Indeksit XML-tietokannoissa

Jan-Erik Löflund

Helsinki 29.10.2013

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta/Osasto – Fakultet/Sektion –Faculty/Section	Laitos – Institution – Department
Matemaattis-luonnontieteellinen tiedekunta	Tietojenkäsittelytieteen laitos

Tekijä – Författare – Author

Jan-Erik Löflund

Työn nimi – Arbetets titel – Title

Indeksit XML-tietokannoissa

Oppiaine – Läroämne – Subject

Tietojenkäsittelytiede

Työn laji – Arbetets art – Level	Aika – Datum – Month and year	Sivumäärä – Sidoantal – Number of pages
Pro gradu -tutkielma	29.10.2013	90 + 5 sivua

Tiivistelmä – Referat – Abstract

XML-tietomallin käyttö on yleistynyt mm. rakenteisissa dokumenteissa, verkkosovellusten toteuttamisessa ja Internetissä tapahtuvassa tiedonsiirrossa. Tämän myötä tarve XML-muotoisen tiedon pysyvään säilyttämiseen on kasvanut. Tähän tarkoitukseen on kehitetty XML-tietomallia tiedonsäilytys- ja käsittelymuotonaan käyttäviä XML-pohjaisia tietokantoja.

XML-muotoiset dokumentit ovat usein rakenteeltaan monimuotoisia ja kooltaan suuria. Tämän vuoksi XML-tietokannanhallintajärjestelmä on suunniteltava ja toteutettava tehokkaaksi, jotta sen avulla voidaan kohtuullisin laitteistoresurssein ja lyhyin vasteajoin suorittaa suuriakin määriä tietokantakyselyitä ja -päivityksiä, jotka voivat myös olla monipuolisia ja rinnakkaisia ja kohdistua suureen määrään tietoa kerrallaan.

Tässä työssä esitetään, miten XML-tietokannanhallintajärjestelmän suorituskykyä voidaan merkittävästi parantaa dokumenttien indeksoinnilla. Indeksoinnissa XML-dokumenttien elementeille luodaan yksikäsitteiset tunnisteet, joihin perustuen luodaan erilaisia indeksihakemistoja. Indeksoinnin avulla tieto voidaan tehokkaasti paikantaa tietokannan tietosivuilta ja siirtää tietokannanhallintajärjestelmän tietosivujen ja puskurin välillä, mikä nopeuttaa tietokannanhallintajärjestelmän toimintaa ja lisää sen kykyä käsitellä rinnakkaisia luku- ja kirjoituspyyntöjä. Indeksoinnin avulla voidaan myös tehostaa tietokannanhallintajärjestelmän kyselynkäsittelyalgoritmien toimintaa mahdollistamalla niiden käyttämien joukkoliitosoperaatioiden tehokas toteutus.

BaseX- ja eXist ovat XML-pohjaisia tietokannanhallintajärjestelmiä, joissa käytettävissä on useita erilaisia indeksejä. Indeksien toteutus näissä järjestelmissä kuvataan, ja näiden järjestelmien tehokkuutta XML-dokumentteihin tehtävien tietokantakyselyiden suorituksessa mitataan ja arvioidaan tätä varten kehitetyn XMark-koetinkuorman avulla.

ACM Computing Classification system (CSS):

Database management system engines

Database query processing

XML Query languages

Avainsanat – Nyckelord – Keywords

XML-tietokannat, indeksointi, XML-kyselykielet, tietokantakyselyiden käsittely

Säilytyspaikka – Förvaringsställe – Where deposited

Muita tietoja – Övriga uppgifter – Additional information

Sisältö

1 Johdanto

XML (Extensible Markup Language) on paljon käytetty tietomalli rakenteisten dokumenttien määrittelyssä ja Internetissä tapahtuvassa tiedonvaihdossa. Rakenteisen tiedon vaihtoon osallistuvan sovelluksen ei välttämättä tarvitse muuntaa tietoa tallennus- ja siirtomuodon välillä, mikäli tietomallina kummassakin on XML. XML-tietomalli sopii myös hyvin dokumenttien käsittelyyn, koska se mahdollistaa rakenteeltaan monipuoliset dokumentit. Lisäksi XML-tietomalliin kuuluva tietosisällön erottaminen esitysasusta mahdollistaa tietosisällön tehokkaan käsittelyn ja monikanavajulkaisemisen, sekä helpottaa tietoa käsittelevien sovellusten kehittämistä ja ylläpitoa [Guh06].

XML-tietomallin käytön yleistymisen myötä on myös lisääntynyt tarve tallettaa ja käsitellä XML-muotoista tietoa tietokannoissa. XML-tietokannoissa käsiteltävät tietomäärät ovat usein suuria ja tiedon rakenne monipuolinen [Sal01]. Esimerkiksi Internetin verkkosovelluksessa tällaisen tiedon käsittelyn on oltava nopeaa ja vasteaikojen lyhyitä, koska verkkopalveluiden käyttäjät eivät ole halukkaita käyttämään sovelluksia, joiden vasteajat ovat pitkiä [Soc09]. Kooltaan suurien ja tietomalliltaan monipuolisten dokumenttien käsittelyyn tarvitaan tehokkaita tietokannanhallintajärjestelmiä, jotka kykenevät käsittelemään suuriakin määriä tietokantaoperaatioita nopeasti ja kohtuullisin laitteistoresurssein.

Indeksoinnilla voidaan merkittävästi parantaa tietokannanhallintajärjestelmän suorituskykyä ylläpitämällä XML-dokumenttien elementtien yksikäsitteisiä tunnisteita sekä indeksihakemistoja, jotka perustuvat elementtien tunnisteisiin, sisältöön ja hierarkkiseen asemaan dokumentissa. Indeksoinnin avulla luku- ja kirjoitusoperaatioiden suorituksessa tarvittavat tiedot ovat tehokkaasti paikannettavissa tietokannan tietosivuilta ja siirrettävissä pysyvän muistin ja tietokantapuskurin välillä sopivan kokoisissa yksiköissä, jolloin sellaista tietoa, jota ei tarvita operaatioiden suorituksessa, tarvitsee siirtää vähemmän. Ilman indeksointia kokonaisia XML-dokumentteja olisi siirrettävä pysyvästä muistista käsiteltäviksi puskuriin ja niiden puskuroitu sisältö olisi selattava kokonaan. Tämä aiheuttaisi suuret tiedonsiirto- ja laskentakustannukset, tekisi kyselyiden suoritusajoista pitkiä, sekä käyttäisi runsaasti järjestelmän muistia. Indeksointi parantaa tieto-

kannanhallintajärjestelmän suorituskykyä myös mahdollistamalla tehokkaiden laskenta-algoritmien käytön tietokantakyselyiden suorituksessa, sillä elementtitunnisteet ovat paljon tehokkaammin käsiteltävissä laskenta-algoritmien käyttämissä välituloksissa ja tulosjoukkojen liitosoperaatioissa, kuin niiden indeksoimat elementit täydellisine tietosisältöineen [Wei10].

Indeksoinnilla voidaan myös parantaa luku- ja kirjoitusoperaatioiden rinnakkaista suoritettavuutta tietokannanhallintajärjestelmässä. Useiden tietokantakyselyiden rinnakkaisessa suorituksessa saattaa olla tarve yhtäaikaisesti lukea samoja dokumentin osia. Tämä ei aiheuta päivityskonflikteja, mutta tiedon yhtäaikainen lukeminen ja toimittaminen usealle lukupyynnölle ei onnistu, ja lukuoperaatiot joutuvat odottamaan vuoroaan. Indeksoinnin avulla lukuoperaatioiden pyytämät tiedot voidaan paikantaa ja siirtää niitä pyytävälle lukuoperaatiolle hienojakoisesti, mikä vähentää samaan tietoon kohdistuvia yhtäaikaisia lukupyyntöjä ja kasvattaa näin järjestelmän rinnakkaisuutta [Hau10].

Tiedon käsittelyn hienojakoisuus taas auttaa eristyvyyskonfliktien tehokkaassa välttämisessä. Eristyvyyskonfliktissa tietokantaoperaatioiden rinnakkainen suoritus aiheuttaa tietokantaan tilan, jollaista ei olisi syntynyt, mikäli operaatiot olisi suoritettu sarjallisesti. Eristyvyyskonfliktien välttämiseksi luku- ja kirjoitusoperaatiot pyytävät luku- ja kirjoituslukkoja dokumentin osiin, ja ristiriidassa jo myönnetyn lukon kanssa olevaa lukkoa tarvitseva operaatio odottaa lukon vapautumista. Indeksoinnin avulla kirjoitusoperaatioiden käsittelyn rinnakkaisuutta voidaan lisätä hienojakostamalla samalla tavoin kuin lukuoperaatioissa [Hau10].

Indeksien käytön kääntöpuolena on tarve päivittää indeksirakennetta vastaamaan tietosivujen päivityksissä muuttunutta tietosisältöä. Tätä varten tarvitaan keskusmuistissa tapahtuvaa laskentaa sekä tietojen siirtoa tietokantapuskurin ja pysyvän muistin välillä, mikä hidastaa tietokannanhallintajärjestelmän toimintaa. Lisäksi tietokantaindeksien ylläpitoa varten tarvitaan tallennustilaa [Wei10]. Indeksoitavat XML-dokumenttien rakenteet ja sisällöt kannattaa valita niin, että indekseistä saatu hyöty niiden ylläpitokustannuksiin verrattuna on mahdollisimman suuri.

Tässä työssä tarkastellaan erilaisia XML-dokumenteille kehitettyjä indeksityyppejä ja niiden toteutusta XML-tietokannanhallintajärjestelmän järjestelmäarkkitehtuurissa. Indeksien käytännön toteutusta tarkastellaan XML-pohjaisissa eXist- ja BaseX-tietokannanhallintajärjestelmissä, joiden suorituskykyä myös mitataan XMark-koetinkuorman avulla. Luvussa 2 käsitellään XML-tietokannoissa käytettävää XML-tietomallia ja XML-muotoisen tiedon käsittelyä varten kehitettyjä kyselykieliä kuten XQuery ja XPath. XML-tietomalliin perustuvien XML-tietokannanhallintajärjestelmien toteutusta ja niiden järjestelmäarkkitehtuurin perusteita käsitellään luvussa 3, ja XML-muotoisen tiedon paikantamisen, tietokannanhallintajärjestelmien komponenttien välillä siirtämisen ja laskentakäsittelyn tehostamista indeksien avulla puolestaan luvussa 4. Luvussa 5 esitellään teorian ja esimerkkien avulla vaihtoehtoisia tapoja suorittaa tietokantakysely. Kyselyn suoritus perustuu usein osakyselyihin, joiden tuloksia yhdistetään keskenään, kunnes saadaan kyselyn lopullinen tulos. Tässä luvussa myös näytetään, kuinka tietokannanhallinnanjärjestelmä tuottaa ja arvioi erilaisia käytettävissä olevia tietokantakyselyiden suoritustapoja ja kustannusarvioita näille, jotta se voi valita tehokkaimmaksi arvioimansa suunnitelman toteutusta varten. Tietokantaindeksien toteutusta ja määrittelymahdollisuuksia eXist- ja BaseX-tietokannanhallintajärjestelmissä kuvataan luvussa 6. XML-tietokannanhallintajärjestelmien suorituskyvyn mittaamiseen kehitetty XMark-koetinkuorma, joka koostuu huutokauppatietokantakuvauksesta, tietokantageneraattorista ja 20 tietokantakyselystä, esitellään luvussa 7. eXist- ja BaseX-järjestelmien suorituskyky mitataan luvussa 8 esitettävällä mittausmenetelmällä, ja tulokset esitetään ja arvioidaan luvussa 9. Yhteenveto mahdollisuuksista tehostaa XML-dokumenttien käsittelyä indeksien avulla on luvussa 10.

2 XML-tietokannanhallintajärjestelmän looginen tietomalli

XML-tietokannanhallintajärjestelmissä tallennetaan ja käsitellään tietoa XML-muodossa. XML on looginen tietomalli ja merkintäkieli, joka kuvaa kohteen rakenteisena, puumuotoisena elementeistä koostuvana dokumenttina. XML-tietokannanhallintajärjestelmässä tietojen käsittely tapahtuu käyttäen XML-dokumentteja varten suunniteltuja kysely- ja päivityskieliä, kuten XPath, XQuery ja XQuery Update Facility [Sal01].

XML-tietokannan looginen malli koostuu XML-dokumenttien lisäksi usein dokumenttien ryhmittelyistä kokoelmiksi, jotka voivat edelleen olla ryhmitelty hierarkkisiksi kokonaisuuksiksi. Malli muistuttaa käyttöjärjestelmien tiedostonhallinnan hakemistorakennetta [Sal01].

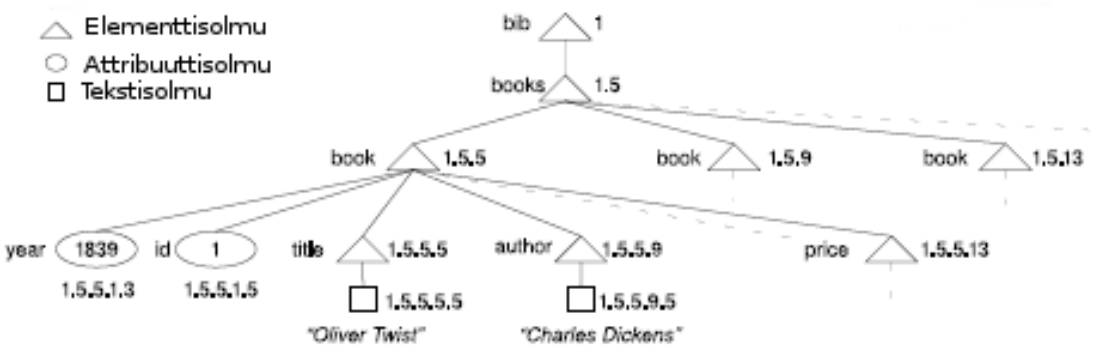
2.1 XML-tietomalli

XML dokumentit ovat rakenteisia ja voivat muodostaa rajattomasti sisäkkäisiä *solmujen* hierarkioita. XML-dokumentin esitysmuoto on tekstimuotoinen dokumentti, jossa dokumentti koostuu elementeistä, joilla on tunnus, sisältö ja mahdollisesti attribuutteja, joilla on nimi ja arvo. Tekstiesityksessä elementti eristetään ympäristöstään alku- ja loppumerkillä.

XML:n oikeellisuudelle on kaksi määritelmää: *hyvinmuodostuneisuus* ja *validius* [Gri11]. Hyvinmuodostetun dokumentin tulee täyttää vähintään seuraavat vaatimukset:

-dokumentissa on yksi ja vain yksi juurielementti,
-ei-tyhjillä elementeillä on sekä alku- että loppumerkki,
-jokainen attribuutti on lainausmerkkien sisällä ja
-elementit suljetaan aina ennen niiden vanhempia.

Validi XML-dokumentti taas on jonkin määritellyn dokumenttityypin mukainen. Kuvassa 1 esitetään XML-esimerkkidokumentti.




```
<?xml version="1.0"?>
<bib>
   <books>
      <book year=”1839” id=”1”>
         <title>Oliver Twist</title>
         <author>Charles Dickens</author>
         <price>20.00</price>
      </book>
   </books>
...
</bib>
```

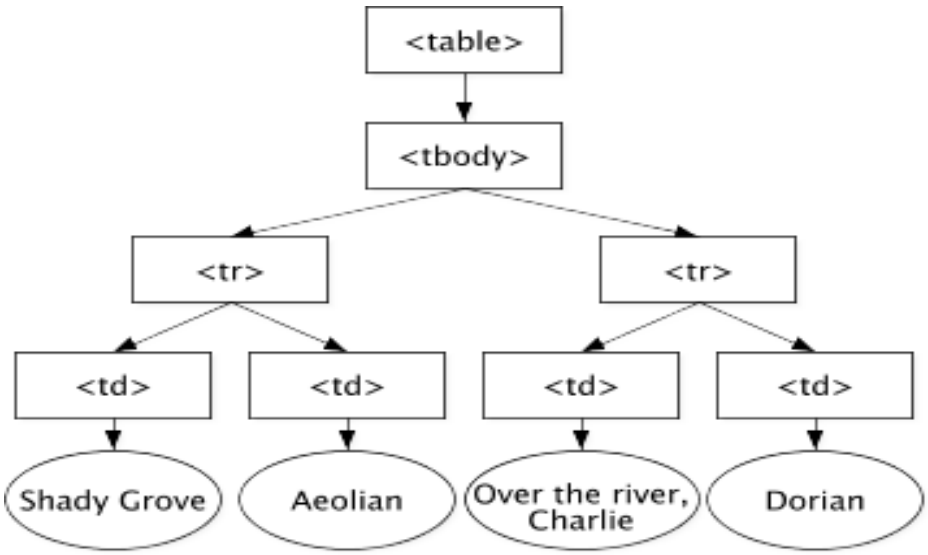
Kuva 1: XML-esimerkkidokumentti puu- ja tekstimuodossa (muokattu lähteestä [Wei10]).

XML-merkintäkielen tapa kuvata tiedon merkitys tiedon yhteyteen helpottaa sovelluslogiikan ja esityskomponenttien toisistaan riippumatonta kehitystä ja ylläpitoa. Sovellus- ja esityslogiikan erottaminen toisistaan helpottaa myös tiedon monikanavajulkaisemista esim. mahdollistamalla saman tietosisällön esittämisen sekä HTML-sivuna että PDF-muotoisena dokumenttina. XML-tietomallin mahdollistama rakenteeltaan monimutkaisen tiedon kuvaus myös parantaa tiedon pitkäaikaissäilyvyyttä ja helpottaa järjestelmien integrointia [Guh06].

XML-tietomallin käytön yleisyys mahdollistaa yhdenmukaisten tietosisältöjen kuvauksien kehittämisen helpottaen näin tiedonhakua ja -siirtoa [Zen13]. Erilaisia määrittelyjä, kuvauskieliä ja sovelluksia XML-muotoisen tiedon käsittelyyn on kehitetty runsaasti, minkä ansiosta riippuvuus yksittäisistä ohjelmistotoimittajista vähenee. Määrittelyiden ja sovellusten suuri määrä, järjestelmien integroitavuus, tiedon rakenteen ja esitystavan erottaminen ja luettavuus sekä ihmiselle että automaattisille järjestelmille auttavat kaikki myös välttämään sisältövirheitä [Guh06].

2.2 DOM-tietomalli

DOM (Document Object Model) on World Wide Web -yhteenliittymän (W3C) määrittelemä alusta- ja ohjelmointikielineutraali ohjelmointirajapinta HTML- ja XML-muotoisten dokumenttien selaamiseen ja päivittämiseen. DOM perustuu oliohierarkiarakenteeseen, joka läheisesti muistuttaa sillä mallinnettavaa dokumenttia [W3Ca]. DOM-puun olioiden osoittaminen ja käsittely perustuu olioiden tunnisteisiin ja sukulaissuhteisiin, ja tapahtuu olioiden metodien avulla. Kuvassa 2 esitetään yksinkertainen HTML-muotoinen XML-dokumentti DOM-puuna.



Kuva 2: HTML-taulukon kuvaus DOM-puuna [W3Ca].

Esimerkiksi verkkosovelluksissa paljon käytetty JavaScript-ohjelmointikieli käsittelee HTML-merkintäkielellä määriteltyjen verkkosivujen sisältöä DOM-rajapinnan välityksellä [Jen11]. Kuvan 2 kuvaaman HTML-taulukon solujen sisältö voidaan käsitellä esimerkin 1 JavaScript-kielisellä ohjelmalla, joka kutsuu HTML-taulukon sisältävää verkkosivua kuvaavan `document`-olion `getElementsByTagName`−metodia, joka palauttaa olion, joka sisältää taulukkona viitteet sivun `td`-nimisiin elementteihin. Tämän jälkeen ohjelma selaa palautteena saamansa taulukon läpi ja noutaa tulostettavaksi taulukkosolujen sisällön näiden `innerHTML`-metodia kutsumalla.

```
<script type=”text/javascript”>
var x= document.getElementsByTagName(”TD”);
for(i=0;i<x.length;i++)
{document.write(x[i].innerHTML); document.write("<br/>");}
</script>
```

Esimerkki 1: DOM-rajapintaa käyttävä JavaScript-ohjelma.

2.3 XPath-kyselykieli

XPath on W3C:n määrittelemä kyselykieli, jolla valitaan XML-dokumentista ehdot täyttäviä solmuja ja luodaan XML-dokumentin rakenteeseen perustuvaa tietoa. XPath-kieli perustuu XML-dokumentin puumuotoiseen esitystapaan, ja mahdollistaa dokumentin osien valitsemisen tietyillä valintakriteereillä. XPath-kielestä on kaksi versiota: aikaisempi versio 1.0 ja uudempi versio 2.0, joka perustuu XQuery 1.0 -ja XPath 2.0 -tietomalliin (XDM).

XPath on ei-XML-pohjainen ja perustuu lausekkeisiin, joista yleisin on polkulauseke. Polkulauseke koostuu peräkkäisistä askeleista, joilla päästään XML-solmusta yhteen tai useampaan toiseen solmuun. Askeleet erotetaan toisistaan vinoviivalla (/). Jokainen askel koostuu kolmesta osasta: akselimäärittely, solmutesti ja predikaatti. XPathin polkulausekkeet ja akselimäärittelyt muistuttavat suuresti Unix-käyttöjärjestelmän hakemistopolkumäärittelyä ja URL-määrittelyä, joka kertoo ensisijaisen tavan paikantaa verkkore-

surssi. XPath-määrittelystä on kaksi muotoa: suppea ja yleinen muoto. Suppea muoto on tiiviimpi ja jonkin verran rajatumpi kuin yleinen muoto. Tässä työssä esitysmuotona on suppea muoto [W3Ca].

Akselimääritykset ilmaisevat liikkumissuuntaa XML-dokumentin puumaisessa esitystavassa. Akselimäärittelyt suppeassa syntaksissa ovat lapsi (oletus), attribuutti ”`@`”, jälkeläinen ”`//`”, vanhempi ”`..`” ja solmu itse ”`.`”.

Solmutestit voivat koostua solmun nimen testauksesta tai yleisemmistä lausekkeista. Esim. `comment()` määrittelee XML-kommenttisolmun kuten `<!-- Comment -->` ja `text()` tekstisolmun kuten `Hamlet` elementissä `<play>Hamlet</play>`.

Predikaatit kirjoitetaan lausekkeina hakasulkujen sisällä. Predikaateilla rajoitetaan XPath-lausekkeen askelien määrittelemää solmujoukkoa annettujen predikaattien ehdot täyttäviin. Esimerkiksi lausekkeessa `//book[name='Charles Dickens']` määritellään dokumentista kaikki `book`-nimiset elementit, joilla on lapsielementti, jonka nimi on `name` ja sisältönä `Charles Dickens`. Predikaatteja voi olla rajaton määrä, ja niitä voidaan asettaa kaikille kyselyn akselissa oleville solmuille.

XPath 2.0 -kielen käyttämä XDM-tietomalli perustuu 21 primitiivitietotyyppiin sekä näiden avulla johdettuihin 25 valmiiseen johdettuun tietotyyppiin. Käyttäjät voivat myös primitiivisten ja valmiiksi johdettujen tietotyyppien avulla edelleen määritellä myös uusia tietotyyppejä [W3Ce].

Operaattorit	Toiminta
+, -, *, div, mod, idiv	Aritmeettisia operaattoreita
=, !=, <, >, <=, >=	Yleisiä vertailuoperaattoreita
and, or	Totuusarvo-operaattoreita
is	Solmuidentiteetin vertailuoperaattori
<<, >>	Solmujen sijainnin vertailu dokumentin järjestykseen perustuen
union, intersect, except	Solmujen käsittelyn joukko-opillisia operaattoreita
eq, ne, lt, gt, le, ge	Arvojen vertailuoperaattoreita
to	Kokonaislukujen arvovälin määrittelyyn käytettävä operaattori
instance of	Arvon määritellyn tietotyypin mukaisuuden testaus
cast as	Arvon muuntaminen määritellyn tietotyypin mukaiseksi.
castable as	Arvon määritellyn tietotyypin mukaiseksi muunnettavuuden testaus

Taulukko 1: XPath 2.0 -kielen operaattorit.

XPath-kielen version 2.0 operaattorit esitetään taulukossa 1. Kolmella ensimmäisellä rivillä esitetyt operaattorit sisältyvät myös versioon 1.0, muut ovat versiossa 2.0 tehtyjä lisäyksiä.

2.4 XQuery-kieli

XQuery on funktionaalinen tietokantakysely- ja ohjelmointikieli, joka on tarkoitettu kyselyiden tekemiseen XML-dokumenteista ja XML-dokumenttien tuottamiseen [Koh06]. XQuery-kielen käyttöalueita ovat mm. tiedon hakeminen tietokannasta verkkopalvelujen käyttöön, verkkopalvelujen tuottaminen, yhteenvetojen tuottaminen XML-dokumenteista ja XML-dokumenttien muuntaminen muodosta toiseen samaan tapaan kuin XSLT-merkintäkielellä määriteltäviä tyylimuunnoksia käyttämällä. XQuery on XPath 2.0 -kielen laajennus ja perustuu tämän tavoin XDM-tietomalliin [W3Cb].

```
<html><head/><body>
{
  for $act in doc("hamlet.xml")//ACT
  let $speakers := distinct-values($act//SPEAKER)
  return
    <div>
      <h1>{ string($act/TITLE) }</h1>
      <ul>
      {
        for $speaker in $speakers
        return <li>{ $speaker }</li>
      }
      </ul>
    </div>
}
</body></html>
```

Esimerkki 2: XQuery-kielinen HTML-dokumentin määrittely.

XQuery perustuu lausekkeisiin ja käyttää XPath-määrittelyä XML-dokumenttien osia paikantavien polkujen kuvaamiseen. XPath-määrittelyn lisäksi kieli perustuu SQL-tyyppisiin ”FLWOR”-lausekkeisiin. FLWOR on lyhenne XQuery-kielen lauseiden `FOR`, `LET`, `WHERE`, `ORDER BY` ja `RETURN` nimistä [W3Cb]. Koska XQuery perustuu lausekkeisiin, ei siinä käytetä lainkaan muuttujia [Koh06].

Esimerkissä 2 määritellään XQuery-kielellä verkkosivun määrittelevä HTML-dokumentti käyttämällä kolmea sisäkkäistä silmukkaa. Solmujen sekvenssejä haetaan `for`-lauseella, ja `let`-lauseella taas sidotaan sekvenssi em. `for`-lauseessa tehtyyn määrittelyyn. `Return`-lause evaluoidaan erikseen jokaiselle sisimmän silmukan palauttamalle solmulle, ja se palauttaa merkkijonon elementin sisällä. Kullakin silmukan osalla on oma kontekstinsa, ja osista voidaan viitata XPath-lausekkeiden ja ulompiin `for`- ja `let`-lauseisiin tehtävän sidonnan avulla muualle dokumenttiin.

