille suorituskykytestit ajettiin käyttäen XML:DB API:a, jota eXistin kehittäjät suosittelevat käytettäväksi Java-sovelluskehityksessä. Existin suorituskykyä XML:DB API:n kautta voidaan pitää kohtuullisena. Existin XML:DB:n dokumentaatiossa on mainittu kaikki oleellinen, mutta se ei kuitenkaan ole MySQL:n dokumentoinnin tasoinen. Existin XML:DB API:ssa ilmeni testien aikana virheitä, jotka saatiin korjattua. Virheiden korjaus vei runsaasti aikaa, sillä ratkaisua täytyi etsiä eri postituslistoilta sekä analysoimalla virheilmoituksia.

Xindicen XML:DB API:n dokumentaatio on pitkälti eXistin dokumentaation tasoista, eli niukkaa, mutta riittävää useimpiin tapauksiin. Xindicen XML:DB API toimi testeissä ongelmitta, mutta sen suorituskyky oli huono. Xindicen osalta kokeiltiin myös XML-RPC:tä, jonka dokumentaatio oli heikkotasoinen. Xindice XML-RPC:n dokumentaatio oli kokonaisuudessaan yksi Perlillä toteutettu esimerkki sekä Java-API (Java Docs), joka oli kommenteiltaan erittäin niukka. XML-RPC:tä käyttävän yhteysmoduulin toteuttamiseen löytyi apua eXistin dokumentaatiosta, mutta yksityiskohdat täytyi etsiä Xindicen lähdekoodista. Lisäksi Xindicen XML-RPC -yhteysrajapinnasta löytyi yhteensopivuusongelmia eri komponenttien kesken, mikä aiheutti lisää ongelmia toteutusvaiheessa.

Tulosformaateiltaan natiivit XML -tietokannat ovat selvästi relaatiotietokantoja monipuolisempia ja käyttökelpoisempia. Siinä missä MySQL:n tulokset ovat aina

tauluobjektina, tarjoavat eXist ja Xindice hakujen tulokset XML-merkkijonoina (String), DOM-objekteina tai suoraan XML-tiedostoksi sarjallistettuna (serialization). MySQL:sta haettu XML-dokumentti on muutettava välikerroksen avulla esimerkiksi DOM-objektiksi tai XML-merkkijonoksi. Natiiveista XML -tietokannoista eXistin XML-merkkijonoformaatti on käyttökelpoisempi, sillä siinä ei ole mitään metatietoa, vaan esimerkiksi elementti tulostuu ruudulle sellaisenaan. Xindice sen sijaan lisää oletuksena jokaiseen tulokseen metatietoja, jolloin yksinkertaisestakin hakutuloksesta tulee erittäin vaikeaselkoista. Tulosformaateista voidaan yhteenvetona todeta, että natiivit XML -tietokannat ovat tässä asiassa relaatiotietokantoja edellä.

Kyselykielten osalta tilanne on tulosformaattien tilanteen kaltainen. MySQL tarjoaa relaatiotietokantojen tapaan vain yhden kyselykielen, SQL:n. SQL:n voidaan todeta olevan kypsä ja tehokas kyselykieli, joka on hyvin standardoitu ja dokumentoitu. Natiiveista XML -tietokannoista Xindice tukee XPathia ja XUpdatea. XPathia käytetään kyselyihin ja XUpdatella päivitetään dokumentteja. Exist on monipuolisempi myös kyselykielissä tarjoten tuen XPathille, XUpdatelle sekä XQuerylle. XPathiin verrattuna XQuery on ilmaisuvoimaisempi, mutta monimutkaisempi kyselykieli. XQuery-tuki on kuitenkin eräs eXistin suurimmista eduista Xindiceen nähden.

5.5.2 Suorituskykytestien valmistelussa tehtyjä havaintoja

Suorituskyky testien olennaisena osana oli suunnitella ja toteuttaa moduulit, joiden avulla XMach-1:n testausosa pystyi kommunikoimaan testattavien järjestelmien kanssa. Moduulit toteutettiin Javalla.

