Tietokuution muodostaminen heterogeenisista XML-dokumenteista

Turkka Näppilä

Tampereen yliopisto
Tietojenkäsittelytieteiden laitos
Tietojenkäsittelyoppi
Pro gradu -tutkielma
Helmikuu 2006

Tampereen yliopisto
Tietojenkäsittelytieteiden laitos
Tekijän nimi: Turkka Näppilä
Pro gradu -tutkielma, 84 sivua, 9 liitesivua
Helmikuu 2006

Tutkielmassa käsitellään tietokuution muodostamista rakenteeltaan heterogeenisista XML-dokumenteista. Tietokuutio on OLAP (Online Analytical Processing) -toiminnallisuuden toteutuksissa yleisesti käytettävä tietomalli ja tiedon visualisointitapa. OLAP tarkoittaa analyysia, jossa yksittäisistä tietoalkioista koostettuja yhteenvetotietoja tarkastellaan moniulotteisesti, so. samanaikaisesti usean, tavallisesti toisistaan riippumattoman dimension suhteen. Organisaatioiden ja yhteisöjen tuottamien jatkuvasti kasvavien tietomäärien vuoksi on ennakoitavissa, että tällainen tietojen analysointitapa on vastaisuudessa entistäkin relevantimpaa. Yhtenä syynä tietomäärien jatkuvaan kasvuun on ollut XML (Extensible Markup Language) -merkintäkielen nousu tietojen vaihdon standardiksi. XML on poistanut yhden keskeisimmistä modernin tietojenkäsittelyn aikana vaikuttaneista laajamittaisen tietojen vaihdon esteistä – tietoformaattien heterogeenisuuden. Nykyään voidaankin jo olettaa, että kaikista tietojärjestelmistä ovat tiedot saatavissa XML-muodossa. XML-esitystapa sallii kuitenkin dokumenttien välisen ja sisäisen rakenteellisen heterogeenisuuden. Kun enenevässä määrin siirrytään toimimaan globaalissa ympäristössä, ei enää voida olettaa, että käsiteltävien tietoaineistojen rakennetta voitaisiin hallita keskitetysti, vaan on kehitettävä uusia menetelmiä, joilla autonomisissa tietolähteissä olevia tietoja saadaan helposti integroitua. Tulevaisuudessa myös ad hoc -tyyppiset tiedontarpeet lisääntyvät, ja sen seurauksena syntyy tarve nykyistä nopeammille tiedon analysointivälineille.

Tutkielmassa esitellään teoreettisia ja käytännöllisiä haasteita, joita liittyy tietokuution muodostamiseen rakenteellisesti toisistaan eroavien XML-dokumenttien perusteella. Teoreettisen viitekehyksen tueksi tutkielman yhteydessä on toteutettu järjestelmäprototyyppi, jossa erikseen tarkoitusta varten kehitetyn kyselyprimitiivin avulla pystytään helposti integroimaan tietoja rakenteeltaan heterogeenisista XML-muotoisista tietolähteistä ja muodostamaan saatujen tietojen perusteella tietokuutioita. Kehitetty kyselyprimitiivi tarjoaa olemassa olevia XML-kyselykieliä intuitiivisemman tavan käsitellä XML-rakenteita. Kyselyprimitiivissä on helpotettu tarkoitusta varten kehitetyn erityisen suppeimman mahdollisen kontekstin semantiikan avulla XML-dokumenttien rakenteellisen tuntemattomuuden käsittelyä, mikä vähentää käyttäjän XML-dokumenteissa suorittaman eksplisiittisen navigoinnin tarvetta.

Avainsanat ja -sanonnat: tiedon integrointi, XML, OLAP, tietovarastointi, tiedonhaku.

Alkusanat

Suuri ansio tämän tutkielman valmistumisesta kuuluu FT Timo Niemelle, jonka herkeämättömän asialleen omistautumisensa innostamana tutkielman kirjoittajakin on vihdoin saattanut urakkansa päätökseen. Kiitos kuuluu myös muille tutkimusryhmäläisillemme, akatemiaprofessori Kalervo Järvelinille ja FM Janne Jämsenille, niistä lukuisista ideoista, joita olen tutkielmani suhteen saanut yhteisistä palavereistamme.

Tahdon kiittää lämpimästi myös Fysiikan tutkimuslaitosta (Helsinki Institute of Physics/CERN) – erityisesti Miika Tuiskua ja Tapio Niemeä – siitä mahdollisuudesta, että sain jatkaa tutkielmani kirjoitusta mitä virkistävimmässä ilmapiirissä Genevessä kesällä 2005.

Tämän tutkielman tekemiseen on saatu tukea Suomen Akatemian projektista numero 52894.

Tampereella 20. helmikuuta 2006

Turkka Näppilä

Sisältö

Luku 1

Johdanto

Moniulotteisessa analyysissa (multidimensional analysis) käsillä olevia tietoja on mahdollista tarkastella samanaikaisesti useasta eri dimensiosta. Analysoitava perusdata on numeerista ja se on tuotettu aggregoimalla yksittäistietoja aritmeettisten operaatioiden avulla. Yksittäinen dimensio (esimerkiksi *aika*) voidaan yleensä jakaa eri tarkkuustasoihin (esimerkiksi aika-dimensiota voidaan tarkastella *päivän*, *kuukauden* ja *vuoden* tarkkuudella). Tarkkuustasot muodostavat lisäksi usein keskenään hierarkkisen rakenteen (esimerkiksi edellä mainitun aika-dimension sisällä voidaan muodostaa hierarkia: *päivä–kuukausi–vuosi*). Kun jonkin dimension tarkkuustaso muuttuu, on muutoksen suunnasta riippuen analysoitavaa perusdataa joko koostettava edelleen tai vaihtoehtoisesti vähennettävä sen koosteisuutta. Moniulotteisen analyysin keskeinen piirre onkin se, että analysoitavaa dataa voidaan tarkastella useasta eri ulottuvuudesta ja eri yksityiskohtaisuuden tarkkuuksilla.

Moniulotteisessa analyysissa pyritään analysoitavasta datasta havaitsemaan (toimintayksikön kannalta) olennaiset muutokset, löytämään toimintoihin liittyvät kehityskulut sekä vertailemaan eri yksilöitä tai yksikköjä keskenään. Täten moniulotteisen analyysin luonnollisia sovellusalueita ovat esimerkiksi organisaatioiden vuosikertomuksissa ja osavuosikatsauksissa tarvittavien tunnuslukujen tuottaminen, sijoitussalkkujen analysointi, kustannus–hyötyanalyysi, markkinointitutkimus ja laadunvalvonta, mutta moniulotteisella analyysilla on ollut liiketoimintaan liittyvien sovellusten lisäksi myös muun muassa lääketieteellisiä ja tiedonhakuun liittyviä sovelluksia [NHJ03].

Moniulotteiseen analyysiin läheisesti liittyviä tutkimusaloja ovat *tiedonlouhinta* (data mining) ja *tietämyksen muodostaminen* (knowledge discovery). Niiden ja moniulotteisen analyysin välillä vallitsee kuitenkin eräs keskeinen eroavuus: tiedonlouhinnassa ja tietämyksen muodostamisessa käsiteltävästä datasta yritetään algoritmisesti löytää aikaisemmin tuntemattomia ominaisuuksia, kun taas moniulotteisessa analyysissa halutaan datasta saada esille jokin tietty ilmiö, vaikka ei välttämättä tiedetäkään, millainen haluttu ilmiö on yksityiskohtiensa tasolla.

Vaikka moniulotteisen analyysin teoreettinen perusta (matemaattinen matriisilaskenta) on suhteellisen vanha, automatisoidusti suoritettavan moniulotteisen analyysin ilmeiset hyödyt etenkin liiketoiminnalle havaittiin vasta 1990-luvun alussa. Vuonna 1993

Codd, Codd ja Salley [CCS93] esittelivät käsitteen OLAP (*On-Line Analytical Processing*) kuvaamaan teknologiaa, jolla moniulotteista analyysia voidaan suorittaa tietokoneiden avulla. Siitä lähtien OLAP on ollut kasvavan mielenkiinnon kohteena, ja nykyisin kaikilla suurilla tietokantatoimittajilla on omat OLAP-järjestelmänsä. Ennen OLAP-sovelluksia tietojen moniulotteinen analyysi tapahtui taulukkolaskentaohjelmissa käytettäviä erillisiä laskentataulukoita (spreadsheet) käyttäen.

Moniulotteisen analyysin toiminnalliset ja suorituskykyyn liittyvät vaatimukset eroavat perinteisille OLTP (*On-Line Transaction Processing*) -relaatiotietokantatoteutuksille asetetuista vaatimuksista. Relaatiotietokantajärjestelmät on suunniteltu ennen kaikkea varmistamaan tavanomaisten tietokantatapahtumien (lisäykset, päivitykset ja poistot) mahdollisimman tehokas prosessointi, mutta niiden kyselyominaisuudet ovat varsin rajoitettuja. Relaatiotietokantojen tietojen koostaminen edellyttää yksittäisten tietokannan taulujen ja niissä olevien tietojen yhdistämistä näkymiksi. Näkymien muodostaminen ja kyselyjen tekeminen muodostettujen näkymien perusteella on kuitenkin käyttäjän kannalta monimutkaista ja kyselyn evaluoiminen johtaa usein huonoon vasteaikaan. OLAP-kehityksen lähtökohtana on ollut tuottaa järjestelmiä, joissa kompleksista laskentaa vaativien kyselyiden prosessointi on suhteellisen nopeaa. OLAP-järjestelmien tulee lisäksi olla käyttäjäystävällisiä, sillä moniulotteisen analyysin suorittajat eivät useinkaan ole tietojenkäsittelyn ammattilaisia.

OLAP-käsitteen rinnalla kehittyivät *tietovarastoinnin* (data warehousing) käsite ja siihen liittyvä teknologia. *Tietovarasto* (data warehouse) tarjoaa keskitetyn pääsyn muutoin heterogeenisissa, autonomisissa ja/tai hajautetuissa tietolähteissä oleviin tietoihin [Abi03]. Tietovaraston sisältö on tyypillisesti johonkin aihepiiriin keskittynyt (subject-oriented), useista tietolähteistä integroitu, ajan suhteen jaksotettu (time-variant) ja suhteellisen vakaa (non-volatile) kokoelma dataa [SS05]. Tietovarastossa rakenteeltaan erilaisista operationaalisista tietojärjestelmistä kerätty data yhdistetään ja harmonisoidaan. Tietovaraston tietojen perusteella muodostetaan erillisillä OLAP-palvelimilla moniulotteisessa analyysissa käytettäviä *tietokuutioita* (data cube).

Tietokuutio on OLAP-järjestelmissä moniulotteisuuden esittämiseen käytetty tietorakenne. Tietokuutio koostuu analysoitavaa numeerista dataa kuvaavista *mitta-attribuuteista* (measure attribute) sekä *dimensioattribuuteista* (dimension attribute), joiden suhteen mitta-attribuuttien arvoja tarkastellaan. Kutakin dimensioattribuuttien arvojen kombinaatiota vastaa mitta-attribuutin yksikäsitteinen arvo. Tietokuutiota kutsutaan toisinaan myös faktatauluksi (fact table). Sellaiset dimensioattribuutteihin liittyvät analysoinnin kannalta olennaiset ominaisuudet (esimerkiksi hierarkiasuhteet), joita ei voida esittää suoraan tietokuutiossa, esitetään omissa *dimensiotauluissaan* (dimension table). Tietokuutiossa moniulotteista analyysia suoritetaan tarkoitusta varten erikseen kehitettyjen OLAP-operaatioiden avulla. OLAP-operaatioilla muun muassa vaihdetaan tietokuution tietojen tarkastelutasoa tai muutetaan tietokuutiosta tuotettavaa näkymää.

Yksittäisen organisaation sisällä on usein suuri määrä itsenäisiä tietokantajärjestelmiä organisaation eri toimintayksikköjä ja toimintoja varten. Tietokantajärjestelmät on lisäksi usein suunniteltu ja toteutettu eri aikakausina ja niiden ylläpito on toisistaan riippumatonta. Tietokannat saattavat lisäksi toimia eri käyttöjärjestelmissä ja laitealustoilla

ja perustua eri tietomalleihin. Organisaation kokonaistilanteen arvioimiseksi tällaisissa erillisissä tietokannoissa oleva informaatio pitää tietovarastossa integroida. Tietokantajärjestelmissä olevista eroavuuksista riippuen tietojen yhdistely voi vaatia tietovaraston ylläpitäjältä huomattaviakin ponnisteluja. Tietoverkkojen kehitys viime vuosikymmenen aikana on mahdollistanut organisaatioiden laajamittaisen verkottumisen. Jos jo yksittäisen organisaation tietokantojen semanttisten, syntaktisten ja järjestelmiin liittyvien eroavuuksien ratkaiseminen voi aiheuttaa ongelmia, voi vain kuvitella, millainen on tilanne silloin, kun toisistaan riippumattomat organisaatiot yhdistävät tietojärjestelmänsä esimerkiksi strategisen kumppanuuden tai yritysfuusioiden seurauksena.

Tietomuotojen heterogeenisuus on ollut koko tähänastisen modernin tietojenkäsittelyn ajan laajamittaisen tietojen vaihdon (information exchange) keskeinen este. XML (*Extensible Markup Language*) on kuitenkin viime vuosina tuonut ratkaisun tähän ongelmaan. XML on varsinkin World Wide Web -ympäristössä yleistyvä tiedon esitysformaatti. XML on puhdas merkintäkieli, mutta se tarjoaa suosittua sukulaistaan, HTML-merkintäkieltä laajemman ja joustavamman välineistön tietojen esittämiseen. XMLmerkintäkielen avulla on lisäksi mahdollista esittää semanttista, datan merkitystä kuvaavaa informaatiota. Nykyisin voidaan olettaa, että kaikista tietojärjestelmistä ovat tiedot saatavissa XML-muodossa.

XML muodostaa SGML (*Standard Generalized Markup Language*) -merkintäkielen osajoukon. Kaikki XML-dokumentit ovat siten myös SGML-dokumentteja. Kaikki SGML-dokumentit eivät kuitenkaan ole XML-dokumentteja, joten osajoukkosuhde ei ole voimassa toiseen suuntaan. SGML-kieli syntyi 1980-luvun alussa tarpeesta kehittää merkintätapa, joka mahdollistaisi sähköisten dokumenttien vaihtamisen laitteisto- ja ohjelmistoriippumattomalla tavalla [Tom89]. SGML:n kehittämisen taustalla vaikuttivat erityisesti tekstinkäsittelyn (text processing) tarpeet.

Havaittiin nimittäin, että tekstinkäsittelyssä voidaan *proseduraalisen* merkintätavan lisäksi soveltaa myös *deskriptiivistä* merkintätapaa [RDSH02]. Proseduraalisessa merkintätavassa tuotettavaan tekstiin merkitään kohtia, joita tekstin tulostusvaiheessa sitten käsitellään tietyllä tavalla; esimerkiksi jokin tekstikatkelma, yksittäinen sana tai merkki tulostetaan korostettuna. Proseduraalisen merkintätavan edellytyksenä on, että tekstinkäsittelyohjelman sisään on ohjelmoitu kunkin proseduurin tulostusvaiheen toteutus. Eri tekstinkäsittelyohjelmat poikkeavat kuitenkin keskenään siinä, miten esimerkiksi jokin muotoilukomento tekstiin merkitään. Deskriptiivisessä merkintätavassa tekstistä tunnistetaan muotoilusääntöjen sijasta rakenteita (loogisia kokonaisuuksia), kuten esimerkiksi kirjan lukuja, alalukuja, otsikkoja, alaotsikkoja, kappaleita ja muita vastaavia. Deskriptiivisellä merkintätavalla voidaan esittää tekstin looginen rakenne ja muiden ohjelmien avulla voidaan sitten huolehtia varsinaisesta ulkoasun tuottamisesta merkitylle tekstille. SGML loi standardin tavan merkitä tekstiä deskriptiivisesti.

SGML-kielen keskeinen piirre on *tunnisteiden* (tag) käyttö. Tunnisteiden avulla saadaan eksplisiittisesti esitettyä dokumentissa olevat loogiset rakenteet ja niiden keskinäiset suhteet. SGML:ssä on kahdenlaisia tunnisteita, *alkutunnisteet* (start tag) ja *lopputunnisteet* (end tag). Keskenään yhteenkuuluvat alku- ja lopputunnisteet rajaavat tekstistä loogiset rakennekokonaisuudet. Tällaista kokonaisuutta kutsutaan *elementiksi* (ele-

ment). Käyttämällä alku- ja lopputunnisteessa samaa *nimeä* (label) rajataan elementti dokumentin muusta informaatiosta. Elementti voi sisältää tekstimuotoista dataa tai toisia elementtejä, jolloin ne muodostavat keskenään hierarkkisen, sisäkkäisen rakenteen. Elementin alkutunnisteessa on lisäksi mahdollista esittää *attribuuttien* (attribute) avulla elementtiin tai elementin sisältöön liittyviä ominaisuuksia.

Kansainvälinen standardoimisliitto ISO julkaisi SGML-standardin [ISO86] vuonna 1986. Alkuinnostuksen jälkeen SGML:n parissa tehty tutkimus ja kehitystyö kuitenkin asteittain vähenivät [RDSH02], mihin oli syynä muun muassa SGML-standardin monimutkaisuus ja syntaksin liiallinen sallivuus (SGML:ssä käyttäjä voi esimerkiksi määritellä itse, millä tavalla tunnisteet esitetään). SGML nousi uudestaan suuremman mielenkiinnon kohteeksi 1990-luvun alussa Internetin synnyttyä. Jotta sähköisiä dokumentteja voitaisiin tietoverkossa vaihtaa mahdollisimman joustavasti ja tehokkaasti, oli luotava tapa esittää niitä laite- ja ohjelmistoriippumattomalla tavalla. SGML:ää pidettiin tähän tarkoitukseen kuitenkin liian monimutkaisena, ja tarkoitukseen kehitettiin HTML (*Hypertext Markup Language*) -merkintäkieli [ISO/IEC00], joka on myös SGML:n osajoukko. HTML:ssä on deskriptiivisen merkintätavan rinnalla myös proseduraalisia piirteitä, sillä siinä voidaan tekstistä erottaa rakennekokonaisuuksien (otsikko, kappale jne.) lisäksi myös muotoilutapoja (kirjasimen tyyppi, tyyli, koko jne.) HTML-kielessä sallittavien tunnisteiden joukko on ennalta kiinnitetty, kun taas SGML:ssä käyttäjä voi määritellä käytettävät tunnisteet omien tarpeidensa mukaan. Kiinteä tunnisteiden joukko asettaa rajoituksia sille, mitä HTML-dokumentissa on mahdollista esittää. SGML:ssä ei vastaavaa rajoitusta esiinny.

Kun Internetin käyttö ja sovellukset laajenivat, syntyi jälleen tarve puhtaasti deskriptiiviselle dokumenttien merkintätavalle. Tähän tarkoitukseen alettiin vuonna 1996 perustetussa *World Wide Web* (W3) -konsortiossa kehittää XML- merkintäkieltä. XML:n merkittävin ero puhtaaseen SGML:ään verrattuna on se, että XML-dokumenteissa esitettävien rakenteiden syntaksi on määritelty tiukemmin kuin SGML-kielen vastaavien rakenteiden syntaksi. SGML-dokumentissa käyttäjä saa – XML-dokumentista poiketen – muun muassa määritellä vapaasti elementeissä käytettävien tunnisteiden esitystavan eikä elementin tarvitse välttämättä päättyä lopputunnisteeseen, jos elementin tarkoitettu vaikutusalue käy kontekstista muuten ilmi. XML-dokumentit on lisäksi tarkoitettu tietokoneiden käsiteltäväksi, kun SGML-dokumentit on tarkoitettu ihmisten luettavaksi. XML-merkintäkielen syntaksi on määritelty W3C-konsortion suosituksessa [BPS+04]. Vaikka edellä mainittu spesifikaatiodokumentti on nimeltään vain "suositus" (recommendation), voidaan sitä kuitenkin pitää XML-merkintäkielen tosiasiallisena standardina.

Vaikka XML mahdollistaakin laajamittaisen tietojen vaihdon, on sillä myös ongelmansa. XML-tiedon ongelma – ja samalla sen ilmeinen etu – on, että se on *puolirakenteista* (semi-structured). Puolirakenteiseksi sanotaan tietoa, jonka rakenne on epäsäännöllinen, implisiittinen ja osittainen [Abi97]. XML-tiedon yhteydessä tämä tarkoittaa sitä, että samaa tarkoittavaa informaatiota on eri XML-dokumenteissa ja myös yksittäisen dokumentin sisällä mahdollista esittää useilla toisistaan poikkeavilla tavoilla, XML-dokumentissa olevat rakenteelliset suhteet on tuotava esiin dokumentin jäsennyksen kautta ja yksittäiseen XML-dokumenttiin voi sisältyä toisaalta sekä vahvasti strukturoituja

että toisaalta täysin rakenteettomia jaksoja. XML-tieto on lisäksi *itsekuvautuvaa* (self-describing), sillä jokaisessa XML-dokumentissa on ilmentymätason (data) lisäksi aina läsnä myös kaaviotaso (elementtien ja attribuuttien nimet).

Koska XML-dokumentin elementit muodostavat keskenään hierarkkisen, sisäkkäisen rakenteen, on XML-dokumenttia luontevaa tarkastella puuna. Tämä on lähtökohtana myös W3-konsortiossa kehitetyissä XML-kyselykielissä, joista yleisimmin käytetty on XQuery [BCF+03]. XQuery on proseduraalinen kyselykieli, jossa on kuvattava vaihe vaiheelta, miten halutut tiedot lähtödokumentista poimitaan ja miten tulosdokumentti muodostetaan. XML-dokumentin rakenteiden osoittamiseen käytetään XQueryssä *polkuilmauksia* (path expression).

Polkuilmaus ilmaisee reitin, joka dokumenttipuun juuresta on kuljettava, jotta haluttu solmu saavutetaan. XML-tietojen integroinnin kannalta polkuilmausten käyttö on kuitenkin XQueryn, kuten myös muiden polkuorientoituneiden (path oriented) kyselykielten suurin heikkous, sillä polkuilmaukset eivät ota riittävästi huomioon XML-dokumenttien puolirakenteisuutta. Jotta yksittäisestä XML-dokumentista voidaan poimia kaikki siihen sisältyvä informaatio, on dokumentin jokaista rakenteeltaan erilaista osaa varten muodostettava oma polkuilmauksensa. Kun tietoja sitten integroidaan useasta XML-dokumentista, pitää niiden jokaista rakenteeltaan erilaista osaa varten muodostaa oma polkuilmauksensa. Polkuilmauksien käytössä on myös oletuksena, että käyttäjä tuntee lähtödokumentin rakenteen. Kun rakenteeltaan heterogeenisten, autonomisista tietolähteistä peräisin olevien XML-dokumenttien tietoja integroidaan, ei ole realistista edellyttää, että integroinnin suorittaja tuntee jokaisen käsiteltävän dokumentin rakenteen yksityiskohtaisesti.

Kun autonomisissa relaatiotietokannoissa olevia tietoja integroidaan, kohdataan usein tilanteita, joissa semanttisesti samanlainen informaatio on eri tietokannoissa esitetty rakenteellisesti eri tavoin. Jos esimerkiksi tietokannassa A on tieto laitoksen nimestä esitetty kaaviotasolla (relaation kaaviossa on attribuutti nimeltä `CS`), voi tietokannassa B vastaava tieto olla esitetty ilmentymätasolla (relaation `dept`-nimisen attribuutin arvo on `CS`). Vaikka tällaisten tilanteiden hallinta olisikin relaatiotietokantojen yhteentoimivuuden kannalta suotavaa, eivät konventionaaliset kyselykielet, esimerkiksi SQL, pysty käsittelemään kaaviotasolla esitettyä informaatiota [LSS01]. Vastaavasti XML-dokumenteissa on relaatiotietokantojen tavoin mahdollista esittää samaa tarkoittavaa informaatiota sekä intensionaalisella että ekstensionaalisella tasolla. Toistaiseksi esimerkiksi XQuery tarjoaa kuitenkin varsin rajalliset keinot XML-dokumenttien kaaviotasolla olevan informaation hyödyntämiseksi kyselyissä, mikä edelleen vaikeuttaa sen käyttöä XML-tietojen integroimisessa.

Edellä on osoitettu XQuery-kyselykielen ilmeiset heikkoudet XML-tietojen integroinnin näkökulmasta. XML-tietojen tehokasta integrointia varten on selvästi olemassa tarve kyselykielelle, joka mahdollistaa XML-dokumentteihin tehtävät kyselyt siten, että (1) yhden kyselyn avulla voidaan poimia tietoja useammasta rakenteeltaan erilaisesta dokumentista, (2) käyttäjän ei tarvitse tuntea lähdedokumenttien rakennetta ja (3) kyselyssä voidaan hyödyntää yhtä hyvin XML-dokumenttien kaavio- kuin ilmentymätasollakin olevaa informaatiota. Tutkielman yhteydessä on tähän tarkoitukseen suunniteltu uuden-

lainen XML-tiedon käsittelyyn tarkoitettu `is_component_of`-kyselyprimitiivi.

Kyselyprimitiivin `is_component_of` avulla on mahdollista poimia XMLdokumenteista semanttisesti yhteenkuuluvaa informaatiota ilman, että kyselyn tekijän tarvitsee tietää liian paljon lähdedokumenttien rakenteesta. Lähtökohtana kuitenkin on, että kyselyn tekijän on tunnettava lähdedokumenttien semantiikka, so. niissä esiintyvien elementtien ja attribuuttien nimet ja dokumenttien tietosisältö. Semanttisesti yhteenkuuluvan informaation katsotaan muodostuvan XML-puuhierarkiassa mahdollisimman lähellä toisiaan sijaitsevista elementeistä ja attribuuteista sekä niihin liittyvästä tekstisisällöstä.

XML-dokumenttien rakenteellisen tuntemattomuuden käsittelemisessä on `is_component_of`-kyselyprimitiivin yhteydessä sovellettu tiedonhakumaista lähestymistapaa. XMLtiedonhaku perustuu kyselyn tekijän antamia *hakusanoja* (keyword) vastaavien mielekkäiden XML-rakenteiden tuottamiseen. Mielekkäiden rakenteiden valitsemisessa on yleisesti käytetty LCA (*Lowest Common Ancestor*) -tyyppistä semantiikkaa, jossa annettuja hakusanoja vastaavia XML-dokumentin komponentteja etsitään niiden pienimmästä yhteisestä esivanhempisolmusta alkavasta alipuusta. LCA-sematiikkaan nojautuvassa tiedonhaussa ei kuitenkaan pystytä kaikissa tilanteissa tuottamaan relevantteja hakutuloksia (esimerkiksi jos XML-dokumentissa ylemmällä hierarkiatasolla olevaa informaatiota pitää toistuvasti liittää alemmalla hierarkiatasolla esitettyyn informaatioon). Tämän vuoksi tutkielmassa on hahmoteltu uudenlainen *suppeimman mahdollisen kontekstin semantiikka*, joka tuottaa LCA-semantiikkaan pohjautuvia hakuja tarkempia tuloksia. Suppeimman mahdollisen kontekstin semantiikassa on ajatuksena, että hakusanoja vastaavan mielekkään XML-rakenteen muodostavat sellaiset XML-dokumentin solmut, joiden välisistä (tietyn ehdon mukaan valituista) kaarista voidaan muodostaa *täydellinen graafi* (complete graph).

Kehitetyn `is_component_of`-kyselyprimitiivin ja suppeimman mahdollisen kontekstin semantiikan hyötyjen osoittamiseksi on kehitetty järjestelmäprototyyppi, jonka avulla on mahdollista muodostaa `is_component_of`-primitiiviä soveltaen XMLdokumenteista yksinkertaisia tietokuutioita. Järjestelmän toimintaa havainnollistetaan esimerkkikyselyjen avulla.

Tutkielman loppuosa on organisoitu seuraavasti: Luvuissa 2 ja 3 esitellään moniulotteisen analyysin ja XML-merkintäkielen perusteita. Luvussa 4 tarkastellaan XML-tietojen integroinnin ongelmia ja esitetään kehitetty kyselyprimitiivi `is_component_of` ja suppeimman mahdollisen kontekstin semantiikka yksityiskohtaisesti. Luvussa 5 esitellään kehitetyn järjestelmäprototyypin toimintaperiaatteet ja havainnollistetaan sen – ja samalla `is_component_of`-kyselyprimitiivin ja suppeimman mahdollisen kontekstin semantiikkaa – toimintaa esimerkkikyselyjen perusteella. Luku 6 sisältää kehitystyön tulosten tarkastelun ja hahmotelmia jatkokehityksestä. Luvussa 7 on esitetty tutkielmasta yhteenveto.

Luku 2

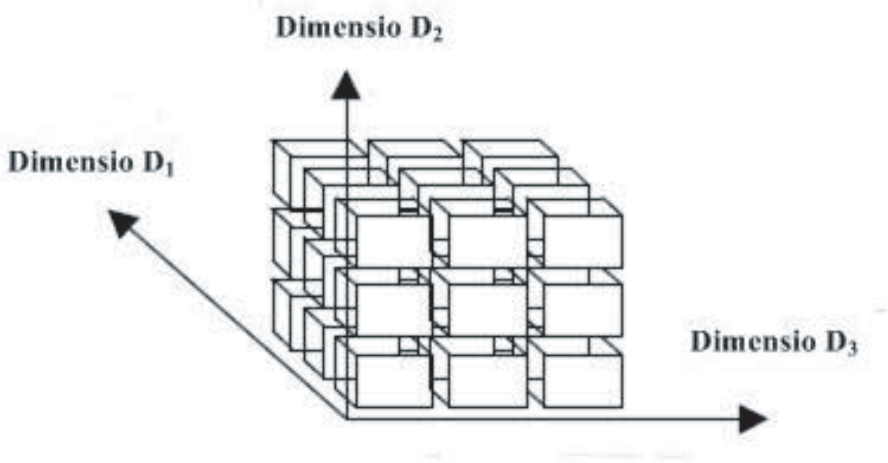
Moniulotteinen analyysi ja tietovarastot

2.1 Tietomalli

Moniulotteinen analysointi edellyttää, että on olemassa looginen tietomalli, jonka avulla moniulotteinen tieto voidaan esittää ja joka tarjoaa keinot sen käsittelyyn. Tietojen moniulotteinen analysointi perustuu siihen, että käytössä olevaa yhteenvetotietoa voidaan organisoida erilaisten näkökulmien/tekijöiden mukaan. Datan analysoinnissa käytettävät näkökulmat voivat olla toisistaan riippuvia tai riippumattomia. Näitä näkökulmia kutsutaan moniulotteisessa analysoinnissa *ulottuvuuksiksi* eli *dimensioiksi*. Dimensioita voisivat olla esimerkiksi "aika" ja "myyntialue". Dimensiot esitetään tietokuutiossa *dimensioattribuuttien* avulla (dimensiota "aika" edustaa esimerkiksi dimensioattribuutti "vuosi", joka puolestaan saa arvoja "2000", "2001", "2002" jne.). Dimensioattribuuttien arvot ovat tyypillisesti atomisia. Dimensioiden avulla analysoitava yhteenvetodata koostuu erilaisista *mitta-arvoista* (measure), jotka ovat yleensä numeerista dataa. Edelliseen esimerkkiin perustuen mitta-arvo voisi olla luku, joka kuvaa kauppakohtaista kokonaismyyntiä vuositasolla. Mitta-arvojen merkitys esitetään *mitta-attribuuttien* (esimerkiksi "kokonaismyynti") avulla. Ei ole olemassa formaalia tapaa ratkaista, mistä attribuuteista tehdään dimensio- ja mistä mitta-attribuutteja, vaan attribuuttien tyyppien valinta on aina riippuvainen käyttökontekstista [AGS97].

2.1.1 Tietokuutio

Tietokuutio (data cube) on yleistermi, jolla kuvataan moniulotteisuuden visualisointeja. Tietokuutio on myös käytetyin looginen malli moniulotteisen tiedon esittämiseksi. Kuvassa 2.1 on esimerkki tietokuutiosta. Tietokuutio on dimensioiden rajoittama avaruus. Yksittäiset dimensioattribuutit tai dimensioattribuuttien joukot rajaavat tietokuutiosta pienempiä osa-avaruuksia: tietokuutio toisin sanoen rakentuu pienemmistä tietokuutioista. Toisiinsa liittyvien tietokuutioiden kokoelma muodostaa moniulotteisen tietokannan (multidimensional database, MDD) [PJ01].



Kuva 2.1: Tietokuution visualisointi.

On syytä huomata, että tietokuutio on ainoastaan metafora moniulotteiselle avaruudelle. Moniulotteisessa analyysissa tarvitaan tavallisesti eri määrä dimensioita, kuin mitä tietokuution graafiseen esitykseen sisältyy [PJ01]. Moniulotteisessa avaruudessa ulottuvuuksia on kaksi tai useampia. Kaksiulotteisen avaruuden luonnollinen visualisointi on riveihin ja sarakkeisiin jakaantuva taulukko. Kun ulottuvuuksia on enemmän kuin kolme, puhutaan usein hyperkuutiosta (hypercube) [Mar99]. Kuitenkin myös useampiulotteinen avaruus visualisoidaan tavallisesti kaksiulotteisten taulukkojen kokoelmana. Tietokuutiota ei voida myöskään implementoida sellaisenaan tietokannan kaavioksi [Sho97], vaan tietokantaratkaisuissa moniulotteinen tietorakenne pitää toteuttaa muilla keinoin. Yleisin tapa esittää moniulotteinen tietorakenne tietokantatasolla on ilmaista se erilaisten taulukkorakenteiden avulla.

Dimensioattribuuttien rajoittamissa tietokuution soluissa on alkioina mitta-arvoja. Tietokuution solu voi olla yksi- tai monipaikkainen, so. yhteen tietokuution soluun voidaan sijoittaa yksi tai useampi mitta-arvo; myös tyhjä arvo on mahdollinen. Jokainen dimensioattribuutti edustaa jotakin tietokuution erillistä dimensiota, ja dimensioattribuuttien arvojen kombinaatiot määrittävät aina yksikäsitteisesti yhden mitta-attribuutin arvon tietokuutiossa. Toisin sanoen mitta-attribuuttien arvot ovat riippuvaisia dimensioattribuuttien arvoista. Tietokuutiossa, jossa on n ulottuvuutta, jokaisella tietokuution solulla on $2n$ naapurisolua [Tho97].

Formaalisti esitettynä n-dimensionaalinen tietokuutio on nelikkö $C = (N,\mathbf{D},\mathbf{M},F)$, missä N on tietokuution nimi, $\mathbf{D} = \{D_1,\ldots,D_n\}$ dimensioattribuuttien joukko ($D_i = \{d_1,\ldots,d_m\}$, $D_i \cap D_j = \emptyset, i \neq j$), $\mathbf{M} = \{M_1,\ldots,M_k\}$ mitta-attribuuttien joukko ($M_r = \{m_1,\ldots,m_p\}$) ja F joukko funktioita $f : \mathrm{D} \rightarrow \mathbf{M}$, jotka liittävät jokaiseen dimensioattribuuttien arvojen kombinaatioon $c \in \{(d_i,\ldots,d_j)|d_i \in D_1,\ldots,d_j \in D_n\}$

täsmälleen yhden mitta-attribuutin $M \in \mathbf{M}$ arvon m.

Samaan dimensioon kuuluvat dimensioattribuutit ovat usein keskenään hierarkkisessa suhteessa. Dimensioattribuutin hierarkiataso ilmaisee sen yksityiskohtaisuuden asteen, jolla analysoitavaa dataa kulloinkin tarkastellaaan. Tyypillinen esimerkki hierarkkisessa suhteessa olevista dimensioattribuuteista on ajan tarkastelu päivän, viikon, kuukauden, vuoden jne. tarkkuudella. Dimensiohierarkia muodostaa puurakenteen. Puuna esitettävä dimensiohierarkia (jossa korkein hierarkiataso on puun juuri) voi muodostaa tiukan (strict) hierarkian, jolloin jokaisella dimensioattribuutilla voi olla ainoastaan yksi vanhempi dimensiohierarkian ylemmällä tasolla, tai suunnatun syklittömän graafin, jolloin solmun vanhempien lukumäärälle ei ole asetettu rajoituksia.

Dimensioattribuutteihin voi liittyä myös sellaisia tietoja, jotka pitää ilmaista eksplisiittisesti mutta joita ei kuitenkaan voida esittää suoraan tietokuution attribuutteina [Hir01]. Kutakin tällaista dimensioita varten pitää luoda erillinen tietorakenteensa, jotka kuvaavat dimensioattribuuttiin liittyviä ominaisuuksia. Tällöin puhutaan usein erityisistä *dimensiotauluista* (dimension table) erotuksena *faktataulusta* (fact table), joka sisältää dimensio- ja mitta-attribuuttien arvoja.

2.1.2 Tietokuution käsittely

Relaatioalgebra määrittää operaatiot, joilla relaatiotietokantoihin tallennettuja tietoja kyetään käsittelemään. Relaatioalgebran operaatioita on kahdenlaisia: klassiset joukko-opilliset operaatiot (unioni, leikkaus, erotus ja tulo) ja erityisesti relaatiotietokantojen käyttötarpeisiin kehitetyt operaatiot [EN00]. (Itse asiassa myös nämä relaatiotietokantaoperaatiot ovat luonteeltaan joukko-opillisia.) Relaatiotietokantaoperaatioista yleisimmät ovat valinta, projektio ja liitos. Valinta-operaatiossa annetusta relaatiotaulusta (joka on relaation visualisointi) poimitaan ne rivit, jotka täyttävät valintaoperaatiossa annetun valintaehdon. Projektiossa puolestaan relaatiotaulusta erotetaan ne sarakkeet, jotka ovat tarkasteltavan asian kannalta olennaisia. Liitoksessa yhdistetään yhdeksi tauluksi relaatiotaulut, jotka toteuttavat annetun liitosehdon.

