XILtoSQL –
hierarkkisten kyselyiden semantiikka relaatiotietokannassa

Johanna Vainio

Tampereen yliopisto
Informaatiotieteiden yksikkö
Tietojenkäsittelyoppi
Pro gradu -tutkielma
Ohjaaja: Marko Junkkari
Toukokuu 2012

Tampereen yliopisto
Informaatiotieteiden yksikkö
Tietojenkäsittelyoppi
Johanna Vainio: XILtoSQL – hierarkkisten kyselyiden semantiikka relaatiotietokannassa
Pro gradu -tutkielma, 106 sivua, 25 liitesivua
Toukokuu 2012

Tässä tutkielmassa esitellään hierarkian käsitteen liittäminen relaatiomalliin, tutkimuksessa kehitetyt käsitteet hierarkian tulkinnalle ja XML-kyselykielellä XIL (XML Information Retrieval Language) tehtyjen kyselyiden kääntäminen SQL-kyselyiksi relaatiomalliin liitettyjen hierarkian käsitteiden avulla. XML-tietomallissa hierarkia on eksplisiittisesti mukana, mutta hierarkia on implisiittisesti mukana myös relaatiotietokannassa. Tämä tarkoittaa, että XML-tietomallissa hierarkia on kiinnitetty, mutta relaatiotietokannassa hierarkkisia näkymiä voidaan muodostaa relaatioiden osista ja niiden välisistä suhteista (ad hoc). Tutkielmassa relaatiotietokannan hierarkia muodostetaan automaattisesti kyselyn perusteella. Tämä takaa joustavamman kyselyn muotoilun, koska kyselyn tekijän ei tarvitse tuntea tietojen välisiä suhteita. Käännös kuvataan formaalista attribuuttikieliopin avulla. XIL-kyselyt käännetään SQL-kyselyiksi kääntämällä XIL-kysely aluksi yleistetyksi kyselyn monikkoesitykseksi ja monikkoesitys SQL-kyselyiksi. Yleistetty kyselyn monikkoesitys on kieliriippumaton esitys hierarkkisille *valitse*- ja *poimi*-kyselyille.

Avainsanat ja -sanonnat: kyselykieli, SQL, XML, relaatiotietokanta, tietomalli, relaatiomalli, hierarkkinen malli, attribuuttikielioppi.

Sisällys

1. Johdanto

Kyselykielet on perinteisesti suunnattu yhdelle tietomallille, jolloin kyselykielellä tehty kysely voidaan kohdistaa vain tähän tietomalliin pohjautuviin tietolähteisiin. Esimerkiksi SQL (Structured Query Language) [Chamberlin and Boyce, 1974] on alun perin suunniteltu relaatiotietokannoille, jotka pohjautuvat relaatiomalliin. Jos tietoa on tallennettuna eri muodoissa, tarvitaan myös useita kyselykieliä käsittelemään näitä tietoja. XML (Extensive markup language) [XML, 2008] pohjautuu hierarkkiseen tietomalliin ja sitä käsittelevien kyselykielien on kyettävä käsittelemään hierarkkista tietoa. Vallitseva lähestymistapa on, että kyselykielet, jotka käsittelevät sekä relaatiomuodossa että XML-muodossa olevaa tietoa, sisältävät erilliset operaatiot XML:lle ja relaatioille [Krishnaprasad *et al.*, 2004; Pal *et al*., 2004; Beyer *et al*., 2005]. Tilanteissa, joissa haluttu tieto on tallennettuna sekä relaatio- että XML-muotoon, tulee käyttää sekä relaatiotietokantaan sopivia operaatioita että XML:ään sopivia operaatioita. Työssä kuvataan, kuinka kysely voidaan toteuttaa käyttäjän kannalta vain XML:ään suunnatuilla operaatioilla, eli kuinka XML-kyselykielellä annettu kysely käännetään SQL:ksi ilman käyttäjän kontrollia.

1.1. Käsitteistö

Integroitaessa tietoa kahdesta tai useammasta erimuotoisesta tietolähteestä puhutaan tiedon integroinnista heterogeenisistä tietolähteistä. Vastakohtana tälle on tiedon integrointi homogeenisistä eli samanmuotoisista tietolähteistä. Toisaalta jos eri muodoissa oleva tieto yhtenäistetään, puhutaan tiedon saattamisesta homogeeniseen muotoon. Jos osa tiedosta esiintyy relaatiomuodossa ja osa XML-muodossa, ja nämä tietolähteet integroidaan esimerkiksi relaatiomuotoon, puhutaan heterogeenisten tietolähteiden yhtenäistämisestä homogeeniseen muotoon. (ks. [Junkkari, 2007] ja [Niemi *et al.*, 2002])

Web-sovelluksissa julkaistavilla ja jaettavilla tekstidokumenteilla, kuten muillakin dokumenteilla, on sisältö ja rakenne. Tekstidokumenttien tapauksessa sisältö tarkoittaa dokumentin tekstiä, eli sanoja ja sanoista muodostettuja lauseita. Dokumentin rakenteella tarkoitetaan loogista rakennetta, joka käsittää dokumentin osat ja näiden väliset suhteet, eli osien järjestyksen dokumentissa. Dokumentin looginen rakenne voi esiintyä implisiittisesti tai eksplisiittisesti. Merkkaamalla dokumentin looginen rakenne eksplisiittisesti mahdollistetaan rakenteen koneellinen käsittely. Käytetyin tapa dokumentin loogisen rakenteen merkkaamiseen on XML. [Lalmas, 2009] Dokumentteja, joiden rakenne on kuvattu formaalilla tavalla, kuten XML, kutsutaan myös rakenteisiksi dokumenteiksi. Dokumentteja, joiden rakennetta ei ole kuvattu formaalilla tavalla, kutsutaan rakenteettomiksi dokumenteiksi, vaikka niiden rakenne voi olla nähtävissä implisiittisesti (esim. lauserakenne).

Tiedon (datan) rakenteisuus voidaan jakaa rakenteiseen, puolirakenteiseen ja rakenteettomaan [Elmasri and Navathe, 2004]. Rakenteinen tieto on järjestettävissä noudattaen ennalta määrättyä rakennetta. Perinteiset tietokannat, kuten relaatiotietokannat, sopivat rakenteisen tiedon tallentamiseen

ja käsittelyyn. Tietokannassa tiedon kuvaus ja varsinainen tieto ovat tyypillisesti erillään. Tietokannan kuvauksessa tiedon muoto, suhteet, arvoalueet ja rajoitteet ovat tarkkaan määriteltyjä. Puolirakenteinen tieto ja rakenteinen dokumentti ovat läheistä käsitteellistä sukua. Näissä ei ole aina ennalta määrättyä rakennetta, tiedon kuvaus ja varsinainen tieto esiintyvät yhdessä, arvoalueita ei voida aina määrittää ja rakenteet voivat olla heterogeenisiä. XML sopii puolirakenteisen, mutta myös rakenteisen, tiedon tallentamiseen ja käsittelyyn. Rakenteeton tieto on tekstiä vailla rakennetta kuvaavia osia. Rakenteettoman tiedon ja rakenteettoman dokumentin käsitteet ovat lähellä toisiaan.

Yleisesti dokumentti voi olla teksti- tai datakeskeinen [Goldfarb & Prescod, 1988]. Tekstikeskeinen dokumentti koostuu yllä kuvatusta puolirakenteisesta tiedosta. Tekstikeskeisessä lähestymistavassa rakenteet kuvaavatkin lähinnä dokumentin sisäistä loogista rakennetta. Esimerkiksi kirjoilla on looginen rakenne. Kirjaan kuuluu muun muassa otsikko, lukuja ja alilukuja sekä näiden keskinäinen järjestys. Tämä looginen rakenne ei ole kuitenkaan yleinen ja kahden eri kirjan loogiset rakenteet voivat erota toisistaan huomattavasti. Datakeskeinen dokumentti koostuu rakenteisesta tiedosta. Siinä tiedon keskinäinen järjestys ei ole yleensä oleellinen, mutta tiedon rakenne on yleinen. EDI (electronic data interchange) on yksi esimerkki datakeskeisten dokumenttien käytöstä. EDI on tekniikka, jota käytetään dokumenttien, kuten laskujen ja tilausten, ja tiedon siirtämiseen tietojärjestelmien välillä. EDI-laskun sisältämällä tiedolla on yleinen rakenne. Siitä on löydyttävä muun muassa laskuttaja, laskutettava, tuotteet, summa ja eräpäivä. Laskun loogisella rakenteella, eli sillä esiintyykö vaikkapa laskuttaja ennen laskutettavaa, ei ole semanttista merkitystä. Perinteisesti relaatiotietokannat on suunnattu datakeskeiseen lähestymistapaan ja XML sekä data- että tekstikeskeiseen lähestymistapaan.

Datakeskeisille kyselyille on ominaista tietokannan ja tiedon rakenteen käyttö ja täsmälliset kyselyt. Datakeskeisessä kyselyssä voidaan hakea laskua, jonka lähettäjä on tietty laskuttaja. Vastauksena kyselyyn saadaan täsmälleen ne laskut, joiden lähettäjä on kyseinen laskuttaja. Tekstikeskeisissä kyselyissä voidaan hyödyntää dokumentin loogista rakennetta ja täsmälliset kyselyt eivät ole tarpeen täsmäytyksen ja relevanssilajittelun ansiosta. Tekstikeskeisessä kyselyssä voidaan hakea kirjan osaa, jossa käsitellään esimerkiksi relaatiotietokantojen muuttamista XML:ksi. Nyt vastauksena kyselyyn saadaan kirjojen osat, jotka täsmäävät parhaiten annettuihin hakusanoihin.

1.2. Tutkimuksen tausta ja kuvaus

Käyttötapauksessa, jossa osa halutuista tiedoista on tallennettu XML-muodossa ja osa relaatioina relaatiotietokantaan, voidaan toimia seuraavasti.

1. Kyselyn tekijä muodostaa XML-kyselykielellä kyselyn, joka kohdistetaan XML-muodossa olevaan tietoon ja relaatiotietokantaan sopivalla kyselykielellä kyselyn, joka kohdistetaan taas relaatiotietokantaan. Voidaan käyttää myös kieltä, jossa on omat operaatiot sekä XML:lle että relaatioille [Krishnaprasad *et al.*, 2004; Pal *et al.*, 2004; Beyer *et al.*, 2005]. Tässä pitää

huomioida, että tiedot voivat olla tallennettuna eri tietokantoihin eri tavalla ja kyselyjä muotoiltaessa on tunnettava kummassakin tietolähteessä olevan tiedon muoto.

2. XML voidaan integroida relaatiotietokantaan [Niemi *et al.*, 2009] tai toisinpäin, samalla yhtenäistäen eri tietokannoissa olleet tiedot. Tällöin haluttu tieto saadaan tietokannasta vain yhdellä kyselyllä. Olemassa oleva tieto joudutaan kuitenkin tallentamaan useampaan kertaan (tieto löytyy sekä XML- että relaatiomuodossa). Tiedon tallentaminen useampaan paikkaan vaatii tietenkin enemmän tilaa, mutta saattaa altistaa myös mahdollisille eheysrikkomuksille, koska päivitykset tulee kohdistaa useampaan paikkaan.
3. Kysely muodostetaan toiseen tietokannoista sopivaksi ja tämä kysely käännetään automaattisesti vastaamaan toiseen tietokantaan sopivaa kieltä. Kyselyn tekijä voi muun muassa muodostaa XML-kyselykielellä kyselyn, joka käännetään automaattisesti relaatiotietokantaan sopivalle kyselykielelle, kuten SQL. Tällöin kyselyn tekijän ei tarvitse tehdä kuin yksi kysely, mutta hän saa tarvittaessa vastauksen myös toisesta tietolähteestä. Tämä tapa sopii erityisesti tiedonhaun kehykseen, jossa pyritään löytämään kyselyyn parhaiten vastaavat tiedot.

Työssä käytetyt kyselykielet ovat SQL ja XIL (XML Information Retrieval Language) [Junkkari *et al.*, 2006]. SQL on relaatiotietokantoihin suunnattu, suurelta osin deklaratiivinen eli kuvaileva kyselykieli. SQL perustuu relaatioalgebraan, mutta se sisältää myös aggregointifunktioita, kuten tulosrivien lukumäärän tai keskiarvon selvittävät funktiot. Nämä aggregointifunktiot eivät sisälly relaatioalgebraan, joten SQL ylittää relaatioalgebran ilmaisuvoiman [Järvelin ja Niemi, 1999]. Ensimmäistä versiota SQL:stä kutsuttiin nimellä SEQUEL ja se kehitettiin 1970-luvun alkupuolella [Chamberlin and Boyce, 1974]. XIL on XML-kyselykieli, joka perustuu SEQUEL-kyselykieleen. XIL mahdollistaa kuitenkin sekä teksti- että datakeskeisen dokumenttien lähestymistavan, mutta SQL:n on perinteisesti ajateltu mahdollistavan vain datakeskeisen lähestymistavan. Tämä tarkoittaa, että XIL-kyselyssä voidaan käyttää sekä dokumentin loogista rakennetta, kuten järjestystä, että tiedon rakennetta. SQL-kyselyssä tulee taas käyttää tiedon rakennetta, mutta sen ei ole oletettu tukevan järjestystä. Dokumentin järjestys kuvataan XML:ssä hierarkiana. XIL-kyselyssä hierarkiassa kulkeminen kuvataan käyttämällä lineaarista polkua sekä SQL-tyyppisiä lohkorakenteita. Hierarkkisen käsittelyn siirtäminen relaatiotietokantoihin on työn keskeisin haaste.

Tutkielmassa keskitytään tapaukseen kolme eli XML-kyselykielen kyselyjen kääntämiseen relaatiotietokantakyselykielelle. Työn tarkoituksena on tutkia kyselyn tekijän muodostaman XIL-kyselyn automaattista kääntämistä SQL-kyselyiksi ja kuvata attribuuttikieliopin avulla kyselyjen kääntäminen formaalisti. Tarkastelun kohteena ovat ennen kaikkea polkuilmaukset ja niiden kääntäminen SQL:ksi, koska polkuilmaukset edustavat hierarkkista lähestymistapaa. Jotta polkuilmaukset voidaan kohdistaa haluttuun tietoon, on tietolähteen hierarkia oltava selvitettävissä. Vaikka SQL:n ei ole perinteisesti oletettu tukevan hierarkkista prosessointia, on Michael M David

[1992; 2003] kuitenkin osoittanut, että myös relaatiotietokantoja voidaan tulkita hierarkkisesti ja niistä voidaan muodostaa hierarkkisia näkymiä SQL:n avulla. XML-muodossa olevassa tiedossa hierarkia on mukana eksplisiittisesti, mutta se on implisiittisesti mukana myös relaatiotietokannassa [David 2003] ja toisaalta polkuilmaukset voidaan esittää SQL:n syntaksin avulla. Työssä relaatiotietokannan hierarkia muodostetaan automaattisesti kyselyn perusteella. Lisäksi tutkielmassa muodostetaan käsitteet relaatiotietokannan hierarkkiselle tulkinnalle.

Pelkästään syntaksiltaan oikeellisen SQL-kyselyn tuottaminen ei ole mielekästä, vaan käännetyn kyselyn on myös vastattava semantiikaltaan alkuperäistä kyselyä. Koska erimuotoisissa järjestelmissä tiedot ovat organisoituna eri tavalla, tulee käännettävän kyselyn semantiikka sovittaa kohteena olevan kyselykielen organisointitapoihin. Tutkimuksen tuloksena on toteutettu XILtoSQL-ohjelma, joka kääntää annetut XIL-kyselyt SQL-kyselyiksi noudattaen työssä muodostettua formalismia. Skenaariona tarkastellaan tilannetta, jossa lomamatkalle lähtevä haluaa tietoa kohdevaltiosta. Tällaista tietoa voi etsiä esimerkiksi Lonely Planet -tietokannasta (http://www.lonelyplanet.com/), The World Factbook -tietokannasta (https://www.cia.gov/library/publications/the-world-factbook/) tai Mondial-tietokannasta (http://www.dbis.informatik.uni-goettingen.de/Mondial/). Oletetaan näistä ensimmäisen olevan XML-muodossa ja toisen sekä kolmannen relaatiomuodossa. Kaikki nämä tietokannat sisältävät hieman erilaista tietoa valtioista. Kyselyn tekijä ei kuitenkaan tiedä, mistä tietokannasta haluttu tieto voisi löytyä. Vaihtoehtoina on kyselyn syöttäminen erikseen joka tietokantaan, jolloin kyselijän tulisi käyttää sekä XML-tietokantaan sopivaa kyselykieltä tai operaatioita että relaatiotietokantaan sopivaa kyselykieltä tai operaatioita, integroida ja harmonisoida nämä kolme tietokantaa, tai kyselijä voisi muodostaa yhden kyselyn ja järjestelmä hoitaisi automaattisesti kyselyn kääntämisen muihin tietokantoihin sopiville kielille.

Työ on organisoitu seuraavasti: luvussa 2 esitellään, kuinka erilaisilla tietomalleilla voidaan kuvata sama tietosisältö ja kuinka tieto organisoituu näissä tietomalleissa. Tavoitteena on osoittaa, että tietoa kuvaavat komponentit ovat tietomalleissa hyvin samanlaisia, mutta ne organisoituvat erilailla. Käännettäessä kysely eri tietomalleihin perustuvien kyselykielten välillä, on oleellista tuntea kummankin tietomallin tietoa kuvaavat komponentit ja näiden organisoituminen tietomallissa. Vaikka työssä keskitytään XML- ja relaatiomalliin, luvussa 2 esitellään myös hierarkkinen ja verkkotietomalli. Hierarkkinen malli esitellään, koska se määrittelee, kuinka tieto voidaan organisoida hierarkkisesti (vrt. XML). Verkkotietomalli esitellään puolestaan, koska se esittää tietoalkioiden yhteydet, joiden perusteella voidaan muodostaa erilaisia hierarkkisia näkymiä tiedosta. Näitä hyödynnetään XILtoSQL-käännöksessä. ER-mallilla esitetään tietokantariippumaton kuvaus tiedosta. Luvussa 3 esitellään tutkimuksessa käytetyt kyselykielet. Luvussa 4 kuvataan hierarkian käsitteen siirtäminen relaatiotietokantoihin. Aiempaa XML:n ja SQL:n yhdistämiseen liittyvää tutkimusta esitellään luvussa 5. Luvussa 6 esitellään tarkemmin oma tutkimus ja käännös kuvataan formaalisti luvussa 7 attribuuttikieliopin avulla. Luvussa 8 kuvataan XILtoSQL-ohjelma, luvussa 9 vertaillaan alkuperäistä

kyselyä (XIL) ja käännettyä kyselyä (SQL) sekä niiden tuottamia tuloksia. Lisäksi luvussa 9 esitellään tutkimuksen jatkokehityssuuntia. Tutkimuksen yhteenveto on luvussa 10.

2. Tietomalli

Tietokantajärjestelmä koostuu erilaisista abstraktiotasoista. Nämä voidaan jakaa fyysiseen tasoon, käsitteelliseen tasoon ja ulkoiseen tasoon ([Tsichritzis and Klug, 1978] viitattu [Elmasri and Navathe, 1989] kautta). Fyysiseen tasoon liittyy fyysinen kaavio, joka kertoo, miten tieto on organisoitu varsinaiseen tallennusmediaan, esimerkkinä tietuejärjestys. Käsitteelliseen tasoon liittyy käsitteellinen kaavio, joka on abstraktio kohdealueesta tietomallin kuvaamana. Vaikka fyysisessä kaaviossa kohdealueen tiedot voivat olla tallennettuna useampaan mediaan tai eri muodoissa, käsitteellinen kaavio kokoaa nämä tiedot yhdeksi kokonaisuudeksi esittäen tietokannan loogisen sisällön. Ulkoiseen tasoon liittyvät näkymät, joiden kautta päästään määrättyihin käsitteellisen tietokannan osiin.

Käsitteellisen tason esittävä käsitteellinen kaavio kuvataan usein jollakin tietomallilla. Tällainen tietomalli voi olla joko käsitteellinen, korkean tason tietomalli tai looginen määrättyyn tietokannanhallintajärjestelmään sopiva tietomalli. [Elmasri and Navathe, 1994] Usein tietokantaa suunniteltaessa mallintaja käyttää aluksi käsitteellistä tietomallia ja muuttaa tämän käsitteellinen tietomallin loogiseksi tietomalliksi. Käsitteellisiä tietomalleja ovat muun muassa ER-malli (Entity-Relationship) [Chen, 1976] ja SDM (Semantic Data Model) [Hammer and McLeod, 1981]. Loogisia tietomalleja ovat muun muassa relaatio- [Codd, 1970], verkko- [DBTG, 1971], hierarkkinen, olio- [Lecluce *et al.*, 1988] ja deduktiivinen malli [Liu, 1999].

Tiedot (data) ja sen väliset suhteet voidaan kuvata intensionaalisella ja ekstensionaalisella tasolla [Bunge, 1967]. Ekstensionaalisella tasolla eli esiintymätasolla käsitellään olioita, arvoja ja olioiden välisiä suhteita. Intensionaalisella eli kaaviotasolla käsitellään oliotyyppejä, attribuutteja ja oliotyyppien välisiä suhteita. Toisin sanoen intensio esittää tiedon kuvaavaa kaaviota ja ekstensio tähän kaavioon pohjautuvaa ilmentymää.

Tietomalli on matemaattinen formalismi, joka koostuu tietoa kuvaavasta merkintätavasta ja joukosta operaatioita, joilla tätä tietoa manipuloidaan [Ullman, 1988]. Elmasri ja Navathe [1994] eivät kuitenkin näe tietoa manipuloivien operaatioiden olevan välttämätön osa tietomallia. Yleisesti tietomalli tarjoaa tavan muodostaa yhtenäinen kokonaisuus tiedosta ja mahdolliset operaatiot tämän tiedon käsittelyyn. Se, miten tiedot on organisoitu kaavioihin, riippuu tietomallista sekä mallintajasta. Aliluvussa 2.1. esitellään ER-malli, koska se on käytetyin käsitteellinen tietomalli. Aliluvuissa 2.2.-2.4. esitellään relaatio-, verkko- ja hierarkkinen malli. Nämä mallit on valittu, koska työssä käytetyt kyselykielet pohjautuvat osittain näihin malleihin, mutta myös kuvaamaan tietomallissa esiintyviä tietoa kuvaavia komponentteja sekä näiden komponenttien organisoitumista eri tietomalleihin. Komponenttirakenteen kuvauksissa laatikot tarkoittavat käsitteitä ja vinoneliöt osa-kokonaisuus -suhteita. Jatkossa loogisesta tietomallista puhutaan vain tietomallina ja käsitteellisestä tietomallista käsitteellisenä tietomallina. Mallien kuvauksessa käytetään apuna kuvitteellista kohdealuetta (valtiot),

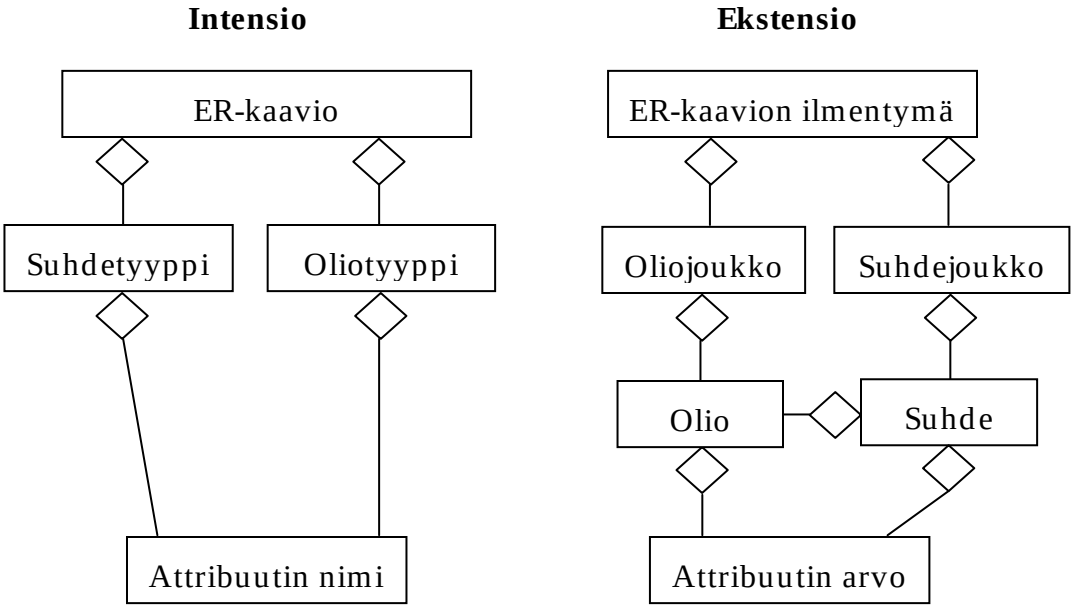
josta kuvataan valtioiden nimet, valtiomuodot, väkiluvut, valtioiden alueella olevien kaupunkien nimet sekä väkiluvut, valtioiden alueella olevien järvien nimet sekä pinta-alat ja valtioiden alueella olevien jokien nimet sekä pituudet.

2.1. ER-malli

ER-mallin [Chen, 1976] tarkoitus on kuvata kohdealue huomioimatta varsinaisen toteutuksen yksityiskohtia. ER-mallin peruskäsitteet ovat olio (entity), attribuutti (attribute) ja suhde (relationship). Olio on oliotyypin ilmentymä. Oliolla on ominaisuuksia eli attribuutteja, jotka kuvaavat sitä ja joiden arvojen avulla oliot eroavat toisistaan. Ominaisuuksilla voi olla yksi tai useampi arvo. Ominaisuutta, jolla on yksi arvo, sanotaan yksiarvoiseksi ja ominaisuutta, jolla on useampia arvoja, sanotaan moniarvoiseksi. Oliotyypillä on tunniste (ominaisuus tai ominaisuuksien joukko), jonka arvo erottaa oliotyypin ilmentymät toisistaan. Tämän tunnisteen arvo ei saa olla sama millään saman intension oliolla. Suhdetyypit kertovat, miten oliotyypit liittyvät toisiinsa. Yleensä suhdetyypit liittävät toisiinsa kaksi oliotyyppiä, jolloin suhdetyyppiä sanotaan binääriseksi. On kuitenkin mahdollista, että suhdetyyppi liittää toisiinsa useamman kuin kaksi oliotyyppiä. Suhdetyyppi voi liittyä samaan oliotyyppiin useasti, jolloin suhdetyyppiä kutsutaan rekursiiviseksi. Suhdetyyppeihin liittyy yleensä kardinaalirajoituksia. Nämä rajoitukset ovat 1:1, 1:n ja m:n. Rajoitus 1:1 tarkoittaa, että suhdetyypin ilmentymä liittää yhden oliotyypin ilmentymän yhteen oliotyypin ilmentymään. Rajoitus 1:n tarkoittaa, että suhdetyypin ilmentymä liittää yhden oliotyypin ilmentymän useaan ilmentymään yhdestä oliotyypistä. Rajoitus m:n tarkoittaa, että suhdetyypin ilmentymä liittää usean ilmentymän yhdestä oliotyypistä useaan ilmentymään yhdestä oliotyypistä. Kardinaalirajoituksia voidaan myös täsmentää esittämällä ylä- ja alarajat, kuten 0..1:n. Tämä tarkoittaa, että suhdetyypin ilmentymä liittää enintään yhden oliotyypin ilmentymän useaan ilmentymään yhdestä oliotyypistä. [Elmasri and Navathe, 1994]

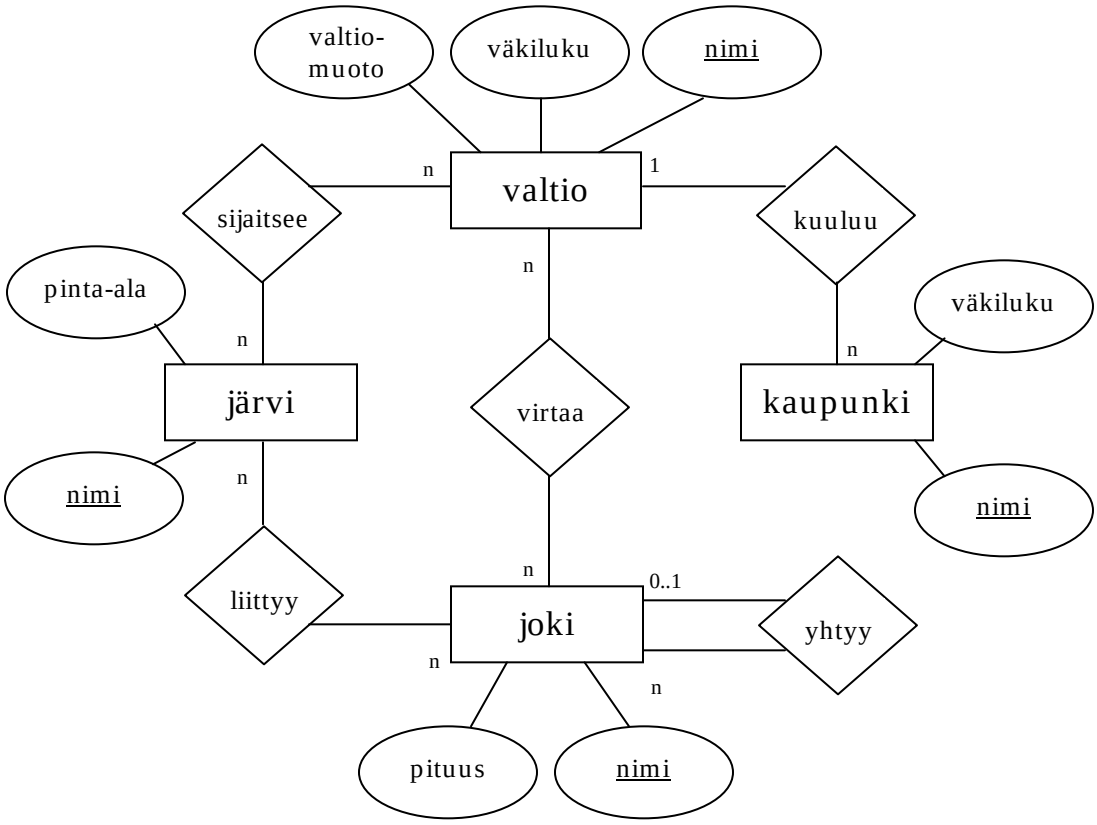
ER-mallissa ER-kaavio koostuu oliotyypeistä ja suhdetyypeistä. Oliotyypit koostuvat olion nimestä ja attribuuteista. ER-kaavion kuvaama ER-kaavion ilmentymä koostuu oliojoukoista ja suhdejoukoista. Oliot voivat liittyä toisiinsa suhteiden välityksellä. Tämä on esitetty kuvassa 1 osoittamalla intension ja ekstension sijoittuminen malliin.

ER-mallin tietoa kuvaavat komponentit



Kuva 1. ER-mallin tietoa kuvaavat komponentit intension ja ekstension tasoilla kuvattuina.

Edellä mainittu kohdealue on kuvattu ER-kaaviona kuvassa 2, missä nelikulmiot kuvaavat oliotyyppiä, vinoneliöt suhdetyyppiä, ovaalit oliotyyppien attribuutteja ja alleviivatut attribuutit tunnisteita. Tässä oliotyypit ovat *valtio*, *kaupunki*, *järvi* ja *joki*. Attribuutteja ovat *nimi*, *väkiluku*, *valtiomuoto*, *pinta-ala* ja *pituus*. Kuvan 2 suhteet tulkitaan niin, että *valtioon* kuuluu yksi tai useampia *kaupunkeja*. Toisaalta yksi *kaupunki* liittyy yhteen *valtioon*. *Valtiossa* voi sijaita yksi tai useampia *järviä* ja sama *järvi* voi sijaita yhden tai useamman *valtion* alueella. Valtiossa voi virrata yksi tai useampia *jokia* ja yksi *joki* voi virrata useamman *valtion* alueella. *Järveen* voi liittyä yksi tai useampi *joki* ja sama *joki* voi liittyä useampaan *järveen*. Lisäksi *joki* voi laskea enintään yhteen (0..1) *jokeen*, mutta samaan *jokeen* voi laskea useampia *jokia*.

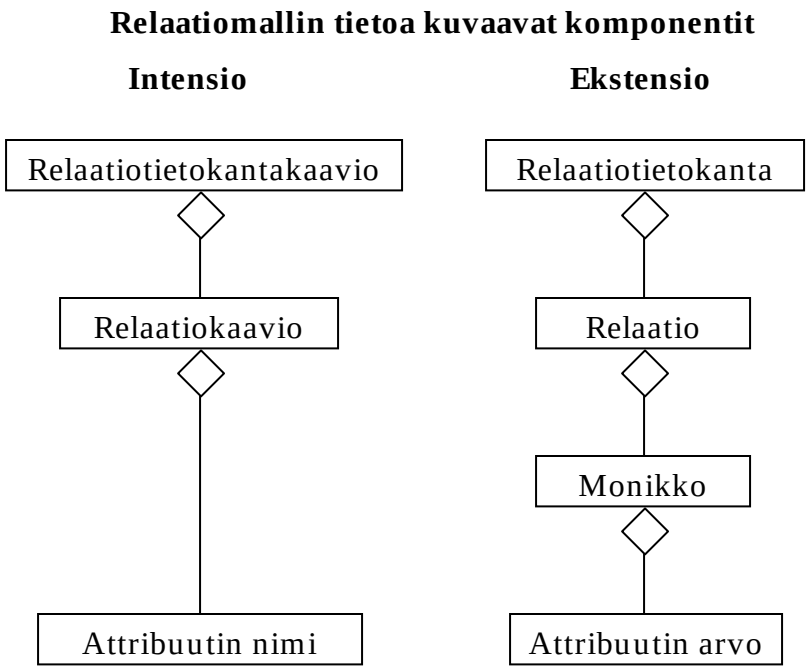


Kuva 2. ER-malli valtiotietokannasta.