Suorituskykytestien valmistelussa ensimmäisenä vaiheena oli toteuttaa lataajamoduulit (bulkloader) kullekin testattavalle järjestelmälle, sillä tuhannen dokumentin manuaalinen lataaminen ei tullut kysymykseen sen vaatiman aikamäärän

takia. Lataajamoduulit toteutettiin Javalla. Lataajamoduuleista eXistin ja Xindicen moduulit olivat lähes identtisiä, kun taas Binary Approachin ja Edge Approachin moduulit olivat yhteneviä vain hakemistodokumentin lataavan metodin osalta. Lataajamoduulien toteutukseen käytetystä ajasta yli 90 % kului Binary Approachin ja Edge Approachin moduuleihin. Tämä huomattavan epätasainen resurssien jakautuminen johtuu siitä, että eXistiin ja Xindiceen XML-dokumentit voidaan ladata parilla yksinkertaisella lauseella, kun taas relaatiotietokantaan ladattaessa jokainen XML-dokumentin solmu täytyy käsitellä ja tallentaa erikseen. Tämä näkyi myös lataukseen käytetyissä ajoissa, jotka esiteltiin kappaleessa 5.4.1. Latausaikojen ero oli suurimmillaan Xindicen ja Binary Approachin välillä; Binary Approach käytti lataukseen lähes satakertaisen ajan verrattuna Xindiceen. Latausmoduulien Java-koodin pituudessa oli myös selvä ero, sillä Binary ja Edge Approachin moduulit sisälsivät lähes nelinkertaisen määrän koodia eXistin ja Xindicen moduuleihin verrattuna.

Olennaisin osa suorituskykytestien valmistelua oli kyselymoduulien toteuttaminen. Haastavin osa kyselymoduulien toteutusta oli kääntää XMach-1:n kyselyt Q1-Q8 ja M1-M3 kullekin käytetylle kyselykielelle. Kuten aiemmin mainittiin, Binary Approach ja Edge Approach käyttivät kyselykielenä SQL:ää, kun taas eXist ja Xindice käyttivät XPathia. XMach-1:n WWW-sivuilla on esitetty esimerkkitoteutukset kyselyistä Q1-Q8 käyttäen XQueryä (Böhme & Rahm 2005). Koska XQuery on kehitetty XML:ää silmälläpitäen, ovat sen perusteet erilaiset kuin SQL:n, vaikka kielillä on myös joitain yhteisiä piirteitä. XPathilla ja XQuerylla on paljon yhteistä, sillä XPath on XQueryn osajoukko ja XQuery käyttää polkumäärityksiin juuri XPathia. XPathin ja XQueryn yhtäläisyydestä johtuen, eXistin ja Xindicen kyselyt olivat huomattavasti helpompia toteuttaa, kuin Binary Approachin ja Edge Approachin SQL-kyselyt. Lisäksi eXistin ja Xindicen kyselyistä voidaan todeta, että ne olivat identtisiä yhtä poikkeusta lukuun ottamatta, joka liittyi XPathin distinct-funkion käyttöön. Koska XPathista ei löydy kaikkia XQueryn

toimintoja, täytyi kaikkien kyselyjen toteutuksessa hyödyntää Javaa. Erityisesti Javan silmukkarakenteet for ja while olivat hyödyllisiä toteutettaessa kyselyitä. XQueryn ominaisuuksiin kuuluvat silmukkarakenteet vakiona. Javan käyttöä ei kuitenkaan voida pitää tuloksia vääristävänä, sillä suorituskykytesteissä oli kyse tiedonhallintajärjestelmän, ei kyselykielen suorituskyvyn testaamisesta. Nykyään verkkosovelluksissa on suurilta osin luovuttu kaksikerrosmallin käytöstä, jossa asiakassovellus kommunikoi suoraan tiedonhallintajärjestelmän kanssa. Sen sijaan asiakassovelluksen ja tiedonhallintajärjestelmän väliin on sijoitettu sovelluskerros, joka huolehtii kyselyjen tulkkauksesta ja välityksestä sekä tulosten välittämisestä niiden välillä. Kyselymoduuleista eXistin ja Xindicen moduulit olivat hieman nopeampia toteuttaa ja lisäksi niiden Java-koodin määrä oli 25 % vähäisempi. Kyselyjen optimoinnissa SQL oli XPathia edellä, sillä SQL:stä oli löydettävissä huomattavasti enemmän dokumentteja koskien kyselyjen optimointia. Toisaalta XPath-kyselyissä noudatettiin mahdollisimman pitkälle XMach-1:n WWW-sivuilla esitettyjä esimerkkikyselyjä, jolloin kyselyjen optimointiin ei nähty tarvetta (Böhme & Rahm 2005). Kyselyjen toteuttamisesta voidaan todeta, että SQL-kyselyjen toteuttaminen oli vaativampaa ja vaati XPath-kyselyjä enemmän resursseja, mutta SQL:n paremman dokumentaation ansiosta tiedon etsimiseen käytettiin vähemmän aikaa.