Toistaiseksi ei OLAP-ratkaisujen tarpeita varten ole vielä olemassa yhtä yleisesti hyväksyttyä algebraa. Esimerkiksi Agrawal ja muut [AGS97], Gyssens ja Lakshmanan [GL97], Vassialidis [Vas98] ja Pedersen ja Jensen [PJ99] ovat esittäneet omat ehdotuksensa OLAP-algebraksi. Codd ja muut hahmottelivat uraauurtavassa työssään [CCS93] joitakin OLAP-operaatioille luonteenomaisia piirteitä, joiden vaikutus on havaittavissa myös edellä mainituissa algebraehdotuksissa. Ilmeistä on, että OLAP-toiminnallisuuden mahdollistavan algebran tulee olla ilmaisuvoimaltaan relaatioalgebran veroinen ja sisältää lisäksi operaatioita, jotka mahdollistavat joustavan navigoinnin moniulotteisessa avaruudessa ja tietokuution uudelleenorganisoimisen.

Osa OLAP-toiminnallisuuden perusoperaatioista on analogisia relaatiotietokantaoperaatioiden kanssa, mutta moniulotteiseen analysointiin sisältyy myös sellaisia operaatiota, joilla ei ole vastineita relaatioalgebrassa. Koska OLAP-terminologia ei ole vakiintunutta [VS99], käytetään kirjallisuudessa OLAP-operaatioista vaihtelevia nimityksiä. On edelleen syytä huomata, että OLAP-perusoperaatioiden nimet ovat luonteeltaan lähinnä kuvailevia, sillä samalle termille on monesti useita tulkintatapoja.

OLAP-operaation suorittaminen merkitsee tietokuutioon tehtävää kyselyä ja usein OLAP-operaatio tuottaa uuden kuution, joka eroaa rakenteellisesti lähtökuutiosta. Tavallisimpia OLAP-operaatioita käytetään tietokuutiossa navigointiin eli tietokuution dimensioattribuuttien yksityiskohtaisuuden määrittelyyn (roll-up ja drill-down), tulosnäkymän muodostamiseen (slice, dice ja pivot) ja attribuuttien merkityksen vaihtamiseen (push ja pull). Useissa OLAP-ratkaisuissa on lisäksi mahdollista suorittaa relaatiomallin perusoperaatiot (ks. esim. [AGS97, GL97, Vas98, PJ99]). Seuraavassa esitellään tyypillisimmät OLAP-operaatiot.

Tietokuutiossa navigointi. Tietokuutiossa navigointi tarkoittaa liikkumista ylös- tai alaspäin dimensioattribuuttien hierarkiassa. Dimensiohierarkiassa korkeammalle hierarkiatasolle siirtymisen mahdollistaa OLAP-operaatio *roll-up*; hierarkiassa alemmalle tasolle siirtyminen on puolestaan *drill-down*-operaation soveltamista. Vaihtoehtoisia nimityksiä roll-up-operaatiolle ovat *aggregation* ja *consolidation* ja drill-down-operaatiolle *roll-down* ja *drill-through* [Vas98]. Käytettävästä loogisen tietomallin toteutuksesta riippuu, voidaanko roll-up- ja drill-down-operaatioita soveltaa ainoastaan dimensioattribuuteille, joille on ennalta määritelty yksi ainoa karkeistushierarkia, vai voidaanko niitä soveltaa myös dimensioattribuuteille, joille ei ole määritelty karkeistushierarkiaa tai joilla on vaihtoehtoisia karkeistushierarkioita. Tietokuutiossa navigoitaessa siitä luodaan näkymä, joka eroaa lähtökuutiosta vähintään jonkin dimension suhteen. Tietokuutiosta luotava näkymä on itsessään (uusi) tietokuutio.

Roll-up-operaation suorittaminen vähentää tietojen yksityiskohtaisuuden tasoa. Dimensiohierarkiassa ylemmille tasoille siirtyminen on suhteellisen suoraviivainen prosessi: jos esimerkiksi halutaan siirtyä tarkastelemaan kauppakohtaisten myyntilukujen sijasta paikkakuntakohtaisia myyntilukuja, riittää, kun tietyn paikkakunnan kauppojen myyntiluvut lasketaan yhteen jatkotoimenpiteitä varten. Intuitiivisesti katsottuna roll-up-operaation suorittaminen merkitsee sitä, että kaikki alemman hierarkiatason dimensioattribuuttien arvot korvataan ylemmän hierarkiatason arvoilla [Vas98].

Kuten on jo aiemmin todettu, tietokuution soluissa olevat mitta-attribuuttien arvot ovat yleensä koostettua tietoa eli ne on saatu laskemalla yhteen yksittäisten tietoalkioiden arvoja. Moniulotteisessa analyysissa mitta-attribuuttien arvoja tyypillisesti yhdistellään dimensioattribuuttien hierarkiatasojen perusteella edelleen, jolloin saadaan koostettuja eli aggregoituja arvoja. Mitta-attribuuttien arvojen yhdistely tarkoittaa käytännössä sitä, että niihin sovelletaan jotakin aggregointifunktiota. Tavallisimpia aggregointifunktioita käytetään tietokuution alkioiden yhteissumman, keskiarvon ja lukumäärän laskemiseen sekä suurimman ja pienimmän arvon määrittämisen. Mitta-attribuuttien arvoista voidaan myös tuottaa johdettuja arvoja. Tyypillinen johtamisoperaatio on mitta-arvojen keskinäisen suhteen laskeminen. Roll-up- ja drill-down-operaatioiden suorittamiseen sisältyy implisiittisesti aina jonkin aggregointifunktion soveltaminen [PJ99].

Ongelmallisempaa sen sijaan on, kun näkymän yksityiskohtaisuuden tasoa kasvatetaan eli kun siirrytään nykyistä hierarkiatasoa alemmille tasoille. Ylemmän hierarkiatason mitta-attribuuttien arvot on saatu soveltamalla jotakin aggregointifunktiota alemman hierarkiatason mitta-attribuutien arvoille, mutta ylemmällä hierarkiatasolla olevia

aggregoituja arvoja ei enää voida purkaa osatekijöikseen ilman, että tiedetään entuudestaan, millaisista arvoista alempien hierarkiatasojen attribuutit koostuvat. Esimerkiksi paikkakuntakohtaisista myyntiluvuista ei pystytä triviaalisti päättelemään kauppakohtaisia myyntilukuja. Ainoana ratkaisuna ongelmaan on alimman hierarkiatason ylläpito ja hierarkiakuvauksen olemassaolo; tämä on myös laskennan lähtökohta drill-down-operaatioiden toteutuksissa.

Drill-down-operaatio voidaan toteuttaa soveltamalla tietokuutioon liitosoperaatiota sellaisen tietokuution kanssa, jonka dimensioattribuutit ovat halutulla hierarkiatasolla. Tietokuutioiden liitos on kuitenkin yleensä raskas operaatio. Vaihtoehtoinen tapa toteuttaa drill-down-operaatio on laskea aggregoidut arvot uudelleen dimensiohierarkian alimmalta tasolta halutulle hierarkiatasolle; tällöin tarkennusoperaation suoritusaikaan vaikuttaa dimensiohierarkian alimman tason ja tavoitellun tarkkuustason välinen etäisyys. Drill-down-operaation suoritusta voidaan nopeuttaa hyödyntämällä materialisoituja näkymiä (materialized view), joihin mahdollisia aggregointeja on laskettu ja talletettu etukäteen, ja indeksointirakenteita (ks. esim. [CD97, CDG01]). Niiden päivitys voi kuitenkin olla ongelmallista. Vassiliadis [Vas98] tehostaa drill-down-operaatiota sisällyttämällä tietokuution formaaliin määritelmään toisen tietokuution, peruskuution (basic cube), jossa on tarvittavalla tarkkuustasolla olevaa tietoa aina nopeasti saatavilla. Peruskuution avulla vältytään suorilta tietokuutioiden liitoksilta.

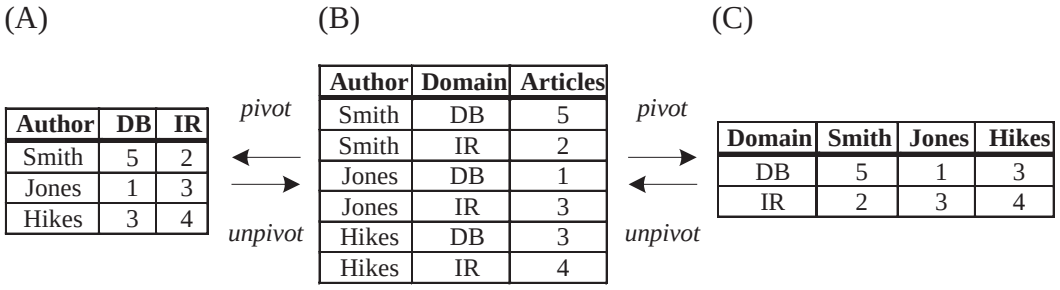
Näkymän muuttaminen. *Slice*- ja *dice*-operaatioita käytetään tietokuution ulottuvuuksien ja sisällön rajaamiseen. Slice- ja dice-operaatioiden suorittaminen muuttaa tietokuutiota rakenteellisesti. Slice-operaatiosta käytetään myös nimitystä *destroy dimension* ja dice-operaatiosta *restriction* [AGS97], *selection*, *screening* ja *filtering* [Vas98]. Sliceoperaatio on analoginen relaatioalgebran projektio-operaation kanssa, ja se merkitsee näkemyksen yksinkertaistamista. Slice-operaatiossa rajataan tietokuutiosta osakuutio, joka täyttää operaation yhteydessä annetun loogisen tai matemaattisen ehdon. Käytännössä tämä tarkoittaa, että tietokuutiosta poistetaan annetun valintaehdon täyttävä dimensio (kuution "siivu"), jolloin tietokuution ulottuvuuksia siis vähennetään ja tietokuutiosta saadaan muodostettua uudenlainen näkemys. Jos tietokuutioon sisältyy aggregoituja mitta-attribuuttien arvoja, ne pitää slice-operaation suorittamisen jälkeen laskea uudelleen jokaista kuutioon jäänyttä dimensioattribuuttien arvojen kombinaatiota kohti.

Dice-operaatiolla tietokuutiosta poistetaan kokonaisen dimension sijasta ainoastaan sellaiset dimensioattribuuttien arvot, joiden katsotaan olevan kyselyn kannalta epäolennaisia. Toisin sanoen dice-operaatiolla tuotetaan tietokuutiosta näkemys, jossa on vain käyttäjää kiinnostavaa tietoa. Dice-operaatio on analoginen relaatioalgebran valinta-operaation kanssa. Dice-operaatiota voidaan soveltaa sekä dimensio- että mitta-attribuutteihin, jos dimensio- ja mitta-attribuutteja käsitellään symmetrisesti (so. dimensioattribuutista voidaan tehdä mitta-attribuutti ja vastaavasti mitta-attribuutista dimensioattribuutti).

Pivot-operaatiota käytetään tietokuutiossa esitettyjen tietojen uudelleenorganisointiin. Kirjallisuudessa esiintyvä vaihtoehtoinen nimitys pivot-operaatiolle on *rotate* [Mar99]. Cunningham, Galindo-Legaria ja Graefe [CGG04] ehdottavat pivot-operaatiota

(ja sen *unpivot*-nimistä käänteisoperaatiota) sisällytettäväksi relationaalisiin tietokannanhallintajärjestelmiin ja osaksi SQL (*Structured Query Language*) -kyselykielen ilmaisuvoimaa. Relaatiotietokannassa pivot-operaatiossa muodostetaan tietokantataulun yhdessä sarakkeessa olevien erillisten arvojen perusteella uusia sarakkeita ja näihin uusiin sarakkeisiin lisätään tietoja saman taulun jossakin toisessa sarakkeessa olevien arvojen perusteella. Pivot-operaatiolla saadaan vähennettyä tietokantataulun rivien määrää, mutta operaation seurauksena taulun sarakkeiden määrä usein vastaavasti lisääntyy.

Tietokuutiossa pivot-operaatio puolestaan merkitsee sitä, että tietokuution jonkin dimensioattribuutin arvoista tehdään mitta-attribuutteja. Tietokuution entisten mitta-attribuuttien arvot on operaation suorittamisen seurauksena järjestettävä uudelleen uusien mitta-attribuuttien mukaisesti. Pivot-operaatiota on havainnollistettu kuvassa 2.2. Siinä taulukossa (B) esitettävästä yksinkertaisen tietokuution taulukkoesityksestä saadaan taulukon (A) tietokuutio, jossa on korostettu tekijää (author), ja taulukon (C) tietokuutio, jossa korostetaan tutkimusalaa (domain). Vastaavasti suorittamalla esimerkin tietokuutioille (A) ja (C) unpivot-operaatio saadaan alkuperäinen tietokuutio (B).



Kuva 2.2: Pivot- ja unpivot-operaatiot.

Pivot-operaation kautta tietokuutiosta tuotettavassa uudessa näkemyksessä korostetaan joitakin dimensiota (analysoimalla yhteenvetotietoja niihin liittyvien dimensioattribuuttien arvojen perusteella) ja samalla peitetään toisia. Pivot-operaatio voi joissakin tilanteissa lisätä saatavan esityksen selkeyttä mutta ei muuten vaikuta tietokuution attribuuttien arvoihin tai niiden rakenteeseen [Vas98].

Attribuuttien merkityksen vaihtaminen. *Push*- ja *pull*-operaatioiden suorittaminen edellyttää, että käytettävässä loogisessa tietomallissa dimensio- ja mitta-attribuutteja on mahdollista käsitellä symmetrisesti. Attribuuttien symmetrinen käsittely tarkoittaa sitä, että dimensioattribuutista on mahdollista tehdä tietokuution mitta-attribuutti ja päinvastoin. Push-operaatiolla dimensioattribuutteja liitetään mitta-arvoihin, jolloin dimensioattribuuteista tulee tietokuution solujen sisältöä. Pull-operaatio on push-operaatiolle vastakkainen operaatio: sen soveltamisella tuotetaan tietokuutio, jossa on uutena dimensioattribuuttina jokin lähtökuution mitta-arvo.

DATA CUBE -operaatio. Edellä esitettyjen operaatioiden lisäksi tunnettu tietokuution käsittelyoperaatio on Grayn ja muiden [GCB+97] esittelemä DATA CUBE -operaa-

tio. Siinä on taustalla ajatus, että moniulotteisen tiedon käsittelytarpeista voidaan suoriutua yhden ilmaisuvoimaisen, relationaalisen operaation avulla. Gray ja muut näkevät tietokuution relaationa, jossa jotkin sen attribuuteista toimivat dimensioattribuutteina ja jotkin mitta-attribuutteina. Tietokuutiorelaation pääavaimen muodostavat sen kaikki attribuutit yhdessä. DATA CUBE -operaation avulla relaatiotaulusta tuotetaan tietokuutio koostamalla taulun tiettyjen attribuuttien arvoja tiettyjen tekijöiden suhteen.

DATA CUBE -operaatio pohjautuu SQL-kyselykieleen, joka on tarkoitettu relaatiotietokantaan tallennettujen tietojen päivittämiseen ja käsittelemiseen. DATA CUBE -operaatio onkin nykyisin toteutettu useimmissa SQL-pohjaisissa tietokannanhallintajärjestelmissä ja se on lisäksi sisällytetty SQL-kielen uusimpaan versioon [ISO/IEC03]. Perinteisesti relaatiotaulun tietojen koostaminen on tehty käyttämällä jotakin SQL:n viidestä aggregointifunktiosta (COUNT, SUM, AVG, MIN, MAX) yhdessä GROUP BY -operaattorin kanssa. Evaluoitaessa aggregointifunktioita ja GROUP BY -lauseen sisältävää SQL-kyselyä, relaatiotaulun rivit jaetaan ensin annetun ryhmittelytekijän (tai ryhmittelytekijöiden) mukaisiin ryhmiin, minkä jälkeen jokaisen ryhmän sisällä suoritetaan halutut aggregoinnit kyselyn SELECT-lauseessa määriteltyjen attribuuttien arvojen suhteen. SQL:n aggregointifunktiot ja GROUP BY -operaatio tuottavat nolla- tai yksiulotteisia aggregaatioita.

(1)

B	C	D
x	y	1
x	y	2
x	y	3
v	z	4
v	z	5

(2)

B	C	D
x	y	6
v	z	9
x	ALL	6
v	ALL	9
ALL	y	6
ALL	z	6
ALL	ALL	15

Kuva 2.3: Kaksi tietokuutioiden taulukkoesitystä.

DATA CUBE -operaatio tuottaa puolestaan n-ulotteisia aggregaatioita (missä n on dimensioiden lukumäärä). Operaation soveltaminen annetulle relaatiotaululle tarkoittaa nimittäin sitä, että ensin suoritetaan perinteiset GROUP BY -aggregoinnit kyselyssä määriteltyjen taulun sarakkeiden kaikkien mahdollisten kombinaatioiden suhteen ja käyttäjälle esitetään sitten kyselyihin saatujen vastausten unioni. Olkoon $A(B, C, D)$ relaatiokaavio, jossa A on relaation (tietokuution) nimi, B ja C dimensioattribuutteja ja D mitta-attribuutti. Relaation A ilmentymä on esitetty kuvan 2.3 kohdassa (1). Ajatellaan nyt, että relaatiosta A halutaan muodostaa DATA CUBE -operaatiolla tietokuutio, jossa (mitta-)attribuutin D arvot on (edelleen) koostettu SUM-funktiota käyttäen. DATA CUBE -operaation suorittaminen relaatiolle A vastaa esimerkin 2.1 neljän SQL-kyselyn

vastausten unionin muodostamista [NHJ02]:

Esimerkki 2.1:

(1)

```
SELECT SUM(D)
FROM A
```

(2)

```
SELECT B, SUM(D)
FROM A
GROUP BY B
```

(3)

```
SELECT C, SUM(D)
FROM A
GROUP BY C
```

(4)

```
SELECT B, C, SUM(D)
FROM A
GROUP BY B, C
```

Esimerkin kyselyssä (1) lasketaan sarakkeen D kaikkien arvojen summa. Kyselyssä (2) lasketaan sarakkeen D arvojen summa sarakkeen B kaikkien erillisten arvojen suhteen ryhmiteltyinä. Kyselyssä (3) lasketaan vastaavasti sarakkeen D arvojen summa sarakkeen C kaikkien erillisten arvojen suhteen ryhmiteltyinä. Lopuksi kyselyssä (4) lasketaan sarakkeen D arvojen summa sarakkeiden B ja C kaikkien erillisten arvokombinaatioiden suhteen ryhmiteltyinä.

Jos ajatellaan kyselyiden (1) - (4) vastauksena saatavaa tietokuutiota, niin siinä on nyt runsaasti soluja, joissa ei ole lainkaan sisältöä (tai vaihtoehtoisesti sisältönä NULL-arvo). Tämä johtuu siitä, että relaation kaikki sarakkeet eivät esiinny kaikissa kyselyissä. Saatava tietokuutio ei enää kuitenkaan ole relaatiomallin mukainen, sillä relaation avaimessa ei voi esiintyä tyhjiä arvoja (eikä NULL-arvoja). Grayn ja muiden ratkaisu ongelmaan on erityisen ALL-arvon käyttöönottaminen. ALL-arvo on arvo, jota ei ole määritelty (samalla tavoin kuin NULL-arvo on tyhjä arvo, jota ei ole määritelty). Käyttämällä kyselyissä (1) - (4) ALL-arvoa korvaamaan puuttuvia sarakkeita (esimerkiksi kyselyn (1) SELECT-lause olisi nyt muotoa SELECT 'ALL', 'ALL', SUM(D)), saadaan kuvan 2.3 kohdan (2) tietokuutio.

DATA CUBE -operaatio on olennaisesti SQL:n GROUP BY -operaation yleistys, sillä DATA CUBE -operaattorilla esimerkin 2.1 neljä erillistä SQL-kyselyä saadaan tiivistettyä yhdeksi kyselyksi:

Esimerkki 2.2:

```
SELECT B, C, SUM(D)
FROM A
GROUP BY CUBE B, C
```

DATA CUBE -operaatio ei kuitenkaan yksinään ole riitävä toteuttamaan moniulotteisessa analyysissä vaadittavaa toiminnallisuutta. Jos esimerkiksi relaatiotaulussa, johon

DATA CUBE -operaatiota sovelletaan, on sarakkeiden välillä funktionaalisia riippuvuuksia, voi operaation suorittamisen seurauksena saatavassa tietokuutiossa olla merkityksetöntä tietoa. Esimerkiksi *päivämäärä* tavallisesti määrää funktionaalisesti *vuoden*, *kuukauden* ja *viikon*. Samoin roll-up-operaatiot vuoden, viikon ja päivän suhteen ovat tavallisia, mutta tietokuutio, joka sisältää kaikki edellä mainitut attribuutit, olisi kuitenkin merkityksetön. Siksi Gray ja muut ovat kehittäneet lisäksi (relationaalisen) ROLLUP -operaattorin, joka tuottaa ainoastaan ylimmän tason aggregaatit eikä kaikkien attribuuttien yhdistelmää.

2.2 Fyysiset toteutukset

Moniulotteisen tietomallin toteuttamiseen tietokantatasolla on kolme perusvaihtoehtoa: tietokuution fyysisessä toteutuksessa voidaan hyödyntää relaatiomallin ominaisuuksia (relationaalinen OLAP eli ROLAP) tai tietokuutio voidaan esittää suoraan moniulotteisena taulukkorakenteena (moniulotteinen OLAP eli MOLAP). Risteytysratkaisuissa (hybridi OLAP eli HOLAP) pyritään minimoimaan kummankin edellä mainitun toteutustavan ei-toivotut piirteet. Lisäksi on kehitetty olio-orientoituneita tapoja (ks. esim. [NMW00]) toteuttaa moniulotteinen tietomalli (olio-orientoitunut OLAP eli OOLAP), mutta ne jäävät tässä tutkielmassa tarkastelun ulkopuolelle.

2.2.1 Moniulotteinen OLAP

MOLAP-järjestelmissä moniulotteinen avaruus linearisoidaan (moniulotteiseksi) taulukoksi [Sho97], jonka dimensiot vastaavat tietokuution dimensioita. Tietokuution mitta-attribuutin sijainti moniulotteisessa taulukossa määritellään indeksein. Indeksoinnin avulla dimensioiden erilliset arvot talletetaan taulukkoon vain kertaalleen. Esimerkiksi ROLAP-toteutuksiin verrattuna indeksointi tuo tilasäästöjä, sillä ROLAP-toteutuksissa samassa relaation visualisoinnin sarakkeessa voivat samat arvot esiintyä useamman kerran. Taulukkolinearisointi toimii hyvin, kun esitettävä moniulotteinen avaruus on tiivis, so. taulukon jokaista solua vastaa jokin mitta-attribuutin arvo eikä ole tyhjiä arvoja. Harvan tietokuution sisällön tiivistämiseksi on kehitetty erilaisia menetelmiä (ks. esim. [NNNT02]).

2.2.2 Relationaalinen OLAP

ROLAP-järjestelmissä hyödynnetään suoraan relaatiotietokantojen piirteitä ja tallennusrakenteina käytetään relationaalisia tauluja. ROLAP-järjestelmässä on relationaalisen taustapalvelimen lisäksi välittäjäkerros, jossa käyttäjän antaman kyselyn perusteella identifioidaan materialisoidut näkemykset, joita pystytään kyselyssä käyttämään hyväksi, muotoillaan käyttäjän tekemä kysely materialisoitujen näkemysten mukaiseksi ja lopuksi lähetetään kysely palvelimelle [CDG01]. ROLAP-järjestelmissä tietokantakyselyt tehdään SQL-tyyppisillä kyselykielillä. Yleisiä ROLAP-järjestelmien toteutustapoja ovat *tähtimalli* (star schema) ja sen muunnelmat *lumihiutale-* (snowflake schema) ja *konstellaatiomalli* (constellation schema), joita tarkastellaan seuraavaksi tarkemmin.

Tähtimalli. Tähtimalli muodostuu yhdestä keskitetystä faktataulusta ja joukosta sen ympärille ryhmitettyjä dimensiotauluja. Faktataulu ja dimensiotaulut yhdistetään toisiinsa relaatiomallin tapaan pää- ja viiteavainten kautta. Faktataulun rivit vastaavat alkuperäisen tietokuution mitta-attribuutteja. Faktataulun riveille sijoitetaan mitta-attribuutteja vastaavat arvot ja niihin liittyvien dimensioattribuuttien arvot, jotka toimivat samalla relaation viiteavaimena. Alkuperäisen tietokuution jokaista dimensiota varten on oma dimensiotaulunsa. Dimensiotaulun sarakkeet koostuvat ominaisuuksista, jotka liittyvät kyseiseen dimensioattribuuttiin.

Dimensiotaulut ovat tähtimallissa normalisoimattomia tai ensimmäisessä normaalimuodossa, mikä tarkoittaa, että niissä esiintyy redundanssia. Redundanssi aiheutuu siitä, että dimensiohierarkioita ei tähtimallissa ilmaista eksplisiittisesti, vaan dimensiohierarkian eri tasojen attribuutit ovat samassa dimensiotaulussa. Dimensiot vievät kuitenkin yleensä vain hyvin pienen osan tietokuution vaatimasta kokonaislevytilasta, joten redundanssista ei aiheudu tilankäytöllisesti suurta haittaa [PJ01]. Koska dimensiohierarkian eri tasojen tiedot on talletettu samaan tauluun, eivät dimensioiden päivitykset vaikuta datan yhdenmukaisuuteen. Tähtimallin etu on yksinkertainen ja helposti käsiteltävä rakenne, joka antaa mahdollisuuden tehokkaasti prosessoitavien kyselyjen yksinkertaiseen muotoilemiseen. Tähtimallissa haittapuolena on, että dimensiohierarkia ei ole selvästi nähtävissä.

Lumihiutalemalli. Lumihiutalemalli on normalisoitu versio tähtimallista. Lumihiutalemallissa on jokaista dimensiohierarkian tasoa kohti oma dimensiotaulunsa, joka on rakenteeltaan tähtimallin dimensiotaulun kaltainen paitsi, että alemman hierarkiatason dimensiotaulussa on viiteavaimena sitä välittömästi ylemmän hierarkiatason dimensiotaulun pääavain. Vaihtoehtoiset hierarkiatasot on lumihiutalemallissa helppo ilmaista. Dimensiohierarkioiden määrästä ja syvyydestä riippuen lumihiutalemallin mukaisessa tietokannassa on tähtimalliin verrattuna suurempi määrä tauluja, mikä voi joissakin tapauksissa olla haittaava tekijä. Lumihiutalemallissa dimensiohierarkiat ovat kuitenkin selvästi nähtävissä.

Konstellaatiomalli. Niemi ja muut [NHJ03] näkevät, että kompleksiset moniulotteiset sovellusalueet sisältävät usein dataa, joka liittyy eri käyttökonteksteihin. Siksi toisinaan on selkeämpää tehdä jokaista kontekstia kohden oma faktataulu kuin sisällyttää kaikki dimensioattribuutit samaan tauluun. Tällöin faktataulussa on ainoastaan ne dimensioattribuutit, jotka ovat kaikille taulun mitta-arvoille yhteisiä. Yleisesti mallia, jossa on dimensiotaulujen lisäksi useita faktatauluja, kutsutaan konstellaatiomalliksi [CD97]. Faktataulut voivat jakaa keskenään samoja dimensiotauluja. Konstellaatiomallissa dimensiohierarkiat voidaan esittää implisiittisesti (tähtimallin tapaan) tai eksplisiittisesti (kuten lumihiutalemallissa). Tähti- ja lumihiutalemalliin verrattuna konstellaatiomallissa on uutta se, että myös faktataulut voivat olla keskenään hierarkkisessa suhteessa. Konstellaatiomallin avulla pystytään tuottamaan tähti- ja lumihiutalemallia ilmaisuvoimaisempi kuvaus moniulotteisesta avaruudesta.

Niemi ja muut [NHJ03] esittelevät konstellaatiomallin variantin, joka poikkeaa stan-

dardimallista siinä, että faktataulussa voi olla dimensioita, joihin ei liity dimensiotaulua ja dimensiohierarkian tasot ilmaistaan erillisessä hierarkiataulussa. Faktatauluihin, joita mallissa kutsutaan tietokuutioiksi, ei liity hierarkiatauluja. Dimensiotaulut linkittyvät keskenään niissä esiintyvien yhteisten arvojen kautta. Faktatauluissa oleva data on aina dimensiohierarkian alimman tason mukaista. Esitetty konstellaatiomallin muunnelma on antanut Niemelle ja muille mahdollisuuden kehittää ilmaisuvoimaisen ja käyttäjäystävällisen OLAP-kyselykielen.

On syytä huomata, etteivät edellä esitellyt ROLAP-tietokannan toteutustavat ole pelkästään fyysisiä tietomalleja. Kyseisissä malleissa ei nimittäin sinällään oteta kantaa siihen, miten ne fyysisesti tietokantajärjestelmässä implementoidaan. Tähti-, lumihiutale- ja konstellaatiomalleja voidaan siten pitää myös moniulotteisina loogisina tietomalleina.

2.3 Summautuvuus ja aggregointifunktiot

Summautuvuus (summarizability) liittyy tietokuution tietojen oikeelliseen koostamiseen. Summautuvuus on erittäin tärkeä tietokuution ominaisuus, sillä virheellisesti tehty tietojen koostaminen (joka ei aina ole ilmeistä) johtaa väärään tilanneanalyysiin ja sitä kautta vääriin päätelmiin ja lopulta vääriin päätöksiin [LS97]. Jos virheellinen analyysi tapahtuu päätöksenteon tukijärjestelmässä, voi sillä olla huomattavat kielteiset seuraukset. Siksi on tärkeää varmistaa tietokuution tietojen oikeellinen koostaminen.

Summautuvuus ilmaisee sen, milloin dimensiohierarkian alemman tason aggregoituja arvoja voidaan käyttää ylemmän tason arvojen laskemiseen ja milloin ylemmän tason arvot pitää laskea lähtien alimmalta hierarkiatasolta tai perusdatasta (base data), joka sisältää yksityiskohtaisia tietoja [PRP02b]. Intuitiivisesti tarkasteltuna aggregoitujen tietojen koostaminen on oikeellinen, jos edelleenkoostamisoperaatio tuottaa täsmälleen saman lopputuloksen kuin, jos koostaminen olisi tehty perusdatasta löytyvien tietojen perusteella. Yhteenvetotietojen summautuvuus (tai ei-summautuvuus) riippuu dimensiohierarkian rakenteesta, käytettävästä aggregointifunktiosta ja dimensio- ja mitta-attribuuttien tyypistä.

Lenz ja Shoshani [LS97] määrittelevät dimensiohierarkioille kaksi summautuvuusehdoiksi kutsumaansa ehtoa, jotka dimensiohierarkian pitää täyttää, jotta mitta-arvoja voidaan koostaa oikeellisesti kyseisen dimension suhteen. Lenz ja Shoshani käyttävät dimensiohierarkian visualisoimisessa Rafanellin ja Shoshanin [RS90] esittämää STORM-mallia, jossa dimensiohierarkia esitetään graafina. Ensimmäisen summautuvuusehdon mukaan dimensioattribuuttien arvojen pitää muodostaa keskenään erilliset (disjoint) joukot, ts. mikään perusdataan kuuluva yksilö/olio ei saa kuulua useampaan kuin yhteen kategoriaan saman dimension sisällä. Jos esimerkiksi työntekijä voi olla kirjoilla saman organisaation sisällä kahdessa eri yksikössä (dimensioattribuutti yksikkö), tuottaa organisaation kokonaistyöntekijämäärän laskeminen sen yksiköiden työntekijämäärien perusteella virheellisen tuloksen. Toinen summautuvuusehto on, että perusdataan kuuluvien yksilöiden/olioiden ryhmittely on täydellinen (complete). Täydellisyys merkitsee kahta asiaa. Ensinnäkin perusdatasta muodostettujen joukkojen unionin pitää muodostaa ko-

ko universumi. Toisekseen jokaisen perusdataan sisältyvän yksilön/olion pitää liittyä johonkin dimensiokategoriaan. Toisen summautuvuusehdon tarkoitus on varmistaa, että koostetiedoista ei jää puuttumaan arvoja ja toisaalta että perusdatassa ei ole myöskään ylimääräisiä arvoja. Täydellisyysehto on riippuvainen mitta-attribuutista.

Lenz ja Shoshani [LS97] esittävät myös kolmannen summautuvuusehdon, jota he kutsuvat tyyppiyhteensopivuudeksi. Tyyppiyhteensopivuuteen vaikuttavat sekä dimensioattribuutin, mitta-attribuutin että käytettävän aggregointifunktion tyyppi. Jotkut dimensioattribuutit ovat luonteeltaan temporaalisia eli liittyvät aikaan. Tietojen koostaminen temporaalisen dimension suhteen eroaa merkittävästi tietojen koostamisesta ei-temporaalisen dimension (esimerkiksi sukupuoli, etninen alkuperä tai maantieteellinen alue) suhteen. Myös mitta-attribuuteista monet liittyvät aikaan. Lenz ja Shoshani jakavat mitta-attribuutit vaihtuviin, kiinteisiin ja mittayksikkösidonnaisiin. Vaihtuvat mitta-attribuutit (flow) liittyvät tiettyyn ajanjakson ja niiden arvo muodostetaan ko. ajanjakson päätteeksi. Esimerkkejä vaihtuvantyyppisestä mitta-attribuutista on "kuukaudessa syntyneiden lasten määrä" ja "vuositulo". Kiinteiden mitta-attribuuttien (stock) arvo puolestaan kuvaa tiettynä ajanhetkenä vallitsevaa tilannetta. Esimerkkejä tämäntyyppisestä mitta-attribuutista ovat "varaston koko" ja "asukasluku". Vaihtuvantyyppiset mitta-attribuutit kuvaavat tiettynä ajanjaksona etenevää kumulatiivista vaikutusta, kun taas kiinteää tyyppiä edustavat mitta-attribuutit heijastavat tiettynä ajanhetkenä mitattua asiantilaa. Mittayksikkösidonnaiset mitta-attribuutit koskevat myös kiinnitettyä ajanhetkeä, mutta ne eroavat kiinteää tyyppiä olevista mitta-attribuuteista siinä, että niissä on mukana jokin mittayksikkö. Mitta-attribuutti "ajoneuvoinventaari" on mittayksikkösidonnaista tyyppiä, jos attribuutin arvo ilmoitetaan esimerkiksi "euroa/ajoneuvo". Jos em. mitta-attribuutin arvo ilmoitetaan pelkästään ajoneuvon rahallisena arvona tai ajoneuvojen lukumääränä, on se kiinteää tyyppiä.

Mitta-attribuutin arvojen ei-summautuvuus riippuu dimensiosta, jonka mukaan arvojen koostaminen suoritetaan, eikä se siis ole mitta-attribuutin absoluuttinen ominaisuus. Yleisesti ei ole mahdollista laskea yhteenvetoja ei-summautuvista mitta-arvoista, sillä mitta-arvojen muodostamisessa käytetyt semanttiset säännöt eivät välttämättä ole tunnettuja. Ei-summautuvien mitta-arvojen yhteenvetotiedot pitää aina laskea yksittäistiedoista koostuvasta perusdatasta lähtien, jos sellainen on saatavilla.

Aggregointifunktiot voidaan erotella niiden laskennan kompleksisuuden mukaan [LT01]. Yksinkertaisimmat aggregointifunktiot perustuvat yksivaiheiseen aggregointiin. Tällaisia aggregointifunktioita ovat (absoluuttinen) frekvenssi (COUNT), aritmeettinen summa (SUM), aritmeettinen keskiarvo (AVG) sekä minimi (MIN) ja maksimi (MAX). Kompleksisemmat aggregointifunktiot perustuvat kumulatiivisten tai liikkuvien tilastollisten arvojen käsittelyyn. Niissä yksittäisiä arvoja tarkastellaan suhteessa koko saatavilla olevaan aineistoon. Kompleksiset aggregointifunktiot perustuvat yleensä kaksivaiheiseen aggregointiin, jonka ensimmäisessä vaiheessa aineisto ryhmitellään ja toisessa vaiheessa haluttu lopputulos muodostetaan jokaisen ryhmän suhteen. Esimerkkejä kompleksisemmista aggregointifunktioista ovat moodi ja keskihajonta.

Lenz ja Thalheim [LT01] luokittelevat aggregointifunktiot distributiivisiin, algebrallisiin ja holistisiin funktioihin. Distributiiviset funktiot, jotka tunnetaan myös induktii-

visina funktioina, perustuvat rakenteelliseen rekursioon (structural recursion). Distributiiviset funktiot säilyttävät näin ollen joukkojen osituksen. Toisin sanoen jos on joukko X ja sen ositus $X = \{X_1 \cup X_2 \cup \cdots \cup X_n\}$, jossa jokainen X_i on joukon X erillinen osajoukko, jokaista distributiivista funktiota f kohden on olemassa funktio g siten, että $f(X) = g(f(X_1), f(X_2), \ldots, f(X_n))$. Frekvenssi, aritmeettinen summa sekä minimi ja maksimi ovat distributiivisia aggregointifunktioita. Algebralliset funktiot voidaan ilmaista distributiivisten funktioiden kautta määriteltyinä äärellisinä algebrallisina ilmauksina. Esimerkki algebrallisesta funktiosta on (aritmeettinen) keskiarvo, joka määritellään summan ja frekvenssin avulla. Jos funktio ei ole distributiivinen eikä algebrallinen, on se holistinen. Holistisia funktioita ovat mm. aste (rank) ja mediaani.

Jotkut kompleksiset tilastolliset funktiot, kuten keskiarvo ja -hajonta, ovat summautuvia, jos niiden komponentit ovat käytettävissä (esimerkiksi keskiarvon tapauksessa summa ja lukumäärä). Toisaalta on myös funktioita, jotka ovat luonnostaan ei-summautuvia. Tällaisia funktioita ovat esimerkiksi mediaani, prosenttipisteet ja painotettu keskiarvo. Näiden funktioiden kohdalla ei-summautuvuus on riippumaton suhteessa käytettävään mitta-arvoon tai dimensioattribuuttiin, jonka suhteen koostaminen tehdään [LS97].