2.2. Relaatiomalli

Käytetyin looginen tietomalli on relaatiomalli. Ullman [1988] arvioi, että relaatiomallin suosion suurin syy lienee mallin tuki yhtäaikaisesti sekä ilmaisuvoimaiselle että helpolle ja deklaratiiviselle kielelle. Mallin intensionaalisen tason peruskäsitteet ovat attribuutti ja relaatiokaavio. Relaatiokaavio koostuu relaation nimestä ja joukosta attribuutteja. Ekstensionaalisen tason peruskäsitteitä ovat relaatiot, monikot ja attribuuttien arvot. Monikko on sekvenssi attribuuttien arvoja ja relaatio on joukko monikkoja, jotka toteuttavat relaatiokaavion. Näiden attribuuttien arvot erottavat saman intension omaavat oliot toisistaan. [Elmasri and Navathe, 1994] Relaatiotietokanta perustuu relaatiotietokantakaavioon ja relaatiot perustuvat relaatiokaavioon. Kuvassa 3 on havainnollistettu relaatiomallin komponentteja intension ja ekstension tasoilla.

Formaalimmin esitettynä relaatiomallilla kuvattu relaatiotietokanta koostuu joukko-opillisista relaatioista. Joukko-opillinen relaatio on osajoukko attribuuttien arvojen karteesisesta tulosta. Relaatiokaavio voidaan esittää antamalla relaation nimi ja luettelemalla joukot, joita karteesisessa tulossa on käytetty.



Kuva 3. Relaatiomallin tietoa kuvaavat komponentit intension ja ekstension tasoilla kuvattuina.

Relaation monikoiden tulee olla erillisiä, mikä tarkoittaa, että mitkään kaksi relaation sisältämää monikkoa eivät saa sisältää samoja arvoja kaikille attribuuteille. Sitä attribuuttia (tai attribuutteja), joka erottaa monikot toisistaan kutsutaan superavaimeksi. Ehdokasavain on superavain, josta ei poistamalla attribuutteja muodostu enää superavainta. Näistä ehdokasavaimista valitaan pääavain (vrt. ER-mallin tunniste). Relaatioiden väliset suhteet esitetään pääavainten ja vierasavainten avulla. Siihen relaatioon, joka viittaa toiseen relaatioon, sijoitetaan viitattavan relaation pääavain attribuutiksi. Näitä relaatioon sijoitettuja attribuutteja kutsutaan vierasavaimiksi.

Tarkastellaan valtiorelaatiota, joka sisältää joukon valtio-olioita. Jokainen valtio-olio sisältää muun muassa ominaisuudet nimi, valtiomuoto ja väkiluku. Näiden ominaisuuksien arvot erottavat olion muista olioista. ER-mallista saadaan johdettua relaatiotietokantakaavio noudattamalla seuraavia sääntöjä. ER-mallin oliotyyppi vastaa relaatiomallin relaatiokaaviota tietyin rajoituksin. Oliojoukko voidaan esittää relaationa, jossa oliojoukon olio vastaa relaation monikkoa. Jos oliotyypin X ja oliotyypin Y välisen suhdetyypin Z kardinaalirajoitus on 1:n, lisätään oliotyyppiin Y oliotyypin X tunniste. Jos oliotyypin X ja oliotyypin Y välisen suhdetyypin Z kardinaalirajoitus on 1:1, voidaan päättää, lisätäänkö oliotyypin X tunniste oliotyyppiin Y vai toisinpäin. Oliotyypin X ja oliotyypin Y välisen suhdetyypin Z kardinaalirajoituksen ollessa n:m, esitetään suhdetyyppi Z relaatiokaaviona, jonka attribuutit ovat X:n tunniste ja Y:n tunniste. Nämä säännöt eivät ole täydelliset, mutta riittävät esimerkkinä käytetyn valtiotietokannan johtamiseksi ER-kaaviosta relaatiotietokantakaavioon. Sääntöjen tarkoitus onkin antaa yleiskuva siitä, kuinka ER-kaaviosta saadaan johdettua

relaatiotietokantakaavio. Tarkemmat säännöt löytyvät esimerkiksi Cheniltä [1976] ja Ullmanilta [1988].

Nyt kuvassa 2 esitetty ER-malli valtiotietokannasta voitaisiin kuvata relaatiokaavioina seuraavasti[1]:

valtio(nimi, valtiomuoto, väkiluku)

kaupunki(nimi, väkiluku, valtio_nimi)

järvi(nimi, pinta-ala)

joki(nimi, pituus, laskujoki)

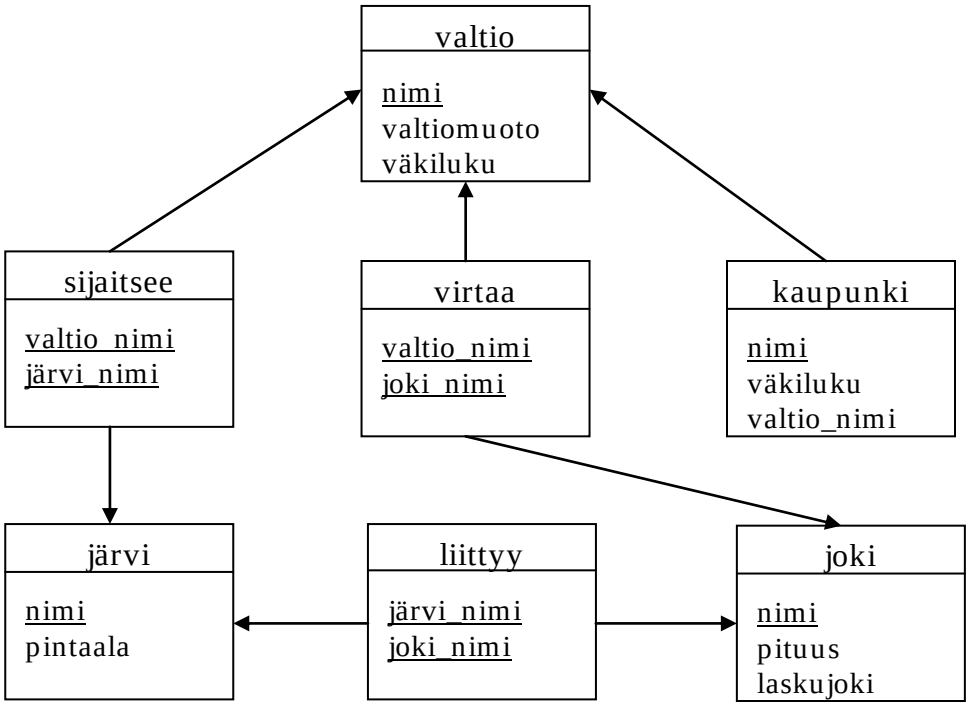
sijaitsee(valtio_nimi, järvi_nimi)

virtaa(valtio_nimi, joki_nimi)

liittyy(joki_nimi, järvi_nimi)

Oliotyypistä *kaupunki* on muodostettu relaatiokaavio liittämällä *valtio*-oliotyypin tunniste *kaupunki*-relaatiokaavioon. *Valtio*- ja *järvi*-, *valtio*- ja *joki*- sekä *järvi*- ja *joki*-oliotyyppien välisistä suhdetyypeistä on muodostettu relaatiokaaviot liittämällä relaatiokaavioihin suhdetyyppeihin liittyvien oliotyyppien tunnisteet. Relaation tunniste kuvataan alleviivauksella, toiseen relaatioon viittaava pääavain kuvataan katkoviivalla ja relaation tunnisteen ja toiseen relaatioon viittaavan pääavaimen yhdistelmä kuvataan kaksoisviivalla. Kuvassa 4 on relaatiotietokantakaavio kuvattu graafisesti. Tässä nelikulmiot kuvaavat oliotyyppiä, missä ensimmäisenä annetaan oliotyypin nimi ja tämän jälkeen oliotyypin attribuutit. Alleviivattu attribuutti tai attribuutit ovat oliotyypin tunniste. Nuolet kuvaavat oliotyyppien välistä suhdetta. Edellä mainittujen kaavioiden ilmentymät on kuvattu tauluina liitteessä 1.

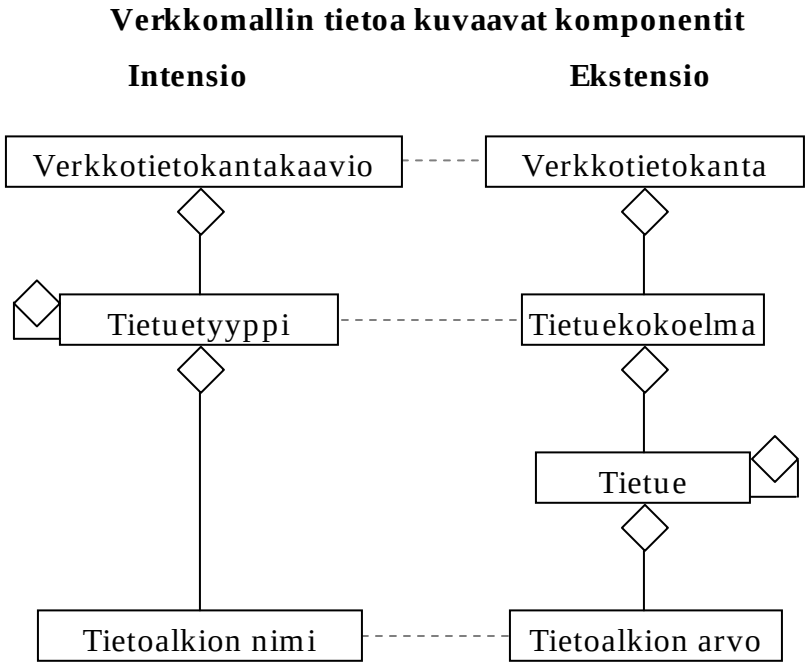
[1] Relaatiokaaviossa on käytetty skandinaavisia aakkosia (ö ja ä) ja väliviivaa (-) esityksen yhdenmukaistamiseksi. Sama käytäntö toistuu esimerkkinä käytetyissä esiintymissä ja kyselyissä läpi työn.



Kuva 4. Valtiotietokannan relaatiotietokantakaavio graafisesti kuvattuna.

2.3. Verkkomalli

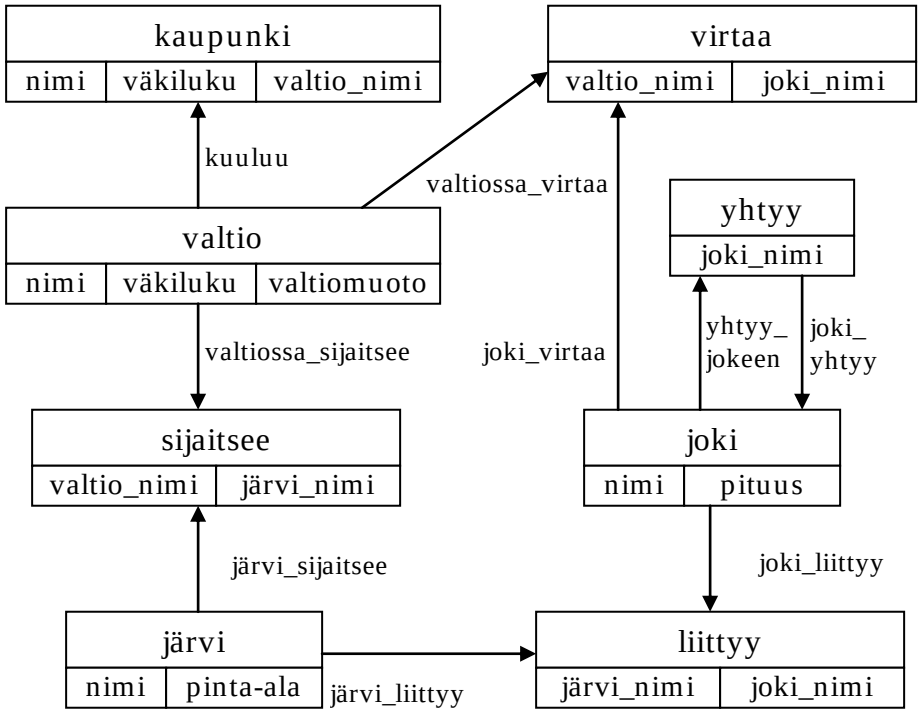
Ensimmäisenä verkkomallin esitteli CODASYL (Conference on Data Languages) komitea [DBTG, 1971]. Verkkomallin peruskäsitteitä ovat tietuekokoelma, tietue ja tietoalkion arvo sekä tietuetyyppien väliset suhteet. Verkkotietokanta koostuu tietuekokoelmista. Tietuekokoelma on joukko saman intension omaavia tietueita. Tietueeseen liittyy yleensä tietoalkion arvoja, mutta ne eivät ole välttämättömiä. Toisaalta yhteen tietueeseen saattaa liittyä useita saman tietoalkion arvoja. Tietuetyyppi kuvaa tietuekokoelman ja koostuu nimestä ja tietoalkioista. Tietoalkiot taas koostuvat nimestä ja tietotyypistä. [Navathe and Elmasri, 1994] Kuvassa 5 on esitetty tämä osoittamalla myös intension ja ekstension sijoittuminen malliin. Nyt tietuetyyppi voisi tarkoittaa valtiota. Sen nimi on valtio ja sisältämät tietoalkiot ovat valtion nimi, valtiomuoto ja väkiluku. *Valtio*-tietuetyyppi kuvaa kokoelman valtiotietueita. Tietuetyyppien välisiä suhteita nimitetään verkkomallissa kokoelmatyypeiksi. Nämä kokoelmatyypit kuvaavat 1:n-suhteet kahden tietuetyypin välillä. Kokoelmatyyppi koostuu nimestä, omistajatietueesta ja jäsentietueista. Kokoelmatyypin esiintymä voi koostua yhdestä omistajatietueesta, mutta useasta jäsentietueesta. Toisin sanoen näitä jäsentietueita voi olla monta tai ei yhtään. Kokoelmatyyppi ei mahdollista n:m-suhteita, vaan nämä korvataan kahdella suhteella ja täydentävällä tietuetyypillä. Esimerkiksi *valtio*-tietuetyypin ja *joki*-tietuetyypin välinen n:m-suhde korvataan vaikkapa täydentävällä tietuetyypillä *virtaa* ja lisäämällä kokoelmatyyppi kuvaamaan *valtion* ja *virtaa*-tietuetyypin suhdetta sekä kokoelmatyyppi kuvaamaan *joen* ja *virtaa*-tietuetyypin suhdetta.



Kuva 5. Verkkomallin tietoa kuvaavat komponentit intension ja ekstension tasoilla kuvattuina.

Verkkomallin voidaankin sanoa olevan ER-malli, jossa suhteet ovat binäärisiä sekä 1:n-muodossa [Ullman, 1988]. Tällöin verkkomalli voidaan kuvata yksinkertaisena suunnattuna graafina, joka estää itseissilmukoiden ja rinnakkaisten kaarien esiintymisen.

Nyt valtiotietokantaa kuvaavasta ER-kaaviosta saadaan johdettua verkkotietokantakaavio käyttämällä seuraavia sääntöjä. Jokainen oliotyyppi vastaa tietuetyyppiä ja oliotyyppiin liittyvät attribuutit tietuetyyppiin liittyviä tietoalkioita. Jokaista ei-rekursiivista ja binääristä 1:1- ja 1:n-suhdetyyppiä vastaa yksi kokoelmatyyppi. 1:1-suhdetyypissä voidaan valita, kumpi kokoelmatyyppiin liittyvistä tietuetyypeistä on omistajatietue ja kumpi jäsentietue. Oliotyypin X ja oliotyypin Y välisen suhdetyypin Z kardinaalirajoituksen ollessa 1:n, suhdetyyppiä Z vastaavan kokoelmatyypin Z' omistajatietueeksi asetetaan oliotyyppiä X vastaava tietuetyyppi X' ja jäsentietueeksi oliotyyppiä Y vastaava tietuetyyppi Y'. Binäärinen n:m suhdetyyppi Z oliotyyppien X ja Y välillä kuvataan lisäämällä täydentävä tietuetyyppi F ja kaksi kokoelmatyyppiä Z' ja Z''. Kumpaankin kokoelmatyyppiin asetetaan omistajatietuetyypiksi täydentävä tietuetyyppi F ja Z':n jäsentietueeksi X' ja Z'':n jäsentietueeksi Y'. Rekursiivinen 1:1 tai 1:n suhdetyyppi Z, joka liittyy yhteen oliotyypin X, kuvataan lisäämällä täydentävä tietuetyyppi F ja kaksi kokoelmatyyppiä Z' ja Z''. Kokoelmatyyppiin Z' asetetaan omistajatietueeksi oliotyyppiä X vastaava tietuetyyppi X' ja jäsentietueeksi täydentävä tietuetyyppi F. Kokoelmatyyppiin Z'' taas asetetaan omistajatietueeksi täydentävä tietuetyyppi F ja jäsentietueeksi tietuetyyppi X'. Tämä ohjeistus ei ole täydellinen, mutta riittää esimerkkinä käytetyn tietokannan kuvaavan ER-kaavion muuttamiseen verkkotietokantakaavioksi sekä antamaan yleiskuva tästä. Tarkempi kuvaus löytyy esimerkiksi Cheniltä [1976] sekä Elmasrilta ja Navathelta [1994].



Kuva 6. Valtiotietokanta verkkomallilla kuvattuna.

Valtiotietokanta on kuvattu verkkomallilla kuvassa 6. Nelikulmiot kuvaavat tietuetyyppiä, missä annetaan ensiksi tietuetyypin nimi ja tämän jälkeen tietuetyypin tietoalkioiden nimet. Nuolet kuvaavat kokoelmatyyppejä, jossa nuoli lähtee omistajatietuetyypistä ja osoittaa jäsentietuetyyppiin. On syytä huomata, että tämä vastaa kuvassa 4 esitettyä relaatiotietokantakaaviota sillä erotuksella, että itseissilmukka *joki*-oliosta on poistettu korvaamalla se *yhtyy*-tietuetyypillä ja kahdella kokoelmatyypillä.

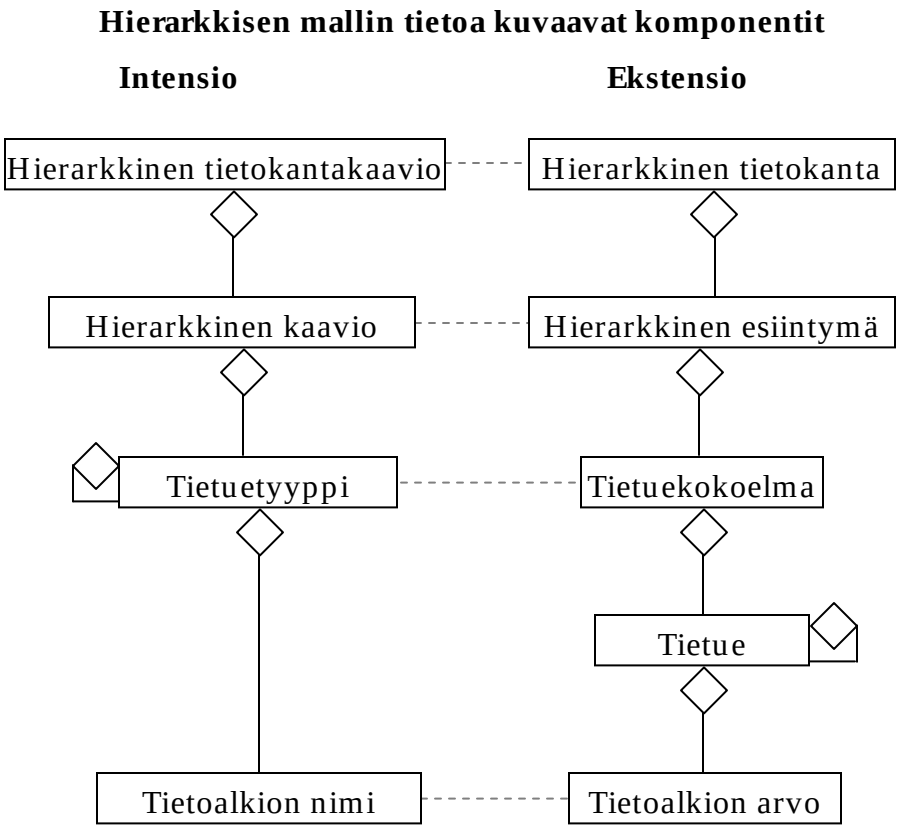
2.4. Hierarkkinen malli

Hierarkkinen malli kehitettiin mallintamaan sitä tiedon hierarkkista organisoitumista, mitä käsitteellisissä jäsennyksissä esiintyy. Tyypillisiä hierarkkisia suhteita käsitteiden välillä ovat is-a ja part-of -suhteet (ks. [Junkkari, 2005]). Hierarkiat, kuten eläinten ja kasvien tieteellinen luokittelu, auttavat ihmisiä ymmärtämään maailmaa. Hierarkkinen malli esittää tiedon hierarkkista organisoitumista luonnollisella tavalla mallinnettavan tiedon välisten suhteiden ollessa luonteeltaan hierarkkisia, mutta tilanne voi olla toinen mallinnettavan tiedon välisten suhteiden ollessa luonteeltaan ei-hierarkkisia. [Navathe and Elmasri, 1994]

Ullman [1988] kuvaa hierarkkisen mallin verkoksi, joka on kokoelma puita, jossa kaikki suhteet osoittavat vanhemmasta lapseen. Hierarkkisen mallin peruskäsitteet ovat samat kuin verkkomallin.

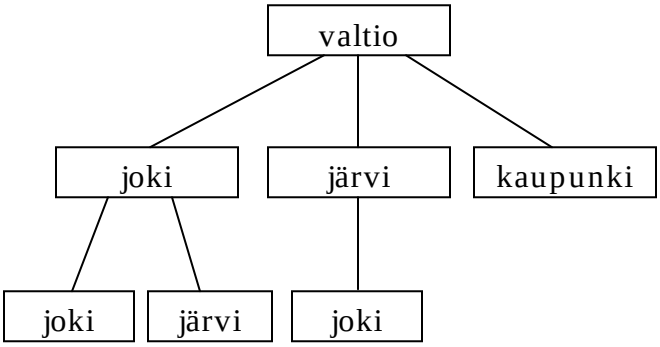
Näitä ovat tietuekokoelma, tietue ja tietoalkion arvo sekä tietuetyyppien väliset suhteet. Hierarkkisen mallin tietoa kuvaavia komponentteja ei voi kuitenkaan esittää täysin samalla tavalla kuin kuvassa 5. Syy tähän selviää myöhemmin tässä aliluvussa. Hierarkkisen mallin tietoa kuvaavat komponentit on kuvattu kuvassa 7.

Tietuetyyppien välisiä suhteita kuvataan vanhempi-lapsi -suhdetyypillä, joka on 1:n-suhde vanhemmasta lapseen. Näin tietuetta (tietuetyyppiä), josta suhde (suhdetyyppi) lähtee, kutsutaan vanhemmaksi ja tietuetta (tietuetyyppiä), johon suhde osoittaa, kutsutaan lapseksi. Suhteeseen liittyy aina yksi vanhempitietue ja mahdollisesti useita saman intension omaavia lapsitietueita. Jos solmulla X on lapsisolmu Y ja solmulla Y on lapsisolmu Z, niin solmun Z sanotaan olevan X:n jälkeläinen ja X:n sanotaan olevan Z:n isovanhempi. Hierarkkisella kaaviolla on seuraavat ominaisuudet: Kaaviolla on yksi juuritietuetyyppi (tietuetyypillä ei ole vanhempitietuetyyppiä). Jokainen tietuetyyppi juurta lukuun ottamatta on lapsitietuetyyppi yhdessä vanhempi-lapsi -suhdetyypissä. Jokainen tietuetyyppi voi olla vanhempitietuetyyppi useammassa vanhempi-lapsi -suhdetyypissä. Tietuetyyppiä, joka ei ole vanhempi missään vanhempi-lapsi -suhdetyypissä, kutsutaan lehdeksi. [Navathe and Elmasri, 1994]

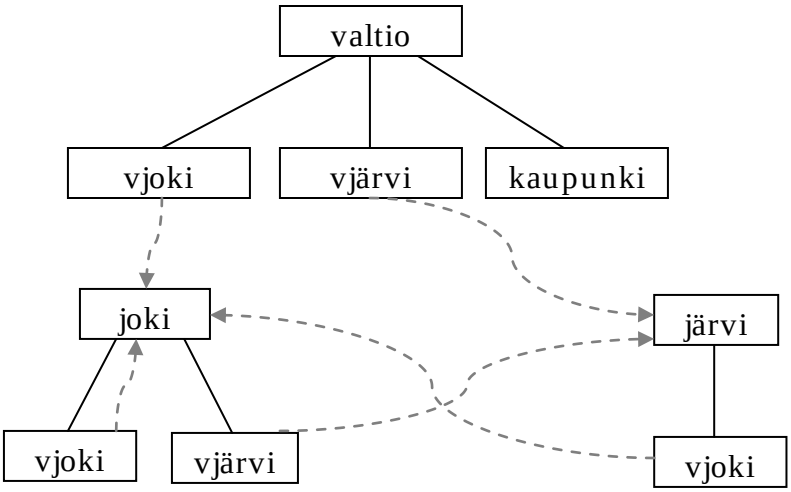


Kuva 7. Hierarkkisen mallin tietoa kuvaavat komponentit intension ja ekstension tasoilla kuvattuina.

Intuitiivisesti valtiotietokannassa olisi luonnollista ajatella valtio hierarkian ylimmäksi osaksi eli juureksi, koska se on kohdealueen laajin käsite. Tästä seuraisi, että järvi, joki ja kaupunki olisivat valtion lapsia. Lisäksi koska tästä hierarkiasta ei vielä nähdä, miten järvet ja joet liittyvät toisiinsa, pitää joki asettaa järven lapseksi ja järvi ja joki joen lapsiksi. Tämä hierarkia on esitetty kuvassa 8. Näkemys kuitenkin rikkoo hierarkkisen kaavion ominaisuuksia, sillä tietuetyypit *joki* ja *järvi* ovat lapsia useammassa suhdetyypissä. Lisäksi on syytä huomioida, että samat tiedot saattavat esiintyä useampaan kertaan varsinaisissa hierarkkisissa esiintymissä, koska joki tai järvi saattaa sijaita useamman valtion alueella. Hierarkkisen kaavion ominaisuuksien rikkominen ja saman tiedon esiintyminen useaan kertaan saadaan poistettua virtuaalisten tietuetyyppien avulla. Virtuaaliset tietuetyypit ovat osoittimia, jotka osoittavat varsinaiseen tietuetyyppiin [Navathe and Elmasri, 1994]. Näin varsinaisen tietuetyypin X ja siihen viittaavan virtuaalisen tietuetyypin VX välillä on virtuaalinen suhdetyyppi, jossa X on virtuaalinen vanhempi ja VX virtuaalinen lapsi. Näihin virtuaalisiin suhteisiin ja suhdetyyppeihin pätevät samat säännöt kuin yllä kuvattuihin suhdetyyppeihin. Kuvan 8 esittämä hierarkkinen kaavio on täydennetty virtuaalisilla tietuetyypeillä ja suhteilla kuvassa 9. Kuvan avulla onkin helpompi ymmärtää kuvassa 7 esitetyt hierarkkisen mallin tietoa kuvaavat komponentit. Varsinainen hierarkkinen tietokantakaavio koostuukin yhdestä tai useammasta hierarkkisesta kaaviosta. Näitä hierarkkisia kaavioita on kuvassa 9 kolme kappaletta, joiden juuret ovat *valtio*, *joki* ja *järvi*. Hierarkkiset kaaviot koostuvat tietuetyypeistä, jotka voivat olla virtuaalisia tai tavallisia tietuetyyppejä. Tietuetyypit koostuvat tietoalkioista.

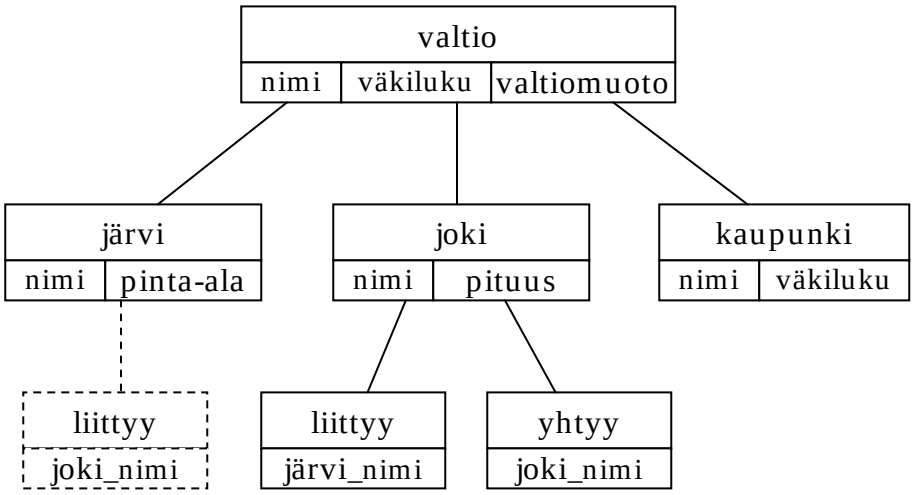


Kuva 8. Ensimmäinen, mutta hierarkkisen kaavion ominaisuuksia rikkova, versio hierarkkisesta tietokantakaaviosta.



Kuva 9. Toinen versio hierarkkisesta tietokantakaaviosta. Hierarkkisen kaavion ominaisuuksien rikkoutuminen on korjattu käyttämällä virtuaalisia tietuetyyppejä.

Elmasri ja Navathe [1994] esittävät kuitenkin vielä yhden vaihtoehdon hierarkkisen tietokantakaavion muodostamiseen, niin että tietokantakaavio koostuu yhdestä hierarkkisesta kaaviosta. Nämä säännöt on tarkoitettu ER-kaavion muuttamiseksi hierarkkiseksi tietokantakaavioksi. Ensinnäkin jokainen oliotyyppi vastaa tietuetyyppiä. Toiseksi m:n suhteisiin suhtaudutaan kuten 1:n suhteisiin, mikä tarkoittaa, että samat tiedot saattavat esiintyä hierarkkisessa esiintymässä useasti. Kolmanneksi hierarkkisen kaavion ominaisuuksia ei rikota, jolloin esimerkkitietokannassamme joen liittyminen järveen ja joen yhtyminen jokeen pitää esittää kuvasta 8 poikkeavalla tavalla. Tämä ratkaistaan lisäämällä tällaiseen suhteeseen toisen tietuetyypin alle niin sanottu toissijainen (subordinate) tietuetyyppi. Kuvassa 10 on esitetty tällainen yhdestä hierarkiakaaviosta koostuva hierarkkinen tietokantakaavio, missä *joen* ja *järven* liittyminen toisiinsa on kuvattu toissijaisella tietuetyypillä *liittyy* ja *joen* yhtyminen *jokeen* kuvattu toissijaisella tietuetyypillä *yhtyy*. Kuvassa 10 on lisätty toissijainen tietuetyyppi *liittyy* katkoviivoilla kehystettynä järvitietuetyypin alle kuvaamaan sitä, että mallintaja olisi voinut lisätä *liittyy*-tietuetyypin myös *järvi*-tietuetyypin alle.



Kuva 10. Kolmas versio hierarkkisesta tietokantakaaviosta. Hierarkkisen kaavion ominaisuuksien rikkoutuminen on korjattu käyttämällä toissijaisia tietuetyyppejä.

2.5. XML-tietomalli

XML on SGML-kielestä (Standard Generalized Markup Language) johdettu merkkauskieli. XML-kieltä voidaan käyttää myös metakielenä määrittämään uusia merkkauskieliä. XML-tietomalli on loogisesti hierarkkinen malli, joka voidaan fyysisesti tallentaa eri tietokantoihin. Tavallisesti XML-tieto tallennetaan tekstimuodossa, jolloin XML on laite- ja ohjelmistoriippumattomassa muodossa.

XML-tietomalleja ovat XML Information Set [Infoset, 2004], XPath 1.0 -tietomalli [XPath, 1999], DOM-malli [DOM, 1998], [DOM, 2000] sekä XPath 1.0 ja XQuery 2.0 -tietomalli [XDM, 2010]. Nämä mallit kuvaavat dokumentin rakennetta ja käsittelyä. Dokumentti kuvataan jokaisessa mallissa puuna, mutta rakenne saattaa olla erilainen. [Salminen and Tompa, 2001] Seuraava kuvaus ei kuvaa suoraan mitään näistä tietomalleista, mutta esittää näiden mallien idean yksinkertaistettuna.