Testattujen järjestelmien infrastruktuurien yleisestä toimivuudesta tehtiin useita havaintoja. MySQL:n päällä toimivat Binary Approach ja Edge Approach toimivat odotetusti. Natiivien XML -tietokantojen, eXistin ja Xindicen kanssa oli sen sijaan useita ongelmia, jotka voivat johtua siitä, että testeissä käytettiin niiden beta-versioita. Perustelu Beta-versioiden käytölle on, että eXistin WWW-sivuilla suositellaan sen käyttöä ja Xindicen viimeisin vakaa versio (1.0) on julkaistu maaliskuussa 2002. Existin kanssa ongelmia aiheuttivat aluksi nimiavaruudet, jotka johtuivat osin epäselvästä dokumentaatiosta sekä testidatasta. Nimiavaruusongelma saatiin korjattua eXistin lähdekoodia muokkaamalla. Toinen eXistin ongelma liittyi

vanhentuneisiin XML-RPC -rajapintoihin, minkä takia kyselyjen tulokset tulostuivat väärässä merkistössä. Existin keskusteluryhmissä tämä ongelma todettiin koskevan vain Windows 2000 Advanced Serverilla toimivia eXist-tietokantoja. XML-RPC -ongelma korjattiin päivittämällä Apachen XML-RPC -luokkakirjastot versiosta 1.1 versioon 2.0 (eXist Community 2005). Korjausten jälkeen eXist toimi suorituskykytesteissä ilman ongelmia. Tehdyt korjaukset aiheuttivat tosin sen, että eXistin verkkokäyttöliittymä lakkasi XQuery-Servletin osalta toimimasta. Xindicen kanssa infrastruktuuriongelmia ei kohdattu testeissä. Ainoat testeissä kohdatut toiminnallisuuteen liittyvät ongelmat koskivat eräiden XPath-funktioiden toimimattomuutta. XPath-funktioiden puutteellinen tuki on tunnustettu Xindicen WWW-sivuilla (Apache Software Foundation 2005a). Kuten kappaleessa 5.4.5 kerrottiin, päätettiin XML:DB:n heikon suorituskyvyn takia kokeilla testejä XML-RPC -rajapinnan kautta. Xindicen XML-RPC -rajapinnan käyttö osoittautui erittäin hankalaksi. Dokumentaatio oli olematonta ja ainoat esimerkit XML-RPC -rajapinnan käytöstä löytyivät Perl-kielisinä. Lisäksi rajapinta kärsi virheistä, joihin löytyi viittauksia Xindicen keskusteluryhmistä, mutta ei ratkaisuja tai korjauksia. Yhteenvetona testattujen järjestelmien infrastruktuurista voidaan todeta, että MySQL ja JDBC ovat luotettavia ja toimivia teknologioita. Natiivit XML -tietokannat ovat vielä sen sijaan hieman keskeneräisiä tuotteita ja niiden käytettäessä voidaan kohdata yllättäviä ongelmia.