2.4 Tietovarastointi

Moniulotteisessa analyysissa käytettävät tietokuutiot konstruoidaan tyypillisesti *tietovarastoihin* (data warehouse) kootuista tiedoista. Tietovaraston tarkoitus on tarjota keskitetty pääsy mahdollisesti heterogeenisissa, autonomisissa ja/tai hajautetuissa tietolähteissä oleviin tietoihin. [Abi03] Tietovaraston sisältö on tyypillisesti johonkin aihepiiriin keskittynyt (subject-oriented), useasta tietolähteestä integroitu, ajan suhteen jaksotettu (time-variant) ja suhteellisen vakaa (non-volatile) kokoelma dataa [SS05]. Perinteisesti tietovaraston tiedot on kerätty operationaalisen tietojärjestelmien tiedoista keskitettyyn paikkaan, mutta tietoverkkojen käytön lisääntyessä on viime aikoina alettu myös puhua ns. verkkotietovarastoista (web warehousing), jotka ovat hajautetun tietovarastoinnin yksi muoto.

Perinteisesti tietovarasto on sijoitettu yksittäiselle palvelimelle tai usean toisiinsa liitetyn tietokoneen muodostamaan klusteriin, johon kaikki relevantti informaatio on keskitetty ja jonka avulla se on hallittavissa (manage). Tietovarasto on useimmiten operationaalisista tietojärjestelmistä erillinen kokonaisuus, vaikka sen tiedot onkin saatu yhdistelemällä operatiivisissa tietokannoissa (ja toisinaan myös muissa sähköisissä tiedonlähteissä) olevia tietoja. Moniulotteisessa analyysissa ei analysoida suoraan tietovaraston lähtötietoja [CT98] vaan siitä koostettuja tietokuutioita, jotka on tallennettu relaatiomuodossa tai moniulotteisina taulukoina erillisille OLAP-palvelimille. OLAP-palvelimilla kyselyjen evaluointi tapahtuu näin ollen useimmiten ilman pääsyä alkuperäisiin tietolähteisiin.

Tietovarasto muodostetaan tavallisesti ETL (*Extract, Transform, Load*) -prosessin kautta. Jos tietovaraston muodostamisessa käytetään keskenään heterogeenisia tietolähteitä, täytyy niissä olevaa dataa yleensä esikäsitellä, ennen kuin se voidaan asettaa

analysointikäyttöön. Jos käytettävissä oleva data perustuu erilaisiin tallennusmuotoihin (eri kaavioita käyttävät relaatiotietokannat, muut tietoformaatit), täytyy data muuntaa (transform) keskenään yhdenmukaiseen muotoon ennen tietovarastoon saattamista. Kaikkea mahdollista saatavissa olevaa dataa ei ole myöskään aina syytä ottaa mukaan tietovarastoon, vaan analysointitarpeiden kannalta epärelevantti informaatio on syytä suodattaa (filter) pois jo tietojen keruuvaiheessa. Näiden toimenpiteiden jälkeenkin datassa voi vielä esiintyä epäjohdonmukaisuutta (inkonsistenssi), esimerkiksi tietojen useampikertaisia esiintymiä, joka poistetaan siistimällä (clean) data, ennen kuin se prosessin päätteeksi ladataan tietovarastoon.

Moniulotteisen analyysin toiminnalliset ja suorituskykyyn liittyvät vaatimukset eroavat perinteisille OLTP-pohjaisille tietokannoille asetetuista vaatimuksista. Koska tietovarastoihin tallennettu tieto on usein johdettu useiden erillisten operationaalisten tietokantojen tiedoista ja koska siellä säilytetään eri ajanjaksoilta peräisin olevia tietoja, on tietovaraston tilantarve operatiivisten tietokantojen tilantarpeeseen nähden tavallisesti huomattavasti suurempi: Chaudhurin ja Dayalin [CD97] mukaan tyypillisen operationaalisen tietokannan koko vaihtelee sadoista megabiteistä gigabitteihin, kun taas tietovarastojen kokoa mitataan yleensä sadoista gigabiteistä terabitteihin. Yksittäisen tietovaraston tilantarvetta voidaan helpottaa hajauttamalla se pienemmiksi paikallisvarastoiksi (data mart).

Tietovarastoihin tallennettu data eroaa myös luonteeltaan operatiivisissa tietojärjestelmissä käytettävästä datasta. Chaudhuri ja Dayal [CD97] korostavat, että moniulotteisessa tiedon analyysissa monesti tarvitaan sellaista dataa, joka saattaa puuttua operationaalisista tietojärjestelmistä: nykyisten tai menneiden asiantilojen arvioiminen ja tulevan kehityksen ennustaminen perustuu nimittäin usein tiettyyn ajanhetkeen sidotun datan analysointiin. Operationaalisissa tietojärjestelmissä sitä vastoin käytetään useimmiten ainoastaan reaaliaikaisesti päivittyvää dataa eikä niissä siten ole tarvetta säilyttää aktiivikäytöstä poistettua dataa. Operationaalisissa tietojärjestelmissä oleva data on usein myös jatkuvasti muuttuvaa, kun taas tietovarastoissa olevaa dataa päivitetään yleensä ainoastaan jaksottaisesti silloin, kun käsiteltävän aineiston aikadimension alin taso muuttuu (esimerkiksi päivittäin tai viikoittain).

Tietovaraston kuormitus on kyselypainotteista. Moniulotteisessa analyysissa ovat tyypillisiä ennakoimattomat, kompleksiset kyselyt, joiden suorittaminen perinteisissä tietokantajärjestelmissä saattaisi tarkoittaa miljoonien tietueiden läpikäyntiä ja niiden liitoksia ja arvojen yhdistelyjä [CD97]. OLTP-pohjaisten relaatiotietokantojen suunnittelun yhteydessä on ollut tärkeää varmistaa tavanomaisten tietokantatapahtumien (lisäykset, poistot, ennalta määrätyt kyselytarpeet) mahdollisimman sujuva suorittaminen. Siksi moniulotteisessa analyysissa tarvittavien, perinteisillä relationaalisilla kyselykielillä toteutettujen kyselyiden suorittaminen relaatiotietokannoissa johtaa helposti heikkoon suorituskykyyn ja pitkiin vasteaikoihin. On kuitenkin syytä painottaa, ettei tietovarastojen ja operationaalisten tietokantojen ero ole kategorinen, vaan niillä on myös runsaasti yhtymäkohtia (ks. esim. [ZS99]).

Metadatan hallinta on tietovarastoarkkitehtuurin olennainen elementti, sillä sen avulla tietovarastoa voidaan valvoa, ymmärtää ja kehittää. Tietovarastoihin liittyy usean

tyyppistä metadataa. Hallinnolliseen metadataan sisältyy kaikki tietovaraston pystyttämiseen ja käyttöön liittyvä informaatio. Liiketoiminnalliseen metadataan sisältyvät liiketoimintatermit ja niiden määritelmät, tiedot tiedon omistajuudesta ja muuta vastaavaa. Operationaaliseen metadataan sisältyy tietovaraston muodostamisen ja toiminnan aikana kerätty informaatio, esimerkiksi tiedot tietovarastoon siirretyn ja siellä muunnetun datan alkuperästä sekä erilaisia valvontatietoja, kuten käyttötilastot, virheraportit ja seurantaketjut (audit trail). Metadata säilytetään yleensä erillään varsinaisesta tietovarastosta. [CDG01]

Perinteiset tietovarastot ovat keskittyneet tietojen *fyysiseen* integrointiin. Tietovarastoihin on kerätty useista tietolähteistä dataa, joka on sitten yhdistetty keskenään ja tallennettu moniulotteisen tietokannan kaavion mukaisena yhteen keskitettyyn säilytyspaikkaan. Tällainen fyysisesti keskitetty tietovarasto mahdollistaa kompleksisten kyselyiden tehokkaan evaluoimisen, mutta asettaa toisaalta myös haasteita tietovaraston ylläpitämiselle. Kuitenkin on myös olemassa sellaista dataa, jota ei voida – esimerkiksi lainsäädännön vaatimusten tai tietojen nopean muuttumisen vuoksi – tallentaa tietovarastoon mutta jota kuitenkin tarvitaan moniulotteisessa analyysissa. Tällaisten tietojen integroitu käyttö vaatii tietojen fyysisen integroinnin sijasta *loogista* integrointia [JMP01].

Vaikka OLAP ja tietovarasto ovat läheisesti toisiinsa liittyviä käsitteitä, niillä on kuitenkin painotuksellinen ero. Tietovarastossa huolehditaan ensisijaisesti sinne tuotavan datan integroinnista ja varastoinnista sekä varmistetaan, että se on eheää ja oikeellista. Tietovarasto tuottaa ja ylläpitää tietokuutiota; OLAP-sovellus puolestaan käsittelee tietovaraston tuottamaa kuutiota luottaen siihen, että se on oikeellinen ja ajantasainen.

Luku 3

XML-merkintäkieli

3.1 Syntaksi

SGML- ja HTML-dokumenttien tavoin myös XML-dokumentin perusyksikkö on elementti. XML-syntaksissa alkutunniste aloitetaan pienempi kuin -merkillä (<), jota seuraa elementin nimi, ja alkutunnisteen päättää suurempi kuin -merkki (>). Lopputunniste on muutoin samanlainen kuin alkutunniste, mutta tunnisteen aloittavaa pienempi kuin -merkkiä seuraa välittömästi kauttaviiva (/). Elementin nimen on elementin alku- ja lopputunnisteessa oltava sama. Elementin nimen (jokin merkkijono, jonka sisällölle on asetettu tiettyjä rajoituksia) dokumentin laatija saa vapaasti valita.

Elementin yhteydessä voidaan esittää myös attribuutteja niihin kuuluvine arvoineen. Attribuutit esitetään elementin alkutunnisteessa elementin nimen ja alkutunnisteen päättävän suurempi kuin -merkin väliin jäävässä osassa. Attribuutin esitys koostuu attribuutin nimestä, jota seuraavat yhtäsuuruusmerkki (=) ja lainausmerkkien (") sisällä attribuutin arvo. Attribuutin arvona voi olla ainoastaan rakenteetonta teksiä. Elementin sisältämät attribuutit erotetaan toisistaan välilyönnillä ja niiden lukumäärälle ei ole asetettu rajoituksia.

Attribuutit ovat jäänne SGML:stä ja HTML:stä, joissa niillä esitetään sellaista tietoa, joka on olennaista dokumentin kannalta mutta jota ei kuitenkaan haluta eksplisiittisesti esittää dokumentin lukijalle. Myös XML:ssä attribuuteilla ilmaistaan tyypillisesti tiettyjä elementtiin liittyviä ominaisuuksia. Koska XML-dokumentti on kuitenkin ensisijaisesti tarkoitettu tietokoneen luettavaksi, ei XML-dokumentissa attribuuteilla ole enää SGML- ja HTML-attribuuttien kaltaista merkitystä. Esimerkiksi Suciu [Suc00] esittää tavan, joilla merkityn tekstin attribuutit voidaan korvata elementtirakenteilla. Attribuuttien käyttöä voidaan perustella joissakin tapauksissa dokumenttirakenteen supistamisella, jos sisäkkäisiä elementtejä on runsaasti.

Elementin alku- ja lopputunnisteiden väliin jäävää osaa sanotaan elementin *sisällöksi* (content). Elementin sisältö voi koostua rakenteettomasta tekstistä ja/tai muista elementeistä. On syytä huomata, että rakenteettomallakin tekstillä voi tosiasiassa olla oma sisäinen rakenteensa. Tässä rakenteettomalla tekstillä tarkoitetaan XML-dokumentin osaa, jonka rakennetta ei ole eksplisiittisesti merkitty. Elementit voivat siis muodostaa keske-

```
(1)  <?xml version="1.1" encoding="UTF-8"?>

(2)  <articles>
(3)    <article no="0086">
(4)      <author>
(5)        <initials> E. F. </initials>
(6)        <lastname> Codd  </lastname>
(7)      </author>
(8)      <title>
(9)        A Relational Model for Large Shared Data Banks
(10)     </title>
(11)     <year> 1970 </year>
(12)   </article>
(13)   <article>
(14)     <author id="0314"> Peter P. Chen </author>
(15)     <title>
(16)       An Entity-Relationships Model:
(17)       <subtitle> Toward a Unified View of Data </subtitle>
(18)     </title>
(19)     <year> 1976 </year>
(20)   </article>
(21) </articles>
```

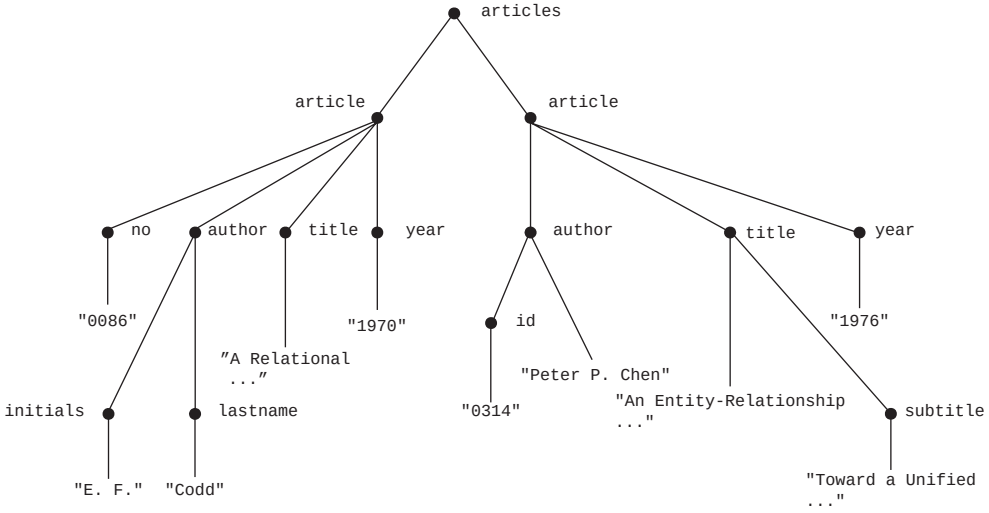
Kuva 3.1: Esimerkkidokumentti.

nään *sisäkkäisen* (nested) rakenteen. Alielementin on sijaittava täydellisesti yläelementtinsä sisällä, eli jos elementin alkutunniste on jonkin toisen elementin sisältöosassa, pitää elementin lopputunnisteenkin kuulua saman elementin sisältöosaan. Kuvassa 3.1 on esimerkki XML-dokumentista.

Elementin sisältöosa voi myös puuttua, jolloin elementtiä sanotaan tyhjäksi. Tyhjän elementin merkitsemiseksi XML:ssä on kaksi tapaa: joko käytetään alkutunnistetta, jota seuraa välittömästi lopputunniste, tai erityistä *tyhjän elementin tunnistetta* (empty-element tag). Tyhjän elementin tunniste on ulkoasultaan kuten ei-tyhjän elementin lopputunniste paitsi, että kauttaviiva sijaitsee nyt välittömästi ennen tunnisteen päättävää suurempi kuin -merkkiä. Tyhjä elementti voi sisältää kuitenkin attribuutteja.

XML-dokumentin elementeistä erikoisasemassa on *dokumenttielementiksi* (document element) kutsuttu elementti, jonka sisällä dokumentin kaikki muut elementit sijaitsevat. Dokumenttielementin ansiosta XML-dokumentti voidaan esittää puumaisena tietorakenteena, jossa dokumenttielementti (jota kutsutaan myös juurielementiksi) on dokumenttipuun juuri ja dokumentin muut elementit ja attribuutit ovat puun solmuja. Attribuutit ja elementit, joilla on alielementtejä tai attribuutteja, ovat dokumenttipuun sisäsolmuja. Puun lehtisolmut muodostuvat attribuuttien arvoista ja rakenteetonta tekstiä sisältävien elementtien sisällöstä. On syytä huomata, että myös sisäsolmut voivat sisältää rakenteetonta tekstiä – seikka, joka kirjallisuudessa usein sivuutetaan. Koska XML-dokumentin elementin alielementtien lukumäärälle ei ole asetettu rajoituksia, on XML-dokumenttia kuvaava puu *asteeton* (unranked).

XML-dokumentin elementit ovat dokumentissa tietyssä järjestyksessä, jota sanotaan *dokumenttijärjestykseksi* (document order) [BCF+03]. Siten XML-dokumenttia kuvaava puurakenne on myös *järjestetty* (ordered). XML-dokumentin dokumenttijärjestys vas-



Kuva 3.2: XML-dokumentin puuesitys.

taa dokumenttipuun läpikäyntiä esijärjestyksessä syvyys ensin -suuntaisesti. Elementtiin kuuluvien attribuuttien voidaan katsoa olevan puuesityksessä elementin lapsisolmuja (ja ne indeksoidaan siinä järjestyksessä, jossa ne on elementin alkutunnisteessa esitetty). Attribuuttisolmulla voi (ja pitää) olla vain yksi lapsisolmu, johon on talletettu attribuutin arvo. Sekä rakenteetonta tekstiä että alielementtejä sisältävän elementin sisältökomponentit indeksoidaan dokumenttijärjestyksen mukaisesti. Koska dokumenttipuun jokaiseen solmuun, lehtisolmuja lukuunottamatta, liittyy solmua vastaavan elementin tai attribuutin nimi, on puu paitsi asteeton ja järjestetty myös *nimiöity* (labelled). XML-dokumenttia vastaava tietorakenne on siis kokonaisuudessaan nimiöity, asteeton, järjestetty puu. Kuvassa 3.2 on esitetty kuvan 3.1 esimerkkidokumentin puuesitys.

XML-dokumenteille on elementtien ja attribuuttien lisäksi määritelty myös muita syntaktisia rakenteita: *esittely* (declaration), *kommentti* (comment), *merkkiviittaus* (character reference), *entiteettiviittaus* (entity reference) ja *prosessointiohje* (processing instruction). Näillä ei ole kuitenkaan XML-esitystavan rakenteellisuuden kannalta keskeistä merkitystä, joten niiden käsittely – esittelyjä lukuun ottamatta – sivuutetaan tässä tutkielmassa. *XML-esittelyssä* (XML declaration) ilmoitetaan dokumentissa käytettävän XML-kielen versio sekä mahdollisesti muita tietoja. Kuvan 3.1 rivillä (1) on esimerkki tyypillisestä XML-esittelystä, jossa on annettu XML-version lisäksi dokumentissa käytössä oleva merkkikoodausjärjestelmä (character encoding). Yksinkertaisimmillaan XML-spesifikaation mukainen *hyvin-määritelty* XML-dokumentti koostuu XML-esittelystä ja yhdestä elementistä.

3.2 Puolirakenteisuus

XML on esimerkki puolirakenteisesta tiedosta (semistructured data). Puolirakenteinen tieto on dataa, joka ei ole "raakadataa" (ääntä tai kuvaa) mutta ei toisaalta myöskään tiukasti tyypiteltyä informaatiota [Abi97]. Puolirakenteista tietoa luonnehtii se, että datan rakenne on tyypillisesti epäsäännöllinen, implisiittinen ja osittainen. Epäsäännöllisellä rakenteella tarkoitetaan tässä sitä, että samaa informaatiota on mahdollista esittää datassa usealla toisistaan poikkeavalla tavalla. Implisiittinen rakenne puolestaan tarkoittaa sitä, että datan rakenne on tuotava esiin suorittamalla jokin laskentaoperaatio (esimerkiksi dokumentin jäsennys). Osittainen rakenne puolestaan tarkoittaa sitä, että jotkin datan osat voivat olla täysin rakenteettomia, joillakin voi olla ainoastaan luonnosmainen rakenne ja jotkin osat voivat olla hyvin rakenteellisia.

Puolirakenteisella tiedolla on ambivalentti suhde *kaavioon* (schema). Ensinnäkin puolirakenteiseen tiedon yhteydessä kaavio voi olla apriorinen tai aposteriorinen. Esimerkiksi relaatiotietokantaan tallennettavaa informaatiota kuvaava kaavio on aina jo (apriorisesti) olemassa, ennen kuin itse informaatio tallennetaan tietokantaan. Puolirakenteisen tiedon kohdalla kaavio pystytään kuitenkin usein tuottamaan vasta varsinaisen datan (esimerkiksi XML-dokumentin) luomisen jälkeen. Puolirakenteiseen dataan liittyvä kaavio on myös usein luonteeltaan ainoastaan suuntaa-antava. Seuraavassa luvussa puhutaan XML-dokumenttiin liitettävästä dokumenttityypinmäärityksestä (DTD), jonka avulla pystytään kuvaamaan dokumentin rakennetta. Dokumenttityypinmääritys ei kuitenkaan ole XML-dokumentin varsinainen kaavio, sillä se pystyy kuvaamaan ainoastaan dokumenttien luokan (dokumenttityypin) rakenteen mutta ei välttämättä yksittäisen dokumentin rakennetta. Puolirakenteisen tiedon yhteydessä ei ole harvinaista, että datan rakennetta kuvaava kaavio tuotetaan evolutionaarisesti, ts. jo olemassa olevia heterogeenisia kaavioita yhdistämällä. Kun sitten ajatellaan tällaista kaaviota, voidaan havaita, että syntyvästä kaaviosta tulee helposti suuri. Evolutionaarisen kaaviontuottamisprosessin kautta saatavien kaavioiden rakenne on lisäksi helposti muuttuva (vrt. relaatiotietokantojen staattinen kaavio). Puolirakenteista dataa on yleensä mahdollista käsitellä myös ilman kaavioinformaatiota.

Puolirakenteisesta tiedosta alettiin varsinaisesti puhua vasta 1990-luvun puolivälissä, jolloin aiheesta kirjoitettiin kaksi vaikutusvaltaista artikkelia ([Abi97] ja [Bun97]) ja kehitettiin useita kyselykieliä puolirakenteisen datan käsittelyyn (esim. Lorel [AQM+97], UnQL [BDHS96], MSL [PAGM96] ja StruQL [FFLS97a, FFLS97b]). Käsite "puolirakenteinen tieto" liitettiin tuolloin lähinnä SGML-merkintäkieleen. Puolirakenteisen tiedon käsittelyyn tarkoitetut kyselykielet ovat sittemmin antaneet vaikutteita monien XML-kyselykielten kehitykseen. Puolirakenteisen tiedon yhteydessä käytetyimmäksi tietomalliksi nousi OEM (Object Exchange Model) [PGMW95]. OEM on graafipohjainen, itsensä kuvaava oliotietomalli.

OEM-mallissa data käsitetään olioiden (object) kokoelmana. Jokainen olio voi olla atominen tai kompleksinen. Jokaisen atomisen olion arvo on jotakin perustietotyyppiä (`integer`, `string` jne.). Kompleksisen olion arvo on puolestaan joukko attribuuttiarvopareja. Data esitetään OEM-mallissa graafina, jonka solmut ovat olioita ja solmu-

jen väliset kaaret on nimetty niitä vastaavien attribuuttien nimien mukaan. Joihinkin graafin lehtisolmuihin liittyy atominen arvo. Graafissa on juurisolmu, jonka kautta graafin kaikki muut solmut ovat saavutettavissa [Suc00]. OEM-malli ei ole sinällään sidottu pelkästään puolirakenteisen tiedon kuvaamiseen, vaan sen avulla on mahdollista kuvata monentyyppistä informaatiota. XML-muotoinen informaatio kuitenkin eroaa puhtaasta OEM-tietomallista siinä, että XML-data on oletusarvoisesti järjestettyä (se noudattaa dokumenttijärjestystä), kun OEM-mallista puolestaan lähdetään siitä, että data on järjestämätöntä.

Puolirakenteista tietoa – ja erityisesti XML-muotoista tietoa – on hedelmällistä tarkastella suhteessa relaatiotietokantoihin tallennettuun informaatioon. Relaatiotietokantoihin tallennettu informaatio on esimerkki vahvasti rakenteisesta tiedosta. Relaatiotauluun (ja laajemmin koko yksittäiseen relaatiotietokantaan) liittyy aina kaavio, joka kuvaa sen, millainen relaatiotaulu on ja minkä tyyppistä dataa taulun kuhunkin sarakkeeseen on mahdollista tallentaa. Puolirakenteiselle tiedolle puolestaan on tyypillistä, että informaatio, joka tavallisesti on liitetty kaavioon, on löydettävissä itse datasta, ts. puolirakenteinen tieto on *itsekuvautuvaa* [Bun97]. Relaatiotietokantoihin tallennettu informaatio on ”litteää” – so. se on järjestetty riveistä ja sarakkeista koostuviksi kaksiulotteisiksi taulukoiksi. XML-muotoinen informaatio on puolestaan tyypillisesti tallennettu sisäkkäisiksi rakenteiksi ja XML-dokumentin informaation sisäkkäisyyden astetta ei oletusarvoisesti ole säädelty eikä se ole ennustettavissa. Relationaalinen data on säännöllistä ja homogeenista. Esimerkiksi jokaisella relaatiotaulun rivillä on aina sama määrä tietoalkioita ja jokaisen sarakkeen tietoalkiot ovat keskenään samaa tietotyyppiä. Tämä mahdollistaa sen, että relaatiotietokannassa metadata voidaan irrottaa itse varsinaisesta datasta ja säilyttää siitä erillään. Koska puolirakenteinen informaatio on puolestaan rakenteeltaan epäsäännöllistä ja heterogeenista, on puolirakenteisen informaation yhteydessä esitettävä relaatiotauluun verrattuna oleellisesti enemmän metadataa (esim. XML-dokumentissa alku- ja lopputunnisteet). Säännöllisen rakenteensa ansiosta on relationaalinen informaatio tiivistä – so. jokaisella rivillä on arvo jokaisessa sarakkeessa. XML-data voi puolestaan olla hyvin harvaa [Cham02].

Kuten jo aikaisemmin on nähty, on puolirakenteiselle tiedolle tyypillistä, että se voidaan esittää graafi- tai puumaisena tietorakenteena. Graafissa ei voida tehdä eroa kaavion ja ilmentymän välillä [Via03], mikä tarkoittaa sitä, että puolirakenteisen tiedon yhteydessä ero kaavion ja ilmentymän välillä häviää. Koska esimerkiksi XML-dokumentissa jokaiseen tietoelementtiin liittyy aina sen alku- ja lopputunnisteet (kaavio), ei tietoelementin sisältöä tai kaaviota ole mielekästä tarkastella toisistaan erillään.

XML-rakenteet tarjoavat lisäksi mahdollisuuksia samaa tarkoittavan informaation esittämiseen sekä kaavio- että ilmentymätasolla. Tarkastellaan asiaa esimerkin avulla. Ajatellaan, että halutaan laatia XML-dokumentti, jossa on sisältönä tietoja tietokantatutkimuksen (`db`) piiriin kuuluvista artikkeleista. Esimerkissä 3.2 on annettu neljä mahdollista tapaa, joilla haluttu informaatio voidaan esittää XML-dokumentissa.

Esimerkki 3.2:

(A)

```
<db_articles>
 <article> ...</article>
 ...
</db_articles>
```

(B)

```
<article>
 <domain> db </domain>
 ...
</article>
```

(C)

```
<article db=''yes'' ir=''no''>
 ...
</article>
```

(D)

```
<article domain=''db''>
 ...
</article>
```

Esimerkin kohdassa (A) tieto tutkimusalasta, jonka piiriin artikkeli kuuluu, on esitetty elementin nimessä. Kohdassa (B) tieto tutkimusalasta on puolestaan ilmaistu elementin sisältönä. Kohdassa (C) elementtiin on liitetty `db`-niminen attribuutti, jonka arvoa säätelemällä (`yes` tai `no`) ilmaistaan, kuuluuko artikkeli tietokantatutkimuksen piiriin. Toisin sanoen kohdassa (C) tieto tutkimusalasta esitetään attribuutin nimessä. Kohdassa (D) tutkimusala ilmoitetaan attribuutin arvona. Kohdissa (A) ja (C) informaatio on siis esitetty kaaviotasolla ja kohdissa (B) ja (D) ilmentymätasolla.

Myös relaatiotietokannoissa samaa tarkoittavaa informaatiota on mahdollista esiintyä sekä kaavio- että ilmentymätasolla (ks. esim. [LSS01]). XML:n yhteydessä lisävaikeutena kuitenkin on, että sama informaatio voi esiintyä kaikissa edellä mainituissa eri positioissa *yhden ja saman* XML-dokumentin sisällä. Informaation jakautuminen XML-dokumentissa eri positioihin aiheuttaa myös haasteita XML-kyselykielten kehittämiselle. Kuten seuraavissa luvuissa tullaan näkemään, useissa nykyisin käytössä olevissa XML-kyselykielissä ei kuitenkaan ole tätä tilannetta otettu huomioon.

Puolirakenteisen tiedon pääasiallinen viehätys on siinä, että se on rakenteeltaan vaihteleva. Puolirakenteisen tiedon yhteydessä esiintyy kuitenkin myös eräitä ongelmia. Ensinnäkin datan tallennus on usein tehotonta, sillä kaavio (esim. XML-dokumentin yhteydessä alku-ja lopputunnisteet) pitää replikoida jokaisen tietoalkion tallennuksen yhteydessä. Toisekseen puolirakenteiseen dataan tehtäviä kyselyitä on vaikea evaluoida tehokkaasti, sillä yksittäinen, yksinkertainenkin kysely voi pahimmassa tapauksessa vaatia koko datagraafin läpikäyntiä. Kyselyt ovat lisäksi hankalia muotoilla, sillä voidakseen muotoilla relevantin kyselyn pitää käyttäjän turvautua dokumentoimattomaan informaatioon datan rakenteesta.

3.3 Dokumenttikieliopit

Kuten on jo aiemmin todettu, dokumentin laatija voi vapaasti määrittää XML-dokumenttinsa rakenteen ja nimetä siinä olevat elementit ja attribuutit. Joissakin tapauk-

sissa on kuitenkin hyvä, jos dokumentin rakennetta pystytään tietyssä määrin luonnehtimaan. Rakenteeltaan samankaltaiset, mutta eivät välttämättä identtiset, XML-dokumentit muodostavat keskenään *dokumenttityypin* (document type). Samaan dokumenttityyppiin kuuluvien XML-dokumenttien rakenteelle pystytään antamaan rajoituksia liittämällä niihin dokumenttikielioppi.

Jos XML-dokumentti noudattaa jotakin dokumenttikielioppia, sen sanotaan olevan kyseisen kieliopin suhteen *validi*. Vastaavasti prosessia, jossa tutkitaan, noudattaako annettu XML-dokumentti annettua kielioppia, kutsutaan *validoinniksi* (validation). Validoinnille käänteinen prosessi on *kaavionpäättely* (schema extraction), jossa annetun XML-dokumentin perusteella yritetään johtaa sen rakenteen määrittelevä kielioppi. Kaavion päättely on usein ongelmallista, sillä dokumentissa oleva informaatio on useasti mahdollista kuvata usealla toisestaan poikkeavalla mutta kuitenkin samaa tarkoittavalla tavalla.

Yleisimmät XML-dokumenttien rakenteen määrittelemiseen käytettävät dokumenttikieliopit ovat dokumenttityypinmääritys eli DTD (*Document Type Definition*) ja XML Schema. Näitä käsitellään tarkemmin seuraavissa alaluvuissa.

3.3.1 DTD

DTD on XML:ään SGML-merkintäkielestä periytynyt piirre, ja XML:ssä käytössä oleva DTD-määrittely on SGML-merkintäkielen DTD:n rajoitettu osajoukko. Formaalisti luonnehdittuna DTD on *laajennettu kontekstiton kielioppi* [Via03]. Tässä yhteydessä "laajennettu" tarkoittaa sitä, että DTD-esityksessä sallitaan, että apumerkin oikealla puolella esiintyy jokin kolmesta säännöllisestä ilmauksesta, joiden avulla on mahdollista ilmaista apumerkkiä vastaavien XML-rakenteiden esiintymiskertojen lukumäärä. Apumerkkejä (nonterminal symbols) ovat DTD:ssä elementtien ja attribuuttien nimet. Ainoana perusmerkkinä (terminal symbol) DTD-kieliopissa on erityissymboli *#pcdata*, jolla merkitään rakenteetonta tekstimuotoista dataa.[1]

Olkoon Σ elementtien ja attribuuttien nimistä koostuva äärellinen aakkosto. DTD koostuu muotoa $e \rightarrow r$ olevista säännöistä, joissa $e \in \Sigma$ ja r on aakkoston Σ merkeistä niihin mahdollisesti liitettyjen esiintymiskertaoperaattoreiden avulla muodostettu säännöllinen ilmaus. DTD:ssä sallittavia esiintymiskertaoperaattoreita ovat kysymysmerkki (?), plusmerkki (+) ja asteriski ($*$). Ilmaus c? tarkoittaa, että apumerkki c voi esiintyä kerran tai ei lainkaan, ilmaus c+ sitä, että c voi esiintyä yhden tai useamman kerran, ja ilmaus $c*$, että c voi puuttua tai toistua useamman kerran. Lisäksi käytössä ovat pilkku (,), pystyviiva (|) ja sulkeet ((,)) mahdollistamassa aakkoston merkeistä muodostettujen ilmausten ryhmittelyn. Pilkulla toisistaan erotettujen ilmausten on esiinnyttävä dokumentissa peräkkäin. Pystyviivalla toisistaan erotetuista ilmauksista jonkin on esiinnyttävä dokumentissa. Yksi aakkoston Σ ilmauksista on dokumenttipuun juuri ja se toimii

[1] DTD:ssä on käytössä myös toinen symboli esittämään rakenteetonta tekstiä, nimittäin *#cdata*. Ero *#pcdata*- ja *#cdata*-määritelmien välillä on se, että XML-dokumenttia jäsennettäessä *#pcdata*-kohdat käydään läpi ja *#cdata*-kohdat sivuutetaan. Attribuutin arvoa merkitään DTD:ssä ainoastaan *#cdata*-määreellä. Jatkossa "rakenteettomalla tekstillä" tarkoitetaan *#pcdata*-muotoista dataa, ellei toisin ole erikseen ilmoitettu.

kieliopin aloitusmerkkinä, eli se esiintyy täsmälleen yhdessä säännössä ja ainoastaan kyseisen säännön vasempana puolena.

Tarkastellaan kuvan 3.1 esimerkkidokumentin formaalia kielioppiesitystä. Esimerkkidokumentin DTD on kielioppi $G = (\Sigma, T, N, S, P)$, missä $\Sigma = \{articles, article, no, author, id, initials, lastname, title, publication\_year, subtitle, \#pcdata\}$, $T = \{\#pcdata\}$ on perusmerkkien joukko, $N = \{articles, article, no, author, id, initials, lastname, title, publication\_year, subtitle\}$ apumerkkien joukko, $S = \{articles\}$ kieliopin aloitussymbolien joukko ja P käsittää seuraavat kymmenen muodostussääntöä:

Esimerkki 3.3:

(1)	$articles$	$\rightarrow$	$article*$
(2)	$article$	$\rightarrow$	$no, author, title, publication\_year$
(3)	$author$	$\rightarrow$	$id, ((initials, lastname) \mid \#pcdata)$
(4)	$title$	$\rightarrow$	$\#pcdata, subtitle?$
(5)	$initials$	$\rightarrow$	$\#pcdata$
(6)	$lastname$	$\rightarrow$	$\#pcdata$
(7)	$subtitle$	$\rightarrow$	$\#pcdata$
(8)	$year$	$\rightarrow$	$\#pcdata$
(9)	no	$\rightarrow$	$\#pcdata$
(10)	id	$\rightarrow$	$\#pcdata$

Säännössä (1) kerrotaan, että elementti *articles* ei koostuu yhdestäkään tai vaihtoehtoisesti koostuu useammasta *article*-elementistä. Sääntö (2) sanoo, että elementti *article* sisältää attribuutin *no* sekä elementit *author*, *title* ja *publication_year* tässä järjestyksessä.[2] Sääntö (3) sanoo, että elementti *author* koostuu attribuutista *id* sekä mahdollisesti elementeistä *initials* ja *lastname* tai, jos niitä ei ole, tekstidatasta. Sääntö (4) sanoo, että elementti *title* koostuu tekstidatasta ja mahdollisesti elementistä *subtitle*. Sääntöjen (5) - (10) oikealla puolella esiintyy kieliopin toinen perusmerkki *#pcdata* eli sääntöjen avulla määritellyt elementit ja attribuutit sisältävät tekstidataa.

On syytä huomata, että edellä annettu kielioppi on ainoastaan yksi mahdollisuus esimerkkidokumentin rakenteen ilmaisemiseksi, sillä jotkin kieliopin säännöt voidaan ilmaista myös toisin. Tarkastellaan esimerkiksi sääntöä (1). Nyt siinä on oletettu, että elementti *articles* ei sisällä joko yhtään tai sitten sisältää useita *article*-elementtejä, vaikka esimerkkidokumentissa *articles*-elementtiin sisältyy täsmälleen kaksi *article*-elementtiä. Kun elementtien esiintymiä rajoitetaan operaattoreilla $*$ ja $+$, kieliopeista tulee sallivampia kuin, jos elementin esiintymiskerrat ilmaistaisiin eksplisiittisesti. Edellä kuvatun kaltaisissa formaaleissa kieliopeissa (ja myös DTD:ssä) ainoa keino ilmaista, että elementillä on jokin täsmällinen määrä esiintymiä jonkin toisen elementin sisällä, on kirjoittaa toistettavan elementin nimi yläelementin määrittävän säännön oikealle puolelle niin monta kertaa, kuin elementin halutaan toistuvan. Esimerkkidokumentin *articles*-elementin rakennetta täsmällisesti vastaava sääntö olisi

[2] Toisin kuin DTD:ssä tässä kielioppiesityksessä ei ole tehty eksplisiittisesti eroa elementtien ja attribuuttien välillä. Lisäksi kielioppiesityksessä oletetaan, että attribuutit ovat elementtin sisällä määrätyssä järjestyksessä, mutta DTD ei ole attribuuttien järjestyksen suhteen normatiivinen.