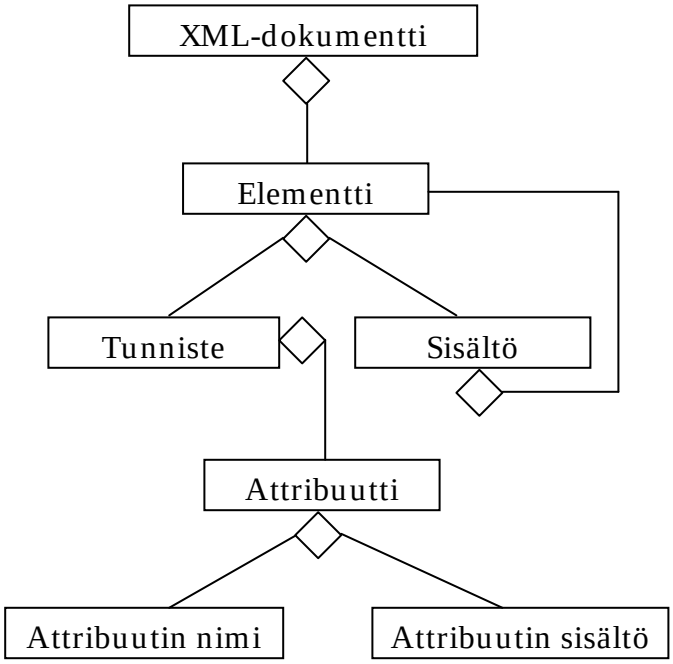
XML-tieto tallennetaan XML-dokumenttiin, joka muodostaa puurakenteen. XML-dokumentti alkaa juurielementistä ja päättyy lehtielementteihin. XML-dokumentti kuvaa tiedon loogisen rakenteen, kuten järjestyksen. Elementit koostuvat tunnisteesta (tag) ja sisällöstä. Elementti alkaa alkutunnisteella (<tunniste>) ja päättyy lopputunnisteella (</tunniste>). Kulmasulkeiden sisäinen osa on elementin nimi ja alku- ja lopputunnisteen väliin jäävä osuus on elementin sisältö. Lisäksi alkutunniste voi sisältää attribuutteja. Elementin sisältö voi koostua tekstistä ja/tai muista elementeistä. Elementtien tarkoitus on olla itseään kuvaavia, jolloin elementin tunniste, tai tarkemmin elementin nimi, kuvaa elementin sisällön intension ja toisaalta elementin sisältö kuuluu elementin tunnisteen ekstensioon.

Tämä on esitetty kuvassa 11. XML-tietomallin tietoa kuvaavat komponentit eroavat muiden tietomallien tietoa kuvaavista komponenteista pääsääntöisesti siinä, että XML-dokumentti ja elementit sisältävät sekä intension että ekstension. Käyttämällä DTD:tä (Document Type Definition) tai XML-

skeemaa voitaisiin kuvaan 11 lisätä erillinen intensio. XML-dokumentti ja elementit sisältäisivät kuitenkin edelleen sekä intension että ekstension.

XML-tietomallin tietoa kuvaavat komponentit

Intensio / Ekstensio



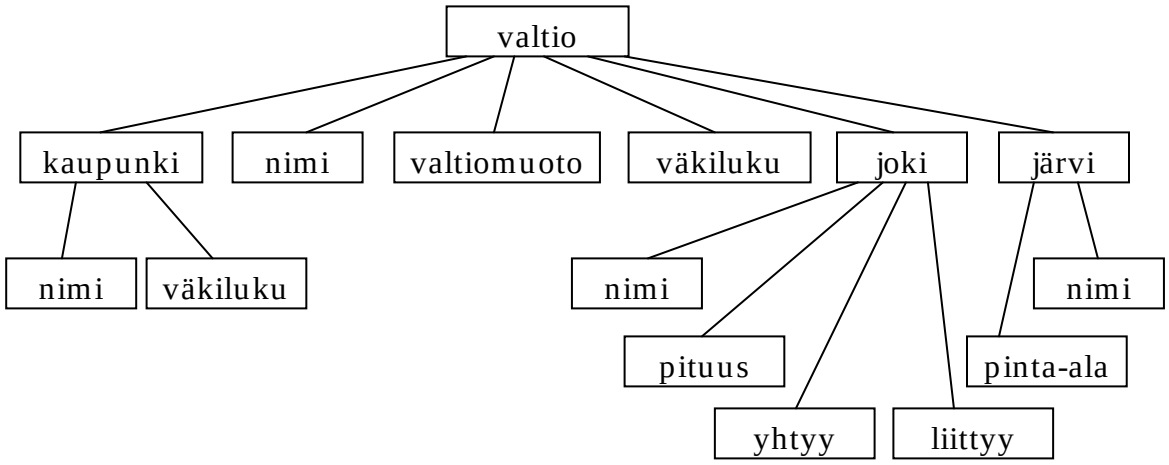
Kuva 11. Pelkistetyn XML-tietomallin tietoa kuvaavat komponentit intension ja esktension tasoilla kuvattuina.

XML-dokumentilla on samoja ominaisuuksia kuin hierarkkisella kaaviolla. Dokumentilla on yksi juurielementti, jolla ei ole vanhempielementtiä. Jokaisella elementillä juurta lukuun ottamatta on yksi vanhempielementti. Kuitenkin jokaisella elementillä voi olla useita lapsielementtejä. Elementtiä, jolla ei ole lapsia, kutsutaan lehdeksi. Hierarkkisesta mallista poiketen elementin ominaisuudet voidaan esittää attribuutteina tai toisina elementteinä. Siitä, pitäisikö elementin ominaisuus kuvata attribuuttina vai toisena elementtinä, ei ole olemassa yksiselitteisiä sääntöjä, mutta usein attribuuttien ymmärretään olevan elementtiä täydentävää tietoa, kuten elementin id. Attribuuttien käyttöä kehotetaan välttämään sisällön esittämisessä eikä niitä ole käytetty kuvattaessa valtiokohdealue XML-muodossa. Valtioelementeille voidaan tarvittaessa lisätä esimerkiksi *id*-attribuutti seuraavasti: *<valtio id=”1”><nimi>Suomi</nimi>…</valtio> ja <valtio id=”2” ”><nimi>Ruotsi</nimi>…</valtio>*.

```
<valtio>
 <nimi>Suomi</ nimi>
 <valtiomuoto>tasavalta</ valtiomuoto>
 <väkiluku>5 391 699</ väkiluku>
 <kaupunki>  … </ kaupunki>
 <järvi> … </ järvi>
 <joki>
  <nimi>Torniojoki</ nimi>
  <pituus>510</ pituus>
  <liittyy>Torniojärvi</ liittyy>
 </ joki>
 …
</ valtio>
```

Kuva 12. XML-dokumentin esiintymä.

XML-dokumentissa juurielementti sisältää dokumentin kaikki muut elementit. Vastaavasti jokainen dokumentin alipuun juuri sisältää sen alla olevat elementit. Kuvasta 12 nähdään, että valtioelementti sisältää omat ominaisuutensa ja lisäksi siihen liittyvät kaupunki-, järvi- ja jokielementit sekä näiden ominaisuuksia kuvaavat elementit. Valtiokohdealueesta muodostettu XML-kaavio on kuvattu puurakenteena kuvassa 13.



Kuva 13. Valtiokohdealue XML-dokumentin kuvaavana puuna.

Kuten jo aiemmin on todettu, XML sopii sekä datakeskeiseen että tekstikeskeiseen lähestymistapaan [Goldfarb & Prescod, 1988]. Datakeskeisessä lähestymistavassa tunnisteet kuvaavat tiedon rakennetta. Tekstikeskeisessä lähestymistavassa tunnisteet kuvaavat dokumentin loogista rakennetta, kuten järjestystä. Lisäksi datakeskeisessä lähestymistavassa tunnisteiden voidaan olettaa kuvaavan elementtien sisältöä tarkemmin kuin tekstikeskeisessä lähestymistavassa. Kuvassa 12 kuvattu dokumentti on datakeskeinen. Dokumentin ilmentymät ovat rakenteeltaan hyvin lähellä toisiaan ja

tunnisteet kuvaavat elementtien sisällön tarkasti. Elementtien lapsielementtien järjestyksellä ei ole myöskään merkitystä. Dokumentteja saattaa kuitenkin olla vaikea jakaa teksti- ja datakeskeisiin. Jos valtiodokumentti sisältäisi vielä elementin nimeltään kuvaus, jonka sisältö olisi vapaamuotoinen kuvaus dokumentin valtiosta, ei dokumentti olisi enää selvästi datasuuntautunut, vaan sisältäisi sekä data- että tekstisuuntautuneita elementtejä.

3. Kyselykieli

Luku 2 käsitteli tietomalleja tiedon kuvauksen näkökulmasta. Tätä kuvattua tietoa tarvitsee myös käsitellä. Tästä syystä tietomallin katsotaan usein [Elmasri and Navathe, 1989] tai aina [Ullman, 1988] sisältävän operaatiot tietomallin avulla kuvatun tiedon käsittelyyn. Tietoa voitaisiin tietysti käsitellä ohjelmointikielillä, mutta tällöin tietoon pääsisi käsiksi vain ohjelmointitaitoinen henkilö. Tarvitaan siis jokin muu lähestymistapa. Vastaus tähän ovat kysely-, tiedonmäärittely- ja tiedonkäsittelykielet, joiden tarkoituksena on tehdä pääsy tietoon helpommaksi [Ullman, 1988]. Kyselykieli on tietojen hakemiseen ja haetusta tiedosta johdetun uuden tiedon muodostamiseen tarkoitettu kieli. Tiedonmäärittelykieli on tarkoitettu tietomallin luomien kaavioiden määrittelemiseen ja tiedonkäsittelykieli on tarkoitettu päivittämään tietoa näihin kaavioihin. [Samet, 1981] [Date, 1989] Vaikka SQL viittaa nimensä mukaan kyselykieleen, on se itse asiassa myös tiedonmäärittely- ja tiedonkäsittelykieli. Tässä luvussa keskitytään esiteltyjen kielten kyselykieliominaisuuksiin.

Kyselykielet voidaan lajitella sen mukaan, mihin tietomalliin ne on suunniteltu. XML-kyselykielet voidaan lisäksi jakaa sen mukaan, ovatko ne suunnattu tekstikeskeisiin, datakeskeisiin vai kumpiinkin dokumentteihin. Tekstikeskeisessä dokumentissa ei tiedon rakenne ole välttämättä tiukka. Tällaisissa dokumenteissa kaaviot kuvaavat lähinnä dokumentin loogista rakennetta, kuten dokumentin osien keskinäistä järjestystä. Tällöin käytetyllä kyselykielellä on kyettävä ilmaisemaan osien järjestys. Datakeskeisissä dokumenteissa sen sijaan tiedon rakenne on tiukka. Tämä johtaa muun muassa siihen, että saman kaavion dokumentit vastaavat rakenteeltaan toisiaan. [Fuhr and Großjohann, 2001] Relaatiomallin kuvaaman tiedon on perinteisesti ajateltu olevan datakeskeistä ja hierarkkisen mallin kuvaaman tiedon joko data- tai tekstikeskeistä tai sekoitus kumpaakin.

3.1. Varhaiset kyselykielet

Varhaisille tietomalleille oli ominaista, että niillä ei ollut varsinaista itsenäistä kyselykieltä, vaan kyselykielen ilmaukset upotettiin isäntäkieleen kuten COBOLiin tai PASCALiin [Elmasri and Navathe, 1989]. Näin on myös verkkotietomallin kohdalla. CODASYL käytti COBOLia verkkomallin kyselykielen ilmausten upottamiseen [DBTG, 1971]. Myös verkkomalliin liittyy tiedonmäärittely- ja käsittelykieli, mutta keskitymme tässä vain kyselykielen piirteisiin.

Kyselykielen keskeinen tehtävä verkkomallissa on löytää ja palauttaa halutut tietueet. Nämä on toteutettu komennoilla get ja find. Verkkotietokantajärjestelmä ja isäntäkieli ovat kaksi eri ohjelmistoa ja viestintä näiden välillä suoritetaan rajapinnan kautta. Verkkokyselykielen komennot ovat tietue kerrallaan -komentoja, joten isäntäkielessä on pystyttävä säilyttämään tieto kullakin hetkellä käsiteltävästä tietueesta. Lisäksi isäntäohjelmassa on oltava muuttujat eri tietuetyyppien tietueille, jotta tietueita voidaan käsitellä. Kuvassa 6 on esitetty valtiotietokanta verkkomallilla. Haluttaessa tietää,

missä maassa Helsinki sijaitsee, kysely voidaan tehdä kuvassa 14 esitetyllä tavalla. Ennen kuvan 14 kyselyä on verkkomallin tietuetyyppi *kaupunki* ja tietoalkiot *nimi* ja *valtio_nimi* pitänyt määritellä muuttujiksi. Tämän jälkeen voidaan suorittaa kuvan 14 kysely, jossa ensiksi määritellään *nimi*-muuttujan arvoksi Helsinki. Tämän jälkeen etsitään kaikki kaupungit, jotka sopivat tähän ehtoon. Jos haku on onnistunut ja hakuehdon täyttäviä tietueita on löytynyt, palautetaan löydettyjen kaupunkien tietoalkio *valtio_nimi*.

```
KAUPUNKI.NIMI := 'Helsinki';
$FIND ANY KAUPUNKI USING NIMI;
If DB_STATUS = 0
        then begin
                $GET KAUPUNKI;
                writeln (KAUPUNKI.VALTIO_NIMI)
                end
        else writeln ('no record found');
```

Kuva 14. PASCALilla kirjoitettu kysely, johon on upotettu verkkomallin kyselykieleen liittyvät komennot FIND ja GET. (Mukailtu Elmasrin ja Navathen [1989] esimerkistä sivulta 308.)

3.2. SQL

Relaatioita voidaan käsitellä relaatioalgebran avulla. Relaatioalgebra koostuu joukko-opin operaatioista unioni (yhdiste), leikkaus, erotus ja karteesinen tulo sekä sen omista operaatioista valinta, projektio ja liitokset. Näiden operaatioiden avulla relaatioista voidaan muodostaa uusia relaatioita. Jokaisen relaatioalgebraan pohjautuvan kyselyn tuloksena saadaan relaatio.

SEQUELin kehittäneet Chamberlin ja Boyce [1974] halusivat tarjota kyselykielen, jota voisivat käyttää sekä ammattimaiset ohjelmoijat että henkilöt, joilla ei ole ohjelmointikokemusta. SEQUELin tarkoituksena oli tarjota deklaratiivinen kyselykieli, joka koostuu lohkoista, ei sisällä muuttujia ja on lineaarinen. Deklaratiivisella kyselykielellä tarkoitetaan, että haluttu, kyselyn tuottama tulos kuvaillaan kyselyssä. Vastakohtana tälle on proseduraalinen kyselykieli, jossa kuvaillaan tuloksen tuottava prosessi.

SEQUEL (SQL) -kysely koostuu SELECT-, FROM- ja WHERE-lohkoista, jossa SELECT-osassa kerrotaan mitä haetaan, FROM-osassa mistä haetaan ja WHERE-osassa asetetaan mahdollisia ehtoja FROM-osassa annetuille tiedoille. SQL:ssä tauluja voidaan liittää toisiinsa liitosoperaatioilla. Tavallisessa liitoksessa ((INNER)JOIN) toisiinsa liitetään vain ne rivit, joille löytyy vastinpari toisesta taulusta. Tämän lisäksi voidaan käyttää joko vasenta ulkoliitosta (LEFT (OUTER) JOIN) tai oikeaa ulkoliitosta (RIGHT (OUTER) JOIN). Vasemmassa ulkoliitoksessa liitetään vasemmanpuoleisesta taulusta kaikki rivit ja oikeanpuoleisesta taulusta vain ne rivit, joille löytyy vastinpari vasemmanpuoleisesta taulusta. Tällöin kaikki vasemmanpuoleisen taulun rivit otetaan mukaan

tulostauluun, mutta oikeanpuoleisesta taulusta vain ne rivit, joihin löytyy vastinpari vasemmanpuoleisesta taulusta. Oikea ulkoliitos toimii kuten vasen ulkoliitos, mutta tulostauluun otetaan mukaan kaikki oikeanpuoleisen taulun rivit ja vasemmanpuoleisesta taulusta vain vastinparilliset rivit. Täysi ulkoliitos (FULL (OUTER) JOIN) yhdistää nämä molemmat. Täysi ulkoliitos liittää vastinparilliset rivit toisiinsa, mutta tulostauluun otetaan mukaan myös vastinparittomat rivit. Kuvan 15 SQL-kyselyissä on esitelty niitä ominaisuuksia, joita tutkimuksessa käytetään. Kyselyssä (b) on taulut *valtio* ja *kaupunki* liitetty toisiinsa täydellä ulkoliitoksella ja kyselyssä (c) ne on liitetty toisiinsa vasemmalla ulkoliitoksella.

```
SELECT nimi     SELECT valtio.nimi, kaupunki.nimi     SELECT valtio.nimi, kaupunki.nimi
FROM valtio     FROM valtio FULL JOIN kaupunki ON     FROM valtio LEFT JOIN kaupunki ON
                valtio.nimi = kaupunki.valtio_nimi    valtio.nimi = kaupunki.valtio_nimi

    (a)                         (b)                                 (c)
```

Kuva 15. Esimerkkejä SQL-kyselyistä.

3.3. XPath ja XQuery

XPathin (XML Path Language) [XPath, 1999] avulla voidaan löytää tietoa XML-merkatusta dokumentista. XPath-kysely käsittelee XML-dokumenttia puutietorakenteena ja navigoi XML-dokumentin elementeissä kuten puun solmuissa. Polut elementteihin ja attribuutteihin osoitetaan polkuilmauksella. Polkuilmaus voi olla joko absoluuttinen tai suhteellinen. Absoluuttinen polku alkaa juurisolmusta ja suhteellinen polku alkaa kontekstisolmusta, joka on kyseisellä hetkellä käsiteltävä solmu. Polun sisällä voidaan osoittaa elementin lapsiin (/) tai jälkeläisiin (//). XML-dokumenttia käsitellään solmujen muodostamana puuna. XPath-kysely valitsee kyselyn osoittamat solmut dokumentista. Solmulla voidaan tarkoittaa elementtiä, attribuuttia, tekstiä, nimiavaruutta, kommenttia tai dokumenttisolmua. Tuloksena kyselyyn voidaan saada joukko solmuja, boolen-arvo, numeerinen arvo tai tekstiä. Kysely voidaan esittää joko lyhennetyssä tai lyhentämättömässä muodossa. Lyhentämättömässä muodossa olevat kyselyt käyttävät paikannusaskeleita, jotka on lyhennetyssä muodossa korvattu lyhennysmerkinnöillä. Kyselyssä voidaan myös käyttää monia funktioita. Esimerkkejä funktioista ovat solmun sijaintiin tai arvon vertailuun liittyvät predikaatit sekä erilaiset aggregointifunktiot. [XPath, 1999]

XQuery [XQuery, 2010] on rakennettu käyttämään XPathin ilmaisuja. XQuery ei ole pelkästään kyselykieli, sillä XQuerylla voidaan solmujen valitsemisen ja tiedon muokkaamisen ja aggregoinnin lisäksi lisätä ja poistaa dokumentin sisältöä. XQuery-kyselyissä voidaan käyttää muuttujia sekä proseduraalisia primitiivejä, kuten if-then-else -lausetta ja for-silmukkaa. Kyselyssä voidaan käyttää pelkkää polkuilmausta tai yhdistää polkuilmaus niin sanottuun FLWOR-lausekkeeseen (for, let, where,

order by, return). For-osa käy osan osoittamat solmut yksi kerrallaan läpi ja käsittelee niitä erillään. Let-osa käsittelee osan osoittamat solmut yhtenä kokonaisuutena. Where-osassa voidaan antaa ehtoja for- tai let-osassa osoitettuihin solmuihin. Order by -osassa annetaan mahdollinen järjestysehto. Return-osassa annetaan palautettavat osat ja palautuksen rakenne. Näistä osista kyselyssä on esiinnyttävä ainakin yksi for- tai let-osa ja yksi return-osa, muiden osien ollessa vapaaehtoisia. [XQuery, 2010]

Kuvassa 16 annetaan kolme eri kielellä tehtyä kyselyä, joiden intentio on sama. Esimerkki havainnoi deklaratiivisuuden ja proseduraalisuuden eroja. Esimerkin SQL-kysely koostuu kolmesta lohkosta, joita ovat SELECT, FROM ja WHERE. SELECT-lohko kuvaa palautettavan tiedon (kohdedata), FROM-lohko kuvaa mistä kohdedata haetaan (lähdedata) ja WHERE-lohko kuvaa ehdon lähdedatalle. Esimerkkikysely voitaisiin lukea: palauta otsikko kirjasta, jonka hinta on pienempi kuin 30 yksikköä. Toisin sanoen SQL:ssä kuvataan kyselyn tulos, ei niinkään kyselyn tuloksen tuottavaa prosessia. XPath on polku-orientoitunut ja sarjallinen kyselykieli. XPath-kysely kuvaa polun haluttuun tietoon. Sarjallisuus kuitenkin tarkoittaa, että polun ei tarvitse olla lineaarinen. Esimerkin XPath-kyselyssä valitaan kirja-elementin otsikko-lapsielementti niistä kirja-elementeistä, joilla on ehdon täyttävä hinta-jälkeläiselementti, jolloin kyselyn polku ei ole lineaarinen. Esimerkin XQuery-kysely kuvaa prosessin, kuinka haluttu tieto tuotetaan. Kohdedokumentin *kirja*-elementtejä käydään läpi for-silmukassa yksi kerrallaan ja valitaan *kirja*-elementin *otsikko*-lapsielementit niistä *kirja*-elementeistä, joilla on ehdon täyttävä *hinta*-lapsielementti.

SQL	**XPath**	**XQuery**

`SELECT otsikko`
`FROM kirja`
`WHERE hinta < 30`
	`/kirja[hinta<30]/otsikko`	`for $x in /kirja`
		`where $x/hinta<30`
		`return $x/otsikko`

Kuva 16. Deklaratiivinen, sarjallinen ja proseduraalinen kyselyilmaus.

3.4. XIL

XIL (XML Information Retrieval Language) [Junkkari *et al.,* 2006] on Tampereen yliopistossa kehitetty XML-kyselykieli. XIL perustuu SEQUEL-kyselykieleen ja on suunnattu erityisesti dokumenttisuuntautuneeseen XML-tiedonhakuun, mutta omaa myös piirteitä tietosuuntautuneista kyselykielistä. XIL on suunniteltu vastaamaan samoihin tavoitteisiin kuin SEQUEL aikanaan [Chamberlin and Boyce, 1974]. Näitä tavoitteita ovat kyselyn jakaminen lohkoihin, muuttujattomat kyselyt, lineaarinen kyselyn muotoilu ja proseduraalisten primitiivien jättäminen pois kyselykielestä. XIL koostuu SELECT-, FROM- ja WHERE-lohkoista, joista SELECT-osassa annetaan haettavat elementit, FROM-osassa elementit, joiden sisältä SELECT-osan elementtejä haetaan ja WHERE-osassa ehtoja FROM-osan elementeille. XIL-kyselyissä ei käytetä proseduraalisia primitiivejä kuten if-then-

else -haarat, for-silmukat tai funktiot. Muita XIL:n päätavoitteita ovat tuki relevanssilajitellulle elementtien ryhmittelylle. Elementtien ryhmittelyssä tuloksena saadut elementit ryhmitellään dokumentin järjestyksen / hierarkian mukaan ja jokaisen ryhmän sisältämien elementtien pisteet vaikuttavat ryhmän sijoittumiseen.

Ollakseen lineaarinen kyselykieli, XIL ei sisällä muuttujia, iteraatioita tai sarjallistettuja tietorakenteita. Koska XIL on kuitenkin XML-kyselykieli, ja XML-tietomalli on hierarkkinen (puu), on tämä hierarkia pystyttävä ilmaisemaan kyselyssä jollakin muulla tavalla. Näitä tapoja ovat useat FROM-WHERE -lohkot sekä lineaariset polut. Lineaarinen polku tarkoittaa, että sarjallistaminen tai ehdot polun sisällä eivät ole sallittuja. Vastakohtana tälle ovat esimerkiksi XPath-kyselyssä käytetyt solmun sijaintiin tai arvon vertailuun liittyvät funktiot. Lineaarisessa polussa voidaan kuitenkin XPathin tapaan viitata elementin lapsiin (/) tai jälkeläisiin (//). Lisäksi lineaarisen polun alkaessa kauttaviivalla (/) tulkitaan polun ensimmäisen solmun viittaavan juurisolmuun ja lineaarisen polun päättyessä kenoviivaan (\) tulkitaan polun viimeisen solmun viittaavan lehtisolmuun. XIL mahdollistaa myös tavanomaiset vertailuoperaattorit vertailtaessa elementtien tai attribuuttien ominaisuuksia.

XIL-kyselyt voidaan kohdistaa yhteen tai useampaan XML-dokumenttiin. Kyselyn tulos ryhmitellään oletuksena dokumenteittain, mutta ryhmittelyä voi muuttaa group by -operaatiolla. Kyselyn tuloksen ryhmittely voidaan myös poistaa käyttämällä unique-liitettä. Kuten SEQUEL (SQL) -kyselyssä, XIL-kyselyn SELECT-osassa on mahdollista luetella useita elementtejä pilkulla erotettuna. Oletusryhmittelyssä nämä elementit kootaan tuloksessa yhteen dokumenteittain. Lisäksi missä tahansa kyselyn osassa voidaan käyttää putki-merkkiä (|) ilmaisemaan vaihtoehtoisia elementin nimiä. Esimerkiksi XIL-kysely `SELECT otsikko|otsake` ilmaisee, että *otsikko* ja *otsake* ovat synonyymejä keskenään ja kysely voi palauttaa kummatkin tai toisen näistä elementeistä.

```
SELECT nimi          SELECT /valtio/nimi    SELECT valtio//nimi     SELECT nimi
                                                                    FROM valtio

    (a)                   (b)                    (c)                    (d)

SELECT nimi          SELECT nimi            SELECT järvi/nimi       SELECT UNIQUE nimi
FROM valtio          FROM valtio/joki       FROM valtio             FROM järvi
WHERE valtiomuoto    WHERE pituus           WHERE väkiluku          FROM valtio
      = tasavalta         != 510            < 6 000 000             WHERE väkiluku
                     GROUP BY joki                                  < 6 000 000

    (e)                   (f)                    (g)                    (h)
```

Kuva 17. Kyselykielen ominaisuuksia esittävät XIL-kyselyt.

Kuvassa 17 on kahdeksan erilaista XIL-kyselyä, jotka kuvaavat edellä mainittuja ominaisuuksia. Vaikka XIL-kyselyn syntaksi joissakin kyselyissä vastaa SQL:n syntaksia, on XIL-kyselyn semantiikka kuitenkin hierarkkinen. Hierarkkinen semantiikka tarkoittaa muun muassa, että SELECT-osan kohteet kohdistuvat kaikkiin FROM-osassa annetun elementin jälkeläisiin. Esimerkiksi kyselyt (d) ja (e) ovat sellaisenaan SQL:n syntaksin mukaisia kyselyjä. Näiden kyselyjen semantiikka SQL-kyselynä on kuitenkin erilainen kuin XIL-kyselynä. Kysely (a) havainnollistaa, että kyselyssä riittää pelkkä SELECT-osa. Tämän kyselyn semantiikka tarkoittaa, että palautetaan kaikki *nimi*-elementit dokumenteista, olivatpa ne missä vain dokumentissa. Esimerkkitietokannastamme palautettaisiin valtioittain ryhmiteltynä valtion nimi, kaupunkien nimet, jokien nimet ja järvien nimet. Kyselyssä (b) viitataan *valtio*-juurielementin *nimi*-lapsielementtiin. Tällöin kyselyn semantiikka tarkoittaa, että palautetaan vain *valtio*-nimisten juurielementtien *nimi*-elementit valtioittain ryhmiteltynä. Koska esimerkkidokumentissa *valtio*-elementtejä on vain juurielementteinä, ensimmäinen kauttaviiva voidaan jättää pois. Kysely (c) viittaa *valtio*-elementin *nimi*-jälkeläiselementteihin, jolloin kyselyn semantiikka tulkitaan niin, että palautetaan kaikki *nimi*-elementit dokumentista valtioittain ryhmiteltynä, kuten kyselyssä (a). Kysely (d) vastaa syntaksiltaan kyselyä (c).

Kyselyssä (e) annetaan *valtio*-elementille ehto WHERE-osassa. Kysely tarkoittaa, että valitaan ne *valtio*-elementit, joilla on jälkeläisenään *valtiomuoto*-elementti arvoltaan tasavalta. Kyselyn tuloksena palautetaan näiden valittujen *valtio*-elementtien *nimi*-jälkeläiselementit valtioittain ryhmiteltynä. Kysely (f) havainnollistaa, että lineaarista polkua voidaan käyttää myös FROM-osassa ja tämän lisäksi sitä voidaan käyttää WHERE-osassa. Kyselyssä valitaan ensiksi *valtio*-elementin *joki*-nimiset lapsielementit, joiden pituus on erisuuri kuin 510 ja palautetaan näiden valittujen *joki*-elementtien *nimi*-jälkeläiselementit. Esimerkkitietokannassa tämä tarkoittaa vain *joen nimi*-elementtejä. Nyt pitää kuitenkin huomata, että palautettavia *nimi*-elementtejä ei ryhmitellä valtioittain vaan jokien mukaan, koska se on kyselyssä osoitettu group by -operaatiolla. Kysely (g) valitsee ensin ne *valtio*-elementit, joilla on *väkiluku*-jälkeläiselementti, jonka arvo on pienempi kuin 6 000 000. Seuraavaksi valitaan valittujen *valtio*-elementtien *järvi*-jälkeläiselementit ja lopuksi palautetaan näiden *järvi*-elementtien *nimi*-lapsielementit valtioittain ryhmiteltyinä. Kysely (h) havainnollistaa sekä unique-määreen käyttöä että useampia FROM-WHERE -osia. Kyselystä selviää, että FROM voi esiintyä yksistään ilman WHERE-osaa. Kysely vastaa semantiikaltaan kyselyä (g), sillä erotuksella, että kyselystä on poistettu ryhmittely unique-määreen avulla.

4. Relaatioiden hierarkkiset näkymät

XML-tietomallissa hierarkia on eksplisiittisesti mukana. Eksplisiittinen hierarkia on systemaattisesti sääntöpohjaisesti muodostettu hierarkkinen kuvaus tiedosta, joka voitaisiin yleensä muodostaa myös toisella tavalla. Tiedosta, jota ei alun perin ole tallennettu hierarkkisessa muodossa, voidaan muodostaa implisiittinen hierarkia. Implisiittisellä hierarkialla tarkoitetaan tiedosta (ad hoc) muodostettua hierarkkista näkymää, joka voidaan muodostaa tiedon osista ja niiden välisistä suhteista. Hierarkkisella näkymällä tarkoitetaan syklitöntä suunnattua graafia, jolla on yksikäsitteinen lähtösolmu. XML-termistöä noudattaen esimerkiksi solmun välitöntä seuraajaa kutsutaan lapseksi ja välitöntä edeltäjää vanhemmaksi. David [2003] on tutkinut, kuinka relationaalisesta tiedosta voidaan muodostaa hierarkkisia näkymiä SQL:n vasemman ulkoliitoksen avulla. Tässä luvussa esitellään hierarkian liittäminen relaatiomalliin Davidin [2003] tutkimukseen pohjautuen. Lisäksi määritellään käsitteet relaatiomallin hierarkkiselle tulkinnalle. Nämä käsitteet mahdollistavat automaattisen hierarkian muodostuksen relaatiotietokannasta.