6. YHTEENVETO

Tässä diplomityössä tutustuttiin erityyppisiin XML-tiedonhallintajärjestelmiin. Eri ratkaisujen esittelyn ja analysoinnin jälkeen valittiin neljä toteutusta suorituskykytestiin ja tekniseen arviointiin. Suorituskykytestiksi valittiin Böhmen ja Rahmin kehittämä XMach-1, koska se mallintaa parhaiten verkkokäytön kaltaista tilannetta, jossa useat yhtäaikaiset asiakkaat lähettävät lyhyitä kyselyjä tietokantaan (Böhme & Rahm 2001). XMach-1:n testidata vastasi ominaisuuksiltaan testidatalle asetettuja kriteereitä. Testidatassa dokumenttien koko ja rakenteen vaihtelevuus vastasivat vaatimuksia. Suorituskykytestit suoritettiin kahdelle relaatiotietokantapohjaiselle ratkaisulle ja kahdelle natiiville XML -tietokannalle. Relaatiotietokantaan perustuvat ratkaisut olivat Florescun ja Kossmanin kehittämät Binary Approach ja Edge Approach (Florescu & Kosmann 1999b). Testatut natiivit XML -tietokannat olivat avoimeen lähdekoodin perustuvat eXist ja Xindice (eXist 2005 & Apache Software Foundation 2005).

Suorituskykytestien perusteella relaatiotietokantapohjainen Binary Approach on testijoukon suorituskykyisin tiedonhallintaratkaisu XML-dokumenteille. Binary Approachin heikkoudet suorituskyvyn osalta liittyvät kokonaisen XML-dokumentin hakemiseen tietokannasta sekä uuden dokumentin lisäämiseen tietokantaan. Lisäksi suuret päivitysoperaatiot voivat olla ongelmallisia Binary Approachia käytettäessä. Vastoin ennakko-odotuksia natiivit XML -tietokannat eivät vakuuttaneet suorituskyvyllään. Xindicen erittäin huonoa suorituskykyä voidaan pitää yllätyksenä. Dokumenttien massalataamisessa (bulkloading) natiivit XML -tietokannat olivat relaatiotietokantapohjaisia ratkaisuja nopeampia.

Natiivit XML -tietokannat ovat vaivattomampia integroida osaksi verkko-sovellusta kuin relaatiotietokannan päällä toimivat Binary Approach ja Edge Approach. Existin

ja erityisesti Xindicen puutteelliset dokumentaatiot vaikeuttavat niiden käyttöönottoa ja integrointia.

Ratkaisujen toimivuudessa ja luotettavuudessa relaatiotietokantapohjaiset ratkaisut ovat natiiveja XML -tietokantoja edellä. Suorituskykytestien tulosten, teknisten ominaisuuksien arviointien ja käytännön kokemuksien perusteella voidaan todeta, että testatuista ratkaisuista Binary Approach on käyttökelpoisin. Testatuista natiiveista XML -tietokannoista eXist on varteenotettava vaihtoehto, mutta siitä kannattaa odottaa ensimmäisen vakaan version (stable version) julkaisemista.

LÄHTEET

Apache Software Foundation. 2005a. Apache Xindice. Saatavissa http://xml.apache.org/xindice/index.html. viitattu 12.12.2005.

Apache Software Foundation. 2005b. Xindice 1.1 Administration Guide. Saatavissa http://xml.apache.org/xindice/guide-administrator.html. viitattu 7.12.2005.

Bourret, Ronald. 2004. XML and Databases. saatavissa http://www.rpbourret.com/xml/XMLAndDatabases.htm. viitattu 13.7.2005.

Brandin, Chris. 2003. Information Modeling with XML. teoksessa XML Data Management. toimittanut Chaudhri, Rashid & Zicari. Addison Wesley. ISBN 0-201-84452-4.

Bressan, Stéphane, Mong, Lee, Mong Li, Li, Ying Guang, Lacroix, Zoé & Nambiar, Ullas B. 2002. The XOO7 Benchmark. Proceedings of the first VLDB Workshop on Efficiency and Effectiveness of XML Tools, and Techniques (EEXTT). Hong Kong, China.

Bressan, Stéphane, Mong, Lee, Mong Li, Li, Ying Guang, Lacroix, Zoé & Nambiar, Ullas B. 2003. XML Management System Benchmarks. teoksessa XML Data Management. toimittanut Chaudhri, Rashid & Zicari. Addison Wesley. ISBN 0-201-84452-4.