Esimerkki 3.4:

$$articles \quad \rightarrow \quad article, article$$

Varsinainen XML:n DTD on yksinkertaistettu versio SGML:n DTD:stä ja se esitetään XML-dokumentissa SGML-syntaksin mukaisena. DTD voidaan liittää XML-dokumenttiin joko suoraan tai ulkoisesti, jolloin XML-dokumenttiin sisällytetään viittaus käytettävään DTD:hen. DTD:ssä XML-dokumentin elementit ja attribuutit esitellään erikseen. XML-dokumentin dokumenttielementti toimii DTD:ssä dokumentin tyypin määritteenä. Jos noudatetaan edellä annettua kielioppiesitystä, esimerkkidokumentin varsinainen DTD-esitys alkaa seuraavasti:

Esimerkki 3.5:

```
<!DOCTYPE articles [ ... ]>
```

Hakasulkeiden (`[,]`) sisällä kolmen pisteen paikalla esitellään XML-dokumenttiin sisältyvät elementti- ja attribuuttityypit. Elementtityypin esittely alkaa ilmauksella `<!ELEMENT`, jota seuraa tavallisten sulkeiden (`(,)`) välissä elementin sisällön määrittely. Elementtityypin esittelyn päättää suurempi kuin -merkki (`>`). Elementin sisältö määritellään edellisen formaalin kielioppiesityksen sääntöjen oikeiden puolien tapaan. Myös DTD:ssä on käytössä kielioppiesityksen yhteydessä mainitut elementtien esiintymiskertoja ilmaisevat operaattorit. Tyhjän elementin elementtityypin esittelyssä elementin sisällön määrittelyn tilalle kirjoitetaan sana `EMPTY`.

Attribuuttityypin esittely alkaa ilmauksella `<!ATTLIST`, jota seuraa sen elementin nimi, johon attribuutti sisältyy, ja sitten varsinaiset attribuuttimäärittelyt. Attribuuttityypin esittely päättyy suurempi kuin -merkkiin (`>`). Attribuuttimäärittelyssä annetaan attribuutin nimi ja tyyppi sekä mahdollisesti attribuutin arvoja koskevia tietoja. Tarkempien yksityiskohtien suhteen lukijaa kehotetaan tutustumaan XML-spesifikaatioon [BPS+04]. Oletetaan tässä, että esimerkkidokumentissa attribuuttien tyyppi on merkkijono (`CDATA`). Aiemmin esitettyä formaalia kielioppia vastaava esimerkkidokumentin DTD on annettu esimerkissä 3.6. Esimerkin DTD:n sääntöjen tulkinta on sama kuin formaalin kieliopin vastaavien sääntöjen tulkinta.

Esimerkki 3.6:

```
<!DOCTYPE articles [
 <!ELEMENT articles (article*)>
 <!ELEMENT article (author, title, publication_year)>
 <!ELEMENT author ((initials,lastname) | #PCDATA)>
 <!ELEMENT title (#PCDATA, subtitle?)>
 <!ELEMENT initials (#PCDATA)>
 <!ELEMENT lastname (#PCDATA)>
 <!ELEMENT subtitle (#PCDATA)>
 <!ELEMENT year (#PCDATA)>
 <!ATTLIST article no CDATA>
 <!ATTLIST author id CDATA>
]>
```

Vaikka DTD on yleisimmin käytetty XML-dokumenttien rakennetta kuvaava kielioppiesitys, on siinä kuitenkin useita puutteita. DTD:n avulla ei voida ilmaista luokkasuhteita (class relationship), kuten *is-a-*, *a-kind-of* ja *part-of*-suhteita, joilla on kuitenkin keskeinen rooli nykyaikaisessa ohjelmistotuotannossa [RDSH02]. DTD:ssä ei voida esittää eksplisiittisesti, että elementin jokin ominaisuus välittyy kaikille elementin alaelementeille, eikä erityisesti, että yläelementin jokin ominaisuus ei vaikuta johonkin tiettyyn alaelementtiin [RDSH02]. Jos DTD esimerkiksi sallii, että jossakin elementissä esiintyy attribuutti–arvopari `lang="eng"` (tulkinta: elementin sisältöosassa käytettävä kieli on englanti), tarkoittaa tämä automaattisesti myös sitä, että kaikissa kyseisen elementin alaelementeissä käytettävä kieli on myös englanti. DTD:ssä ei ole myöskään mahdollista määrittää, että elementin tai attribuutin sisältö on jotakin tiettyä tietotyyppiä (kokonaisluku, liukuluku, merkkijono, merkki jne.).[3] Tämä olisi kuitenkin monesti toivottavaa: esimerkiksi esimerkkidokumentin yhteydessä olisi hyvä, jos pystyttäisiin määrittämään, että `publication_year`-elementin sisällön on oltava kokonaisluku, mutta DTD:n avulla niin ei voida nyt tehdä. Ja kuten on jo aiemmin todettu, DTD ei tarjoa joustavaa keinoa ilmaista lukumäärärajoituksia. Jos esimerkiksi halutaan ilmaista DTD:n avulla, että elementti `article` voi esiintyä elementin `articles` sisällä vähintään 2 mutta enintään 15 kertaa, joudutaan elementin `article` nimi kirjoittamaan elementin `articles` elementtityypin esittelyyn yhteensä 119 kertaa.

3.3.2 XML Schema

XML Schema [FW04, TBMM04, BM04] on W3-konsortion kehittämä ehdotus XML-kieliopiksi, jossa on pyritty häivyttämään useimmat DTD:n tunnetut puutteet sekä lisäämään uusia piirteitä. XML Scheman kehittely aloitettiin konsortiossa vuonna 1998, ja nykyisin voimassa oleva suositus on vuodelta 2004. Toisin kuin DTD, jonka esitetään SGML-syntaksin mukaisesti, XML Schema käyttää XML-syntaksia ja siten jokai-

[3] Toisaalta kuitenkin `#PCDATA`- ja `#CDATA`-määritysten voidaan katsoa olevan tietotyyppejä. Näiden lisäksi DTD:ssä on kahdeksan muuta, attribuuttien yhteydessä käytettävää, tällaista ”tietotyyppiä”.

nen XML Schema -kuvaus on itsessään myös (hyvin-määritelty) XML-dokumentti (joka tästä johtuen voidaan puolestaan määritellä toisella XML Schema -kuvauksella tai antamalla sille DTD). Kokonaisuudessaan XML Schema on laaja ja monimutkainen standardi (XML Schema -suositus käsittää itse asiassa kolme erillistä dokumenttia), joten seuraavassa käydään läpi vain sen peruspiirteet. Tarkemmat yksityiskohdat ovat löydettävissä edellä mainituista W3-suosituksista.

Keskeinen rakennetekijä XML Schemassa on XML:n *nimiavaruuksien* (namespace) käyttö. XML-nimiavaruus on kokoelma URI-tunnisteen avulla identifioitavia nimiä, joita käytetään XML-dokumentin elementtien ja attribuuttien nimeämisessä. Nimiavaruuksien avulla elementtien ja attribuuttien nimissä käytettävä sanasto pystytään yksilöimään eri konteksteissa. Jos kahdella elementillä tai attribuutilla on sama nimi, ne voidaan yksiselitteisesti erottaa toisistaan kiinnittämällä ne omiin nimiavaruuksiinsa. XML-nimiavaruuksien syntaksi on määritelty W3-konsortion suosituksessa *Namespaces in XML* [BHL99]. Elementti tai attribuutti kiinnitetään tiettyyn nimiavaruuteen liittämällä sen nimeen etuliitteeksi kaksoispisteellä (`:`) varsinaisesta nimestä erotettuna käytettävän nimiavaruuden tunnus. Käytettävän nimiavaruuden tunnus on URI (*Universal Resource Identifier*) -tunniste, joka viittaa siihen lähteeseen, jossa käytettävä nimiavaruus on esitetty.

Oletetaan, että kuvan 3.1 esimerkkidokumentissa käytetty sanasto perustuu osoitteessa `http://www.DMPapers.org/namespaces/` määriteltyyn nimiavaruuteen, joka identifioidaan etuliitteellä '`dmp:`'. Tällöin dokumentissa käytettävä nimiavaruus määriteltäisiin kirjoittamalla elementin `articles` alkutunniste muotoon

Esimerkki 3.7:

```
<dmp:articles xmns:dmp="http://www.DMPapers.org/namespaces/">
```

ja liittämällä dokumentin kaikkien elementtien alku- ja lopputunnisteisiin sekä attribuuttien nimiin etuliite '`dmp:`'. Jos elementti on sidottu johonkin nimiavaruuteen, tulevat toisaalta sen kaikki alielementit ja attribuutit automaattisesti sidotuiksi samaan nimiavaruuteen, ellei toisin ole ilmoitettu.

XML Schema käyttää omaa nimiavaruuttaan, joka on määritelty osoitteessa `http://www.w3.org/2001/XMLSchema`. XML Schemassa käytettävä nimiavaruus ilmaistaan tavallisesti liittämällä XML Schema -komponenttien nimien eteen etuliite '`xsd:`'. Itse asiassa myös mikä tahansa muu etuliite on mahdollinen, kunhan se on jossakin yhteydessä kiinnitetty osoittamaan XML Scheman nimiavaruuteen.

XML Schemassa elementeillä ja attribuuteilla on paitsi nimi myös tyyppi. Attribuuttien tyyppi on aina jokin XML Schemalle määritellyistä yli 40 yksinkertaisesta tietotyyppistä, joihin kuuluvat mm. perinteiset `string` ja `integer`, tai vaihtoehtoisesti käyttäjän itsensä määrittelemä yksinkertainen tietotyyppi (ei käsitellä tässä). Elementti voi olla jotakin yksinkertaista tyyppiä tai – siinä tapauksessa, että siihen sisältyy attribuutteja ja alielementtejä – kompleksista tyyppiä.

XML:ssä tietotyyppien käyttö eroaa hieman tietotyyppien käytöstä tietojenkäsittelyn

muilla osa-alueilla. Perinteisesti arvo vastaa tietotyyppiä – kun on annettu arvo ja tietotyyppi, arvo joko on tai ei ole annettua tietotyyppiä. XML:ssä arvo validoituu tietotyypin suhteen – kun on annettu (ulkoinen) arvo ja tietotyyppi, validointiprosessi tuottaa (sisäisen) arvon tai epäonnistuu. Toisin sanoen (vasta) validointiprosessi liittää XML-dataan tietotyypin. [SW03]

XML Schema -kuvaus esitetään XML:n elementtirakenteita käyttäen. XML Schema -dokumenttiin kuuluu XML-esittely, ja dokumentin juurena on `schema`-niminen elementti. Juurielementissä ilmaistaan dokumentissa käytettävä nimiavaruus. XML Schema -dokumentin juurielementti kirjoitetaan seuraavasti:

Esimerkki 3.8:

```
<xsd:schema xmns:xsd="http://www.w3.org/2001/XMLSchema">
```

XML-dokumentin elementit esitellään XML Schema -kuvauksessa `element`-nimisissä elementeissä, joilla on vähintään omat attribuuttinsa `name` ja `type`, jotka viittaavat elementin nimeen ja tyyppiin. Jos kuvattava elementti on yksinkertaista tyyppiä, saa `type`-attribuutti arvokseen jonkin XML Scheman sisäänrakennetun perustietotyypin tai käyttäjän itsensä määrittelemän yksinkertaisen tietotyypin nimen. (Yksinkertaista tyyppiä olevaa elementtiä esiteltäessä XML Schemassa esittelyyn käytettävä elementti voi myös olla tyhjä.) Jos määriteltävä elementti on kompleksista tyyppiä, `type`-attribuutin arvona on sen kompleksisen tyypin nimi, jonka mukainen kuvattava elementti on.

Kompleksinen tyyppi määritellään `complexType`-elementin avulla. `complexType`-elementti sisältää attribuutin `name` ja elementin sisältöosassa esitellään ne elementit ja attribuutit, jotka sisältyvät kyseistä kompleksista tyyppiä edustavaan XML-elementtiin. Elementit kirjoitetaan `sequence`-nimisen elementin sisään siinä järjestyksessä, jossa niiden halutaan esiintyvän varsinaisen XML-elementin sisältöosassa. Jos halutaan, että elementit voivat esiintyä yläelementtinsä sisällä missä tahansa järjestyksessä, kirjoitetaan elementit XML Schema -elementin `all` sisään. Jos taas taas haluaan ilmaista, että ainoastaan jokin esiteltävistä alaelementeistä voi esiintyä (vrt. DTD:n operaattori "|"), kirjoitetaan elementtien esittelyt `choice`-nimisen elementin sisään. Kompleksisen tyypin sisällä esiteltävät elementit voivat edelleen yksinkertaista tai kompleksista tyyppiä. Elementin tyyppi ilmaistaan XML Schema -kuvauksessa siis joko attribuutilla `type` tai elementillä `complexType`, mutta samaan elementin esittelyyn ei voida sisällyttää kumpaakin tapaa.

Elementin mahdollisia esiintymiskertoja määriteltävässä XML-dokumentissa rajoitetaan XML Schemassa liittämällä elementin esittelyyn attribuutit `minOccur` ja `maxOccur`. Attribuutin `minOccur` avulla ilmoitetaan esiteltävän elementin esiintymiskertojen vähimmäismäärä ja se voi saada arvokseen nollan ja sitä suuremman kokonaisluvun. Attribuutti `maxOccur` ilmaisee elementin esiintymiskertojen enimmäismäärän ja voi saada arvokseen nollan ja sitä suuremman kokonaisluvun tai arvon '`unbounded`', jolla ilmaistaan, ettei elementin esiintymiskerroille XML-dokumentissa ole asetettu ylärajaa. Jos elementin esittelyyn ei ole liitetty esiintymiskerta-attribuutteja, vastaa se tilannetta, että attribuuttien `minOccur` ja `maxOccur` arvo on `1` eli elementti esiintyy täsmälleen kerran. Attribuutit

`minOccur` ja `maxOccur` voivat esiintyä ainoastaan, jos elementin esittely muodostaa sisäkkäisen rakenteen.

Attribuutit esitellään XML Schemassa käyttämällä `attribute`-elementtiä, jolla on jälleen omat attribuuttinsa `name` ja `type`. Näiden lisäksi XML Scheman `attribute`-elementtiin voidaan liittää attribuutit `use`, `fixed` ja `default`, joilla määritellään attribuutin esiintymisen tyyppi (attribuutti voi olla pakollinen, vapaaehtoinen tai kielletty) sekä attribuutin arvoihin ja oletusarvoihin liittyviä tietoja. Attribuutin tyyppi voi olla jokin XML Scheman sisään-rakennetuista tietotyypeistä tai käyttäjän itsensä määrittelemä yksinkertainen tyyppi.

XML Schemassa elementtien ja attribuuttien esittelyt voivat olla *paikallisia* (local) tai *globaaleja* (global). Paikallinen esittely tarkoittaa, että elementti tai attribuutti määritellään jonkin kompleksisen tyypin määrittelyn yhteydessä. Tällöin määriteltyjä elementtejä ja attribuutteja koskevat rajoitukset ovat voimassa ainoastaan niiden "yläelementin" vaikutusalassa. Globaalissa esittelyssä elementit tai attribuutit esitellään itsenäisesti suoraan `schema`-elementin alaisuudessa sitomatta niitä muihin kompleksisiin tyyppeihin. Globaalisti esittelyjä elementtejä ja attribuutteja voidaan sisällyttää muihin elementtien esittelyihin viittauksella. Globaalisti esiteltyjen elementtien ja attribuuttien nimet ovat yksilöiviä. (DTD:ssä kaikki elementtien esittelyt ovat globaaleja ja kaikki attribuuttien esittelyt paikallisia.) Paikalliset esittelyt mahdollistavat, että samassa XML-dokumentissa voi esiintyä eri konteksteissa samannimisiä elementtejä ja attribuutteja, jotka kuitenkin eroavat toisistaan jonkin ominaisuutensa suhteen.

Vaihtoehtoinen tapa määritellä XML Schemassa samannimisiä elementtejä, jotka kuitenkin eroavat keskenään joidenkin ominaisuuksien suhteen, on *johtaa* kompleksista tyyppiä olevasta elementistä uusia elementtejä. Uusia elementtejä voidaan johtaa *laajentamalla* (derivation by extension) tai *rajoittamalla* (derivation by restriction) aiemmin esiteltyjä kompleksisia elementtejä. Laajentamalla johtaminen tarkoittaa sitä, että aiemmin esiteltyyn kompleksiseen elementtiin lisätään uusia elementtejä tai attribuutteja, ja rajoittamalla johtaminen sitä, että aiemmin määritellystä kompleksisesta elementistä poistetaan elementtejä tai attribuutteja. Käytännössä laajentamalla johtaminen toteutetaan XML Schemassa määrittelemällä uusi `complexType`-elementti (jonka attribuutissa `name` kerrotaan kompleksisen tyypin nimi), jolla on alaelementti `complexContent` ja tällä edelleen alaelementti `extension`. Elementillä `extension` on attribuutti `base`, jonka arvo on laajennettavan kompleksisen tyypin nimi, ja elementin sisältöosassa esitellään kyseiseen kompleksiseen tyyppiin lisättävät elementit. Rajoittamisen kautta tehtävä uuden elementin johtaminen toteutetaan samalla tavoin kuin laajentamisen kautta johtaminen, mutta rajoitukset ilmaistaan nyt `restriction`-nimisen elementin sisällä. Käytännössä rajoittamalla johtamisessa joudutaan kirjoittamaan uudelleen kaikki rajoitettavaan kompleksiseen tyyppiin kuuluvat elementtien määrittelyt lukuunottamatta niitä, jotka rajataan pois. Uusien elementtien ja attribuuttien johtaminen tuo XML Schemaan *olio-orientoituneita* piirteitä (vrt. perintä).

XML Scheman merkittävimmät edut DTD:hen verrattuna ovat sen käyttämä notaatio ja parantunut tietotyyppien käyttö. Koska XML Schema -dokumentissa käytettävä syntaksi on sama kuin XML-merkintäkielen syntaksi, ei käyttäjän tarvitse opetella uutta

merkintätapaa määritellekseen XML-dokumentilleen kieliopin. XML Schema tarjoaa yli neljäkymmentä valmiiksi määriteltyä tietotyyppiä (DTD:ssä ”tietotyyppejä” on kymmenen) ja antaa käyttäjälle mahdollisuuden määritellä itse uusia tietotyyppejä. Mahdollisuus tietotyyppien ilmaisemiseen lisää merkittävästi dokumenttikieliopin ilmaisuvoimaa. XML Schemassa on myös mahdollista määritellä useita samannimisiä elementtejä ja attribuutteja, joilla on kuitenkin erilaisia ominaisuuksia. XML Schemassa on näiden lisäksi myös muita kehittyineitä piirteitä. Esimerkiksi elementin sisältö on mahdollista määritellä ainutkertaiseksi (keys on content) ja ainutkertaisuus on mahdollista määritellä koskemaan jotakin vaikutusaluetta (region). Samoin XML Schema tekee mahdolliseksi myös määritellä elementin, jolla ei ole sisältöä (ns. *nil*-sisältö, joka eroaa tyhjästä elementistä). XML Schemassa voidaan määritellä, että jokin elementti on *korvautuva* (substitutable) jollakin toisella elementillä. Lisäksi liittämällä elementit `any` ja `anyAttribute` kompleksisen tyypin määrittelyyn sallitaan se, että XML-dokumentissa esiintyy tietyssä kohdassa elementtejä ja attribuutteja, joita ei ole esitelty dokumentin XML Schema -kieliopissa. XML Scheman ilmeinen heikkous on se, että se on laaja ja monimutkainen standardi, jossa samoja asioita voi ilmaista monella vaihtoehtoisella tavalla. Lisäksi XML Schema -dokumentista itsestään tulee helposti hyvin laaja.

3.4 Semantiikka

XML-dokumentissa on mahdollista esittää semanttista informaatiota elementtien ja attribuuttien nimissä ja sijoittelemalla elementtejä sisäkkäisiksi rakenteiksi. Näin syntyvä semanttinen informaatio on kuitenkin ainoastaan ihmisten tulkittavissa. Koska dokumentin laatija voi vapaasti valita dokumentissaan käytettävän sanaston ja dokumentin rakenteen (so. sanaston merkityksen), on XML:ssä runsaasti tilaa merkityksen vaihtelulle. Esimerkiksi XML-dokumentin sisällä elementin tai attribuutin nimi ei välttämättä takaa sitä, että sen merkitys – tai edes tietotyyppi – olisi sama kuin saman dokumentin muilla samannimisillä elementeillä tai attribuuteilla. On myös syytä huomata, että XML-ilmauksella ei ole itsessään luontaista semantiikkaa, vaan sen semantiikka määritellään ainoastaan niiden toimenpiteiden kautta, joita yksi tai useampi tietokoneohjelma sille suorittaa (esimerkiksi tulkitsee sisäkkäiset elementit jonkin suhdetyypin ilmentymiksi) [DHB+00].

Myöskään DTD:n avulla ei ole mahdollista liittää XML-dokumenttiin semanttista informaatiota. DTD nimittäin esittää ainoastaan XML-dokumentissa käytettävän sanaston ja sen syntaktisen käytön (dokumentin rakenteen), mutta ei tarjoa sanastolle semantiikkaa [RDSH02]. XML Scheman avulla on mahdollista liittää XML-dokumenttiin hieman DTD:tä enemmän semanttista informaatiota (esim. tietotyyppien ja luokkasuhteiden esittämisen avulla), mutta senkään avulla ei ole mahdollista määrittää käytettävälle sanastolle täsmällistä semantiikkaa.

Eräs viime vuosina suosituksi tullut ratkaisu tietoverkoissa ja organisaatioiden sisällä lisääntyvän semanttisen anarkian ehkäisemiseksi on ollut *ontologioiden* laatiminen. Ontologia (ks. esim. [Gru95, GG95]) on tiettyyn kontekstiin sidottu käsitekaavio, jossa esitetään yksiselitteisellä tavalla kontekstissa validit käsitteet, niiden merkitys ja rakenne ja

käsitteiden väliset suhteet. Saman ontologian piirissä olevilla käyttäjillä on yhteinen sanasto ja jaettu tulkinta sen sanojen merkityksestä. Yleinen tapa esittää XML-muodossa ontologioiden kautta esitettyä informaatiota on RDF (*Resource Description Framework*). RDF:ää käsitellään tarkemmin seuraavassa luvussa.

3.4.1 RDF

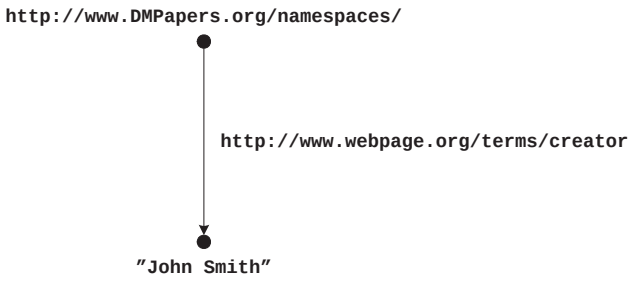
RDF on W3-konsortion suositus [MM04, KC04, Bec04, Hay04, BG04, GB04] kieleksi World Wide Webissä olevien tietoresurssien kuvaamiseen. RDF:n kehittämisen taustalla vaikuttaa voimakkaasti ajattelu *semanttisesta tietoverkosta* (Semantic Web) [BLHL01]. Tällä hetkellä Webin sisältö on suunniteltu pääasiassa ihmisten ymmärrettäväksi, ja tietokoneilla on varsin rajoitettu kyky käsitellä sen sisältämää informaatiota. Semanttinen tietoverkko laajentaa nykyistä tietoverkkoa lisäämällä siihen sellaisia ominaisuuksia, jotka mahdollistavat sen, että tietokoneohjelmat pystyvät käsittelemään tietoresurssien merkitystä.

Semanttisen tietoverkon pyrkimys on siis mahdollistaa tietokoneohjelmien yhteentoimivuus semanttisella tasolla Webissä. Semanttinen yhteentoimivuus ei vaadi standardeja ainoastaan tietoresurssien syntaktiselle muodolle vaan myös niiden semanttiselle sisällölle [DHB+00]. Tietoresursseihin luodaan semanttinen ulottuvuus liittämällä niihin asianmukaista metadataa, tietoa tiedosta. RDF on yksi malli metadatan määrittelemiseksi.

RDF perustuu siihen, että verkkoympäristössä resurssit voidaan identifioida ja esittää URI-tunnisteidensa kautta. Resurssilla tarkoitetaan mitä tahansa entiteettiä (oliota), joka voidaan identifioida URI-tunnisteella. URI-tunnisteella voidaan identifioida paitsi suoraan tietoverkossa olevia resursseja (verkkosivut ja -dokumentit) myös tietoverkon ulkopuolella olevia reaalimaailman entiteettejä (esim. ihmisiä tai kulutushyödykkeitä). RDF:ssä tietoresurssit kuvataan tunnistetuista entiteeteistä sekä niiden yksinkertaisista ominaisuuksista ja ominaisuuksien arvoista muodostettuina ilmaisuina (statement).

Loogisella tasolla RDF-ilmaisu on olio–ominaisuus–arvo-kolmikko: oliolla O on ominaisuus A, jolla on arvo V. RDF-ilmaisu muodostaa graafin, jossa oliot ja arvot ovat graafin solmuja ja ominaisuudet solmujen välisiä suunnattuja kaaria, jotka osoittavat aina oliosta arvoon. Esimerkki olio–ominaisuus–arvo-kolmikosta voidaan löytää luonnollisen kielen lauseesta "`http://www.DMPapers.org/namespaces/ has a creator whose value is John Smith`." [MM04]. Siinä `http://www.DMPapers.org/namespaces/` on olio (olion URI-tunniste), `creator` ominaisuus ja `John Smith` ominaisuuden arvo. Edellisestä esimerkistä on helppo havaita se ilmeinen analogia, joka vallitsee olio–ominaisuus–arvo-kolmikon ja luonnollisen kielen subjekti–predikaatti–objekti-suhteen välillä: olio voidaan samaistaa subjektiin, ominaisuus predikaattiin ja ominaisuuden arvo objektiin. RDF-ilmaisua kutsutaankin usein olio–ominaisuus–arvo-kolmikon sijasta subjekti–predikaatti–objekti-kolmikoksi.

Käytännössä sekä olio (entiteetti), ominaisuus että ominaisuuden arvo ilmaistaan RDF-esityksessä URI-tunnisteensa kautta. Ainoastaan ominaisuuden arvo voi olla merkkijonovakio, joka on muu kuin URI-tunniste. Jos oletetaan, että URI-tunniste `http://www.webpage.org/terms/creator` vastaa yllä olevassa esimerkissä käytettyä predikaattia `creator` ja että esimerkissä käytetyn ominaisuuden arvon annetaan olla



Kuva 3.3: RDF-graafi.

yksinkertainen literaali, saadaan näin kuvassa 3.3 esitetty RDF-graafi.

Olio–ominaisuus–arvo-kolmikossa olion ja arvon roolit voidaan vaihtaa: toisin sanoen mistä tahansa oliosta voidaan tehdä jonkin ominaisuuden arvo ja vastaavasti yksittäisestä ominaisuuden arvosta voidaan muodostaa uusi olio (edellyttäen, että kyseinen arvo on URI-tunnisteen avulla identifioitavissa). Graafiesityksessä tämä tarkoittaa, että RDF-ilmauksia on mahdollista ketjuttaa. Lisäksi mistä tahansa olio–ominaisuus–arvo-kolmikosta voidaan tehdä olio tai arvo johonkin toiseen olio–ominaisuus-arvo–kolmikkoon. Tällaista menettelyä kutsutaan *reifikaatioksi* (reification). Reifikaation seurauksena RDF-graafista tulee sisäkkäinen (nested) [DHB+00].

Verkkoympäristössä RDF-graafit esitetään tavallisesti XML-syntaksin mukaisesti, mutta RDF ei ole sidottu XML-esitystapaan. Tässä mielessä RDF ja XML ovat toisiaan täydentäviä [FCD02]. RDF muodostaa XML-merkintäkielen osajoukon, jota toisinaan kutsutaan myös RDF/XML:ksi, ja sen mukaisesti esitetty RDF-kuvaus on (hyvinmääritelty) XML-dokumentti. Kuvan 3.3 RDF-graafia vastaa seuraava XML-muotoinen esitys:

Esimerkki 3.9:

```
<?xml version="1.1" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:term="http://www.webpage.org/terms/"
 <rdf:Description about="http://www.DMPapers.org/namespaces/">
 <term:creator>John Smith</term:creator>
 </rdf:Description>
</rdf:RDF>
```

RDF-dokumentti alkaa XML-esittelyllä. RDF-dokumentti erotetaan muista XML-dokumenteista kiinnittämällä siinä käytettävä sanasto omaan nimiavaruuteen, joka ilmaistaan RDF-dokumentin komponenteissa etuliitteellä `rdf:`. RDF-dokumentin juurielementti on nimeltään `RDF` ja siinä esitellään (attribuutilla) RDF-nimiavaruus (`http://www.w3.org/1999/02/22-rdf-syntax-ns#`) sekä dokumentissa mahdollisesti käytettävät muut nimiavaruudet. Varsinainen RDF-ilmaus esitetään `Description`-nimisen elementin sisällä. Elementillä on attribuutti `about`, joka saa arvokseen määri-

teltävän entiteetin URI-tunnisteen. RDF-ilmauksessa käytettävä ominaisuus esitetään elementtinä ja ominaisuuden arvo tämän elementin sisältönä.

Edellisessä esimerkissä kuvan 3.3 RDF-graafin ominaisuus `http://www.webpage.org/terms/creator` on esitetty `creator`-nimisenä elementtinä, joka on kiinnitetty juurielementissä esiteltyyn nimiavaruuteen, mikä ilmaistaan elementin nimessä etuliitteellä `term`. Kun elementti on kiinnitetty nimiavaruuteen, ei ominaisuuden täydellistä URI-tunnistetta tarvitse kirjoittaa uudelleen elementin nimessä. Jos ominaisuuden arvo on URI-tunniste (oletetaan, että esimerkin literaalia "`John Smith`" vastaa URI-tunniste `http://www.webpage.org/staff/id12345`), kirjoitetaan ominaisuutta ja sen arvoa esittävä elementti muotoon

Esimerkki 3.10:

```
<rdf:Description>
 <term:creator rdf:resource="http://www.webpage.org/staff/id12345">
</rdf:Description>
```

RDF/XML-syntaksissa on käytössä runsaasti lyhennysmerkintöjä ja vaihtoehtoisia tapoja esittää samaa tarkoittavaa informaatiota. Edellä on käsitelty tiivistetysti RDF:n perussyntaksi. Tarkempien yksityiskohtien selvittämiseksi lukijaa kehotetaan tutustumaan aiemmin mainittuihin RDF-spesifikaatioihin.

RDF:ssä ominaisuudet kuvaavat resurssien välisiä suhteita. RDF ei kuitenkaan itsessään tarjoa mitään mekanismia näiden ominaisuuksien ja muiden resurssien välisten suhteiden kuvaamiseen. Tätä tehtävää varten RDF:ään on kehitetty sanastonkuvauskieli (vocabulary description language), RDF Schema. RDF Schema kuvaa luokkia ja ominaisuuksia, joita sitten voidaan käyttää uusien luokkien, ominaisuuksien ja muiden resurssien kuvaamiseen. RDF Schema on RDF:n semanttinen laajennos. Se tarjoaa mekanismin ryhmitellä keskenään yhteenkuuluvia resursseja ja kuvata resurssien välisiä suhteita. RDF Schemassa sanastonkuvaukset kirjoitetaan RDF-notaatiolla, jossa käytetään RDF Schema -spesifikaatiossa [Hay04] määriteltyä terminologiaa.

RDF Scheman tapa käsitellä luokkia ja ominaisuuksia muistutttaa monien olio-orientoituneiden ohjelmointikielten tapaa käsitellä tyyppejä (type systems). Monissa tällaisissa tyypityssysteemeissä tiettyyn luokkaan kuuluvilla olioilla on samat ominaisuudet. RDF Schema eroaa näistä lähestymistavoista kuitenkin siinä, että siinä ominaisuudet kuvataan luokittelemalla resurssit, joilla on tietyt ominaisuudet. Tätä tarkoitusta varten RDF:ssä ominaisuuksille voidaan määritellä *kohdealue* (domain) ja *vaikutusalue* (range). Esimerkiksi ominaisuudelle `eg:author` voitaisiin määritellä kohdealueeksi `eg:Document` ja vaikutusalueeksi `eg:Person`. Perinteisissä olio-orientoituneissa lähestymistavoissa määriteltäisiin vastaavasti luokka `eg:Book`, johon kuuluu attribuutti `eg:author`, joka on puolestaan tyyppiä `eg:Person`. RDF-lähestymistavassa kuka tahansa voi määritellä kohde- tai vaikutusalueeseen lisäominaisuuksia ilman, että luokkien alkuperäinen kuvaus jouduttaisiin määrittelemään uudelleen. RDF:n ominaisuuskeskeisen lähestymistavan yksi hyvä piirre onkin, että kuka tahansa voi laajentaa olemassa olevien resurssien kuvauksia. [Hay04]

3.5 XML-kyselykielet

XML-dokumenteissa olevan informaatio etsimiseen ja käsittelyyn tarvitaan kyselykieliä. Seuraavissa alaluvuissa esitellään kolme W3-konsortiossa kehitettyä XML-kyselykieltä, XPath, XSLT ja XQuery.

3.5.1 XPath

XPath [CD99] on XML-dokumenteissa olevan informaation paikantamiseen tarkoitettu kieli. Se perustuu XML-dokumentin elementtirakenteissa kulkemiseen ja dokumentin komponenttien poimimiseen. XPath jäsentää XML-dokumentin solmuista koostuvana puuna. Jokainen dokumenttipuun solmu on yhtä solmutyyppiä, joita XPathissa on määritelty seitsemän: juuri-, elementti-, attribuutti-, teksti-, kommentti-, nimiavaruus- ja prosessointiohjesolmu. XML-dokumentissa on ainoastaan yksi juurisolmu. Juurisolmu ei vastaa täysin XML-dokumentin juurielementtiä, sillä XPathin juurisolmu sisältää paitsi koko juurielementin myös sitä ennen ja sen jälkeen tulevat kommentit ja prosessointiohjeet. XPath ei kuitenkaan pysty käsittelemään dokumentin XML-esittelyä eikä dokumenttiin mahdollisesti liittyvää DTD:tä. XPath-ilmaukset evaluoidaan aina suhteessa käsiteltävään solmuun, jota kutsutaan *kontekstisolmuksi* (context node). XPath-kyselyn suorittamisen seurauksena kontekstisolmu tavallisesti vaihtuu.

XML-dokumentin osiin viittaamiseen XPathissa käytetään *polkuilmauksia* (path expressions). Polkuilmaus ilmaisee paikannuspolun (dokumenttipuun alipuun), joka kontekstisolmusta on kuljettava, jotta tavoitetaan haluttu solmu. Yhtä paikannuspolkua kohden dokumenttipuussa on tavallisesti useampi alipuu. Siten polkuilmauksen tuloksena saadaan yleensä useita dokumentin solmuja. Syntaktisesti paikannuspolku koostuu symbolilla "/" toisistaan erotetuista *paikannusaskelista* (location step). Paikannusaskel koostuu puolestaan kolmesta komponentista: akselista, solmutestistä ja predikaatista.

Akseli (axis) kuvaa kontekstisolmun ja valittavien solmujen välisen suhteen puurakenteessa (alipuiden joukon). XPath-spesifikaatiossa [CD99] määritellään kaikkiaan 13 erilaista solmujen välistä suhdetta. Seuraavaksi esitellään niistä tavallisimmat. Dokumentin juurisolmua lukuunottamatta jokaisella elementti- ja attribuuttisolmulla on yksi *vanhempisolmu* (parent). Elementtisolmulla voi olla *lapsisolmuja* (child). Solmuja, joilla on yhteinen vanhempisolmu, kutsutaan *sisarussolmuiksi* (sibling). Tietyn solmun edeltäjäsolmuja (ancestor) ovat solmuun transitiivisessa suhteessa olevat, dokumenttihierarkian ylemmillä tasoilla sijaitsevat solmut. Vastaavasti dokumenttihierarkiassa alemmilla tasoilla sijaitsevat, tiettyyn solmuun transitiivisessa suhteessa olevat solmut ovat kyseisen solmun *seuraajasolmuja* (descendant). Akseleihin viitataan niiden englanninkielisillä nimillä.

Joillekin akseleille on kuitenkin määritelty myös lyhennysmerkintöjä, ja varsinaisessa XPath-ilmauksessa akseleista tavallisesti käytetään niitä vastaavia lyhennysmerkintöjä. Esimerkiksi symbolilla "." viitataan kontekstisolmuun. Symbolilla ".." viitataan kontekstisolmun vanhempisolmuun. Dokumenttipuun juurisolmua merkitään symbolilla "/". Merkinnällä "//" viitataan dokumenttipuun mielivaltaisella syvyystasolla olevaan elementtiin. Aina kun polku alkaa symbolilla "/" on kyseessä absoluuttinen polku, jonka

evaluointi aloitetaan dokumentin juuresta. Jos taas polku alkaa solmun nimellä on polku suhteellinen ja sen evaluointi aloitetaan kulloisestakin kontekstisolmusta.

Solmutesti (node test) on paikannusaskeleen komponenteista ainoa pakollinen. Solmutestin avulla paikannusaskeleeseen sisältyvät solmut yksilöidään. Tyypillisimmät solmutestit kohdistuvat solmun nimeen ja tyyppiin. Solmun nimitestin (node name test) avulla voidaan eksplisiittisesti nimetä solmut, jotka halutaan valita. Esimerkiksi kirjoittamalla "`child`" valitaan kaikki kontekstisolmuun sisältyvät `child`-nimiset solmut. Korvaamalla solmun nimi symbolilla "`*`" (wildcard) voidaan viitata mihin tahansa elementtiin. Aina ei kuitenkaan voida taata, että ainoastaan nimeämällä solmu saadaan haluttu solmu, sillä samassa dokumenttipuussa voi olla useita eri tyyppiä olevia solmuja, joilla on kuitenkin sama nimi. Nimitestillä saadaan valittua ainoastaan solmut, joiden nimi vastaa annettua solmun nimeä ja jonka tyyppi on sama kuin akselin solmujen oletustyyppi, joka on elementtisolmu. Jos halutaan valita solmuja, jotka ovat muuta tyyppiä kuin elementtisolmuja, on käytettävä solmun tyyppitestiä (node type test). Tyyppitestin avulla siis ohitetaan akselin solmuille määritelty oletustyyppi. Tyyppitestin avulla voidaan valita erikseen kommentti-, prosessointiohje- tai tekstisolmu sekä solmu, jolle ei haluta määritellä tyyppiä.