4.1. XML-tietomalli vs. relaatiomalli

XML-tietomallin ja relaatiomallin erojen ja yhtäläisyyksien ymmärtäminen on tutkimuksen keskeinen osa. XML-tietomalli pohjautuu hierarkiaan ja tähän tietomalliin pohjautuvat kyselykielet mahdollistavat hierarkian käytön kyselyssä. XML-tietomallissa esivanhempielementit (ancestors) sisältävät jälkeläiselementtinsä (descendants) eli hierarkian samassa haarassa korkeammalla olevat elementit sisältävät alempana olevat elementit. Esimerkiksi aiemmin esitetyssä kuvan 12 *valtio*-dokumentissa *valtio*-elementti sisältää ominaisuuksiensa lisäksi *kaupunki*-, *joki*- ja *järvi*-elementit sekä näiden elementtien lapsielementit. Kyselyn `SELECT nimi FROM valtio` (etsi nimi kohteesta valtio) voidaan intuitiivisesti, mihinkään kyselykieleen tai tietomalliin liittämättä, ajatella tarkoittavan valtioon sisältyviä nimiä. Toisin sanoen kysely tulkitaan hierarkkista semantiikkaa noudattaen. Nyt XML-tietomallissa kysely `SELECT nimi FROM valtio` tarkoittaa *valtio*-elementin sisällä olevia *nimi*-elementtejä, joita ovat valtion nimen lisäksi kaupungin, joen ja järven nimet. Vastaavasti relaatiomalli pohjautuu relaatioihin ja tähän tietomalliin pohjautuvat kyselykielet mahdollistavat relaatioiden kanssa operoimisen. Relaatiomallissa monikoiden voidaan katsoa sisältävän ominaisuutensa eli attribuuttiensa arvot, mutta ei muita monikoita. Sen sijaan monikot voivat liittyä muihin monikoihin (ensimmäinen normaalimuoto). Esimerkiksi *valtio*-monikon ⟨*Suomi, tasavalta, 5391699*⟩ tai *kaupunki*-monikon ⟨Tampere, 591892, Suomi⟩ voidaan katsoa sisältävän attribuuttiensa arvot. Nämä kaksi monikkoa eivät eksplisiittisesti sisällä toisiaan, vaikkakin selvästi liittyvät toisiinsa. Tarkemmin sanottuna *kaupunki*-monikko sisältää *valtio*-monikkoon viittaavan vierasavaimen, jolloin nämä kaksi monikkoa liittyvät toisiinsa vierasavaimen kautta. Tällöin relaatiomalliin pohjautuvana kyselynä `SELECT nimi FROM`

`valtio` tarkoittaa *valtio*-monikon *nimi*-attribuuttia, koska *valtio*-monikko sisältää attribuutin nimeltään *nimi*. Vaikka *valtio*-monikkoon *liittyy* muiden monikkojen kautta useita *nimi*-attribuutteja, ne eivät kuitenkaan sisälly tähän monikkoon. Tämän eron ymmärtäminen auttaa kääntämään XIL-kyselyt SQL-kyselyiksi, siten että alkuperäisen kyselyn semantiikka säilyy.

Kyselyn `SELECT kaupunki FROM valtio` (etsi kaupunki kohteesta valtio) voidaan intuitiivisesti, mihinkään kyselykieleen tai tietomalliin liittämättä, ajatella tarkoittavan valtioon sisältyviä kaupunkeja. Ylempänä esitetyssä kuvan 12 *valtio*-dokumentissa tämä tarkoittaa *valtio*-elementin *kaupunki*-lapsielementtiä. Kuvan 4 relaatiotietokannassa kyselyn merkitystä on syytä tarkastella lähemmin. Kuvan 4 relaatiotietokantakaaviosta näemme, että valtio ei sisällä *kaupunki*-attribuuttia, mutta *valtio*-relaatio liittyy *kaupunki*-relaatioon. XML-tietomallissa kysely `SELECT kaupunki FROM valtio` on mahdollinen eksplisiittisen hierarkian ansiosta. Kuitenkin tarkasteltaessa esimerkiksi kuvan 4 relaatioita ja niiden liittymistä toisiinsa nähdään, että hierarkia on selvitettävissä myös relaatiomallista. Relaatiomallista voidaankin muodostaa implisiittinen hierarkia. Muodostunut hierarkia riippuu siitä, mistä relaatiosta on lähdetty liikkeelle, eli mitä relaatiota on käytetty hierarkian juurena. Tämän implisiittisen hierarkian ansiosta relaatiotietokantaan on mahdollistaa kohdistaa kysely, jolla on sama merkitys kuin alkuperäisellä XIL-kyselyllä. Merkityksellä tarkoitetaan sitä, että esimerkiksi XIL-kyselyn viitatessa lapsielementtien lisäksi jälkeläiselementteihin, myös SQL-kyselyssä viitataan jälkeläissolmuihin. Seuraavaksi tarkastellaan implisiittisen hierarkian tulkintaa tarkemmin.

4.2. Relaatiomallin hierarkkinen tulkinta

Relaatiomallissa implisiittisesti esiintyvä hierarkia voidaan tulkita tiukasti tai väljemmin. Tiukassa tulkinnassa relaation x lapsia ovat ne relaatiot y, jotka sisältävät vierasavaimen relaatioon x. Näin ollen relaation y sisältäessä vierasavaimen relaatioon x ja relaation z sisältäessä vierasavaimen relaatioon y, tulkitaan relaatio z relaation x jälkeläiseksi. Tällöin relaatiomallista voidaan määrittää myös juuri- ja lehtisolmut (vrt. elementit). Juurisolmun tunnistaa siitä, että sillä ei ole vanhempisolmua. Lehtisolmun taas tunnistaa siitä, että sillä ei ole lapsisolmuja. Edellä esitettiin hierarkian käsitteen liittäminen relaatiotietokantoihin. Tästä voimme johtaa myös juuri- ja lehtisolmujen käsitteet relaatiotietokannoissa. Tämä käsittely jaetaan relaatioihin ja attribuutteihin. Attribuutteihin liittyy aina vanhempisolmu, nimittäin relaatio, johon attribuutti sisältyy. Näin ollen attribuutti ei voi koskaan olla juurisolmu. Toisaalta attribuutti ei sisällä muita solmuja, joten se on aina lehtisolmu. Seuraavaksi käsitellään pelkkiä relaatioita. Relaatio x, joka ei sisällä mihinkään muuhun relaatioon y viittaavaa vierasavain-attribuuttia, on juurisolmu. Relaatio x, johon minkään muun relaation y vierasavain ei osoita on lehtisolmu. Näiden käsitteiden avulla saadaan myös ne relaatiot, jotka esiintyvät erillään muista relaatioista. Jos relaatio x on sekä juuri- että lehtisolmu, ei relaatio x liity mihinkään muuhun relaatioon y. Hierarkian tiukassa tulkinnassa esiintyy kuitenkin ongelmia. Esimerkiksi muodostettaessa

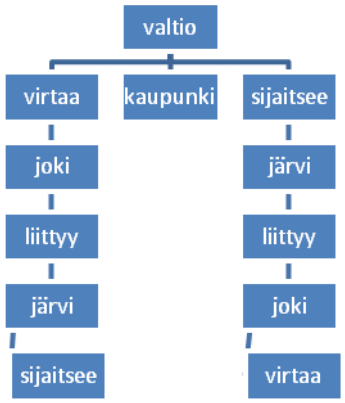
hierarkia kuvassa 4 esiintyvien *valtio*- ja *järvi*-relaatioiden välille, tulee *valtio*-relaation esiintyä juurena, jonka lapsena on *sijaitsee*-relaatio ja jälkeläisenä *järvi*-relaatio. Tämä ei tiukan tulkinnan avulla onnistu, koska *järvi*-relaatio ei sisällä vierasavainta *sijaitsee*-relaatioon, vaan *sijaitsee*-relaatio sisältää vierasavaimen *järvi*-relaatioon.

Relaatiotietokannassa implisiittisesti esiintyvä hierarkia voidaan tulkita myös väljemmin. Jos relaatio y sisältää vierasavaimen relaatioon x, voidaan relaatio y tulkita relaation x lapseksi tai relaatio x relaation y lapseksi. Lisäksi jos relaatio y sisältää vierasavaimen relaatioon x ja relaatioon z, voidaan relaatio z tulkita relaation x jälkeläiseksi tai relaatio x relaation z jälkeläiseksi. Kuitenkin jos relaatio r ei sisällä vierasavainta relaatioon x ja relaatio x ei sisällä vierasavainta relaatioon r, ei relaatiota r tulkita relaation x lapseksi eikä relaatiota x relaation r lapseksi. Väljemmässä tulkinnassa relaatioiden väliset suhteet tulkitaan siis symmetrisinä (xRy → yRx). Tästä eteenpäin tätä väljempää tulkintaa kutsutaan symmetriseksi tulkinnaksi. Symmetrisen tulkinnan avulla on mahdollista muodostaa hierarkia kuvassa 4 esiintyvien *valtio*- ja *järvi*-relaatioiden välille. Hierarkia voidaan aloittaa joko *valtio*- tai *järvi*-relaatiosta. Seuraavaksi tarkastellaan hierarkiaa, joka on aloitettu *valtio*-relaatiosta. Ensinnä *valtio*-relaatioon liitetään *sijaitsee*-relaatio. Tämä onnistuu, koska *sijaitsee*-relaatio sisältää *valtio*-relaatioon viittaavan *valtio_nimi*-attribuutin. Lopuksi liitetään *sijaitsee*-relaatioon *järvi*-relaatio. Tämä onnistuu, koska *sijaitsee*-relaatio sisältää *järvi*-relaatioon viittaavan *järvi_nimi*-attribuutin. Nyt symmetrisen tulkinnan seurauksena juuri- ja lehtisolmujen määritteleminen relaatiomallissa ei ole enää niin selvää. Tiukan tulkinnan mukaan relaatio x on juuri, jos se ei sisällä vierasavaimia mihinkään toiseen relaatioon y. Symmetrisessä tulkinnassa voidaan kuitenkin tulkita relaation x olevan relaation y lapsi, vaikka relaatio x ei eksplisiittisesti vierasavainta sisälläkään. Riittää, että relaatio y sisältää vierasavaimen relaatioon x. Toiseksi tiukan tulkinnan mukaan relaatio x on lehti, jos mikään relaatio y ei sisällä relaatioon x viittaavaa vierasavainta. Kuitenkin relaation x sisältäessä relaatioon y viittaavan vierasavaimen, voidaan symmetrisessä tulkinnassa relaatio y tulkita relaation x lapseksi ja näin ollen relaatio x ei ole enää lehtisolmu.

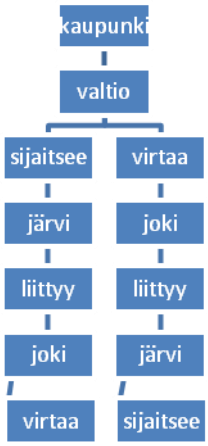
Näin ollen tiukka tulkinta rajoittaa hierarkian muodostusta, mutta poistaa hierarkian muodostuksessa esiintyvää monitulkintaisuutta. Symmetrinen tulkinta ei rajoita hierarkian muodostusta, mutta lisää hierarkian muodostuksessa esiintyvää monitulkintaisuutta. Symmetrisen tulkinnan etuna on myös se, että se vähentää tarvetta tietolähteen rakenteen tuntemiselle. Kyselyn tekijän ei tarvitse tietää, kuinka kaksi tietokannan relaatiota liittyy toisiinsa, vaan se selvitetään automaattisesti. Symmetrisessä tulkinnassa, esimerkiksi relaatioiden x ja y liittyessä toisiinsa, ei tarvitse tietää, onko vierasavain sijoitettu relaatioon x vai relaatioon y tai liittyvätkö relaatiot toisiinsa transitiivisesti muiden relaatioiden välityksellä. Myös nämä asiat on tärkeää ottaa huomioon käännettäessä XIL-kyselyjä SQL-kyselyiksi.

Kuvassa 18 on kuvattu millainen symmetrisen tulkinnan mukainen hierarkia kuvan 4 relaatiotietokannasta muodostuu valittaessa *valtio*-relaatio hierarkian juureksi. Kuten kuvasta nähdään,

hierarkia aloitetaan valitusta juurirelaatiosta ja kootaan käyttäen relaatioiden välisiä suhteita. Hierarkia muodostetaan haaroittain ja hierarkian muodostus haarassa lopetetaan, kun törmätään haarassa jo esiintyvään relaatioon. Toisin sanoen kuvan 18 lehtisolmuissa *sijaitsee*, *kaupunki* ja *virtaa* ei ole enää jatkettu *valtio*-relaatioon, koska se esiintyy jo näissä haaroissa. Tämän relaatiotietokannasta muodostetun hierarkian avulla onkin jo selvää, miten kysely `SELECT kaupunki FROM valtio` tulkitaan relaatiokannassa. Vastaavasti kuvassa 19 on muodostettu symmetrisen tulkinnan mukainen hierarkia käyttäen *kaupunki*-relaatiota juurisolmuna.



Kuva 18. Symmetrisen tulkinnan mukainen hierarkia valtiotietokannasta valittaessa valtio-relaatio juureksi (attribuutit on jätetty pois hierarkian yksinkertaistamiseksi).



Kuva 19. Symmetrisen tulkinnan mukainen hierarkia valtiotietokannasta valittaessa kaupunki-relaatio juureksi (attribuutit on jätetty pois hierarkian yksinkertaistamiseksi).

5. XML-to-SQL taustaa

XML:n ja relationaalisen tiedon yhdistämistä on tutkittu erilaisista näkökulmista. Artikkelissa *XML-to-SQL Query Translation Literature* [Krishnamurthy *et al.*, 2004] nämä tutkimukset on jaettu kahteen osaan: niihin, joissa olemassa oleva relationaalinen tieto tai osa siitä julkaistaan XML-muodossa [Benedikt *et al.*, 2002; Bohannon *et al.*, 2002b; Deutsch and Tannen, 2003; Eisenberg and Melton, 2002; Fernández *et al.*, 2002; Fernández *et al.*, 2000; Jain *et al.*, 2002; Li *et al.*, 2003; Manolescu *et al.*, 2001; Shanmugasundaram *et al.*, 2001a; Shanmugasundaram *et al.*, 2000] ja niihin, joissa olemassa oleva XML-muotoinen tieto tallennetaan relaatiotietokantaan XML-kaavioon (XML-skeema tai DTD) pohjautuen [Bohannon *et al.*, 2002a; Chen *et al.*, 2002; Hongwei *et al.*, 2002; Klettke and Meyer, 2000; Krishnamurthy *et al*., 2003; Lee and Chu, 2000; Mani and Lee, 2002; Runapongsa and Patel, 2002; Shanmugasundaram *et al.*, 2001b; Shanmugasundaram *et al.*, 1999; Tatarinov *et al.*, 2002] tai XML-kaaviosta riippumattomasti [Dehaan *et al.*, 2003; Deutsch *et al.*, 1999; Florescu and Kossmann, 1999; Grust, 2002; Li and Moon, 2001; Schmidt *et al.*, 2000; Teubner *et al.*, 2003; Tatarinov *et al.*, 2002; Yoshikawa *et al.*, 2001; Zhang *et al.*, 2001]. Kumpaankin lähestymistapaan liittyy XML-kyselykielellä olevan kyselyn kääntäminen SQL:ksi. Ensiksi mainitussa relationaalisesta tiedosta muodostetaan XML-näkymä ja tätä näkymää vastaan muodostetaan XML-kysely. Jälkimmäisessä tapauksessa jo olemassa olevaa XML-näkymää vastaan muodostetaan XML-kysely. XML-kysely käännetään SQL:ksi käyttäen XML-näkymän ja relaatiokaavion välistä kuvausta. Lisäksi voidaan erottaa vielä tutkimukset, joissa XML-muotoisen tiedon muuttamiseen relaatiomuotoon ei liity kyselykielen kääntämistä [Niemi *et al.* 2009]. Seuraavaksi näitä lähestymistapoja verrataan XIL-kyselyt SQL-kyselyiksi kääntävään XILtoSQL-ohjelmaan.

5.1. XML relaatioiksi

XML-muodossa oleva tieto voidaan tallentaa XML-kaavioon pohjautuen tai ilman XML-kaaviota. Myös asiaa käsittelevät tutkimukset voidaan jakaa tällä tavalla. XML-kaavioon perustuvien tutkimusten päätehtäviä on annettujen XML-kaavioiden avulla valita sopiva relationaalinen kaavio ja näiden kaavioiden avulla muuttaa XML-muodossa oleva tieto relaatiomuotoon sekä kääntää XML-kysely relaatiotietokantaan sopivaksi kyselyksi käyttäen apuna kuvausta, jolla XML-muotoinen tieto on muutettu relaatiomuotoon. XML-kaavioon perustumattomien tutkimukset keskittyvät pääosin siihen mitä, geneeristä relationaalista kaaviota tulisi käyttää muutettaessa tietoa XML:stä relaatioiksi sekä kuinka XML-kyselyt muutetaan relaatiotietokannan kyselyiksi käyttäen apuna valittua relationaalista kaaviota. Relationaalisen kaavion valintaan ei vaikuta mahdollisesti olemassa oleva XML-kaavio. Työn kannalta on mielekästä tarkastella eroja tämän tavan ja XILtoSQL-ohjelman välillä vain kyselyn

muotoilun kannalta. XILtoSQL-ohjelma eroaa kyselyn muotoilussa siinä, että kyselyn ei muotoiluvaiheessa tarvitse sopia ennalta määrättyyn XML-näkymään.

Niemi *et al.* [2009] muuttavat heterogeenistä XML-tietoa relaatiomuotoon. Muutettaessa tieto relaatiomuotoon se harmonisoidaan poistamalla mahdolliset datakonfliktit. Niemen *et al*. [2009] mukaan tämä tapa sopii hyvin esimerkiksi liiketoimintatiedon hallintaan. Kun tieto on muutettu ja harmonisoitu yhtenäiseen muotoon, tähän tietoon on helpompi kohdistaa kyselyitä. Tämä tapa eroaa XILtoSQL-ohjelmasta siinä, että tieto muutetaan heterogeenisestä homogeeniseksi, kun XILtoSQL-ohjelmassa tietolähteiden muotoa ei muuteta, mutta kysely muutetaan vastaamaan tietolähdettä.

5.2. Relaatiot XML:ksi

Krishnamurthy *et al.* [2004] erittelevät relationaalista tietoa XML:ksi julkaistaessa päätehtäviksi XML-näkymän muodostamisen relationaalisesta tiedosta, XML-näkymän materialisoinnin ja XML-kyselyn evaluoinnin koostamalla se näkymää käyttäen. Pitää kuitenkin huomata, että XML-näkymien materialisointi kokonaisuudessaan on vain erikoistapaus. XML-näkymiä ja niiden osien materialisointia on tavallisimmin käytetty kyselyn evaluoinnissa. Tutkimukset eroavat muun muassa siinä, sallitaanko niissä rekursiivinen (elementti voi sisältää itsensä) vai ei-rekursiivinen XML-kaavio. Yhteinen piirre lähes kaikissa artikkelissa esitetyissä tutkimuksissa on XML-kyselykielellä annetun kyselyn kääntäminen relaatiotietokantaan sopivan kyselykielen kyselyksi. Usein relaatiotietokantakyselykielenä on käytetty SQL-kyselykieltä. Yhtenä avoimena ongelmana Krishnamurthy *et al.* [2004] nostavat XML-kyselyt, jotka sisältävät polkuilmauksen, kuten viittauksen jälkeläiseen (//). Artikkelin kuvaamissa tutkimuksissa ei ole kyetty ratkaisemaan tällaisia tilanteita XML-kaavion ollessa rekursiivinen.

Ero näiden tutkimusten ja XILtoSQL-ohjelman välillä on se, että kyselyn ei tarvitse muotoiluvaiheessa sopia XML-näkymään, vaan annettu kysely pyritään sovittamaan käytössä olevien tietolähteiden kaavioon. Tällöin päätehtäviksi muodostuvat vain XML-kyselyn kääntäminen SQL:ksi käyttäen kaaviota relationaalisesta tiedosta ja relaatioina saadun tuloksen kääntäminen XML:ksi esimerkiksi sovittamalla vastaus alkuperäisen XML-kyselyn tarjoamaan kuvaukseen.

5.3. LINQ

LINQ [LINQ, 2010] on työkalu, jonka sisältämällä hakutekniikalla voidaan hakea tietoa muun muassa relaatiotietokannasta, XML-tiedostoista tai oliotietokannasta. LINQ on työn kannalta mielenkiintoinen, koska yhden hakutekniikan avulla voidaan hakea tietoa useassa muodossa olevasta tietolähteestä. LINQ eroaa XILtoSQL-ohjelmasta muun muassa siinä, että haun yhteydessä on eksplisiittisesti kerrottava mihin tietolähteeseen kysely kohdistuu ja hakua muotoiltaessa on tiedettävä tietolähteen muoto. LINQ-haut kohdistetaan aina oliokokoelmaa vastaan, joten käytetty tietolähde on muutettava oliomuotoon. Vaikka tietolähde muutetaan oliomuotoon, ei erimuotoisiin tietolähteisiin voi kohdistaa samaa LINQ-

hakua, koska haun muotoilu riippuu käytetyn tietolähteen muodosta. XILtoSQL-ohjelma taas on suunniteltu käymään läpi saatavilla olevat tietolähteet ilman, että kyselyn tekijän tarvitsee tätä kertoa. Tällöin myöskään kyselyä muotoiltaessa ei tietolähteen muotoon tarvitse ottaa kantaa.

5.4. SQL:n hierarkkinen prosessointi

SQL-kyselykieli sisältää operaattoreita, joiden prosessointi voidaan tulkita hierarkkisesti. David [2003] kuvaa tätä artikkelissa *ANSI SQL hierarchical processing can fully integrate native XML*. David esittää SQL:n hierarkkisen prosessoinnin koostuvan kolmesta osasta: hierarkkinen tiedon mallintaminen, hierarkkisen joukon muodostaminen ja hierarkkinen karteesisen tulon prosessointi. Kun hierarkkinen tiedon mallintaminen on tehty, kaksi muuta osaa SQL suorittaa automaattisesti. Hierarkkinen tiedon mallinnus toteutetaan SQL:ssä vasemmalla ulkoliitoksella. Tällöin kaikki vasemmanpuoleisen taulun rivit otetaan mukaan tulostauluun, mutta oikeanpuoleisesta taulusta vain ne rivit, joihin löytyy vastinpari vasemmanpuoleisesta taulusta. David [2003] toteaa, että tämä käyttäytyminen viittaa vasemmanpuoleisen komponentin olevan hierarkiassa ylempänä kuin oikeanpuoleisen. SQL:n avulla voidaan mallintaa monitasoisia ja monihaaraisia hierarkioita. Vasen ulkoliitos muodostaa taulujen hierarkian ja ON-ehto määrittää, miten nämä taulut liitetään toisiinsa [David, 1992]. Monihaarainen hierarkia muodostuu, kun kuvatussa hierarkiassa sama ylemmän tason taulu liitetään useita kertoja toiseen tauluun. XILtoSQL-ohjelman hierarkian muodostus pohjautuu Davidin [1992; 2003] tutkimukselle SQL:n hierarkkisesta prosessoinnista ja tavasta tulkita taulut ja niiden yhteydet hierarkkisesti. XILtoSQL käyttää muun muassa vasenta ulkoliitosta ja ON-ehtoa taulujen välisen hierarkian muodostukseen. XIL-kyselyjen kääntäminen SQL-kyselyiksi ei ole kuitenkaan mahdollista pelkästään SQL:n operaatioiden avulla, vaan käännöksen perusta on relaatiomallin hierarkkisen tulkinnan automatisoinnilla. Tällöin kahden taulun välinen suhde voidaan määritellä käännöksen aikana, vaikka sitä ei alkuperäisessä kyselyssä (XIL) ole välttämättä määritelty.

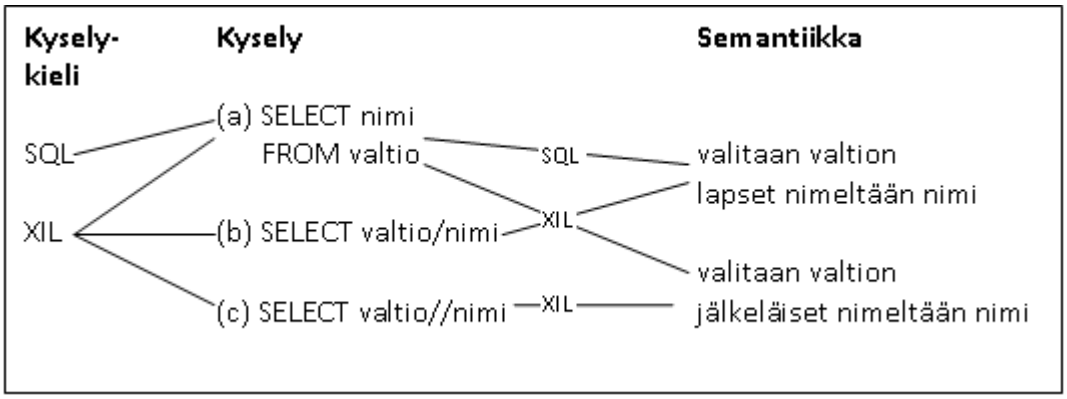
6. XILtoSQL – informaali kuvaus

Luvussa 5 annetun jaottelun suhteen XILtoSQL-ohjelma liittyy lähiten tutkimusalueeseen, jossa relationaalinen tieto julkaistaan XML-muodossa. Tämän tyyppisessä lähestymistavassa on ollut keskeistä muodostaa XML-kaavio relaatiotietokannasta ja materialisoida tämä kaavio joko kokonaan tai osittain. Materialisointi tapahtuu usein muodostamalla kysely relaatiotietokantaan ja sovittamalla tulos XML-kaavioon. Kuten luvusta 5 käy ilmi, tyypillisesti muodostetaan ensin kysely käyttäen XML-kyselykieltä (huomaa, että kyselyn tulee sopia relaatiotietokannasta muodostettuun XML-kaavioon). Tämä kysely käännetään automaattisesti SQL:ksi käyttäen relaatiotietokannan käännöstä XML-kaavioksi. Lopuksi relaatiotietokannasta saatu relaatiomuodossa oleva vastaus sovitetaan XML-kaavioon. Näin koko XML-kaaviota ei välttämättä materialisoida, vaan vain kyselyn vaatima osa siitä.

Toisin kuin edellä kuvatussa lähestymistavassa, tässä tutkimuksessa lähdetään siitä, että kyselyn tekijä ei välttämättä tunne käytetyn tietolähteen rakennetta. Kyselyn tekijä saattaa tietää tietolähteessä esiintyvien olioiden ja olioiden ominaisuuksien nimet, tai osan niistä, mutta tieto olioiden keskinäisestä järjestyksestä tai siitä, kuinka oliot liittyvät toisiinsa ei ole välttämätön. Kyselyn tekijä esimerkiksi tietää hakevansa tietoa lähteistä, joissa esiintyy tietoja valtioista, kaupungeista, järvistä ja joista. Hän ei kuitenkaan tiedä, missä muodossa tämä tieto tietolähteessä esiintyy eikä tiedon rakennetta tai hierarkiaa. Tällöin tieto niistä järvistä, jotka sijaitsevat valtiossa nimeltä Ruotsi, saadaan XIL-kyselyllä `SELECT järvi FROM valtio WHERE nimi = Ruotsi`. Kuvan 4 relaatiotietokannassa relaatiot *järvi* ja *valtio* eivät liity suoraan toisiinsa, vaan ne liittyvät toisiinsa *sijaitsee*-relaation kautta. Kyselyn tekijän ei kuitenkaan tarvitse tietää tätä, vaan tämä päätellään automaattisesti tietokannan suhteista. XILtoSQL-ohjelman suorittama kyselyn käännös eroaakin muista tiedossa olevista tutkimuksista siinä, että XIL-kyselykielellä annetun kyselyn ei oleteta vastaavan mitään XML-kaaviota tai -rakennetta, vaan kysely pyritään sovittamaan tietolähteen tietokantakaavioihin. Tästä syystä XIL-kysely käännetään ensiksi yleistetyksi kyselyn monikkoesitykseksi. Tämä ensimmäinen käännös muodostaa XIL-kyselystä abstraktimman välimuodon, jolloin kysely ei ole enää tiukasti sidottu alkuperäisen kyselykielen syntaksiin. Käännettäessä XIL-kyselyitä SQL:ksi on olemassa sekä *staattisia* että *dynaamisia* sääntöjä. Staattiset säännöt ovat käännöksen ydin, jota ei voida muuttaa. Dynaamiset säännöt taas ovat muutettavissa, ja niiden arvoja voidaan muuttaa joko tilanteen tai kyselyn tekijän johdosta. Näitä sääntöjä tarkastellaan luvussa 6.1. ja 6.2.

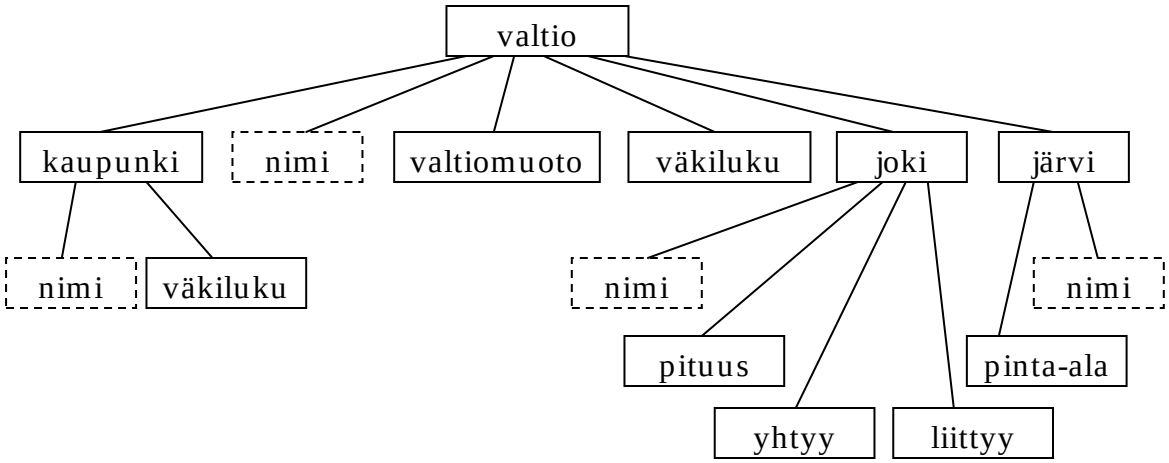
Kuten jo aiemmin on todettu, XIL- ja SQL-kyselyt saattavat vastata syntaksiltaan toisiaan, mutta niiden semanttinen tulkinta on erilainen. Esimerkiksi kuvan 20 kysely (a) `SELECT nimi FROM valtio`, voi syntaksinsa puolesta olla joko XIL- tai SQL-kysely. XIL-kyselynä tämä tarkoittaa, että valitaan kaikki *nimi*-elementit *valtio*-elementin alta. Sama semantiikka on myös kyselyllä `SELECT valtio//nimi`. Tämä ei vastaa enää syntaksiltaan SQL-kyselyä, koska kyselyssä käytetään

polkuilmausta ja toisaalta kyselyssä ei esiinny FROM-osaa. SQL-kyselynä `SELECT nimi FROM valtio` tarkoittaa, että valitaan *valtio*-taulussa oleva *nimi*-sarake. Tätä semantiikkaa vastaa XIL-kysely `SELECT valtio/nimi`.



Kuva 20. Kyselykielen syntaksin mukaisia kyselyjä ja kyselyjen semantiikka.

Kuvan 20 kyselyiden semanttisesta tulkinnasta nähdään sama kuin vertailtaessa kuvia 4 (sivu 12) ja 12 (sivu 20); relaatiotietokannan taulun sarakkeet tulkitaan taulun lapsiksi. Nyt kuvassa 4 esiintynyt relaatiotietokannan *valtio*-taulu esiintyy kuvassa 12 XML-dokumentin juurena. Lisäksi *valtio*-solmun lapsina ovat sekä *valtio*-taulussa esiintyneet sarakkeet että taulut *kaupunki*, *joki* ja *järvi*. *Valtio*-taulun jälkeläisiä edellä mainittujen lisäksi ovat muun muassa *kaupunki*-, *joki*- ja *järvi*-taulujen sarakkeet. Nyt voidaan olettaa, että kyselyn (a) ollessa XIL-kysely, *valtio*-elementti viittaa relaatiotietokannassa taulun nimeen ja *nimi*-elementti viittaa relaatiotietokannassa joko tauluun, joka liittyy *valtio*-tauluun tai sarakkeeseen, joka on *valtio*-taulun sarake tai jonkin *valtio*-tauluun liittyvän taulun sarake. Se, kuinka muiden tietokannan taulujen oletetaan liittyvän *valtio*-tauluun, ei ole triviaalia, koska kyselyn tekijän ei oleteta tuntevan käytettävän tiedon osien rakennetta tai hierarkiaa. Kuvassa 21 on kuvattu, mihin elementteihin kuvan 20 XIL-kysely (a) viittaa esimerkkinä käytetyn valtiotietolähteen ollessa XML-muodossa. Noudatettaessa symmetristä tulkintaa, voidaan kuvassa 4 esitetystä relaatiotietokannasta valita kaikki *nimi*-sarakkeet, koska kaikki taulut liittyvät *valtio*-tauluun, ainakin muiden taulujen välityksellä. Toisaalta voitaisiin noudattaa tiukkaa tulkintaa, jolloin *valtio*-tauluun oletetaan liittyvän vain ne taulut, jotka sisältävät *valtio*-tauluun viittaavan vierasavaimen. Tällöin relaatiotietokannasta tulisi valituksi vain *valtio*-taulun *nimi*-sarake ja *kaupunki*-taulun *nimi*-sarake. Seuraavaksi esitellään, kuinka tiukan ja väljän tulkinnan käyttö on tutkimuksessa ratkaistu.



Kuva 21. Kuvassa 20 esitetyn XIL-kyselyn (a) viittaamat elementit kuvan 13 XML-dokumentissa.