Böhme, Timo & Rahm, Erhard. 2001. XMach-1: A Benchmark for XML Data Management. Proceedings of the German Database Conference (BTW2001). Oldenburg, Germany.

Böhme, Timo & Rahm, Erhard. 2005. Benchmarking XML Data Management Systems. Saatavissa: http://dbs.uni-leipzig.de/en/projekte/XML/XmlBenchmarking.html. viitattu 13.12.2005.

Celko, Joe. 2000. Trees in SQL. Intelligence Enterprise Magazine vol 3 no 16. saatavissa http://www.intelligententerprise.com/001020/celko.jhtml?_requestid=235427. viitattu 1.8.2005.

Chamberlin, Don. 2002. XQuery: An XML Query Language. IBM Systems Journal, vol 41, no 4.

Chaudri, Akmal B., Rashid, Awais & Zicari, Roberto. 2003. Preface. teoksessa XML Data Management. toimittanut Chaudhri, Rashid & Zicari. Addison Wesley. ISBN 0-201-84452-4.

Chen, Qun, Lim, Andrew & Ong, Kian Win. 2003. D(K)-Index: An Adaptive Structural Summary for Graph-Structured Data. Proceedings of the 2003 ACM SIGMOD international conference on Management of data. ACM Press. ISBN 1-58113-634-X.

Choi, Eun-Hye & Kanai, Tatsunori. 2003. XPath-based Concurrency Control for XML Data. Proceedings of the 14th Data Engineering Workshop (DEWS 2003).

Chung, Chin-Wan, Min, Jun-Ki & Shim, Kyuseok. 2002. APEX: An Adaptive Path Index for XML Data. Proceedings of the 2002 ACM SIGMOD international conference on Management of data. ACM Press. ISBN 1-58113-497-5.

Connolly, Dan. 2003. XML development history. W3C. Saatavissa http://www.w3.org/XML/hist2002. viitattu 12.7.2005.

Cooper, Brian F., Sample, Neal, Franklin, Michael J., Hjaltason, Gísli R. & Shadmon, Moshe. 2001. A Fast Index for Semistructured Data. Proceedings of the 27th International Conference on Very Large Data Bases. Morgan Kaufmann Publishers Inc. ISBN 1-55860-804-4.

Dekeyser, Stijn & Hidders, Jan. 2004. Conflict scheduling of transactions on XML documents. Proceedings of the fifteenth conference on Australasian database - Volume 27. Australian Computer Society, Inc.

Develnet.org. 2005. Das visuelle Modell. Saatavissa http://www.develnet.org/Tech/DasVisuelleModell. viitattu 1.8.2005.

Edvards, Richard. 2003. A Generic Architechture for Storing XML Documents in a Relational Database. teoksessa XML Data Management. toimittanut Chaudhri, Rashid & Zicari. Addison Wesley. ISBN 0-201-84452-4.

Eswaran, K. P., Gray, J. N., Lorie, R. A. & Traiger, I. L. 1976. The notions of consistency and predicate locks in a database system. Communications of the ACM Volume 19 , Issue 11. ACM Press. ISSN 0001-0782.

eXist. 2005. eXist - Open Source Native XML Database. Saatavissa http://exist.sourceforge.net. Viitattu 12.12.2005.

eXist Community 2005. eXist mailing lists. saatavissa http://news.gmane.org/gmane.text.xml.exist. viitattu 13.12.2005.

Feinberg, George. 2004. Anatomy of a Native XML Database. Sleepycat Software. saatavissa http://www.sleepycat.com/offers/xml/index.php?From=homepage. viitattu 5.8.2005.

Feng, Ling, Chang, Elizabeth & Tharam, Dillon. 2002. A Semantic Network-Based Design Methodology for XML Documents. ACM Transactions on Information Systems, Vol. 20, No. 4.

Fiebig, Thorsten, Helmer, Sven, Kanne, Carl-Christian, Moerkotte, Guido, Neumann, Julia, Schiele, Robert & Westmann, Till. 2002. Anatomy of a native XML base management system. The VLDB Journal volume 11 issue 4 (2002). Springer-Verlag. ISSN 1066-8888.