Predikaattien avulla voidaan valita dokumenttipuusta solmuja, jotka täyttävät määrätyt ehdot. XPathin predikaatit ovat analogisia SQL-kyselykielen WHERE-lauseessa käytettäville suodatusehdoille. Predikaatit kirjoitetaan paikannusaskeleessa välittömästi solmutestien jälkeen hakasulkeiden (`[,]`) sisään. Predikaateissa käytettävät suodatusehdot voivat sisältää mm. Boolen lausekkeita sekä vertailu- ja joukko-operaatioita. Predikaattien avulla on myös mahdollista indeksoida elementtisolmun alaelementtejä. Esimerkiksi ilmaus "`child[1]`" palauttaa kontekstisolmun (dokumenttijärjestyksessä) ensimmäisen alaelementin.

Kontekstielementin attribuuttien hakemiseen on määritelty erikseen oma akselinsa, joka on nimeltään `attribute`. Symboli "`@`" on attribuuttiakselin lyhennysmerkintä ja (ilman siihen liitettyä solmutestiä) sillä viitataan kontekstisolmun mihin tahansa attribuuttisolmuun. Kun XPathissa halutaan viitata nimettyyn attribuuttiin liitetään "`@`"-symboli halutun attribuutin nimen eteen. Kuitenkaan kaikkia elementeille sovellettavissa olevia predikaatteja ei voida soveltaa attribuuteille. Esimerkiksi elementtisolmun attribuutteja ei ole mahdollista indeksoida, sillä attribuuttien järjestystä ei ole määritelty.

3.5.2 XSLT

XSLT (*Extensible Stylesheet Language Transformations*) [Cla99] on XML-dokumenttien muunnoksiin erikoistunut kieli. Se on XML-dokumenttien ulkoisen esitystavan määrittelemiseen käytettävän XSL (*Extensible Stylesheet Language*) -kielen toinen komponentti. XSL on verrattavissa HTML:ssä käytettävään CSS (*Cascading Stylesheet*) -tyylimäärityskieleen. Sen avulla XML-dokumentin laatija pystyy määrittelemään, miten XML-dokumentin sisäinen esitystapa (elementti- ja attribuuttirakenteet) esitetään loppukäyttäjän kannalta tarkoituksenmukaisimmassa muodossa. XSL-kielen toisen komponentin, XSL-FO (*Extensible Stylesheet Language - Formating Objects*) [ABC+01] avulla voidaan määritellä yksityiskohtaisesti XML-dokumentin ulkoasun tulostus (esimerkik-

si kirjasinten koko ja väri). XSLT puolestaan tarjoaa keinon, jolla rakenteeltaan erilaisten XML-dokumenttien sisäinen esitysmuoto voidaan saattaa keskenään yhdenmukaiseksi – itse käsiteltäviä dokumentteja kuitenkaan muuttamatta. XSLT:n avulla voidaan esimerkiksi mikä tahansa (hyvin-määritelty) XML-dokumentti muuntaa HTML-muotoon, jolloin sitä voidaan lukea HTML-selaimen avulla. XSLT:ssä noudatetaan XML-syntaksia, ja siten jokainen XSLT-dokumentti on itsessään (hyvin-muodostettu) XML-dokumentti. XSLT-dokumentit käyttävät omaa nimiavaruuttaan, joka on määritelty osoitteessa `http://www.w3.org/1999/XSL/Transform`.

XSLT-muunnosprosessissa XML-muotoinen *lähtödokumentti* (source tree) muunnetaan *tulosdokumentiksi* (result tree). Käytännössä muuntaminen tarkoittaa esimerkiksi sitä, että lähtödokumenttiin lisätään tai siitä poistetaan elementtejä tai lähtödokumentin elementtejä lajitellaan tai järjestetään muuten uudelleen. XSLT-määritys koostuu joukosta muunnossääntöjä, joita kutsutaan *malleiksi* (template). Jokainen malli sisältää sääntöjä määrittelemään ne lähtödokumentin osat, joiden tulisi *täsmäytyä* (match) yhteen tai useampaan ennalta määriteltyyn *hahmoon* (pattern). Kun täsmäytyminen tapahtuu, muuntaa XSLT lähtödokumentin täsmäytyneen osan tulosdokumentin osaksi. Tulosdokumentti voi olla lähtödokumentin variantti tai täysin uusi XML-dokumentti. XSLT-muunnosprosessi ei muuta lähtödokumenttia fyysisesti.

XSLT-dokumentin juurena on joko `<xsl:stylesheet>`- tai `<xsi:transform>`-niminen elementti. Elementit ovat toistensa täydellisiä synonyymeja ja kumpaa tahansa voidaan käyttää. Muunnossäännöt esitetään `<xsl:template>`-elementtien sisältöosassa, ja niillä kullakin on `match`-niminen attribuutti, jota käytetään sen ilmaisemiseen, mitä XML-dokumentin osaa tai osia muunnos koskee. Dokumenttipuun osien osoittamiseen käytetään XSLT:ssä XPath-polkuilmauksia. Asettamalla `match`-attribuutin arvoksi "`/`" ilmaistaan, että muunnossäännöt koskevat koko käsiteltävää XML-dokumenttia.

Muunnossääntöjä varten XSLT käyttää ohjausrakenteita, jotka vastaavat konventionaalisten proseduraalisten ohjelmointikielten ohjausrakenteita. Esimerkiksi elementillä `<xsl:value-of>` saadaan tulosdokumenttiin käsiteltävän solmun arvo. Elementillä on attribuutti `select`, jonka arvoksi annetaan valittavan solmun identifioiva polkuilmaus. Jos tulosdokumenttiin halutaan lisätä useita elementtejä tai niiden arvoja, käytetään `<xsl:for-each>`-elementtiä. Sillä on attribuutti `select`, jonka arvoksi asetetaan ehto, jonka perusteella elementit tai niiden arvot tulosdokumenttiin valitaan. Elementtiä `<xsl:sort>` käytetään tuloksen lajittelemiseen. Sen `select`-attribuutin arvo osoittaa, mitkä elementit lajitellaan. Elementti `<xsl:if>` sisältää mallin, jota sovelletaan tulosdokumentin konstruointiin, jos sen `test`-attribuutissa esitetty ehto toteutuu. Elementtiä `<xsl:choose>` käytetään yhdessä `<xsl:when>`- ja `<xsl:otherwise>`-elementtien kanssa esittämään moniosaista ehtoa. XSLT-elementtien ehto-osissa voidaan käyttää puhtaiden polkuilmausten lisäksi XSLT:lle erikseen määriteltyjä funktioita sekä kaikkia XPathin funktioita.

XSLT-tyylitiedosto on itse asiassa sinällään pieni tietokoneohjelma ja XSLT on itsessään ohjelmointikieli. XSLT-tyylitiedosto sisältää toisin sanoen proseduurin, joka suorittamalla XML-dokumentti on muunnettavissa toiseksi XML-dokumentiksi. XSLT-tyylitiedostossa esitetty muunnos toimii ohjeena XSL-muuntimen (XSL processor) toi-

minnalle. XSL-muunnin kuvaa yhden tai useamman XML-dokumentin tasan yhdeksi XML-dokumentiksi. Ohjelmointikielenä XSLT noudattaa LISP-peräistä ajattelutapaa, jossa ohjelmaa itseään voidaan pitää sen alla olevan kielen tietomalliin kuuluvana yksikkönä. Toisin sanoen jokaista XSLT-ohjelmaa voidaan käsitellä edelleen toisella XSLT-ohjelmalla.

Vaikka XSLT on siis alunperin suunniteltu mahdollistamaan XML-dokumenttien muuntaminen yhdestä esitystavasta toiseen, XSLT voidaan tästä huolimatta nähdä myös puhtaasti kyselykielenä. Esimerkiksi tarkasteltaessa XSLT-ohjelmaa, jolla valitaan lähtödokumentista tulosdokumenttiin tietyt ehdot täyttävät elementit, voidaan havaita ilmeinen analogia SQL-kyselyn kanssa, jolla haetaan tietokannasta tietyt ehdot täyttävät tietoalkiot. Erona SQL:ään luonnollisesti on se, että XSLT:ssä käyttäjän pitää proseduraalisesti määrittää, miten halutut rakenteet XML-dokumentista poimitaan, kun SQL:ssä riittää, että käyttäjä osoittaa, mitkä (minkätyyppiset) tietoalkiot poimitaan.

Vaikka XSLT:llä voidaan deklaratiivisesti osoittaa, millainen XML-dokumenttien välinen muunnos on, on itse muunnos kuitenkin toteutettava perinteisen proseduraalisen ohjelmoinnin keinoin. Vakava puute XSLT-kielen avulla tapahtuvassa XML-dokumenttien esitystavan muuntamisessa on myös se, että XSLT-ohjelmoijan on tiedettävä syvällisesti kahden tai useamman esitystavan välillä vallitsevat vastaavuudet. XSLT on monimutkainen kieli ja pienenkin muunnoksen tehdäkseen käyttäjän on kirjoitettava XSLT-ohjelma, mikä vaatii XSLT-kielen täydellistä hallintaa ja siis näin ollen ohjelmointitaitoa. Lisäksi jokaista rakenteeltaan erilaisen XML-dokumentin välistä muunnosta varten loppukäyttäjän pitää kirjoittaa uusi XSLT-ohjelma.

3.5.3 XQuery

XQuery [BCF+03] on ensimmäinen W3-konsortiossa nimenomaisesti XML-muotoisen tiedon käsittelyyn kehitetty kieli. XPath tarjoaa mekanismin XML-rakenteissa navigoimiseen, mutta ei keinoa itse rakenteiden poimimiseen, ts. varsinaista kyselykieltä. Lisäksi XPath-ilmauksia voidaan käyttää ainoastaan viittaamaan jo olemassa oleviin rakenteisiin, mutta niiden avulla ei ole mahdollista muodostaa uusia rakenteita. XSLT:n avulla on mahdollista luoda uusia XML-rakenteita, mutta siinä varsinaiset kyselyominaisuudet ovat kuitenkin vain toissijainen piirre. XQueryssä on puolestaan sekä perinteisten kyselykielten että varsinaisten ohjelmointikielten piirteitä.

XQueryn kehittämisen taustalla ovat useista aikaisemmista XML- ja puolirakenteisen tiedon käsittelyyn tarkoitetuista kyselykielistä (XQL [RLS98], XML-QL [DFF+99], Quilt [CRF00], Lorel [AQM+97] ja YATL [CDSS98]) saadut kokemukset. Lisäksi XQuery on ottanut vaikutteita myös SQL- ja OQL(*Object Query Language*) -kyselykielistä [CA96]. XQuery on myös yhteensopiva useiden W3-konsortion standardien kanssa (XPath, XML Namespaces, XML Schema ja XSLT). XQuery on kuitenkin kehittyvä kieli eikä se ole saanut vielä (heinäkuu 2005) W3-konsortion suosituksen asemaa, vaan on ainoastaan työluonnos (working draft). Tästä huolimatta XQuery-toiminnallisuus on jo nyt toteutettu yleisimmissä tietokantajärjestelmissä (Oracle, IBM ja Microsoft).

XQuery on funktionaalinen kieli, mikä tarkoittaa sitä, että XQuery koostuu *ilmauksista*, jotka palauttavat *arvoja*, jotka ovat tiettyä *tietotyyppiä*. XQuery-ilmauksia ovat

mm. funktiot, polkuilmaukset, FLWOR-ilmaukset ja konstruktorit. XQueryssä käytetään *funktioita* XML-dokumenteissa olevien tietojen poimimiseen. Funktiokutsu koostuu funktion nimestä, jota seuraavat välittömästi, sulkujen (`(,)`) väliin kirjoitettuina ja pilkulla toisistaan erotettuina funktion mahdolliset argumentit. Esimerkiksi kutsumalla funktiota `doc("example.xml")`, saadaan avattua XML-dokumentti, jonka nimi on `example.xml`. XQueryssä on itsessään laaja valmiiksi määriteltyjen funktioiden kirjasto, mutta käyttäjän on sen lisäksi mahdollista myös määritellä omia funktioitaan. Koska käyttäjän määrittelemät funktiot voivat olla rekursiivisia ja keskenään rekursiivisia (mutually recursive), on XQueryllä Turingin koneen ilmaisuvoima [FS03].

XML-dokumentin osien osoittamiseen käytetään XQueryssä XPath-polkuilmauksia. Itse asiassa XQuery 1.0 ja XPath 2.0 jakavat saman tietomallin ([FMM+05]), ja XPath sisältyy osajoukkona XQueryyn. Vaikka *polkuilmaukset* ovat sinällään ilmaisuvoimaisia, ne sisältävät yhden tärkeän rajoituksen: polkuilmaukset voivat kohdistua vain jo olemassa oleviin solmuihin. *FLOWR-ilmaukset* tarjoavat pelkkiä polkuilmauksia suuremmat mahdollisuudet XML-muotoisen informaation käsittelyyn. Niiden avulla on mm. mahdollista järjestää kyselyn tulosjoukkoa ja luoda uusia XML-rakenteita. Akronyymi FLWOR (äännetään kuten sana "flower") tulee ilmauksen komponenttilauseiden nimien alkukirjaimista. FLWOR-ilmaukseen on mahdollista sisällyttää viidentyyppisiä komponenttilauseita:

- `for`-lause, jossa suoritetaan iteraatio ja kiinnitetään annetut muuttujat kullakin iteraatiokierroksella lauseen ehto-osan (tyypillisesti polkuilmaus) palauttamaan tulokseen,
- `let`-lause, jossa annetut muuttujat kiinnitetään lauseen ehto-osaa vastaaviin XML-dokumentin arvoihin, mutta ilman iteraatiota,
- `where`-lause, jossa määritellään kyselyn tulosjoukkoon kuuluvien arvojen valintakriteerit,
- `order by` -lause, jolla määritetään tulosjoukolle lajittelujärjestys,
- `return`-lause, jossa lopuksi määritellään, mitä kysely palauttaa.

`Return`-lause on FLWOR-ilmauksen ainoa pakollinen komponentti. Muuttujat esitetään XQueryssä liittämällä symboli `$` halutun muuttujanimen eteen. Seuraava esimerkki valaisee `for`- ja `let`-lauseiden välistä eroa:

Esimerkki 3.11:

(A)

```
for $i in 1 to 5
return <no> $i </no>
```

(B)

```
let $i in 1 to 5
return <no> $i </no>
```

Kysely, jossa on käytetty `for`-lausetta, palauttaa viisi `no`-nimistä elementtiä, joilla on sisältönään numerot yhdestä viiteen siten, että kukin erillinen numero on aina yhden `no`-elementin sisältönä. Kysely, jossa käytettiin `let`-lausetta, palauttaa puolestaan elementin

<no> 1 2 3 4 5 </no>. Tarkastellaan vielä esimerkkiä FLWOR-ilmauksesta, jossa on käytetty kaikkia komponenttilauseita:

Esimerkki 3.12:

```
for       $d in doc("departments.xml")//name
let       $e in doc("employees.xml")//employee
where     $e/department_name = $d
order by  $d descending
return    <departments>
           <department_name> $d </department_name>
           <no_of_emps> $e count </no_of_emps>
          </departments>
```

Esimerkin 3.12 kysely palauttaa siis XML-elementin, joka sisältää nousevaan järjestykseen lajiteltuna listan dokumentissa `dept.xml` mainittujen osastojen nimistä ja dokumentista `emp.xml` saatujen tietojen perusteella lasketuista osastokohtaisista työtekijämääristä. Esimerkin 3.12 kysely evaluoidaan seuraavasti: `For`-lauseessa käydään dokumenttijärjestyksessä läpi dokumenttia `dept.xml` ja kiinnitetään muuttuja `$d` dokumentista vuorollaan löytyviin `name`-elementtien arvoihin. `Let`-lauseessa haetaan dokumentin `emp.xml` kaikkien `emp`-nimisten elementtien arvot ja liitetään saatu arvomonikko `for`-lauseen palauttamaan tulokseen. `Where`-lauseessa `for-` ja `let`-lauseiden yhteistuloksesta poistetaan sellaiset muuttujan `$e` kiinnitykset, joissa `deptname`-elementin arvo ei vastaa muuttujaan `$d` kulloinkin kiinnitettyä arvoa. `order by` -lauseessa (`for-`, `let-` ja `where`-lauseissa muodostettu) välitulos järjestetään muuttujan `$d` kiinnitysten mukaisesti nousevaan järjestykseen. Kun dokumentti `dept.xml` on käyty `for`-lauseessa kokonaan läpi, muodostetaan `return`-lauseessa tulosdokumentti korvaamalla muuttujan `$d` esiintymät dokumentista `dept.xml` saaduilla todellisilla arvoilla ja laskemalla kuhunkin osaston nimeen liittyjien `emp`-elementtien määrä muuttujan `$e` erilaisten kiinnitysten perusteella. Kuten esimerkistä käy ilmi, voidaan FLWOR-ilmauksen sisältävässä XQuery-kyselyssä käyttää samanaikaisesti useata XML-dokumenttia, mikä ei ole mahdollista pelkkiä polkuilmauksia käytettäessä.

XQueryssä käytetään *konstruktoreja* uuden informaation (so. uusien XML-rakenteiden) luomiseen. Konstruktoreiden avulla voidaan tuottaa kokonaisia XML-dokumentteja tai yksittäisiä elementtejä, attribuutteja, tekstijaksoja, kommentteja ja prosessointiohjeita. Yleensä konstruktorilla on sama nimi kuin sitä vastaavalla XML-rakenteella. XQueryssä on suoria ja laskettuja konstruktoreja. Suorat konstruktorit (direct constructors) noudattavat suoraan XML-notaatiota ja muistuttavat XML-dokumenttia tai sen osaa. Esimerkeissä 3.11 ja 3.12 on jo itse asiassa käytetty suoria konstruktoreita kyselyiden tuloksen muodostamisessa. Lasketuissa konstruktoreissa (computed constructors) kerrotaan eksplisiittisesti, millainen XML-rakenne halutaan luoda ja ilmaistaan aaltosulkujen ("{","}") välissä millainen arvo tai sisältö ko. rakenteelle halutaan antaa. Esimerkissä 3.13 on annettu esimerkki lasketun konstruktorin käytöstä (A) ja sen kautta luotavasta XML-rakenteesta(B). (Esimerkissä luotava XML-rakenne on myös it-

sessään suora konstruktori).

Esimerkki 3.13:

(A)

```
element book {
 attribute isbn { "0-262-19338-9" },
 element title { "Art of Prolog, 2nd Ed." },
 element authors {
 element author { "Leon Sterling" },
 element author { "Ehud Shapiro" }
 }
}
```

(B)

```
<book isbn="0-262-19338-9">
 <title> Art of Prolog, 2nd Ed. </title>
 <authors>
 <author> Leon Sterling </author>
 <author> Ehud Shapiro </author>
 </authors>
</book>
```

Kuten edellä on nähty, voidaan konstruktoreja sisällyttää FLWOR-ilmauksiin. Attribuuttien ja tekstijaksojen tuottamiseen voidaan XQueryssä käyttää ainoastaan laskettuja konstruktoreja.

XQuery-kyselyssä on mahdollista käyttää myös tavanomaisia ehdollisia ilmauksia (perinteinen `if–else–then`-rakenne, jossa `else`-lause on pakollinen), sekä artimeettisia, loogisia ja vertailuilmauksia. Lisäksi XQuery tukee kvantifioituja ilmauksia eli ilmauksia, jotka sisältävät eksistentiaali- tai universaalikvantifioinnin. XQueryssä avainsana `some` vastaa eksistentiaalikvanttoria. Ilmaus, joka sisältää avainsanan `some`, palauttaa totuusarvon tosi, jos XML-dokumentin (dokumenttien) jokin tietyntyyppinen osa täyttää ilmauksessa annetun ehdon. Universaalikvanttoria vastaa puolestaan avainsana `every`, ja universaalikvanttorin sisältävä XQuery-ilmaus palauttaa totuusarvon tosi, jos kaikki käsiteltävän XML-dokumentin (dokumenttien) tietyntyyppiset osat täyttävät annetun ehdon. Kvantifioituihin ilmauksiin sisältyy FLOWR-ilmausten tavoin iteraatio.

XQuery on tyypitetty kieli. Toisin sanoen XQuery-ilmausten palauttamat arvot (solmu tai atominen arvo) on aina jotakin tietotyyppiä. XQueryn tietomalliin sisältyy itsessään kielensisäisen tyyppijärjestelmä (XPath 2.0:n tietotyypit), mutta XQueryllä on kuitenkin myös käytössään laajempi kielenulkoinen tyyppijärjestelmä, nimittäin XML Scheman tietotyypit. XQueryssä käytettävä tietotyyppi voi olla (sen lisäksi, että se voi olla alatyyppi tai johdettu tietotyyppi) heikko tai vahva. Heikko tietotyyppi tarkoittaa

sitä, että käsiteltävä arvo pakottautuu implisiittisesti operaation edellyttämäksi (esimerkiksi operaatio "+" edellyttää, että molemmat sen operandit ovat numeerisia arvoja) tietotyypiksi, jos se ei ole jo sitä (XPathin tietotyyppi `xdt:untypedAtomic` on esimerkki heikosta tietotyypistä: numeerisen arvon 1 lisääminen tyypitöntä atomista tietotyyppiä edustavaan arvoon 41 tuottaa tulokseksi arvon 42). Vahva tietotyyppi puolestaan aiheuttaa virheen (type error), jos käsiteltävä arvo ei ole operaation edellyttämää tyyppiä tai operaation edellyttämän tyypin alatyyppi (esimerkki vahvasta tietotyypistä on `xs:string`: numeerisen arvon 1 lisääminen merkkijonoon 41 aiheuttaa virheen).

XQueryssä on käytössä sekä dynaaminen että staattinen tyypitys. Dynaaminen tyypitys tarkoittaa sitä, että jos tietotyyppi-informaatiota on kyselyn analysointivaiheessa saatavilla ainoastaan niukasti, tarkistaa XQuery kyselyn suorituksen aikana dynaamisesti, vastaako kulloinkin käsiteltävä ilmaus tai arvo siltä vaadittavaa (esimerkiksi operaation ilmaukselta tai arvolta edellyttämää) tietotyyppiä. XQuery suorittaa staattisen tietotyyppien tarkistuksen, jos tyyppi-informaatiota (esimerkiksi dokumenttiin liittyvä XML Schema) on jo kyselyn analysointivaiheessa riittävästi saatavilla. Staattista tietotyyppien tarkistusta on kahdenlaista. Staattisessa konservatiivisessa (conservative) tyypityksessä nostetaan käännöksen aikana virhe, jos johdettu tyyppi ei ole vaaditun tyypin alatyyppi. Staattissa osittaisessa (partial) tyypityksessä virhe nostetaan käännöksen aikana ainoastaan siinä tapauksessa, että ei ole mahdollista, että virhettä ei nostettaisi myös ajon aikana.

XQuery sisältää myös ilmauksia tietotyyppien käsittelyyn. Ilmausten avulla on mahdollista

- määritellä, että muuttuja on jotakin tiettyä tietotyyppiä (`as Type`),
- tarkistaa, onko käsiteltävä arvo tiettyä tietotyyppiä (`instance of` ja `typeswitch`; lisäksi on funktio `castable`, jonka avulla tarkastetaan, voidaanko käsiteltävän atomisen arvon tietotyyppi vaihtaa joksikin toiseksi tietotyypiksi),
- suorittaa ilmauksen tai arvon tyypinmuunnos (typecast) (`treat as`, `cast as` ja `validate`).

Kuten jo aikaisemmin on voitu havaita, noudattaa XQuery omaa notaatiotaan, joka eroaa XML-syntaksista. XQuerystä on olemassa myös XML-syntaksia noudattava versio [MMRR03], joka on nimeltään XQueryX.

Luku 4

XML-tietojen integroiminen

4.1 Tietojen integroinnin ongelmat

Kun tietokuutio muodostetaan XML-dokumenteissa olevien tietojen perusteella, on tärkeää voida varmistua siitä, että XML-dokumenteista kerätään juuri oikeat tiedot. XML-muotoisten tietojen integrointi on erittäin haastavaa, sillä XML-merkintäkielen puolirakenteisuudesta johtuen XML-dokumenteissa on samaan asiayhteyteen kuuluva tieto mahdollista esittää usealla erilaisella tavalla. XML-dokumenteissa tietojen esitystavan heterogeenisuus voi olla semanttista ja/tai rakenteellista.

Semanttinen heterogeenisuus merkitsee sitä, että keskenään samaa tarkoittavaa tietoa sisältävät elementit tai attribuutit on nimetty eri tavalla. Semanttista heterogeenisuutta esiintyy myös relaatiotietokantojen yhteydessä, ja se on eräs keskeinen relationaalisten tietolähteiden integroinnin ongelma. Ontologioiden käytöstä on viime aikoina tullut yleinen semanttisen heterogeenisuuden ongelman ratkaisemisessa käytetty lähestymistapa sekä relaatio- että XML-muotoisten tietojen integroimisessa. Semanttisen heterogeenisuuden ongelman ratkaiseminen on tietojen integroinnin vakava haaste, mutta se sivuutetaan tässä tutkielmassa.

Rakenteellista heterogeenisuutta on XML-dokumenteissa kahdenlaista. Ensimmäinen liittyy tietojen ryhmittelyyn XML-dokumentin puurakenteessa ja toinen tietojen esittämiseen ekstensionaalisella ja intensionaalisella tasolla. Koska XML-dokumenttia vastaava tietorakenne on (yleinen) puu, XML-merkintäkielessä on olemassa samaa tarkoittavan tiedon esittämiseksi lukuisia vaihtoehtoja. Vastaava ongelma esiintyy myös relaatiotietokantojen yhteydessä: eri tietokannoissa samaa tarkoittava tieto on esitetty eri tavoin (erinimiset attribuutit, erilainen attribuuttien ryhmittely, samannimiset attribuutit ovat eri tietotyyppiä jne.). XML-dokumenttien kohdalla erityisenä vaikeutena on, että samaa tarkoittava tieto voidaan esittää eri tavoin myös yksittäisen dokumentin sisällä. Relaatiotietokantojen yhteydessä tämä ei ole mahdollista, sillä yksittäisen tietokantataulun riveillä samaa tarkoittava tieto on aina esitetty täsmälleen samalla tavalla.

Sekä relaatiotietokannan tauluissa että XML-dokumenteissa samaa tarkoittava tieto voidaan esittää sekä intensionaalisella että ekstensionaalisella tasolla eli siis kaaviossa tai datana. Relaatiotaulussa tämä tarkoittaa, että tieto voi olla attribuuttin nimenä taulun

kaaviossa tai arvona attribuuttia vastaavassa taulun sarakkeessa. XML-dokumentissa tieto on puolestaan intensionaalisella tasolla, jos se on elementin tai XML-attribuutin nimenä, ja ekstensionaalisella tasolla, jos se esitetään elementin sisällössä tai attribuutin arvona. Relaatiotietokantojen yhteydessä on perinteisesti otaksuttu, että tarvittava tieto sijaitsee nimenomaan ekstensionaalisella tasolla. Esimerkiksi SQL-kyselykieli perustuu tälle otaksumalle. Lakshmanan, Sadri ja Subramanian [LSS01] ovat kehittäneet SQL:n laajennokseksi SchemaSQL-nimisen kyselykielen, jolla on mahdollista ulottaa SQL-kysely myös tietokannan kaaviotasolle. Myös XML-kyselykielissä on ollut pääosin lähtökohtana se, että etsittävää informaatiota esiintyy ainoastaan dokumenttien ilmentymätasolla.

4.2 Polkuorientoituneiden kyselykielten puutteet XML-tiedon integroinnissa

XQuerystä on tulossa käytetyin XML-kyselykieli, mutta tietojen integroinnin näkökulmasta se on kuitenkin hieman ongelmallinen. Pääsyynä tähän on XQueryn voimakas polkuorientoituneisuus eli se, että tietojen poiminta XML-dokumenteista perustuu XQueryssä XPath-polkuilmausten käyttöön. Jotta XML-dokumentista saadaan poimittua kyselyn kannalta relevanttia informaatiota, pitää kyselyn tekijän osata laatia oikeanlaiset polkuilmaukset. Oikeanlaisen polkuilmauksen laatiminen puolestaan edellyttää tarkkaa tietoa lähdedokumentin rakenteesta.

Jos lähdedokumenttiin liittyy dokumenttikielioppi, joko DTD- tai XMLSchema-kuvaus, kyselyntekijä voi käyttää sitä apuna polkuilmausten laatimisessa. Kuten on jo aiemmin todettu, dokumenttikielioppi ei kuitenkaan välttämättä kuvaa tarkasti yksittäisen XML-dokumentin rakennetta vaan ainoastaan tietyntyyppisten dokumenttien luokan rakenteen. Lisäksi jos dokumenttikielioppi on hyvin laaja, voi siitä olla polkuilmausten laatimisessa enemmän haittaa kuin hyötyä. Jos dokumenttikielioppeja ei ole saatavilla, on käyttäjän tutustuttava suoraan lähdedokumentteihin voidakseen muotoilla oikeanlaiset polkuilmaukset.

Jos kyselyntekijä ei kuitenkaan tunne dokumentin rakennetta, hän voi käyttää tarkan XPath-polkuilmauksen sijasta polkuilmausten lyhennysmerkintöjä. Esimerkiksi ilmauksella `doc("example.xml")//author` viitataan XML-dokumentissa dokumenttipuun millä tahansa tasolla olevan `author`-nimisen elementin sisältöön. Lyhennysmerkintöjen käyttö johtaa kuitenkin epätarkkuuteen. Jos esimerkiksi dokumentissa `example.xml` on `author`-elementtejä on sijoitettu sekä `book`- että `article`-nimisten elementtien sisältöosaan, viitataan edellisellä polkuilmauksella kummankin elementin sisällä oleviin `author`-elementteihin, vaikka tarkoituksena olisikin ollut poimia ainoastaan `article`-elementeissä olevien `author`-elementtien sisällöt.

Kun tietoja integroidaan laajasta kokoelmasta rakenteeltaan heterogeenisia XML-dokumentteja, polkuilmausten käytöstä aiheutuu myös se, että kyselyn tekijä joutuu laatimaan jokaista rakenteeltaan (ja itse asiassa myös nimennältään) erilaista XML-dokumenttia kohden oman XQuery-kyselyn. Jos rakenne-eroavuuksia on paljon, on helppo kuvitella, millainen tehtävä XQuery-kyselyn laatijalla on edessään. Lisäksi rakenne-eroavuuksien huomioiminen edellyttää periaatteessa dokumenttikokoelman kaikkien do-

kumenttien läpikäymistä. (Dokumenttikielioppien mukanaolo helpottaa hieman tätä tehtävää.) XQuery-kysely on mahdollista "kääntää" (translate) automaattisesti (tai puoliautomaattisesti) toiseksi XQuery-kyselyksi, mutta tämänkaltaisessa kyselyn muuntamisprosessissa on vaarana, että lopputuloksena saadaan kysely, joka tuottaa alkuperäisen kyselytarpeen kannalta epärelevantteja vastauksia.

XPath-polkuilmauksissa otaksutaan, että kyselyssä etsittävä tieto on pääasiassa XML-dokumentin elementin tai attribuutin sisältönä, ts. ekstensionaalisella tasolla. Elementtien ja attribuuttien nimiin, so. intensionaaliselle tasolle, viittaaminen ei onnistu pelkkien polkuilmausten avulla, vaan edellyttää XQueryssä funktioiden ja muuttujien käyttöä. W3-konsortion spesifikaatiossa [FMM+05], jossa määritellään XPath 2.0 ja XQuery 1.0 -kyselykielten tietomalli, on esitelty funktio `node-name()`, joka palauttaa XML-dokumentin minkä tahansa solmun nimen. Jos esimerkiksi muuttuja `$i` on sidottu polkuilmaukseen `doc("example.xml")//.`, palauttaa funktiokutsu `node-name($i)` dokumentin `example.xml` kaikkien solmujen nimet. Lisäksi on syytä huomata, että polkuilmauksen kohdistuessa sellaiseen elementtiin, jolla on alaelementtejä, polkuilmaus palauttaa kyseisestä elementistä alkavan koko alipuun XML-muodossa. Jos halutaan viitata pelkästään elementin tekstisisältöön, on käytettävä funktiota `data()` ja muuttujia.

Edellä on esitetty muutamia polkuorientoituneiden kyselykielten puutteellisuuksia XML-tiedon integroinnin näkökulmasta: vaikeus käsitellä dokumenttien rakenteellista tuntemattomuutta ja rakenteellista (ja semanttista) heterogeenisuutta sekä ulottaa kyselyt koskemaan sekä intensionaalista että ekstensionaalista tasoa. XML-muotoisten tietojen integroinnin kohdalla on siis selkeästi olemassa tarve kyselyprimitiiville, joka paikkaisi edellä mainitut polkuorientoituneiden kyselykielten puutteet ja tarjoaisi lisäksi intuitiivisen ja deklaratiivisen tavan kohdistaa kyselyjä XML-dokumentteihin. Ennen kuin seuraavassa luvussa esitellään tällainen kyselyprimitiivi, tarkastellaan vielä lyhyesti erästä keinoa käsitellä XML-dokumenttien rakenteellista tuntemattomuutta.

Tiedonhakututkimuksessa (information retrieval) on ollut lähtökohtana, että käyttäjä tietää, millaista tietoa tarvitsee, mutta ei tiedä sitä, mistä ja millaisista dokumenteista sitä on saatavilla. Tiedonhakututkimuksen tarkoitus on tarjota menetelmiä, joilla käyttäjä voi tyydyttää tiedontarpeensa ilman, että tämän tarvitsee laatia kompleksisia tai vahvasti strukturoituja kyselyjä tai tietää, miten kysely tiedonhakujärjestelmässä evaluoidaan. Tiedonhaku perustuu tavallisimmin hakusanojen (avainsanojen) käyttöön: toisin sanojen kyselyn tulokseksi tuotetaan dokumentteja, joihin sisältyvät kaikki tai osa annetuista hakusanoista. Internetin hakukoneet toimivat vastaavalla periaatteella. Niissä käyttäjä antaa hakukoneelle syötteeksi hakusanoja, ja hakukone palauttaa tulokseksi joukon sellaisia dokumentteja, joissa esiintyy käyttäjän antamat avainsanat. Tiedonhakujärjestelmissä hakutulokset järjestetään esimerkiksi hakusanojen esiintymien lukumäärien perusteella. Haun tulosta arvioidaan mm. saannin ja tarkkuuden suhteen. Tiedonhakututkimuksessa on aikaisemmin keskitytty pääasiassa tiedonhakuun HTML-dokumenteista ja rakenteettomista tekstidokumenteista, mutta myös XML-tiedonhaku on kasvavan kiinnostuksen kohteena.

On kehitetty myös sellaisia XML-kyselykieliä, joissa tavanomaisten strukturoitujen kyselyjen lisäksi on mahdollista tehdä myös tiedonhakutyyppisiä kyselyjä (ks. esim.

[FG01]). XML-tiedonhaussa hakusanojen esiintymiä etsitään paitsi elementtien ja attribuuttien sisältöosasta myös niiden nimistä. Voidaan nimittäin ajatella Lin, Yun ja Jagadishin [LYJ04] tavoin, että XML-dokumentin yksittäinen solmu ja siihen mahdollisesti liittyvä alirakenne kuvaa jotakin reaalimaailman entiteettiä. Solmun nimi identifioi ko. entiteetin. Lisäksi ei ole enää riittävää, että kyselyn tulokseksi palautetaan XML-dokumentteja tai niiden osia pelkästään esimerkiksi hakusanojen esiintymistiheyksien perusteella, vaan hakusanoja vastaavien XML-rakenteiden tulee olla keskenään mielekkäässä suhteessa. Jos XML-dokumenteista poimittaisiin tietoja yksinomaan solmujen nimissä tai sisällössä esiintyvien hakusanojen perusteella, olisi mahdollista, että yhdistetään keskenään semanttisesti yhteen kuulumatonta informaatiota.

Yksi ratkaisu hakutuloksen kontekstualisointiin on ollut *Lowest Common Ancestor* (LCA) - eli *matalimman yhteisen esivanhemman semantiikan* käyttö (ks. esim. [LYJ04], [XP05]). Siinä on lähtökohtana, että hakusanoja vastaavia XML-rakenteita etsitään koko dokumentin sijasta ainoastaan dokumentin sellaisista alirakenteista, joiden sisällä kaikki annetut avainsanat esiintyvät. LCA-semantiikka toimii joissakin tapauksissa hyvin, mutta myös sillä on ongelmansa, kuten edempänä tullaan näkemään. Siirrytään nyt kuitenkin tarkastelemaan XML-tietojen poimintaan kehitettyä uutta kyselyprimitiiviä.

4.3 Uusi kyselyprimitiivi XML-tiedon käsittelyyn

4.3.1 Komponentti-ilmaukset

Kyselyprimitiivin kehittämisen lähtökohtana on ollut, että XML-dokumentin elementtejä ja attribuutteja on voitava käsitellä kyselyissä yhdenvertaisesti ja että sekä niiden nimiin että sisältöihin on voitava viitata helposti. XML-dokumentin elementtien ja attribuuttien välillä ei nimittäin ole muuta eroa kuin, että ainoastaan elementillä voi olla omia alirakenteita, so. attribuutteja tai alaelementtejä. XML-dokumentin elementtejä ja attribuutteja kutsutaan tästä eteenpäin komponenteiksi. XML-dokumentti on näin ollen komponenteista koostuva yhdistelmä eli komposiitti. Jokaisella XML-dokumentin komponentilla on nimi ja arvo. Komponentin nimi vastaa elementin tai attribuutin nimeä. Komponentin arvo on puolestaan attribuutin arvo tai elementin linearisoitu sisältö, joka on saatu poistamalla elementin ja siihen mahdollisesti sisältyvien alaelementtien alku- ja lopputunnisteet (ja siten myös attribuutit) ja konkatenoimalla jäljellejäävä tekstisisältö.