6.1. Staattiset säännöt

Staattiset säännöt kuvaavat sen, voidaanko XIL-kyselyn polun solmun tulkita viittaavan vain relaatiotietokannan tauluun vai voidaanko sen olettaa viittaavan myös sarakkeeseen. Lisäksi staattiset säännöt kuvaavat sen, kuinka solmuun liitetty ehto tulkitaan. Tulkinta jaetaan erikseen SELECT-, FROM- ja WHERE-osiin. Ennen staattisia sääntöjä tulee tuntea XIL-kyselyn sisältämän polun rakenne. XIL-kysely sisältää aina SELECT-osan sekä enintään yhden WHERE-osan ja mahdollisesti yhdestä useaan FROM-WHERE -osaa. Lisäksi jokaisessa näistä osista esiintyy yhdestä useaan polkua. Kyselyn `SELECT nimi` tulkitaan sisältävän polun, jossa on yksi solmu, kyselyn `SELECT valtio/nimi` tulkitaan sisältävän polun, jossa on kaksi solmua, kyselyn `SELECT valtio/järvi/nimi` tulkitaan sisältävän polun, jossa on kolme solmua jne. Kysely `SELECT nimi, asukasluku` sisältää kaksi polkua. Kysely `SELECT valtio//järvi|joki/nimi` sisältää eksplisiittisesti yhden polun, mutta se voidaan lukea: valitse *valtio*-solmun alla olevan *järvi*-solmun *nimi*-solmu ja/tai *valtio*-solmun alla olevan *joki*-solmun *nimi*-solmu. Tällöin yksi polku voi sisältää useampia tulkintoja polulle ja näiden tulkintojen ei tarvitse olla toisensa poissulkevia. Toisin sanoen kysely tuottaa tuloksen, jos tietolähteestä löytyy *joki*-solmun *nimi*-lapsisolmu tai *järvi*-solmun *nimi*-lapsisolmu tai kummatkin.

Käännettäessä XIL-kyselyjä SQL:ksi SELECT- ja WHERE-osan sisältämien polkujen solmut, viimeistä solmua lukuun ottamatta, tulkitaan viittaavan aina tauluun ja viimeisen solmun tulkitaan viittaavan joko tauluun tai sarakkeeseen. Tällöin `SELECT` $s_1/s_2/\ldots s_n$ sisältää polun, jonka solmujen $s_1,s_2,\ldots,s_{n-1}$ tulkitaan viittaavan relaatiotietokannan tauluun ja solmun s_n tulkitaan viittaavan joko relaatiotietokannan tauluun tai sarakkeeseen. Kun SELECT-osan solmuun s_k on liitetty ehto ja s_k viittaa sarakkeeseen, liitetään ehto tähän sarakkeeseen. Kun SELECT-osan solmuun s_k on liitetty ehto ja s_k viittaa tauluun, liitetään ehto tämän taulun sarakkeisiin niin, että ehdon tulee olla tosi vähintään yhdessä näistä sarakkeista. Esimerkiksi XIL-kyselyn `SELECT kaupunki = Tampere` sisältämä *kaupunki*-solmu viittaa tauluun, mutta ehtoa ei ole mahdollista liittää tauluun. Tällöin riittää, että ehto on tosi

jossakin *kaupunki*-taulun sarakkeessa. Toisin sanoen rakenne tulkitaan sumeasti, koska kyselyn tekijän ei ole välttämätöntä tuntea käytettävän tietokannan rakennetta.

Vastaavasti *WHERE* $s_1/s_2/\ldots s_n$ sisältää polun, jonka solmujen $s_1,s_2,\ldots,s_{n-1}$ tulkitaan viittaavan relaatiotietokannan tauluun ja solmun s_n tulkitaan viittaavan joko relaatiotietokannan tauluun tai sarakkeeseen. Kun WHERE-osan solmuun s_k ($k \in \{1, \ldots, n\}$) on liitetty ehto ja s_k viittaa sarakkeeseen, liitetään ehto tähän sarakkeeseen. Kun WHERE-osan solmuun s_k on liitetty ehto ja s_k viittaa tauluun, liitetään ehto tämän taulun sarakkeisiin niin, että ehdon tulee olla tosi vähintään yhdessä näistä sarakkeista. Lisäksi kun WHERE-osan sisältämän polun tai polkujen viimeiseen solmuun ei ole lisätty ehtoa, tulee tähän solmuun liittää ehto *IS NOT NULL*. Tämä siksi, että XIL-kyselykielessä WHERE-osan solmu sisältää implisiittisesti ehdon, että kyseisen solmun on esiinnyttävä muodostetussa hierarkiassa. Jos solmu s_n on sarake, liitetään ehto tähän sarakkeeseen. Jos taas solmu s_n on taulu, liitetään ehto tämän taulun sarakkeisiin niin, että ehdon tulee olla tosi vähintään yhdessä näistä sarakkeista.

FROM-osan sisältämien polkujen jokaisen solmun tulkitaan viittaavan tauluun. Tämä johtuu siitä, että polussa kuljettaessa on päästävä eteenpäin. Lisäksi SELECT-osan on löydyttävä FROM-osan osoittamasta polusta, jolloin FROM-osan viimeisestäkin solmusta on päästävä eteenpäin. Vasta SELECT- ja WHERE-osan polkujen viimeisten solmujen kohdalla on mahdollista, että niistä ei päästä eteenpäin. XML-muodossa olevassa tiedossa lehtisolmuista ja relaatiomuodossa olevassa tiedossa sarakkeista ei pääse enää etenemään. Tästä ovat kuitenkin poikkeuksena vierasavaimena toimivat sarakkeet. Näiden sarakkeiden voidaan olettaa sisältävän viite tauluun ja näin mahdollistavan polussa etenemisen. Oletetaan yrityksen tietokannasta löytyvän taulut työntekijä ja huollettava. Lisäksi huollettava-taulu sisältää vierasavaimen huoltaja, joka viittaa työntekijä-tauluun. Tällöin kysely `SELECT huollettava FROM huoltaja WHERE nimi = Alpo Ahkera`, sisältää FROM-osassa viittauksen sarakkeeseen. Viitattu sarake on kuitenkin vierasavain, jolloin kyselyä käännettäessä oletetaan huoltajan tarkoittavan työntekijä-taulua.

Kun XIL-kyselyn solmuja sovitetaan relaatiotietokantaan, tarkistetaan ensin, mihin relaatiotietokannan osaan (taulu, vierasavain, sarake) solmu voi viitata. Kun tämä on tehty, pyritään solmu sovittamaan tämän osan esiintymiin. Tässä sovittamisella tarkoitetaan merkkijonojen samuutta. Solmun viitatessa vain tauluun, pyritään solmu sovittamaan ensiksi taulujoukon alkioihin ja lopuksi vierasavainjoukon alkioihin. Solmun viitatessa tauluun tai sarakkeeseen, pyritään solmu sovittamaan ensiksi taulujoukon alkioihin, seuraavaksi vierasavainjoukon alkioihin ja lopuksi sarakejoukon alkioihin. Taulujen ja vierasavainten kohdalla sovittaminen päätetään, kun ensimmäinen sopiva alkio on löydetty. Sarakkeiden kohdalla sovittamista jatketaan kunnes koko joukko on käyty läpi. Esimerkiksi XIL-kyselyn `SELECT kaupunki//nimi` sisältämän polun ensimmäinen solmu, *kaupunki*, voi viitata vain tauluun tai vierasavaimeen. Ensimmäisenä käydään läpi taulujoukko. Kohdistettaessa kysely esimerkkitietokantaan löytyy tietokannasta kaupunki-niminen taulu, jolloin oletetaan kyselyn

kaupunki-solmun viittaavan *kaupunki*-tauluun. XIL-kyselyn sisältämän polun toinen ja samalla viimeinen solmu voi viitata tauluun, vierasavaimeen tai sarakkeeseen. Esimerkkitietokannasta ei löydy *nimi*-nimistä taulua eikä vierasavainta, mutta esimerkkitietokannasta löytyy useita *nimi*-sarakkeita. Staattiset säännöt sallivat *nimi*-solmun viittaavan kaikkiin tietokannasta löydettyihin *nimi*-sarakkeisiin, mutta eivät tarjoa välineitä päätellä, mitkä näistä ovat relevantteja alkuperäisen kyselyn kannalta. Sen sijaan dynaamiset säännöt tarjoavat välineet tähän päättelyyn.

6.2. Dynaamiset säännöt

Kuten edellisessä luvussa todettiin, kyselyn kääntäminen ei onnistu pelkästään staattisten sääntöjen avulla. Näiden lisäksi tarvitaan myös dynaamisia sääntöjä. Nimitys dynaaminen sääntö tulee siitä, että nämä säännöt voivat muuttua tilanteen tai kyselyn tekijän johdosta. SQL-tietokannasta muodostettavissa oleva hierarkia, hierarkiasta valitut polut ja polussa kuljettavien askeleiden määrä ovat näitä sääntöjä. SQL-tietokannassa suhteet taulujen välillä voidaan määrittää vieras- ja pääavainten avulla. Lisäksi suhteita voitaisiin yrittää määrittää sarakkeiden nimien ja arvojen perusteella. Tutkimuksessa käytetään vierasavain/pääavain-suhteita ja oletuksena SQL-tietokannassa nämä suhteet on hyvin määritelty. SQL-tietokannan hierarkia muodostetaankin näiden suhteiden avulla, mutta tämä ei vielä yksin riitä. On lisäksi päätettävä, miten suhteita tulkitaan.

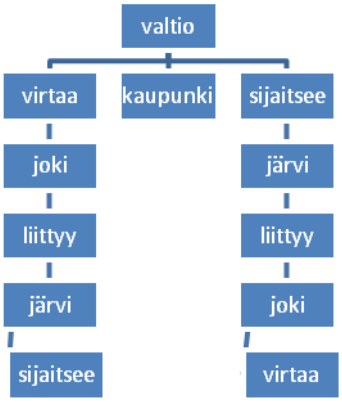
Tiukassa tulkinnassa vierasavaimen sisältävät taulut ovat viitattavan taulun lapsia, eli hierarkiassa alempana. Tällöin valtiotietokannassa kysely `SELECT kaupunki FROM valtio` tuottaisi tuloksen, mutta kysely `SELECT valtio FROM kaupunki` ei sitä tuottaisi, koska *valtio*-taulu ei sisällä *kaupunki*-tauluun viittaavaa vierasavainta. XILtoSQL-ohjelma mahdollistaa kummatkin kyselyt, koska kyselyn tekijän ei oleteta tuntevan tietolähteen rakennetta (luvussa 4.2. esitettyjen syiden lisäksi). Ohjelmassa suhteet tulkitaan symmetrisinä, jolloin suhde valtion ja kaupungin välillä toimii kumpaankin suuntaan. Tällöin SQL-tietokannan hierarkia syntyy sääntöjen lisäksi kyselystä. Ensimmäisessä kyselyssä valtio on hierarkiassa ennen kaupunkia ja toisessa kyselyssä kaupunki on ennen valtiota. Hierarkia pyritään siis muodostamaan käyttäjän ehdottamalla tavalla. Tilanteissa, joissa XIL-kyselyssä ilmaistaan eksplisiittisesti juurisolmu (`SELECT /valtio`) tai lehtisolmu (`SELECT kaupunki\`), käytetään juuri- ja lehtisolmujen selvittämiseen hierarkian tiukkaa tulkintaa.

Koska SQL-tietokanta voi sisältää rekursiivisia suhteita, tai suhteista voi muodostua silmukoita, päätetään hierarkia, kun suhteita ei enää löydy tai suhteeseen liittyvä taulu löytyy jo kyseisestä haarasta. Jos kuvan 4 relaatiotietokannasta muodostetaan hierarkia käyttämällä *valtio*-taulua hierarkian ylimpänä, muodostuu hierarkia kuvan 22 mukaiseksi. Käytettäessä *kaupunki*-taulua hierarkian ylimpänä, syntyy kuvan 23 mukainen hierarkia ja käytettäessä *järvi*-taulua hierarkian ylimpänä, syntyy kuvan 24 mukainen hierarkia. Jos sama taulu saisi esiintyä samassa haarassa useasti, tulisi rekursiosta ja hierarkiasta päättymätön. Tämä johtuu paitsi siitä, että SQL-tietokanta saattaa sisältää rekursiivisia

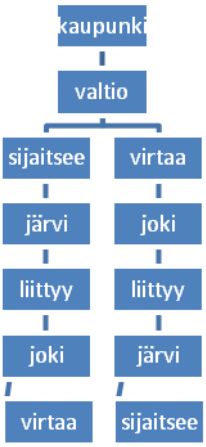
suhteita sekä suhteiden muodostamia silmukoita, mutta myös siitä, että tutkimuksessa suhteet tulkitaan symmetrisinä, mikä itsessään muodostaa silmukan.

Kun hierarkia on saatu muodostettua, pitää vielä päättää, mikä mahdollisista poluista valitaan. Esimerkiksi kuvassa 22 taulut *joki* ja *järvi* löytyvät hierarkiasta kahteen kertaan. XIL-kysely `SELECT joki FROM valtio` voitaisiin kääntää SQL:ksi käyttäen polkua *valtio-virtaa-joki* tai *valtio-sijaitsee-järvi-liittyy-joki*. Jos XIL-kyselyn tarkoituksena on saada kaikki valtion joet, tuottaa ensimmäinen polku oikean vastauksen. Toinen polku taas tuottaa ne joet, jotka liittyvät valtion alueella oleviin järviin. XILtoSQL-ohjelmassa käytetään oletuksena aina lyhyintä mahdollista polkua. Lisäksi, jos lyhyimpiä polkuja löytyy useita, valitaan näistä lyhyimmistä poluista ensimmäisenä löytynyt. Muita vaihtoehtoja olisi käyttää ohjelman ensimmäisenä löytämää polkua, mikä on toteutusriippuvainen tai kaikkia löytyneitä polkuja. Kolmantena asiana on polussa kuljettujen askelten määrä. XIL-kysely `SELECT nimi FROM kaupunki` viittaa kaupungin jälkeläisiin. Toisin sanoen kyselyn tulkitaan viittaavan paitsi *kaupunki*-taulussa olevaan sarakkeeseen *nimi*, myös *valtio*-, *joki*- ja *järvi*-tauluissa oleviin *nimi*-sarakkeisiin. Esimerkkinä käytetty valtiotietokanta on pieni ja siinä syntyvät polut ovat lyhyitä. Isommassa tietokannassa voi kuitenkin syntyä pitkiäkin polkuja ja tällöin tulee päättää, huomioidaanko kaikki mahdolliset *kaupunki*-tauluun liittyvät *nimi*-sarakkeet kyselyä käännettäessä. Tutkimuksessa tämä on ratkaistu rajoittamalla polun askelten määrää. Oletuksena käytetään viittä askelta, mutta askelten määrä on mahdollista muuttaa.

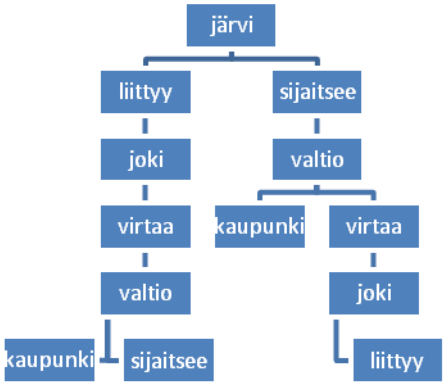
Yhteenvetona tässä tutkimuksessa hierarkia muodostetaan tietokannan suhteista, jotka määritellään vierasavain/pääavain-suhteiden avulla, poluista valitaan aina lyhyin polku ja polussa kuljettujen askelten määrä on asetettu viiteen.



Kuva 22. Valtio-relaatiotietokannasta muodostunut hierarkia, käytettäessä juurena *valtio*-taulua (taulujen sarakkeet on jätetty pois hierarkian yksinkertaistamiseksi).



Kuva 23. Valtio-relaatiotietokannasta muodostunut hierarkia, käytettäessä juurena *kaupunki*-taulua (taulujen sarakkeet on jätetty pois hierarkian yksinkertaistamiseksi).



Kuva 24. Valtio-relaatiotietokannasta muodostunut hierarkia, käytettäessä juurena *järvi*-taulua (taulujen sarakkeet on jätetty pois hierarkian yksinkertaistamiseksi).

6.3. SQL-kyselyn muodostus

Kuvassa 20 (sivu 36) esitetyille XIL-kyselyille (a) ja (c) ei esitetty semanttisesti samanlaista SQL-kyselyä. Edellä kuvattujen staattisten ja dynaamisten sääntöjen avulla se voidaan kuitenkin muodostaa. Ensinnäkin *valtio*-solmun oletetaan viittaavan relaatiotietokannassa tauluun ja *nimi*-solmun oletetaan viittaavan joko tauluun tai sarakkeeseen. Tämä pitää paikkansa valtiorelaatiotietokannassa. Relaatiotietokannasta löytyy *valtio*-taulu ja *nimi*-sarake löytyy tauluista *valtio*, *kaupunki*, *järvi* ja *joki*. Toiseksi oletetaan, että *nimi* on *valtion* jälkeläinen. Koska tietokannan hierarkia muodostetaan kyselyn perusteella, on *valtio* hierarkiassa ylimpänä. Tämän jälkeen jokainen taulu, josta löytyy *nimi*-sarake, pyritään liittämään hierarkiaan lyhyintä mahdollista polkua käyttäen. Lyhyin polku liittää *valtio* ja *järvi* on *valtio-sijaitsee-järvi*. *Valtio* ja *joki* liittyvät toisiinsa lyhyimmällä polulla *valtio-virtaa-joki* sekä

valtio ja *kaupunki* liittyvät toisiinsa polulla *valtio-kaupunki*. Taulut liitetään toisiinsa SQL:n vasemmalla ulkoliitoksella. Liittäminen tehdään järjestyksessä juurisolmuna toimivasta taulusta alkaen, jolloin kyselyssä muodostuu hierarkia. Vasenta ulkoliitosta käytetään, jotta vanhempisolmu voi esiintyä ilman lapsisolmua. Tämä siksi, että esimerkiksi kyselyssä `SELECT nimi FROM valtio`, ei haluta rajoittua vain niihin valtioihin, jotka sisältävät *kaupunki*-, *joki*- ja *järvi*-tauluja. Myös sellainen valtio, joka sisältää vain osan näistä tauluista tai ei mitään niistä, on validi. Tästä saadaan SQL-kysely:

```
SELECT valtio.nimi, järvi.nimi, joki.nimi, kaupunki.nimi FROM valtio
LEFT JOIN sijaitsee ON valtio.nimi = sijaitsee.valtio_nimi LEFT JOIN
järvi ON sijaitsee.järvi_nimi = järvi.nimi LEFT JOIN Virtaa ON
valtio.nimi = virtaa.valtio_nimi LEFT JOIN joki ON virtaa.joki_nimi =
joki.nimi LEFT JOIN kaupunki ON valtio.nimi = kaupunki.valtio_nimi
```

Tätä SQL-kyselyä tulee kuitenkin vielä täydentää useasta syystä. Näitä syitä on helpompi lähestyä sellaisen kyselyn kautta, jonka SELECT-osassa on annettu usea polku. Tällainen kysely on esimerkiksi `SELECT joki, järvi FROM valtio`. Tässä kyselyssä haetaan sekä jokea että järveä kohteesta valtio. Hierarkia aloitetaan yksiselitteisesti *valtio*-taulusta. SELECT-osan poluista voi muodostua kuitenkin hyvinkin erilaiset hierarkiat (hierarkian haarat) ja nämä voivat olla osittain päällekkäiset. Jotta tätä ei tarvitse huomioida SQL-kyselyssä, muodostetaan jokaisesta polusta oma alikysely varsinaisen SQL-kyselyn FROM-osaan. Jotta alikyselyjen tulokset voidaan yhdistää kyselyn osoittamalla tavalla, tulee alikyselyn SELECT-osaan lisätä FROM-osassa annettujen tai FROM-osasta muodostuneiden taulujen pääavaimet. Näiden pääavainten ja tavallisen liitoksen (INNER JOIN) avulla alikyselyiden tulostaulut yhdistyvät alkuperäisen kyselyn osoittamalla tavalla (ks. esim. luku 9 / kysely 9). Lisäksi SQL-kyselyn tulos tulee ryhmitellä kuten XILlissä. Tämä hoidetaan lisäämällä varsinaisen kyselyn SELECT-osaan hierarkian juuritaulun pääavain sarake tai sarakkeet ja liittämällä nämä ORDER BY -operaattoriin. Lopuksi varsinaisen kyselyn WHERE-osassa tulee vielä varmistaa, että alkuperäisen kyselyn SELECT-osan jokaisesta polusta on löytynyt tulos. Tämä varmistetaan lisäämällä jokaiseen polun viittaamaan sarakkeeseen IS NOT NULL -ehto ja yhdistämällä ehdot OR-operaattorilla. Lisäksi jokainen polku yhdistetään AND-operaattorilla. Yllä mainittuja asioita on pyritty havainnollistamaan kyselyn `SELECT joki, järvi FROM valtio` käännöksellä.

Alkuperäisen kyselyn ensimmäisestä polusta muodostuu kysely:

```
SELECT valtio.nimi, joki.nimi, joki.pituus, joki.laskujoki
FROM valtio LEFT JOIN virtaa ON valtio.nimi = virtaa.valtio_nimi LEFT
JOIN joki ON virtaa.joki_nimi = joki.nimi
```

Toisesta polusta muodostuu kysely:

```
SELECT valtio.nimi, järvi.nimi, järvi.pinta-ala
FROM valtio LEFT JOIN sijaitsee ON valtio.nimi = sijaitsee.valtio_nimi
LEFT JOIN järvi ON sijaitsee.järvi_nimi = järvi.nimi
```

Näistä muodostetaan varsinainen SQL-kysely:

```
SELECT t1.valtionimi, t1.jokinimi, t1.jokipituus, t1.jokilaskujoki,
t2.järvinimi, t2.järvipintaala
FROM
(SELECT valtio.nimi AS valtionimi, joki.nimi AS jokinimi, joki.pituus
AS jokipituus, joki.laskujoki AS jokilaskujoki FROM valtio LEFT JOIN
virtaa ON valtio.nimi = virtaa.valtio_nimi LEFT JOIN joki ON
virtaa.joki_nimi = joki.nimi) AS t1 INNER JOIN
(SELECT valtio.nimi AS valtionimi, järvi.nimi AS järvinimi,
järvi.pinta-ala AS järvipintaala FROM valtio LEFT JOIN sijaitsee ON
valtio.nimi = sijaitsee.valtio_nimi LEFT JOIN järvi ON
sijaitsee.järvi_nimi = järvi.nimi) AS t2 ON t1.valtionimi =
t2.valtionimi WHERE (t1.jokinimi IS NOT NULL OR t1.jokipituus IS NOT
NULL OR t1.jokilaskujoki IS NOT NULL) AND (t2.järvinimi IS NOT NULL OR
t2.järvipintaala IS NOT NULL)
ORDER BY t1.valtionimi
```

Kuvan 20 XIL-kyselyistä (a) ja (c) muodostuu edellä mainittujen muutosten jälkeen kysely:

```
SELECT t1.valtionimi, t1.järvinimi, t1.jokinimi, t1.kaupunkinimi
FROM
(SELECT valtio.nimi AS valtionimi, järvi.nimi AS järvinimi, joki.nimi
AS jokinimi, kaupunki.nimi AS kaupunkinimi
FROM valtio LEFT JOIN sijaitsee ON valtio.nimi = sijaitsee.valtio_nimi
LEFT JOIN järvi ON sijaitsee.järvi_nimi = järvi.nimi LEFT JOIN virtaa
ON valtio.nimi = virtaa.valtio_nimi LEFT JOIN joki ON virtaa.joki_nimi
= joki.nimi LEFT JOIN kaupunki ON valtio.nimi = kaupunki.valtio_nimi)
AS t1
WHERE (t1.valtionimi IS NOT NULL OR t1.järvinimi IS NOT NULL OR
t1.jokinimi IS NOT NULL OR t1.kaupunkinimi IS NOT NULL)
ORDER BY t1.valtionimi
```

XIL-kysely `SELECT nimi` eli kysely, jossa esiintyy vain attribuutti, on täysin validi XIL-kysely. Haasteena tämän kyselyn kääntämisessä on se, että kyselyssä ei ole annettu minkäänlaista hierarkiaa. Koska hierarkiaa ei ole annettu, pyritään muodostamaan kaikki mahdolliset hierarkiat. Tässäkin tapauksessa käytetään kuitenkin aina lyhyintä mahdollista polkua. Jokaisesta mahdollisesta hierarkiasta muodostetaan oma SQL-kysely. Esimerkkitietokannasta löytyy *nimi*-attribuutti *valtio*-, *kaupunki*-, *järvi*- ja *joki*-tauluista. Tällöin alkuperäisestä kyselystä muodostuu neljä SQL-kyselyä, joista ensimmäisessä hierarkian muodostus aloitetaan *valtio*-taulusta, toisessa *kaupunki*-taulusta, kolmannessa *järvi*-taulusta

ja neljännessä *joki*-taulusta. Nämä SQL-kyselyt löytyvät luvusta 9. Jos hierarkian muodostuksessa käytettäisiin muitakin polkuja kuin lyhyintä, muodostuisi käännöksessä enemmän kuin neljä SQL-kyselyä. XIL-kyselyä käännettäessä pyritään aina muodostamaan hierarkia, jotta tieto olioiden yhteyksistä tietokannassa säilyy. Lisäksi hierarkian muodostus mahdollistaa tuloksen ryhmittelyn samalla tavalla kuin se on toteutettu XILlissä.

Myös silloin, kun alkuperäisessä kyselyssä on esiintynyt putki-merkki, saattaa kyselyn käännöksestä syntyä useita SQL-kyselyitä. Tämä johtuu siitä, että vaihtoehtoisista solmun nimistä saattaa muodostua erilainen hierarkia. XIL-kyselystä `SELECT väkiluku|asukasluku FROM valtio` muodostuu vain yksi SQL-kysely, koska esimerkkitietokannasta ei löydy *asukasluku*-solmua. XIL-kyselystä `SELECT väkiluku FROM valtio|kaupunki` muodostuu kuitenkin kaksi SQL-kyselyä, koska esimerkkitietokannasta löytyy sekä *valtio*- että *kaupunki*-solmu. XIL-kyselystä saattaa siis generoitua useita SQL-kyselyitä, jos XIL-kysely ei sisällä yksiselitteistä juurisolmua tai kyselyssä esiintyy putki-merkkejä.

7. XILtoSQL – formaali kuvaus

XILtoSQL-käännöksen syntaksi ja semantiikka esitetään formaalisti attribuuttikieliopin avulla. Attribuuttikielioppi esitellään luvussa 7.1. Varsinainen XILtoSQL-käännös suoritetaan kahdessa osassa. Ensiksi XIL-kysely käännetään XILtoTuples-attribuuttikieliopin avulla yleistetyksi kyselyn monikkoesitykseksi ja seuraavaksi tämä monikkoesitys käännetään TuplesToSQL-attribuuttikieliopin avulla SQL-kyselyiksi. Nämä käännökset on kuvattu luvuissa 7.2. ja 7.3.

7.1. Käytetty formalismi

Attribuuttikieliopissa liitetään semantiikka syntaktiseen, kontekstivapaaseen (context-free), kielioppiin [Knuth, 1968]. Kontekstivapaan kieliopin avulla voidaan tarkistaa, onko jokin lause kieliopin mukainen eli onko lause syntaksiltaan oikeellinen. Tällainen syntaksiltaan oikeellinen lause voi olla kuitenkin semantiikaltaan mahdoton. Lauseen jäsennyksen tuloksena muodostuu jäsennyspuu. Attribuuttikieliopissa tähän jäsennyspuuhun lisätään attribuutteja, joiden avulla lauseen semantiikka määritellään. Lause jäsennettään terminaali- ja nonterminaalisymboleja käyttäen. Terminaalisymbolit ovat symboleita, joita ei voida enää purkaa pienempiin osiin eli lauseen tai sen osan jäsentämistä ei enää voida jatkaa. Nonterminaalisymbolit voidaan purkaa muiksi nonterminaalisymboleiksi tai terminaalisymboleiksi tai molemmiksi. Nonterminaalisymbolien kohdalla lauseen tai sen osan jäsentämistä jatketaan eteenpäin. Attribuuttikieliopissa esiintyy perittyjä ja synteettisiä attribuutteja. Perityt attribuutit määritellään jäsennyspuussa juuresta lehtiä kohti ja synteettiset attribuutit lehdistä juurta kohti.

Attribuuttikielioppi AG = <G, A, R> koostuu kontekstivapaasta kieliopista G, äärellisestä joukosta attribuutteja A ja äärellisestä joukosta semanttisia sääntöjä R. Kielen syntaksin määrittävä kontekstivapaa kielioppi on G = <N, T, P, D>, missä N on äärellinen joukko nonterminaalisymboleja, T on äärellinen joukko terminaalisymboleja, P on äärellinen joukko produktioita ja $D \in N$ on G:n aloitussymboli. Terminaalien ja nonterminaalien muodostaman joukon $V = N \cup T$ elementtejä kutsutaan kielioppisymboleiksi. P:hen kuuluvat produktiot esitetään muodossa $X \rightarrow \alpha$, missä $X \in N$ ja $\alpha \in V^*$ (sekvenssi kielioppisymboleja).

Äärellinen joukko attribuutteja A(Y) liitetään symboliin $Y \in V$, missä joukko A(Y) muodostuu kahdesta erillisestä (disjoint) joukosta I(Y), perityt attribuutit, ja S(Y), synteettiset attribuutit. Synteettisien attribuuttien tulos liitetään produktion vasemmanpuoleisille symboleille ja perittyjen attribuuttien tulos liitetään produktion oikeanpuoleisille symboleille. [Paakki, 1995] Attribuuttikielioppia on käytetty esimerkiksi ohjelmointikielten määrittelyssä. Attribuuttikieliopista on kehitetty erilaisia versioita, kuten S-attribuoitu ja L-attribuoitu kielioppi. S-attribuoidun ja perinteisen attribuuttikieliopin erot ovat siinä, että S-attribuoidussa kieliopissa ei esiinny lainkaan perittyjä

attribuutteja. L-attribuoitu kielioppi taas eroaa perinteisestä attribuuttikieliopista siinä, että attribuutit voidaan laskea yhdellä puun läpikäynnillä vasemmalta oikealle, jolloin jokaisessa sisäproduktiossa käydään ennen ja jälkeen lapsiproduktioiden läpikäynnin ja lapsiproduktioiden välissä.

Attribuuttikieliopin funktiot esitetään perinteisen joukko-opin notaatiolla. Monikoille (järjestetyille joukoille) käytetään seuraavaa merkintätapaa:

- Monikko ilmaistaan kulmasulkeiden välissä, esimerkiksi ⟨a,b,c⟩.
- Tyhjä monikko merkitään ⟨⟩.
- Monikon alkio voi olla itsessään monikko tai joukko.
- Monikon pituus saadaan len-funktiolla. Jos m on monikko ja siinä on n alkiota, len(m) = n.
- Monikon alkioon viitataan hakasulkeiden avulla. Jos m on monikko, tarkoittaa m[1] monikon m ensimmäistä alkiota ja m[last] monikon m viimeistä alkiota. Mikäli i saa arvot 1, …, len(m), merkintää m[i] käytetään määrittelemään monikko m. Merkintä m[i] on lyhennys merkinnästä $\overset{len(m)}{\underset{i=1}{m[i]}}$
- Mikäli x on monikon m alkio, merkitään x ∈ m.
- Olkoon m1 ja m2 kaksi monikkoa, missä m1 = ⟨x1, x2, x3⟩ ja m2 = ⟨y1, y2, y3⟩. Tällöin m1 ○ m2 = ⟨x1, x2, x3, y1, y2, y3⟩.
- Olkoon m1 ja m2 kaksi monikkoa, missä m1 = ⟨x1, x2, x3⟩ ja m2 = ⟨y1, y2, y3⟩. Tällöin m1 $○_{\vee}$ m2 = ⟨x1, x2, x3, OR, y1, y2, y3⟩.
- Olkoon m1 ja m2 kaksi monikkoa, missä m1 = ⟨x1, x2, x3⟩ ja m2 = ⟨y1, y2, y3⟩. Tällöin m1 $○_{\wedge}$ m2 = ⟨x1, x2, x3, AND, y1, y2, y3⟩.

Lisäksi

- Kommentit liitetään /* ja */ -merkkien väliin, esimerkiksi /* tämä on kommentti */.
- Loogisissa ehtovertailuissa yhtäsuuruutta kuvataan == -operaatiolla ja erisuuruutta != -operaatiolla.

7.2. XILtoTuples

XILtoTuples-attribuuttikielioppi kääntää oikeellisen XIL-kyselyn yleistetyksi kyselyn monikkoesitykseksi. Kysely käännetään monikkoesitykseksi, koska monikoita on helpompi käsitellä ohjelmallisesti sekä kääntää toisen attribuuttikieliopin avulla SQL-kyselyiksi.