Florescu, Daniel & Kossman, Donald. 1999a. A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database. Institut National de Rescherche en Informatique et en Automatique (INRIA). ISSN 0249-6399.

Florescu, Daniel & Kossman, Donald. 1999b. Storing and Querying XML Data using an RDMBS. Bulletin of the Technical Committee on Data Engineering Vol. 22 No. 3. IEEE Computer Society.

Fong, Joseph, Wong, H. K. & Wong, Anthony. 2003. Performance Analysis between an XML-Enabled Database and a Native XML Database. teoksessa XML Data Management. toimittanut Chaudhri, Rashid & Zicari. Addison Wesley. ISBN 0-201-84452-4.

Han, Wook-Shin, Lee, Ki-Hoon & Lee, Byung Suk. 2003. An XML Storage System for Object-Oriented/Object-Relational DBMSs. Journal of Object Technology vol. 2, no. 3.

Haustein, Michael P. & Härder, Theo. 2004. A Lock Manager for Collaborative Processing of Natively Stored XML Documents. Proceedings of 19th Brazilian Symposium on Databases (SBBD 2004).

Helmer, Sven, Kanne, Carl-Christian & Moerkotte, Guido. 2004. Evaluating Lock-based Protocols for Cooperation on XML Documents. SIGMOD Record, Vol. 33, No. 1, March 2004.

Hidders, Jan, Paredaens, Jan, Vercammen, Roel & Demeyer, Serge. 2004. A Light but Formal Introduction to XQuery. Database and XML Technologies - Proceedings of the Second International XML Database Symposium, XSym 2004. Springer-Verlag. ISBN 3-540-22969-8.

Hohenstein, Uwe. 2003. Supporting XML in Oracle9i. teoksessa XML Data Management. toimittanut Chaudhri, Rashid & Zicari. Addison Wesley. ISBN 0-201-84452-4.

Härder, Theo & Reuter, Andreas. 1983. Principles of Transaction-Oriented Database Recovery. ACM Computing Surveys volume 15 number 4.

Jagadish, H. V., Al-Khalifa, Shurug, Chapman, Adriane, Lakshmanan, Laks V. S., Nierman, Andrew, Paparizos, Stelios, Patel, Jignesh M., Srivastava, Divesh, Wiwatwattana, Nuwee, Wu, Yuqing & Yu, Cong. 2002. Timber: A native XML database. . The VLDB Journal volume 11 issue 4 (2002). Springer-Verlag. ISSN 1066-8888.

Jea, Kuen-Fang, Chen, Shih-Ying & Wang, Sheng-Hsien. 2002. Concurrency Control in XML Document Databases: XPath Locking Protocol. Proceedings of the Ninth International Conference on Parallel and Distributed Systems (ICPADS'02).

Lawrence, Ramon. 2004. The Space Efficiency of XML. Information and Software Technology volume 46 issue 11. Elsevier.

Lewis, Philip M., Bernstein, Arthur & Kifer, Michael. 2002. Databases and Transaction Processing - An Application-Oriented Approach. Addison-Wesley. ISBN 0-201-70872-8.

Li, Quanzhong & Bongki, Moon. 2001. Indexing and Querying XML Data for Regular Path Expressions. Proceedings of the 27th International Conference on Very Large Data Bases. Morgan Kaufmann Publishers Inc. ISBN 1-55860-804-4.

Lu, Hongjun, Jiang, Haifeng, Xu Yu, Jeffrey, Yu, Ge, Wang, Guoren, Zhou, Aoying & Zhieng, Shihui. 2005. What Make the Differences: Benchmarking XML Database Implementations. artikkeli lehdessä ACM Transactions on Internet Technology volume 5 issue 1. ACM Press. ISSN 1533-5399.

Meier, Wolfgang M. 2003. eXist Native XML Database. teoksessa XML Data Management. toimittanut Chaudhri, Rashid & Zicari. Addison Wesley. ISBN 0-201-84452-4.