2.5 XQuery Update Facility

XQuery Update Facility on XML-dokumenttien päivitykseen tarkoitettu XQuery-kielen laajennus. Se sisältää lausekkeita, joilla voidaan tehdä muutoksia XQuery 1.0 - ja XPath 2.0 -tietomallin mukaisiin XML-dokumentteihin [Jun12], [W3Cd].

XQuery Update -lauseiden avulla XML-dokumenttiin voidaan lisätä solmuja, siitä voidaan poistaa solmuja, solmujen sisältöä voidaan päivittää säilyttäen solmun solmuidentiteetti ja solmusta voidaan luoda kopio uudella solmuidentiteetillä [W3Cd]. Esimerkeissä 3 ja 4 lisätään ja poistetaan XML-dokumentin solmu.

```
Insert node <year>2005</year> after
fn:doc("bib.xml")/books/book[1]/publisher
```

Esimerkki 3: solmun lisäys XML-dokumenttiin.

```
let $user := doc("users.xml")/users/user_tuple[name="Dee Linquent"]
return
  delete nodes $user
```

Esimerkki 4: solmun poisto XML-dokumentista.

3 XML-tietokannanhallintajärjestelmän toteutus

Luvussa 2 esitellyt XML-tietomalli ja XML-kyselykielet XPath ja XQuery ovat pääosin deklaratiivisia; niissä tietokannan tietomalli ja tiedonkäsittelyoperaatiot määritellään loogisella tasolla, mutta ei oteta kantaa tiedon tallennuksen tai operaatioiden toteutustapaan. Näiden toteutus määritellään tietokannan fyysisessä tietomallissa, johon kuuluvat mm. tiedostojärjestelmä, tietokantakyselyiden käsittely, indeksointi, tietokantaloki ja tietokannan palautus. Tietokannanhallintajärjestelmä huolehtii fyysiseen malliin kuuluvien osien toteutuksesta, eivätkä ne sellaisenaan näy käyttäjälle. XML-tietokannanhallintajärjestelmän järjestelmäarkkitehtuuri perustuu siis jakoon, jossa sovellusohjelmoijalle tarjotaan looginen rajapinta tiedon käsittelyyn ja tietokannanhallintajärjestelmän fyysinen malli määrittelee loogisten operaatioiden suorituksen toteutuksen.

XML-tietokannanhallintajärjestelmät voidaan niiden käyttämän tiedon tallennusmuodon perusteella jaotella *XML-tietomallia tukeviin* ja *XML-pohjaisiin*. XML-tietomallia tukevat tietokannanhallintajärjestelmät käsittelevät XML-muotoista tietoa toteuttamalla tallennuksen käyttäen jotain muuta tietorakennetta kun XML-dokumenttia. Tällaisia tietorakenteita voivat olla esim. relaatiotietokannan rivit, joiden sarakkeisiin XML-muotoiset dokumentit tallennetaan merkkijonoiksi muutettuina, joukko XML-skeeman tai dokumentin rakenteen mukaan muodostettuja relaatioita tai relaatiotietokannan rivit, joiden sarakkeiden tietotyypiksi on mahdollista valita XML. Sarakkeisiin merkkijonomuotoisena talletettavan XML-tiedon ongelmana on siihen kohdistuvien kyselyiden toteuttamisen tehottomuus, kun taas dokumentin rakenteen mukaan muodostettuja relaatioita käytettäessä tietokannan taulurakennetta voidaan joutua muuttamaan dokumentin rakenteen muuttuessa [Sal01].

XML-pohjaisessa tietokannanhallintajärjestelmässä XML-dokumentti taas on sekä tiedon tallennuksen että käsittelyn perusyksikkö. Tällöin tiedon tallennuksen ja käsittelyn tulee tukea ainakin elementtejä, attribuutteja, Parsed Character Data (PCDATA) -määrittelyä ja dokumentin järjesteisyyttä. XML-pohjaisessa tietokannassa dokumentteja ei välttämättä talleteta suoraan tekstimuodossa, vaan riittää, että

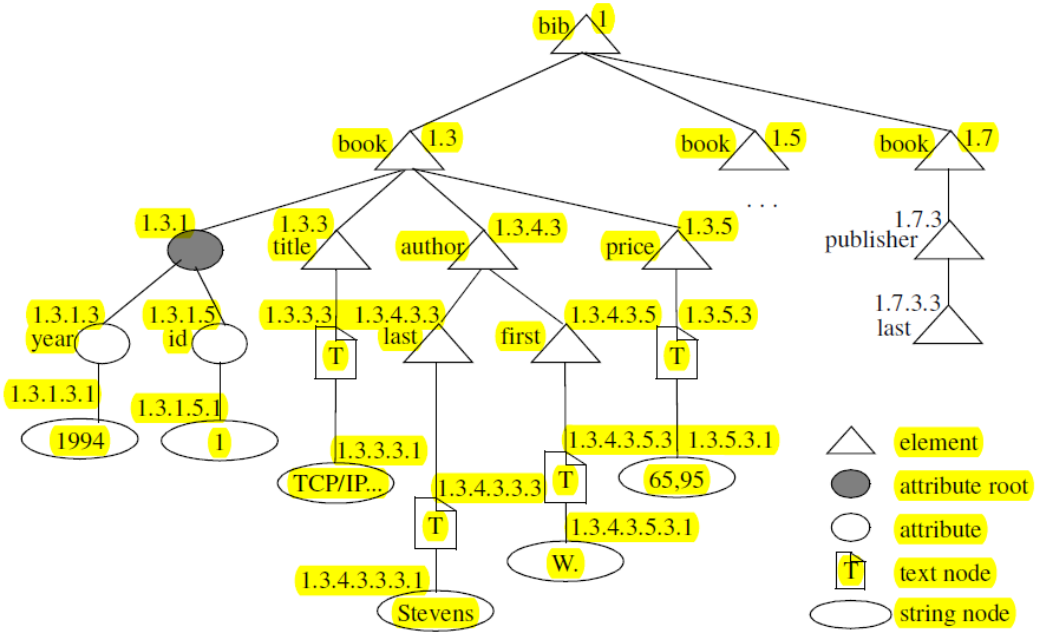
tietokannanhallintajärjestelmä kykenee muuttamaan tiedon tallennusmuodosta XML-dokumentiksi ja toisinpäin [Sal01].

XML-pohjaisia tietokantoja on alettu kehittää 2000-luvulla, joten esimerkiksi relaatiotietokantoihin verrattuna ne ovat varsin uusi keksintö ja niiden käyttö on vasta yleistymässä. XML-pohjaisia tietokantoja ovat mm. vapaaseen ohjelmistolisenssiin perustuvat BaseX, eXist, Sedna ja XTC [Base], [Mei02], [Tar10], [Wei10]. Näissä tietokannanhallintajärjestelmissä tieto säilytetään dokumenteista koostuvissa kokoelmissa ja tiedonkäsittelykielenä käytetään XPathia, XQueryä ja XQuery Update Facilityä.

3.1 Dewey-indeksointi

XML-tietokannanhallintajärjestelmän tehokkaan toteutuksen mahdollistamiseksi tarvitaan menetelmä, jolla XML-dokumenttien solmut voidaan merkitä ja linkittää fyysisen ja loogisen tietomallin välillä. Useita erilaisia merkintämenetelmiä on esitetty kirjallisuudessa. Menetelmältä on voitava vaatia ainakin lukitusprotokollan lukitsemien solmujen merkinnän muuttumattomuus niin kauan, kuin niihin on varattu lukkoja. Lisäksi merkintämenetelmän tulee säilyttää XML-dokumentin elementtien järjestys lisättäessä uusia solmuja dokumenttiin. Merkintätavan tulee myös mahdollistaa elementtien tason ja edeltäjäsolmujen helppo havaitseminen [Hau10].

Hausteinin ja kumppanien mukaan Dewey-indeksointi sopii XML-tietokannan solmujen indeksointiin, koska se täyttää kaikki edellä mainitut kriteerit [Hau10]. Dewey-indeksointia käytetään sekä Hausteinin ja kumppanien kehittämässä XTC-tietokannanhallintajärjestelmässä että eXist-tietokannanhallintajärjestelmässä [EX], [Hau10]. Dewey-indeksoinnin merkintätavasta käytetään myös nimitystä ”dynamic level numbering” [EX].



Kuva 3: Dewey-indeksoitu XML-dokumentti [Hau10].

Dewey-indeksoinnissa dokumentin elementeille ja attribuuteille luodaan yksikäsitteiset elementtitunnisteet. Dewey-indeksointi perustuu dokumentin kokonaisluvuilla kuvattaviin elementteihin ja pisteillä kuvattaviin jakoihin. Dokumentin juurielementti saa aina arvon 1 ja kukin lapsielementti saa vanhempansa arvon, jaon ja lapsielementtiä kuvaavan luvun, jonka arvo kasvaa järjestyksessä vasemmalta oikealle. Esimerkiksi kuvan 3 dokumentin elementit indeksoinneilla 1.3, 1.5 ja 1.7 ovat elementin 1 lapsielementtejä, pisteen jälkeisen luvun määritellessä näiden keskinäisen järjestyksen. Elementin järjestys, taso ja edeltäjäelementit on suoraan kuvattu sen merkinnässä. Merkintä mahdollistaa myös uusien elementtien lisäämisen olemassa oleva numerointi ja järjestys säilyttäen [Hau10]. Kuvan 3 esitystapa poikkeaa rakenteeltaan samanlaista dokumenttia kuvaavan kuvan 1 esitystavasta, sillä kuvassa 3 myös elementtien tietosisällöt on indeksoitu.

Jotta elementtitunnisteita voidaan lisätä dokumentin indeksointiin myöhemminkin, jätetään indeksoinnissa osa mahdollisista numeroinneista käyttämättä mahdollisia myöhempiä lisäyksiä varten. Käyttämättä jätettävien numerointien määrä määritellään esimerkiksi muuttujan `distance` avulla. Mitä suurempi on tämän muuttujan arvo, sitä enemmän lisäyksiä kahden vierekkäisen elementin indeksointien väliin voidaan tehdä käyttämättömän numeroinnin loppumatta kesken. Esimerkiksi kun muuttujan arvo on 8, saavat dokumentin juurielementin kolme lasta XTC-järjestelmässä aluksi elementtitunnisteet 1.9, 1.17 ja 1.25, koska XTC:ssä ensimmäinen lapsi indeksoidaan aina tunnisteella, joka saa arvon `distance+1`. Joissakin järjestelmissä, kuten eXist, parillisetkin arvot ovat käytössä, jolloin esimerkin elementtien väleihin on mahdollista tehdä seitsemän lisäystä. XTC-tietokannanhallintajärjestelmässä taas parilliset numerot on varattu jakojen ylivuotomekanismia varten. Tällä tavoin kuvan 3 dokumentin Dewey-tunnisteilla 1.3.3 ja 1.3.5 indeksoitujen elementtien väliin lisättävä sisarelementti saa tunnisteen 1.3.4.3 parillisen luvun osoittaessa, että kyseessä on ylivuotoa merkitsevä jako, joka kuvaa, että elementti 1.3.4.3 on elementin 1.3 lapsi ja elementtien 1.3.3 ja 1.3.5 sisarelementti, joka sijaitsee näiden välissä. Käyttämällä XTC-tietokannanhallintajärjestelmän ylivuotonumerointitapaa, ainoastaan numerointiin käytettävän tietorakenteen koko rajoittaa indeksointien lisäyksiä, ja dokumentti on indeksoitava uudestaan vain silloin, kun jonkin elementtitunnisteen vaatima tila ylittäisi tähän varatun tietorakenteen koon. Ylivuotomekanismia käyttäen tehdyt lisäykset indeksointiin kuluttavat kuitenkin enemmän tallennustilaa, ja ovat työläämpiä käsitellä laskennassa kuin normaalilla tavalla indeksoitujen elementtien väliseen vapaaseen tilaan lisätyt indeksoinnit [Hau10].

Dewey-indeksointi tukee tehokkaasti myös DOM-mallin mukaista tiedonkäsittelyä kuvaamalla dokumentin aivan samalla tavalla hierarkkisena puurakenteena kuin DOM-malli. Lisäksi Dewey-indeksoinnin tapa kuvata elementit ja niiden välinen hierarkia kokonaisluvuilla ja pisteillä tukee DOM-mallille keskeistä dokumenttirakenteessa navigoimista, sillä Dewey-indeksointi kuvaa elementtien edeltäjäelementit, sisarelementit ja lapsielementit tukien näin DOM-mallin mukaisten tiedonkäsittelyoperaatioiden linkittämistä XML-dokumentteihin [Hau10].

Dewey-indeksoinnissa on tärkeää optimoida indeksoitavien elementtien lisättävyyden ja tilankäytön välinen suhde. Käyttämättä jätettävien numerointien jättäminen elementtitunnisteiden väliin dokumenttia tietokantaan ladattaessa parantaa indeksointien lisättävyyttä, mutta tallennustilan kulutuksen kustannuksella. Mitä suurempi muuttujan `distance` arvo on, sitä enemmän tietokannan Dewey-indeksoinnin ylläpitoon tarvitaan tallennustilaa [Hau10].

XML-dokumentin Dewey-indeksointi suoritetaan XML-dokumentin tietokantaan lataamisen yhteydessä. Koko dokumentti on indeksoitava uudelleen tilanteessa, jossa lisättävän elementin elementtitunnisteen pituus ylittäisi tunnisteelle varatun tilan. Uudelleenindeksointi voidaan suorittaa joko pitäen järjestelmä käytettävissä indeksoinnin ajan, tai keskeyttämällä palvelupyyntöjen käsittely uudelleenindeksoinnin ajaksi [Soc09].

3.2 Tiedostojärjestelmä

XML-tietokannan fyysisessä mallissa tiedon käsittelyn perusyksikkönä on yleensä tietokannanhallintajärjestelmän tietosivu. Tieto voidaan säilyttää tietosivuilla dokumentin rakenteen mukaan järjestettynä ja Dewey-indeksoituna elementtien kasvavassa numerointijärjestyksessä, esim. muodossa

```
|1_bib|1.3_book|1.3.1.3_year|1.3.1.3.1_1900|
```

[Wei10], [EX2]. Toinen mahdollinen tapa säilyttää tieto tietosivuilla perustuu nimeltään ja hierarkkiselta asemaltaan yhtenevien elementtien ryvästykseen samoille tietosivuille, esim. muodossa

```
|1_bib|1.3_book|1.5_book|1.7_book|
```

[Tar10]. Edellinen tapa on tietokantahaun suorituksessa tehokkaampi, jos elementtien tietoja haetaan lapsielementteineen, jälkimmäinen taas siinä tapauksessa, että lapsielementtien tietoja ei tarvitse hakea.

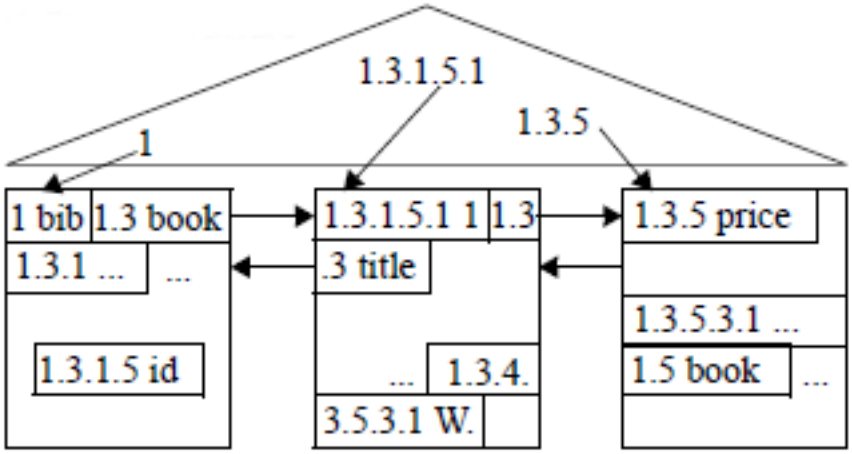
Tietosivut talletetaan tietokannan pysyvään muistiin, josta niitä tarvittaessa noudetaan tietokannanhallintajärjestelmän puskuriin käsittelyä varten. Muuttuneet sivut taas päivitetään aika-ajoin puskurista pysyvään muistiin.

Tietokannanhallintajärjestelmän puskuri on keskusmuistissa ylläpidettävä tietorakenne, johon voidaan tuoda ja jossa voidaan käsitellä tietosivuja. Tietokantajärjestelmä toteuttaa luku- ja kirjoitusoperaatiot käsittelemällä puskurissa sijaitsevia tietosivuja, ja noutaa tarvittaessa pysyvästä muistista puskuriin sellaisia tietokantakyselyn suorituksessa tarvittavia sivuja, joita siellä ei vielä ole. Puskurissa tietosivu saattaa päivittyä, jolloin tehdyn päivityksen tiedot viedään myös tietokannanhallintajärjestelmän pysyvässä muistissa olevaan lokiin. Aika-ajoin puskurissa päivitettyjen tietosivujen pysyvässä muistissa olevia versioita päivitetään vastaamaan puskuriversiota. Puskurissa oleva tietosivu saattaa siis välillä olla tietosisällöltään uudempi kuin pysyvässä muistissa oleva. Kun tietosivua ei enää tarvitse käsitellä, sitä ei muistin riittäessä välttämättä tarvitse poistaa puskurista, vaan se voidaan jättää sinne. Näin sivun tiedot ovat tallennusjärjestelmästä tehtävää sivujen noutoa nopeammin saatavissa, jos niiden sisältöä on tarvetta lukea tai päivittää myöhemmin [Hel94].

```
<db-connection cacheSize="48M" collectionCache="24M" database="native"
files="webapp/WEB-INF/data" pageSize="4096" nodesBuffer="-1">
```

Esimerkki 5: eXist-tietokannanhallintajärjestelmän oletusarvoiset tietosivu- ja tietopuskurimääritykset tiedostossa `conf.xml`.

eXist-tietokannanhallintajärjestelmän käyttämät tietosivujen ja tietokantapuskurin oletusasetukset esitetään esimerkissä 5. Oletusasetuksissa puskurin koko on määritelty 48 megatavuksi ja tietosivun koko 4096 tavuksi. Puskurin koon oletusarvo sopii hyvin pienemmälle tietokannalle, mutta suuressa, esim. satojen megatavujen kokoisessa, tietokannassa tämä on liian vähän, koska vapaan puskuritilan loppuessa tietosivuja on puskuritilan vapauttamiseksi poistettava puskurista sen sijaan, että ne voitaisiin jättää sinne odottamaan tilannetta, jossa niitä tarvitaan uudelleen [EX4].



Kuva 4: dokumentin rakenteeseen perustuva tietosivujen indeksi- ja tallennusrakenne [Hau10].

Dewey-indeksointi ja tiedon tallennus tietosivuille Dewey-tunnisteiden mukaisessa järjestyksessä mahdollistaa kuvan 4 mukaisen dokumentin hierarkkiseen rakenteeseen perustuvan indeksoinnin, jotka perustuu B+ -puiden käyttöön. Dokumentin rakenteeseen perustuvassa indeksissä säilytetään monikoita, joissa avaimena on elementtitunniste ja osoitteena suora osoite siihen tietosivuun, jolla avaimen indeksoiman elementin tietoja säilytetään.

Rakenneindeksin avulla voidaan toteuttaa myös elementtien nimiin ja tietosisältöön perustuvia indeksejä, joissa ylläpidetään monikoita, joiden avaimena on esim. elementin nimi, tietosisältö tai polkurakenne. Tällöin tietosisältönä on elementin epäsuora osoite. Epäsuora osoite on elementtitunniste, jonka perusteella elementin sijainti tietosivuilla on haettavissa rakenneindeksistä. Elementtien nimiin ja tietosisältöön perustuvissa indekseissä on mahdollista käyttää myös suoria osoitteita, mutta tässä vaihtoehdossa indeksin ylläpitoon tarvitaan enemmän tilaa ja tietokannan päivitykset ovat hitaampia, koska tie-

to elementin fyysisestä tietosivuosoitteesta on ylläpidettävä ja päivitettävä erikseen kuhunkin elementin nimeen tai sisältöön perustuvaan indeksiin.

Ylläpitämällä indeksejä, joiden avaimet vastaavat tietokantakyselyissä käytettäviä polkumäärittelyjä ja predikaattitestejä, voidaan tehokkaasti paikantaa ja siirtää puskuriin kyselyn suorituksessa tarvittavat tietosivut. Hakemistojen käytön tehokkuus perustuu niiden koko tietokannan tietosisältöä pienempään kokoon ja siihen, että puurakenteen ansiosta itse indeksistäkään ei tarvitse yleensä noutaa kovin montaa tietosivua tallennusjärjestelmästä puskuriin käsittelyä varten [Wei10].

Indeksointi toimii tehokkaimmin tietosivujen ollessa tarpeeksi pienikokoisia. Tällöin tieto on hienojakoisesti paikannettavissa, ja tallennusjärjestelmän ja puskurin välillä siirrettävä tietomäärä pysyy mahdollisimman pienenä. Lisäksi tallennusjärjestelmän ja puskurin välillä siirrettävän tiedon määrän minimointi vähentää laskentakustannuksia ja ehkäisee tietokantapuskurin täyttymistä. Pitkälle viety hienojakoisuus tietosivujen käsittelyssä auttaa myös vähentämään luku- ja kirjoitusoperaatioiden yhteydessä lukittavien tietosivujen määrää, mikä taas lisää tietokannanhallintajärjestelmän rinnakkaisuutta vähentäen tilanteita, joissa transaktiot joutuvat odottamaan toisten transaktioiden suorituksessa tarvittavien lukkojen vapautumista [Hau10].

Elementtien ja indeksien tallennuksessa on tärkeää käyttää levytilaa tehokkaasti; tietosivujen korkea täyttöaste vähentää sekä säilytystilan tarvetta että säilytysjärjestelmän ja tietokantapuskurin välillä siirrettävien tietosivujen määrää. Samalla tavoin kuin luvussa 3.1 esitellyssä tietokannan latauksen yhteydessä sovellettavassa elementtitunnisteiden numeroinnissa, myös tietosivun optimaalisen täyttöasteen määrittelyssä on otettava huomioon tietokantaa käyttävien sovellusten luonne. Sovelluksissa, joissa tietoa enimmäkseen luetaan, sivujen korkea täyttöaste on edullinen, koska tällöin tieto siirtyy mahdollisimman tehokkaasti pysyvän muistin ja tietokantapuskurin välillä. Sen sijaan tietoa usein päivittävissä sovelluksissa matalampi sivujen täyttöaste mahdollistaa tietokannan tehokkaan päivitettävyyden ja lyhyet vasteajat vähentämällä tietokannan käyttöä

hidastavia tilanteita, joissa tietosivu täyttyy ja tietosivujakoa ja indeksisivuja joudutaan laskemaan uudelleen [Mor12].

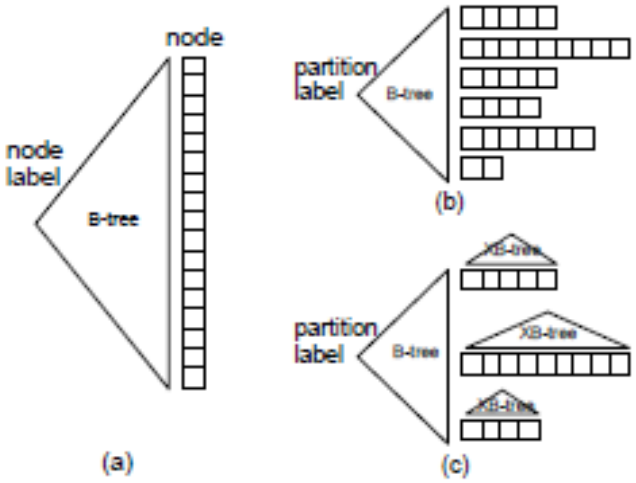
3.3 Tietosivujen ja laskenta-algoritmien rajapinta

XML-tietokannanhallintajärjestelmässä tietokantakyselyiden suoritus perustuu kahteen loogiseen kerrokseen: *tiedonsiirto* pysyvän muistin ja puskurin välillä puskuroitujen tietosivujen sisällön *laskentakäsittely* järjestelmämuistissa. Tietosivujen sisällön käsittely järjestelmämuistissa ei kohdistu suoraan puskuroituihin tietosivuihin, vaan esimerkiksi eXist-tietokannanhallintajärjestelmässä dokumenttien käsittelyn yksikkönä muistissa on DOM-puu, johon tiedot ladataan tietosivuilta [EX3]. Vastaavasti tietokannanhallintajärjestelmä huolehtii DOM-puussa tapahtuneiden muutoksien viennistä puskuroiduille tietosivuille [EX3].

4 XML-dokumentin indeksointitapoja

XML-dokumentin indeksointi ei ole osa XML-tietokantojen loogista mallia, vaan se liittyy XML-tietokannanhallintajärjestelmän toteutukseen. XML-dokumenttien ja niiden muodostamien kokoelmien indeksoinnille ei ole standardia, ja toteutus vaihtelee järjestelmäkohtaisesti. Indeksejä voidaan joko jättää vain tietokannanhallintajärjestelmälle näkyväksi osaksi, tai sovellusohjelmoijalle voidaan antaa mahdollisuus ottaa valinnaisia indeksejä käyttöön.

Indeksejä on kahta perustyyppiä: dokumentin rakenteeseen perustuva rakenneindeksi ja ositukseen perustuva ositusindeksi. Rakenneindeksissä avaimena on elementtitunniste (kuva 5a) ja tietosisältönä viite tietosivuun, jolla indeksoidun elementin tietoja säilytetään. Rakenneindeksin avulla voidaan johtaa dokumentin sisältöön ja rakenteeseen perustuvia indeksejä, joissa avaimena on elementin sisältö, hierarkkinen asema tai näiden yhdistelmä, ja tietosisältönä elementtitunniste, jonka avulla tietosivuja voidaan paikantaa epäsuorasti rakenneindeksin avulla. Ositusindeksoinnissa avaimena on hajautusavain, johon perustuen elementit indeksoidaan listoissa, joiden sisältö on järjestetty elementtitunnisteen perusteella. Ositusindeksit voivat viitata tietosivuihin joko suoraan tai epäsuorasti rakenneindeksin kautta. Samaan dokumenttiin on siis mahdollista luoda sekä rakenneindeksi että ositusindeksejä. Luvuissa 4.1–4.7 kuvatut indeksit perustuvat rakenneindeksiin ja siihen tehtäviin viittauksiin, luvussa 4.8 kuvattu indeksi taas ositukseen.