7.2.1. Yleistetty kyselyn monikkoesitys

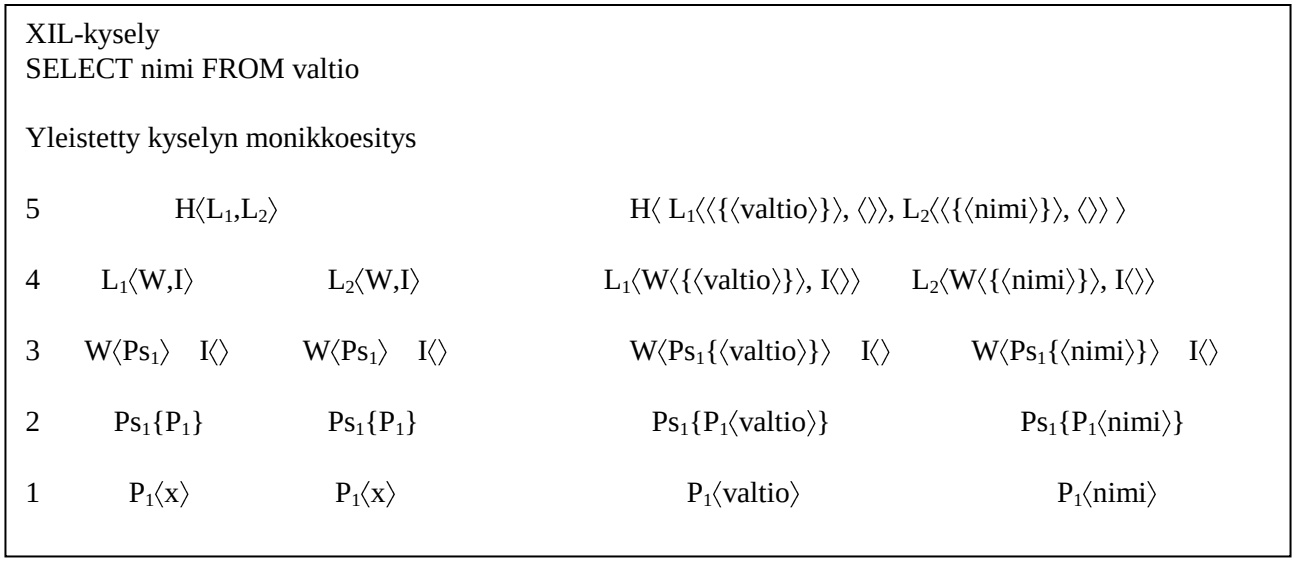
Yleistetty kyselyn monikkoesitys koostuu viisitasoisesta hierarkiasta, joka esitetään sisäkkäisinä monikkoina. Kuvassa 25 on havainnollistettu hierarkian tasot. Ylin taso esitetään H-monikkona (Hierarchy), joka koostuu L-pareista (Layer), eli $H = \langle L_1, L_2, \ldots, L_n \rangle$. L-pari on $\langle W, I \rangle$, missä W-monikko (What) sisältää kyselyalkiota (elementtejä/solmuja) ja I-monikko (Include) näihin kohdistuvia ehtoja. Esimerkiksi jäsennettäessä XIL-kyselyn SELECT-WHERE -osaa, SELECT-osa sijoitetaan W-monikkoon ja WHERE-osa sijoitetaan I-monikkoon. Vastaavasti jäsennettäessä FROM-WHERE -osaa FROM-osa sijoitetaan W-monikkoon ja WHERE-osa I-monikkoon. Sekä W että I koostuvat Ps-polkujoukoista (Paths). I-monikossa Ps-joukot on yhdistetty OR- tai AND-operaattoreilla, kuten alkuperäisessä kyselyssä. Yksittäinen polku esitetään P-monikkona (Path), joka koostuu solmun (elementin) nimistä (ENAME), viittauksesta juurisolmuun (/), polkuerottimista (//) tai viittauksesta lehtisolmuun (\). Solmun nimeen voi liittyä myös vertailuehto. Mikäli alkuperäinen polku sisältää putki-merkin, generoidaan oma polku kullekin vaihtoehtoiselle solmun nimelle. Nämä vaihtoehtoisen solmun sisältävät polut yhdistetään samaan Ps-joukkoon. Jos polku ei sisällä putki-merkkiä, Ps-joukkoon muodostuu yksi P-monikko.

H = Hierarchy	$H\langle L_1, L_2, \ldots, L_n \rangle$	taso 5
L = Layer	$L\langle W, I \rangle$	taso 4
W = What	$W\langle Ps_1, Ps_2, \ldots, Ps_m \rangle$	taso 3.1
I = Include	$I\langle Ps_1, y_1, Ps_2, y_2, \ldots, y_{l-1}, Ps_l \rangle$	taso 3.2
Ps = Paths	$Ps\{P_1, P_2, \ldots, P_k\}$	taso 2
P = Path	$P\langle x_1, x_2, \ldots, x_h \rangle$	taso 1
$x_i \in \text{ENAME} \cup \{/, //, \backslash\}$, $y_j \in \{\text{OR, AND}\}$		

Kuva 25. XILtoTuples-attribuuttikieliopin muodostaman yleistetyn monikkoesityksen rakenne.

Esimerkiksi kyselyä `SELECT nimi FROM valtio` vastaa monikko ⟨⟨⟨{⟨valtio⟩}⟩, ⟨⟩⟩, ⟨⟨{⟨nimi⟩}⟩, ⟨⟩⟩⟩. H-monikko sisältää siis kaksi L-paria ⟨⟨{⟨valtio⟩}⟩, ⟨⟩⟩ ja ⟨⟨{⟨nimi⟩}⟩, ⟨⟩⟩. Ensimmäisenä on tallennettu FROM-osa, joka koostuu vain solmun nimestä. Tähän ei ole liitetty WHERE-osaa, joten I esitetään tyhjänä monikkona. Sama pätee SELECT-osaan, johon on myös tallennettu vain solmun nimi. Monikon osat on esitetty yksityiskohtaisemmin kuvassa 26 siten, että vasemmalla puolella näkyvät kyselystä muodostuneiden monikoiden nimet ja oikealla puolella nämä monikot on täytetty kyselyssä esiintyvillä solmun nimillä. Hivenen mutkikkaampi kysely `SELECT nimi FROM järvi FROM valtio` käännetään monikkoesitykseksi ⟨⟨⟨{⟨valtio⟩}⟩,⟨⟩⟩, ⟨⟨{⟨järvi⟩}⟩,⟨⟩⟩, ⟨⟨{⟨nimi⟩}⟩,⟨⟩⟩⟩. Näin monikkoesitys ja sen alkioiden järjestys kuvaa kyselyyn sisältyvän hierarkian, jossa etsitään ensin

valtio-elementtiä, seuraavaksi *valtio*-elementin *järvi*-jälkeläiselementtejä ja lopuksi näiden *järvi*-elementtien *nimi*-jälkeläiselementtejä.



Kuva 26. Esimerkki XIL-kyselyn osien sijoittumisesta yleistettyyn kyselyn monikkoesitykseen.

Kuvassa 27 on esitetty kyselyä

```
SELECT nimi, asukasluku|väkiluku
FROM valtio WHERE järvi|joki/nimi = Näsijärvi
```

vastaava monikkoesitys ja se demonstroi tarpeen viisitasoiselle monikkoesitykselle. Kyselyn SELECT-osassa esiintyy kaksi polkua, FROM-osassa yksi polku ja FROM-osaan liittyvässä WHERE-osassa yksi polku. Kuitenkin WHERE-osan polulla ja toisella SELECT-osan polulla on kaksi mahdollista merkitystä, koska niissä esiintyy putki-merkki. SELECT-osa kertoo, että kyselyn kuvaamasta hierarkiasta etsitään *nimi*-elementtiä ja *asukasluku*- tai *väkiluku*-elementtiä. Toisin sanoen hierarkiasta on löydyttävä *nimi*-elementti ja ainakin toinen *asukasluku*- tai *väkiluku*-elementeistä. On kuitenkin mahdollista, että hierarkiasta löytyy kumpikin näistä elementeistä. WHERE-osassa annetaan ehto, jonka mukaan joko järven nimen tulee olla Näsijärvi tai joen nimen tulee olla Näsijärvi tai kummankin näistä tulee olla Näsijärvi. Kuvassa 27 ensimmäisellä tasolla kyselyn jokainen polku sekä polun jokainen merkitys on sijoitettuna omaan P-monikkoonsa. Nämä P-monikot sijoitetaan omiin Ps-joukkoihinsa, kuitenkin niin että alkuperäisen polun vaihtoehtoisia polkuja kuvaavat P-monikot sijoitetaan samaan Ps-joukkoon. Kolmannella tasolla jokaisen kyselyn osan sisältämät polut sijoitetaan kyseistä kyselyn osaa kuvaavaan W- tai I-monikkoon. Koska kyselyn FROM-osassa on yksi polku, sijoitetaan sitä kuvaavaan W-monikkoon yksi Ps-joukko, kyselyn WHERE-osassa on yksi polku, sijoitetaan sitä kuvaavaan I-monikkoon yksi Ps-joukko ja kyselyn SELECT-osassa on kaksi polkua, sijoitetaan sitä kuvaavaan W-monikkoon kaksi Ps-joukkoa. Jos W- ja I-monikot olisivat kaksiulotteisia

(vrt. kaksiulotteinen taulukko), ei monikkoesityksestä kävisi ilmi, mikä polku on vaihtoehtoinen ja mikä pakollinen. Neljännellä tasolla W- ja I-monikot sijoitetaan L-monikkoihin. L_1-monikkoon asetetaan sekä W- että I-monikko, koska FROM-osaan liittyy WHERE-osa. L_2-monikkoon asetetaan W-monikko sekä tyhjä monikko, koska SELECT-osaan ei liity WHERE-osaa. Lopuksi viidennellä tasolla L-monikot sijoitetaan H-monikkoon kuvaamaan kyselyn hierarkiaa ja sen muodostusjärjestystä.

Edellisen kyselyn muodostama hierarkia kohdistettaessa kysely esimerkkitietokantaan, on esitetty kuvassa 28. Kyselyn FROM-osan polusta saadaan *valtio*-juurisolmu. FROM-osaan liittyvä WHERE-osa liittää ehtoja FROM-osalle. Ehtoihin liittyvät polut on kuvattu kuvassa 28 vaaleilla laatikoilla. Sitä että ehto voi liittyä joko järven tai joen nimeen, on kuvattu näiden kahden polun väliin sijoitetulla OR-operaattorilla. Nämä ehdot liittyvät FROM-osaan, mutta eivät SELECT-osaan. Siksi ne on kuvattu kuvassa hieman erillään SELECT-osan tuottamista poluista. SELECT-osan polku *nimi* tuottaa esimerkkitietokannasta muodostettuun hierarkiaan neljä eri polkua. SELECT-osan polku *asukasluku*|*väkiluku* tuottaa hierarkiaan vain yhden polun, koska esimerkkitietokannasta ei löydy *asukasluku*-solmua. Jos *asukasluku*-solmu löytyisi, voitaisiin tämä kuvata sijoittamalla *väkiluku*-solmun viereen *asukasluku*-solmu ja sijoittamalla OR-operaattori näiden polkujen väliin. Kyselyn mahdollisesti palauttamat solmut on kuvattu mustalla fontilla. Sanaa mahdollisesti käytetään, koska on esimerkiksi täysin mahdollista, että johonkin valtioon ei tietokannassa ole liitetty yhtään jokea ja tällöin kysely ei tämän valtion kohdalla palauta jokien nimiä.

XIL-kysely
SELECT nimi, asukasluku|väkiluku FROM valtio WHERE järvi|joki/nimi = Näsijärvi

Yleistetty kyselyn monikkoesitys

5 **H**⟨ **L**$_1$⟨⟨{⟨valtio⟩⟩}⟩,⟨{⟨järvi,nimi=Näsijärvi⟩,⟨joki,nimi=Näsijärvi⟩}⟩⟩, **L**$_2$⟨⟨{⟨nimi⟩},{⟨asukasluku⟩,⟨väkiluku⟩}⟩, ⟨⟩⟩ ⟩

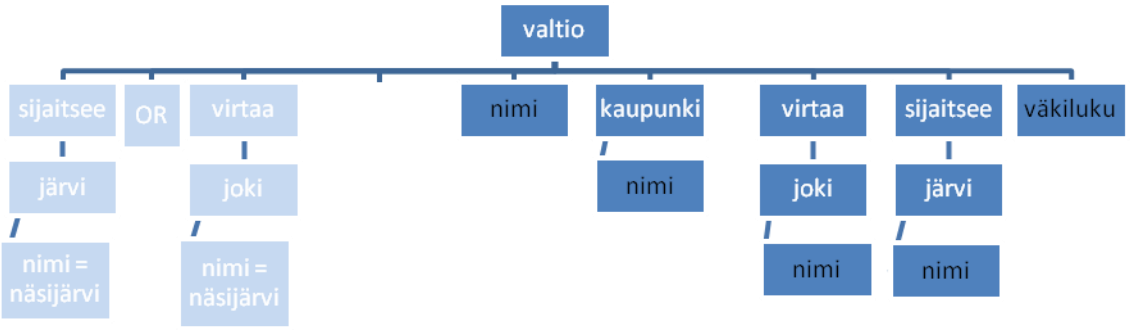
4 **L**$_1$⟨**W**⟨{⟨valtio⟩}⟩, **I**⟨{⟨järvi,nimi=Näsijärvi⟩, ⟨joki,nimi=Näsijärvi⟩}⟩⟩ **L**$_2$⟨**W**⟨{⟨nimi⟩},{⟨asukasluku⟩,⟨väkiluku⟩}⟩, **I**⟨⟩⟩

3 **W**⟨**Ps**$_1$⟨{⟨valtio⟩}⟩ **I**⟨**Ps**$_1$⟨{⟨järvi,nimi=Näsijärvi⟩, ⟨joki,nimi=Näsijärvi⟩}⟩ **W**⟨**Ps**$_1$⟨{⟨nimi⟩},**Ps**$_2$⟨{⟨asukasluku⟩,⟨väkiluku⟩}⟩ **I**⟨⟩

2 **Ps**$_1$⟨{**P**$_1$⟨valtio⟩} **Ps**$_1$⟨{**P**$_1$⟨järvi,nimi=Näsijärvi⟩, **P**$_2$⟨joki,nimi=Näsijärvi⟩} **Ps**$_1$⟨{**P**$_1$⟨nimi⟩} **Ps**$_2$⟨{**P**$_1$⟨asukasluku⟩,**P**$_2$⟨väkiluku⟩}

1 **P**$_1$⟨valtio⟩ **P**$_1$⟨järvi,nimi=Näsijärvi⟩ **P**$_2$⟨joki,nimi=Näsijärvi⟩ **P**$_1$⟨nimi⟩ **P**$_1$⟨asukasluku⟩ **P**$_2$⟨väkiluku⟩

Kuva 27. Toinen esimerkki XIL-kyselyn osien sijoittumisesta yleistettyyn kyselyn monikkoesitykseen.



Kuva 28. Kuvassa 27 esitetyn XIL-kyselyn muodostama hierarkia valtiotietokannassa. Vaaleat laatikot kuvaavat kyselyn ehtoja ja tummalla fontilla olevat laatikot kuvaavat kyselyn mahdollisesti palauttamia osia.

7.2.2. XILtoTuples-attribuuttikielioppi

Seuraavaksi esitellään attribuuttikielioppi, joka muuttaa XIL-kyselyn yleistetyksi monikkoesitykseksi. Kielioppi esitetään ensin kokonaisuudessaan, jonka jälkeen sen osia tarkastellaan kohta kohdalta yksityiskohtaisemmin.

XILtoTuples-attribuuttikielioppi on $AG_{XTT} = <G_{XTT},A_{XTT},R_{XTT}>$, missä kontekstivapaa kielioppi on $G_{XTT} = <N_{XTT}, T_{XTT}, P_{XTT}, Q>$, attribuuttien joukko on A_{XTT} = {pt, it, wt, lt, ht} ja R_{XTT} on P_{XTT}:n liittyvät semanttiset säännöt. XILtoTuples-attribuuttikielioppi sisältää vain synteettisiä attribuutteja eli on S-attribuoitu kielioppi.

Kontekstivapaassa kieliopissa nonterminaalien ja terminaalien joukot ovat

N_{XTT} = {Q, S, C, TP, P, FP, PP, T, EN, AN, FWS, FW, F, W, WC}

T_{XTT} = {SELECT, UNIQUE, FROM, WHERE, AND, OR, @, /, //, \, |, ‘,’} ∪ E-names ∪ A-names ∪ V ∪ {=, <, >, !=}, missä E-names on joukko elementtien (solmujen) nimiä ja A-names on joukko attribuuttien nimiä. V on joukko mahdollisten arvojen nimiä. Produktioiden joukko P_{XTT} on esitetty taulukon 1 vasemmassa sarakkeessa. Seuraavaksi tarkastellaan XIL-kielen jäsennystä näiden sääntöjen avulla, jonka jälkeen palataan attribuuttikieliopin muihin komponentteihin.

Taulukko 1. XILtoTuples S-attribuoituna attribuuttikielioppina.

productions	synthetic attributes
p1 Q → S	ht(Q) = ht(S)
p2 S → SELECT C	ht(S) = ⟨⟨wt(C), ⟨⟩⟩⟩
p3 S → SELECT C FWS	ht(S) = ht(FWS) ○ ⟨⟨wt(C), ⟨⟩⟩⟩
p4 S → SELECT C W	ht(S) = ⟨⟨wt(C), it(W) ⟩⟩
p5 S → SELECT C W FWS	ht(S) = ht(FWS) ○ ⟨⟨wt(C), it(W) ⟩⟩
p6 C → TP	wt(C) = ⟨pt(TP) ⟩
p7 C → TP, C2	wt(C) = ⟨pt(TP) ⟩ ○ wt(C2)
p8 TP → P	pt(TP) = pt(P)
p9 TP → UNIQUE P	pt(TP) = pt(P)
p10 P → FP	pt(P) = pt(FP)
p11 P → FP A[2]	pt(P) = pt(FP)
p12 FP → //PP	pt(FP) = pt(PP)
p13 FP → /PP	pt(FP) = {⟨/⟩ ○ x | x ∈ pt(PP)}
p14 FP → PP	pt(FP) = pt(PP)
p15 PP → EN	pt(PP) = pt(EN)
p16 PP → @AN	pt(PP) = pt(AN)
p17 PP → EN T	pt(PP) = {x ○ y | x ∈ pt(EN) ∧ y ∈ pt(T)}
p18 T → /PP	pt(T) = pt(PP)
p19 T → //PP	pt(T) = {⟨//⟩ ○ x | x ∈ pt(PP)}
p20 EN1 → EN2 | EN3	pt(EN1) = pt(EN2) ∪ pt(EN3)
p21 EN → ename\ O V	pt(EN) = {⟨ename O V, \⟩}
p22 EN → ename O V	pt(EN) = {⟨ename O V⟩}
p23 EN → ename\	pt(EN) = {⟨ename, \⟩}
p24 EN → ename	pt(EN) = {⟨ename⟩}
p25 AN → aname O V	pt(AN) = {⟨aname O V⟩}
p26 AN →aname	pt(AN) = {⟨aname⟩}
p27 FWS → FW	ht(FWS) = ⟨lt(FW) ⟩
p28 FWS1 → FW FWS2	ht(FWS1) = ⟨lt(FW) ⟩ ○ ht(FWS2)
p29 FW → F	lt(FW) = ⟨wt(F), ⟨⟩⟩
p30 FW → F W	lt(FW) = ⟨wt(F), it(W) ⟩
p31 F → FROM P	wt(F) = ⟨pt(P) ⟩
p32 W → WHERE WC	it(W) = it(WC)
p33 WC1 → WC2 OR WC3	it(WC1) = it(WC2) $\circ_{\vee}$ it(WC3)
p34 WC1 → WC2 AND WC3	it(WC1) = it(WC2) $\circ_{\wedge}$ it(WC3)
p35 WC → P	it(WC) = ⟨pt(P) ⟩
missä ename ∈E-names, aname ∈A-names, O∈ {=, <, >, !=}, V ∈ Number ∪ String	

[2] Työssä ei käsitellä ABOUT-ilmausta.

Produktiot ja XIL-kyselyn jäsennys

XIL-kyselyn jäsennys alkaa aloitussymbolista Q eli produktiosta p1. Produktiot p2, p3, p4 ja p5 vastaavat XIL-kyselyä seuraavasti: p2:ssa esiintyy vain SELECT-osa, p3:ssa SELECT- ja FROM-WHERE -osa, p4:ssa SELECT- ja WHERE-osa sekä p5:ssä SELECT-, WHERE- ja FROM-WHERE -osa.

Produktiot p6 ja p7 vastaavat SELECT-osan sisältöä, jossa ensimmäisessä on yksi polku ja toisessa kaksi tai useampia polkuja. Produktiot p8 ja p9 vastaavat SELECT-osan sisältämää polkua, josta erotetaan mahdollinen UNIQUE-ilmaus.

Produktiot p27 ja p28 vastaavat XIL-kyselyn FROM-WHERE -osia siten, että p27 vastaa yhtä FROM-WHERE -osaa ja p28 kahta tai useampaa FROM-WHERE -osaa. Produktiot p29 ja p30 vastaavat FROM-WHERE -osaa p29:ssä esiintyy vain FROM-osa ja p30:ssä sekä FROM- että WHERE-osa. Produktioissa p31 ja p32 erotetaan osien sisältö (polut). Produktiot p33, p34 ja p35 vastaavat WHERE-osan sisältöä (polkuja), joista p33:ssa on ensimmäinen ja toinen polku erotettu OR-ehdolla, p34:ssä on ensimmäinen ja toinen polku erotettu AND-ehdolla ja p35:ssä esiintyy vain yksi polku.

Produktiot p10-p14 vastaavat polkuja siten, että p10:ssä ei esiinny ABOUT-ilmausta, p11:sta ABOUT-ilmaus esiintyy, p12:sta polku alkaa kahdella kauttaviivalla, p13:sta yhdellä kauttaviivalla ja p14:sta polku ei ala kauttaviivalla. Produktiot p12 ja p14 vastaavat toisiaan ja ne vastaavat polkua, jossa polun ensimmäinen elementti voi viitata hierarkiassa mihin tahansa elementtiin. Produktio p13 vastaa polkua, jossa polun ensimmäinen elementti on oltava juurielementti.

Produktiot p15-p17 vastaavat polkua tai polun osaa siten, että p15:sta esiintyy yksi elementti, p16:sta attribuutti ja p17:sta kaksi tai useampia elementtejä (solmuja). Produktiot p18 ja p19 vastaavat polun osia siten, että p18 viittaa lapsielementtiin ja p19 jälkeläiselementtiin.

Produktiot p20-p26 vastaavat elementtien tai attribuuttien nimiä. Produktiossa p20 on kaksi vaihtoehtoista elementin nimeä. Produktiossa p21 elementin nimi päättyy kenoviivaan, joka tarkoittaa, että elementin on oltava lehtielementti ja elementtiin on liitetty ehtoja. Produktiossa p22 ja p25 elementin ja attribuutin nimeen on liitetty ehtoja. Produktiossa p23 elementin nimi päättyy kenoviivaan, eli elementin on oltava lehtielementti. Produktio p24 ja p26 vastaavat tilannetta, jossa elementin ja attribuutin nimeen ei ole liitetty ehtoja. Seuraavaksi tarkastellaan attribuutteja, joiden avulla jäsennetystä XIL-kyselystä muodostetaan yleistetty kyselyn monikkoesitys.

Attribuutit

Attribuuttikieliopin AG_{XTT} attribuutit joukossa A_{XTT} muuntavat XIL-kyselyn yleiseen kyselyn monikkoesitykseen. Kuten aikaisemmin on todettu, niin AG_{XTT} on S-attribuuttikielioppi eli kaikki attribuutit ovat synteettisiä. Attribuuteilla on seuraava yhteys yleistettyyn kyselyn monikkoesitykseen:

- pt-attribuutti vastaa XIL-kielen polkuilmauksia. Toisin sanoen tämä vastaa yleisen monikkoesityksen Ps-joukkoa, joka sisältää yhdestä useaan vaihtoehtoista polkua. Tämä attribuutti liittyy nonterminaaleihin TP, P, FP, PP, EN, AN ja T.
- it-attribuutti, joka tuottaa polkuilmausten joukkoja sisältävän monikon, vastaa I-monikkoa. Tämä attribuutti liittyy nonterminaaleihin W ja WC.
- wt-attribuutti, joka tuottaa polkuilmausten joukkoja sisältävän monikon, vastaa W-monikkoa. Tämä attribuutti liittyy nonterminaaleihin F ja C.
- lt-attribuutti sisältää W ja I monikoista koostuvia monikoita sisältävän monikon eli se vastaa L-monikkoa. Tämä attribuutti liittyy nonterminaaliin FW
- ht-attribuutti sisältää L-monikoista koostuvan monikon eli se vastaa korkeinta hierarkiatasoa (H-monikkoa). Tämä attribuutti liittyy nonterminaaleihin Q, S ja FWS.

Seuraavaksi tarkastellaan näiden käyttöä semanttisten sääntöjen yhteydessä.

Semanttiset säännöt

Produktioihin (P_{XTT}) liittyvät semanttiset säännöt (R_{XTT}) on esitetty taulukon 1 oikeanpuoleisessa sarakkeessa. Näiden sääntöjen soveltaminen aloitetaan produktiosta, johon jäsennys on päättynyt. Käytännössä tämä tarkoittaa XIL-kyselyn sisältämien polkujen yksittäisiä solmuja. Noudattaen jäsennykselle päinvastaista järjestystä yleistetty monikkoesitys konstruoidaan askel kerrallaan.

Produktiot p8-p26 liittyvät yksittäisten polkujen jäsentämiseen eli niihin liittyy pt-attribuutti. Produktioissa p21-p26 kyseinen attribuutti alustetaan. Säännössä p24 attribuutti alustetaan solmun nimellä. Sääntö p23 kohdistuu lehtielementtiin, eli attribuutti alustetaan parilla, joka koostuu solmun nimestä ja lehtisolmusymbolista. Säännöt p21 ja p22 sisältävät vertailuoperaation (O) näille tapauksille, joten myös nämä vertailuoperaattorit ja verrattava arvo liitetään esitykseen. Attribuutit käsitellään säännöissä p25 ja p26 vastaavasti. Säännössä p20 yhdistetään unionilla kaksi polkujoukkoa, koska sääntö sisältää vaihtoehtoisen määrittelyn solmun nimelle. Säännöt p12-p14 liittyvät kokonaisiin, vielä pilkkomattomiin, polkuihin. Säännöt p15-p19 liittyvät jo pilkottuihin polun osiin. Säännössä p19 asetetaan polkujoukon ensimmäiseksi alkioksi jälkeläiselementtiin viittaava symboli viittaamaan peräkkäisten solmujen jälkeläis-esivanhempi -suhteeseen. Säännössä p18 polkujoukkoa ei muuteta, koska kaksi peräkkäistä polkujoukon solmua liittyvät toisiinsa oletusarvoisesti lapsi-vanhempi -suhteella. Säännössä p17 liitetään polun jäsennyksessä erotetut osat toisiinsa. Säännöissä p14-p16 attribuutteja ei muuteta, koska näissä produktioissa ei jäsennyksen aikana olla erotettu monikkoesityksessä tarvittavia tietoja. Säännössä p13 liitetään polkujoukon ensimmäiseksi solmuksi juurisolmuun viittaava symboli, koska polusta on jäsennyksen aikana erotettu juurisolmuun viittaava symboli. Sääntö p12 ei muuta polkujoukkoa, koska polkujoukon ensimmäinen solmu viittaa oletuksena

mihin tahansa tietokannan tauluun. Säännöissä p10 ja p11 ei muuteta polkujoukkoja. Säännöt p8 ja p9 liittyvät SELECT-osan polkujen jäsennykseen. Näissä säännöissä ei muuteta polkujoukkoa.

Produktiot p6 ja p7 liittyvät SELECT-osan polkujen jäsentämiseen ja niihin liittyy wt-attribuutti. Säännössä p7 yhdistetään kaksi tai useampia pt-attribuutteja, koska sääntö sisältää kahdesta useaan polkua. Säännössä p6 wt-attribuutti alustetaan pt-attribuutilla, koska sääntö sisältää yhden polun. Produktiot p32-p35 liittyvät WHERE-osan polkujen jäsentämiseen eli niihin liittyy it-attribuutti. Säännössä p35 it-attribuutti alustetaan pt-attribuutilla, koska sääntö sisältää vain yhden polun. Säännössä p33 liitetään kaksi it-attribuuttia yhteen OR-operaattorilla, koska sääntö sisältää kaksi vaihtoehtoista polkua. Säännössä p34 liitetään kaksi it-attribuuttia yhteen AND-operaattorilla, koska sääntö sisältää kaksi ei-vaihtoehtoista polkua. Säännössä p32 ei attribuuttia muuteta, koska polusta ei jäsennyksen aikana olla erotettu monikkoesityksessä tarvittavia tietoja. Produktio p31 liittyy FROM-osan jäsennykseen eli siihen liittyy wt-attribuutti. Säännössä p31 wt-attribuutti alustetaan pt-attribuutilla. FROM-osaan voi liittyä vain yksi polku, joten ei tarvita produktiota ja sääntöä, jossa jäsennetään ja liitetään useampia polkuja.

Produktiot p29 ja p30 liittyvät yhteen FROM-WHERE -osaan eli niihin liittyy lt-attribuutti. Säännössä p29 lt-attribuutti alustetaan parilla, joka koostuu wt-attribuutista ja tyhjästä monikosta, koska wt-attribuuttiin ei kohdistu ehtoja. Säännössä p30 lt-attribuutti alustetaan parilla, joka koostuu wt-attribuutista ja it-attribuutista. Produktiot p27 ja p28 liittyvät yhdestä useaan FROM-WHERE -osaan eli niihin liittyy ht-attribuutti. Säännössä p27 ht-attribuutti alustetaan lt-attribuutilla, koska sääntöön liittyy vain yksi FROM-WHERE -osa (L-monikko). Säännössä p28 ht-attribuutti alustetaan kahdella tai useammalla lt-attribuutilla, koska sääntöön liittyy kaksi tai useampia FROM-WHERE -osia (L-monikoita).

Produktiot p1-p5 liittyvät koko kyselyn jäsentämiseen eli niihin liittyy ht-attribuutti. Säännössä p5 liitetään ht-attribuutti, joka sisältää yhdestä useaan L-monikkoa (FROM-WHERE -osaa), ja ⟨wt, it⟩-parista (SELECT-WHERE) koostuva monikko yhteen. Säännössä p4 alustetaan ht-attribuutti ⟨wt, it⟩-parilla (SELECT-WHERE), koska sääntö ei sisällä muita L-monikoita (FROM-WHERE -osia). Säännössä p3 liitetään ht-attribuutti, joka sisältää yhdestä useaan L-monikkoa (FROM-WHERE -osaa), ja ⟨wt, ⟨⟩⟩-parista koostuva monikko yhteen. Parin toinen alkio on tyhjä, koska säännössä W-monikkoon (SELECT-osa) ei liity ehtoja. Säännössä p2 alustetaan ht-attribuutti ⟨wt, ⟨⟩⟩-parilla, koska sääntö ei sisällä muita L-monikoita (FROM-WHERE -osia). Parin toinen alkio on tyhjä samasta syystä kuin edellä. Säännössä p1 päätetään yleistetyn kyselyn monikkoesityksen konstruointi.

Jäsennysesimerkki

XIL-kyselyn `SELECT nimi` jäsentäminen XILtoTuples-attribuuttikieliopin avulla esitetään kuvassa 29. Tässä jäsentäminen aloitetaan kohdasta p1. Seuraavaksi siirrytään kohtaan p2, koska kysely sisältää vain SELECT-osan. SELECT-osasta erotetaan osan sisältö ja siirrytään kohtaan p6, koska sisältö

sisältää vain yhden polun. Jäsentämistä jatketaan kohtaan p8, koska polku ei ala UNIQUE-määreellä ja tästä kohtaan p10, koska polku ei sisällä myöskään ABOUT-ilmausta. Jäsentämistä jatketaan edelleen kohtaan p14, koska polku ei ala kauttaviivalla ja kohtaan p15, koska polussa sijaitsee vain yksi elementti, eikä se viittaa XML-attribuuttiin. Seuraavaksi jäsentämisessä siirrytään lehtielementtiin p24, koska elementin nimeen ei sisälly ehtoja tai viittausta lehteen (\). Kyselyn jäsentäminen on saatu päätökseen ja seuraavaksi siirrytään muodostamaan kyselyn semantiikkaa.

Elementin nimi asetetaan pt-attribuuttiin seuraavasti: pt(EN) = {⟨nimi⟩}. Kohdissa p15, p14, p10 ja p8 synteettisten attribuuttien kuvaamiin joukkoihin ei tehdä muutoksia, vaan ne asetetaan kyseisen kohdan monikkoon. Kohdassa p6 asetetaan pt(TP) wt(C)-monikon alkioksi. Tämä kuvaa sitä, että kyseiseen kyselyn osaan liittyy vain yksi polku. Kohdassa p2 asetetaan ht(S)-monikkoon alkioksi wt(C)-monikon ja tyhjän monikon muodostama pari. Kohdassa p1 asetetaan monikon ht(S) arvo monikkoon ht(Q). Tuloksena tästä on monikkoesitys ⟨⟨⟨{⟨nimi⟩}⟩, ⟨⟩⟩⟩. Muut XILtoTuples-jäsennysesimerkit löytyvät liitteen 3 kohdista (1), (3) ja (4).

p1 S = SELECT nimi
p2 C = SELECT nimi
p6 TP = nimi
p8 P = nimi
p10 FP = nimi
p14 PP = nimi
p15 EN = nimi
p24 pt(EN) = {⟨nimi⟩}
p15 pt(PP) = pt(EN)
p14 pt(FP) = pt(PP)
p10 pt(P) = pt(FP)
p8 pt(TP) = pt(P)
p6 wt(C) = ⟨pt(TP) ⟩
p2 ht(S) = ⟨⟨wt(C), ⟨⟩⟩⟩ = ⟨⟨⟨{⟨nimi⟩}⟩, ⟨⟩⟩⟩
p1 ht(Q) = ht(S) = ⟨⟨⟨{⟨nimi⟩}⟩, ⟨⟩⟩⟩

Kuva 29. Esimerkki XIL-kyselyn kääntämisestä monikkoesitykseksi attribuuttikieliopin avulla.