Mielivaltaiseen XML-dokumentin komponenttiin viitataan kyselyprimitiivissä *komponentti-ilmauksella*. Komponentti-ilmauksen yleinen rakenne on esitetty määritelmässä 4.1.

Määritelmä 4.1:

$<CompName>(<CompValue>)$

Varsinaisessa komponentti-ilmauksessa määritelmässä 4.1 esiintyvä ilmaus $<CompName>$ korvataan komponentin nimeen viittaavalla muuttujalla tai atomisella arvolla ja vastaavasti ilmaus $<CompValue>$ komponentin arvoon viittavalla muuttujalla tai ato-

misella arvolla. Logiikkaohjelmoinnista omaksuttua tapaa noudattaen muuttujan nimen on aina alettava isolla alkukirjaimella. Vastaavasti atominen ilmaus alkaa pääsääntöisesti pienellä alkukirjaimella. Jos kuitenkin halutaan, että atominen ilmaus alkaa isolla alkukirjaimella, on ilmaus kirjoitettava yksinkertaisten lainausmerkkien (') väliin. Komponentti-ilmauksessa atominen ilmaus on lisäksi aina kirjoitettava yksinkertaisten lainausmerkkien väliin, jos se sisältää muita kuin aakkos-numeerisia merkkejä tai väli- tai alaviivan.

Komponentti-ilmauksessa esiintyvistä muuttujista tai atomisista ilmauksista riippuen yksittäinen komponentti-ilmaus voi viitata intensionaaliselle, ekstensionaaliselle tai intensionaalis-ekstensionaaliselle tasolle. Koska komponentti-ilmauksessa `N(smith)` komponentin nimeen viitataan muuttujalla, kohdistuu ilmaus intensionaaliselle eli kaaviotasolle. Ilmauksessa `author(V)` viitataan muuttujalla puolestaan komponentin arvoon, joten se kohdistuu ekstensionaaliselle eli ilmentymätasolle. Komponentti-ilmaus `N(V)` kohdistuu sekä intensionaaliselle että ekstensionaaliselle tasolle, sillä siinä sekä komponentin nimeen että sisältöön viitataan muuttujalla. Jos komponentti-ilmauksessa ei esiinny ilmauksen `author(smith)` tapaan lainkaan muuttujia, testataan ilmauksella loogista ehtoa, ts. sisältyykö se vai eikö se sisälly kulloinkin käsiteltävään XML-dokumenttiin, ja tapauksesta riippuen se palauttaa arvon tosi tai epätosi. Komponentti-ilmauksessa voidaan viitata myös pelkästään komponentin nimeen. Esimerkiksi komponentti-ilmaus `author` viittaa XML-dokumentin mihin tahansa `author`-nimiseen komponenttiin ja ilmaus `X` dokumentin mielivaltaisen komponentin nimeen.

4.3.2 Kyselyprimitiivin perusmuodot

Varsinainen kyselyprimitiivi on nimeltään `is_component_of` ja sen avulla on mahdollista poimia XML-dokumenteissa esiintyviä komponentteja komponentti-ilmauksia käyttäen. Kyselyprimitiivistä on kaksi perusmuotoa. Niiden avulla haetaan (1) XML-dokumentin komponentteja tai vaihtoehtoisesti (2) XML-dokumentin komponentteja ja niiden alikomponentteja. Kyselyprimitiivin `is_component_of` perusmuotojen yleinen rakenne on esitetty määritelmässä 4. 2.

Määritelmä 4.2:

(1)

<*CompExpr*> `in` <*DocName*>

(2)

<*CompExpr2*> `is_component_of` <*CompExpr1*> `in` <*DocName*>

Varsinaisissa kyselyissä määritelmän 4.2 ilmaus <*DocName*> korvataan XML-dokumentin nimeen viittaavalla muuttujalla tai atomisella ilmauksella ja ilmaukset <*CompExpr*>, <*CompExpr1*> ja <*CompExpr2*> millä tahansa komponentti-

ilmauksella. Jos dokumentin nimeen viitataan muuttujalla, etsitään – muuttujien alustuksista riippuen – annettuja komponentti-ilmauksia vastaavia komponentteja tai niiden haluttuja ominaisuuksia saatavilla olevan dokumenttikokoelman kaikista dokumenteista. Jos dokumentin nimi on puolestaan annettu, etsitään kyselyssä annettuja komponentti-ilmauksia vastaavia XML-rakenteita ainoastaan kyseisestä dokumentista (mikäli se sisältyy dokumenttikokoelmaan). Dokumentin nimi merkitään muodossa '<*FileName*>.xml', missä <*FileName*> on mikä tahansa sallittu tiedostonimi.

Esimerkiksi kysely `author(V) in 'example.xml'` palauttaa XML-dokumenttiin `example.xml` kuuluvien author-nimisten komponenttien esiintymien arvot. Sitä vastaavat XPath-ilmaukset `doc("example")//author` ja `doc("example")//@author`. Kysely `author(V) is_component_of article in 'example.xml'` palauttaa vastaavasti XML-dokumentissa `example.xml` `article`-nimisiin komponentteihin sisältyvien `author`-nimisten alikomponenttien esiintymien arvot. Kyselyä vastaa XPath-ilmaus `doc("example")//article//author`. Vastaavasti kysely `E in D` palauttaa dokumenttikokoelmaan kuuluvan mielivaltaisen XML-dokumentin mielivaltaisen komponentin nimen ja kysely `C(V) is_component_of E in D` palauttaa mielivaltaisen XML-dokumentin kaikki sellaiset komponentit (itse asiassa elementit) `E`, joilla on alikomponentteja, ja kaikki komponentin `E` välittömät ja välilliset alikomponentit nimineen ja sisältöineen. Edellisiä kyselyitä ei vastaa suoraan mikään yksittäinen XPath-ilmaus.

Toisin kuin XPath-ilmauksissa `is_component_of`-kyselyprimitiivin avulla on mahdollista viitata suoraan useampaan alikomponenttiin. Tällöin haluttuihin XML-rakenteisiin viittaavat komponentti-ilmaukset kirjoitetaan kyselyprimitiivin toisessa perusmuodossa `is_component_of`-sanan vasemmalle puolelle aaltosulkeiden (`{,}`) väliin pilkulla (`,`) toisistaan erotettuna. Näin esimerkiksi kysely `{author(A), year(Y), title(T)} is_component_of bibliography in 'example.xml'` palauttaa kaikki XML-dokumentissa `example.xml` oleviin `bibliography`-nimisiin komponentteihin sisältyvien `author`-, `year`- ja `title`-nimisten komponenttien arvot. Edellisen kaltaisissa kyselyissä aaltosulkeiden väliin kirjoitettujen komponentti-ilmausten järjestys ei ilmaise etsittävien komponenttien keskinäistä järjestystä (sekvenssiä) yksittäisessä XML-dokumentissa, vaan etsittävät komponentit voivat sijaita yhteisen yläelementtinsä sisällä missä tahansa järjestyksessä.

Jos johonkin elementtiin sisältyy useita samannimisiä komponentteja, on nyt kuitenkin toisaalta mahdollista, että toisiinsa ainoastaan satunnaisesti liittyvät komponentit tulevat yhdistetyiksi keskenään. Jos esimerkiksi kysely `{author(A), year(Y), title(T)} is_component_of publications in D` kohdistetaan kuvan 4.1 kohtaa (1) vastaavaan XML-dokumenttiin, saadaan vastaukseksi 8 erilaista komponenttien arvojen kombinaatiota. Voidaan kuitenkin helposti nähdä, että relevantteja arvokombinaatioita on ainoastaan kaksi. Edellä kuvatun kaltaisen tilanteen estämiseksi pitää jotenkin olla mahdollista kontrolloida, millaisia komponentteja kombinaatioihin hyväksytään. Yksi mahdollisuus on LCA-tyyppisen semantiikan hyödyntäminen sopivien komponenttien valinnassa. Kun nimittäin `is_component_of`-kyselyssä poistetaan aaltosulkeiden välissä olevista komponentti-ilmauksista muuttujat, voidaan ajatella, että jäljelle jääneet atomiset arvot ovat ikään kuin avainsanoja, joiden esiintymiä XML-dokumenteista sitten etsitään.

4.4 LCA-semantiikka

Oletetaan ensin, että XML-dokumentin solmut on indeksoitu. Indeksoidaan XML-dokumentin solmut artikkelissa [Nie83] esitetyn indeksointitavan mukaisesti. Indeksi on väkästen (<,>) välissä esitetty äärellinen joukko luonnollisia lukuja. Dokumenttipuun juurisolmu saa indeksin <1> Minkä tahansa muun kuin juurisolmun indeksi on $<\xi, i>$, joka saadaan konkatenoimalla indeksoitavan solmun vanhempisolmun indeksi ξ luvulla i, joka seuraa, kun solmusta ξ alkavaa osadokumenttipuuta kuljetaan esijärjestyksessä syvyys ensin -suuntaisesti. Jokaisella dokumenttipuun tasolla indeksointi aloitetaan uudestaan luvusta 1. Kuvassa 4.1 on indeksoitu joukko XML-dokumentteja edellä esitettyä menetelmää käyttäen. Dokumenttipuun solmuiksi katsotaan dokumentin elementtien ja attribuuttien nimien esiintymät sekä elementtien ja attribuuttien sisältöosaan kuuluvat muut kuin välilyöntimerkistä koostuvat yksittäiset merkkijonot (so. sanat ja numerot).

LCA-semantiikka ei varsinaisesti edellytä dokumenttien indeksointia, mutta indeksointi helpottaa sen periaatteen esittämistä. Kuten on jo aiemmin todettu, LCA-semantiikka tarjoaa menetelmän, jonka avulla hakusanojen etsintä voidaan kohdentaa koko dokumentin sijasta sellaisiin dokumentin osadokumentteihin, joiden juurisolmu on hakusanoja vastaavien XML-dokumentin solmujen matalin yhteinen esivanhempi. Juurisolmu on XML-dokumentin ainoa solmu, jolla ei ole esivanhempisolmua. Jos jokin etsittävä hakusana samaistuu XML-dokumentin juurisolmuun, ei kyseiselle hakusanakombinaatiolle ole määritelty matalinta yhteistä esivanhempaa. Esimerkiksi kuvan 4.1 kohtaa (2) vastaavassa XML-dokumentissa hakusanoja `author`, `year` ja `title` vastaavien solmujen matalimmat yhteiset esivanhempisolmut ovat solmun `article` esiintymät indekseillä <1,1> ja <1,2>. Kun hakusanoja vastaavien solmujen etsintä suoritetaan näistä solmuista alkavien osadokumenttien sisällä, saadaan tulokseksi relevantteja solmukombinaatioita. Kun tarkastellaan vielä löydettyjä relevantteja solmuja vastaavien indeksien kombinaatioita (<1,1,1,1>, <1,1,2>, <1,1,3>), (<1,1,1,2>, <1,1,2>, <1,1,3>), (<1,2,1>, <1,2,2>, <1,2,3>), havaitaan helposti, että löydettyjen solmujen matalimpien yhteisten esivanhempisolmujen indeksit vastaavat indeksimonikkojen indeksien *pisimpiä yhteisiä alukkeita* (longest common prefix).

Xu ja Papakonstantinou [XP05] ovat määritelleet *Smallest Lowest Common Ancestor* -semantiikan (SLCA). SLCA-semantiikka lisää perinteiseen LCA-semantiikkaan sen rajoituksen, että matalimman yhteisen esivanhempisolmun pitää olla myös *suppein*. Toisin sanoen annettuja hakusanoja vastaavasta LCA-solmusta alkavaan dokumenttipuuhun ei saa sisältyä sellaista alipuuta, jossa esiintyy kaikkia annettuja avainsanoja vastaavat solmut. SLCA-semantiikan avulla voidaan varmistaa, että jos XML-dokumentissa on usealla eri dokumenttihierarkian tasolla samannimisiä tai -sisältöisiä solmuja, niistä valituksi tulevat ainoastaan hierarkiassa toisiaan lähinnä olevat solmut. Esimerkiksi kuvan 4.1 kohtaa (3) vastaavassa XML-dokumentissa hakusanoja `author`, `year`, ja `title` vastaavien LCA-solmujen indeksit ovat <1,1> ja <1>. Niistä indeksiä <1> vastaava solmu on selvästi epärelevantti. Edellisiä hakusanoja vastaavan SLCA-solmun indeksi on puolestaan <1,1>, ja kyseisestä solmusta alkavassa alipuussa evaluoitava kysely tuottaa oikean vastauksen. Li, Yu ja Jagadish [LYJ04] ovat määritelleet *Meaningful Lowest*

Common Ancestor -semantiikan (MLCA), joka on periaatteeltaan samanlainen SLCA-semantiikan kanssa, mutta siinä annetaan pelkän SLCA-solmun lisäksi selitys (luettelo solmuista, joiden SLCA kyseinen solmu on) siitä, miksi solmu on mielekäs matalin yhteinen esivanhempisolmu.

Jos XML-dokumentissa on toistuvia alarakenteita, joissa oleviin tietoihin pitää yhdistää dokumentin ylärakenteissa olevia tietoja, tuottavat LCA-semantiikkaa ja sen edellä esitettyjä johdannaisia hyödyntävät avainsanahaut helposti vääriä tuloksia. Esimerkiksi kuvan 4.1 kohtaa (4) vastaavassa XML-dokumentissa hakusanoja `author`, `year`, ja `title` vastaavan LCA- ja SLCA-solmun indeksi on <1>. Indeksi <1> vastaa XML-dokumentin juurisolmuja, joten haku tuottaa jälleen tulokseksi hakusanoja vastaavien solmujen mielivaltaisia kombinaatioita. Myös hakusanoja vastaava MLCA-solmu on indeksiltään <1>, mutta MLCA-semantiikassa tuotetaan pelkän solmun lisäksi tieto siitä, minkä solmujen MLCA löydetty solmu on.

4.5 Suppeimman mahdollisen kontekstin semantiikka

Edellä kuvatun ongelman ratkaisemiseksi on tutkielman yhteydessä kehitetty *suppeimman mahdollisen kontekstin semantiikka*. Suppeimman mahdollisen kontekstin semantiikassa on – SLCA- ja MLCA-semantiikkojen tavoin – lähtökohtana, että hakutulokseen valitaan ainoastaan sellaisia solmukombinaatioita, joissa solmut ovat dokumenttihierarkiassa mahdollisimman lähellä toisiaan. Solmujen välisten mielekkäiden suhteiden määrittely poikkeaa siinä kuitenkin SLCA- ja MLCA-semantiikkojen tavasta määritellä solmujen keskinäiset suhteet. Kun LCA-pohjaisissa semantiikoissa palautetaan solmu, josta alkavassa alipuussa kyselyn evaluoiminen tuottaa mahdollisimman hyvän lopputuloksen, palautetaan suppeimman mahdollisen kontekstin semantiikassa suoraan sellaiset annettuja hakusanoja vastaavien XML-dokumenttien solmujen monikot (tuple), joihin sisältyvät solmut ovat keskenään mielekkäässä suhteessa. Palautetut solmumonikot muodostavat annettuja hakusanoja vastaavien dokumentin solmujen *suppeimmat mahdolliset kontekstit*.

Tarkastellaan suppeimman mahdollisen kontekstin muodostamisen ideaa esimerkin avulla. Ajatellaan, että halutaan poimia kuvan 4.1 kohtaa (4) vastaavasta XML-dokumentista hakusanoja `author`, `year` ja `title` vastaavien solmujen mielekkäät kombinaatiot. Hakusanaa `author` vastaavan solmun indeksi on <1,1>. Hakusanaa `year` vastaavat puolestaan solmujen indeksit <1,2,1>, <1,3,1> ja hakusanaa `title` indeksit <1,2,2,1>, <1,3,2,1>. Muodostetaan hakusanoja vastaavien solmujen indekseistä järjestettyjä pareja siten, että jokaiseen indeksipariin kuuluvilla indekseillä on mahdollisimman pitkät yhteiset alukkeet. Esimerkkidokumentin solmuista muodostetaan siis indeksiparien joukko

A = {<(author,<1,1>),(year,<1,2,1>)>,
<(author,<1,1>),(year,<1,3,1>)>,
<(author,<1,1>),(title,<1,2,2,1>)>,
<(author,<1,1>),(title,<1,3,2,1>)>,
<(year,<1,2,1>),(author,<1,1>)>,
<(year,<1,3,1>),(author,<1,1>)>,
<(year,<1,2,1>),(title,<1,2,2,1>)>,
<(year,<1,3,1>),(title,<1,3,2,1>)>,
<(title,<1,2,2,1>),(author,<1,1>)>,
<(title,<1,3,2,1>),(author,<1,1>)>,
<(title,<1,2,2,1>),(year,<1,2,1>)>,
<(title,<1,3,2,1>),(year,<1,3,1>)> }.

Nyt esimerkiksi indeksipari <(year, <1,2,1>), (title, <1,3,2,1>)> ei tule valituksi joukkoon A, sillä indeksiparin <(year, <1,2,1>), (title, <1,2,2,1>)> pisin yhteinen aluke (<1,2>) on pidempi kuin sen pisin yhteinen aluke (<1>). Näin ollen ainoastaan jälkimmäinen indeksipari valitaan.

Muodostetaan seuraavaksi joukon A alkioista indeksiparikolmikkoja (<(author, i_1),(year, i_2)>, <(year, i_2), (title, i_3)>, <(title, i_3), (author, i_1)>) siten, että indeksipari valitaan kolmikkoon ainoastaan silloin, jos indeksiparin käänteispari (so. sellainen indeksipari, jossa komponentit ovat käänteisessä järjestyksessä) sisältyy joukkoon A ja indeksiparikolmikoiden ensimmäisen komponentin ensimmäinen indeksi ja viimeisen komponentin toinen indeksi ovat samat (i_1). Tällaiset indeksiparikolmikot muodostavat joukon

C = {(<(author,<1,1>),(year,<1,2,1>)>,
<(year,<1,2,1>),(title,<1,2,2,1>)>,
<(title,<1,2,2,1>),(author,<1,1>)>),
(<(author,<1,1>),(year,<1,3,1>)>,
<(year,<1,3,1>),(title,<1,3,2,1>)>,
<(title,<1,3,2,1>),(author,<1,1>)>) }.

Annettuja hakusanoja vastaavien solmujen suppein mahdollinen konteksti muodostetaan nyt poimimalla joukon C kolmikkoihin sisältyvät erilliset indeksit. Näin ollen hakusanojen author, year ja title suppeimmat mahdolliset kontekstit annetun esimerkkidokumentin tapauksessa ovat <(author, <1,1>), (year, <1,2,1>), (title, <1,2,2,1>)> ja <(author, <1,1>), (year, <1,3,1>), (title, <1,3,2,1>)>.

Joukon A indeksiparien voidaan itse asiassa ajatella olevan dokumenttipuun nimettyjen solmujen välisiä *suunnattuja kaaria* (directed arc). Joukon C alkiot ovat tällöin puolestaan joukkoon A kuuluvista kaarista muodostettuja *suunnattuja syklisiä graafeja* (directed cyclic graph). Joukon C graafien solmut on lisäksi valittu siten, että jokaisesta graafin solmusta on (joukossa A) oltava yhteys graafin jokaiseen toiseen solmuun. Tällaista graafia, jossa graafin jokainen solmu on jokaisen muun solmun naapuri (adjacent), kutsutaan *täydelliseksi graafiksi* (complete graph). Suppeimman mahdollisen kontekstin

muodostavat näin ollen dokumenttipuun solmuista muodostettujen täydellisten graafien erilliset solmut.

Suppeimman mahdollisen kontekstin semantiikka mahdollistaa tiedontarpeen kannalta relevanttien tietojen poimimisen XML-dokumenteista kaikissa sellaisissa tapauksissa, joissa SLCA- ja MLCA-semantiikat tuottavat relevantteja tietoja. Tämän lisäksi suppeimman mahdollisen kontekstin semantiikkaan perustuva haku tuottaa relevantteja vastauksia evaluoitaessa kyselyä sellaisessa XML-dokumentissa, jossa dokumentin ylärakenteissa esitettyä informaatiota pitää yhdistää dokumentin alarakenteissa esitettyjen tietojen kanssa. Tällaisessa tapauksessa sekä SLCA- että MLCA-semantiikkaan perustuvat tiedonhakujärjestelmät tuottavat epärelevantteja vastauksia.

4.6 Kyselyprimitiivin käyttö

Havainnollistetaan `is_component_of`-kyselyprimitiivin toimintaa vielä yksinkertaisen kyselyesimerkin avulla. Seuraavassa kyselyesimerkissä primitiivin `is_component_of` avulla tehdyn kyselyn rinnalla esitetään XQueryn avulla tehty vastaava kysely.

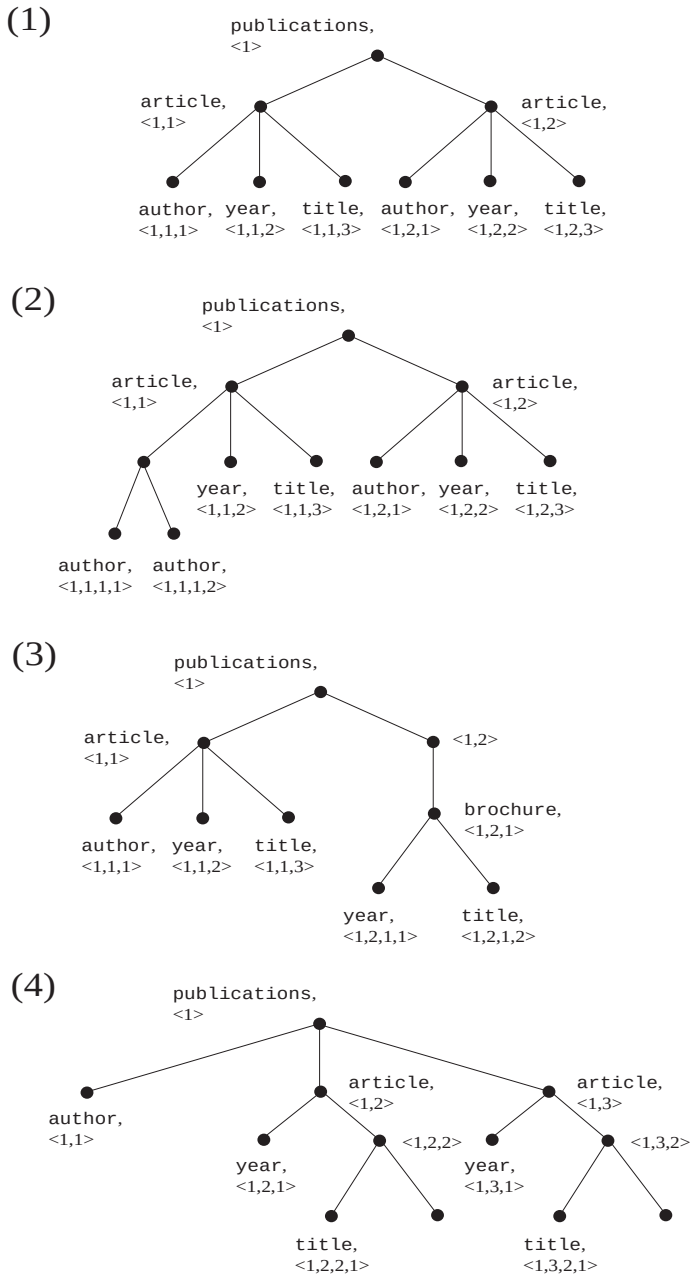
Tarkastellaan ensin kuvassa 4.2 esitettyä XML-dokumenttia, joka sisältää tieteellisten artikkeleiden bibliografisia tietoja. Artikkeleista on ilmoitettu niiden tekijä (`author`), otsikko (`title`), julkaisupäivämäärä (`publication_date`) ja tyyppi (`article_type`). Esimerkkidokumentissa esiintyy kaikkia aiemmin esitettyjä XML-dokumenttien rakenteellisen heterogeenisuuden lajeja. Esimerkiksi tieto artikkelin tyypistä esiintyy dokumentin rivillä (3) attribuutin nimessä, rivillä (13) attribuutin arvossa, rivillä (27) elementin arvossa ja rivillä (31) elementin nimessä.

Nyt halutaan tehdä kysely, jonka tuloksena saadaan esimerkkidokumentin kaikkien artikkelien otsikot, kirjoittajat, julkaisuvuodet ja tyypit artikkelin kirjoittajan mukaan järjestettynä. Oletetaan, että kyselyn tekijä tuntee käytettävän XML-dokumentin rakenteen. XQuery-kyselykielellä toteutettu kysely on esitetty kuvassa 4.3. Kyselyn rivillä (1) annetaan kyselyssä käytettävän XQuery-kielen versio. Rivillä (2) esitellään iteraattori `$i`, joka käy `for`-silmukassa esimerkkidokumentin läpi dokumenttijärjestyksen mukaisesti. Riveillä (3 - 6) kiinnitetään `let`-lauseissa XPath-ilmausten avulla muuttujat `$a1` - `$a4` esimerkkidokumentissa artikkelien kirjoittajan ilmaisevien solmujen ilmentymiin. Vastaavasti riveillä (7 - 10), (11 - 14) ja (15 - 18) kiinnitetetään muuttujat `$t1` - `$t4`, `$y1` - `$y4` ja `$p1` - `$p4` esimerkkidokumentissa artikkelin otsikon, julkaisuvuoden ja tyypin ilmaisevien solmujen ilmentymiin. Kaikki palautettavat solmut on määriteltävä erikseen, sillä yksittäistä XPath-ilmausta vastaa useissa tapauksissa *joukko* XML-dokumentin solmuja eikä ainoastaan yksittäinen solmu. Kuten havaitaan, muuttujia `$a3` ja `$a4` vastaa sama XPath-ilmaus, mutta muuttujat on tässä selkeyden vuoksi haluttu määritellä erikseen. Riviltä (19) alkavassa return-lauseessa määritellään `results`-niminen XML-elementti, jonka sisältöön riveillä (2 - 18) esitetyn kyselyn tulos palautetaan. `return`-lauseessa on käytetty XQuery-funktioita `data()` ja `name()` solmun (rakenteellisen) sisällön ja nimen palauttamiseen. Tutkielman kirjoittaja ei ole aktiivisesta etsimisestä huolimatta löytänyt XQuery-funktiokirjastosta funktiota, joka palauttaisi – esimerkiksi rivin (15) tapauksessa – solmun nimen silloin, kun solmun sisältö on tiedetty. XQuery-kyselyn vastaus on

annettu riveillä (26 - 31).

Tehdään seuraavaksi vastaava kysely `is_component_of`-kyselyprimitiiviä käyttäen. Kysely on annettu kuvassa 4.4. Kuten havaitaan, `is_component_of`-pritimiivin avulla tehtävä kysely vaatii XQuery-kyselyn 25 rivin sijaan viisi riviä. Kyselyn rivit (1 - 4) on erotettu toisistaan puolipisteellä (`;`), mikä merkitsee sitä, että niissä olevat `is_component_of`-ilmaukset ovat disjunktiivisessa suhteessa toisiinsa. Koko kyselyn tulos on näiden välikyselyjen palauttamien tulosten unioni. Rivin (1) `is_component_of`-ilmauksella saadaan palautettua ensimmäiseen artikkeliin liittyvät tiedot. Vastaavasti rivien (2) ja (3) `is_component_of`-ilmauksilla saadaan palautettua toisen ja kolmannen artikkelin tiedot.

Neljännen artikkelin tietojen hakeminen on hieman monimutkaisempaa. Se tapahtuu riveillä (4) ja (5) olevien kahden `is_component_of`-ilmauksen avulla. Ilmaukset on erotettu toisistaan pilkulla (`,`), mikä merkitsee sitä, että ne yhdistetään toisiinsa konjunktiolla. Ilmauksien palauttama tulos muodostuu siten ilmauksien palauttamien tulosten leikkauksena. (Ilmauksien palauttamat tulokset on yhdistetty niiden yhteisten komponenttien perusteella.) Ensimmäinen, rivin (4) `is_component_of`-ilmaus palauttaa kaikki sellaiset elementit (itse asiassa elementin nimet), joilla on välittöminä tai välillisinä komponentteina `article`-, `title`- ja `year`-nimiset elementit tai attribuutit. Näiksi elementeiksi valikoituvat dokumentin juurielementti `articles` sekä elementit `hikes` esimerkkidokumentin riviltä (22) ja `non_refereed` esimerkkidokumentin riviltä (31). Toisella, rivillä (5) olevalla `is_component_of`-ilmauksella tuotetaan puolestaan kaikki sellaiset esimerkkidokumentin elementit, joihin sisältyy `articles`-, `hikes`- tai `non_refereed`-elementti tai ne kaikki. Rivin (5) kysely palauttaa elementit `articles` (`hikes`- ja `non_refereed`-elementit sisältyvät siihen) ja `hikes` (`non_refereed`-elementti sisältyy siihen). Suppeimman mahdollisen kontekstin semantiikan nojalla muuttuja `P` arvottuu lopulta elementin nimellä `non_refereed` ja muuttuja `A` elementin nimellä `hikes`.



Kuva 4.1: Esimerkkejä tietojen ryhmittelystä XML-dokumenteissa.

```
(1)  <?xml version="1.0" encoding="UTF-8"?>

(2)  <articles>
(3)    <article refereed="yes" non_refereed="no">
(4)      <authors>
(5)        <author> smith </author>
(6)      </authors>
(7)      <title> art1 </title>
(8)      <publication_date>
(9)        <month> 3 </month>
(10)       <year> 2002 </year>
(11)     </publication_date>
(12)   </article>
(13)   <article author="jones" article_type="non_refereed">
(14)     <titles>
(15)       <title> art2 </title>
(16)     </titles>
(17)     <publication_date year="2003">
(18)       <day> 23 </day>
(19)       <month> 8 </month>
(20)     </publication_date>
(21)   </article>
(22)   <hikes>
(23)     <article title="art3">
(24)       <publication_date>
(25)         <day> 18 </day>
(26)         <month> 6 </month>
(27)         <article_type> refereed </article_type>
(28)         <year> 2004 </year>
(29)       </publication_date>
(30)     </article>
(31)     <non_refereed>
(32)       <article>
(33)         <title> art4 </title>
(34)         <publication_date year="2005">
(35)           <month> 11 </month>
(36)           <day> 11 </day>
(37)         </publication_date>
(38)       </article>
(39)     </non_refereed>
(40)   </hikes>
(41) </articles>
```

Kuva 4.2: Rakenteeltaan heterogeeninen XML-dokumentti `articles.xml`.

Kysely:

```
(1)  xquery version "1.0";

(2)  for $i in doc("articles.xml")

(3)   let $a1 := /articles/article/authors/author
(4)   let $a2 := /articles/article/@author
(5)   let $a3 := /articles/hikes
(6)   let $a4 := /articles/hikes

(7)   let $t1 := /articles/article/title
(8)   let $t2 := /articles/article/titles/title
(9)   let $t3 := /articles/hikes/article/@title
(10)  let $t4 := /articles/hikes/non_refereed/article/title

(11)  let $y1 := /articles/article/publication_date/year
(12)  let $y2 := /articles/article/publication_date/@year
(13)  let $y3 := /articles/hikes/article/publication_date/year
(14)  let $y4 := /articles/hikes/non_refereed/article/publication_date/@year

(15)  let $p1 := /articles/article/@refereed[data("yes")]
(16)  let $p2 := /articles/article/@article_type
(17)  let $p3 := /articles/hikes/article/publication_date/article_type
(18)  let $p4 := /articles/hikes/non_refereed

(19) return

(20) <results>

(21)  author = {data($a1)}, title = {data($t1)}, year = {data($y1)}, publication_type = {name($p1)} ;
(22)  author = {data($a2)}, title = {data($t2)}, year = {data($y2)}, publication_type = {data($p2)} ;
(23)  author = {name($a3)}, title = {data($t3)}, year = {data($y3)}, publication_type = {data($p3)} ;
(24)  author = {name($a4)}, title = {data($t4)}, year = {data($y4)}, publication_type = {name($p4)}

(25) </results>
```

Vastaus:

```
(26) <results>

(27)  author = smith, title = art1, year = 2002, publication_type = refereed ;
(28)  author = jones, title = art2, year = 2003, publication_type = non_refereed ;
(29)  author = hikes, title = art3, year = 2004, publication_type = refereed ;
(30)  author = hikes, title = art4, year = 2005, publication_type = non_refereed

(31) </results>
```

Kuva 4.3: Kyselyesimerkki XQuery-kyselykielellä suoritettuna.

Kysely:

```
(1)  {author(A),title(T),year(Y),P(yes)} is_component_of article in 'articles.xml';
(2)  {author(A),title(T),year(Y),article_type(P)} is_component_of article in 'articles.xml';
(3)  {article,title(T),year(Y),article_type(P)} is_component_of A in 'articles.xml';
(4)  {article,title(T),year(Y)} is_component_of P in 'articles.xml',
(5)  P is_component_of A in 'articles.xml'.
```

Vastaus:

```
(6)  A = smith,
(7)  T = art1,
(8)  Y = 2002,
(9)  P = refereed ;

(10) A = jones,
(11) T = art2,
(12) Y = 2003,
(13) P = non_refereed ;

(14) A = hikes,
(15) T = art3,
(16) Y = 2004,
(17) P = refereed ;

(18) P = non_refereed,
(19) T = art4,
(20) Y = 2005,
(21) A = hikes
```

Kuva 4.4: Kyselyesimerkki `is_component_of`-primitiivillä suoritettuna.

Luku 5

Tietokuution konstruoiminen XML-dokumenteista

5.1 Järjestelmäprototyypin toiminta

5.1.1 Kyselyn rakenne

Tässä luvussa esitellään tutkielman yhteydessä kehitetty järjestelmäprototyyppi, jonka avulla loppukäyttäjän on mahdollista konstruoida XML-dokumenteissa olevista tiedoista tietokuutioita. Järjestelmäprototyyppi on toteutettu Prolog-ohjelmointikielellä, ja siinä on siten käytössä logiikkaohjelmoinnin ja deduktiivisten tietokantojen muuttujakäsite (ks. esim. [SS94], [Liu99]). Logiikkaohjelmoinnin ja deduktiivisten tietokantojen muuttujakäsite poikkeaa proseduraalisten ohjelmointikielten muuttujakäsitteestä siten, että siinä ei ole hävittävää sijoitusta. Lisäksi järjestelmäprototyyppiin tehtävissä kyselyissä käytetään logiikkaohjelmoinnista omaksuttua tapaa merkitä atomiset ilmaukset pienellä ja muuttujat suurella alkukirjaimella.

Käyttäjän järjestelmälle antama tietokuution konstruointi-ilmaus on kaksiosainen ja sen yleinen rakenne on annettu määritelmässä 5.1.

Määritelmä 5.1:

`CREATE CUBE` <*tietokuution määrittely*>
`WHERE` <*XML-rakenteiden osoittaminen*>

Tietokuution konstruointi-ilmauksen ensimmäisessä osassa, `CREATE CUBE` -lauseessa käyttäjä määrittelee muodostettavan tietokuution rakenteen. Ilmauksen toisessa osassa, `WHERE`-lauseessa käyttäjä määrittelee puolestaan mallin joukolle `is_component_of`-ilmauksia, joiden avulla käytettävissä olevista XML-dokumenteista saadaan poimittua relevantteja tietoja. Käyttäjän antaman mallin perusteella järjestelmäprototyyppi generoi tietokuution muodostamisessa tarvittavat `is_component_of`-ilmaukset. Tietokuution määrittelyn yhteydessä annetaan `CREATE CUBE` -lauseessa myös joukko muuttujia, joiden ja `WHERE`-lauseessa annettujen, niitä nimiltään vastaavien muuttujien avulla kiinnitetään

XML-dokumenteista poimittavat tiedot muodostettavan tietokuution attribuuttien arvoihin. Tietokuution konstruointi-ilmauksen rakenneosia käsitellään seuraavaksi tarkemmin. Konstruointi-ilmauksen BNF-kieliopin mukainen määrittely on annettu liitteessä A.

Tietokuution määritteleminen merkitsee toteutetussa järjestelmäprototyypissä sitä, että käyttäjä nimeää tietokuution ja sen dimensio- ja mitta-attribuutit sekä liittää dimensio- ja mitta-attribuuttien yhteyteen tarvittavat muuttujat. Tarkastellaan järjestelmäprototyypissä tehtävää tietokuution määrittelyä esimerkin avulla. Ajatellaan, että halutaan määritellä tietokuutio, joka sisältää kirjoittaja-, tieteenala- ja julkaisuvuosikohtaisesti ryhmiteltyinä tietoja julkaistujen tieteellisten artikkelien kokonaismääristä. Olkoon muodostettavan tietokuution nimi `article_cube` ja nimettäköön sen attribuutit vastaavasti `author`, `domain`, `year` ja `no_of_articles`. Näistä `author`, `domain` ja `year` ovat dimensioattribuutteja ja `no_of_articles` mitta-attribuutti. Vastaava tietokuution määrittely on esitetty esimerkissä 5.2.

Esimerkki 5.2:

```
CREATE CUBE   article_cube(dim(author,A),dim(domain,D),dim(year,Y),
              mes(no_of_articles,N,count))
```

Tietokuution määrittely alkaa varatuilla sanoilla `CREATE CUBE`, joita seuraa, välilyönnillä erotettuna, tietokuution nimi ja sulkeiden (`(`,`)`) sisällä, pilkulla (`,`) toisistaan erotettuina tietokuution dimensio- ja mitta-attribuuttien määrittelyt. Tietokuution määrittelyssä käyttäjä voi antaa tietokuutiolle haluamansa määrän dimensio- ja mitta-attribuutteja. Käyttäjä voi nimetä tietokuution ja sen dimensio- ja mitta-attribuutit sekä tietokuution määrittelyssä käytettävät muuttujat haluamallaan tavalla, mutta nimien pitää kuitenkin olla yksikäsitteiset. Lisäksi tietokuution ja sen attribuuttien nimien pitää alkaa pienellä kirjaimella ja muuttujien isolla kirjaimella.