Kuva 5: Rakenneindeksi (a), ositusindeksi (b) ja ositusindeksi listojen indeksoinnilla (c) [Bac12].

4.1 Rakenneindeksi

Yksinkertaisin XML-dokumentin indeksi on *rakenneindeksi*. Rakenneindeksi indeksoi XML-dokumentin elementtirakenteen esim. B+-puumuodossa (kuva 4). Rakenneindeksin avulla elementit ja niiden lähisukulaiselementit ovat nopeasti paikannettavissa tietosivuilta niiden tunnisteiden perusteella. Esimerkiksi elementin 1.3.5 tunnisteesta voidaan nähdä sen esivanhempien tunnisteiden olevan 1.3 ja 1. Lisäksi rakenneindeksi tukee elementin sisarusten nopeaa löytämistä. Esimerkiksi jos elementin 1.3.5 edeltäjä tietosivulla on 1.3.3.5, seuraa tästä se, että 1.3.5 on elementin 1.3.3 järjestyksessä seuraava sisarelementti. Rakenneindeksin avulla ei voida paikantaa elementtejä sisällön perusteella, vaan sisällön perusteella haettaessa kaikki tietokannan tietosivut on noudettava pysyvästä muistista puskuriin käsittelyä varten, mikä on erittäin hidasta.

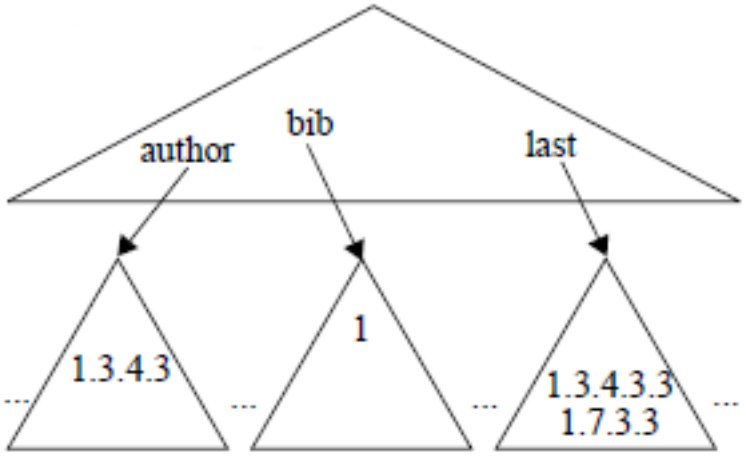
Kun tieto talletetaan tietosivuille elementtitunnisteiden mukaisessa nousevassa järjestyksessä [Wei10], voidaan rakenneindeksi toteuttaa ns. harvana indeksinä indeksoiden vain osa elementeistä, sillä elementtien paikallistamisessa tietosivuilta voidaan käyttää apuna myös elementtien keskinäistä järjestystä, ts. esimerkiksi jos indeksin mukaan ele-

mentit 1.3 ja 1.9 sijaitsevat samalla tietosivulla, niin tämän perusteella myös elementti 1.7 sijaitsee tällä sivulla. Jos tieto tallennetaan tietosivuille muussa järjestyksessä kuten esimerkiksi tietosisällön perusteella ryvästämällä, rakenneindeksi toteutetaan aina tiheänä.

Rakenneindeksissä avaimena on aina elementtitunniste, ja tietosisältönä sen tietosivun suora osoite, jolle indeksoitu elementti on tallennettu. Muissa indeksityypeissä ei välttämättä ole tarpeen säilyttää tietoa elementtien sijainnista tietosivuilla, sillä tieto voidaan hakea rakenneindeksistä. Muissa indeksityypeissä siis riittää, että indeksi sisältää tiedon indeksoitavien elementtien elementtitunnisteista, mutta tietosivuviitteidenkin ylläpitäminen on mahdollista.

4.2 Elementti-indeksi

Elementti-indeksi on kaksitasoinen indeksi, jossa nimihakemistossa indeksoidaan B+-puumuodossa dokumentissa esiintyvien elementtien nimet ja esiintymähakemistossa näiden nimien esiintymät dokumentissa (kuva 6). Harva indeksi ei ole mahdollinen, koska tieto on tallennettu tietosivuille elementeittäin dokumentin puurakenteeseen perustuvassa sukulaisuusjärjestyksessä, ei siis elementtien sisältöön perustuvassa järjestyksessä, ja esiintymähakemisto toteutetaan siis tiheänä indeksinä [Wei10]. Elementti-indeksin avulla voidaan esimerkiksi kuvan 1 dokumentista XPath-lauseketta `//book` evaluoitaessa indeksin avulla paikantaa `book`-nimiset elementit.



Kuva 6: elementti-indeksi [Wei10].

4.3 Sisältöindeksi

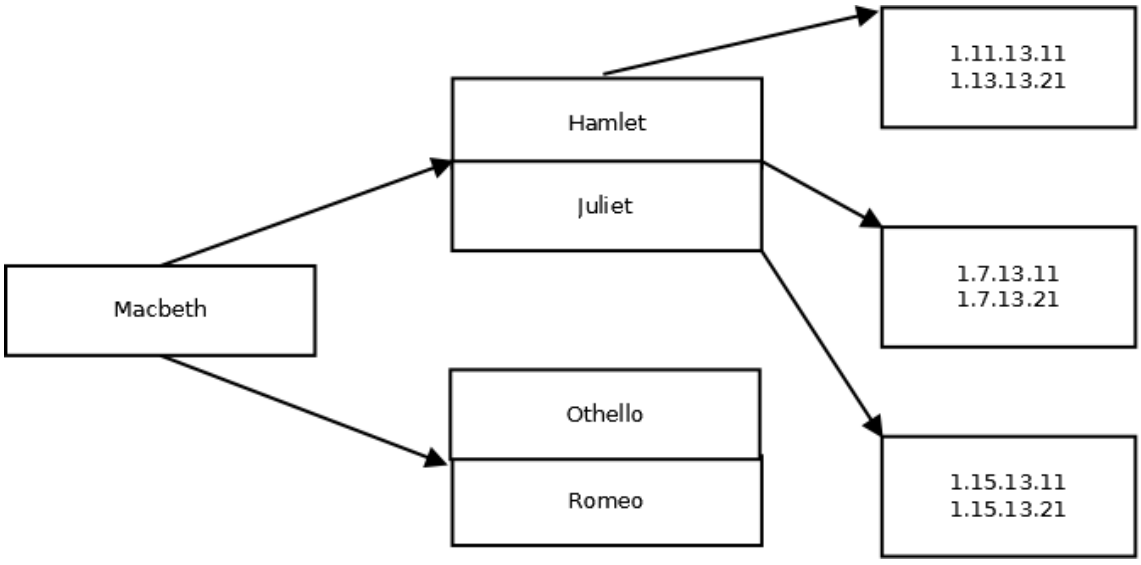
Sisältöindeksissä XML-dokumentti indeksoidaan elementtien sisällön perusteella. Sisältöindeksi voidaan toteuttaa elementtien tietosisältöä indeksoivana B+-puuna tai linkitettyihin listoihin viittaavana B+-puuna. Elementtien tietosisältöä indeksoivassa B+-puussa avaimena on elementin sisältö ja osoitteena elementtitunniste [Wei10].

Listoihin osoittava indeksi (kuva 7) taas perustuu B+ -puuhun, jossa avaimena on elementin tietosisältö ja arvona osoitelista, joka sisältää niiden elementtien tunnisteet, joiden sisällön arvo on yhtä suuri tai suurempi kuin avaimen arvo, mutta pienempi kuin puussa määritellyn seuraavan avaimen arvo. Kuvan 7 hakemistossa indeksoidun hakusanan `Hamlet` osoittamassa listassa säilytetään tunnisteet niistä elementeistä, joiden sisältö on yhtä suuri tai suurempi kuin `Hamlet`, mutta pienempi kuin `Juliet`.

Kummassakin toteutustavassa indeksi on dokumentin elementtitunnisteiden osalta tiheä, koska elementtien järjestys tietosivuilla perustuu dokumentin rakenteeseen. Jälkimmäisessä toteutustavassa sisällön indeksoinnin ei tarvitse olla tiheä, vaan riittää että indeksi kuvassa 7 esitetyn indeksin tapaan jakaa dokumentin sisällön perusteella arvoalueisiin.

Tällainen hakusanojen harva indeksointi vaatii vähemmän tallennustilaa kuin tiheä, mutta harvassa hakusanojen indeksoinnissa hakusanojen osoittamien listojen käsittely taas vaatii enemmän tiedonsiirtoa ja laskentaa kuin tiheässä indeksoinnissa. Harva hakusanojen indeksointi sopii parhaiten tilanteisiin, joissa dokumentissa on paljon elementtejä, mutta erilaisia elementtien arvoja on vähän.

Sisältöindeksi sopii hyvin arvopredikaatti- ja merkkijonohakujen suorittamiseen. Tietosivujen dokumentin rakenteeseen perustuvan elementtien järjestyksen vuoksi sisältöindeksi toteutetaan elementti-indeksin tapaan tiheänä indeksinä. Tiheä indeksi kuluttaa enemmän tallennustilaa kuin harva indeksi, ja sitä tulee päivittää jokaisen indeksoituun elementtiin kohdistuvan päivityksen jälkeen, mutta toisaalta se mahdollistaa esim. indeksoitujen elementtien lukumääriin kohdistuvien tietokantahakujen, kuten `count( doc('factbook')//play[name='Hamlet'])` suorittamisen pelkkien indeksisivujen tietojen perusteella, ilman että varsinaisia tietosivuja tarvitsee käsitellä lainkaan.



Kuva 7: sisältöindeksi hakusanojen harvalla indeksoinnilla (muokattu lähteestä [Base]).

Sisältöindeksin käyttöön perustuvassa tietokantahaun suorituksessa arvovälipredikaatit rajaavat indeksipuun selattavaa osaa, ja osia indeksipuusta voidaan mahdollisesti jättää selaamatta. Mitä rajaavampi on arvovälipredikaatti, sitä suurempi on indeksistä saatava hyöty. Koska osia indeksistä voidaan mahdollisesti kokonaan jättää selaamatta, rajaa-

vuuden myötä myös varsinaisia tietosivuja tarvitsee noutaa vähemmän pysyvästä muistista puskuriin. Käytettäessä hakuperusteena elementtien arvoja, on sisältöindeksi huomattavasti rakenneindeksiä tehokkaampi.

4.4 Polkuindeksi

Polkuindeksi on rakenneindeksiä monipuolisempi indeksi, jonka avulla indeksoidaan elementtejä niiden hierarkkisen aseman perusteella. Polkuindeksointi perustuu kahteen eri tietorakenteeseen: dokumentin erilaiset polkurakenteet kuvaavaan polkukuvaajaan, joka esitetään kuvassa 8, sekä polkuindeksiin. Polkukuvaajassa (kuva 8) kukin erilainen tietokannan polkurakenne saa oman polkutunnisteen (PCR, ”path class reference”). Polkuindeksi taas indeksoi polkukuvaajan kuvaamien polkujen esiintymät dokumentissa. Polkuindeksi sisältää puumuodossa monikoita, joiden avaimena on polkutunniste ja osoitteena polkutunnisteen määrittelyn täyttävän elementin tunniste.

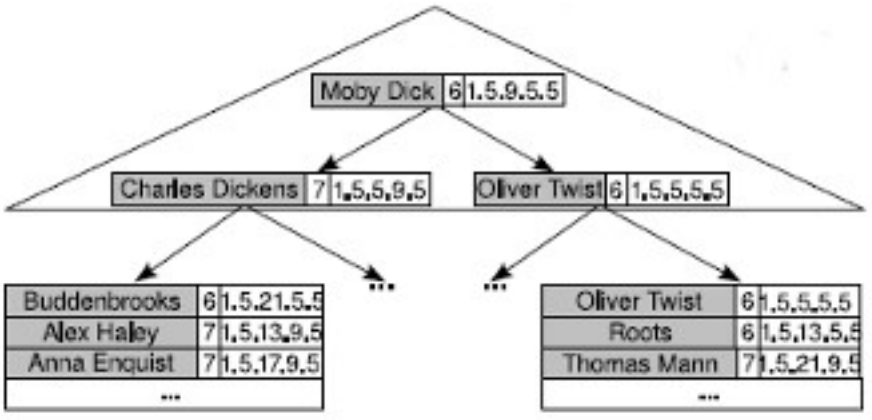
Kuva 8: polkukuvaaja polkutunnisteilla [Wei10].

Esimerkiksi XPath-lausekkeen `/bib/books/book/title` käsittely polkuindeksin avulla on kaksivaiheinen. Ensiksi polkukuvaajasta haetaan tieto lausekkeen ehdot täyttävistä polkutunnisteista. Kun XPath-kyselyn viittaaman dokumentin polkukuvaaja on kuvan 8 mukainen, em. XPath-lausekkeen ehdot täyttäviin elementteihin liittyy polkutunniste 6. Tämän perusteella tarvittavat polkujen esiintymät ovat haettavissa polkutunnisteen avulla polkuindeksistä. Polkuindeksi tehostaa XPath-lausekkeiden suoritusta merkittävästi, sillä se rajaa käsiteltävää dokumentin sisältöä XPath-askelmäärittelyn perusteella. Pol-

kuindeksi on myös tehokas tilankäytöltään, sillä polkukuvaajan ansiosta kunkin polkutyypin linkitys polkutunnisteeseen tarvitsee säilyttää vain yhdessä paikassa, ja polkutunnisteiden käyttö polun esiintymät kuvaavassa polkuindeksissä käyttää vain vähän tallennustilaa [Wei10].

4.5 Rakenne- ja sisältöindeksi

Rakenne- ja sisältöindeksissä (Content and structure index) polkuindeksiin yhdistetään sisältöindeksi. XML-dokumentin elementtejä indeksoidaan monikoilla, joiden avaimena on elementin sisältö, ja tietosisältönä elementin polkutunniste ja elementtitunniste. Kuvassa 9 esitetään esimerkki rakenne- ja sisältöindeksistä. Rakenne- ja sisältöindeksi tehostaa erityisesti sellaisten kyselyiden käsittelyä, joissa on määritelty sekä polkulausekkeita että arvopredikaatteja. Esimerkiksi XPath-lausekkeen `count( doc('factbook')//play[name='Hamlet')` määrittelemät solmut voidaan paikantaa suoraan indeksin perusteella hakemalla ensin polkuindeksistä polun `//play/name` polkutunniste, ja tämän jälkeen hakemalla rakenne- ja sisältöindeksistä niiden elementtien tunnisteet, joiden polkutunniste vastaa em. polkua ja joiden tietosisältö on `Hamlet`. Lausekkeen määrittelemät solmut ovat indeksin perusteella paikannettujen `name`-solmujen vanhempia, ja paikannettavissa suoraan elementtitunnisteen avulla.



Kuva 9: rakenne- ja sisältöindeksi [Wei10].

4.6 N-grammi-indeksi

N-grammi-indeksillä indeksoidaan merkkijonomuotoisen tiedon osamerkkijonoja. N-grammi-indeksi voidaan toteuttaa esimerkiksi ylläpitämällä indeksiä, johon hakuavaimiksi kerätään kaikki tiedostossa esiintyvät tietyn pituiset osamerkkijonot. Hakuavaimiin linkitetään kaikki hakuavaimen esiintymien elementtitunnisteet. Tällaisessa indeksissä esimerkiksi kolmen merkin mittaisten osamerkkijonojen indeksoinnissa merkkijonon `abcde` esiintymään viitataan indeksissä hakusanoilla `abc`, `bcd`, `cde`, `de␣` ja `e␣␣` [Mou09].

N-grammi-indeksi nopeuttaa esim. XQuery-kielen `contains`-funktioon perustuvia osamerkkijonohakuja merkittävästi, sillä muuten tällaisia osamerkkijonohakuja suoritettaessa jouduttaisiin käymään läpi koko sisältö- tai rakenne- ja sisältöindeksi, ja jos tällaisia ei ole käytössä, suorittamaan peräkkäishaku tietosivuihin. Tekstihakujen lisäksi n-grammi-indeksejä voidaan käyttää esimerkiksi kuviin ja genotyypitystietoihin tehtävien hakujen tehostamiseen [Mou09].

4.7 Full Text -indeksi

XML-dokumenttien sisältämän tiedon luonne vaihtelee suuresti. Vahvasti tyypitetylle tiedolle ominaista ovat mm. dokumentteja määrittelevät tietokantaskeemat, elementtien tyypitys ja hierarkkisuus. Heikosti tyypitetyille dokumenteille ominaista taas ovat mm. skeemattomuus ja elementtien sisällä olevat pitkät tyypittämättömät vapaata tekstiä sisältävät merkkijonot. On tärkeää, että heikommin tyypitettyihin dokumentteihin voidaan tehdä hakuja käyttäen merkkijonohakuja kehittyneempiä tiedonhakutekniikoita, kuten tiedon tyyppiesiintymien haku ja hakutulosten pisteytys [W3Cc]. Jotkut XML-tietokannanhallintajärjestelmät tarjoavat tähän tarkoitukseen Full Text -indeksin, josta seuraavassa esitellään toiminnallisuus, mutta ei toteutustapaa.

Full Text 1.0 on W3C:n määrittely tekstidokumentteihin tehtävien kehittyneiden hakujen suoritukseen. Merkkijonojen hakemisen sijasta haut perustuvat tiedon tyyppiesiintymien hakuun. Esimerkiksi haettaessa termillä `DOM` hakua käsitellessä ei anneta kovin paljoa painoa solmulle, jossa esiintyy sana `KINGDOM`. Lisäksi kielten ominaisuudet ja synonyymit huomioidaan hakuja suoritettaessa, esim. englannin kielen sana `mouse` ja sen monikko `mice` ymmärretään saman tyypin esiintymiksi, vaikka niiden esiintymien kirjoitusasut ovat erilaisia. Tuloksia voidaan pisteyttää ja painottaa moni eri tavoin, esimerkiksi termillä `hiiri` tehdyn haun tuloksia voidaan painottaa joko termin `tietokone` tai `jyrsijä` avulla [Mah12].

Esimerkin 6 XQuery-kysely palauttaa kirjoittajien nimet niille kirjoille, joiden nimestä löytyy merkkijono `cat` ja termi `dog`, jonka kirjoitusasu voi olla muukin kuin `dog`. Esimerkin 7 XQuery-kysely taas palauttaa nimet ja tulospisteet niistä kirjoista, joiden sisällössä esiintyvät termit `web site` ja `usability`. Tulosten pisteytyksessä termiä `usability` painotetaan enemmän kuin termiä `web site`.

```
for $b in /books/book
where $b/title contains text ("dog" using stemming) ftand "cat"
return $b/author
```

Esimerkki 6: Hakutermin tyyppiesiintymiä hakeva XQuery-kysely.

```
for $b in /books/book
let score $s := $b/content contains text ("web site" weight {0.5})
                                ftand ("usability" weight {2})
return <result score="{$s}">{$b}</result>
```

Esimerkki 7: Hakutulokset pisteyttävä XQuery-kysely.

4.8 Ositusindeksi

Ositusindeksissä XML-dokumentin elementit indeksoidaan linkitetyissä listoissa, kunkin elementin indeksointiin käytettävän listan määräytyessä hajautusavaimen perusteella. Listat on järjestetty *pinopuuksi* (stack tree), jossa listat säilytetään puumuodossa, hajautusarvon toimiessa avaimena (kuva 5, kohdat b ja c) [Mei02], [Bac12].

Hajautus voidaan tehdä dokumentin rakenteen, sisällön tai kyselyiden suorituksen työmäärän perusteella. Dokumentin rakenteeseen perustuvassa hajautuksessa hajautusavaimena voidaan käyttää esimerkiksi elementtien nimiä tai polkutunnisteita. Dokumentin sisältöön perustuvassa hajautuksessa hajautusavaimena voi olla esimerkiksi elementtien sisältö. Kyselyiden suorittamisen työmäärään perustuvassa hajautuksessa hajautus perustuu esimerkiksi elementtien esiintymien määrään tai tilastotietoon tietokannanhallintajärjestelmässä suoritetuissa kyselyissä käsitellyistä elementeistä [Bac12].

Ositusindeksoinnissa tapahtuva samankaltaisten solmujen ryvästys mahdollistaa esimerkiksi käyttötarkoitukseltaan samankaltaisten solmujen tehokkaan hakemisen XML-dokumentista, erityisesti tapauksessa jossa kaikki tietyn tyyppiset solmut on haettava. Toinen osoitusindeksin etu on se, että linkitetyn listan käyttö mahdollistaa elementtien osoitteiden tehokkaan ohjaamisen syötevirtana esimerkiksi sovellukselle, joka jakaa elementtien käsittelyä useille eri suorittimille esim. hajautusavaimella tai kiertovuoromenetelmällä. Tällä tavoin on mahdollista käsitellä tietokantakyselyä rinnakkaisesti ja käyttää suoritinresursseja tehokkaasti.

Linkitetyt listat tallennetaan binääripuumuodossa, jossa muilla kuin juurisolmulla on enintään yksi lapsi, joka ei ole puun lehti. Yksi lapsi on suora tai epäsuora osoite XML-dokumentin solmuun ja toinen lapsi osoite indeksilistan mahdolliseen seuraavaan solmuun.

5 Tietokantakyselyn käsittely XML-tietokannassa

XML-tietokantakyselyn suoritus toteutetaan noutamalla XPath-lausekkeen määrittelemän kyselyn suorittamisessa tarvittavat solmut sisältävät tietosivut pysyvästä muistista tietokantapuskuriin, ja käsittelemällä tietosivujen sisältämät solmut käyttäen laskenta-algoritmia, joka saa syötteenä yhden tai useamman solmujoukon. Laskenta-algoritmi tekee puskuriin tuotuihin solmuihin XPath-lausekkeen predikaattimääritysten perusteella rajauksia ja suorittaa akselimääritysten perusteella liitoksia, palauttaen lopulta tuloksena tulossolmujoukon [Wei10].

Kyselyn suorituksessa tarvittavat dokumentin solmut tuodaan puskuriin paikantamalla ne rakenneindeksiä käyttäen. Tietosivujen paikannuksessa pyritään käyttämään tietokantaindeksejä mahdollisimman tehokkaasti. Kyselyn suoritus on tehokkaimmillaan, kun käytettävissä on indeksejä, jotka luokittelevat ja rajaavat tietokannan tietoa mahdollisimman pitkälle samalla tavalla kuin suoritettava tietokantakysely. Tällä tavoin levyjärjestelmästä puskuriin tuotavan tiedon määrä pysyy mahdollisimman pienenä, kuten myös muistissa tapahtuvan laskennan laskentakustannukset [Wei10].

XPath-lausekkeen askeleiden määrittelemien solmujoukkojen liittämisessä on kaksi päätyyppiä: binäärinen rakenneliitos ja kokonaisvaltainen oksaliitos. Binääriliitoksessa XPath-lausekkeen määrittelemät solmut valitaan liittämällä polun askelien perusteella haettavia solmujoukkoja aina kaksi kerrallaan. Kokonaisvaltaisessa oksaliitoksessa taas solmut valitaan XPath-lausekkeen polkumäärittelyn perusteella lomittamalla XPath-lausekkeiden askeleiden määrittelemiä solmuja rinnakkaisesti [Bac12].

Weiner ja Härder [Wei10] luettelevat kyselyn tehokkaan suoritussuunnitelman valintaa varten ekvivalenssisäännöt ja kustannusmallin. Ekvivalenssisääntöjen perusteella voidaan johtaa kaikki erilaiset kyselysuunnitelmat, jotka käytettävissä olevat operaattorit, indeksit ja tallennusrakenteet mahdollistavat. Kustannusmallin perusteella voidaan las-

kea kaikille mahdollisille kyselysuunnitelmille kustannusarvio. Selingerin ja kumppanien [Sel79] kyselysuunnitelman valintamallin avulla voidaan tehokkaasti laskea kaikista mahdollisista ekvivalenteista kyselysuunnitelmista tehokkain. Malli perustuu lehdistä kohti juurta muodostettavaan kyselysuunnitelmapuuhun, jossa hierarkkisesti kuvataan kyselyn suorituksen koostuminen osista puun juuren kuvatessa koko suunnitelman. Menetelmän tehokkuus perustuu siihen, että kyselysuunnitelmapuusta poistetaan mahdollisimman aikaisessa vaiheessa solmut, joiden kuvaamille osasuunnitelmille on olemassa ekvivalentti, tehokkaampi vaihtoehto.

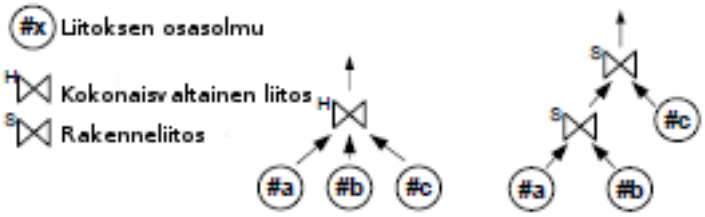
Weinerin ja Härderin mukaan Selingerin ja kumppanien malli sopii pohjaksi myös XML-tietokannanhallintajärjestelmien kyselyoptimointiin. Weiner ja Härder esittävät periaatteet XML-tietokannanhallintajärjestelmän kyselysuunnitelmien ekvivalenssisääntöjen ja laskentakustannusten määrittelemiseksi. Nämä esitellään luvuissa 5.2 ja 5.5. Näiden avulla voidaan muodostaa Selingerin ja kumppanien esittämä kyselysuunnitelmaverkko, kun ekvivalentit kyselysuunnitelmat ja niiden kustannukset voidaan määritellä.

5.1 Polkulausekkeen käsittely laskenta-algoritmissa

XML-dokumentissa primitiiviset rakenteelliset suhteet kahden elementin välillä ovat vanhemman ja lapsen sekä ja esivanhemman ja jälkeläisen välinen suhde. Näiden rakenteellisten suhteiden esiintymien löytäminen XML-dokumenteista on tietokantakyselyprosessorien keskeisimpiä tehtäviä. Tietokannanhallintajärjestelmät käsittelevät yleensä tietokantakyselyt XPath-lausekkeen askeliin perustuvina osakyselyinä, joiden tulokset liitetään toisiinsa [Bac12], [Wei10].

XML-tietokantakyselyjä käsittelevillä algoritmeilla on kolme päätapaa liittää XPath-lausekkeen askeleiden määrittelemät solmut keskenään: puuselausalgoritmit, rakenneliitosalgoritmit (Kuva10) ja kokonaisvaltainen liitos (Kuva 10) [Wei10], [Bac12]. Puuselausalgoritmit perustuvat XML-dokumenttipuihin ja dokumentin solmujen lapsiin ja vanhempiin osoittavien osoittimien käyttöön. Puuselausalgoritmit

perustuvat ”elementti kerrallaan” -lähestymistapaan, ja ovat tehottomampia kuin ”joukko kerrallaan” -lähestymistapaan perustuvat rakenne- ja oksaliitosalgoritmit [Kha02], [Luk12].