7.3. TuplesToSQL

XILtoTuples-kieliopin tuottama yleistetty kyselyn monikkoesitys käännetään SQL-kyselyiksi TuplesToSQL-attribuuttikieliopin avulla. Monikkoesitys voidaan kääntää SQL-kyselyiksi vain relaatiotietokantaa vastaan. Tämä tarkoittaa, että käännös on tietokantaspesifi, eli ennen käännöksen

aloittamista on tiedettävä tietokannan rakenne. Tähän rakenteeseen kuuluu tieto tauluista, taulujen sarakkeista ja taulujen välisistä suhteista.

7.3.1. Relaatiotietokannan rakenne

Relaatiotietokannan rakenteesta muodostetaan joukot *Tables*, *Rtables*, *LFTables*, *Cols*, *PKCols*, *FKCols* ja *Edges*.

- *Tables* sisältää kaikki tietokannan taulut taulunumero-taulunimi -pareina. Esimerkkitietokannasta saadaan joukko *Tables'* = {⟨1, virtaa⟩, ⟨2, joki⟩, ⟨3, liittyy⟩, ⟨4, järvi⟩, ⟨5, sijaitsee⟩, ⟨6, valtio⟩, ⟨7, kaupunki⟩}.
- *Rtables*, eli juuritaulut, sisältää kaikki tietokannan taulut, jotka eivät sisällä vierasavainta. *Rtables'* = {⟨4, järvi⟩, ⟨6, valtio⟩}.
- *LFTables*, eli lehtitaulut, sisältää kaikki tietokannan taulut, joihin ei viitata vierasavaimen kautta mistään tietokannan taulusta. *LFTables'* = {⟨1, virtaa⟩, ⟨3, liittyy⟩, ⟨5, sijaitsee⟩, ⟨7, kaupunki⟩}.
- *Cols* sisältää kaikki tietokannan sarakkeet taulunumero-sarakenimi -pareina. *Cols'* = {⟨1,valtio_nimi⟩, ⟨1,joki_nimi⟩, ⟨2,nimi⟩, ⟨2,pituus⟩, ⟨2,laskujoki⟩, ⟨3,joki_nimi⟩, ⟨3,järvi_nimi⟩, ⟨4,nimi⟩, ⟨4,pintaala⟩, ⟨5,valtio_nimi⟩, ⟨5,järvi_nimi⟩, ⟨6,nimi⟩, ⟨6,valtiomuoto⟩, ⟨6,väkiluku⟩, ⟨7,nimi⟩, ⟨7,väkiluku⟩, ⟨7,valtio_nimi⟩}.
- *PKCols* sisältää kaikki tietokannan pääavainsarakkeet taulunumero-pääavain -pareina. *PKCols'* = {⟨1, valtio_nimi⟩, ⟨1, joki_nimi⟩, ⟨2, nimi⟩, ⟨3, joki_nimi⟩, ⟨3, järvi_nimi⟩, ⟨4, nimi⟩, ⟨5, valtio_nimi⟩, ⟨5, järvi_nimi⟩, ⟨6, nimi⟩, ⟨7, nimi⟩}.
- *FKCols* sisältää kaikki tietokannan vierasavainsarakkeet lähtötaulu-kohdetaulu-vierasavain -monikkoina, niin että lähtötaulu on vierasavaimen sisältävän taulun numero, kohdetaulu on vierasavaimen viittaaman taulun numero ja vierasavain on vierasavainsarakkeen nimi. *FKCols'* = {⟨1, 6, valtio_nimi⟩, ⟨5, 6, valtio_nimi⟩, ⟨7, 6, valtio_nimi⟩, ⟨1, 2, joki_nimi⟩, ⟨3, 2, joki_nimi⟩, ⟨2, 2, laskujoki⟩, ⟨3, 4, järvi_nimi⟩, ⟨5, 4, järvi_nimi⟩}.
- *Edges* sisältää kaikki tietokannan taulujen väliset suhteet symmetrisinä, taulunumero-taulunumero -pareina annettuina. *Edges'* = {⟨1,6⟩, ⟨6,1⟩, ⟨1,2⟩, ⟨2,1⟩, ⟨2,2⟩, ⟨2,3⟩, ⟨3,2⟩, ⟨3,4⟩, ⟨4,3⟩, ⟨4,5⟩, ⟨5,4⟩, ⟨5,6⟩, ⟨6,5⟩, ⟨6,7⟩, ⟨7,6⟩}.

7.3.2. TuplesToSQL-attribuuttikielioppi

Seuraavaksi esitellään attribuuttikielioppi, joka muuttaa yleistetyn kyselyn monikkoesityksen SQL-kyselyiksi. Kielioppi esitetään ensin kokonaisuudessaan, jonka jälkeen sen osia tarkastellaan kohta kohdalta yksityiskohtaisemmin.

TuplesToSQL-attribuuttikielioppi on AG_{TTS} = <G_{TTS},A_{TTS},R_{TTS}>, missä kontekstivapaa kielioppi on G_{TTS} = <N_{TTS}, T_{TTS}, P_{TTS}, Q>, attribuuttien joukko on A_{TTS} = {c, t, qs, qss, sq} ja R_{TTS} on P_{TTS}:n liittyvät semanttiset säännöt.

Kontekstivapaassa kieliopissa nonterminaalien ja terminaalien joukot ovat

N_{TTS} = {S, H, L, Hu, W, I, Ps, P, Pc, EN}

T_{TTS} = {⟨, ⟩, {, }, ‘,’, /, //, \, fk, t, c, r}

Produktioiden joukko P_{TTS} on esitetty taulukon 2 vasemmassa sarakkeessa. Seuraavaksi tarkastellaan yleistetyn kyselyn monikkoesityksen jäsennystä näiden sääntöjen avulla, jonka jälkeen palataan attribuuttikieliopin muihin komponentteihin.

Taulukko 2. TuplesToSQL-attribuuttikielioppi.

productions	inherited attributes	synthetic attributes
p1 S → H		sq(S) = *string*(qss(H))
p2 H → ⟨L⟩	c(L) = 2	qss(H) = qss(L)
p3 H → Hu ∘ ⟨L⟩	c(Hu) = 1, c(L) = 2	qss(H)[i] = z: z[j] = ⟨z1, z2, z3, z4⟩: z2 =*hi*(x2, y2) ∧ z1 = *re*(z2, y1) ∧ z3 = x3 ∘ *we*(z2, y3) ∧ z4 = x4 ∘ (y4′[p] = (t′[l] = ⟨t1′,t2′,t3′⟩: t2′ = *hi*(z2,t2) ∧ t3′ = *we*(t2′,t3) ∧ t1′ = t1) ∨ (t′[l] = t[l]: t[l] == AND/OR)): y4[p] = t: t[l] = ⟨t1,t2,t3⟩ ∨ AND/OR: l ∈ {1, …, len(t)}: p∈{1, …, len(y4)}): qss(Hu)[r] = ⟨⟨⟩,x2,x3,x4⟩: r ∈ {1, …, len(qss(Hu))} ∧ qss(L)[k] = y: y[h] = ⟨y1,y2,y3,y4⟩: h ∈ {1, …, len(y)}: k∈{1, …, len(qss(L))}: i ∈ {1, …, len(qss(Hu))* len(qss(L))}
p4 Hu → ⟨L⟩	c(L) = c(Hu)	qss(Hu)[i] = ⟨⟨⟩, z2, z3, z4⟩: z2 = *hi*(x2) ∧ z3 = *we*(z2, x3) ∧ z4 = (x4′[j] = (t′[l] = ⟨t1′,t2′,t3′⟩: t2′ = *hi*(z2,t2) ∧ t3′ = *we*(t2′,t3) ∧ t1′ = t1) ∨ (t′[l] = t[l]: t[l] == AND/OR)): x4[l] = t: t[l] = ⟨t1,t2,t3⟩ ∨ AND/OR: l ∈ {1, …, len(t)}: j ∈ {1, …, len(y4)}): qss(L)[i] = ⟨⟨⟩,x2,x3,x4⟩: i ∈ {1, …, len(qss(L))}:
p5Hu1 → Hu2 ∘ ⟨L⟩	c(Hu2) = c(Hu1), c(L) = c(Hu1)	qss(Hu1)[i] = ⟨⟨⟩, z2, z3, z4⟩: z2 = *hi*(x2, y2) ∧ z3 = x3 ∘ *we*(z2, y3) ∧ z4 = x4 ∘ (y4′[p] = (t′[l] = ⟨t1′,t2′,t3′⟩: t2′ = *hi*(z2,t2) ∧ t3′ = *we*(t2′,t3) ∧ t1′ = t1) ∨ (t′[l] = t[l]: t[l] == AND/OR)): y4[p] = t: t[l] = ⟨t1,t2,t3⟩ ∨ AND/OR: l ∈ {1, …, len(t)}: p ∈ {1, …, len(y4)}): qss(Hu2)[j] = ⟨⟨⟩,x2,x3,x4⟩: j ∈ {1, …, len(qss(Hu2))}: qss(L)[k] = ⟨⟨⟩,y2,y3,y4⟩: k ∈ {1, …, len(qss(L))}: i ∈ {1, …, len(qss(Hu2))*len(qss(L))}
p6 L → ⟨W, I⟩	c(W) = c(L), c(I) = 3	qss(L)[i] = z: z[k] = ⟨z1, z2, z3, z4⟩: z1 = x1 ∧ z2 = x2 ∧ z3 = x3 ∧ (z4[j] = r: (r[p] = ⟨r1,r2,r3⟩: r2 = y2 ∧ r3 = y3 ∧ r1 = {⟨⟨n,t⟩,c⟩ | ⟨n,c⟩∈*PKCols* ∧ (z2[last] = ⟨⟨n,t⟩,⟨⟩⟩ ∨ z2[last] = ⟨_, ⟨n,t⟩⟩)}) ∨ (r[p] = y[p]: y[p] == AND/OR): qss(I)[j]=y: y[p]=⟨⟨⟩,y2,y3,⟨⟩⟩ ∨ AND/OR: p∈{1, …, len(y)}: j∈{1, …, len(qss(I))}): qss(W)[i]=x: x[k]=⟨x1,x2,x3, ⟨⟩⟩: k∈{1, …, len(x)}: i∈{1, …, len(qss(W))}
p7 L → ⟨W, ⟨⟩⟩	c(W) = c(L)	qss(L) = qss(W)
p8 W →⟨Ps⟩	c(Ps) = c(W)	qss(W) = qss(Ps)
p9 W1 → ⟨Ps⟩ ∘ W2	c(Ps) = c(W1), c(W2) = c(W1)	qss(W1)[i] = qss(Ps)[k] ∘ qss(W2)[j]: k ∈ {1, …, len(qss(Ps))} ∧ j ∈ {1, …, len(qss(W2))} ∧ i ∈ {1, …, len(qss(Ps)) * len(qss(W2))}
p10 I → ⟨Ps⟩	c(Ps) = c(I)	qss(I) = qss(Ps)
p11 I1 →⟨Ps⟩ AND/OR I2	c(Ps) = c(I1), c(I2) = c(I1)	qss(I1)[i] = qss(Ps)[k] $\circ_{\vee/\wedge}$ qss(I2)[j]: k ∈ {1, …, len(qss(Ps))} ∧ j ∈ {1, …, len(qss(I2))} ∧

		i ∈ {1, …, len(qss(Ps)) * len(qss(I2))}
p12 Ps → {P}	c(P) = c(Ps)	qss(Ps) = ⟨qs(P)⟩
p13 Ps1 → {P} ∪ Ps2, Ps2=(Ps\{P})	c(P) =c(Ps1), c(Ps2) = c(Ps1)	qss(Ps1) = ⟨qs(P)⟩ ○ qss(Ps2)
p14 P → ⟨//⟩ ○ Pc	t(Pc) = *RTables*, c(Pc) = c(P)	qs(P) = qs(Pc)
p15 P → Pc	t(Pc) = *Tables*, c(Pc) = c(P)	qs(P) = qs(Pc)
p16 Pc → ⟨EN⟩	t(EN) = t(Pc), c(EN) = c(Pc)	qs(Pc) = qs(EN)
p17 Pc → ⟨EN, \⟩	t(EN) = t(Pc) ∩ *LFTables*, c(EN)=c(Pc)	qs(Pc) = qs(EN)
p18 Pc1 → ⟨EN⟩ ○ Pc2	t(EN) = t(Pc1), t(Pc2) = *Tables*, c(EN) = 1, c(Pc2) = c(Pc1)	qs(Pc1) = ⟨⟨z1, z2, z3, ⟨⟩⟩⟩: (z1 = ⟨⟨⟨i,k⟩,c⟩⟩ ∧ z2 = ⟨⟨⟨i,k⟩,⟨⟩⟩⟩ : ⟨⟨i,l⟩,c⟩ Ɛ y1 ∧ x2 = ⟨⟨⟨i,k⟩,⟨⟩⟩⟩ ∧ (z3 = ⟨⟨⟨⟨i,k⟩,c,r⟩⟩⟩ ○ x3 : ⟨⟨i,l⟩,c,r⟩ Ɛ y3[1] ∨ z3 = x3 : ∄ ⟨⟨i,l⟩,c,r⟩ Ɛ y3[1])) **jos y2==⟨⟩ ∧ (c(Pc1)==1 V c(Pc1)==2)** (z1 = y1 ∧ z3 = x3 ○ y3 ∧ z2 = ⟨⟨⟨i,k⟩,⟨j,l⟩⟩⟩ ○ y2: x2 = ⟨⟨⟨i,k⟩,⟨⟩⟩⟩ ∧ y2[1] = ⟨⟨j,l⟩,_⟩ ∧ ⟨i,j⟩ ∈ *Edges*) **jos y2 != ⟨⟩ ∧ (c(Pc1) == 1Vc(Pc1) == 2)** (z3 = ⟨⟨⟨⟨i,k⟩,c,r⟩⟩⟩ ○ x3 ∧ z2 = ⟨⟨⟨i,t⟩,⟨⟩⟩⟩: ⟨⟨i,k⟩,c,r⟩ Ɛ y3[1] ∧ x2 = ⟨⟨⟨i,t⟩,⟨⟩⟩⟩) **jos y2 == ⟨⟩ ∧ c(Pc1) == 3** (z2 = ⟨⟨⟨i,t⟩,⟨j,k⟩⟩⟩ ○ y2 ∧ z3 = x3 ○ y3: x2 = ⟨⟨⟨i,t⟩,⟨⟩⟩⟩ ∧ y2[1] = ⟨⟨⟨j,k⟩,_⟩⟩ ∧ ⟨i,j⟩ ∈ *Edges*) **muutoin /* y2 != ⟨⟩ ∧ c(Pc1) == 3 */** missä qs(EN) = ⟨⟨⟨⟩,x2,x3, ⟨⟩⟩⟩ ∧ qs(Pc2) = ⟨⟨y1,y2,y3, ⟨⟩⟩⟩
p19 Pc1 → ⟨EN⟩ ○ ⟨//⟩ ○ Pc2	t(EN) = t(Pc1), t(Pc2) = *Tables*, c(EN) = 1, c(Pc2) = c(Pc1)	qs(Pc1) = ⟨⟨z1,z2,z3, ⟨⟩⟩⟩: z1 = y1 ∧ z2 = x2 ○ y2 ∧ z3 = x3 ○ y3 missä qs(EN) = ⟨⟨⟨⟩,x2,x3, ⟨⟩⟩⟩ ∧ qs(Pc2) = ⟨⟨y1,y2,y3, ⟨⟩⟩⟩
p20 EN → fk	/*c(EN)==1|2|3*/	qs(EN) = ⟨⟨⟨⟩,⟨⟨⟨i,fk⟩,⟨⟩⟩⟩,⟨⟩,⟨⟩⟩⟩: ⟨i,j,fk⟩ ∈ *FKCols* ∧ ⟨i,t⟩ ∈ t(EN)
p21 EN → fk r	/*c(EN)==1|2|3*/	qs(EN) = ⟨⟨⟨⟩,⟨⟨⟨i,fk⟩,⟨⟩⟩⟩,⟨z⟩,⟨⟩⟩⟩: z[j] = ⟨⟨i,t⟩,c,r⟩: ⟨i,j,fk⟩ ∈ FKCols ∧ ⟨i,t⟩ ∈ t(EN) ∧ ⟨i,c⟩ ∈ *Cols*: j ∈ {1, …, |{⟨i,c⟩ | ⟨i,c⟩ ∈ *Cols*}|}
p22 EN → t	/*c(EN)==1|2|3*/	qs(EN) = ⟨⟨⟨⟩, ⟨⟨⟨i,t⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩: ⟨i,t⟩ ∈ t(EN)
p23 EN → t r	/*c(EN)==1|2|3*/	qs(EN) = ⟨⟨⟨⟩, ⟨⟨⟨i,t⟩,⟨⟩⟩⟩, ⟨z⟩, ⟨⟩⟩⟩: z[j] = ⟨⟨i,t⟩,c,r⟩: ⟨i,t⟩∈ t(EN)∧⟨i,c⟩ ∈*Cols*: j ∈ {1, …, |{⟨i,c⟩ | ⟨i,c⟩ ∈ *Cols*}|}
p24 EN → c	/*c(EN) == 2|3*/	(qs(EN) = ⟨⟨s, ⟨⟩, ⟨⟩, ⟨⟩⟩⟩: (s[j] = ⟨⟨i,t⟩,c⟩ : ⟨i,t⟩ ∈ t(EN) ∧ ⟨i,c⟩ ∈ *Cols*) **jos c(EN) == 2** (qs(EN) = ⟨⟨⟨⟩, ⟨⟩, ⟨z⟩, ⟨⟩⟩⟩: z[j] = ⟨⟨i,t⟩,c,r⟩: ⟨i,c⟩ ∈ *Cols* ∧ ⟨i,t⟩ ∈ t(EN) ∧ r = "IS NOT NULL") **muutoin /* c(EN) == 3 */** missä j ∈ {1, …, |{⟨i,c⟩ | ⟨i,c⟩ ∈ *Cols*}|}
p25 EN → c r	/*c(EN) == 2|3*/	(qs(EN) = ⟨⟨s, ⟨⟩, ⟨z⟩, ⟨⟩⟩⟩: (s[j] = ⟨⟨i,t⟩,c⟩ ∧ z[j] = ⟨⟨i,t⟩,c,r⟩: ⟨i,t⟩ ∈ t(EN) ∧ ⟨i,c⟩ ∈ *Cols*) **jos c(EN) == 2** (qs(EN) = ⟨⟨⟨⟩, ⟨⟩, ⟨z⟩, ⟨⟩⟩⟩: z[j] = ⟨⟨i,t⟩,c,r⟩: ⟨i,c⟩ ∈ *Cols* ∧ ⟨i,t⟩ ∈ t(EN)) **muutoin /* c(EN) == 3 */** missä j ∈ {1, …, |{⟨i,c⟩ | ⟨i,c⟩ ∈ *Cols*}|}

Produktiot ja yleistetyn kyselyn monikkoesityksen jäsennys

Yleistetyn kyselyn monikkoesityksen jäsennys alkaa produktiosta p1. Produktiot p2-p5 vastaavat monikkoesityksen korkeinta hierarkiatasoa seuraavasti: p2:ssa H-monikossa esiintyy vain yksi L-monikko (SELECT-WHERE -osa), p3:ssa H-monikossa esiintyy kahdesta useaan L-monikkoa (ainakin yksi FROM-WHERE -osa ja yksi SELECT-WHERE -osa), p4:ssä Hu-monikossa esiintyy yksi L-monikko (FROM-WHERE -osa) ja p5:ssä Hu-monikossa esiintyy kahdesta useaan L-monikkoa (FROM-WHERE -osaa).

Produktiot p6 ja p7 vastaavat monikkoesityksen tasoa 4 (ks. kuva 25 sivu 47). Produktiossa p6 esiintyy sekä W- että I-monikko ja p7:ssä esiintyy W-monikko, mutta I-monikko on tyhjä. Produktiot p8-p11 vastaavat monikkoesityksen tasoa 3 seuraavasti: p8:ssa W-monikossa esiintyy vain yksi Ps-joukko (W-monikkoon liittyy vain yksi polku), p9:ssä W-monikossa esiintyy kahdesta useaan Ps-joukkoa (W-monikkoon liittyy kaksi tai useampia polkuja), p10:ssä I-monikossa esiintyy vain yksi Ps-joukko ja p11:ssä I-monikossa esiintyy kahdesta useaan Ps-joukkoa AND- tai OR-operaattorilla yhdistettynä.

Produktiot p12-p13 vastaavat monikkoesityksen tasoa 2. Produktiossa p12 Ps-joukko sisältää yhden P-monikon. Tämä tarkoittaa, että Ps-joukon kuvaaman polun solmujen nimille ei ole vaihtoehtoista määritelmää. Produktiossa p13 Ps-joukko sisältää kaksi tai useampia P-monikoita. Tällöin polun solmujen nimille on vaihtoehtoisia määritelmiä. Produktiot p14-p19 vastaavat monikkoesityksen tasoa 1 eli niissä käsitellään polkuja tai polun osia. Produktiossa p14 P-monikon ensimmäisenä alkiona esiintyy juurisolmuun viittaava symboli. Produktiossa p15 P-monikko ei ala tällä symbolilla. Produktiot p16-p19 kuvaavat polkua tai sen osaa seuraavasti: p16:ssa esiintyy vain yksi solmu, p17:ssä esiintyy viimeisenä solmuna lehtisolmuun viittaava symboli, p18:ssa esiintyy kaksi tai useampia solmuja ja p19:ssä esiintyy kaksi tai useampia solmuja niin, että monikkoesityksen ensimmäisen ja kolmannen solmun välissä on jälkeläiselementtiin viittaava symboli.

Produktiot p20-p26 vastaavat polun yksittäisiä solmuja seuraavasti: p20:ssä solmu viittaa relaatiotietokannan vierasavaimeen, p22:ssa solmu viittaa tauluun, p24 solmu viittaa relaatiotietokannan sarakkeeseen ja p21:ssä, p23:ssa sekä p25:ssä vierasavaimeen, tauluun ja sarakkeeseen on liitetty ehtoja. Seuraavaksi tarkastellaan attribuutteja, joiden avulla kyselyn yleistetystä monikkoesityksestä muodostetaan yhdestä useaan SQL-kyselyä.

Attribuutit

Attribuuttikieliopin AG_{TTS} attribuutit joukossa A_{TTS} muuntavat yleistetyn kyselyn monikkoesityksen SQL-kyselyiksi. Attribuuteilla on seuraava yhteys SQL-kyselyyn ja sen muodostamiseen:

- c on peritty attribuutti, joka sisältää tiedon, voiko monikkoesityksen polussa esiintyä viimeisenä solmuna sarake ja liittyvätkö polut include-osaan. Luvussa *6.1. Staattiset säännöt* on kerrottu

tarkemmin sarakkeen mahdollisesta esiintymisestä polussa. Arvo 1 tarkoittaa, että viimeinen solmu voi olla vain taulu (t) tai vierasavain (fk), arvo 2 tarkoittaa, että viimeinen solmu voi olla myös sarake (c) ja arvo 3 tarkoittaa, että viimeinen solmu voi olla myös sarake ja polut liittyvät include-osaan. Tämä attribuutti liittyy nonterminaaleihin L, Hu, W, I, Ps, P, Pc ja EN.

- t on peritty attribuutti, joka sisältää tiedon, mihin tauluihin polku tai polun osa voi viitata. Polun alkaessa kauttaviivalla, polun ensimmäinen solmu voi viitata vain juuritauluihin (*RTables*). Polun päättyessä kenoviivaan polun viimeinen solmu voi viitata vain lehtitauluihin (*LFTables*). Juuri- ja lehtitauluja on käsitelty luvussa 4.2. ja kuvattu luvussa 7.3.1. Tämä attribuutti liittyy nonterminaaleihin Pc ja EN.
- qs on synteettinen attribuutti, joka sisältää SQL-kyselyn monikkomuodossa, jossa monikko sisältää yhdestä useaan ⟨x1, x2, x3, x4⟩-monikkoa. Attribuutti qs vastaa luvussa 6.3. esitettyä varsinaista SQL-kyselyä (viimeistään jäsennyksen loppuvaiheessa) ja ⟨x1, x2, x3, x4⟩-monikot tämän kyselyn FROM-osan alikyselyitä. Tällöin qs-attribuutin arvona oleva monikko sisältää useampia alkioita siinä tapauksessa, että alkuperäisen kyselyn SELECT- tai WHERE-osassa on esitetty useampia polkuja. Tarkemmin sanottuna qs-attribuutin arvona olevan monikon liittyessä alkuperäisen kyselyn SELECT-osaan qs-attribuutin arvona olevan monikon jokainen alkio liittyy yhteen SELECT-osan polkuun. WHERE-osan kohdalla qs-attribuutin arvona oleva monikko sisältää lisäksi AND- tai OR-operaattoreita. Arvot qs-attribuutin arvona olevaan monikkoon muodostuvat yleistetyn kyselyn monikkoesityksen sisältämistä poluista, polkujen solmuista ja käytetystä relaatiotietokannasta. Tarkemmin
    - x1 on monikko, joka koostuu SQL-kyselyn SELECT-osan taulu-sarake -monikoista. Taulu-sarake -monikko kuvataan ⟨⟨i,t⟩,c⟩-monikkona, missä i on taulun numero, t on taulun nimi tai alias ja c on sarakkeen nimi.
    - x2 on monikko, joka koostuu SQL-kyselyn FROM-osan tauluista. Monikko x2 koostuu alkioista ⟨⟨i,t⟩,⟨⟩⟩ ja/tai ⟨⟨i,t1⟩,⟨j,t2⟩⟩, missä i ja j ovat taulun numeroita ja t, t1 ja t2 ovat taulun nimiä. Alkio ⟨⟨i,t⟩,⟨⟩⟩ tarkoittaa yhtä taulua, johon ei ole vielä liitetty toista taulua. Alkio ⟨⟨i,t1⟩,⟨j,t2⟩⟩ tarkoittaa kahta taulua, jotka on liitetty toisiinsa suoraan. Esimerkiksi XIL-kyselystä `SELECT nimi FROM valtio/kaupunki` muodostuu x2-monikko ⟨⟨⟨6,valtio⟩,⟨7,kaupunki⟩⟩⟩. On kuitenkin syytä huomauttaa, että tätä monikkoa ei olisi syntynyt, jos tietokannan suhteet eivät olisi sitä mahdollistaneet. Kyselystä `SELECT nimi FROM valtio//kaupunki` muodostuu aluksi x2-monikko ⟨⟨⟨6,valtio⟩,⟨⟩⟩, ⟨⟨7,kaupunki⟩,⟨⟩⟩⟩. Tässä ensimmäinen x2-monikko, ⟨⟨⟨6,valtio⟩,⟨7,kaupunki⟩⟩⟩, viittaa siihen, että valtio ja kaupunki liitetään toisiinsa suoraan käyttämättä muita tauluja. Jälkimmäinen x2-monikko, ⟨⟨⟨6,valtio⟩,⟨⟩⟩,

⟨⟨7,kaupunki⟩,⟨⟩⟩⟩, taas viittaa siihen, että valtio ja kaupunki saavat liittyä toisiinsa transitiivisesti, muiden taulujen välityksellä.

- x3 on monikko, joka koostuu SQL-kyselyn WHERE-osan taulu-sarake-ehto -monikoista koostuvista monikoista. Taulu-sarake-ehto -monikko kuvataan ⟨⟨i,t⟩,c, r⟩-monikkona, missä i on taulun numero, t on taulun nimi tai alias, c on sarakkeen nimi ja r on sarakkeeseen liittyvä ehto. Monikko x3 tai sen alkiot eivät sisällä AND- tai OR-operaattoreita. Monikon x3 alkiot yhdistetään kuitenkin implisiittisesti AND-operaattorilla ja näiden alkioiden alkiot OR-operaattorilla.
- x4 on monikko, joka koostuu SQL-kyselyn WHERE-osan alikyselyistä. Alikyselyt muodostuvat XIL-kyselyn WHERE-osasta ja siihen liittyvästä SELECT- tai FROM-osasta. Monikko x4 voi sisältää yhdestä useaan monikkoa, jotka koostuvat yhdestä tai useammasta ⟨x5, x6, x7⟩-monikosta ja niitä yhdistävistä AND- tai OR-operaattoreista. Tarkemmin, monikko x4 sisältää yhtä monta alkiota, kuin WHERE-osasta on mahdollista generoida hierarkioita (kyseisellä jäsennyshetkellä). Monikon x4 alkioiden välissä on implisiittisesti OR-operaattori. Monikon x4 sisältämä alkio taas sisältää yhtä monta ⟨x5, x6, x7⟩-monikkoa kuin alkuperäisen kyselyn WHERE-osassa on polkuja ja näiden monikoiden välissä on AND- tai OR-operaattori. XIL-kyselyn WHERE-osasta muodostetaan alikysely, koska XIL-kyselyn WHERE-osan ehdossa saatetaan viitata hierarkiassa eri haaraan kuin varsinainen palautettava tieto viittaa (ks. kuva 28 sivu 50). Joukko x5 koostuu monikoista ⟨⟨i,t⟩,c⟩, missä i on taulun numero, t on taulun nimi tai alias ja c on taulun pääavainsarakkeen nimi. Joukon x5 taulu on kyseiseen WHERE-osaan liittyvän FROM-osan sisältämän polun viimeisen solmun viittaama taulu. Esimerkiksi kuvan 33 kuvaamasta kyselystä on muodostunut joukko x5 {⟨⟨6,valtio⟩,nimi⟩}, missä *valtio* viittaa FROM-osan viimeisen solmun viittaamaan *valtio*-tauluun ja *nimi* viittaa taulun pääavainsarakkeeseen. Kun pääavainsarakkeita on useita, muodostuu joukkoon x5 myös useampia alkioita. Monikko x6 koostuu alkioista ⟨⟨i,t⟩,⟨⟩⟩ ja/tai ⟨⟨i,t1⟩,⟨j,t2⟩⟩ kuten monikko x2 ja monikko x7 koostuu monikoista ⟨⟨i,t⟩,c, r⟩ koostuvista monikoista kuten monikko x3.

Tämä attribuutti liittyy nonterminaaleihin P, Pc ja EN.

- qss on synteettinen attribuutti, jonka arvona on qs-attribuutin arvona olevista monikoista koostuva monikko. Tässä monikossa on yhtä monta alkiota, kuin alkuperäisestä kyselystä on mahdollista generoida hierarkioita (kyseisellä jäsennyshetkellä). Tämä attribuutti liittyy nonterminaaleihin H, Hu, L, W, I ja Ps.
- sq on synteettinen attribuutti, joka sisältää qs-attribuuttien arvona olevista monikoista määritellyn SQL-kyselyn tai -kyselyt. Tämä attribuutti liittyy nonterminaaliin S.

Seuraavaksi tarkastellaan näiden käyttöä semanttisten sääntöjen yhteydessä.

Semanttiset säännöt

Produktioihin (P_{TTS}) liittyvät semanttiset säännöt (R_{TTS}) on esitetty taulukon 2 keskimmäisessä ja oikeanpuoleisessa sarakkeessa. Perittyjä attribuutteja sovelletaan jäsennyksen aikana ja synteettisiä attribuutteja sovelletaan noudattaen jäsennykselle päinvastaista järjestystä. Perityt attribuutit voivat käyttää kyseisessä produktiossa vasemmanpuoleiseen nonterminaaliin liittyvän perityn attribuutin arvoja. Synteettiset attribuutit voivat käyttää kyseisessä produktiossa vasemmanpuoleiseen nonterminaaliin liittyvän perityn attribuutin arvoja ja kyseisessä produktiossa oikeanpuoleisiin nonterminaaleihin liittyvien synteettisten attribuuttien arvoja.