Dimensioattribuutin määrittely alkaa varatulla sanalla `dim`, jota seuraa sulkeiden sisällä, pilkulla toisistaan erotettuina dimensioattribuutin nimi ja muuttuja, jonka alustukset tulevat `WHERE`-lauseessa liitetyiksi XML-dokumenteissa oleviin tietoihin. Dimensioattribuutin määrittelyyn voidaan liittää myös useampia muuttujia ja niiden avulla on mahdollista esittää dimensioattribuutin yhteydessä sen muitakin ominaisuuksia. Viimeksi mainittu mahdollisuus johtuu siitä, että järjestelmäprototyypin avulla pyritään tyydyttämään myös ad hoc -tyyppisiä tietotarpeita, jolloin ei aina ole kannattavaa muodostaa erillisiä dimensio- ja faktatauluja esimerkiksi tähtimallin tapaan, vaan kaikki tietotarpeen kannalta relevantti informaatio on pyrittävä esittämään kompaktissa muodossa.

Mitta-attribuutin määrittely alkaa varatulla sanalla `mes`, jota seuraa – samalla tavoin kuin dimensioattribuutteja määriteltäessä – sulkeiden sisällä, pilkulla toisistaan erotettuina mitta-attribuutin nimi ja muuttuja. Mitta-attribuuttiin voidaan liittää ainoastaan yksi muuttuja. Mitta-attribuutin määrittelyn yhteydessä annetaan myös sen aggregointifunktion nimi, jota mitta-attribuutin arvojen laskemisessa sovelletaan. Järjestelmäprototyypissä mahdolliset aggregointifunktiot ovat minimi- ja maksimiarvo, absoluuttinen frekvenssi, aritmeettinen summa ja aritmeettinen keskiarvo. Mitta-attribuutin arvon las-

kennassa käytettävä aggregointifunktio ilmaistaan kirjoittamalla mitta-attribuutin määrittelyssä annettavan muuttujan jälkeen käytettävän aggregointifunktion tunnus eli jokin atomeista `min`, `max`, `count`, `sum` tai `avg`.

Tietojen poimimiseen XML-dokumenteista `WHERE`-lauseessa käytetään `is_component_of`-kyselyprimitiiviä. Järjestelmäprototyypissä on lähdetty siitä oletuksesta, että käyttäjä tuntee käytettävissä olevien XML-dokumenttien sisällön (mitä tietoja XML-dokumentit sisältävät ja mitkä ovat tietojen mahdolliset organisointiperiaatteet) mutta ei yksittäisen XML-dokumentin tarkkaa rakennetta (miten tiedot yksittäisessä dokumentissa on organisoitu). Tästä johtuen on luonnollista ajatella, että konstruointi-ilmauksen `WHERE`-lauseessa käyttäjä antaa joukon `is_component_of`-ilmauksia, joista kukin vastaa yhtä tai useampaa XML-dokumenteissa mahdollista tietojen organisointitapaa. Seurauksena tästä kuitenkin on, että käyttäjän kirjoitettavaksi tulevien `is_component_of`-ilmausten määrä nousee helposti suureksi.

Käyttäjän tehtävää on järjestelmäprototyypissä kuitenkin helpotettu siten, että käyttäjän ei itse tarvitse kirjoittaa kaikkia vaihtoehtoisia `is_component_of`-ilmauksia, vaan järjestelmäprototyyppi generoi tietokuution konstruoinnissa tarvittavat `is_component_of`-ilmaukset käyttäjän antaman mallin pohjalta. Käyttäjän tietokuution konstruointi-ilmauksen `WHERE`-lauseessa antama malli koostuu joukosta `is_component_of`-ilmauksen vasemmalla puolella (ns. komponenttiosassa) mahdollisista komponentti-ilmauksista ja joukosta `is_component_of`-ilmauksen oikealla puolella (ns. elementtiosassa) mahdollisista komponentti-ilmauksista sekä `is_component_of`-ilmauksesta, johon käyttäjä on sijoittanut komponentti-ilmauksien joukkoja vastaavat muuttujat.

Havainnollistetaan tilannetta laatimalla esimerkin 5.2 `CREATE CUBE` -lauseeseen liittyvä `WHERE`-lause. Oletetaan, että käyttäjä tietää, että tietokuution muodostamisessa käytössä olevissa XML-dokumenteissa artikkeleita voidaan ryhmitellä artikkelin, artikkelin kirjoittajan, tutkimusalan ja artikkelin julkaisuvuoden lisäksi artikkelin tyypin (`type`, `db` tai `ir`) mukaan. Näiden tietojen perusteella muodostettava `WHERE`-lause on annettu esimerkissä 5.3. Saadussa `WHERE`-lauseessa on annettu kaksi komponentti-ilmauksia sisältävää joukkoa, joukot `X` ja `Y`, ja `is_component_of`-ilmaus, joka selittää joukkoihin kuuluvien komponentti-ilmausten aseman generoitavissa `is_component_of`-ilmauksissa. Joukkoon `X` kuuluvat komponentti-ilmaukset voivat esiintyä `is_component_of`-ilmauksen vasemmalla puolella (ns. komponenttiosassa) ja joukkoon `Y` kuuluvat komponentti-ilmaukset `is_component_of`-ilmauksen oikealla puolella (ns. elementtiosassa).

Esimerkki 5.3:

```
(1)  WHERE  X =  {author(A),domain(D),year(Y),article(N)},
(2)         Y =  {author(A),domain(D),year(Y),article(N),author,domain,
                  year,type,D = db,D = ir},
(3)         X is_component_of Y in Doc
```

Tietokuution konstruointi-ilmauksen `WHERE`-lause alkaa varatulla sanalla `WHERE`, jota seuraa, siitä välilyönnillä erotettuna muuttuja, jota edelleen seuraa yhtäsuuruusmerkki

(=) ja aaltosulkeiden ({,}) välissä, pilkulla (,) toisistaan erotettuina, (ei-tyhjä) joukko komponentti-ilmauksia. Ensimmäistä komponentti-ilmausten joukon määrittelyä seuraa siitä pilkulla erotettuna toinen komponentti-ilmausten joukon määrittely, joka alkaa vastaavasti muuttujalla (jonka on oltava eri kuin ensimmäisessä komponentti-ilmausten joukon määrittelyn yhteydessä annettu muuttuja), jota seuraa yhtäsuuruusmerkki ja aaltosulkeiden välissä, pilkulla toisistaan erotettuina (ei-tyhjä) joukko komponentti-ilmauksia. Toinenkin komponentti-ilmausten joukon määrittely päättyy pilkkuun, jota seuraa `is_component_of`-ilmaus, jossa komponentti-ilmausten joukkojen määrittelyjen yhteydessä annetut muuttujat sijoitetaan `is_component_of`-ilmauksen komponentti- ja elementtiosaan. Jos yksi tällainen edellä kuvattu `is_component_of`-ilmausten määrittely ei ole relevanttien tietojen poiminnan kannalta riittävä, voi käyttäjä kirjoittaa `WHERE`-lauseessa puolipisteellä (;) toisistaan erotettuina useampia `is_component_of`-ilmauksien määrittelyjä.

Tarkastellaan nyt sitä, miten järjestelmäprototyyppi generoi käyttäjän antaman mallin perusteella `WHERE`-lauseessa `is_component_of`-ilmaukset. Järjestelmän muodostamien `is_component_of`-ilmausten generoinnissa noudatetaan seuraavia neljää sääntöä:

1. Yksittäisessä `is_component_of`-ilmauksessa tulee esiintyä kaikki `CREATE CUBE` -lauseessa tietokuution määrittelyssä esitellyt muuttujat.

2. Kaikkien komponenttiosan komponentti-ilmauksissa olevien erillisten muuttujien tulee esiintyä aina jokaisessa `is_component_of`-ilmauksessa.

3. Kaikkien komponenttiosan komponentti-ilmauksissa olevien muuttujien tulee esiintyä jokaisessa `is_component_of`-ilmauksessa täsmälleen yhden kerran.

4. Jos elementtiosan komponentti-ilmauksessa on muuttuja, joka esiintyy myös komponenttiosassa olevassa komponentti-ilmauksessa, ei komponenttiosaan kuuluvaa komponentti-ilmausta oteta mukaan `is_component_of`-ilmaukseen.

Edellä esitettyjä periaatteita noudattamalla järjestelmäprototyyppi generoi esimerkissä 5.3 annetun mallin perusteella kymmenen `is_component_of`-ilmausta, jotka on esitetty esimerkissä 5.4.

Esimerkki 5.4:

```
(1)  {author(A),domain(D),year(Y)} is_component_of article(N) in Doc;
(2)  {domain(D),year(Y),article(N)} is_component_of author(A) in Doc;
(3)  {author(A),year(Y),article(N)} is_component_of domain(D) in Doc;
(4)  {author(A),domain(D),article(N)} is_component_of year(Y) in Doc;
(5)  {author(A),domain(D),year(Y),article(N)} is_component_of author in Doc;
(6)  {author(A),domain(D),year(Y),article(N)} is_component_of domain in Doc;
(7)  {author(A),domain(D),year(Y),article(N)} is_component_of year in Doc;
(8)  {author(A),domain(D),year(Y),article(N)} is_component_of type in Doc;
(9)  {author(A),year(Y),article(N)} is_component_of db in Doc;
(10) {author(A),year(Y),article(N)} is_component_of ir in Doc;
```

5.1.2 Kyselyn prosessointi

Seuraavaksi käsitellään lyhyesti käyttäjän ja järjestelmäprototyypin välillä tietokuution muodostamisen aikana tapahtuvaa vuorovaikutusta. Tietokuution muodostamisprosessi alkaa siitä, että käyttäjä avaa ja käynnistää sovelluksen (käytännössä tämä tarkoittaa yhden Prolog-tiedoston kääntämistä ja siihen sisältyvän faktan evaluoinnin aloittamista). Kun sovellus on käynnistetty, antaa järjestelmä käyttäjälle kehotteen, jossa käyttäjää pyydetään kirjoittamaan tietokuution konstruointi-ilmaus, jossa annettujen tietojen perusteella järjestelmä muodostaa halutun tietokuution. Käyttäjän antaman tietokuution konstruointi-ilmauksen tulee noudattaa edellisessä alaluvussa sen rakenteesta annettuja määräyksiä. Ennen konstruointi-ilmauksen varsinaisen evaluoinnin aloittamista järjestelmä tarkastaa, että annettu konstruointi-ilmaus on syntaktisesti oikein. Jos näin ei ole, annetaan virheilmoitus ja pyydetään käyttäjää antamaan konstruointi-ilmaus uudestaan. Uudelle konstruointi-ilmaukselle suoritetaan jälleen syntaksin tarkastus, ja jos uusikaan konstruointi-ilmaus ei ole syntaksiltaan oikein, toistetaan edellä kuvattua prosessia, kunnes käyttäjä antaa syntaksiltaan oikean konstruointi-ilmauksen tai keskeyttää ohjelman. Jos konstruointi-ilmaus on syntaktisesti oikein, aloitetaan tietokuution muodostaminen.

Kun tietokuutio on onnistuneesti muodostettu, avaa järjestelmä tiedoston, jonne muodostettu tietokuutio on tallennettu XML-muotoisena (XML-tiedostolla on sama nimi kuin tietokuutiollakin). Tietokuution muodostamisen jälkeen käyttäjä voi muodostaa uuden tietokuution, editoida juuri muodostettua tietokuutiota tai sulkea sovelluksen. Jos käyttäjä päättää muodostaa uuden tietokuution, on hänellä käytössään edellisessä kyselyssä muodostamansa tietokuutio. Tietokuution editoiminen tarkoittaa käytännössä sitä, että käyttäjä pääsee tarkastelemaan välitulosta, jonka perusteella kuution mitta-attribuuttien laskeminen on suoritettu, ja poistamaan siinä mahdollisesti olevia virheellisiä tietoja. Käyttäjän välitulokseen tekemien muutosten jälkeen lasketaan tietokuution mitta-attribuuttien arvot uudelleen.

5.2 Kyselyn tuloksen esittäminen

Järjestelmän konstruoima tietokuutio esitetään siis XML-dokumenttina. Tietokuution XML-esityksessä dokumentin juurielementtinä on `datacube`-niminen elementti, jolla on `name`-niminen attribuutti, joka saa arvokseen tietokuution nimen. Juurielementin sisällä on joukko `tuple`-nimisiä elementtejä, jotka sisältävät kaikki dimensioattribuuttien arvojen kombinaatiot ja niihin liittyvät mitta-attribuuttien arvot. Dimensioattribuutit esitetään `dimension`-nimisinä elementteinä, joilla on attribuutti `name`, joka saa arvokseen dimensioattribuutin nimen. Dimensioattribuutin arvo on `dimension`-elementin sisältöosassa. Vastaavasti mitta-attribuutit esitetään `measure`-nimisten elementtien avulla, joilla on `name`-niminen attribuutti, jonka arvo on mitta-attribuutin nimi. Mitta-attribuutin arvo esitetään `measure`-elementin sisältö-osassa.

5.3 Esimerkkiympäristö ja esimerkkikyselyt

5.3.1 Sovellusalue

Edellisissä luvuissa esiteltyä uutta lähestymistapaa XML-tietojen integrointiin sovelletaan tässä luvussa informetriikkaan (informetrics). Informetriikka on bibliografista informaatiota hyväksi käyttäen kirjallisuuteen ja julkaisutoimintaan liittyviä tilastollisia ilmiöitä tutkiva tiede. Informetriikassa tutkittaviin tilastotieteellisiin ilmiöihin kuuluvat mm. kirjoittaja-, maa- ja/tai julkaisukohtaiset tuottavuusluvut, julkaisujen tai kirjoittajien arvottamisessa käytettävät *generalized impact factor* -tunnusluvut (julkaisussa i ajanhetkellä T_1 julkaistuissa artikkeleissa julkaisussa j ajanhetkellä T_2 julkaistuihin artikkeleihin kohdistettujen viittausten lukumäärä suhteessa julkaisussa j ajanhetkeen T_2 mennessä julkaistujen artikkelien kokonaismäärään), kirjoittajien, organisaatioiden tai julkaisujen aktiviteettiprofiilit, siteerausverkostot sekä kirjallisuuden kasvu ja ikääntyminen. Informetriikassa käytettävä tutkimusaineisto on nykyisin yleensä saatavilla Internetin kautta. On osoitettu, että moniulotteinen analyysi soveltuu hyvin informetrisen aineiston käsittelyyn [NHJ03].

5.3.2 Esimerkkidokumentit

Valitun sovellusalueen perusteella on laadittu joukko XML-muotoisia esimerkkidokumentteja, joiden sisältämiä tietoja on mahdollista käyttää informetrisessa analyysissa. Esimerkkidokumentit on laadittu siten, että samaa tarkoittavat elementit ja attribuutit on kaikissa esimerkkikokoelman dokumenteissa nimetty samalla tavoin. Tämä siis tarkoittaa sitä, että esimerkkidokumenteissa ei esiinny semanttista heterogeenisuutta.

Rakenteellista heterogeenisuutta esimerkkidokumenteissa sen sijaan esiintyy. Esimerkkidokumenttien rakenteellinen heterogeenisuus ilmenee siten, että samaa tarkoittava tieto voi – niin eri dokumenteissa kuin saman dokumentin sisälläkin – esiintyä elementin nimenä tai arvona ja attribuutin nimenä. Lisäksi elementit voivat olla keskenään mielivaltaisella tavalla ryhmiteltyjä.

Esimerkkidokumentit voidaan jakaa kolmeen tyyppiin. Tyypitysperusteena on käytetty dokumenttien sisältöä: samaa tyyppiä edustavat dokumentit sisältävät samankaltaista informaatiota. Vaikka samaan tyyppiin kuuluvilla dokumenteilla onkin samankaltainen tietosisältö, esitetään samaa tarkoittava tieto yksittäisissä dokumenteissa rakenteellisesti erilaisilla tavoilla.

Esimerkkidokumenttien määrästä johtuen niitä ei voida sellaisenaan esittää tämän tutkielman yhteydessä. Esimerkkidokumenttien tietosisällöt on esitetty tiivistetysti liitteen B taulukoissa 1 - 3. Kaikki esimerkkidokumentit ovat hyvin-määriteltyjä XML-dokumentteja. Seuraavassa kuvaillaan lyhyesti, minkä nimisiä elementtejä ja attribuutteja kuhunkin tyyppiin kuuluvassa yksittäisessä XML-dokumentissa voi esiintyä ja miten kuvattuja elementtejä ja attribuutteja voidaan esimerkkidokumentteihin sijoittaa.

- Tyyppiin `publishing` kuuluvat esimerkkidokumentit (4 kpl) sisältävät tietoja eri kustantajien julkaisemista artikkeleista. Artikkeleita voidaan ryhmitellä kirjoittajan (`author`), julkaisuvuoden (`year`), tutkimusalan (`domain`) tai tyypin (`type`; tie-

to siitä, onko artikkeli käynyt läpi arviointimenettelyn vai ei, mitä vastaavat arvot `refereed` ja `non_refereed`) mukaan tai ryhmitellä vapaasti. Yksittäisen dokumentin sisällä voi esiintyä rinnakkain kaikkia mainittuja ryhmittelyvaihtoehtoja. Tyyppiin publishing kuuluvien esimerkkidokumenttien tietosisältö on esitetty liitteen B taulukossa 1.

- Tyyppiin `institution` kuuluvat dokumentit (7 kpl) sisältävät tietoja organisaatioiden (`name`) eri henkilöille myöntämistä tutkimusapurahoista (`grant`). Dokumenteissa apurahat on voitu ryhmitellä apurahan suuruuden (`amount`), myöntämisajankohdan (`date`, `year`), tutkimusalan (`domain`) tai saajan (`grantee`) mukaan tai jättää kokonaan ryhmittelemättä. Yksittäisen dokumentin sisällä on voitu käyttää rinnakkain kaikkia mainittuja ryhmittelytapoja. Tyyppiin institution kuuluvien esimerkkidokumenttien tietosisältö on esitetty liitteen B taulukossa 2.

- Lopuksi tyyppiä `article` olevat dokumentit (25 kpl) edustavat varsinaisia artikkeleita. Tämän tyypin dokumenteissa ei ole varsinaista asiasisältöä, vaan ne sisältävät ainoastaan tiedot artikkelin tekijästä tai tekijöistä (`author`), artikkelin otsikon (`title`), artikkelin julkaisuvuoden (`year`), artikkelin kustantajan nimen (`publisher`) sekä viittaukset artikkelissa käytettyihin lähteisiin (`reference`). Jokaisen viitteen kohdalta on annettu kirjoittaja (`ref_author`), otsikko (`ref_title`), julkaisuvuosi (`ref_year`) ja kustantaja (`ref_publisher`). Viitetiedoissa esiintyy jälleen rakenteellista heterogeenisuutta. Tyyppiin article kuuluvien esimerkkidokumenttien tietosisältö (julkaisuvuosi- ja kustantajainformaatiota lukuun ottamatta) on esitetty liitteen B taulukossa 3.

Seuraavaksi tarkasteltavissa esimerkkikyselyissä on lähtökohtana, että kyselyn tekijä tietää XML-dokumenttien semantiikan mutta ei yksittäisten dokumenttien tarkkaa rakennetta. Esimerkkikyselyjen vastausten tiivistelmät on esitetty liitteen C taulukoissa 4 - 6.

5.3.3 Esimerkkikyselyt

Esimerkkikyselyssä 1 halutaan tuottaa järjestelmäprototyypin avulla yksinkertainen tietokuutio, jossa on esitetty artikkelien lukumäärät julkaisuvuoden, kirjoittajan ja artikkelin tyypin mukaan ryhmiteltyinä. Muodostettavan tietokuution attribuuttien ja esimerkkidokumenttien yhteydessä annetun informaation nojalla on selvää, että tietokuutio kannattaa muodostaa `publishing`-tyyppisissä dokumenteissa olevien tietojen perusteella.

Annetaan muodostettavalle tietokuutiolle nimeksi `article_cube` ja sen dimensioattribuuteille nimiksi `year`, `author` ja `type` ja ainoalle mitta-attribuutille `number_of_articles`. Mitta-attribuutin arvojen muodostamisessa käytetään aggregaatiofunktiota `count`. Tietokuution muodostamisessa tarvittavissa `is_component_of`-ilmauksissa pitää etsittävinä komponentteina olla ainakin `year`-, `author`- ja `type`-nimiset komponentit. Julkaisut on lisäksi mahdollista ryhmitellä niiden tyypin mukaan, joten myös komponenttinimet `refereed` ja `non_refereed` on otettava mukaan. Kuten jo edellä on todettu, kerätään tietokuution muodostamisessa tarvittavaa informaatiota

publishing-tyyppisistä dokumenteista, ja siten is_component_of-ilmausten yläelementiksi on luontevaa valita publishing-niminen elementti. Esimerkkikysely 1 on kokonaisuudessaan seuraavanlainen:

Esimerkkikysely 1:

```
(1)  CREATE CUBE   article_cube(dim(year,Y),dim(author,A),dim(type,T),
                   mes(number_of_articles,N,count))
(2)  WHERE         C = {author(A),year(Y),article(N),type(T),
                   T = refereed,T = non_refereed},
(3)                S = {publishing},
(4)                C is_component_of S in X.
```

Kyselyn WHERE-lauseessa joukoissa C ja S annetuista komponentti-ilmauksista muodostetaan seuraavat kolme is_component_of-ilmausta:

```
(1)  {author(A),year(Y),article(N),type(T)} is_component_of publishing in X.
(2)  {author(A),year(Y),article(N), refereed} is_component_of publishing in X.
(3)  {author(A),year(Y),article(N), non_refereed} is_component_of publishing in X.
```

Edellä esitettyjen kolmen is_component_of-ilmauksen avulla on mahdollista esittää kaikki halutun tietokuution muodostamisessa tarvittava informaatio publishing-tyyppisistä esimerkkidokumenteista. Esimerkkikyselyn 1 vastauksen tiivistelmä on esitetty liitteen C taulukossa 4.

Esimerkkikyselyssä 2 halutaan tuottaa tietokuutio, joka sisältää tiedot myönnettyjen apurahojen keskimääräisestä suuruudesta, myönnettyjen apurahojen yhteissummasta sekä pienimmästä ja suurimmasta myönnetystä apurahasta ryhmiteltynä vuosi- ja organisaatio- ja tieteenalakohtaisesti. Tuotettavan tietokuution attribuuttien perusteella on selvää, että tietokuutio muodostetaan institution-tyyppisissä esimerkkidokumenteissa olevien tietojen perusteella.

Annetaan muodostettavalle tietokuutiolle nimeksi grant_cube ja sen dimensioattribuuteille year, institution ja domain ja mitta-attribuuteille avg_grants, sum_grants, min_grant ja max_grant. Mitta-attribuuttien arvojen laskemisessa käytetään aggregaatiofunktioita avg, sum, min ja max. Tietokuution muodostamisessa käytettävissä is_component_of-lauseissa pitää poimia tietoja ainakin year-, name-, domain- ja amount-nimisistä komponenteista. Tarvittavia komponentteja on luontevaa etsiä institution-nimisen elementin alaisuudesta. Esimerkkikysely 2 on kokonaisuudessaan seuraavanlainen:

Esimerkkikysely 2:

```
(1)  CREATE CUBE   grant_cube(dim(year,Y),dim(institution,I),dim(domain,D),
                   mes(avg_grants,AVG,avg),mes(sum_grants,SUM,sum),
                   mes(min_grant,MIN,min),mes(max_grant,MAX,max))
(2)  WHERE         C = {domain(D),amount(AVG),amount(SUM),amount(MIN),
                   amount(MAX),year(Y),name(I)},
(3)                S = {institution},
(4)                C is_component_of S in X.
```

Kyselyn `WHERE`-lauseen informaation perusteella järjestelmäprototyyppi konstruoi yhden `is_component_of`-ilmauksen, jolla kyetään `institution`-tyyppisistä esimerkkidokumenteista keräämään kaikki tietokuution muodostamisessa tarvittava informaatio. Esimerkkikyselyn 2 vastauksen tiivistelmä on esitetty liitteen C taulukossa 5.

Esimerkkikyselyssä 3 halutaan muodostaa tietokuutio, jossa on annettu kustantajan kustantamiin artikkeleihin kohdistuneiden viittausten lukumäärä viittausvuoden ja kustantajan mukaan ryhmiteltyinä. Tietokuutio muodostetaan nyt `article`-tyyppisissä esimerkkidokumenteissa olevien tietojen perusteella.

Annetaan muodostettavalle tietokuutiolle nimeksi `reference_cube` ja sen dimensioattribuuteille `year` ja `publisher` ja mitta-attribuutille `number_of_refs`. Mitta-attribuutin arvojen laskentaan käytetään jälleen aggregaatiofunktiota `count`. Valitaan `is_component_of`-ilmauksissa käytettäväksi yläelementiksi `article` ja etsittäviksi komponenteiksi `year`, `ref_publisher` ja `reference`. Esimerkkikysely 3 on kokonaisuudessaan seuraava:

Esimerkkikysely 3:

```
(1)  CREATE CUBE   reference_cube(dim(year,Y),dim(publisher,P),
                   mes(number_of_refs,R,count))
(2)  WHERE         C = {ref_publisher(P),reference(R),year(Y)},
(3)                S = {article},
(4)                C is_component_of S in X.
```

Kyselyn `WHERE`-lauseessa muodostetaan jälleen yksi `is_component_of`-ilmaus, jolla pystytään poimimaan kaikki tietokuution muodostamisen kannalta tarpeellinen informaatio. Esimerkkikyselyn 3 vastauksen tiivistelmä on esitetty liitteen C taulukossa 6.

Luku 6

Tulosten tarkastelu

6.1 Kehitetyn lähestymistavan arvioiminen

Kehitetty `is_component_of`-kyselyprimitiivi tarjoaa joustavan keinon poimia tietoja rakenteeltaan heterogeenisista XML-dokumenteista ilman, että käyttäjän tarvitsee etukäteen tietää kyselyssä käytettävien lähdedokumenttien tarkkaa rakennetta. Riittää, kun käyttäjä tuntee lähdedokumenttien tietosisällön tai sen osan. Lisäksi yksittäisen `is_component_of`-kyselyn avulla pystytään yleensä poimimaan relevantteja tietoja useammasta XML-dokumentista, kun taas esimerkiksi XQuery-kysely on aina sidottu koskemaan vain yhtä dokumenttia. Kehitetty kyselyprimitiivi tarjoaa lisäksi deklaratiivisen tavan suorittaa kyselyitä; so. kyselyn tekijän ei tarvitse määritellä yksityiskohtaisesti, miten tiedot yksittäisistä dokumenteista poimitaan, tai välittää kyselyn prosessoinnista.

Kehitetty `is_component_of`-kyselyprimitiivi ei pysty käsittelemään XML-dokumentteihin liittyvää semanttista heterogeenisuutta. Jos lähtödokumenttien samaa tarkoittavat elementit ja attribuutit on nimetty erilailla, joudutaan – XQuery-kyselyiden tavoin – jokaista nimikombinaatiota kohti laatimaan oma `is_component_of`-kyselynsä. XML-dokumenteissa oleva semanttinen heterogeenisuus on rakenteelliseen heterogeenisuuteen merkitykseltään ja yleisyydeltään verrattavissa oleva ongelma, joka on otettava jollakin tavalla huomioon, jotta XML-muotoisia tietoja voidaan laajamittaisesti integroida. Semanttisen heterogeenisuuden ongelman ratkaiseminen onkin `is_component_of`-kyselyprimitiivin jatkokehityksen prioriteetti.

Koska kyselyprimitiivissä `is_component_of` sovelletaan XML-muotoisten tietojen poimintaan tiedonhakutyyppistä lähestymistapaa, jossa käyttäjän ei oleteta tietävän mitään lähdedokumenttien rakenteesta, on kyselyntekijällä mahdollisesti olevaa, integroinnissa käytettävien dokumenttien rakennetta koskevaa tietämystä vaikea hyödyntää `is_component_of`-primitiivin avulla tehtävissä kyselyissä. Esimerkiksi Li, Yu ja Jagadish [LYJ04] ovat kehittäneet lähestymistavan, jossaa XQuery-kyselyllä saadaan joustavasti poimittua tietoja XML-dokumenteista riippuen siitä, millainen on kyselyn tekijän tietämys lähdedokumenttien rakenteesta.

Kyselyprimitiivin `is_component_of` avulla tehty kysely tuottaa tulokseksi muuttujien alustuksia eikä esimerkiksi XML-rakenteita. Kyselyprimitiivi ei näin ollen tarjoa

käyttäjälle suoraan keinoa muodostaa kyselyn tuloksesta esimerkiksi XML-dokumenttia kuten esimerkiksi XQuery-kysely. Toisaalta se, että `is_component_of`-kyselyn tulosta ei esitetä missään ennalta määrätyssä muodossa, kuitenkin mahdollistaa `is_component_of`-primitiivin esimerkiksi XQueryä laajempialaisen soveltamisen, sillä kyselyn tulos voidaan tarvittaessa helposti muuntaa esimerkiksi XML-muotoon (kuten järjestelmäprototyypin tapauksessa).

Muutoin periaatteessa integroitavissa olevan aineiston integroiminen ei ole mahdollista tiedonhakutyyppistä tietojen etsintää ja poimintaa noudattavissa XML-kyselykielissä, kuten `is_component_of`-primitiivi, silloin, kun integroitavaan aineistoon kuuluvassa yksittäisessä XML-dokumentissa esiintyy samannimisiä elementtejä ja/tai attribuutteja, jotka kuitenkin esittävät eri asioita. Tällaisissa tilanteissa ei ole mahdollista päätellä laskennallisesti pelkän dokumentin rakenteen perusteella, mitkä komponentit kuuluvat yhteen, vaan yhteenkuuluvat komponentit pitää osoittaa eksplisiittisesti, esimerkiksi polkuilmausten avulla.

6.2 Tutkimusaiheeseen liittyvä kirjallisuus

XML-merkintäkielen hyödyntämistä OLAP-toiminnallisuuden yhteydessä on toistaiseksi tutkittu vain vähän. Seuraavassa esitellään joitakin tutkielman viitekehyksen kannalta kiinnostavia tutkimuksia.

Golfarelli, Rizzi ja Vrdoljak [GRV01] näkevät, että tulevaisuudessa tietovarastoissa käytetään materiaalina yhä enemmän XML-dokumentteja, ja he ovat kehittäneet puoli-automaattisen lähestymistavan tuottaa tietovarastoa kuvaava käsitekaavio suoraan XML-dokumenttien perusteella. Kirjoittajien mukaan kaikissa tietovarastojen kuvaamiseen kehitetyissä käsitteellisissä malleissa on painotettu funktionaalisia riippuvuuksia eli attribuuttien välisiä $n : 1$ -suhteita. Näin tulee olla myös XML-pohjaisessa tietovarastojen suunnittelussa. Golfarellillä, Rizzillä ja Vrdoljakilla on lähtökohtana, että XML-merkintäkielestä saadaan suurin hyöty silloin, kun jokaiseen käsiteltävään XML-dokumenttiin liittyy DTD-kielioppi (tai XML Schema -kuvaus). Niiden avulla voidaan myös esittää XML-dokumenttien tieto-objektien välisiä suhteita. Kuitenkaan niissä esitetyt suhteet eivät ole useinkaan yksiselitteisiä. Kirjoittajat esittävät interaktiivisen algoritmin, jonka avulla – tarvittaessa tietovaraston suunnittelijaa konsultoiden – DTD-kielioppia voidaan yksinkertaistaa niin, että siinä esitetyt suhteet tulevat yksiselitteisiksi. Näin saatavaa DTD-kielioppia voidaan sitten käyttää XML-pohjaisen tietovaraston käsitekaaviona. Olemassa olevat XML-dokumentit voidaan muuntaa uuden DTD-kuvauksen mukaisiksi ja tietovarastoon tuotavat uudet XML-dokumentit validoidaan DTD-kieliopin suhteen ja tarvittaessa muunnetaan rakenteeltaan sitä vastaaviksi.

Hümmer, Bauer ja Harde [HBH03] esittelevät XML-dokumenttipohjien standardin, joiden avulla jo olemassa olevien tietovarastojen tiedot on mahdollista esittää XML-muodossa siten, että niitä voidaan helposti siirtää ja kysellä tietoverkkojen kautta. Kirjoittajien lähtökohta on, että tietojen vaihtelevasta laadusta johtuen esimerkiksi Internetissä olevia resursseja ei ole kannattavaa integroida suoraan tietovarastoon, mutta toisaalta sitä vastoin on luonnollista integroida keskenään tietovarastoja, sillä niissä oleva

data on yleensä käynyt läpi jonkin asteisen laadunvarmistuksen ja se on myös esikäsiteltyä. Tietovarastojen data voi kuitenkin olla keskenään erilaisissa tietoformaateissa. Kirjoittajat näkevät XML-merkintäkielessä ratkaisun tähän ongelmaan, sillä nykyään lähes kaikista tietojärjestelmistä on tiedot saatavissa XML-muodossa ja toisaalta XML-muodossa esitetyt tiedot on mahdollista konvertoida lähes mihin tahansa muuhun tietoformaattiin. Hümmerin, Bauerin ja Harden luoman XCube-standardin ytimessä on kolme XML Schema -kaaviota, joiden avulla voidaan kuvata tietokuution moniulotteinen kaavio (XCubeSchema), yksittäiseen dimensioon liittyvät rakenteet (XCubeDimension) ja tietokuution varsinaisen sisältö (XCubeFact). Kirjoittajat esittävät useita tapausesimerkkejä, joiden avulla he osoittavat mallipohjiensa hyödyllisyyden. XCube-spesifikaation lisäksi kirjoittajat ovat kehittäneet järjestelmäprototyyppejä, joiden avulla he ovat testanneet dokumenttipohjien hyödyllisyyttä todellisissa käyttötilanteissa.

Jensen, Møller ja Pedersen [JMP01] näkevät, että seurauksena kehityksestä, jossa Internetissä on enenevässä määrin tarjolla liiketoiminnan kannalta relevanttia informaatiota (esimerkiksi B2B-kauppapaikoilla) ja jossa organisaatiot toimivat entistä tiiviimmässä yhteistyössä liikekumppaneidensa kanssa, on syntynyt selkeä tarve hajautettujen tietolähteiden yhdistämiselle. Internetissä olevat tietoresurssit ovat yhä useammin XML-muotoisia, kun taas organisaatioiden tietojärjestelmien ytimessä ovat tavallisesti relaatiotietokannat. Tällaisia resursseja ei kirjoittajien mukaan ole aina kannattavaa – tai edes mahdollista – yhdistää fyysisesti, vaan niiden integroitu käyttö vaatii tietojen loogista ja käsitteellistä integrointia. Tähän tarkoitukseen Jensen, Møller ja Pedersen esittävät UML (*Unified Modelling Language*) -kuvauskieltä hyödyntävän lumihiutalemallin, jonka avulla on mahdollista määritellä moniulotteinen tietokanta, jonka muodostamisessa on käytetty useita XML-pohjaisia ja/tai relaatiomalliin perustuvia tietolähteitä. XML-dokumenttien rakennetta visualisoidaan muuntamalla niihin liittyvät DTD-kuvaukset UML-kaavioiksi.

Pedersen, Riis ja Pedersen [PRP02a] ovat määritelleet moniulotteisen tietomallin ja algebran ja kehittäneet niihin perustuvan ilmaisuvoimaisen ja SQL-yhteensopivan OLAP-kyselykielen. Ilmaisuvoimaisuus merkitsee tässä sitä, että kyselykielen avulla pystytään käsittelemään epäsäännöllisiä dimensioattribuuttien hierarkioita ja aggregoimaan tietoja automaattisesti ja oikein. SQL-yhteensopivuus puolestaan merkitsee sitä, että kyselykieli on yhdistettävissä olemassa olevaan relaatiotietokantateknologiaan. Kirjoittajien kehittämässä kyselykielessä on mahdollista XPath-polkuilmauksia käyttäen liittää OLAP-kyselyyn ulkoista XML-informaatiota, jonka avulla tulokseen voidaan liittää täydentäviä tietoja, ryhmitellä tulosta tai suorittaa valintaoperaatioita. Kirjoittajat ovat laajentaneet tätä lähestymistapaa mm. artikkeleissa [PRP02b] ja [PP03], joissa käsitellään lisäksi kyselyn prosessointia ja optimointia ja kaavion muutosten hallintaa.

Niemi, Niinimäki, Nummenmaa ja Thanisch [NNNT02] ovat kehittäneet tähtimallityyppisen formalismin, jonka avulla tietokuutio ja sen kaavio voidaan esittää XML-muodossa. XML-muotoinen tietokuutio voidaan myös helposti muuttaa muotoon, jossa OLAP-palvelimet pystyvät sitä käsittelemään. Formalismin lisäksi kirjoittajat ovat toteuttaneet järjestelmän, jolla tietokuutioita voidaan suunnitella ja konstruoida. Koska tietokuutio voidaan muodostaa heterogeenisissa tietolähteissä olevasta informaatiosta, käytetään tiedon keräämisessä XML:ää, sillä lähes kaikista järjestelmistä tiedot on saa-

tavissa XML-muodossa. Järjestelmässä kyselyt tehdään MDX (*Multidimensional Expressions*) -kyselykielellä (ks. esim. [Spo01]).