Kuva 10: Rakenneliitos ja kokonaisvaltainen oksaliitos (muokattu lähteestä [Bac12]).

Rakenneliitos on solmujoukkojen yhdistämiseen käytettävä algoritmi (yhtälö 1). Se perustuu puoliliitokseen, jossa joukosta A valitaan alkioiden osajoukko, jonka alkioille löytyy akselipredikaatin $AkseliA_i B_j$ määrittelemässä sukulaissuhteessa oleva alkio joukosta B [Luk07]. Akselipredikaatti saa arvon tosi, mikäli sen parametrina olevat elementit ovat rakenteellisesti akselipredikaatin määrittelemässä suhteessa keskenään, esimerkiksi lapsi ja vanhempi [Kha02]. Puoliliitoksen etuna on se, että sen laskennassa riittää käsitellä liitettävistä joukoista liitosehtona oleva osa, mikä pienentää tiedonsiirtokustannuksia, ja että puoliliitos on liitettävää joukkoa A rajaava, toisin kuin esim. karteesinen tulo. Rajaavuus pienentää välituloksia ja vähentää laskentakustannuksia [Hug89]. Rakenneliitokseen osallistuvien solmujoukkojen laskentajärjestykseen vaikuttaa se, onko kysely osittuva vai iteratiivinen. Osittuvan kyselyn, kuten liitteessä kuvatun XMark-koetinkuorman kyselyn Q1, osia voidaan suorittaa ja niiden tuloksia liittää toisiinsa rinnakkaisesti. Iteratiivisissa kyselyissä, kuten XMark-koetinkuorman kyselyssä Q14, osakyselyitä suoritetaan kontekstissa toisen osakyselyn palauttamiin tietoihin, eikä tällaisia osakyselyitä voida suorittaa rinnakkaisina rakenneliitoksina [Bid13].

$$A \otimes_{akseliA_i B_j} B = \{(x, y) | x \in A, y \in B, akseliA_i B_j(x, y) = tosi\}$$

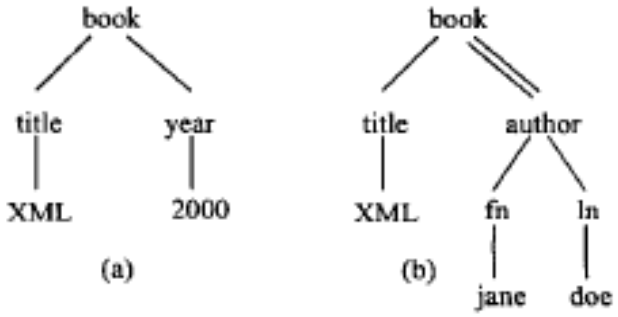
Yhtälö 1. Rakenneliitos (muokattu lähteestä [Luk12]).

Rakenneliitosalgoritmeja on kahta tyyppiä käytettävästä indeksistä riippuen. *Tikapuuliitos* perustuu polku-, rakenne- tai sisältöindeksiin tai peräkkäishakuun. Tikapuuliitoksessa tehdään hierarkkisesti joukko liitoksia, joissa XPath-lausekkeen askeleiden määrittelemät solmujoukot toimivat syötteinä liitoksille, joissa liitosalgoritmi tekee käsittelemänsä XPath-kyselyn osan perusteella solmujoukkohaun ja suorittaa tämän perusteella mahdollista karsintaa saamalleen syötejoukolle. Karsittu syötejoukko on liitoksen tulos, ja tulos taas toimii syötteenä seuraavalle liitokselle. Liitoksia jatketaan niin kauan kunnes kyselypuussa on enää juurisolmu jäljellä. Ositusindeksiin perustuvassa *binääriliitoksessa* liitokset suoritetaan lomittamalla aina kaksi XPath-lausekkeeseen perustuvaa linkitettyä listaa kerrallaan, kunnes jäljellä on enää yksi lista, joka sisältää kyselyn tuloksen [Bac12].

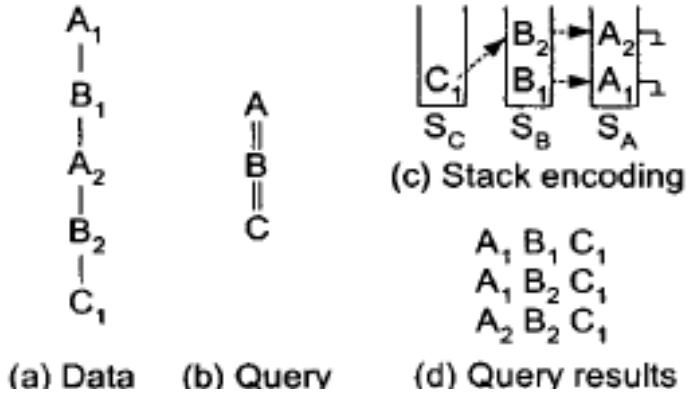
Kokonaisvaltainen oksaliitos on algoritmityyppi, joka nimensä mukaisesti pyrkii käsittelemään XPath-lausekkeen kokonaisvaltaisesti ”oksana” ja löytämään oksan avulla annetusta XML-dokumentista vastaavuudet, joiden perusteella kyselyn tulos lasketaan. Kyselyä kuvaava oksa voidaan esittää esimerkiksi kuvassa 11 esitetyllä tavalla. Oksan solmut voivat esittää elementtimerkintöjä, merkkijonoja ja attribuuttien vertailuja. Oksan kyselyakselia kuvaavat kaaret merkitään yksinkertaisella solmujen välisellä viivalla (vanhemman ja lapsen välinen rakenteellinen suhde) tai kaksinkertaisella viivalla (esivanhemman ja lapsen välinen rakenteellinen suhde).

Kokonaisvaltainen oksaliitos voidaan toteuttaa esimerkiksi Brunon ja kumppanien [Bru02] esittämällä pinoja tietorakenteena käyttävällä algoritmilla, jonka periaate esitetään kuvassa 12, jossa suoritetaan esimerkkikysely `//a//b//c`. Algoritmissa kyselypuun (kuva 12, kohta b) solmuihin yhdistetään XML-dokumentista (kuva 12, kohta a) juuresta aloittaen kyselypuun rakennepredikaattien ehdot täyttävien solmujen sijaintitiedot. Kun löydetään vastaavuus dokumentin ja kyselypuun välillä, tallennetaan tieto vastaavuudesta pinoon, jossa säilytetään kyselypuun kyseisen solmun esiintymätie-

toja (kuva 12, kohta c). Jokainen pinoon tallennettava solmu koostuu kyselysolmun esiintymän sijaintitiedosta dokumentissa ja mahdollisesta linkistä vanhempaan kysely-puussa. Solmujen vertikaalisessa suunnassa esitetyt paikat osoittavat esiintymien rakenteellisen järjestyksen dokumentissa, esimerkiksi A_1 on pinossa elementin A_2 alapuolella ja tämän esivanhempi. Kyselyn tulos esitetään kuvan 12 kohdassa d. Esimerkiksi $[A_2, B_2, C_1]$ sisältyy tulokseen, koska C_1 osoittaa solmuun B_2 ja solmu B_2 taas solmuun A_2 . Myös $[A_1, B_1, C_1]$ sisältyy tulokseen, koska C_1 osoittaa solmuun B_2 , jonka esivanhempi B_1 taas on. Huomattavaa on että $[A_2, B_1, C_1]$ ei sisälly lopputulokseen, koska solmu A_2 sijaitsee solmun A_1 yläpuolella pinossa S_A , johon solmu B_1 osoittaa. Solmu A_2 ei siis näin voi olla solmun B_1 esivanhempi.



Kuva 11: XPath-kyselyiden esityksiä puun oksina [Bru02].



Kuva 12: Esimerkki kokonaisvaltaisen oksaliitosalgorimin suorituksen vaiheista [Bru02].

```
Algorithm PathStack(q)
01 while (¬end(q))
02   n_min = getMinSource(q)  // find the next node
03   for query node n_i of q in descending i order // clean stack
04     while((¬empty(S_i)∧(top(S_i) is not an ancestor of next(T_min))
05       e_i = pop(S_i)
06       if (n_i is the output leaf node ∧ e_i.satisfy=true)
07         showSolutions(e_i)  // output solution
08       else if (n_i is a non-output node)
09         updateSatisfy(n_i, e_i)
10   //push the next node
11   moveStreamToStack(n_min, T_min, S_min, pointer to top(S_min-1))
12 repeat steps 03 to 09 for the remaining nodes in the stacks
```

Kuva 13: PathStack-algoritmi [Bru02], [Jia05].

Esimerkki edellä kuvattua tapaa kokonaisvaltaiseen oksaliitokseen toteutukseen käyttävästä olevasta algoritmista on PathStack (kuva 13). PathStack-algoritmi saa syötevirtana XML-dokumentin solmuja dokumentin rakenteen mukaisessa järjestyksessä (kuvan 13 rivit 1 ja 2). Uuden solmun käsittelyn aluksi algoritmi poistaa pinoista ne solmut, jotka eivät voi kuulua lopputulokseen, ts. ne solmut jota eivät ole kyselypuun lehtiä ja joiden dokumenttijärjestyksessä oikealla puolella uusi käsiteltävä elementti on (rivit 3-5). Tämän jälkeen tarkistetaan, löytyykö käsiteltävälle elementille vastaavuutta kyselypuusta, ja jos näin on viedään tieto tästä kyseisen kyselysolmun vastauspinoon. Jos käsiteltävä solmu on puun lehtisolmu ja vastaavuus kyselypuuhun löytyy, voidaan tieto solmusta viedä suoraan kyselyn tulosjoukkoon.

Kokonaisvaltaisessa oksaliitoksessa keskeinen osa on valintamekanismi, jonka avulla mahdollisimman aikaisessa vaiheessa lomitettavista listoista havaitaan ja jätetään jatkokäsittelemättä solmut, jotka eivät täytä kyselylausekkeen määrittelyä. Solmuja voidaan sivuuttaa ennen listojen lomittamista (esikäsittely) ja listojen lomittamisen jälkeen (jälkikäsittely) [Gri10].

Kokonaisvaltaisen oksaliitosalgoritmin etuna on se, että XPath-lausekkeen määrittelemiä solmuja voidaan rajata mahdollisimman aikaisessa vaiheessa. Näin oksaliitosten käyttö vähentää välitulosten kokoa, mikä puolestaan vähentää laskentakustannuksia ja käsittelyssä tarvittavan muistin määrää [Bac12]. Kokonaisvaltainen liitos on tehok-

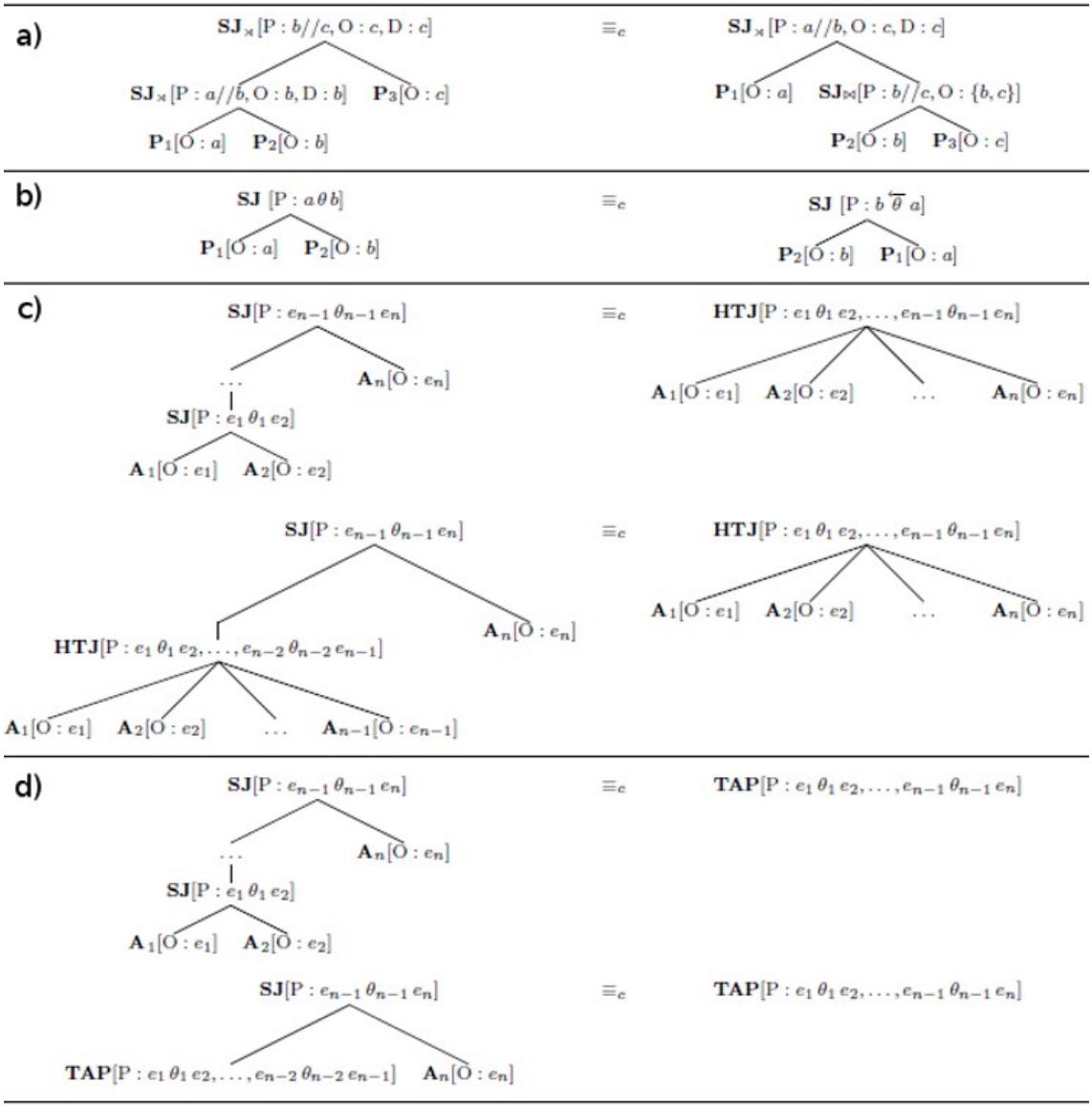
kaampi kuin kyselyn suoritus binääriliitoksina, joissa ei voida optimaalisesti käyttää tietoa koko XPath-lausekkeen määrittelystä eikä toisten alikyselyiden palauttamia tuloksia, ja välituloksien rajaus tapahtuu vasta liitoksissa ja niiden yhdistelmissä [Sch02].

5.2 Kyselysuunnitelmien ekvivalenssi

XML-tietokannanhallintajärjestelmä tekee ennen kyselyn suorittamista kyselylle vaihtoehtoisia suoritussuunnitelmia valiten erilaisista vaihtoehtoisista suunnitelmista tehokkaimmaksi arvioimansa. Tehokkaimman suunnitelman valitsemiseksi on määriteltävä ekvivalenssisäännöt sille, mitä suunnitelmia ja suunnitelmien osia voidaan pitää keskenään ekvivalentteina, ts. mitkä suunnitelmat ja niiden osat tuottavat yhtäläisen tuloksen [Hel94].

Starburst oli IBM-yhtiön 1980- ja 1990-luvuilla toiminut projekti, jonka tarkoituksena oli kehittää helposti laajennettavan relaatiotietokannan prototyyppi. Projekti esitteli useita merkittäviä tietokantaoptimoinnin käsitteitä. Yksi näistä on Query Graph Model (QGM), jossa relaatioalgebraa käyttäen esitetään tietokantakyselyiden evaluointi- ja ekvivalenssisääntöjä. Lisäksi projekti esitteli kyselyoptimoijiin ja kyselyiden uudelleenmuotoiluun liittyvää käsitteistöä [Pir92].

Weiner ja Härder [Wei10] esittelevät XML-tietokannoille tarkoitetun XML Query Graph Model -mallin, joka on QGM-mallin laajennus. XQGM on yhteensopiva XQuery 1.0-määrittelyn kanssa, tukien mm. FLWOR-lausekkeiden käyttöä.



Kuva 14: kyselysuunnitelmien ekvivalenssisäännöt: rakenneliitoksen liitännäisyys (a), rakenneliitoksen vaihdannaisuus (b), rakenneliitoksen fuusiointi (c) ja kysely-suunnitelman korvaaminen indeksihaulla (d) [Wei10].

Kyselyn optimointi aloitetaan luomalla kyselylle joukko suoritussuunnitelmia. Kussakin suunnitelmassa huomioidaan kyselyn käsittelemien dokumenttien rakennepredikaatit, elementtien sisältöön liittyvät predikaatit, hakutulosten lajittelu ja hakutulosten käsitte-

ly. Weiner ja Härder esittelevät notaation, jossa kyselysuunnitelma P koostuu määritteistä $\left[P_1, P_{2,}, \ldots, p_n\right]$. Kyselysuunnitelma voi hierarkkisesti sisältää alikyselysuunnitelmia [Wei10]. Erilaiset kyselysuunnitelmat esitetään taulukossa 2 ja kyselysuunnitelmissa käytettävät määritteet taulukossa 3.

P_i	Satunnainen kyselysuunnitelma (Plan)
A_j	Tiedonhakusuunnitelma (Access plan). Tietosivujen siirto puskuriin rakenne- tai elementti-indeksiä käyttäen
SJ_t	Rakenneliitos (Structural join)
HTJ	Kokonaisvaltainen oksaliitos (Holistic twig join)
TAP	Kolmannen sukupolven tiedonhakusuunnitelma (Tertiary access path). Tietosivujen siirto puskuriin polku- tai rakenne- ja sisältöindeksiä käyttäen

Taulukko 2: Erilaiset kyselysuunnitelmat ekvivalenssisääntömallissa.

I	Fyysinen operaattori
P	Predikaatti kuten XPath-akseli (Predicate)
D	Syötesekvenssi, josta duplikaatit poistetaan (Input sequence where duplicates are removed)
O	Tulossekvenssi (Sequence to opt out)

Taulukko 3: Kyselysuunnitelmien määritteet ekvivalenssisääntömallissa.

Toteutuksen ekvivalenssi. Weinerin ja Härderin mukaan kyselysuunnitelman P toteutus voidaan vaihtaa kuvassa 14 esitettyjen periaatteiden mukaan toiseen vastaavaan toteutukseen jos ja vain jos korvaava suunnitelma on mahdollista toteuttaa käytettävissä olevilla operaattoreilla, ja korvaava suunnitelma on ekvivalentti alkuperäisen suunnitelman kanssa.

Rakenneliitoksen liitännäisyys (assosiatiivisuus). Toisin kun relaatiotietokannoissa, XML-tietokannoissa liitettävien tietojen järjestys on säilytettävä välituloksissa ja lopputuloksessa. XML-tietokantojen käyttämä rakenneliitos säilyttää järjestyksen, ja liitoksen osat voidaan suorittaa kuvan 14 kohdan a osoittamalla tavalla missä tahansa järjestyksessä, sillä akselipredikaattien perusteella saadaan sama lopputulos liitosjärjestyksestä riippumatta.

Rakenneliitoksen järjestyksen vaihdannaisuus. Rakenneliitoksessa liitettävien kahden joukon järjestys voidaan vaihtaa keskenään vaihtamalla samalla liitoksessa käytettävä XPath-akseli sen käänteisakseliin. Esimerkiksi XPath-lausekkeen `//PLAY//TITLE` vasemman- ja oikeanpuolisten askeleiden määrittelemien solmujoukkojen liitos tuottaa ekvivalentin tuloksen XPath-lausekkeen `//TITLE[ancestor::PLAY]` askeleiden määrittelemien solmujoukkojen rakenneliitoksen kanssa [Wei10].

Rakenneliitosten fuusiointi. Kyselyn osa, joka koostuu joukosta rakenneliitoksia, on mahdollista korvata kokonaisvaltaisella oksaliitoksella, mikäli tietokannanhallintajärjestelmä mahdollistaa tämän.

Kyselysuunnitelman korvaaminen indeksihaulla. Rakenneliitoksiin tai kokonaisvaltaiseen oksaliitokseen perustuva kyselysuunnitelma tai sen osa voidaan korvata yhdellä solmujen indeksihaulla, mikäli sopiva indeksi on määritelty.

5.3 Tietokantakyselyn suoritus osakyselyinä ja liitoksina

XML-tietokantakyselyn suoritus koostuu usein osakyselyistä, joiden tulokset liitetään toisiinsa. Esimerkiksi kuvan 1 dokumenttiin suoritettavan XPath-lausekkeen `/bib/books/book[author='Charles Dickens']/title` suoritussuunnitelma voidaan tehdä useilla eri tavoilla, joista esitetään muutamia esimerkkejä.

Suoritussuunnitelma 1: polkuindeksi, sisältöindeksi ja tikapuuliitos. Polkuindeksi on määritelty kaikille poluille ja sisältöindeksi kaikille elementeille. Kysely suoritetaan kahdessa osassa: osassa P1 `[O:/bib/books/book/title]` haetaan polkuindeksin perusteella kaikki polkulausekkeen määrittelemät `title`-solmut, ja osassa P2 `[O://book[author='Charles Dickens']]` haetaan kaikki `book`-solmut, joista predikaatin perusteella rajataan ne, joiden sisältö on `Charles Dickens`. P1 palauttaa solmun tunnisteella 1.5.5.5 ja P2 palauttaa solmun tunnisteella 1.5.5. Tämän jälkeen tehdään puoliliitos [P1 ⋉ P2], jossa joukosta P1:stä valitaan akselipredikaatin / ja osakyselyn P2 palauttaman tuloksen perusteella ne solmut, joiden vanhemmat löytyvät joukosta P2. Tulokseksi saadaan solmu 1.5.5.5 sisältönä `<title>Oliver Twist</title>`.

Suoritussuunnitelma 2: elementti-indeksi ja binäärinen tikapuuliitos. Elementti-indeksi on määritelty kaikille elementeille. Kysely koostuu neljästä osasta, jotka ovat P1`[O:bib]`, P2`[O:books]`, P3`[O:book]`, P4`[O:author[P='Charles Dickens']]` ja P5`[O:title]`. Tässä suoritussuunnitelmassa kyselyn osissa P1, P2, P3, P4 ja P5 haetaan tulosjoukot indeksin perusteella. Osakyselyssä P4 tehdään myös akselipredikaatin määrittelemään solmujoukkoon rajaus arvopredikaatin perusteella. P1 palauttaa solmun tunnisteella 1, P2 palauttaa solmun tunnisteella 1.5, P3 palauttaa solmut tunnisteilla 1.5.5, 1.5.9 ja 1.5.13, P4 solmun tunnisteella 1.5.5.9 ja P5 solmun tunnisteella 1.5.5.5. Välituloksiin tehdään neljä liitosta, esimerkiksi järjestyksessä [P5 ⋉ P4 ⋉ P3 ⋉ P3 ⋉ P1], jossa välitulosten P5 ja P4 liitoksessa rakennepredikaattina on solmujen sisaruus ja muissa liitoksissa se, että liitoksen vasemman joukon solmut ovat liitoksen oikean joukon solmujen lapsia. Tulokseksi saadaan solmu 1.5.5.5.

Suoritussuunnitelma 3: rakenne- ja sisältöindeksi. Kuvan 8 mukaisesta polkukuvauksesta haetaan polkutunnisteet poluille `/bib/book/book/title` (polkutunniste on 6) ja `/bib/book/book/author` (polkutunniste on 7). Tämän jälkeen suoritetaan osakyselyt P1: `[O:/bib/books/book/title]` ja

P2:`[O:/bib/books/book/author[P='Charles Dickens'].`

P1 palauttaa solmun 1.5.5.5 ja P2 solmun 1.5.5.9. Koska P1 ja P2 ovat rakennepredikaatin perusteella sisaruksia, valitaan P1:stä ne solmut, joiden vanhempi löytyy P2:n

solmujen vanhemmista. Rakenneliitos [P1 ⋉ P2] rakennepredikaatilla ”sisarus” tuottaa vastaukseksi solmun 1.5.5.5.

Suoritussuunnitelma 4: kokonaisvaltainen oksaliitos käyttäen elementin mukaan järjestettyjä listoja, jotka on indeksoitu B+-puuna elementin mukaan, ja joiden sisältö on:

lista 1 (bib) `[1]`,

lista 2 (books) `[1.5]`,

lista 3 (book) `[1.5.5|1.5.9|1.5.13]`,

lista 4 (author) `[1.5.5.9Charles Dickens|1.5.9.13William Shakespeare]` ja

lista 5 (title) `[1.5.5.5Oliver Twist]`.

Suoritussuunnitelmassa 4 kaikki listat lomitetaan toisiinsa jättäen `William Shakespeare` sisältöpredikaatin perusteella pois lomituksesta mahdollisimman aikaisessa vaiheessa. Tulokseksi saadaan listasta 5 solmu 1.5.5.5 sisältönä `<title>Oliver Twist</title>`.

5.4 Kyselyjen uudelleenmuotoilu

Tietokantakyselyn uudelleenmuotoilussa etsitään tietokantakyselylle rakenteeltaan yksinkertaisempia tai optimoitavuudeltaan tehokkaampia vaihtoehtoja, jotka ovat määriteltävältä tulosjoukoltaan ekvivalentteja alkuperäisen kyselyn kanssa. Näin voidaan tuottaa kyselyoptimoijalle helpommin käsiteltäviä kyselyitä. Tietokannanhallintajärjestelmä voi kyselyiden uudelleenmuotoilussa käyttää hyväksi tietoa kyselyoptimoijansa ominaisuuksista ja suosia uudelleenmuotoilussa kyselyitä, jotka ovat tehokkaimmin suoritettavissa.

Gueni ja kumppanit [Gue08] esittävät algoritmin, jolla XQuery-kyselyitä voidaan yksinkertaistaa madaltamalla sisäkkäisten kyselyiden muodostamia hierarkioita, joissa ulomman tason kyselyiden tulokset vaikuttavat sisempien kyselyiden palauttamiin tuloksiin. Tämä tapahtuu esimerkiksi yhdistämällä sisäkkäisiä kyselyitä polkukyselyiksi.

Näin indeksien käyttöä voidaan tehostaa, ja välitulosten määrä ja niiden liitostarve vähenevät. Kyselyoptimoija laskee uudelleenmuotoillulle kyselylle vaihtoehtoiset suoritussuunnitelmat samoilla periaatteilla, kuin ne olisi tehty alkuperäiselle kyselylle. Sekä BaseX- että eXist-tietokannanhallintajärjestelmät voidaan määritellä muotoilemaan kyselyitä uudelleen ennen niiden suoritussuunnitelmien tekemistä. BaseX:n graafisessa käyttöliittymässä käyttäjälle myös näytetään uudelleenmuotoiltu kyselysuunnitelma.

a)
```
let $act := /PLAY/ACT
for $scene in $act/SCENE
return <scene>{$scene/TITLE}</scene>
```

b)
```
for $scene in /PLAY/ACT/SCENE
return <scene>{$scene/TITLE}</scene>
```

Kuva 15: XQuery-kysely (a) ja sama kysely uudelleenmuotoiltuna (b).