MySQL AB. 2005. MySQL Reference Manual. Saatavissa http://dev.mysql.com/doc/refman/4.1/en/index.html. Viitattu 12.12.2005.

Pardede, Eric, Rahayu, J.Wenny & Taniar, David. 2004. On Using Collection for Aggregation and Association Relationships in XML Object-Relational Storage. Proceeding of ACM Symposium on Applied Computing.

PostgreSQL Global Development Group. 2005. PostgreSQL 8.0.3 Documentation. Saatavissa http://www.postgresql.org/docs/8.0/interactive/. Viitattu 27.7.2005.

Price, Roger. 1997. Beyond SGML. teoksessa International Conference on Digital Libraries Proceedings of the third ACM conference on Digital libraries (Pittsburgh 1998). ACM Press. ISBN 0-89791-965-3.

Rao, Praveen & Moon, Bongki. 2004. PRIX: Indexing And Querying XML Using Prüfer Sequences. Proceedings of the 20th International Conference on Data Engineering. IEEE Computer Society. ISBN 0-7695-2065-0.

Runapongsa, Kanda, Patel, Jignesh M., Jagadish, H. V., Chen, Yun & Al-Khalifa, Shurug. 2003. The Michigan Benchmark: Towards XML Query Performance Diagnostics. Proceedings of the 29th VLDB Conference. Berlin, Germany.

Runapongsa, Kanda, Patel, Jignesh M., Bordawekar, Rajesh & Padmanabhan, Sriram. 2004. XIST: An XML Index Selection Tool. Database and XML Technologies - Proceedings of the Second International XML Database Symposium, XSym 2004. Springer-Verlag. ISBN 3-540-22969-8.

SAX. 2005. SAX. saatavissa http://www.saxproject.org. viitattu 5.8.2005.

Schmidt, A. R., Waas, F., Kersten, M. L., Florescu, D., Manolescu, I., Carey, M. J. & Busse, R. 2001a. The XML Benchmark Project. Technical Report INS-R0103, CWI. Amsterdam, The Netherlands.

Schmidt, A. R., Waas, F., Kersten, M. L., Florescu, D., Manolescu, I., Carey, M. J. & Busse, R. 2001b. Why And How To Benchmark XML Databases. SIGMOD Record, Vol. 30, No 3.

Schwentick, Thomas. 2004. XPath Query Containment. SIGMOD Record, Vol. 33, No. 1. ACM Press.

Schöning, Harald. 2003. Tamino - Software AG's Native XML Server. teoksessa XML Data Management. toimittanut Chaudhri, Rashid & Zicari. Addison Wesley. ISBN 0-201-84452-4.

Smith, Wayne D. 2001. TPC-W: Benchmarking An Ecommerce Solution. Technical White Paper, Transaction Processing Performance Council. Saatavissa http://www.tpc.org/tpcw/TPC-W_wh.pdf. viitattu 21.9.2005.

Staken, Kimbro. 2005. Xindice User Guide. saatavissa http://xml.apache.org/xindice/guide-user.pdf. viitattu 8.8.2005.

Tian, Feng, DeWitt, David J., Chen, Jianjun & Zhang, Chun. 2001. The Design and Performance Evaluation of Alternative XML Storage Strategies. Saatavissa http://www.cs.wisc.edu/niagara/papers/xmlstore.pdf. viitattu 26.7.2005.

Vanha-Vuori, Vuokko. 2005. Proseduraalinen ja rakenteinen dokumentti. Saatavissa http://myy.helia.fi/~vanvu/html/rdokumentti.html. viitattu 12.7.2005.

W3C. 1999. XML Path Language (XPath) version 1.0 W3C Recommendation. saatavissa: http://www.w3.org/TR/1999/REC-xpath-19991116. viitattu 22.8.2005.

W3C. 2005a. Extensible Markup Language. saatavissa http://www.w3.org/XML. viitattu 12.7.2005.

W3C. 2005b. XML Schema. saatavissa http://www.w3.org/XML/Schema. viitattu 12.7.2005.

W3C. 2005c. W3C Document Object Model. saatavissa http://www.w3.org/DOM. viitattu 5.8.2005.