Artikkelissa [NNNT03] Niemi, Niinimäki, Nummenmaa ja Thanisch selvittävät Grid-teknologioiden käyttöä tietokuution konstruoinnissa. Grid on ohjelmistoinfrastruktuuri, joka mahdollistaa (laskennallisten ja tieto-) resurssien joustavan, turvallisen ja koordinoidun jakamisen dynaamisesti vaihtuvan yksilöiden, instituutioiden ja resurssien joukon sisällä [FK98]. Grid-pohjaisessa tietokuution muodostamisessa jokaisen osaresurssin kohdalla muodostetaan osakuutio paikallisesti, minkä jälkeen kukin osakuutio siirretään verkon yli keskuspalvelimelle, jossa osakuutiot yhdistetään yhdeksi tietokuutioksi. Tietokuutio ja osakuutiot esitetään jälleen XML-muodossa.

6.3 Jatkokehitys

XML-dokumenttien elementeissä ja attribuuteissa käytettävien nimien erilaisuudesta tiedon integroinnille aiheutuvan ongelman ratkaisemisen lisäksi ilmeisiä jatkotutkimusalueita on kolme. XML-dokumenteissa voi semanttista heterogeenisuutta ilmetä paitsi elementtien ja attribuuttien nimissä myös elementtien ja attribuuttien sisällössä esitettävässä informaatiossa. Esimerkiksi dokumenteissa voidaan käyttää eri valuuttayksiköitä ja eri ajanjaksoina valuuttayksikköjen arvot vaihtelevat. Tästä aiheutuvien ns. konversioongelmien ratkaiseminen on tärkeää, kun halutaan varmistua, että OLAP-analyysi perustuu oikeelliseen informaatioon.

Nykyisen järjestelmäprototyypin avulla pystytään tuottamaan ainoastaan tähti- ja lumihiutalemalleissa käytettävän faktataulun kaltainen peruskuutio. Vastaisuudessa järjestelmäprototyyppiä on kehitettävä niin, että sen avulla tulee mahdolliseksi esittää myös dimensioattribuutteihin liittyviä ominaisuuksia. Vastaisuuteen jää myös XML-dokumenttien komponenttien suppeimman mahdollisen kontekstin tehokkaasti laskevan algoritmin suunnittelu ja sen aikavaatimuksen määrittäminen.

Luku 7

Päätelmät

Tutkielmassa käsiteltiin tietokuution muodostamista rakenteeltaan heterogeenisista XML-dokumenteista. Esimerkiksi Internetissä esiintyvän relevantin XML-muotoisen tiedon jatkuvasti kasvavasta osuudesta on seurauksena, että tietokuutioita on luontevaa muodostaa suoraan XML-tiedon perusteella. Tietokuution muodostaminen edellyttää autonomisista tietolähteistä saatavissa, heterogeenisissa XML-tietolähteissä olevien tietojen integrointia. XML-tiedon integroinnissa on omat erityisongelmansa. XML-merkintäkielen puolirakenteisesta luonteesta aiheutuu, että eri dokumenteissa ja jopa yksittäisen dokumentin sisällä voidaan samaa tarkoittava informaatio esittää useilla rakenteellisesti toisistaan poikkeavilla tavoilla. Samaa tarkoittavaa informaatiota voidaan XML-dokumenteissa esittää lisäksi erinimisten elementtien ja attribuuttien avulla. XML-dokumentteihin liittyvää semanttista heterogeenisuutta ei tutkielmassa kuitenkaan käsitelty.

XML-tietojen integroinnin edellytyksenä on, että rakenteeltaan heterogeenisista ja/tai tuntemattomista (mutta semantiikaltaan tunnetuista) XML-dokumenteista saadaan joustavasti poimittua tarvittavat tiedot. XML-dokumenteissa olevien tietojen kyselemiseen yleisesti käytettävien polkuorientoituneiden kyselykielten (esimerkiksi XQuery) suurin ongelma XML-tietojen integroinnin näkökulmasta on, että ne edellyttävät kyselyntekijän tuntevan lähdedokumenttien rakenteen ja että jokaista rakenteeltaan erilaista XML-dokumenttia ja niiden osaa varten on muodostettava oma polkuilmauksensa. Tietojen laajamittaisen integroinnin yhteydessä tällaiset edellytykset eivät ole realistisia.

Tutkielman yhteydessä on kehitetty `is_component_of`-kyselyprimitiivi, joka tarjoaa helpon ja deklaratiivisen tavan XML-muodossa esitettyjen tietojen poimintaan. Kyselyprimitiivissä sovelletaan XML-dokumenttien komponenttien poimintaan tiedonhakutyyppistä lähestymistapaa, jossa XML-dokumentin komponentteja käsitellään ikään kuin hakusanoina. Kyselyn kannalta relevanttien komponenttien kombinaatiot valitaan tutkielmassa hahmotellun uuden suppeimman mahdollisen kontekstin semantiikan avulla. Esimerkkien avulla osoitettiin, että suppeimman mahdollisen kontekstin semantiikalla saadaan joissakin tilanteissa tuotettua perinteisiin LCA-pohjaisiin semantiikkoihin verrattuna tarkempia vastauksia kyselyihin.

Kehitetyn `is_component_of`-kyselyprimitiivin ilmaisuvoiman osoittamiseksi tutkiel-

man yhteydessä on kehitetty järjestelmäprototyyppi, jonka avulla käyttäjä voi helposti muodostaa yksinkertaisia tietokuutioita XML-muotoisten tietolähteiden tietojen perusteelta. Järjestelmäprototyypin toimintaa demonstroitiin useiden esimerkkikyselyiden perusteella.

Lähteet

[ABC+01] Sharon Adler, Anders Berglund, Jeff Caruso, Stephen Deach, Tony Graham, Paul Grosso, Eduardo Gutentag, Alex Milowski, Scott Parnell, Jeremy Richman, and Steve Zilles. *Extensible Stylesheet Language (XSL) Version 1.0.* W3C Recommendation, October 15, 2001. Available as *http://www.w3.org/TR/xsl/*.

[Abi97] Serge Abiteboul. Querying semi-structured data. In *Proceedings of the 6th International Conference on Database Theory, Delphi, Greece, January 8–10, 1997*, 1–18.

[Abi03] Serge Abiteboul. Managing an XML warehouse in a P2P environment. In J. Elder, M. Missikoff (Eds.): *Proceedings of the 15th International Conference Advanced Information Systems Engineering.* LNCS **2681** (2003), Springer, 4–13.

[AGS97] Rakesh Agrawal, Ashish Gupta, and Sunita Sarawagi. Modeling multidimensional databases. In A. Gray, P.-Å. Larson (Eds.): *Proceedings of the 13th International Conference on Data Engineering, Birmingham, UK, April 7–11, 1997.* IEEE Computer Society, 1997, 232–243.

[AQM+97] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Janet L. Wiener. The Lorel Query Language for Semistructured Data. *International Journal on Digital Libraries* **1**, 1 (April 1997), 68–88.

[BCF+03] Scott Boag, Don Chamberlin, Mary F. Fernàndes, Daniela Florescu, Jonathan Robie, and Jérôme Siméon. *XQuery 1.0: An XML Query Language.* W3C Working Draft, November 2003. Available as *http://www.w3.org/TR/xquery.*

[BDHS96] Peter Buneman, Susan Davidson, Gerd Hillebrand, and Dan Suciu. A query language and optimization techniques for unstructured data. *ACM SIGMOD Record* **25**, 2 (June 1996), 505–516.

[Bec04] Dave Beckett. *RDF/XML Syntax Specification (Revised).* W3C Recommendation, February 10, 2004. Available as *http://www.w3.org/TR/rdf-syntax-grammar/.*

[BG04] Dan Brickley and R.V. Guha. *RDF Vocabulary Description Language 1.0: RDF Schema.* W3C Recommendation, February 10, 2004. Available as *http://www.w3.org/TR/rdf-schema/.*

[BHL99] Tim Bray, Dave Hollander, and Andrew Layman. *Namespaces in XML*. W3C Recommendation, January 1999. Available as *http://www.w3.org/TR/REC-xml-names*.

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American* **5**, May 2001.

[BM04] Paul V. Biron and Ashok Malhotra. *XML Schema Part 2: Datatypes (Second Edition)*. W3C Recommendation, October 28, 2004. Available as *http://www.w3.org/TR/xmlschema-2/*.

[BPS+04] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau, and John Cowan. *Extensible Markup Language (XML) 1.1: Third Edition*. W3C Recommendation, February 2004. Available as *http://www.w3.org/TR/xml11*.

[Bun97] Peter Buneman. Semistructured data. In *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Tucson, AZ, May 12–14, 1997*, 117–121.

[CA96] R. G. G. Cattell and Tom Atwood (Eds). *The Object Database Standard: ODMG-93, Release 1.2*. Morgan Kaufmann, 1996.

[CCS93] E. F. Codd, Sharon B. Codd, and Clynch T. Salley. *Providing OLAP (online analytical processing) to user-analysts: An IT mandate*. Technical Report. E. F. Codd Associates, 1993.

[CD97] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and OLAP technology. *ACM SIGMOD Record* **26**, 1 (March 1997), 65–74.

[CD99] James Clarke and Steve DeRose. *XML Path Language (XPath). Version 1.0*. W3C Recommendation, November 1999. Available as *http://www.w3.org/TR/xpath*.

[CDG01] Surajit Chaudhuri, Umeshwar Dayal, and Venkatesh Ganti. Database technology for decision support systems. *Computer* **34**, 12 (December 2001), 48–55.

[CDSS98] Sophie Cluet, Claude Delobel, Jérôme Siméon, and Katarzyna Smaga. Your mediators need data conversion! *ACM SIGMOD Record* **27**, 2 (June 1998), 177–188.

[CGG04] Conor Cunnigham, César A.Galindo-Legaria, and Goetz Graefe. PIVOT and UNPIVOT: Optimization and execution strategies in an RDBMS. In *Proceedings of the 30th International Conference on Very Large Data Bases, Toronto, Canada, 2004*, 998–1009.

[Cham02] Don Chamberlin. XQuery: An XML query language. *IBM Systems Journal* **41**, 4 (2002), 597–615.

[Cla99] James Clark. *XSL Transformations (XSLT)*. W3C Recommendation, November X, 1999. Available as: *http://www.w3.org/TR/xslt*.

[CRF00] Don Chamberlin, Jonathan Robie, and Daniela Florescu. Quilt: an XML Query Language for Heterogeneous Data Sources. In Dan Suciu and Gottfried Vossen (Eds.): *Selected papers from the 3rd International Workshop on The World Wide Web and Databases, Dallas, TX, May 18–19, 2000.* LNCS **1997** (2000), Springer, 1–25.

[CT98] Luca Cabibbo and Riccardo Torlone. A logical approach to multidimensional databases. In H.-J. Schek, F. Saltor, I. Ramos, G. Alonso (Eds.): *Advances in Database Technology – Proceedings of the 6th International Conference on Extending Database Technology, Valencia, Spain, March 23–28, 1998.* LNCS **1377** (1998), Springer, 183–197.

[DFF+99] Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. A Query Language for XML. In Alberto Mendelzon (Ed.): *Proceedings of the 8th International World Wide Web Conference, Toronto, Canada, May 11–14, 1999*, 77–91.

[DHB+00] Stefan Decker, Sergey Melnik, Frank Van Harmelen, Dieter Fensel, Michel Klein, Jeen Broekstra, Michael Erdmann, and Ian Horrocks. The semantic web: The roles of XML and RDF. *IEEE Internet Computing* **15**, 3 (October 2000), 63–74.

[EN00] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems. Third Edition.* Addison-Wesley, 2000.

[FCD02] Ling Feng, Elizabeth Chang, and Tharam Dillon. A Semantic Network-Based Design Methodology for XML Documents. *ACM Transactions on Information Systems* **20**, 4 (October 2002), 390–421.

[FFLS97a] Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. A query language and processor for a web-site management system. In *Proceedings of the Workshop on Management of Semi-structured Data, Tucson, AZ, May 1997*, 26–33.

[FFLS97b] Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. A query language for a web-site management system. *ACM SIGMOD Record* **26**, 3 (September 1997), 4–11.

[FG01] Norbert Fuhr and Kai Grossjohann. XIRGL: A query language for information retrieval in XML documents. In: *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, New Orleans, LA, September 9–12, 2001*, 172–180.

[FK98] Ian Foster and Carl Kesselman (Eds.). *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann, 1998.

[FMM+05] Mary Fernández, Ashok Malhotra, Jonathan Marsh, Marton Nagy, and Norman Walsh. *XQuery 1.0 and XPath 2.0 Data Model.* W3C Working Draft, April 2005. Available as *http://www.w3.org/TR/xpath-datamodel/.*

[FS03] Mary Fernández and Jérôme Siméon. Growing XQuery. In: L. Cardelli (Ed.): *Proceedings of the 17th European Conference on Object-Oriented Programming, Darmstadt, Germany, July 21–25, 2003.* LNCS **2743** (2003), Springer, 405–430.

[FW04] David C. Fallside and Priscilla Walmsley. *XML Schema Part 0: Primer (Second Edition).* W3C Recommendation, October 28, 2004. Available as *http://www.w3.org/TR/xmlschema-0/.*

[GB04] Jan Grant and Dave Beckett. *RDF Test Cases.* W3C Recommendation, February 10, 2004. Available as *http://www.w3.org/TR/rdf-testcases/.*

[GCB+97] Jim Gray, Surijit Chaudhuri, Adam Bosworth, Andrew Layman, D. Reichart, M. Venkatrao, F. Pellow, and Hamid Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery* **1**, 1 (March 1997), 29–54.

[GG95] Nicola Guarino and Pierdaniele Giaretta. Ontologies and knowledge bases: Towards a terminological clarification. In N. Mars (Ed.): *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing.* IOS Press, 1995, 25–32.

[GL97] Marc Gyssens and Laks V. S. Lakshmanan. A foundation for multi-dimensional databases. In M. Jarke, M. J. Carey, K. R. Dittrich, F. H. Lochovsky, P. Loucopoulos, M. A. Jeusfeld (Eds.): *Proceedings of the 23rd International Conference on Very Large Data Bases, Athens, Greece, August 25–29, 1997.* Morgan Kaufmann, 1997, 106–115.

[Gru95] Thomas R. Gruber: Toward principles for the design of ontologies used for knowledge sharing. *International Journal for Human-Computer Studies* **43**, 5/6 (1995), 907–928.

[GRV01] Matteo Golfarelli, Stefano Rizzi, and Boris Vrdoljak. Data warehouse design from XML sources. In *Proceedings of the 3rd ACM International Workshop on Data Warehousing and OLAP, Atlanta, GE, November 9, 2001*, 40–47.

[Hay04] Patrick Hayes. *RDF Semantics.* W3C Recommendation, February 10, 2004. Available as *http://www.w3.org/TR/rdf-mt/.*

[HBH03] Wolfgang Hümmer, Andreas Bauer, and Gunnar Harde. XCube: XML for data warehouses. In *Proceedings of the 5th ACM International Workshop on Data Warehousing and OLAP, New Orleans, LA, November 7, 2003*, 33–39.

[Hir01] Lasse Hirvonen. *Helppokäyttöisen OLAP-kyselykielen suunnittelu ja toteutus.* Pro gradu -tutkielma, Tietojenkäsittelytieteiden laitos, Tampereen yliopisto, maaliskuu 2001. Saatavana sähköisessä muodossa *http://www.cs.uta.fi/opiskelu/tutkielmat/pro_gradut.html.*

[ISO86] International Organization for Standardization. *ISO 8879: Information processing – text and office systems – Standard Generalized Markup Language (SGML).* International Organization for Standardization, Geneva, 1986.

[ISO/IEC00] International Organization for Standardization and International Electrotechnical Commission. *ISO/IEC 15445: Information technology – Document description and processing languages – HyperText Markup Language (HTML).* International Organization for Standardization, Geneva, 2000.

[ISO/IEC03] International Organization for Standardization and International Electrotechnical Commission. *ISO/IEC 9075-2:2003: Information technology – Database languages – SQL – Part 2: Foundation (SQL/Foundation).* International Organization for Standardization, Geneva, 2003.

[JMP01] Mikael R. Jensen, Thomas H. Møller, Torben Bach Pedersen: Specifying OLAP cubes on XML data. In *Proceedings of the 13th International Conference on Scientific and Statistical Database Management, Fairfax, VA, July 18–20, 2001*, 101–112.

[KC04] Graham Klyne and Jeremy J. Carroll. *Resource Description Framework (RDF): Concepts and Abstract Syntax.* W3C Recommendation, February 10, 2004. Available as *http://www.w3.org/TR/rdf-concepts/.*

[Liu99] Menchi Liu. Deductive database languages: Problems and solutions. *ACM Computing Surveys* **31**, 1 (March 1999), 27–62.

[LS97] Hans-Joachim Lenz and Arie Shoshani. Summarizability in OLAP and statistical data bases. In Y. E. Ioannidis, D. M. Hansen (Eds.): *Proceedings of the 9th International Conference on Scientific and Statistical Database Management, Olympia, WA, August 11–13, 1997.* IEEE Computer Society, 1997, 132–143.

[LSS01] Laks V. S. Lakshmanan, Fereidoon Sadri, and Subbu N. Subramanian. SchemaSQL - An extension to SQL for multidatabase interoperatibility. *ACM Transactions on Database Systems* **26**, 4 (December 2001), 476–519.

[LT01] Hans-Joachim Lenz and Bernhard Thalheim: OLAP Databases and Aggregation Functions. In: *Proceedings of the 13th International Conference on Scientific and Statistical Database Management, Fairfax, VA, July 18–20, 2001*, 91–100.

[LYJ04] Yunyao Li, Cong Yu, and H. V. Jagadish. Schema-free XQuery. In *Proceedings of the 30th International Conference on Very Large Databases, Toronto, Canada, August 31 - September 3, 2004*, 72–83.

[Mar99] Patrick Marcel. Modeling and querying multidimensional databases: An overview. *Networking and Information Systems Journal* **2**, 5–6 (December 1999), 515–548.

[MM04] Frank Manola and Eric Miller. *RDF Primer.* W3C Recommendation, February 10, 2004. Available as *http://www.w3.org/TR/rdf-primer/.*

[MMRR03] Ashok Malhotra, Jim Melton, Jonathan Robie, and Michael Rys. *XML Syntax for XQuery 1.0 (XQueryX).* W3C Working Draft, December 19, 2003. Available as *http://www.w3.org/TR/xqueryx.*

[NHJ02] Timo Niemi, Lasse Hirvonen, and Kalervo Järvelin. *Multidimensional Data Model and Query Language for Advanced Applications.* Report **A-2002-10**, Department of Computer and Information Sciences, University of Tampere, June 2002.

[NHJ03] Timo Niemi, Lasse Hirvonen, and Kalervo Järvelin. Multi-dimensional data model and query language for informetrics. *Journal of the American Society for Information Science and Technology* **54**, 10 (August 2003), 939–951.

[Nie83] Timo Niemi. A seven-tuple representation for hierarchical data structures. *Information Systems* **8**, 3 (1983), 151–157.

[NMW00] Thanh Binh Nguyen, A Min Tjoa, and Roland Wagner. An object oriented multidimensional data model for OLAP. In H. Lu, A. Zhou (Eds.): *Proceedings of the 1st International Conference on Web-Age Information Management, Shanghai, China, June 21–23, 2000.* LNCS **1846** (2000), Springer, 83–94.

[NNNT02] Tapio Niemi, Marko Niinimäki, Jyrki Nummenmaa, and Peter Thanisch. Constructing an OLAP cube from distributed XML data. In: *Proceedings of the 5th ACM International Workshop on Data Warehousing and OLAP, McLean, VA, November 4–9, 2002*, 22–27.

[NNNT03] Tapio Niemi, Marko Niinimäki, Jyrki Nummenmaa, Peter Thanisch: Applying Grid technologies to XML based OLAP cube construction. In: *Proceedings of the 5th International Workshop on Design and Management of Data Warehouses, Berlin, Germany, September 8, 2003*, 4–13.

[PAGM96] Yannis Papakonstantinou, Serge Abiteboul, and Hector Garcia-Molina. Object fusion in mediator systems. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, Nandlal L. Sarda (Eds.): *Proceedings of 22nd International Conference on Very Large Data Bases, Mumbai (Bombay), India, September 3–6, 1996.* Morgan Kaufmann, 1996, 413–424.

[PGMW95] Yannis Papakonstantinou, Hector Garcia-Molina, and Jennifer Widom. Object exchange across heterogeneous information sources. In: *Proceedings of the 11th International Conference on Database Engineering, Taipei, Taiwan, March 6–10, 1995*, 251–260.

[PJ99] Torben Bach Pedersen and Cristian S. Jensen. Multidimensional data modeling for complex data. In: *Proceedings of the 15th International Conference on Data Engineering, Sydney, Australia, March 23–26, 1999*, 336–345.

[PJ01] Torben Bach Pedersen and Cristian S. Jensen. Multidimensional database technology. *Computer* **34**, 12 (December 2001), 40–46.

[PP03] Dennis Pedersen and Torben Bach Pedersen. Achieving adaptivity for OLAP-XML federations. In: *Proceedings of the 6th ACM International Workshop on Data Warehousing and OLAP, New Orleans, LA, November 7, 2003*, 25–32.

[PRP02a] Dennis Pedersen, Karsten Riis, and Torben Bach Pedersen. A powerful and SQL-compatible data model and query language for OLAP. In: *Proceedings of the 13th Australasian Database Conference, Melbourne, Australia, January 28 - February 1, 2002*, 121–130.

[PRP02b] Dennis Pedersen, Karsten Riis, and Torben Bach Pedersen. XML-extended OLAP querying. In: *Proceedings of 14th International Conference on Scientific and Statistical Database Management, Edinburgh, Scotland, July 24–26, 2002*, 195–206.

[RDSH02] Allen Renear, David Dubin, C. M. Sperberg-McQueen, and Claus Huitfeldt. Towards a semantic for XML Markup. In: *Proceedings of the 2002 ACM Symposium on Document Engineering, McLean, VA, November 8–9, 2002*, 119–126.

[RLS98] J. Robie, J. Lapp, D. Schach. *XML Query Language (XQL)*. 1998 Available as *http://www.w3.org/TandS/QL/QL98/pp/xql.html*.

[RS90] Maurizio Rafanelli and Arie Shoshani. STORM: A statistical object representation model. In: *Proceedings of the 5th Conference on Statistical and Scientific Database Management, Charlotte, NC, April 3–5, 1990*, 14-Ũ29.

[Sho97] Arie Shoshani. OLAP and statistical databases: Similarities and differences. In: *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Tucson, Arizona, May 12–14, 1997*, 185–196.

[Spo01] George Spofford. *MDX-Solutions: With Microsoft SQL Server Analysis Services*. John Wiley & Sons, 2001

[SS94] Leon Sterling and Ehud Shapiro. *The Art of Prolog: Advanced Programming Techniques. Second Edition*. The MIT Press, 1994.

[SS05] Arun Sen and Atish P. Sinha. A comparison of data warehousing methodologies. *Communications of the ACM* **48**, 3 (March 2005), 79–84.

[Suc00] Dan Suciu. Semistructured data and XML. In: *Proceedings of the 5th International Conference of Foundations of Data Organization, Kobe, Japan, November 12-13, 1998*, 1 – 12.

[SW03] Jérôme Siméon and Philip Wandler. The essence of XML. In: *Proceedings of the POPL'03, New Orleans, LA, January 15–17, 2003*, 1–13.

[TBMM04] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. *XML Schema Part 1: Structures (Second Edition)*. W3C Recommendation, October 28, 2004. Available as *http://www.w3.org/TR/xmlschema-1/*.

[Tho97] Erik Thomsen. *OLAP Sollutions: Building Multidimensional Information Systems*. John Wiley & Sons, 1997.

[Tom89] Frank Tompa. What is (tagged) text? In: *Proceedings of the 5th Annual Conference of University of Waterloo Centre for the New OED, Oxford, UK, September 18–19, 1989*, 81–93.

[Vas98] Panos Vassiliadis. Modeling multidimensional databases, cubes and cube operations. In: *Proceedings of the 10th International Conference on Scientific and Statistical Data Management, Capri, Italy, April 1–3, 1998*, 53–62.

[Via03] Victor Vianu. A web odyssey: from Codd to XML. *ACM SIGMOD Record* **32**, 2 (June 2003), 68–77.

[VS99] Panos Vassiliadis and Timos Sellis. A survey of logical models for OLAP databases. *ACM SIGMOD Record* **28**, 4 (December 1999), 64–69.

[XP05] Yu Xu and Yannis Papakonstantinou. Efficient keyword search for smallest LCAs in XML databases. In: *Proceedings of the ACM SIGMOD 2005, Baltimore, MD, June 14–16, 2005*, 527–538.

[ZS99] Thomas Zürek and Markus Sinnwell. Data warehousing has more colours than just black and white. In: *Proceedings of the 25th International Conference on Very Large Data Bases, Edinburgh, Scotland, September 7–10, 1999*, 726–729.

Liite A. Konstruointi-ilmauksen BNF-määrittely

Seuraavassa on määritelty järjestelmäprototyyppille annettavan tietokuution konstruointi-ilmauksen yleinen rakenne BNF-notaatiota[1] käyttäen.

```
              query  =  create_cube, " ", where ;
       create_cube   =  "CREATE CUBE", " ", cube_specification ;
cube_specification   =  cube_name, "(", dim, (",", dim)*, ",",
                        mes, (",", mes)*, ")", ;
         cube_name   =  atom ;
               dim   =  "dim", "(", atom, ",", variable,
                        (",", variable)*, ")" ;
               mes   =  "mes", "(", atom, ",", variable, ",", func, ")" ;
              func   =  "min" | "max" | "count" | "sum" | "avg" ;
             where   =  "WHERE", " ", model_expression,
                        (";", model_expression)* ;
  model_expression   =  set_expression, ",", set_expression,
                        ico_expression ;
    set_expression   =  variable, " ", "=", " ", "{", comp_expression,
                        (",", comp_expression)*, "}" ;
    comp_expression  =  atom, "(", atom, ")" | atom, "(", variable, ")"
                        | variable, "(", atom, ")", | variable, "(", variable, ")"
                        | variable, "=", atom ;
     ico_expression  =  variable, " ", "is_component_of", " ",
                        variable, " ", "in", " ", variable ;
               atom  =  lc_letter+, (lc_letter* | uc_letter* | char*)* ;
           variable  =  uc_letter+, (lc_letter* | uc_letter* | char*)* ;
         lc_letter   =  "a" | "b" | "c" | "d" | "e" | "f" | "g"
                        | "h" | "i" | "j" | "k" | "l" | "m" | "n"
                        | "o" | "p" | "q" | "r" | "s" | "t" | "u"
                        | "v" | "w" | "x" | "y" | "z" ;
         uc_letter   =  "A" | "B" | "C" | "D" | "E" | "F" | "G"
                        | "H" | "I" | "J" | "K" | "L" | "M" | "N"
                        | "O" | "P" | "Q" | "R" | "S" | "T" | "U"
                        | "V" | "W" | "X" | "Y" | "Z" ;
              char  =  "-" | "_" ;
```

[1] International Organization for Standardization and International Electrotechnical Commission. *ISO/IEC 14977: Information Technology – Syntactic Metalanguage – Extended BNF*. International Organization for Standardization, Geneva, 1996

Liite B. Esimerkkidokumenttien tiivistelmät

Taulukko 1: Tyyppiä `publishing` olevien esimerkkidokumenttien tietosisältö.

article	author				year	publisher	domain	type
	smith	jones	hikes	wilkins				
art1	x				1995	publisher4	db	refereed
art2	x	x			1995	publisher2	ir	refereed
art3				x	1995	publisher1	db	refereed
art4		x	x		1996	publisher3	db	non_refereed
art5	x				1996	publisher3	ir	refereed
art6		x		x	1997	publisher2	ir	non_refereed
art7	x	x		x	1997	publisher4	db	refereed
art8	x				1997	publisher4	db	refereed
art9	x				1998	publisher1	ir	refereed
art10		x	x	x	1998	publisher2	db	non_refereed
art11	x		x		1999	publisher1	db	refereed
art12		x			1999	publisher1	ir	refereed
art13			x	x	1999	publisher4	db	non_refereed
art14				x	2000	publisher3	ir	refereed
art15	x		x		2000	publisher2	db	refereed
art16	x				2001	publisher3	db	refereed
art17		x			2001	publisher4	db	refereed
art18	x	x	x		2001	publisher2	ir	refereed
art19			x	x	2002	publisher4	ir	non_refereed
art20				x	2002	publisher1	db	refereed
art21	x		x		2003	publisher1	db	refereed
art22	x			x	2003	publisher3	db	refereed
art23		x	x		2003	publisher4	ir	non_refereed
art24		x		x	2004	publisher3	db	non_refereed
art25	x	x		x	2004	publisher2	ir	refereed

Taulukko 2: Tyyppiä `institution` olevien esimerkkidokumenttien tietosisältö.

	db							ir							
	inst1	inst2	inst3	inst4	inst5	inst6	inst7	inst1	inst2	inst3	inst4	inst5	inst6	inst7	
1995	80	20								5				10	smith
1996	20	15												20	
1997	30	5			3					5	25		25	20	
1998	10				2										
1999	100	25		25											
2000	100	20													
2001	50	15		25	4										
2002	20									25		4	25	20	
2003	10				4					5		2			
2004	100	25		25	4										
1995	20											3			jones
1996		15										3			
1997	10				3							6		20	
1998	50	20		5	3					15			100	20	
1999	20				3							1	25	20	
2000	30			5						10	25				
2001	10				2					10		3		20	
2002		20			4							6			
2003	50				6					25		3	25	20	
2004													100		
1995	50	10													hikes
1996	50											1			
1997															
1998															
1999															
2000	20	10		5								3			
2001	20	15		5						15		2			
2002	20	25		5	4							6			
2003	30	15								15					
2004	50	25		25						10		3			
1995		10										2		10	wilkins
1996	20	15		5						15		2			
1997	30	5		5	4					10	25				
1998	100	15		5	4										
1999	25	25		5	3					25	25		25	20	
2000	20	20		5						25	25	5	25		
2001		10			2							3			
2002	20				2					5	25	2			
2003	100	15		5						5	25		70	20	
2004	100	20								25	25		35	25	

Taulukko 3: Tyyppiä `article` olevien esimerkkidokumenttien tietosisältö.

article	references											
	art1	art2	art3	art4	art5	art6	art7	art8	art9	art10	art11	art12
art2	x											
art3	x	x										
art4	x		x									
art5				x								
art6	x		x		x							
art7			x		x	x						
art8		x		x	x		x					
art9						x		x				
art10	x		x						x			
art11		x		x	x	x	x	x		x		
art12	x		x		x	x			x	x		
art13					x	x		x	x		x	x
art14		x	x	x		x	x	x			x	
art15	x		x		x		x					x
art16	x		x	x		x	x	x		x	x	
art17		x	x	x		x		x		x	x	x
art18	x		x		x	x		x	x	x		x
art19	x										x	
art20	x	x	x	x			x		x			x
art21		x		x			x		x	x		x
art22		x			x			x				
art23	x		x		x	x	x	x		x		x
art24	x	x	x		x	x			x		x	
art25		x	x	x			x		x	x	x	

article	references											
	art.13	art.14	art.15	art.16	art.17	art.18	art.19	art.20	art.21	art.22	art.23	art.24
art.2												
art.3												
art.4												
art.5												
art.6												
art.7												
art.8												
art.9												
art.10												
art.11												
art.12												
art.13												
art.14	x											
art.15	x	x										
art.16	x		x									
art.17	x		x	x								
art.18		x		x	x							
art.19	x	x		x		x						
art.20	x	x	x	x	x		x					
art.21		x		x			x	x				
art.22	x		x		x	x	x	x	x			
art.23			x			x	x	x	x	x		
art.24	x		x		x		x	x	x	x	x	
art.25			x	x	x		x	x	x	x	x	x

Liite C. Esimerkkikyselyjen vastaukset

Taulukko 4: Esimerkkikyselyn 1 vastausten tiivistelmä.

article_cube	year	author	type	number_of_articles
	1995	jones	refereed	1
	1995	smith	refereed	2
	1995	wilkins	refereed	1
	1996	hikes	non_refereed	1
	1996	jones	non_refereed	1
	1996	smith	refereed	1
	1997	jones	non_refereed	1
	1997	jones	refereed	1
	1997	smith	refereed	2
	1997	wilkins	non_refereed	1
	1997	wilkins	refereed	1
	1998	hikes	non_refereed	1
	1998	jones	non_refereed	1
	1998	smith	refereed	1
	1998	wilkins	non_refereed	1
	1999	hikes	non_refereed	1
	1999	hikes	refereed	1
	1999	jones	refereed	1
	1999	smith	refereed	1
	1999	wilkins	non_refereed	1
	2000	hikes	refereed	1
	2000	smith	refereed	1
	2000	wilkins	refereed	1
	2001	hikes	refereed	1
	2001	jones	refereed	2
	2001	smith	refereed	2
	2002	hikes	non_refereed	1
	2002	wilkins	non_refereed	1
	2002	wilkins	refereed	1
	2003	hikes	non_refereed	1
	2003	hikes	refereed	1
	2003	jones	non_refereed	1
	2003	smith	refereed	2
	2003	wilkins	refereed	1
	2004	jones	non_refereed	1
	2004	jones	refereed	1
	2004	smith	refereed	1
	2004	wilkins	non_refereed	1
	2004	wilkins	refereed	1

Taulukko 5: Esimerkkikyselyn 2 vastausten tiivistelmä.

grant_cube	year	institution	domain	avg_grants	sum_grants	min_grant	max_grant
	1995	institution1	db	50	150	20	80
	1995	institution2	db	17.5	35	10	25
	1995	institution3	ir	5	5	5	5
	1995	institution5	ir	2.5	5	2	3
	1995	institution7	ir	10	10	10	10
	1996	institution1	db	35	70	20	50
	1996	institution2	db	15	15	15	15
	1996	institution3	ir	15	15	15	15
	1996	institution4	db	5	5	5	5
	1996	institution5	ir	2	6	1	3
	1996	institution7	ir	20	20	20	20
	1997	institution1	db	20	40	10	30
	1997	institution2	db	10	20	5	15
	1997	institution3	ir	10	10	10	10
	1997	institution4	db	15	30	5	25
	1997	institution4	ir	25	25	25	25
	1997	institution5	db	3.5	7	3	4
	1997	institution5	ir	6	6	6	6
	1997	institution6	ir	25	25	25	25
	1997	institution7	ir	20	20	20	20
	1998	institution1	db	53.33	160	10	100
	1998	institution2	db	17.5	35	15	20
	1998	institution3	ir	15	15	15	15
	1998	institution4	db	5	5	5	5
	1998	institution5	db	3	9	2	4
	1998	institution6	ir	100	100	100	100
	1998	institution7	ir	20	20	20	20
	1999	institution1	db	48.33	145	20	100
	1999	institution2	db	25	25	25	25
	1999	institution3	ir	25	25	25	25
	1999	institution4	db	5	5	5	5
	1999	institution4	ir	25	25	25	25
	1999	institution5	db	3	3	3	3
	1999	institution5	ir	1	1	1	1
	1999	institution6	ir	25	25	25	25
	1999	institution7	ir	20	20	20	20
	2000	institution1	db	50	150	20	100
	2000	institution2	db	15	30	10	20
	2000	institution3	db	17.5	35	10	25

grant_cube	year	institution	domain	avg_grants	sum_grants	min_grant	max_grant
	2000	institution4	db	5	5	5	5
	2000	institution4	ir	25	25	25	25
	2000	institution5	ir	4	8	3	5
	2000	institution6	ir	25	25	25	25
	2001	institution1	db	26.66	80	10	50
	2001	institution2	db	12.5	25	10	15
	2001	institution3	ir	12.5	25	10	15
	2001	institution4	db	15	30	5	25
	2001	institution4	ir	25	25	25	25
	2001	institution5	db	3	6	2	4
	2001	institution5	ir	2.5	5	2	3
	2001	institution7	ir	20	20	20	20
	2002	institution1	db	60	120	20	100
	2002	institution2	db	22.5	45	20	25
	2002	institution3	ir	15	30	5	25
	2002	institution4	db	5	5	5	5
	2002	institution4	ir	25	25	25	25
	2002	institution5	db	3	6	2	4
	2002	institution5	ir	4	12	2	6
	2002	institution6	ir	25	25	25	25
	2002	institution7	ir	20	20	20	20
	2003	institution1	db	47.5	190	10	100
	2003	institution2	db	15	15	15	15
	2003	institution3	ir	15	45	5	25
	2003	institution4	db	5	5	5	5
	2003	institution4	ir	25	25	25	25
	2003	institution5	db	5	10	4	6
	2003	institution5	ir	2.5	5	2	3
	2003	institution6	ir	47.5	95	25	70
	2003	institution7	ir	20	20	20	20
	2004	institution1	db	75	150	50	100
	2004	institution2	db	22.5	45	20	25
	2004	institution3	ir	17.5	35	10	25
	2004	institution4	db	15	30	5	25
	2004	institution4	ir	25	25	25	25
	2004	institution5	db	4	4	4	4
	2004	institution5	ir	3	3	3	3
	2004	institution6	ir	67.5	135	35	100
	2004	institution7	ir	25	25	25	25

Taulukko 6: Esimerkkikyselyn 3 vastausten tiivistelmä.

reference_cube	year	publisher	number_of_refs
	1995	publisher2	1
	1995	publisher4	1
	1996	publisher1	1
	1996	publisher3	1
	1996	publisher4	1
	1997	publisher1	1
	1997	publisher2	2
	1997	publisher3	2
	1997	publisher4	2
	1998	publisher1	2
	1998	publisher2	1
	1998	publisher4	2
	1999	publisher1	4
	1999	publisher2	2
	1999	publisher2	2
	1999	publisher4	3
	2000	publisher1	4
	2000	publisher2	2
	2000	publisher3	3
	2000	publisher4	4
	2001	publisher1	4
	2001	publisher2	4
	2001	publisher3	4
	2001	publisher4	5
	2002	publisher1	4
	2002	publisher2	2
	2002	publisher3	3
	2002	publisher4	4
	2003	publisher1	5
	2003	publisher2	5
	2003	publisher3	5
	2003	publisher6	6
	2004	publisher1	5
	2004	publisher2	4
	2004	publisher3	5
	2004	publisher4	6