Kuvan 15 kohdan a XQuery-kysely hakee eXist-tietokannanhallintajärjestelmän esimerkkidokumenttina käytettävän näytelmäaiheisen dokumentin näytelmien näytökset, joiden perusteella haetaan näytelmien kohtausten nimet. Tämä kysely voidaan korvata ekvivalentilla kuvan 15 kohdan b kyselyllä, joka on rakenteeltaan yksinkertaisempi ja mahdollistaa suoritussuunnitelmassa polkuindeksin tehokkaamman käytön, minkä ansioista vältetään `/PLAY/ACT` ja `/SCENE`-polkumääritysten tuottamien välitulosten käsittely liittämällä.

5.5 Kustannusmalli

Weiner ja Härder esittävät XML-tietokannanhallintajärjestelmän kustannusperustaisen kyselyoptimointimallin, jossa tietokantakyselyn suorituskustannukset koostuvat indeksejä ja elementtejä sisältävien tietosivujen tietokantapuskuriin siirtämisestä syntyvistä tiedonsiirtokustannuksista sekä muistissa tapahtuvan käsittelyn laskentakustannuksista [Wei10]. Mallissa tietokannan elementtejä sisältävät tietosivut käsitetään indeksipuun lehtinä.

Weinerin ja Härderin mallissa kyselyn suorituksen tiedonsiirto- ja laskentakustannukset määräytyvät tietosivujen pysyvästä muistista paikantamisessa käytettävän indeksin sekä mahdollisten polkulausekkeiden ja arvopredikaattien rajaavuuden perusteella. Kustannuksen laskentamalli eri indeksityypeille on esitetty taulukossa 4. Indeksityypistä riippumatta tiedonsiirtokustannusten laskennassa siirrettävien tietosivujen kokonaismäärästä vähennetään aina yksi, koska dokumentin indeksipuun juurisolmun oletetaan aina olevan valmiiksi puskurissa [Wei10].

Operaattori	Tiedonsiirtokustannus
a) rakenne-indeksihaku	$[h(i)+PCard(i)-1]*PageFetchCost$
b) elementti-indeksihaku	$[h(i_n)+h(i_r)+PCard(i_r)-1]*PageFetchCost$
c) polkuindeksihaku	$[h(i)+[PathSel_i(e)\,xPCard(i)]-1]*PageFetchCost$
d) rakenne- ja sisältöindeksihaku	$[h(i)+[PathSel_i(e)*Sel(p)*PCard(i)]-1]$
e) pinopuu	$Cost_{IO}(vasen)+Cost_{IO}(oikea)$
f) navigaatiopuu	$Cost_{IO}(vasen)+TCard(vasen)*Cost_{IO}(oikea)$
g) kokonaisvaltainen oksaliitoshaku	$Cost_{IO}(vasen)+Cost_{IO}(oikea)$

Operaattori	Laskentakustannus
a) rakenne-indeksihaku	$TCard(i)*EvalCost(p)$
b) elementti-indeksihaku	$TCard(i_r)*EvalCost(p)$
c) polkuindeksihaku	$PathCard(e)*EvalCost(p)$
d) rakenne- ja sisältöindeksihaku	$PathCard(e)*Sel(p)*EvalCost(p)$
e) pinopuu	$Cost_{CPU}(vasen)+Cost_{CPU}(oikea)+TCard(vasen)+EvalCost(p)$
f) navigaatiopuu	$Cost_{CPU}(vasen)+TCard(vasen)+Cost_{CPU}(oikea)*EvalCost(p)$
g) kokonaisvaltainen oksaliitoshaku	$Cost_{CPU}(vasen)+Cost_{CPU}(oikea)+TCard(vasen)+EvalCost(p)$

Taulukko 4: Kustannusarvioiden laskentakaavat erilaisille kyselysuunnitelmille [Wei10].

Mallissa käytetyt termit ja niiden selitykset ovat:

$PageFetchCost$ Tietosivun levyltä noutamisen ja tietokantapuskuriin lataamisen kustannus,

$TCard(x)$ x:n sisältämien elementtien määrä,

$PCard(x)$ x:n tallentamiseen käytettyjen tietosivujen määrä,

$PathCard(x)$ polkulausekkeen esiintymien määrä,

$Sel(p)$ arvopredikaatin p rajaavuus,

$PathSel(e) = \frac{PathCard(e)}{TCard(x)}$,

$h(i)$ indeksin i korkeus,

$EvalCost(p)$ valinnaisen predikaatin käsittelykustannus,

$Cost_{IO}(y)$ lapsioperaattorin y tiedonsiirtokustannus ja

$Cost_{CPU}(y)$ lapsioperaattorin y laskentakustannus.

Rakenneindeksiin perustuvassa haussa tietyn nimisten elementtien löytämiseksi on selattava kaikki indeksipuun lehtisivut läpi. Ensimmäisen lehtisivun löytämiseksi selataan indeksipuun sisäsolmut siirtyen juurisolmusta kohti järjestykseltään vasemmanpuoleista sisäsolmua. Tiedonsiirtokustannus syntyy siitä, että noudetaan indeksipuun korkeuden verran sisäsolmutietoja sisältäviä tietosivuja ja kaikki indeksipuun lehtisivut pysyvästä muistista tietokannanhallintajärjestelmän puskuriin. Laskentakustannus on dokumentin tietosivujen sisältämien elementtien määrän ja valinnaisen arvopredikaatin käsittelykustannuksen tulo (taulukko 4, kohta a).

Elementti-indeksiin perustuvassa selauksessa elementit haetaan nimen perusteella selaamalla ensin elementtinimihakemisto, jotta löydetään elementin oma hakemisto. Tästä syntyy elementtinimihakemistopuun korkeuden verran tietosivunoutoja levyltä puskuriin. Kun elementin oma hakemisto on paikannettu, selataan tämän hakemiston sisäsolmut juuresta vasemmanpuolimmaiseen jälkeläiseen, ja tämän jälkeen kaikki lehtisolmut. Laskentakustannus on valitun elementin tiedon sisältämien tietosivujen sisältämien elementtien määrän ja valinnaisen arvopredikaatin käsittelykustannuksen tulo (taulukko 4, kohta b).

Polkuindeksin hakemistoselaamisessa selataan polkutunnisteen perusteella polkuhakemistopuun sisäsolmuja puun juuresta alkaen siirtyen kohti alimman tason sisäsolmua, joka osoittaa ensimmäiseen lehtisolmun, jossa viitataan polun esiintymiin. Tämän jälkeen selataan ensimmäisestä tällaisesta lehtisolmusta alkaen järjestyksessä vasemmalta lehtisolmuja niin pitkään kun ne sisältävät polun esiintymiä. Kaikkia lehtisolmuja ei siis

tarvitse selata, vaan selattavien lehtisolmujen osuus kaikista puun lehtisolmuista on sama kuin polkuehdon täyttävien solmujen osuus kaikista tietokannan solmuista. Laskentakustannus on polkulausekkeen esiintymien määrän ja valinnaisen arvopredikaatin käsittelykustannuksen tulo (taulukko 4, kohta c).

Rakenne- ja sisältöindeksiin perustuvassa hakemistoselaamisessa selataan hakemistoa elementin nimen ja polkutunnisteen perusteella puun juuresta alkaen kohti polkuhakemistopuun sisäsolmuja samalla periaatteella kuin polkuindeksiä selattaessa. Tämän jälkeen selataan ensimmäisestä lehtisolmusta alkaen järjestyksessä vasemmalta lehtisolmuja niin pitkään kun ne sisältävät nimen ja polun esiintymiä. Selattavien lehtisolmujen osuus kaikista lehtisolmuista on polkutunnisteen rajaamien elementtien osuus kaikista elementeistä, joista edelleen rajataan elementin nimen rajaama osuus kaikista näistä polun esiintymistä. Laskentakustannus on polun esiintymien, indeksin sisältöpredikaatin rajaavuuden ja valinnaisen arvopredikaatin käsittelykustannuksen tulo (taulukko 4, kohta d).

Pinopuulistojen yhdistelyssä tiedonsiirtokustannus on listojen puskuriin siirtämisen tiedonsiirtokustannus, ja laskentakustannus on vasemman ja oikean pinon laskentakustannus, johon lisätään vasemman pinopuulistan sisältämien elementtien lukumäärän ja valinnaisen arvopredikaatin käsittelykustannuksen tulo (taulukko 4, kohta e). Koska kyseessä on binäärinen liitos, tarvitsee arvopredikaatti laskea vain toiselle lomitettavista listoista.

Navigaatiopuu on operaattori, joka perustuu ohjelmointikielellä määrittelyihin sisäkkäisiin lausekkeisiin, kuten XQuery-kielen `FOR`-lausekkeisiin, joissa ulompien lausekkeiden määrittelemät solmut vaikuttavat sisemmissä lausekkeissa määriteltäviin solmuihin. Tiedonsiirtokustannus on vasemman listan puskuriin siirtämisen kustannuksen ja vasemman puun solmujen sisällön perusteella noudettavien listojen puskuriin siirtämisen kustannuksen summa. Laskentakustannus on vasemman listan laskentakustannus johon lisätään kutakin vasemman listan solmua kohden noudettavan listan ja valinnaisen laskentapredikaatin käsittelykustannuksen tulo (taulukko 4, kohta f).

Kokonaisvaltaisen oksaliitoksen laskennassa käytettävän pinopuuliitoksen kustannus käsitellään mallissa erikoistapauksena, jossa lomitettavia listoja on kaksi. Tiedonsiirto- ja laskentakustannuksen kaava on sama kuin kahden pinopuulistan käsittelyssä (taulukko 4, kohta g), mutta predikaatin laskentakustannus $EvalCost(p)$ on erilainen näiden kahden tavan välillä.

5.6 Suoritussuunnitelman valinta

Kyselyoptimoija on tietokannanhallintajärjestelmän osa, jonka tehtävänä on määritellä mahdollisimman tehokas kyselyn suoritussuunnitelma. Tämä tapahtuu laskemalla kustannusarvioita erilaisille mahdollisille kyselysuunnitelmille, ja valitsemalla kustannuksiltaan pienin vaihtoehto. Kyselyn suorituksen ajallisesta kestosta suurimman osan muodostavat laskenta- ja tiedonsiirtokustannukset [Sch09]. Suurin osa kyselyn tiedonsiirtokustannuksista syntyy tietosivujen hakemisesta puskuriin, ja laskentakustannuksista taas tietosivujen käsittelystä, liitosten laskemisesta ja välitulosten käsittelystä. Tietosivujen tehokas siirto puskuriin vähentää puskuroitujen sivujen käsittelyyn tarvittavaa laskentakapasiteettia, ja tehokas liitosjärjestysten valinta taas vähentää puskuriin siirrettävien tietosivujen määrää [Hel94]. Kyselyoptimoija huomioi sekä hakupolut (esim. peräkkäishaku ja indeksihaku) että liitosjärjestykset [Wei10], [Bac12]. Koska tietokantakyselyn optimoinnissa vaihtoehtoisia arvioitavia suoritussuunnitelmia voi olla hyvin paljon, ja suoritussuunnitelmien kustannusten laskenta voi kuluttaa runsaasti laskenta- ja tiedonsiirtokapasiteettia, on tärkeää että myös optimoijan toiminta optimoidaan [Sel79], [Sch02].

Tehokkaassa suoritussuunnitelmassa osakyselyn käsittelemät solmut noudetaan puskuriin käyttäen tehokkainta mahdollista indeksiä. Esimerkiksi XPath-lausekkeen `/bib/books/book/year` määrittelemä kysely voitaisiin toteuttaa tehokkaaksi käyttämällä polkuindeksiä, mikäli tällainen on määritelty. Indeksin perusteella noudetut tietosivut sisältävät kyselyn tuloksen, eikä alikyselysuunnitelmia tai tulosjoukkojen liitoksia ole tarpeen suorittaa. Liitosjärjestyksen valinta vaikuttaa merkittävästi

tietokantakyselyn suoritustehokkuuteen, sillä aloittamalla liitokset mahdollisimman pieniä tulosjoukkoja tuottavista liitoksista, pysyvät välitulokset mahdollisimman pieninä. Tämä pienentää paitsi laskenta-, myös tiedonsiirtokustannuksia, sillä välitulosten koko vaikuttaa myöhemmin suoritettavia kyselyn osia varten haettavien tietosivujen määrään [Hel94]. Koska XPath-lausekkeet mahdollistavat suuren määrän erilaisia polkujen, liitosten ja predikaattien permutaatioita, suoritetaan kyselyt usein osakyselyinä, joiden tulokset liitetään toisiinsa. Tilanne vastaa relaatiotietokannan hakua, jossa suoritetaan kahteen tai useampaan tietokantatauluun kohdistuva liitoshaku. Tietosivuhakuihin ja liitoksiin perustuvassa tietokantakyselyn optimoinnissa optimoijan on löydettävä tehokkain tapa suorittaa tietokantakysely, kun käytössä voi olla suuriakin määriä erilaisia hakutapoja tietosivujen noutamiseksi puskuriin, ja erilaisia tapoja liittää puskuriin noudetut solmujoukot toisiinsa [Hel94].

Weinerin ja Härderin mukaan XML-tietokantojen kyselyoptimoija voidaan toteuttaa Selingerin ja kumppanien esittelemään System R -tietokannanhallintajärjestelmän kyselyoptimoijaan perustuen [Wei10]. Selinger ja kumppanit esittävät System R -tietokannanhallintajärjestelmän käyttämän kyselyn suoritusjärjestyksen valinta-algoritmin [Sel79]. Useimpien relaatiotietokannanjärjestelmien kyselyoptimoija perustuu nykyäänkin kyseiseen algoritmiin [Wei10].

System R- järjestelmässä kyselysuunnitelma valitaan alhaalta ylös -periaatteella. Mallissa tietokantakyselyn suorittamiselle vaihtoehtoina on erilaisia tietosivuhakuja, predikaattirajauksia ja liitosjärjestyksiä, ja optimoijan tehtävänä on löytää näiden mahdollisimman tehokas, oikeat kyselytulokset tuottava yhdistelmä. Mallissa erilaisten kyselysuunnitelmavaihtoehtojen oletetaan olevan puita, jotka hierarkkisesti koostuvat kyselysuunnitelman osista niin, että puun lehdet ovat tietosivuhakuja, jotka noutavat tietosivuja pysyvästä muistista. Mallissa sisäsolmut ovat valittujen tietojen liitoksia ja puun juuri valittu kyselysuunnitelma. Kuljettu polku puun lehdistä juureen kuvaa niitä päätöksiä, jotka optimoijan tehtävä on tehtävä ennen lopullisen suunnitelman valintaa [Hel94].

Optimoija tekee suunnitelmapuiden rajausta käymällä läpi puun alimman tason solmuja ja tutkimalla mitä erilaisia uusia päätöksiä optimoijan on tehtävä kustannusten laskemiseksi tässä vaiheessa. Tämän jälkeen optimoija laskee vaihtoehdot ja kustannukset uusille päätöksille, mm. tiedonsiirtokustannusten, käytettävissä olevan laskentakapasiteetin ja relaatioiden koosta kerättyjen tilastotietojen perusteella, ja säilyttää tiedot, jotta niitä ei tarvitse enää laskea uudelleen. Tämän jälkeen optimoija poistaa kyselysuunnitelmapuista laskettujen vaihtoehtojen perusteella vaihtoehdot, jotka eivät ole tehokkaita. Algoritmin suoritusta jatketaan siirtymällä lehdistä kohti juurta ja jatkamalla uusien vaihtoehtojen kustannusten laskentaa ja vaihtoehtojen karsintaa, kunnes jäljellä on vain yksi polku, joka on toteutettava kyselysuunnitelma [Hel94].

6 Indeksien toteutus eXist- ja BaseX-järjestelmissä

XML-pohjaisissa eXist- ja BaseX-tietokannanhallintajärjestelmissä XML-dokumentit indeksoidaan aina rakenneindeksin avulla. Tämän lisäksi eXist ja BaseX mahdollistavat valinnaisten indeksien käytön [Mei02].

6.1 eXist

Avoimeen lähdekoodiin perustuva eXist on XML-pohjainen tietokannanhallintajärjestelmä, joka on suunniteltu helposti integroitavaksi erilaisiin XML-muotoista tietoa käsitteleviin järjestelmiin, kuten verkkopalvelimiin. eXist perustuu XML-dokumenttien skeemattomaan tallennukseen hierarkkisissa kokoelmissa, jotka vastaavat tyypillistä hierarkkista tiedostojärjestelmää hakemistoineen. eXist on toteutettu kokonaan Java-ohjelmointikielellä ja on näin käyttöjärjestelmäriippumaton. eXist käyttää kyselyprosessorinaan eXistiä varten kehitettyä XQuery engineä. XQuery engine pyrkii kyselyiden suorituksessa välttämään puiden ylhäältä alas- ja alhaalta ylös -tyyppisiä selauksia ja käyttämään näiden sijaan rakenneliitoksia, joiden suoritusjärjestys optimoidaan [Mei02].

eXistissä kaikki indeksit perustuvat B+-puihin. Kaikki elementtien, attribuuttien ja avainsanojen indeksit ovat kokoelma-, ei dokumenttikohtaisia (kuva 16). Esimerkiksi `author`-elementit indeksoidaan tässä määrittelyssä koko kokoelman laajuisesti [Mei02].

eXistin tiedonsäilytys perustuu sivutettuun tiedostoon `dom.dbx`, jossa kaikki dokumentit talletetaan DOM-mallin mukaisessa muodossa. Kokoelmahierarkiatiedot säilytetään tiedostossa `collection.dbx`, rakenneindeksi tiedostossa `elements.dbx` ja tiedot tietokannan tietosisällön sanojen esiintymistä tiedostossa `words.dbx`.

Rakenne- ja elementti-indeksit luodaan eXist-järjestelmässä tietokannan kokoelmiin automaattisesti. Rakenne- ja elementti-indeksien avulla dokumentin solmut ovat

löydettävissä elementtitunnisteen ja elementin nimen avulla. eXistissä ei oletusarvoisesti luoda elementtien arvoihin perustuvia indeksejä, ja oletusarvoilla esimerkiksi kysely `//SPEECH[SPEAKER = "HAMLET"]` suoritetaan elementti-indeksiä käyttäen selaamalla kaikki `SPEAKER`-nimiset elementit.

eXistissä ei käytetä `create index` -tyyppisiä komentoja indeksien luontiin ja ylläpitoon, vaan indeksit määritellään kokoelmakohtaisissa XML-dokumenteissa, kuten kuvan 16 määrittelyssä. Siinä elementille `title` otetaan käyttöön Full Text -indeksi, elementeille `title` ja `author` merkkijonotyyppinen rakenne- ja sisältöindeksi, elementille `year` kokonaislukutyyppinen rakenne- ja sisältöindeksi ja elementeille `author` ja `title` n-grammi-indeksit [EX5].

```
<collection xmlns="http://exist-db.org/collection-config/1.0">
    <index>
        <!-- New full text index based on Lucene -->
        <lucene>
            <text qname="title"/>
        </lucene>
        <!-- Range indexes -->
        <create qname="title" type="xs:string"/>
        <create qname="author" type="xs:string"/>
        <create qname="year" type="xs:integer"/>
        <!-- N-gram indexes -->
        <ngram qname="author"/>
        <ngram qname="title"/>
       </index>
   </collection>
```

Kuva 16: indeksien mahdollinen määrittely eXist-tietokannanhallintajärjestelmässä.

eXist luo ja ylläpitää aina rakenneindeksin elementti- ja attribuuttisolmuille. Rakenneindeksiä ei voi poistaa käytöstä. eXist myös luo ja ylläpitää aina elementtien nimiin perustuvaa indeksiä, jota ei myöskään voi poistaa käytöstä.

Rakenne- ja sisältöindeksiä kutsutaan eXistissä nimellä ”Range index”. Rakenne- ja sisältöindeksi luodaan kullekin eri nimiselle elementille erikseen, ja se indeksoi kaikki

kyseisen nimisen elementin esiintymät ja niihin viittaavat erilaiset polut elementin arvon ja polkurakenteen mukaisesti. Rakenne- ja sisältöindeksi luodaan esimerkiksi määrittelyllä `<create qname="title" type="xs:string"/>`. Rakenne- ja sisältöindeksiä käytetään sekä elementin arvoon että arvoväliin perustuvien kyselyiden suorituksessa. eXistin sisältöindeksi tyypittää indeksoitavat elementit indeksiin indeksissä käytettävän tietotyypin perusteella. Esimerkiksi elementti `<price>200</price>` muutetaan `double`-tietotyypin mukaisessa indeksoinnissa muotoon `<price>200.00</price>`. Sama muunnos suoritetaan hakujen suorituksessa.

eXistissä n-grammi-indeksointi määritellään elementtikohtaisesti. Esimerkiksi elementille `speaker` indeksi otetaan käyttöön määrittelyllä `<ngram qname="speaker"/>` . Full Text -indeksi toteutetaan eXist-järjestelmässä Apache Lucene-sovelluskehykseen perustuvalla indeksointimoduulilla. Moduuli on integroitu eXistin modulaariseen indeksiarkkitehtuuriin, jossa erilaiset indeksit toimivat liitännäisinä, joita voidaan käyttää toisistaan riippumattomasti. Kun indeksi on määritelty, kaikki tietokantapäivitykset, kuten kokoelmien lisääminen ja elementtien muutokset, näkyvät indeksille. Lucene-indeksin käyttö on automaattisesti mahdollista kaikissa uusissa eXist-versioissa.

6.2 BaseX

BaseX on XML-pohjainen alustariippumaton XML-pohjainen tietokannanhallintajärjestelmä. BaseX kehitettiin Saksassa Konstanzin yliopiston tutkimusprojektissa, jossa tutkittiin puurakenteisen tiedon tehokasta tallennusta. Osoittautui, että BaseX:n tallennusjärjestelmä mahdollisti tehokkaan tiedonhaun ja visuaalisen esittämisen, kun tietokantaan luetteloitiin XML-muodossa tiedot Konstanzin yliopiston kirjaston aineistosta. Tämän jälkeen BaseX:n kehitystyö jatkui, ja vuodesta 2007 BaseX on ollut vapaasti saatavilla BSD-lisensoituna ja avoimeen lähdekoodiin perustuvana. BaseX:n ydin on kirjoitettu kokonaan Java-ohjelmointikielellä. BaseX-tietokannanhallintajärjestelmää voidaan käyttää paitsi itsenäisenä sovelluksena graafisella käyttöliittymällä, jonka avulla on mahdollista mm. tarkastella dokumentteja tarkennettavana graafisena esityksenä, myös tietokantapalvelimena, jonka käyttämiseksi asiakassovelluksesta käsin on toteutettu ajurit 15:lle eri ohjelmointikielelle, kuten Java, C, Haskell, Perl, PHP, Ruby ja Scala [Mah10].

BaseX-tietokannanhallintajärjestelmässä ylläpidetään automaattisesti rakenneindeksiä, polkuindeksiä ja kokoelmissa sijaitsevat dokumentit paikantavaa resurssi-indeksiä. Sisältö- ja Full Text -indeksit ovat valinnaisia [Base].

BaseX-tietokannanhallintajärjestelmä luo ja ylläpitää automaattisesti rakenneindeksin tietokannan dokumenteille. Rakenneindeksiä ei voi poistaa käytöstä. Polkuindeksi on myös aina käytössä, ja siinä indeksoidaan kaikki erilaiset dokumentin elementtipolut. BaseX käyttää polkuindeksiä kyselyjen uudelleenmuotoiluun, esim. kysely `doc('factbook.xml')//province` voidaan muotoilla muotoon `doc('factbook.xml')/mondial/country/province`, jos elementti `province` esiintyy vain elementin `country` lapsena. XPath-lausekkeen `count(doc('factbook')//country)` tulos taas saadaan suoraan polkuindeksin tilastotietojen perusteella. Polkuindeksin sisältöä voidaan tarkastella XQuery-funktion `index:facets()` avulla.

Resurssi-indeksi indeksoi tietokannan kaikki dokumentit, ja sitä ylläpidetään automaattisesti. Esimerkiksi operaation `db:open('DatabaseWithLotsOfDocuments')` suorituksessa käytetään resurssi-indeksiä. Sisältöindeksiä kutsutaan BaseX-tietokannanhallintajärjestelmässä nimellä ”Text index”. Sisältöindeksi on merkkijonopohjainen ja se indeksoi kaikki XML-tietokannan elementtien ja attribuuttien sisällöt. Sisältöindeksi luodaan automaattisesti, mutta käyttäjällä on mahdollisuus poistaa se käytöstä. Sisältöindeksi nopeuttaa sisältöpredikaattiin perustuvien kyselyjen kuten `doc('factbook.xml')//country[name = 'Germany']` suoritusta. Myös arvovälikyselyiden kuten `db:open('Library')//Medium[Year >= '2005' and Year <= '2007']` suorituksessa voidaan käyttää sisältöindeksiä. Sen sijaan esimerkiksi kyselyn `db:open('Cars')//Model[Price >= '3000' and Price <= '20000']` suorituksessa ei voida käyttää apuna sisältöindeksiä, koska sisältöindeksi on merkkijonopohjainen.

BaseX:n Full Text -indeksi on valinnainen, ja voidaan ottaa käyttöön joko graafisesta käyttöliittymästä tai komentoriviltä esim. komentosarjalla `SET FTINDEX true; CREATE DB input.xml CREATE INDEX fulltext.` Full Text -indeksi nopeuttaa sekä ”`contains`”-tyyppisten osamerkkijonokyselyjen, kuten `//country/name[text() contains text 'and']` että monipuolisempien merkkijonokyselyiden, kuten `//religions[. contains text { 'Catholic', 'Roman' } using case insensitive distance at most 2 words]` suoritusta [Grü09].