W3C. 2005d. XML Query (XQuery). saatavissa http://www.w3.org/XML/Query/. Viitattu 24.8.2005.

Walkama, Pekka & Laakkonen, Aapo. 2004. Inside XML-skeema. Edita Prima Oy. Helsinki. ISBN 951-826-665-4.

Wang, Haixun, Park, Sanghyun, Fan Wei & Yu, Philip S. 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 Press. ISBN 1-58113-634-X.

Xmach-1. 2005. Xmach1: A Benchmark for XML Data Management. Saatavissa http://dbs.uni-leipzig.de/en/projekte/XML/XmlBenchmarking.html. viitattu 7.12.2005.

Yao, B. B., Özsu, M. T. & Khandelwal, N. 2004. XBench Benchmark and Performance Testing of XML DBMSs. Proceedings of 20th International Conference on Data Engineering. Boston, USA.

Zou, Qinghua, Liu, Shaorong & Chu, Wesley 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 Press. ISBN 1-58113-978-0.

LIITTEET

LIITE 1. Testidatan DTD:t

Hallittavan dokumentin DTD

```
<!ELEMENT document (title, chapter+)>
<!ATTLIST document
author CDATA #IMPLIED
doc_id ID #IMPLIED>
<!ELEMENT author (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT chapter (author?, head, section+)>
<!ATTLIST chapter
id ID #REQUIRED>
<!ELEMENT section (head, paragraph+, section*)>
<!ATTLIST section
id ID #REQUIRED>
<!ELEMENT head (#PCDATA)>
<!ELEMENT paragraph (#PCDATA | link)*>
<!ELEMENT link EMPTY>
<!ATTLIST link
xlink:type (simple) #FIXED "simple"
xlink:href CDATA #REQUIRED>
```

Hakemistodokumentin DTD

```
<!ELEMENT directory (host+) >
<!ELEMENT host (host+ | path+) >
<!ATTLIST host
name CDATA #REQUIRED >
<!ELEMENT path (path+ | doc_info) >
<!ATTLIST path
name CDATA #REQUIRED >
<!ELEMENT doc_info EMPTY >
<!ATTLIST doc_info
doc_id ID #REQUIRED
loader CDATA #REQUIRED
insert_time NMTOKEN #REQUIRED
update_time NMTOKEN #IMPLIED >
```

LIITE 2. Tietokantataulujen rakenteet

Binary Approach, hallittava dokumentti:

- Taulu nimetään elementin tai attribuutin nimen mukaan
- Samannimiset elementit tai attribuutit tallennetaan samaan tauluun

Field	Type	Null	Key	Default	Extra
id	int(10) unsigned		PRI		auto_increment
doc_id	varchar(20)	YES			
parent_element	varchar(30)	YES			
parent	int(10) unsigned	YES			
node_order	int(11)	YES			
name	varchar(40)	YES			
value	text	YES			

Edge Approach, hallittava dokumentti:

- XML-dokumentti tallennetaan kokonaisuudessaan yhteen tauluun
- Taulu nimetään DTD:n perusteella
- Saman DTD:n omaavat dokumentit tallennetaan samaan tauluun

Field	Type	Null	Key	Default	Extra
id	int(10) unsigned		PRI		auto_increment
doc_id	varchar(20)	YES			
parent	int(10) unsigned	YES			
node_order	int(11)	YES			
name	varchar(40)	YES			
value	text	YES			

Binary Approach ja Edge Approach, hakemistodokumentti:

- Molempien menetelmien hakemistodokumentin rakenne on sama

Field	Type	Null	Key	Default	Extra
id	int(10) unsigned		PRI		auto_increment
ahost	varchar(20)	YES			
bhost	varchar(20)	YES			
chost	varchar(20)	YES			
apath	varchar(20)	YES			
bpath	varchar(20)	YES			
cpath	varchar(20)	YES			
doc_name	varchar(30)	YES			
doc_id	varchar(20)	YES			
insert_time	varchar(18)	YES			
loader	varchar(15)	YES			
update_time	varchar(18)	YES			
doc_dtd	varchar(20)	YES			