7 XMark-koetinkuorma

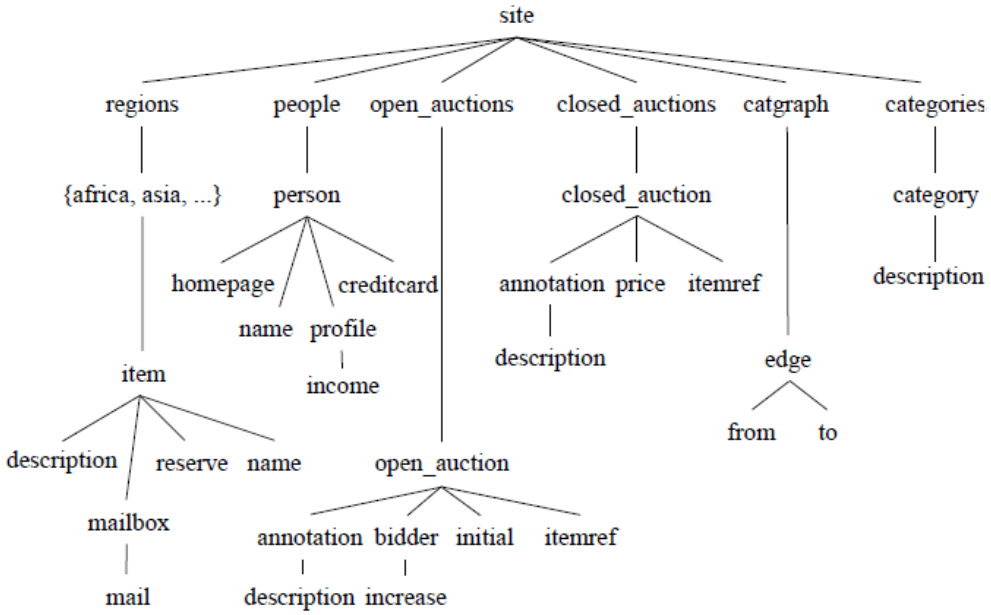
XMark on XML-tietokannanhallintajärjestelmien suorituskykyä mittaava koetinkuorma. XMark koostuu XMark-tietokantakuvauksesta, XMLgen-tietokantageneraattorista ja XMark-testikyselyjoukosta [Sch02].

Alankomaalaisessa Centrum Wiskunde & Informatica -tutkimuslaitoksessa kehitetyssä XMark-koetinkuormassa keskitytään XML-tietokannanhallintajärjestelmän ydintoimintojen suorituskyvyn mittaamiseen. Näitä ovat kyselyprosessorin toiminta ja tietokantapuskurin ja tallennusjärjestelmän välinen tiedonvaihto. XML-dokumentteja käsittelevien järjestelmien suorituskyvyn kannalta keskeisen tiedon hajauttamisen, verkkoliikenteen ja siihen liittyvien rajapintojen ja tiedonsiirtokerrosten toimintaa ei XMarkissa mitata. Suorituskyvyn mittaamisen lisäksi XMarkia voidaan käyttää myös tietokantaprosessorin toiminnan validointiin. XMarkia voidaan käyttää sekä XML-pohjaisten että XML:ää tukevien tietokannanhallintajärjestelmien testaamiseen [Sch02].

7.1 XMark-tietokantakuvaus

XMarkin tietokantakuvauksessa (kuva 17) kuvataan kansainvälisen, useassa maanosassa toimivan huutokauppayrityksen toimintaa. Huutokauppayritys ylläpitää tietoa toimipisteistä, huutokaupattavista kohteista, asiakkaista, kohteiden myyjistä, käynnissä olevista ja päättyneistä huutokaupoista sekä kohteisiin tehdyistä huudoista. Huutokaupat perustuvat ns. englantilaisen huutokaupan periaatteeseen, jossa kohde myydään korkeimman huudon tekijälle. Kohteilla on lähtöhinta, ja kohteen myyjä voi määritellä ns. hintavarauksen, jota alittavalla hinnalla kohdetta ei myydä. Tietokantakuvaus perustuu rakenteellisuuteen ja pitkälle vietyyn tyypitykseen, poikkeuksena `annotation`- ja `description`-elementit, joiden sisältö voi olla vapaamuotoista, tyypittämätöntä tekstiä [Sch02].

Tietokantakuvaus on suunniteltu niin, että sitä käyttäen voidaan laatia tietokannanhallinnanjärjestelmän toimintaa monipuolisesti testaavia kyselyitä. Kuvauksen suunnittelussa on myös pyritty siihen, että se muistuttaisi toimivan organisaation tietojärjestelmää, mikä kannustaisi koetinkuorman käyttöön ja tekisi siitä tiiviin ja helposti omaksuttavan [Sch02].



Kuva 17: huutokauppatietokannan rakennekuvaus [Sch02].

7.2 XMLgen-tietokantageneraattori

XMLgen on XMark-projektin kehittämä tietokantageneraattoriohjelma, joka tuottaa XMark-tietokantakuvauksen mukaisia XML-dokumentteja. Tuotettavan tietokannan kuvaustekstielementeissä käytetään sanastona William Shakespearen kirjoittamien näytelmien sanastoa, ja muissa kentissä, kuten nimissä ja sähköpostiosoitteissa, oikean muotoisia, Internetistä haettuja tietoja. Sovellus on pyritty pitämään mahdollisimman yksinkertaisena, ja luotavan tietokannan koko on ainoa käyttäjän määriteltävissä oleva parametri. XMLgenin avulla voidaan tuottaa sekä pieniä että suuria, useiden gigatavujen kokoisia testitietokantoja. XMLgen on toteutettu C-ohjelmointikielellä, joten se on alus-

tariippumaton. XMLgen tuottaa samoilla syöteparametreilla aina sisällöltään yhtäläisen testitietokannan, joten generaattori on deterministinen, eikä luotavaan tietokantaan liity satunnaisuutta. XMLgenillä tuotettavan tietokannan koko määritellään lineaarisesti skaalausparametrillä *f,* jonka arvolla 1 tuotetaan sadan megatavun kokoinen testitietokanta [Sch02], [CWI].

7.3 XMark-tietokantakyselyt

XMark-koetinkuormaan kuuluu 20 tietokantakyselyä (liite 1), joista kukin on suunniteltu testaamaan yhtä tai muutamaa keskeistä tietokannanhallinnan suorituskykyyn liittyvää ominaisuutta kuten indeksien käyttö, kyselyn uudelleenmuotoilu, liitosten järjestyksen valinta ja tulosten lajittelu [Sch02]. XMark-kyselyt on suunniteltu testaamaan monipuolisesti XML-tietokantaprosessorin toiminnallisuutta ulottuen merkkijonohauista monimutkaiseen data-analyysiin ja erilaisiin tilapäiskyselyihin. Kyselyjä suunniteltaessa on pyritty siihen, että ne ovat tietokannanhallintajärjestelmille kohtuullisen vaikeita, mutta pääsääntöisesti kuitenkin suoritettavissa. Kyselyistä tarkastellaan muutamaa, painottaen kyselyitä, jotka mittaavat tietokannan indeksoinnin osuutta suoritusnopeudessa. Kyselyiden suomenkieliset kuvaukset esitetään taulukossa 5.

Tunnus	Kuvaus
Q1	Nimet henkilölle, jonka tunnus on 'person0'
Q2	Käynnissä olevien huutokauppojen avaushuudot
Q3	Niiden käynnissä olevien huutokauppojen avaushuudot ja voittavat huudot, joissa voittava huuto on suuruudeltaan vähintään kaksinkertainen avaushuutoon verrattuna
Q4	Niiden huutokauppojen hintavaraukset, joissa tietty henkilö on tehnyt huudon ennen tiettyä toista henkilöä
Q5	Niiden myytyjen esineiden lukumäärä, joiden loppuhinta oli 40 tai enemmän
Q6	Listattujen kohteiden lukumäärät maanosittain
Q7	Tietokannassa olevien kuvausten yhteislukumäärä
Q8	Asiakkaiden nimet ja heidän ostamiensa kohteiden lukumäärät
Q9	Henkilöiden nimet ja heidän Euroopasta ostamansa kohteet
Q10	Kiinnostuskohdeluokat asiakasrekisterissä ja niihin liittyvät asiakkaat
Q11	Kaikki asiakkaat ja niiden kohteiden lukumäärä, joiden lähtöhinta ei ylitä 0,02 % asiakkaan tuloista
Q12	Määritellyn tulorajan ylittävät asiakkaat ja niiden kohteiden lukumäärä, joiden lähtöhinta ei ylitä 0,02 % asiakkaan tuloista
Q13	Australiassa listatut kohteet sekä niiden nimet ja kuvaukset
Q14	Niiden kohteiden nimet, joiden kuvauksessa esiintyy sana ”gold”
Q15	Suljettuihin kohteisiin liittyvien kommenttien korostetut avainsanat
Q16	Myyjien tunnukset niissä suljetuissa huutokaupoissa, joihin liittyy yksi tai useampia korostettuja avainsanoja
Q17	Asiakkaat, joilla ei ole kotisivua
Q18	Käynnissä olevien huutokauppojen hintavaraukset ilmaistuna toisessa valuutassa
Q19	Huutokauppakohteet sijaintimaan mukaan järjestettynä
Q20	Tuloryhmittäiset asiakasluokat ja niihin kuuluvien asiakkaiden määrä

Taulukko 5: XMark-testikyselyiden kuvaukset (suomennettu lähteestä [CWI2]).

Kysely Q2 on suunniteltu mittaamaan suoritusnopeutta kyselyssä, jossa elementin lapsi-elementtejä käsitellään järjestettyinä indeksoidun taulukon kautta. XML-pohjaiselle tietokannanhallintajärjestelmille kysely on helpompi käsitellä tallennusmuodon osalta, sillä niissä lapsielementit ovat yleensä tallennettu elementtitunnisteiden mukaisessa järjestyksessä, toisin kuin esim. XML:ää tukevissa

relaatiotietokannanhallintajärjestelmissä. Lapsielementin järjestysnumeron määrittely yleisesti on myös monimutkainen tehtävä sekä XML-pohjaisille että XML:ää tukeville tietokannanhallintajärjestelmille.

Kysely Q6 mittaa mm. tietokannanhallintajärjestelmän kykyä käyttää polkukuvausta, elementti-indeksiä ja rakenneindeksin tunnisteita polkulausekkeen esiintymien löytämiseksi. Lisäksi kyselyllä testataan tietokantaprosessorin kykyä havaita, että kyselyssä kysytään elementtien kokonaislukumäärää eikä näiden sisältöä, ja rajata selattavaa dokumentin osaa tämän perusteella.

Kyselyt Q8–Q12 ovat liitoskyselyitä, joissa kyselyt koostuvat sisäkkäisistä silmukoista, joissa osassa alikyselyissä myös tuotetaan uusia tietorakenteita. Näistä kyselyistä kyselyt Q11 ja Q12 ovat tietokannanhallintajärjestelmille erityisen vaikeita suoritettavia, sillä niissä on iteratiivisia kyselyn osia, jotka tuottavat dokumenteista uusia tietorakenteita, joihin toiset kyselyn osat taas tekevät hakuja. Tämä tekee hakutuloksia mahdollisimman aikaisessa vaiheessa rajaavien kokonaisvaltaisten oksaliitosalgoritmien käytön vaikeaksi, ja jos näiden kyselyiden suoritukseen käytetään rakenneliitosalgoritmeja, niin nämä kyselyt voivat tuottaa suuria välituloksia, joiden käsittelyä varten tarvitaan runsaasti muistia ja laskentakapasiteettia [Bon06].

Kyselyssä Q14 mitataan osamerkkijonoon perustuvan sisältöhaun ja dokumentin rakenteeseen perustuvan polkuhaun yhdistelmän suoritusta. Mikäli tietokantaprosessori osaa liittää polkuindeksin ja osamerkkijonoindeksin tuottamat tulokset keskenään, tai kykenee käyttämään rakenne- ja sisältöindeksiä tehokkaasti, nopeuttaa tämä kyselyn suoritusta.

Kyselyssä Q15 mitataan tietokannanhallintajärjestelmän kykyä käsitellä suuresta määrästä askelia koostuva XPath-lauseke. Koska kyselyssä määritellään ainoastaan rakenteeseen, ei sisältöön, kohdistuvia predikaatteja, testaa se erityisesti dokumentin rakenteen indeksoinnin toimintaa. Polkuindeksi on tehokas tämänlaisen kyselyn suorituksessa. Polkuindeksit mahdollistavissa järjestelmissä voidaan myös testata, kuin-

ka hyvin tietokannanhallintajärjestelmä kykenee käsittelemään pitkiä polkulausekkeita, mikäli polkuindeksiä ei ole määritelty kyselyssä määritellylle polulle.

Kysely Q19 mittaa tietokantaprosessorin kykyä lajitella hakutulokset. Lisäksi kyselyssä mitataan tietokannanhallintajärjestelmän kykyä suorittaa kyselyn uudelleenmuotoilu esimerkiksi yhdistämällä kaksi XPath-lauseketta yhdeksi, mikä helpottaa indeksien käyttöä ja poistaa välitulosten käsittelyn tarpeen.

Kaikissa koetinkuorman kyselyissä testataan tietokannanhallintajärjestelmän kykyä käyttää tietokantakuvausta apuna kyselyiden suorituksessa. Huutokauppatietokannan tietokantakaavio voidaan määritellä esim. W3C:n määrittelemää XML Schema -määrittelyä käyttäen, ja validoida huutokauppatietokantatietokannan sisältö käyttäen tehtyä määrittelyä. Tietokantakaavion määrittelyä voidaan käyttää apuna tietokantakyselyiden suorituksessa, sillä tietokannanhallintajärjestelmä voi tietokannan sallitun rakenteen ja sisällön perusteella rajata tehtävää kyselysuunnitelmaa ja sen osia sulkemalla pois sellaisia vaihtoehtoja, jotka eivät voi tietokantakuvauksen perusteella olla mahdollisia käsiteltävässä dokumentissa. Lisäksi tietokantakuvauksessa mahdolliset elementtien tietotyyppimäärittelyt auttavat tietokannanhallintajärjestelmää käsittelemään tehokkaasti esim. kyselyn Q5 kaltaisia kyselyitä, joissa arvopredikaattina on luku. Tietokantakuvauksen avulla voidaan suorituskyvyn parantamisen lisäksi myös tarkistaa käsiteltävän XPath-kyselylausekkeen oikeellisuus. Mikäli lauseke ei ole kuvauksen mukainen, voidaan tästä ilmoittaa käyttäjälle, mikä auttaa välttämään kirjoitusvirheiden aiheuttamia tahattomia tyhjiä tulosjoukkoja [Sch02].

8 eXist- ja BaseX-tietokannanhallintajärjestelmien nopeusmittaus

eXist- ja BaseX-tietokannanhallintajärjestelmät testataan suorittamalla niissä koko XMark-koetinkuorman 20 kyselyn kyselyjoukko kahdella erikokoisella tietokannalla. Molempien tietokannanhallintajärjestelmien suorituskyky mitataan käyttäen kahta eri laajuista indeksointia. Suppea indeksointi suoritetaan käyttämällä vain järjestelmän pakollisia indeksejä, ja laajassa indeksoinnissa taas määritellään ja otetaan käyttöön kaikki mahdolliset indeksit, joiden voidaan olettaa nopeuttavan tietokannanhallintajärjestelmän toimintaa tietokantakyselyitä suoritettaessa. Testin tarkoituksena on mitata indeksoinnin laajuuden vaikutusta XML-tietokannanhallintajärjestelmien suorituskykyyn yleensä. Suppeaa indeksointia käytettäessä BaseX-järjestelmässä on käytössä enemmän indeksejä kuin eXistissä, minkä takia järjestelmien keskinäisen suorituskyvyn vertailu ei ole tässä tilanteessa mielekästä. Suppean indeksoinnin määrittelytapa valittiin tällaiseksi, koska näin eXist tarjoaa mahdollisuuden testata tietokannanhallintajärjestelmän suorituskykyä, kun käytössä ei ole minkäänlaista polkuindeksiä. Laajaa indeksointia käytettäessä myös järjestelmien nopeusvertailu on mahdollinen, koska kummankin järjestelmän indeksointi on toteutettu tehokkaimmalla mahdollisella tavalla. Indeksoinnin testaus testataan siis kolmella eri tasoisella indeksoinnilla: rakenneindeksi (eXist), rakenne- ja polkuindeksi (BaseX) ja rakenne-, polku- ja sisältöindeksi (eXist ja BaseX).

Koetinkuorman kyselyiden suoritusta varten luodaan XMark-projektin verkkosivulta ladatulla XMLgen-tietokantageneraattoriohjelmalla kaksi eri kokoista tietokantaa, joista pieni on yhden ja suuri sadan megatavun kokoinen. Tietokannat tuotetaan käyttämällä pienen tietokannan tuottamisessa skaalauskertointa *f=0.001* ja suuren tietokannan tuottamisessa kerrointa *f=1*. Pienen tietokannan käyttö mittaa korostetusti tietokannanhallintajärjestelmän kyselyoptimoijan nopeutta kyselyn suoritussuunnitelmien tekemisessä. Suuremman tietokannan koko valittiin 100 megatavuksi sen perusteella, että XMark-kyselyiden suoritus tämän kokoisessa tietokannassa on aiemmissa mittauksissa osoittautunut riittävän vaikeaksi tietokannanhallintajärjestelmille [Bon06], [Cam13]. Suurempaa tietokantaa käytettäessä kyselyoptimoijan nopeuden

merkitys pienenee ja tietosivujen haun, välitulosten liitosjärjestyksen valinnan ja välitulosten käsittelyn tehokkuuden merkitys kasvaa [Bon06] [Con12].

Koetinkuorman kaikki 20 kyselyä suoritettiin kaikissa kahdeksassa mittaussarjassa aloittamalla suoritus ns. ”kylmästä puskurista” käynnistämällä tietokonelaitteisto ja aloittamalla koetinkuorman kyselyiden suoritus tilanteesta jossa tietokannanhallintajärjestelmän puskurissa ei ole puskuroituja tietosivuja.

Mittaussarjassa suoritettiin kaikki XMark-koetinkuorman kyselyt niiden numeroinnin mukaisessa järjestyksessä tyhjentämättä puskuroituja tietosivuja kyselyiden suorituksen välillä. Tällä menetelmällä puskurin sisältö voi täyttyä kyselyiden suorituksen aikana. Huutokauppaorganisaation kaltaisen organisaation tietojärjestelmään tehdään jatkuvasti kyselyitä eri toimipisteistä ja verkkosovelluksista, joten tietokannanhallintajärjestelmän kyky nopeuttaa tulevia hakuja säilyttämällä tietosivuja puskurissa on tärkeä ominaisuus, jonka mittaaminen on oleellista. Kukin mittaussarja suoritettiin kaksi kertaa käynnistäen laitteisto uudelleen myös saman mittaussarjan kahden suorituskerran välillä. Kustakin mittaussarjasta valittiin kunkin kyselyn suoritusajaksi kahdesta suorituskerrasta nopeampi. Mikäli kyselyn suoritus ei ollut valmis puolen tunnin (1800 sekuntia) kuluttua, kyselyn suoritus keskeytettiin, ja mittauskerran tulokseksi määriteltiin ”ei valmistunut”. Testilaitteistona toimi Windows 7 -käyttöjärjestelmää käyttävä työasema, jossa oli Intel E4500 -kaksoisydinsuoritin ja 2 gigatavun kokoinen järjestelmämuisti. Muista kyselyistä poiketen taulukkoon on lisätty luvussa 9.7 kuvatun mukaisesti kyselyn Q11 samaa mittausmenetelmää käyttäen saadut suoritusaikatulokset, kun kyselyn suorituksen aikarajaksi asetettiin 12 tuntia.

	Exist, suppea	Exist, laaja	BaseX, suppea	BaseX, laaja
Rakenneindeksi	x	x	x	x
Polkuindeksi		x	x	x
Elementti-indeksi	x	x		
Rakenne- ja sisältöindeksi		x		
Full Text -indeksi				x
N-grammi -indeksi		x		
Sisältöindeksi		x		x

Taulukko 6. Suppeassa ja laajassa indeksoinnissa käytetyt indeksit.

eXist-järjestelmän suorituskykyä mitattiin järjestelmän versiolla 2.0. Tietokannan määrittelytiedostossa `conf.xml` määriteltiin tietokantapuskurin koko kasvatettavaksi oletusasetusten 128 megatavusta 512 megatavuun ja kyselyiden uudelleenmuotoilu otettavaksi käyttöön.

Suppeassa ja laajassa indeksoinnissa käytetyt indeksit esitetään taulukossa 6. Suppea indeksointi toteutettiin oletusarvoisella indeksoinnilla, jossa oli määritelty rakenneindeksi ja elementti-indeksi. Laajassa indeksoinnissa taas indeksoitiin kaikki kyselypoluissa esiintyvät elementit ja attribuutit ”range index” -rakenne- ja sisältöindeksillä, sekä kyselyssä Q14 suoritettavaa osamerkkijonohakua varten elementille `description` luotiin vielä n-grammi-indeksi, jonka yleiseksi pituudeksi määriteltiin määrittelytiedostossa `conf.xml` arvo 4, koska kyselyssä esiintyvä osamerkkijono on neljän merkin pituinen. Full Text -indeksiä ei otettu käyttöön, sillä testikyselyjoukossa ei arvioitu olevan kyselyitä, joiden suoritusta Full Text -indeksin käyttö nopeuttaisi, koska käytössä on jo rakenne- ja sisältöindeksi sekä n-grammi-indeksi.

Kuvassa 18 esitetään esimerkki muutamasta eXistin indeksoinnissa käytetystä indeksointimäärittelystä. eXistissä indeksoitavat elementit ja attribuutit indeksoidaan aina

erikseen, ja XMark-huutokauppatietokannan kaltaisessa dokumentissa indeksejä tarvitaan jo useita kymmeniä. Kyselyt suoritettiin käyttämällä eXistin web-pohjaista eXide-käyttöliittymää. Koska eXistin kyselyoptimoija voi käyttää lukuarvoisten elementtien ja attribuuttien indeksiä vain, mikäli kaikki vertailuun osallistuvat luvut ovat muodoltaan samaa tietotyyppiä, suoritettiin eXistissä kyselyiden versioita, joissa tehdään tyyppimuunnoksia, kuten kuvassa 19.

```
<collection xmlns="http://exist-db.org/collection-config/1.0">
    <index>
       ...
       <create qname="auction" type="xs:string"/>
       <create qname="increase" type="xs:double"/>
       <create qname="@person" type="xs:string"/>
       <create ngram="description"/>
    </index>
</collection>
```

Kuva 18: eXist-järjestelmän nopeusmittauksessa käytettyjen indeksien määrittelyjä.

```
let $auction := doc("auction.xml") return
count(
  for $i in $auction/site/closed_auctions/closed_auction
  where $i/price >= xs:double(40)
  return $i/price
)
```

Kuva 19: eXist-järjestelmän indeksihakua varten muokattu XMark-kysely Q5.

```
let $auction := doc('auction') return
for $b in $auction/site/people/person[@id = "person0"] return
$b/name/text()
```

Kuva 20: BaseX-järjestelmän indeksihakua varten muokattu XMark-kysely Q1.

BaseX-järjestelmä testattiin käyttäen versiota 7.6 järjestelmän oletusasetuksilla. Testit suoritettiin käyttäen BaseX:n graafista käyttöliittymää. Suppeassa indeksoinnissa poistettiin oletusarvoiset elementtien ja attribuuttien sisältöindeksit käytöstä, jolloin käyttöön jäivät vielä rakenne- ja polkuindeksit, joita ei voi poistaa käytöstä. Laaja indeksointi taas toteutettiin ottamalla oletusarvoiset elementtien ja attribuuttien sisältöindeksit käyttöön. Full Text -indeksi otettiin myös käyttöön, koska sen oletettiin nopeuttavan osamerkkijonoperustaisen kyselyn Q14 suoritusta, ja koska BaseX-järjestelmässä ei ole käytettävissä toista tähän tarkoitukseen sopivaa indeksiä, n-grammi-indeksiä. BaseX:ssä indeksit ovat kokoelmakohtaisia, ja koko kokoelma indeksoidaan käyttöön otettavan indeksityypin mukaisesti.

9 Mittaustulokset

Tässä luvussa käsitellään XMark-koetinkuorman kyselyitä suoritettaessa saatuja mittaustuloksia. Mittausten yhteenveto esitetään luvuissa 9.1–9.3. Indeksoinnin ja tietokannan koon vaikutusta suoritusnopeuteen käsitellään luvuissa 9.4 ja 9.5. Suoritusaikojen merkitystä tietokannanhallintajärjestelmän käytettävyydelle taas käsitellään luvussa 9.6. Koetinkuormaa suoritettaessa kysely Q11 osoittautui tietokannanhallintajärjestelmille erityisen vaikeaksi. Tätä kyselyä ja sen suoritusta käsitellään luvussa 9.7.

9.1 Tulosten yhteenveto

XMark-koetinkuorman kyselyiden mittaustulokset BaseX- ja eXist-tietokannanhallintajärjestelmissä esitetään taulukossa 7. Kyselyiden suoritusajat ilmoitetaan sekunteina, ja ”-”-merkki suoritusajan kohdalla merkitsee, että kyselyn tulos ei ollut valmistunut puolen tunnin kuluessa kyselyn suorittamisen aloittamisesta. Taulukossa 8 esitetään tiivistelmä indeksoinnin lisäämisen vaikutuksesta hakujen suoritusnopeuteen.

	1 megatavu				100 megatavua			
	eXist, suppea	eXist, laaja	BaseX, suppea	BaseX laaja	eXist, suppea	eXist, laaja	BaseX, suppea	BaseX, laaja
Q1	0,198	0,021	0,162	0,144	1,592	0,515	0,136	0,039
Q2	0,089	0,028	0,026	0,018	4,900	3,902	0,266	0,238
Q3	0,028	0,010	0,024	0,019	1,759	1,375	0,445	0,436
Q4	0,015	0,033	0,021	0,013	5,743	1,719	0,284	0,286
Q5	0,105	0,033	0,013	0,006	1,956	1,207	0,135	0,119
Q6	0,185	0,046	0,016	0,008	0,032	0,124	0,275	0,271
Q7	0,162	0,195	0,025	0,024	0,379	0,395	2,293	2,256
Q8	0,918	0,760	0,197	0,067	-	-	-	3,740
Q9	1,488	1,527	0,226	0,084	-	-	-	516,946
Q10	0,369	0,390	0,064	0,053	-	55,838	212,363	8,090
Q11	1,735	1,744	0,444	0,537	-	-	-	4780,758
Q12	1,321	1,349	0,133	0,150	-	-	1385,680	1319,353
Q13	0,041	0,025	0,019	0,011	2,007	0,161	0,947	0,615
Q14	0,020	0,016	0,037	0,003	11,896	1,340	4,964	3,218
Q15	0,061	0,056	0,009	0,002	0,627	0,314	0,508	0,089
Q16	0,031	0,005	0,008	0,002	3,416	1,348	0,093	0,094
Q17	0,015	0,023	0,009	0,003	5,906	0,985	0,286	0,212
Q18	0,008	0,016	0,013	0,006	6,695	3,259	0,254	0,256
Q19	0,116	0,113	0,020	0,012	38,941	6,881	0,796	0,828
Q20	0,038	0,077	0,017	0,013	13,552	12,711	0,444	0,424

Taulukko 7: XMark-koetinkuorman kyselyiden suoritusajat sekunteina eXist- ja BaseX-tietokannanhallintajärjestelmissä 1 ja 100 megatavun kokoisissa tietokannoissa suppeaa ja laajaa indeksointia käyttäen.

9.2 Tulokset pienessä testitietokannassa

Pienikokoisella tietokannalla eXist kykeni suorittamaan suurimman osan kyselyistä alle sekunnissa, ja kaikki kyselyt alle kahdessa sekunnissa (taulukko 7). BaseX taas kykeni suorittamaan kaikki kyselyt alle yhdessä sekunnissa.

9.3 Tulokset suuressa testitietokannassa

Suuremman kokoisella tietokannalla kyselyiden suoritusajat kasvoivat pienikokoiseen tietokantaan verrattuna sekä laajaa että suppeaa indeksointia käytettäessä (taulukko 7). Osaa kyselyistä ei kyetty suorittamaan kummallakaan järjestelmällä laajaakaan indeksointia käyttäen.

9.4 Indeksoinnin vaikutus kyselyiden suoritusnopeuteen

Indeksoinnin vaikutus kyselyiden suoritusnopeuteen esitetään taulukossa 8. Pienempikokoisella tietokannalla eXist-järjestelmässä kahdeksan kyselyn suoritusnopeus kasvoi indeksoinnin käyttöä laajentamalla ja BaseX-järjestelmässä 18 kyselyn nopeus kasvoi indeksien käyttöä kasvattamalla. Syynä siihen, että BaseX kykeni paremmin hyödyntämään indeksejä pienikokoisessa tietokannassa, on ilmeisesti sen kyky luoda kyselysuunnitelma eXistiä nopeammin. Kyselysuunnitelman luomiseen käytettävän ajan osuus tietokantakyselyn kokonaissuoritusajasta on suhteellisesti suurempi pienessä kuin suuressa tietokannassa [Sel79].

Suurempikokoisessa tietokannassa kolme neljästä kyselystä pystyttiin suorittamaan nopeammin kummallakin järjestelmällä ottamalla laaja indeksointi käyttöön. Lisäksi sekä BaseX- että eXist-järjestelmä kykenivät laajan indeksoinnin avulla suorittamaan XMark-kyselyitä, joita ne eivät suppeaa indeksointia käyttämällä kyenneet suorittamaan lainkaan. Myös nämä kyselyiden suoritukset luetaan nopeutuneiksi.

	Laajan indeksoinnin nopeuttamat haut (1 megatavu)	Laajan indeksoinnin nopeuttamat haut (100 megatavua)
BaseX	18/20	15/20
eXist	8/20	15/20

Taulukko 8: indeksien käyttöä lisäämällä saavutettu XMark-kyselyiden nopeutuminen.

9.5 Tietokannan koon vaikutus kyselyiden suoritusnopeuteen

Tietokantakyselyiden nopeuden riippuvuutta tietokannan koosta voidaan mitata skaalauskertoimella, joka määrittelee kyselyn suoritusajan kasvun suhteessa tietokannan koon kasvuun [Wei11]. Koska luvun 8 testissä suurempi tietokanta oli kooltaan sata kertaa pienemmän tietokannan kokoinen, merkitsee skaalauskerroin 100 tietokantakyselyn suoritusajan täydellistä lineaarista riippuvuutta tietokannan koon kanssa, ts. kyselyn suoritusaika kasvaa samassa suhteessa kuin tietokannan koko. Tätä suuremmat arvot merkitsevät suoritusajan kasvavan tietokannan kokoa nopeammin, ja pienemmät arvot vastaavasti hitaammin. XMark-koetinkuormatestin laajaa indeksointia käyttävien kyselyiden suoritusaikojen skaalauskertoimet esitetään taulukossa 9. Kyselyt Q8, Q9, Q11, Q12 ja Q14 erottuvat muista, koska näissä ainakin toisessa järjestelmässä kyselyä ei voitu suorittaa alle tuhatkertaisessa ajassa pienempään tietokantaan verrattuna.

XMark-kysely	eXist	BaseX
Q1	24,523	0,271
Q2	139,357	13,222
Q3	137,500	22,947
Q4	52,091	22,000
Q5	36,576	19,833
Q6	2,696	33,875
Q7	2,026	94,000
Q8		55,821
Q9		6154,119
Q10	143,174	152,642
Q11		8902,715
Q12		8795,687
Q13	6,440	55,909
Q14	83,750	1072,667
Q15	5,607	44,500
Q16	269,600	47,000
Q17	42,826	70,667
Q18	203,688	42,667
Q19	60,894	69,000
Q20	165,078	32,615

Taulukko 9: XMark-kyselyiden suoritusajan skaalauskerroin suppean ja laajan tietokannan välillä eXist- ja BaseX-järjestelmissä.

9.6 XMark-kyselyiden suoritusaikojen tarkastelua

Card ja kumppanit [Car91] luokittelevat käyttöliittymäsovelluksen vasteajat kolmeen luokkaan. Alle 0,1 sekunnin vasteajalla käyttäjä kokee käsittelevänsä suoraan käyttöliittymässä olevia kohteita ilman vasteaikaa. 0,1 – 1 sekunnin vasteajan käyttäjä huomaa, mutta tällainen vasteaika ei vielä häiritse käyttöä. 1 – 10 sekunnin vasteaika tarkoittaa

sitä että käyttäjä kokee joutuvansa odottamaan, ja yli 10 sekunnin vasteaika on jo niin pitkä, että se haittaa käyttäjän keskittymistä tehtäväänsä.

Kumpikin testattava tietokannanhallintajärjestelmä pystyy suuressakin tietokannassa suorittamaan suhteellisen nopeasti sellaisia XMark-koetinkuorman kyselyitä, joissa XPath-polkumäärittelyihin ja elementtien arvoihin perustuen haetaan olemassa olevan dokumentin osia rajaamalla. Tällöin suoritusajat ovat niin nopeita, että tällaisia kyselyitä voidaan, muiden verkkosovelluksen osien tämän mahdollistaessa, suorittaa verkkosovelluksessa vasteaikojen olematta liian pitkiä käyttäjille [Car91]. Sen sijaan kyselyissä, joissa kyselyn osat luovat dokumentista uusia tietorakenteita, joihin toiset kyselyn osat vielä kohdistavat kyselyitä, pitkät suoritusajat rajoittavat jo kyselyiden käyttöä verkkosovelluksissa. Tällaisia kyselyitä voidaan suorittaa esim. sovelluksissa, joissa palvelimet vaihtavat tietoa keskenään.

9.7 XMark-koetinkuorman kyselyn Q11 tarkastelua

XMark-koetinkuorman kysely Q11 osoittautui luvun 8 mittauksessa poikkeuksellisen vaikeaksi suoritettavaksi. eXist- ja BaseX-tietokannanhallintajärjestelmissä kyselyä ei pystytty suorittamaan puolen tunnin määräajassa sadan megatavun kokoisessa testitietokannassa. Myös muissa aiemmassa nopeusmittauksessa kysely Q11 on osoittautunut vaikeaksi suoritettavaksi useille XML-tietokannanhallintajärjestelmille, kuten eXistille [Bon06]. Koetinkuormatestissä mitattujen pitkien suoritusaikojen perusteella tätä kyselyä sekä sen suoritussuunnitelman toteutusta tarkastellaan tarkemmin. Kyselyn loogista rakennetta tarkastellaan XQGM-mallin avulla ja sen fyysistä suoritussuunnitelmaa tarkastellaan BaseX-tietokannanhallintajärjestelmässä. Lisäksi kyselyn suoritusaika mitattiin uudelleen BaseX- ja eXist-järjestelmissä väljentämällä puolen tunnin aikarajoitusta ja kokeilemalla kyselyn uudelleenmuotoiluun perustuvia vaihtoehtoisia suoritussuunnitelmia. Kysely Q11 on käsiteltäviltä tiedoiltaan ja rakenteeltaan huomattavan samankaltainen kyselyn Q12 kanssa, joten kyselyn Q11 analyysi pätee monilta osin myös kyselyyn Q12.

XMark-koetinkuorman kyselyssä Q11 haetaan huutokauppatietokannasta asiakastietoja ja yhdistetään kunkin asiakkaan nimeen näihin niiden avoimien huutokauppakohteiden lukumäärä, joissa kohteen avaushinta ei ylitä 0,02 % kyseisen henkilön tuloista. Kyselyä Q11 ei voida suorittaa pelkästään rajaamalla huutokauppadokumenttia XPath-lausekkeen avulla, kuten kyselyssä Q1, vaan kyselyssä Q11 luodaan huutokauppadokumentin alipuiden tietoja lomittamalla kokonaan uusi tietorakenne, jonka XML schema -kuvaus esitetään kuvassa 21. Kyselyn tulos saadaan lomittamalla XPath-lausekkeen `/site/people/person/profile/name`

määrittelemien elementtien sisältämät henkilöiden nimitiedot ja XPath-lausekkeen `/site/open_auctions/open_auction/initial`

määrittelemistä elementeistä löytyvät huutokauppakohteiden lähtöhintatiedot. Lähtöhintatietoja ei viedä kyselytulokseen sellaisenaan, vaan lähtöhintatiedoista lasketaan kokonaislukumääriä funktion `count` avulla. Henkilö- ja huutokauppakohdetietojen lomittamisessa predikaattina toimii ehto

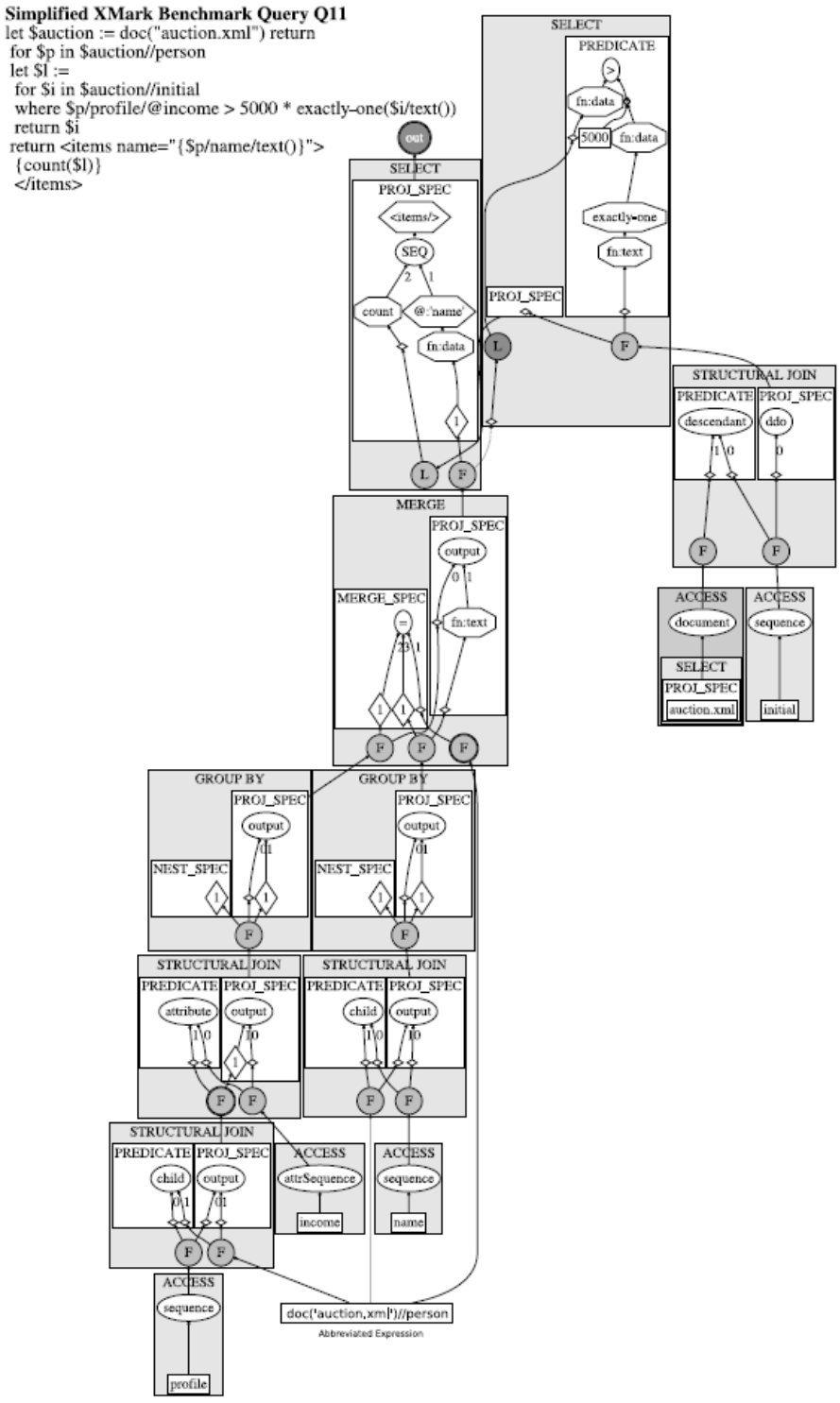
```
/site/people/person/profile/@income <
/site/open_auctions/open_auction[initial*5000].
```

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="items" type="xs:integer"/>
      <xs:attribute name="name" type="xs:string" use="required"/>
   </xs:element>
</xs:schema>
```

Kuva 21: XMark-kyselyn Q11 palauttaman tuloksen XML schema -kuvaus.

Kuvassa 22 esitetään XMark-kysely Q11 XQGM-mallin avulla. Kuva on tuotettu XTC-tietokannanhallintajärjestelmän kyselysuunnitelman visualisoijan avulla [Wei10]. XQGM-mallissa kysely kuvataan loogisella tasolla tietohakujen ja operaattorien muodostamana verkkona, eikä mallissa oteta huomioon esim. tiedon tallennusmuotoa tai indeksointia. Kukin XQGM-verkon operaattori saa syötteenä ja palauttaa tuloksena joukon monikoita. XQGM-mallin operaattorit ovat `ACCESS`, `SELECT`, `STRUCTURAL_JOIN`,

SPLIT, MERGE, SET, GROUP-BY, UNNEST, HTJ (Holistic twig join), REVERSE ja TUP_VAR_REF. XTC-tietokannanhallintajärjestelmässä XQGM-verkko annetaan syötteenä kyselysuunnitelman laatijalle ja optimoijalle, jotka tekevät verkon pohjalta suoritussuunnitelman luvussa 5.2 esitetyllä tavalla. Kuvan verkossa ACCESS-solmuissa kuvataan tiedon hakua tallennuspaikasta, STRUCTURAL_JOIN-solmuissa rakenneliitoksia, MERGE-solmuissa lomitusta, GROUP-BY-solmuissa lajittelua ja SELECT-solmuissa projektiota. F-kirjain tarkoittaa mallissa XQuery-kielen FOR-silmukkaa ja L-kirjain LET-lauseella määriteltävää sidontaa.



Kuva 22: XMark-koetinkuorman kyselyn Q11 esitys XQuery Graph Model -verkkona [Wei10].

Kuvan 22 XQGM-verkossa on XQuery-kyselyn logiikan kannalta merkittävää vasemmalla ylhäällä esitettävä `SELECT`-operaattori, jonka F-kirjaimella merkitystä silmukasta kulkee katkoviivalla esitettävä yhteys keskellä ylhäällä esitettävään `SELECT`-operaattoriin. Katkoviivalla kuvataan kontekstiriippuvaa alipuun evaluointia. Tässä tapauksessa evaluoitava alipuu muodostuu huutokaupattavien esineiden lähtöhinnoista, ja on kontekstissa henkilö- ja tulotietoja sisältävään alipuuhun. XQGM-malli määrittelee siis kyselyn olevan luvussa 5.1 esitetyllä tavalla iteratiivinen.

Kun kysely Q11 suoritetaan BaseX-järjestelmällä laajassa testitietokannassa laajaa indeksointia käyttäen, järjestelmä tarkastaa kyselyn XQuery-syntaksin mukaisuuden ja suorittaa kyselyn uudelleenmuotoilun tuottaen kuvassa 23 esitetyt ilmoitukset. Uudelleenmuotoiltu kysely esitetään kuvassa 24. Uudelleenmuotoillulle kyselylle laaditaan XML-muodossa esitettävä suoritussuunnitelma, joka esitetään kuvassa 25. Suunnitelmassa luodaan kaksi `IterPath`-elementtiä XPath-polkulauseen evaluointiin, ensimmäinen uudelleenmuotoillun kyselyn `FOR`-lauseessa kuvattavalle henkilötietojen haulle ja jälkimmäinen `LET`-lauseessa kuvattavalle huutokauppakohteiden tietojen haulle. Jälkimmäisen `IterPath`-elementin sisällä olevassa `VarRef`-elementissä kuvataan polkulauseen evaluoinnin tapahtuvan kontekstissa ensimmäisessä `IterPath`-elementissä esiteltyyn lausekkeeseen `$p`. Tämä merkitsee, että kyselyn suoritustapa on taulukon 4 kohdassa f esitettävä navigaatiopuu, ja suorituskustannukset määräytyvät kohdan tämän mukaan. Suoritettaessa kysely navigaatiopuuna, käsitellään kaikki huutokauppatietokannan `person`-elementit ja kutakin `person`-elementtiä kohden haetaan silmukkana kaikki `initial`-elementit, jotka täyttävät `person`-elementin `income`-attribuutin perusteella määräytyvän predikaatin ehdot.

Kyselyn Q11 tuloksen laskennan kannalta keskeiset testitietokannan tunnusluvut on haettu testitietokannasta XPath-lausekkeiden avulla BaseX-järjestelmää käyttäen ja ne esitetään taulukossa 8. Taulukon perusteella nähdään elementtien lukumäärän skaalautuvan lineaarisesti tietokannan koon kasvaessa ja olevan suuressa tietokannassa noin satakertainen pieneen tietokantaan verrattuna. Koska kyselyn Q11 looginen rakenne on kuvassa 22 esitetyn kaltaisesti kontekstiriippuva ja suoritussuunnitelma kuvassa 25 esi-

tetyn kaltaisesti iteratiivinen ja silmukkamainen, merkitsee tämä karteesisen tulon käyttöä välituloksen laskennassa. Tämän vuoksi sekä henkilöiden ja huutokauppakohteiden määrän satakertaistuminen merkitsee lopputuloksen laskennan kannalta oleellisten henkilö- ja huutokauppatietojen yhdistelmien eksponentiaalista kasvua taulukossa 10 esitetyn mukaisesti, mikä kasvattaa käsiteltävien välitulosten määrän hyvin suureksi suurikokoisissa tietokannoissa ja kasvattaa kyselyiden suoritusaikoja.

```
Compiling:
- pre-evaluating fn:doc("standard")
- binding static variable $auction
- rewriting where clause to predicate(s)
- simplifying flwor expression
- removing variable $auction
- simplifying flwor expression
```

Kuva 23: BaseX-järjestelmän suoritussuunnitelman luonnin vaiheet kyselylle Q11.

```
Optimized Query:
for $p in document-node { "standard.xml" }/*:site/*:people/*:person
let $l := document-node
{ "standard.xml" }/*:site/*:open_auctions/*:open_auction/*:initial[($p
/profile/@income > (5000 * fn:exactly-one(text())))]
return element items { (attribute name { ($p/name/text()) },
fn:count($l)) }
```

Kuva 24: BaseX-järjestelmän uudelleenmuotoilema kysely Q11.

```
Query plan:
<QueryPlan>
  <FLWR>
    <For var="$p">
      <IterPath>
        <DBNode name="standard" pre="0"/>
        <IterStep axis="child" test="*:site"/>
        <IterStep axis="child" test="*:people"/>
        <IterStep axis="child" test="*:person"/>
      </IterPath>
    </For>
    <Let var="$l">
      <IterPath>
        <DBNode name="standard" pre="0"/>
        <IterStep axis="child" test="*:site"/>
        <IterStep axis="child" test="*:open_auctions"/>
        <IterStep axis="child" test="*:open_auction"/>
        <IterStep axis="child" test="*:initial">
          <CmpG op="&gt;">
            <AxisPath>
              <VarRef>
                <Var name="$p" id="1"/>
              </VarRef>
              <IterStep axis="child" test="profile"/>
              <IterStep axis="attribute" test="income"/>
            </AxisPath>
            <Arith op="*">
              <Int value="5000" type="xs:integer"/>
              <FNSimple name="exactly-one(item)">
                <AxisPath>
                  <IterStep axis="child" test="text()"/>
                </AxisPath>
              </FNSimple>
            </Arith>
          </CmpG>
        </IterStep>
      </IterPath>
    </Let>
```

```
    <Return>
      <CElem>
        <QNm value="items" type="xs:QName"/>
        <CAttr>
          <QNm value="name" type="xs:QName"/>
          <AxisPath>
            <VarRef>
              <Var name="$p" id="1"/>
            </VarRef>
            <IterStep axis="child" test="name"/>
            <IterStep axis="child" test="text()"/>
          </AxisPath>
        </CAttr>
        <FNAggr name="count(item)">
          <VarRef>
            <Var name="$l" id="2"/>
          </VarRef>
        </FNAggr>
      </CElem>
    </Return>
  </FLWR>
</QueryPlan>
```

Kuva 25: BaseX-tietokannanhallintajärjestelmän suoritussuunnitelma kyselylle Q11 isommassa testitietokannassa laajaa indeksointia käyttäen.

```
let $auction := doc('standard') return
for $p in $auction//person
let $r := ($p/profile/@income div 5000)
let $l := count($auction//open_auction[initial < $r])
return <items name="{$p/name}">{$l}</items>
```

Kuva 26: XMark-kyselyn Q11 uudelleen muotoiltu vaihtoehto 1.

```
let $auction := doc('standard') return
for $p in $auction//person
let $l := count($auction//open_auction[initial <
($p/profile/@income div 5000)])
return <items name="{$p/name}">{$l}</items>
```

Kuva 27: XMark-kyselyn Q11 uudelleen muotoiltu vaihtoehto 2.

XMark-kyselyn Q11 osoittauduttua vaikeasti suoritettavaksi luvussa 8 esiteltyä mittausmenetelmää käyttäen, luotiin nyt kyselyn lisätestausta varten myös kuvissa 26 ja 27 esitettävät kaksi vaihtoehtoista versiota, jotka testitietokannassa tuottavat alkuperäisen kyselyn kanssa ekvivalentit tulokset. Kummassakin vaihtoehdossa kyselyn sisäkkäistä hierarkiaa madallettiin niin että `for`-lauseiden määrä väheni kahdesta yhteen. `exactly-one` -funktion käytöstä luovuttiin koemielessä, koska pienessä ja laajassa testitietokannassa jokaisella `open_auction`-elementillä on tasan yksi `initial`-alielementti. Tämä oletus ei kuitenkaan päde XMLgen-generaattorilla luoduissa tietokannoissa yleisesti. Lisäksi elementtien sisällön tarkastamiseen käytetään XPath-lauseketta ilman alkuperäisessä kyselyssä käytettävää `text`-funktiota. Kuvan 26 kyselyssä henkilön tulotietoihin tehdään sidonta `let`-lauseella sen kokeilemiseksi, vaikuttaako jakolaskun alkuperäinen paikka silmukkarakenteen sisimmässä silmukassa välituloksien määrää kasvattamalla hidastavasti tulotietojen yhdistelyyn henkilötietoihin.

	Pieni tietokanta	Suuri Tietokanta
Henkilöitä	255	25504
Henkilöiden mediaanitulo	40142,50	40142,50
Avoimia kohteita	120	12000
Kohteita, jotka täyttävät mediaanituloehdon	12	925
Henkilöiden ja mediaani-tuloehdon täyttävien kohteiden lukumäärien tulo	3060	23591200

Taulukko 10: huutokauppatietokannan haun Q11 suorituksen keskeisiä tunnuslukuja.

XMark-kysely Q11 ja sen vaihtoehtoiset versiot suoritettiin laajassa testitietokannassa laajaa indeksointia käyttäen luopumalla puolen tunnin aikarajoituksesta ja mittaamalla suoritusajat. Nyt kaikkien kolmen kyselyn Q11 version suoritus valmistui BaseX-järjestelmässä alle kahdessa tunnissa, ja erot suoritusaikojen välillä olivat pienehköjä. eXist-järjestelmässä näiden kyselyiden suoritus ei ollut valmistunut 12 tunnin määräajassa. Suoritusajat esitetään taulukossa 11.

Suoritettava kysely	suoritusaika
Alkuperäinen kysely Q11	4780,758
Kyselyn Q11 uudelleenmuotoiltu vaihtoehto 1	4732,154
Kyselyn Q11 uudelleenmuotoiltu vaihtoehto 2	5427,027

Taulukko 11: XMark-kyselyn Q11 eri versioiden suoritusajat sekunteina BaseX-järjestelmässä, isommassa testitietokannassa ja laajalla indeksoinnilla.

Kyselyn Q11 eri versioiden suorittamiseen kuluva aika on varsin pitkä. Mittaustuloksia tarkasteltaessa on huomioitavaa, että sekä eXist- että BaseX-tietokannanhallintajärjestelmille varattiin niin paljon muistia tietokannan tietosivujen puskurointia varten, että sekä pieni että suuri testitietokanta oli mahdollista tuoda kokonaan tieto- ja indeksisivuina tietokannanhallintajärjestelmän puskuriin. Tämän ja sen perusteella, että kyselyä Q11 ei kyetty kummallakaan järjestelmällä suorittamaan alle puolessa tunnissa, voidaan päätellä, että tietokantakyselyiden suorittamisessa tietosivujen sisällön ja käsittelyalgoritmien tietorakenteiden välisestä linkityksestä, liitoksista, elementtien predikaattien perusteella tapahtuvasta valinnasta ja välitulosten käsittelystä johtuvien laskentakustannusten osuus voi kyselyn suorituksessa olla hyvinkin merkittävä, sillä kaikki tietokannan tietosivut olisivat olleet siirrettävissä tallennusjärjestelmästä puskuriin murto-osassa tästä ajasta. Tämän perusteella tiedonsiirtokustannus ei voi selittää kyselyn pitkiä suoritusaikoja.

Weiner toteaa XMark-kyselyiden suoritusaikojen XTC-järjestelmässä kasvavan maltillisesti tietokannan koon kasvaessa, kyselyiden Q11 ja Q12 muodostaessa tästä poikkeuksen suoritusaikojen kasvaessa eksponentiaalisesti suurten välitulosten takia [Wei11]. Weinerin mukaan kyselyiden Q11 ja Q12 pitkät suoritusajat XTC-järjestelmässä eivät johtuneet tietokannanhallintajärjestelmän virheestä vaan kyselyiden loogisista ominaisuuksista, jotka määrittelevät tulosjoukon niin että sen saamiseksi on tehtävä liitoksia, jotka eivät ole rajaavia. BaseX- ja eXist-tietokannanhallintajärjestelmillä nyt tehdyt XMark-koetinkuormatestit tuottavat samanlaisen tuloksen.

10 Yhteenveto

Verkkosovelluksissa, tiedonvaihdossa ja rakenteisissa dokumenteissa yleistynyt XML-tietomallin käyttö on kasvattanut tarvetta XML-muotoisen tiedon säilyttämiseen tietokannoissa. XML-muotoista tietoa voidaan säilyttää esimerkiksi relaatiotietokannoissa, mutta XML-tietomallia sekä tiedon varastoinnin että käsittelyn perusyksikkönä käyttävä XML-pohjainen tietokanta mahdollistaa parhaiten XML-kyselykielten käytön, säilytettävien dokumenttien rakenteessa tapahtuvat muutokset sekä XML-dokumenttien indeksoinnin ja tähän perustuvan tallentamisen.

XML-dokumenttien käsittelyssä erityispiirteenä esimerkiksi relaatiotietokantoihin verrattuna on XML-dokumenttien monipuolinen rakenne, joka voi myös alati muuttua, ja XML-dokumenttien laaja käyttö Internetissä tapahtuvassa tiedonvaihdossa. Kumpikin näistä lisää tarvetta XML-dokumenttien ja niihin tehtävien hakujen tehokkaaseen ja nopeaan käsittelyyn. XML-tietokantojen tehokkuutta voidaan parantaa dokumenttien indeksoinnilla. Dokumenttien indeksointi perustuu yleensä Dewey-indeksointiin, joka kuvaa dokumentin elementtien hierarkkisen rakenteen ja järjestyksen kokonaisluvuilla ja niitä erottavilla pisteillä.

XML-dokumenttien säilytys perustuu yleensä tietokannan tietosivuihin, joilla tietokannan tieto talletetaan elementtitunnisteisiin perustuvassa järjestyksessä. Erilaisia tapoja toteuttaa tämä järjestys ovat mm. elementtitunnisteiden mukaan nouseva järjestys, elementtien nimien ja hierarkkisen aseman mukaan tapahtuva ryvästäminen sekä elementtien tallettaminen pinopuihin hajautusavaimen perusteella. XML-dokumenttien säilytystapa vaikuttaa siihen, minkälaisia indeksejä ja käsittelyalgoritmeja niiden käsittelyyn voidaan käyttää.

Elementtitunnisteiden nousevaan järjestykseen tai ryvästämiseen perustuvassa dokumentin tallennuksessa tallennus perustuu B-puihin ja indekseinä voidaan käyttää

dokumentin elementtien rakenteen kuvaavaa rakenneindeksiä, elementin polkurakenteen kuvaavaa polkuindeksiä sekä elementtien sisällön indeksoivaa sisältöindeksiä. Indeksi voi olla myös näiden yhdistelmä, kuten sisällön ja polkurakenteen yhdistävä rakenne- ja sisältöindeksi. B-puihin perustuvassa tallennuksessa haut dokumenteista suoritetaan yleensä perustuvina yhtenä tai useampana XPath-lausekkeen askeliin kohdistuvina osakyselyinä, joiden tulokset liitetään yksi kerrallaan toisiinsa tulosjoukkoa rajaavilla rakenneliitoksilla. Tämä tapa muistuttaa paljon relaatiotietokannan tapaa suorittaa tietokantakysely liitoksina. Rakenneliitosten käytön haittapuolena on se, että suoritettaessa kysely pienissä osissa ei täysimääräisesti voi käyttää tietoa koko XPath-lausekkeen rakenteesta, mikä toisinaan johtaa tiettyjen solmujen karsimiseen lopputuloksesta tarpeettoman myöhään. Tämä voi johtaa suuriin välituloksiin ja tarpeettomaan tiedon siirtämiseen tietokantajärjestelmän puskuriin.

Hajautusavaimeen ja pinopuihin perustuvassa indeksonnissa tieto indeksoidaan hajautusavaimen perusteella määräytyvissä listoissa, jotka indeksoidaan B+-puussa. Hajautusavaimeen perustuvassa indeksoinnissa voidaan kyselyiden suorittamiseen käyttää kokonaisvaltaisia oksaliitosalgoritmeja, joissa pinopuita lomitetaan yhtäaikaisesti ja kyselyn lopputuloksen kannalta tarpeettomat solmut karsitaan välituloksesta mahdollisimman aikaisessa vaiheessa.

Indeksoinnin toimintaa ja sen vaikutusta tietokannanhallintajärjestelmän suorituskykyyn tarkasteltiin BaseX- ja eXist-tietokannanhallintajärjestelmillä, joissa kummassakin tiedon tallennus ja käsittely perustuu elementtitunnisteiden mukaan järjestettyihin B+-puihin, tiedon sivutukseen ja rakenneliitosalgoritmien käyttöön. Näiden tietokannanhallintajärjestelmien suorituskykyä mitattiin suorittamalla XMark-koetinkuorman 20 testikyselyä XMark-koetinkuorman rakennekuvauksen mukaiseen huutokauppa-aiheiseen tietokantaan. Testikyselyt suoritettiin sekä kuvauksen mukaiseen pieneen että suureen tietokantaan, jotta voitiin mitata myös tietokannan koon vaikutusta kyselyiden suoritusnopeuteen. Pelkkää rakenneindeksiä (eXist) käyttämällä kyselyitä pystyttiin suorittamaan pääsääntöisesti sujuvasti pienessä, 1 megatavun kokoisessa tietokannassa, mutta suuremmassa tietokannassa hakujen suoritusajat kasvoivat liian pitkiksi esimerkiksi verkkosovelluksissa toteutettaviksi. Rakenne- ja polkuindeksiä käyttämällä

(BaseX) suurin osa XMark-testikyselyistä pystyttiin suorittamaan alle sekunnissa suuressakin testitietokannassa. Ottamalla kaikki mahdolliset testitietokantaan tehtäviä hakuja nopeuttavat indeksit käyttöön, pystyttiin kummassakin järjestelmässä suorittamaan kyselyitä huomattavasti nopeammin kuin suppeampaa indeksointia käytettäessä. Rakenneindeksin käytön todettiin tekevän yksinkertaisten kyselyiden suorittamisen pienikokoiseen tietokantaan mahdolliseksi, ja muiden indeksien lisäämisen tehostavan kyselyiden suoritusta merkittävästi. Indeksoinnin avulla voidaan nopeuttaa hakuja ja mahdollistaa nopeat haut erityisesti silloin kun haut vastaavat käytössä olevia indeksejä tai ovat osittuvia. Tällaisia hakuja ovat mm. haut, jotka perustuvat yhteen XPath-lausekkeeseen, kuten luvuissa 7 ja 8 käsitelty XMark-koetinkuorman kysely Q1. Sen sijaan esimerkiksi XMark-kyselyn Q11 kaltaisten iteroituvien XQuery-kyselyiden, joissa yhden osakyselyn tulokset toimivat syötteenä toiselle osakyselylle, tehokas suoritus vaatii indeksoinnin lisäksi kyselyoptimoijaa, joka pystyy tuottamaan tehokkaita suoritussuunnitelmia.

Erilaisia tapoja XML-tietokannanhallintajärjestelmien suorituskyvyn kasvattamiseen on esitetty. Nopeampien SSD-levyjen käyttö tiedon tallennukseen mahdollistaisi tietosivujen tehokkaamman siirron puskurin ja tallennusjärjestelmän välillä [Sch09]. Verkkopalveluiden tarpeisiin voidaan tietokannanhallintajärjestelmiä paljon kuormittavien kyselyjen, kuten XMark-koetinkuorman kyselyn Q11, tai niiden osien tuloksia laskea valmiiksi materialisoituihin näkymiin. Tätä menetelmää voidaan käyttää varsinkin tapauksissa, joissa tieto ei päivity usein tai näkymien tietojen ei aina tarvitse olla aivan ajan tasalla [Sha09]. Laskentatehon ja käytettävissä olevan muistin määrää kasvattamalla voidaan kasvattaa tietokannanhallintajärjestelmän suorituskykyä, kuten myös laitteistokiihdytyksellä, jossa laskentatehtäviä suoritetaan erikoiskomponentilla yleissuorittimen sijaan [Kri12]. Oksaliitosalgoritmien tutkimusta on tehty paljon, ja kehittämällä niiden käytännön toteutusta on myös mahdollista tehostaa XML-tietokannanhallintajärjestelmien toimintaa [Wei10].

Lähteet

Bac12 Bača, Radim & Krátký, Michal. 2012. XML query processing: efficiency and optimality. IDEAS '12: Proceedings of the 16th International Database Engineering & Applications Symposium, 8–13.

Base BaseX GmbH. Indexes. http://docs.basex.org/wiki/Indexes. 26.4.2013.

Bid13 Bidoit, Nicole, Colazzo, Dario, Malla, Noor, Ulliana, Federico, Nolè, Maurizio & Sartiani, Carlo. 2013. Processing XML queries and updates on map/reduce clusters. EDBT '13 Proceedings of the 16th International Conference on Extending Database Technology, 745–748.

Bon06 Boncz, Peter, Grust, Torsten, van Keulen, Maurice, Manegold, Stefan, Rittinger, Jan & Teubner, Jens. 2006. MonetDB/XQuery: a fast XQuery processor powered by a relational engine. SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data, 479–490.

Bru02 Bruno, Nicolas, Koudas, Nick & Srivastava, Divesh. 2002. Holistic twig joins: optimal XML pattern matching. SIGMOD '02 Proceedings of the 2002 ACM SIGMOD international conference on Management of data. 310–321.

Cam13 Camacho-Rodríguez, Jesús, Colazzo, Dario & Manolescu, Ioana. 2013. Web data indexing in the cloud: efficiency and cost reductions. EDBT '13 Proceedings of the 16th International Conference on Extending Database Technology, 41 –52.

Car91 Card, Stuart K., Robertson, George G. & Mackinlay, Jock D. 1991. The information visualizer, an information workspace. CHI '91 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 181–186.

Con12 Cong, Gao, Fan, Wenfei, Kementsietsidis, Anastasios, Li, Jianzhong & Liu, Xianmin. 2012. Partial Evaluation for Distributed XPath Query Processing and Beyond. Transactions on Database Systems (TODS), 37 (4).

CWI Centrum Wiskunde & Informatica. XMark-An XML Benchmark Project. http://www.xml-benchmark.org. 26.4.2013.

CWI2 Centrum Wiskunde & Informatica. XMark test queries.

http://www.ins.cwi.nl/projects/xmark/Assets/xmlquery.txt. 26.4.2013.

EX eXist Solutions GmbH. Two Indexing Schemes for XML Documents. http://exist-db.org/exist/apps/doc/xmlprague06.xml. 26.4.2013.

EX2 eXist Solutions GmbH. Facts. http://exist-db.org/exist/apps/doc/facts.xml. 26.4.2013.

EX3 eXist Solutions GmbH. Roadmap. http://exist-db.org/exist/apps/doc/roadmap.xml. 26.4.2013.

EX4 eXist Solutions GmbH. Tuning the Database. http://exist-db.org/exist/apps/doc/tuning.xml 26.4.2013.

EX5 eXist Solutions GmbH. Understanding the New Indexing Features. http://atomic.exist-db.org/blogs/eXist/NewIndexing. 26.4.2013.

Gue08 Gueni, Bilel, Abdessalem, Talel, Cautis, Bogdan & Waller, Emmanuel. 2008. Pruning nested XQuery queries. CIKM '08 Proceedings of the 17th ACM conference on Information and knowledge management, 541–550.

Guh06 Guha, Sudipto, Jagadish, H.V., Koudas, Nick, Srivastava, Divesh & Yu, Ting. 2006. Integrating XML data sources using approximate joins. ACM Transactions on Database Systems 31 (1), 161–207.

Gri10 Grimsmo, Nils, Bjørklund, Truls A. & Hetland, Magnus Lie. 2010. Fast optimal twig joins. Proceedings of the VLDB Endowment 3 (1–2), 894–905.

Gri11 Grijzenhout, Steven, & Marx, Maarten. 2011. The quality of the XML web. CIKM '11 Proceedings of the 20th ACM international conference on Information and knowledge management, 1719–1724.

Grü09 Grün, Christian, Gath, Sebastian, Holupirek, Alexander & Scholl, Marc H. 2009. XQuery Full Text Implementation in BaseX. XSym '09 Proceedings of the 6th International XML Database Symposium on Database and XML Technologies, 114–128.

Hau10 Haustein, Michael P., Härder, Theo, Mathis, Christian & Wagner, Markus. 2010. DeweyIDs - The Key to Fine-Grained Management of XML. Journal of Information and Data Management 1 (1), 147–160.

Hel94 Helman, Paul. 1994. The science of database management. Burr Ridge (IL): Irwin.

Hug89 Hughes, John G. 1988. Database technology : a software engineering approach. New York: Prentice Hall.

Jen11 Jensen, Simon Holm, Madsen, Magnus & Møller, Magnus. 2011. Modeling the HTML DOM and browser API in static analysis of JavaScript web applications. ESEC/FSE '11 Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering, 59–69.

Jia05 Jiao, Enhua, Ling, Tok Wang, Chan, Chee-Yong. 2005. Pathstack¬: A holistic path join algorithm for path query with not-predicates on XML data. In proceeding of: Database Systems for Advanced Applications, 10th International Conference, DASFAA 2005, Beijing, China.

Jun12 Junedi, Muhammad, Genevès, Pierre & Layaïda, Nabil. 2012. XML query-update independence analysis revisited. DocEng '12: Proceedings of the 2012 ACM symposium on Document engineering, 95–98.

Kha02 Al-Khalifa, Shurug, Jagadish, H. V., Koudas, Nick, Patel, Jignesh M., Srivastava, Divesh & Yuqing, Wu. 2002. Structural Joins: A Primitive for Efficient XML Query Pattern Matching. ICDE02, 141–152.

Koh06 Koch, Cristoph. 2006. On the complexity of nonrecursive XQuery and functional query languages on complex values. Transactions on Database Systems (TODS), 31 (4), 1215–1256.

Kri12 Krishna, Anil, Heil, Timothy, Lindberg, Nicholas, Toussi, Farnaz & VanderWiel, Steven. 2012. Hardware acceleration in the IBM PowerEN processor: architecture and performance. PACT '12 Proceedings of the 21st international conference on Parallel architectures and compilation techniques, 389–400.

Luk07 Lukichev, Maxim & Barashev, Dmirty. 2007. XML Query Algera for Cost-based Optimization. Proceedings of the SYRCODIS'07 Colloquium on Databases and Information Systems, SYRCoDIS 2007, Moscow, Russia, June 2007.

Luk12 Lukichev, Maxim, Novikov, Boris & Mehra, Pankaj. 2012. An XML-algebra for efficient set-at-a-time execution. Computer Science and Information Systems 9 (1), 63–80.

Mah12 Mahlow, Cerstin, Grün, Christian, Holupirek, Alexander & Scholl, Marc H. 2012. A framework for retrieval and annotation in digital humanities using XQuery full text and update in BaseX. DocEng '12 Proceedings of the 2012

ACM symposium on Document engineering, 195–204.

Mei02 Meier, Wolfgang. 2002. eXist: An Open Source Native XML Database. Revised Papers from the NODe 2002 Web and Database-Related Workshops on Web, Web-Services, and Database Systems. London: Springer-Verlag, 169–183.

Mor12 Morelli, Eduardo, Almeida, Ana, Lifschitz, Sérgio, Monteiro, José Maria & Machado, Javam. 2012. Autonomous re-indexing. SAC '12: Proceedings of the 27th Annual ACM Symposium on Applied Computing, 893–897.

Mou09 Du Mouza, Cédric, Litwin, Witold, Rigaux, Philippe & Schwarz, Thomas. 2009. AS-index: a structure for string search using n-grams and algebraic signatures. CIKM '09 Proceedings of the 18th ACM conference on Information and knowledge management, 295–304.

Pir92 Pirahesh, Hamid, Hellerstein, Joseph M. & Hasan, Waqar. 1992. Extensible/rule based query rewrite optimization in Starburst. SIGMOD '92: Proceedings of the 1992 ACM SIGMOD international conference on Management of data, 39–48.

Sal01 Salminen, Airi & Tompa, Frank Wm. Requirements for XML document database systems. 2001. DocEng '01 Proceedings of the 2001 ACM Symposium on Document engineering, 85–94.

Sch02 Schmidt, Albrecht, Waas, Florian, Kersten, Martin, Carey, Michael J., Manolescu, Ioana & Busse, Ralph. 2002. XMark: A Benchmark for XML Data Management. VLDB'02: Proceedings of the 28th international conference on Very Large Data Bases, 974–985.

Sch09 Schmidt, Karsten, Ou, Yi & Härder, Theo. The promise of solid state disks: increasing efficiency and reducing cost of DBMS processing. C3S2E '09 Proceedings of the 2nd Canadian Conference on Computer Science and Software Engineering, 35–41.

Sel79 Selinger, P. G., Astrahan, M. M., Chamberlin, D. D., Lorie, R. A. & Price, T. G. 1979. Access Path Selection in a Relational Database Management System. Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data, 23–34.

Sha09 Shao, Feng, Guo, Lin, Botev, Chavdar, Bhaskar, Anand, Chettiar, Muthiah, Yang, Fan & Shanmugasundaram, Jayavel. 2009. Efficient keyword search

over virtual XML views. The VLDB Journal — The International Journal on Very Large Data Bases 18 (2), 543–570.

Soc09 Sockut, Gary H. & Iyer, Balakrishna R. 2009. Online reorganization of databases. Computing Surveys (CSUR) 41 (3).

Tar10 Taranov, Ilya, Shcheklein, Ivan, Kalinin, Alexander, Novak, Leonid, Kuznetsov, Sergei, Pastukhov, Roman, Boldakov, Alexander, Turdakov, Denis, Antipin, Konstantin, Fomichev, Andrey, Pleshachkov, Peter, Velikhov, Pavel, Zavaritski, Nikolai, Grinev, Maxim, Grineva, Maria & Lizorkin, Dmitry. 2010. Sedna: native XML database management system (internals overview). SIGMOD '10 Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, 1037–1046.

W3Ca World Wide Web Consortium. What is the Document Object Model? http://www.w3.org/TR/DOM-Level-3-Core/introduction.html. 26.4.2013.

W3Cb World Wide Web Consortium. XQuery 1.0: An XML Query Language. http://www.w3.org/TR/xquery/. 26.4.2013.

W3Cc World Wide Web Consortium. XQuery and XPath Full Text 1.0. http://www.w3.org/TR/xpath-full-text-10/. 26.4.2013.

W3Cd World Wide Web Consortium. XQuery Update Facility 1.0. http://www.w3.org/TR/xquery-update-10/. 26.4.2013.

W3Ce World Wide Web Consortium. XQuery 1.0 and XPath 2.0 Data Model (XDM) (Second Edition). http://www.w3.org/TR/xpath-datamodel/. 20.5.2013.

Wei10 Weiner, Andreas M. & Härder, Theo. 2010. An integrative approach to query optimization in native XML database management systems. IDEAS' 10. Proceedings of the Fourteenth International Database Engineering & Applications Symposium, 64–74.

Wei11 Weiner, Andreas M. 2011. Advanced Cardinality Estimation in the XML Query Graph Model. Datenbanksysteme für Business, Technologie und Web (BTW), 14. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" (DBIS), 207–226.

Zen13 Zeng, Yong, Bao, Zhifeng & Ling, Tok Wang. 2013. Supporting range queries in XML keyword search. National University of Singapore. EDBT '13 Proceedings of the Joint EDBT/ICDT 2013 Workshops, 97–104.

Liite 1: XMark-koetinkuorman testikyselyjoukko

```
-- Q1.Return the name of the person with ID `person0'.

let $auction := doc("auction.xml") return
for $b in $auction/site/people/person[@id = "person0"] return $b/name/text()

-- Q2. Return the initial increases of all open auctions.

let $auction := doc("auction.xml") return
for $b in $auction/site/open_auctions/open_auction
return <increase>{$b/bidder[1]/increase/text()}</increase>

-- Q3. Return the IDs of all open auctions whose current
--     increase is at least twice as high as the initial increase.

let $auction := doc("auction.xml") return
for $b in $auction/site/open_auctions/open_auction
where zero-or-one($b/bidder[1]/increase/text()) * 2 <= $b/bidder[last()]/increase/text()
return
  <increase
  first="{$b/bidder[1]/increase/text()}"
  last="{$b/bidder[last()]/increase/text()}"/>

-- Q4. List the reserves of those open auctions where a
--     certain person issued a bid before another person.

let $auction := doc("auction.xml") return
for $b in $auction/site/open_auctions/open_auction
where
  some $pr1 in $b/bidder/personref[@person = "person20"],
       $pr2 in $b/bidder/personref[@person = "person51"]
  satisfies $pr1 << $pr2
return <history>{$b/reserve/text()}</history>

-- Q5.  How many sold items cost more than 40?

let $auction := doc("auction.xml") return
count(
  for $i in $auction/site/closed_auctions/closed_auction
  where $i/price/text() >= 40
  return $i/price
)

-- Q6. How many items are listed on all continents?

let $auction := doc("auction.xml") return
for $b in $auction//site/regions return count($b//item)

-- Q7. How many pieces of prose are in our database?}

let $auction := doc("auction.xml") return
for $p in $auction/site
return
  count($p//description) + count($p//annotation) + count($p//emailaddress)
```

```
-- Q8. List the names of persons and the number of items they bought.
--     (joins person, closed\_auction)}

let $auction := doc("auction.xml") return
for $p in $auction/site/people/person
let $a :=
  for $t in $auction/site/closed_auctions/closed_auction
  where $t/buyer/@person = $p/@id
  return $t
return <item person="{$p/name/text()}">{count($a)}</item>

-- Q9. List the names of persons and the names of the items they bought
--     in Europe.  (joins person, closed\_auction, item)}

let $auction := doc("auction.xml") return
let $ca := $auction/site/closed_auctions/closed_auction return
let
    $ei := $auction/site/regions/europe/item
for $p in $auction/site/people/person
let $a :=
  for $t in $ca
  where $p/@id = $t/buyer/@person
  return
    let $n := for $t2 in $ei where $t/itemref/@item = $t2/@id return $t2
    return <item>{$n/name/text()}</item>
return <person name="{$p/name/text()}">{$a}</person>

-- Q10. List all persons according to their interest;
--      use French markup in the result.

let $auction := doc("auction.xml") return
for $i in
  distinct-values($auction/site/people/person/profile/interest/@category)
let $p :=
  for $t in $auction/site/people/person
  where $t/profile/interest/@category = $i
  return
    <personne>
      <statistiques>
        <sexe>{$t/profile/gender/text()}</sexe>
        <age>{$t/profile/age/text()}</age>
        <education>{$t/profile/education/text()}</education>
        <revenu>{fn:data($t/profile/@income)}</revenu>
      </statistiques>
      <coordonnees>
        <nom>{$t/name/text()}</nom>
        <rue>{$t/address/street/text()}</rue>
        <ville>{$t/address/city/text()}</ville>
        <pays>{$t/address/country/text()}</pays>
        <reseau>
          <courrier>{$t/emailaddress/text()}</courrier>
          <pagePerso>{$t/homepage/text()}</pagePerso>
        </reseau>
      </coordonnees>
      <cartePaiement>{$t/creditcard/text()}</cartePaiement>
    </personne>
return <categorie>{<id>{$i}</id>, $p}</categorie>
```

```
-- Q11. For each person, list the number of items currently on sale whose
--      price does not exceed 0.02% of the person's income.

let $auction := doc("auction.xml") return
for $p in $auction/site/people/person
let $l :=
  for $i in $auction/site/open_auctions/open_auction/initial
  where $p/profile/@income > 5000 * exactly-one($i/text())
  return $i
return <items name="{$p/name/text()}">{count($l)}</items>

-- Q12.  For each richer-than-average person, list the number of items
--       currently on sale whose price does not exceed 0.02% of the
--       person's income.

let $auction := doc("auction.xml") return
for $p in $auction/site/people/person
let $l :=
  for $i in $auction/site/open_auctions/open_auction/initial
  where $p/profile/@income > 5000 * exactly-one($i/text())
  return $i
where $p/profile/@income > 50000
return <items person="{$p/profile/@income}">{count($l)}</items>

-- Q13. List the names of items registered in Australia along with
--      their descriptions.

let $auction := doc("auction.xml") return
for $i in $auction/site/regions/australia/item
return <item name="{$i/name/text()}">{$i/description}</item>

-- Q14. Return the names of all items whose description contains the
--      word `gold'.

let $auction := doc("auction.xml") return
for $i in $auction/site//item
where contains(string(exactly-one($i/description)), "gold")
return $i/name/text()

-- Q15. Print the keywords in emphasis in annotations of closed auctions.

let $auction := doc("auction.xml") return
for $a in
  $auction/site/closed_auctions/closed_auction/annotation/description/parlist/
   listitem/
   parlist/
   listitem/
   text/
   emph/
   keyword/
   text()
return <text>{$a}</text>
```

```
-- Q16. Return the IDs of those auctions
--      that have one or more keywords in emphasis. (cf. Q15)

let $auction := doc("auction.xml") return
for $a in $auction/site/closed_auctions/closed_auction
where
  not(
    empty(
      $a/annotation/description/parlist/listitem/parlist/listitem/text/emph/
       keyword/
       text()
    )
  )
return <person id="{$a/seller/@person}"/>

-- Q17. Which persons don't have a homepage?

let $auction := doc("auction.xml") return
for $p in $auction/site/people/person
where empty($p/homepage/text())
return <person name="{$p/name/text()}"/>

-- Q18.Convert the currency of the reserve of all open auctions to
--     another currency.

declare namespace local = "http://www.foobar.org";
declare function local:convert($v as xs:decimal?) as xs:decimal?
{
  2.20371 * $v (: convert Dfl to Euro :)
};

let $auction := doc("auction.xml") return
for $i in $auction/site/open_auctions/open_auction
return local:convert(zero-or-one($i/reserve))

-- Q19. Give an alphabetically ordered list of all
--      items along with their location.

let $auction := doc("auction.xml") return
for $b in $auction/site/regions//item
let $k := $b/name/text()
order by zero-or-one($b/location) ascending empty greatest
return <item name="{$k}">{$b/location/text()}</item>
```

```
-- Q20. Group customers by their
--      income and output the cardinality of each group.

let $auction := doc("auction.xml") return
<result>
  <preferred>
    {count($auction/site/people/person/profile[@income >= 100000])}
  </preferred>
  <standard>
    {
      count(
        $auction/site/people/person/
         profile[@income < 100000 and @income >= 30000]
      )
    }
  </standard>
  <challenge>
    {count($auction/site/people/person/profile[@income < 30000])}
  </challenge>
  <na>
    {
      count(
        for $p in $auction/site/people/person
        where empty($p/profile/@income)
        return $p
      )
    }
  </na>
</result>
```