Synteettisen attribuutin qss yhteydessä käytetään funktioita *hi(x), hi(x,y), re(x,y), we(x,y)* ja synteettisen attribuutin sq yhteydessä käytetään funktiota *string(x).*

- *hi(x)* palauttaa hierarkiaa kuvaavan monikon. Hierarkia muodostetaan monikosta x täydentämällä monikossa määrätty hierarkia vastaamaan tietokannan hierarkiaa. Funktio saa argumenttina qs-attribuutin arvona olevan monikon alkion toisen alkion (SQL-kyselyn FROM-osa). Argumentti on siis monikko, joka koostuu monikoista ⟨⟨i,t⟩,⟨⟩⟩ ja/tai ⟨⟨i,t1⟩,⟨j,t2⟩⟩. Esimerkiksi *hi*(⟨⟨⟨6,valtio⟩,⟨⟩⟩⟩) palauttaa monikon ⟨⟨⟨6,valtio⟩,⟨⟩⟩⟩, koska monikko sisältää jo itsessään yhden solmun hierarkian. Toinen esimerkki *hi*(⟨⟨⟨6,valtio⟩,⟨⟩⟩, ⟨⟨4,järvi⟩,⟨⟩⟩⟩) palauttaa esimerkkitietokantaan kohdistettuna monikon ⟨⟨⟨6,valtio⟩,⟨5,sijaitsee⟩⟩, ⟨⟨5, sijaitsee⟩,⟨4,järvi⟩⟩⟩, koska tämä on lyhyin polku solmujen *valtio* ja *järvi* välillä.
- *hi(x,y)* palauttaa hierarkiaa kuvaavan monikon. Hierarkia muodostetaan monikoista x ja y täydentämällä ja yhdistämällä monikoissa määrätty hierarkia vastaamaan tietokannan hierarkiaa. Funktio saa kumpanakin argumenttina qs-attribuutin arvona olevan monikon alkion toisen alkion kuten edellä on kuvattu (SQL-kyselyn FROM-osa). Argumentit *x* ja *y* yhdistetään $x \circ y = x'$ ja kutsutaan *hi*-funktiota tällä argumentilla (*hi(x')*). Esimerkiksi funktiokutsussa *hi*(⟨⟨⟨6,valtio⟩,⟨⟩⟩⟩, ⟨⟨⟨7,kaupunki⟩,⟨⟩⟩⟩) argumenteista muodostetaan yksi argumentti ⟨⟨⟨6,valtio⟩,⟨⟩⟩, ⟨⟨7,kaupunki⟩,⟨⟩⟩⟩ ja välitetään tämä argumenttina funktiolla *hi(x)*. Esimerkkitietokantaan kohdistettuna funktio palauttaa monikon ⟨⟨⟨6,valtio⟩,⟨7,kaupunki⟩⟩⟩, koska tämä on lyhyin polku solmujen *valtio* ja *kaupunki* välillä.
- *re(x,y)* palauttaa monikon, joka koostuu niistä monikon y alkioista, jotka sijaitsevat monikon x määräämässä hierarkiassa. Funktion ensimmäinen argumentti vastaa qs-attribuutin arvona olevan monikon alkion toista alkiota (SQL-kyselyn FROM-osa) ja toinen argumentti vastaan qs-attribuutin arvona olevan monikon alkion ensimmäistä alkiota (SQL-kyselyn SELECT-osa). Esimerkiksi *re(⟨⟨⟨6,valtio⟩, ⟨1,virtaa⟩⟩, ⟨⟨1,virtaa⟩, ⟨2,joki⟩⟩⟩, ⟨⟨⟨2,joki⟩, nimi⟩, ⟨⟨4,järvi⟩, nimi⟩,⟨⟨6, valtio⟩, nimi⟩,⟨⟨7, kaupunki⟩ nimi⟩⟩)* palauttaa esimerkkitietokantaan kohdistettuna

monikon *⟨⟨⟨2,joki⟩,nimi⟩, ⟨⟨4, järvi⟩, nimi⟩⟩*, koska nämä sijaitsevat x-monikon määräämässä hierarkiassa (ks. esim. kuvan 22 sivu 40 vasen haara).

- *we(x,y)*-funktion ensimmäinen argumentti vastaa qs-attribuutin arvona olevan monikon alkion toista alkiota (SQL-kyselyn FROM-osa) ja toinen argumentti vastaan qs-attribuutin arvona olevan monikon alkion kolmatta alkiota (SQL-kyselyn WHERE-osa). Argumentti y on siis monikko, joka koostuu monikoista ja nämä monikot koostuvat ⟨⟨i,t⟩,c⟩-monikoista. Funktio palauttaa ne ⟨⟨i,t⟩,c⟩-monikot, jotka sijaitsevat monikon x määräämässä hierarkiassa. Palautettavat ⟨⟨i,t⟩,c⟩-monikot ryhmitellään palautettavan monikon alkioihin, kuten ne on y-monikossa ryhmitelty.
- *string(x)* palauttaa joukossa x kuvatun SQL-kyselyn tai SQL-kyselyt. Funktio saa argumenttina monikon, joka koostuu qs-attribuuttien arvona olevista monikoista. Kuten jo edellä on kuvattu, qs-attribuutti sisältää SQL-kyselyn monikkomuodossa, joka koostuu ⟨x1, x2, x3, x4⟩-monikkoja sisältävistä monikoista. Jokaisesta qs-attribuutista muodostetaan oma kyselynsä ja näitä kyselyitä ei yhdistetä toisiinsa. Seuraavaksi kuvataan, kuinka qs-attribuutin sisältämästä monikosta muodostetaan kysely.

  Kun kyselyn muodostuksessa päästään tähän funktioon, qs-attribuutti sisältää yhtä monta alkiota, ⟨x1, x2, x3, x4⟩-monikkoa, kuin alkuperäisen XIL-kyselyn SELECT-osassa on polkuja. Jotta polut tulisi ryhmiteltyä lopullisessa kyselyssä samalla tavalla kuin XILlissä ja tieto polkujen yhteydestä toisiinsa tietokannassa ei katoaisi, on polut yhdistettävä samaan SQL-kyselyyn. Monikoiden ⟨x1, x2, x3, x4⟩ kuvaamat hierarkiat saattavat kuitenkin erota toisistaan, joten niistä ei voi suoraan muodostaa yhtä kyselyä. Vaikka hierarkia saattaakin erota, on saman qs-attibuutin sisällä olevien alkioiden hierarkioiden juuritaulun oltava aina sama. Juuritaulu on sama kolmessa tilanteessa: alkuperäisessä XIL-kyselyssä on annettu FROM-osa, alkuperäisessä kyselyssä ei ole annettu FROM-osaa, mutta SELECT-osassa on annettu vain yksi polku tai alkuperäisessä kyselyssä ei ole annettu FROM-osaa ja SELECT-osassa on useampia polkuja pilkulla erotettuna, mutta jokainen näistä poluista alkaa samalla solmulla. Tilannetta, jossa näin ei ole käsitellään alempana erikoistapausten yhteydessä. Koska hierarkiat saattavat erota toisistaan, jokaisesta ⟨x1, x2, x3, x4⟩-monikosta muodostetaan oma alikysely varsinaisen SQL-kyselyn FROM-osaan ja alikyselyiden tulostauluista liitetään alkuperäisessä kyselyssä kuvatut tiedot varsinaisen SQL-kyselyn tulokseen seuraavasti:

```
SELECT t1.juuritaulunpääavain, t1.a, t1.b, t2.c, t2.d, …, tn.w
FROM (alikysely) AS t1 INNER JOIN (alikysely) AS t2 ON
t1.juuritaulunpääavain = t2.juuritaulunpääavain AND … AND
t1.viimeisenyhteisentaulunpääavain = t2.
viimeisenyhteisentaulunpääavain INNER JOIN … INNER JOIN (alikysely)
```

```
AS tn ON tn-1.juuritaulunpääavain = tn.juuritaulunpääavain AND … AND
tn-1. viimeisenyhteisentaulunpääavain = tn.
viimeisenyhteisentaulunpääavain
WHERE (t1.a IS NOT NULL OR t1.b IS NOT NULL) AND (t2.c IS NOT NULL OR
t2.d IS NOT NULL) AND … AND (tn.w IS NOT NULL)
ORDER BY t1.juuritaulunpääavain
```

Tässä `juuritaulunpääavain` viittaa hierarkian ensimmäiseen tauluun ja *`viimeisenyhteisentaulunpääavain`* viittaa alkuperäisen XIL-kyselyn FROM-osasta muodostuneeseen viimeisen yhteisen taulun pääavaimeen. Alikyselyiden liitoksessa suoritetaan vertailu kaikkien alkuperäisen kyselyn FROM-osasta saatujen taulujen pääavainten välillä, jotta taulut liitetään kyselyn osoittamalla tavalla yhteen. Varsinaiseen SQL-kyselyyn otetaan kuitenkin mukaan alkuperäisessä kyselyssä osoitettujen sarakkeiden lisäksi vain hierarkian juuritaulun pääavain, koska tämä mahdollistaa kyselyn tulosten ryhmittelyn.

Varsinaisen kyselyn FROM-osan alikyselyt muodostetaan ⟨x1, x2, x3, x4⟩-monikoista seuraavasti: monikko x1 koostuu ⟨⟨i,t⟩,c⟩-monikoista. Sarakkeet luetellaan SELECT-osassa *taulu.sarake*-notaatiolla (t.c) ja pilkulla erotettuina. Lisäksi SELECT-osan ensimmäiseksi sarakkeeksi lisätään yhteisen hierarkian muodostavien taulujen pääavaimet, jotta alikyselyiden tulostaulut on mahdollista liittää yhteen. Monikko x2 koostuu monikoista ⟨⟨i,t⟩,⟨⟩⟩ ja/tai ⟨⟨i,t1⟩,⟨j,t2⟩⟩. Taulut luetellaan FROM-osassa x2-monikon järjestyksessä siten, että niiden välissä on vasen ulkoliitos. Vasempaan ulkoliitokseen liittyvä ON-ehto määritellään *Edges* ja *FKCols*-joukkojen avulla. Monikko x3 koostuu ⟨⟨i,t⟩,c,r⟩-monikoista koostuvista monikoista. Sarakkeet ja sarakkeisiin liittyvät ehdot luetellaan WHERE-osassa *taulu.sarake ehto* -notaatiolla ja AND- tai OR-operaattorilla yhdistettyinä. Joukko x4 koostuu ⟨x5, x6, x7⟩-monikoita sisältävistä monikoista. Näistä muodostetaan alikyselyt WHERE-osaan, missä x5 muodostetaan kuten x1, x6 muodostetaan kuten x2 ja x7 muodostetaan kuten x3. Jos FROM-osasta (x2 tai x6) ei löydy kaikkia SELECT- ja WHERE-osan sarakkeiden tauluja, lisätään FROM-osaan tarvittavat taulut.

Kyselyn muodostukseen liittyy myös erikoistapauksia, joissa qs-monikoita ja niiden alkioiden sisältöä joudutaan vielä muokkaamaan. Yhtenä erikoistapauksena x2-monikko on tyhjä (XIL-kyselyssä ei ole esitetty taulun nimiä), jolloin myös x4-monikko on tyhjä (XIL-kyselyssä ei ole esitetty hierarkiaa, johon ehdon voisi liittää). Tällöin SELECT-osan osoittamista/vaatimista tauluista muodostetaan hierarkia kaikilla mahdollisilla tavoilla ja jokaisesta tavasta muodostetaan oma qs-monikko (oma SQL-kysely). Käytännössä tämä tarkoittaa sitä, että hierarkioita muodostetaan yhtä monta kuin SELECT-osassa on tauluja. Tällöin muodostetaan hierarkia käyttäen jokaista näistä tauluista vuorollaan juurena. Toisena

erikoistapauksena qs-monikon sisältämien alkioiden x2-monikot eivät sisällä samaa taulua ensimmäisenä alkiona. Tällainen tilanne syntyy vain, jos alkuperäisessä kyselyssä ei ole annettu FROM-osaa ja SELECT-osassa on lueteltu useampia polkuja pilkulla erotettuna. Myös tällöin pyritään muodostamaan kaikki mahdolliset hierarkiat käyttäen jokaista qs-monikon alkioiden mahdollistamaa taulua juuritauluna ja jokaisesta eri hierarkiasta muodostetaan uusi qs-monikko. Sekä ensimmäisessä että toisessa erikoistapauksessa pyritään siis löytämään kaikki mahdolliset tavat, kuinka alkuperäisen kyselyn SELECT-osan viittaamat sarakkeet voivat käytetyssä tietokannassa liittyä toisiinsa. Kolmannessa erikoistapauksessa x1-monikko on tyhjä. Tällöin SELECT-osaan lisätään FROM-osan viimeisen taulun sarakkeet.

Perityt attribuutit

Produktiot p1-p13 kuvaavat yleistetyn kyselyn monikkoesityksen tasoja 5-2. Näissä produktioissa oikeanpuoleiseen nonterminaaliin liitetään peritty attribuutti c kuvaamaan sitä, saako nonterminaalin kuvaamien polkujen viimeinen elementti viitata sarakkeeseen ja onko polku include-osasta. Säännössä p2 attribuutti alustetaan arvolla 2, koska L-monikon polkujen viimeinen solmu saa viitata sarakkeeseen. Säännössä p3 nonterminaaliin Hu liittyvä attribuutti alustetaan arvolla 1 ja nonterminaaliin L liittyvä attribuutti alustetaan arvolla 2. Tämä siksi, että vain hierarkian viimeinen L-monikko voi sisältää viittauksia sarakkeisiin (vain viimeinen L-monikko liittyy SELECT-WHERE -osaan). Säännössä p4 attribuutti alustetaan nonterminaaliin Hu liittyvän attribuutin arvolla, mikä on 1. Samoin menetellään p5:ssä. Säännössä p6 nonterminaaliin W liittyvä attribuutti alustetaan arvolla 1 tai 2 riippuen siitä, mistä produktiosta tähän on tultu ja nonterminaaliin I liittyvä attribuutti alustetaan arvolla 3. Säännöissä p7-p13 attribuutit alustetaan produktioiden vasemmanpuoleiseen nonterminaaliin liittyvän attribuutin arvolla.

Produktiot p14-p19 kuvaavat monikkoesityksen tasoa 1. Näissä produktion oikeanpuoleiseen nonterminaaliin liitetään perityn attribuutin c lisäksi peritty attribuutti t. Peritty attribuutti t kuvaa taulujen joukkoa, johon polun tai polun osan ensimmäinen elementti voi viitata. Säännöissä p14-p17 c-attribuutit alustetaan produktioiden vasemmanpuoleiseen nonterminaaliin liittyvän c-attribuutin arvolla ja t-attribuutit muodostetaan seuraavasti: p14:ssä attribuutti alustetaan *RTables*-joukolla, p15:ssä attribuutti alustetaan *Tables*-joukolla, p16:ssa attribuuttiin asetetaan nonterminaaliin Pc liittyvän t-attribuutin arvo ja p17:ssä attribuuttiin asetetaan nonterminaaliin Pc liittyvän t-attribuutin ja *LFTables*-joukon leikkaus. Säännössä p18 nonterminaaliin EN (polun tai polun osan ensimmäinen alkio) liittyvään t-attribuuttiin asetetaan nonterminaalin Pc1 liittyvän t-attribuutin arvo ja nonterminaaliin Pc2 liittyvään t-attribuuttiin asetetaan *Tables*-joukko. Tämä siksi, että t-attribuutissa määritellyt taulut liittyvät aina vain polun ensimmäiseen solmuun. Edelleen nonterminaaliin EN liittyvään c-attribuuttiin asetetaan arvo 1, koska vain polun viimeinen elementti voi viitata sarakkeeseen, ja nonterminaaliin Pc2

liittyvään c-attribuuttiin asetetaan nonterminaalin Pc1 c-attribuutin arvo. Attribuutit käsitellään säännössä p19 kuten säännössä p18.

Synteettiset attribuutit

Produktiot p20-p25 ovat lehtiproduktioita. Kuten jo aiemmin on todettu, tämä tarkoittaa, että produktion oikealla puolella ei ole nonterminaaleja ja jäsennys päätetään. Näin ollen säännöissä siirrytään synteettisiin attribuutteihin. Näissä säännöissä polkujen solmut asetetaan solmua vastaavalle paikalle qs-monikossa. Solmu voi viitata tauluun (myös vierasavaimen kautta) tai sarakkeeseen.

Säännössä p24 käsitellään saraketta. Attribuutin c arvon ollessa 2 qs-attribuutti alustetaan monikolla ⟨⟨s, ⟨⟩, ⟨⟩, ⟨⟩⟩⟩: s[j] = ⟨⟨i,t⟩,c⟩, missä i on taulun numero, t on taulun nimi ja c on taulun i sarake, jos ⟨i,t⟩-pari löytyy jäsennyksessä muodostuneesta t-attribuutista ja ⟨i,c⟩-pari löytyy Cols-joukosta. Toisin sanoen säännössä sarakkeeseen liitetään tieto taulusta, johon sarake liittyy. Monikkoon s liitetään yhtä monta alkiota kuin tietolähteessä on c-nimisiä sarakkeita. Säännössä p24 c-attribuutin arvon ollessa 3 qs-attribuutti alustetaan monikolla ⟨⟨⟨⟩, ⟨⟩, ⟨z⟩, ⟨⟩⟩⟩: z[j] = ⟨⟨i,t⟩,c,r⟩, missä i, t sekä c määritellään kuten edellä ja r:n arvo on ’IS NOT NULL’. Säännössä p25 sarakkeeseen liittyy ehto r. Attribuutin c arvon ollessa 2 qs-attribuutti alustetaan monikolla ⟨⟨s, ⟨⟩, ⟨z⟩, ⟨⟩⟩⟩: s[j] = ⟨⟨i,t⟩,c⟩ ∧ z[j] = ⟨⟨i,t⟩,c,r⟩, missä i, t, c määritellään kuten edellä. Attribuutin c arvon ollessa 2 on kyseiseen sarakkeeseen liittyvä solmu peräisin XIL-kyselyn SELECT-osasta. Tästä syystä sarake liitetään qs-attribuutissa sekä ensimmäiseen että kolmanteen alkioon eli sarake lisätään myös SQL-kyselyssä sekä SELECT- että WHERE-osaan. Attribuutin c arvon ollessa 3 qs-attribuutti alustetaan monikolla ⟨⟨⟨⟩, ⟨⟩, ⟨z⟩, ⟨⟩⟩⟩: z[j] = ⟨⟨i,t⟩,c,r⟩. Tämä siksi, että sarake on nyt peräisin XIL-kyselyn WHERE-osasta ja lisätään siksi vain SQL-kyselyn WHERE-osaan.

Säännössä p22 käsitellään taulua ja qs-attribuutti alustetaan monikolla ⟨⟨⟨⟩, ⟨⟨⟨i,t⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩, missä i ja t määritellään kuten edellä. Säännössä p23 tauluun liittyy ehto ja qs-attribuutti alustetaan monikolla ⟨⟨⟨⟩, ⟨⟨⟨i,t⟩,⟨⟩⟩⟩, ⟨z⟩, ⟨⟩⟩⟩: z[j] = ⟨⟨i,t⟩,c,r⟩, missä i, t, c ja r määritellään kuten edellä. Koska ehto ei SQL:ssä voi liittyä tauluun, on ehto liitetty taulun sarakkeisiin.

Säännössä p20 qs-attribuutti alustetaan monikolla ⟨⟨⟨⟩,⟨⟨⟨i,fk⟩,⟨⟩⟩⟩,⟨⟩,⟨⟩⟩⟩, missä fk on vierasavaimen nimi ja i on sen taulun numero, johon vierasavain viittaa, jos ⟨i,j,fk⟩-monikko kuuluu *FKCols*-joukkoon ja ⟨i,t⟩-pari kuuluu attribuuttiin t(EN). Toisin sanoen käytetystä relaatiotietokannasta tulee löytyä fk-niminen vierasavain, ja taulun, johon tämä vierasavain viittaa, tulee kuulua jäsennyksessä määräytyneeseen t-attribuuttiin. Säännössä p21 qs-attribuutti alustetaan monikolla ⟨⟨⟨⟩,⟨⟨⟨i,fk⟩,⟨⟩⟩⟩,⟨z⟩,⟨⟩⟩⟩: z[j] = ⟨⟨i,t⟩,c,r⟩ ja sääntö muodostetaan kuten p20:ssä ja p23:ssa.

Säännöissä p14-p19 liitetään jäsennyksessä poluista erotetut solmut qs-attribuuttiin oikeille paikoilleen niin, että solmujen järjestys säilyy. Säännössä p19 asetetaan qs-attribuuttiin monikko ⟨⟨z1, z2, z3, ⟨⟩⟩⟩, missä z1:een asetetaan nonterminaalin Pc2 qs-attribuutin arvona olevan monikon alkion ensimmäinen alkio. Tähän ei liitetä nonterminaalin EN qs-attribuutin arvona olevan monikon alkion

ensimmäistä alkiota, koska EN ei tässä voi viitata sarakkeeseen, jolloin tämä ensimmäinen alkio on tyhjä. Monikkoon z2 asetetaan EN:n qs-attribuutin arvona olevan monikon alkion toisen alkion ja Pc2:n qs-attribuutin arvona olevan monikon alkion toisen alkion konkatenaatio. Suoritettaessa konkatenaatio tässä järjestyksessä solmujen (taulujen) järjestys säilyy. Viimeisenä asetetaan z3:een EN:n qs-attribuutin kolmannen alkion ja Pc2:n qs-attribuutin kolmannen alkion konkatenaatio. Sääntö p18 eroaa edellisestä siinä, että nonterminaalin EN viittaama taulu liittyy suoraan nonterminaalin Pc2 ensimmäiseen tauluun tai sarakkeeseen (välillä ei ole //-terminaalisymbolia). Tällöin säännössä tarkastetaan, mahdollistavatko käytetyn tietokannan suhteet tämän. Säännössä asetetaan qs-attribuuttiin monikko ⟨⟨z1, z2, z3, ⟨⟩⟩⟩.

- Pc1:n c-attribuutin arvon ollessa 1 tai 2 ja Pc2:n qs-attribuutin arvona olevan monikon alkion toisen alkion ollessa tyhjä (Pc2 sisältää vain yhden solmun ja tämä solmu viittaa sarakkeeseen) asetetaan z1:een Pc2:n qs-attribuutin arvona olevan monikon alkion ensimmäisestä alkiosta sellainen taulu-sarake -pari, että taulu täsmää EN:n qs-attribuutin arvona olevan monikon alkion toisessa alkiossa esiintyvään tauluun. Tämä taulu asetetaan z2:een, ja jos sarakkeeseen on liittynyt ehto, asetetaan se z3:een.
- Pc1:n c-attribuutin arvon ollessa 1 tai 2, ja kun Pc2:n qs-attribuutin arvona olevan monikon alkion toinen alkio ei ole tyhjä (EN viittaa tauluun ja ainakin Pc2:n ensimmäinen solmu viittaa tauluun), asetetaan z1:een Pc2:n qs-attribuutin arvona olevan monikon alkion ensimmäinen alkio sellaisenaan ja z3:een Pc2:n qs-attribuutin arvona olevan monikon ja EN:n qs-attribuutin arvona olevan monikon alkioiden kolmansien alkioiden konkatenaatio. Säännössä tarkistetaan, että EN:n viittaama taulu, joka esitetään EN:n qs-attribuutin arvona olevan monikon alkion toisessa alkiossa ja Pc2:n ensimmäisen solmun viittaama taulu, joka esitetään Pc2:n qs-attribuutin arvona olevan monikon alkion toisen alkion ensimmäisessä alkiossa, ovat tietokannassa välittömässä suhteessa toisiinsa (Edges). Tällöin z2:een asetetaan näiden taulujen parin ja Pc2:n qs-attribuutin arvona olevan monikon alkion toisen alkion konkatenaatio.
- Pc1:n c-attribuutin arvon ollessa 3 ja Pc2:n qs-attribuutin arvona olevan monikon alkion toisen alkion ollessa tyhjä (tulkitaan kuten edellä) muodostetaan z2 ja z3 kuten c-attribuutin arvon ollessa 1 tai 2, mutta z1 jää tyhjäksi.
- Pc1:n c-attribuutin arvon ollessa 3, ja kun Pc2:n qs-attribuutin arvona olevan monikon alkion toinen alkio ei ole tyhjä, muodostetaan z2 ja z3 kuten c-attribuutin ollessa 1 tai 2, mutta z1 jää tyhjäksi.

Säännöissä p14-p17 qs-attribuuttiin asetetaan produktiossa oikeanpuoleisen qs-attribuutin arvo.

Säännöissä p2-p13 siirrytään qss-attribuutteihin, jotka sisältävät yhdestä useaan qs-attribuuttia. Säännössä p13 vaihtoehtoisista poluista muodostuneet qs-attribuutit asetetaan qss-attribuuttiin ja p12:ssa produktion oikeanpuoleisen nonterminaalin qs-attribuutti asetetaan qss-attribuutin alkioksi.

Säännössä p11 AND- tai OR-operaattorilla liitetyistä poluista muodostuneiden qss-attribuuttien alkiot liitetään yhteen, niin että jokainen qss(Ps)-attribuutin alkio liitetään konkatenaatiolla jokaiseen qss(I2)-attribuutin alkioon (vrt. joukkojen karteesinen tulo). Näiden alkioiden väliin sijoitetaan joko AND- tai OR-operaattori riippuen siitä, kumpi tähän liittyvässä kyselyn yleistetyssä monikkoesityksessä on. Säännössä p9 jokainen qss(Ps)-monikon alkio liitetään jokaiseen qss(W2)-monikon alkioon. Säännöissä p7, p8 ja p10 produktion oikeanpuoleisen nonterminaalin qss-attribuutin arvo asetetaan säännön qss-attribuuttiin.

Säännössä p6 yhdistetään nonterminaalit W ja I siten, että jokaiseen W:n qss-attribuutin alkioon liitetään jokainen I:n qss-attribuutin alkio. Tämä siksi, että I asettaa ehtoja W:lle ja tilanteessa, jossa W:stä muodostuu useita alkioita qss-attribuuttiin (useita SQL-kyselyjä), pitää I:ssä asetetut ehdot liittää näihin kaikkiin (ks. esim. liite 3 kysely (9)).

Säännöissä p4 ja p5 qss-attribuuttiin asetetaan ⟨⟨⟨⟩, z2, z3, z4⟩⟩-alkioista koostuva monikko. Sääntöjen qss-attribuuttien alkioiden ensimmäinen alkio on aina tyhjä, koska näihin sääntöihin ei liity kyselyssä palautettaviin sarakkeisiin viittaavia solmuja. Sen sijaan säännöissä voi esiintyä sarakkeita, joihin liittyy ehto (z3) ja alikyselyitä (z4). Säännössä p5 Hu2:een liittyvä qss-attribuutti koostuu ⟨⟨⟨⟩, x2, x3, x4⟩⟩-alkioista ja L:ään liittyvä qss-attribuutti koostuua ⟨⟨⟨⟩, y2, y3, y4⟩⟩-alkioista. Säännössä Hu2:sta ja L:stä muodostuneet hierarkiat yhdistetään (x2 ja y2) ja sovitetaan tietokannan rakenteeseen. Tällä tarkoitetaan Hu2:sta ja L:stä muodostuneita tauluja. Tuloksena saatu hierarkia asetetaan z2:een. Alkiot x3 ja x4 säilyvät ennallaan, koska ne on jo aiemmin sovitettu x2:n kuvaamaan hierarkiaan. Sen sijaan y3 ja y4 tulee sovittaa z4:n hierarkiaan, koska niiden sijoittuminen hierarkiaan riippuu myös Hu2:sta muodostuneesta hierarkiasta (ks. esim. liite 3 kysely (8)). Säännössä p4 L:ään liittyvä qss-attribuutti koostuu ⟨⟨⟨⟩, x2, x3, x4⟩⟩-alkioista. Alkion x2 kuvaama hierarkia sovitetaan tietokannan rakenteeseen ja sekä x3:n että x4:n alkiot sovitetaan tähän hierarkiaan.

Säännössä p3 qss-attribuuttiin asetetaan monikko, joka koostuu ⟨z1, z2, z3, x4⟩-alkioista koostuvista monikoista. Näihin sääntöihin voi jo liittyä tietokannan sarakkeita. Säännössä p3 Hu:n qss-attribuutti koostuu ⟨⟨⟨⟩, x2, x3, x4⟩⟩-alkioista ja L:n qss-attribuutti koostuu ⟨y1, y2, y3, y4⟩-alkioista koostuvista monikoista. Alkiot z2, z3 ja z4 muodostetaan kuten edellä ja y1:n sisältämät alkiot sovitetaan z2:n kuvaamaan hierarkiaan ja asetetaan z1:een. Säännössä p2 ei qss-attribuuttia muuteta. Tämä johtuu siitä, että tähän sääntöön tultaessa alkuperäisessä kyselyssä ei ole annettu FROM-osaa ja tällöin kyselyä muodostettaessa voi esiintyä yksi tai useampia erikoistapauksia, jotka selvitetään vasta *string*-funktiossa Säännössä p1 muodostetaan SQL-kysely sanallisessa muodossa aiemmin kuvatulla *string*-funktiolla.

7.4. TuplesToSQL-jäsennysesimerkit

```
p1 H = ⟨⟨⟨{⟨nimi⟩}⟩, ⟨⟩⟩⟩
p2 L = ⟨⟨{⟨nimi⟩}⟩, ⟨⟩⟩, c(L) = 2
p7 W = ⟨{⟨nimi⟩}⟩, c(W) = c(L)
p8 Ps = {⟨nimi⟩}, c(Ps) = c(W)
p12 P = ⟨nimi⟩, c(P) = c(Ps)
p15 Pc = ⟨nimi⟩, t(Pc) = Tables, c(Pc) = c(P)
p16 EN = nimi, t(EN) = t(Pc), c(EN) = t(Pc)
p24 qs(EN) = ⟨⟨⟨⟨⟨2,joki⟩,nimi⟩, ⟨⟨4,järvi⟩,nimi⟩, ⟨⟨6,valtio⟩,nimi⟩, ⟨⟨7,kaupunki⟩,nimi⟩⟩,
⟨⟩, ⟨⟩, ⟨⟩⟩⟩
p16 qs(Pc) = qs(EN)
p15 qs(P) = qs(Pc)
p12 qss(Ps) = ⟨qs(P)⟩
p8 qss(W) = qss(Ps)
p7 qss(L) = qss(W)
p2 qss(H) = ⟨⟨⟨⟨⟨2,joki⟩,nimi⟩, ⟨⟨4,järvi⟩,nimi⟩, ⟨⟨6,valtio⟩,nimi⟩, ⟨⟨7,kaupunki⟩,nimi⟩⟩,
⟨⟩, ⟨⟩, ⟨⟩⟩⟩

p1 string(S) =

SELECT t1.jokinimi, t1.järvinimi, t1.valtionimi, t1.kaupunkinimi
FROM (SELECT joki.nimi AS jokinimi, järvi.nimi AS järvinimi, valtio.nimi AS
valtionimi, kaupunki.nimi AS kaupunkinimi
FROM joki LEFT JOIN liittyy ON joki.nimi = liittyy.joki_nimi LEFT JOIN järvi ON
liittyy.järvi_nimi = järvi.nimi LEFT JOIN virtaa ON joki.nimi = virtaa.joki_nimi
LEFT JOIN valtio ON virtaa.valtio_nimi = valtio.nimi LEFT JOIN kaupunki ON
valtio.nimi = kaupunki.valtio_nimi) AS t1
WHERE (t1.jokinimi IS NOT NULL OR t1.järvinimi IS NOT NULL OR t1.valtionimi IS NOT
NULL OR t1.kaupunkinimi IS NOT NULL) ORDER BY t1.jokinimi

SELECT t1.järvinimi, t1.jokinimi, t1.valtionimi, t1.kaupunkinimi
FROM (SELECT järvi.nimi AS järvinimi, joki.nimi AS jokinimi, valtio.nimi AS
valtionimi, kaupunki.nimi AS kaupunkinimi
FROM järvi LEFT JOIN liittyy ON järvi.nimi = liittyy.järvi_nimi LEFT JOIN joki ON
liittyy.joki_nimi = joki.nimi LEFT JOIN sijaitsee ON järvi.nimi =
sijaitsee.järvi_nimi LEFT JOIN valtio ON sijaitsee.valtio_nimi = valtio.nimi LEFT
JOIN kaupunki ON valtio.nimi = kaupunki.valtio_nimi) AS t1
WHERE (t1.järvinimi IS NOT NULL OR t1.jokinimi IS NOT NULL OR t1.valtionimi IS NOT
NULL OR t1.kaupunkinimi IS NOT NULL) ORDER BY t1.järvinimi

jatkuu
```

```
jatkuu

SELECT t1.valtionimi, t1.järvinimi, t1.jokinimi, t1.kaupunkinimi
FROM (SELECT valtio.nimi AS valtionimi, järvi.nimi AS järvinimi, joki.nimi AS
jokinimi, kaupunki.nimi AS kaupunkinimi
FROM valtio LEFT JOIN sijaitsee ON valtio.nimi = sijaitsee.valtio_nimi LEFT JOIN
järvi ON sijaitsee.järvi_nimi = järvi.nimi LEFT JOIN virtaa ON valtio.nimi =
virtaa.valtio_nimi LEFT JOIN joki ON virtaa.joki_nimi = joki.nimi LEFT JOIN kaupunki
ON valtio.nimi = kaupunki.valtio_nimi) AS t1
WHERE (t1.valtionimi IS NOT NULL OR t1.järvinimi IS NOT NULL OR t1.jokinimi IS NOT
NULL OR t1.kaupunkinimi IS NOT NULL) ORDER BY t1.valtionimi

SELECT t1.kaupunkinimi, t1.valtionimi, t1.järvinimi, t1.jokinimi
FROM (SELECT valtio.nimi AS valtionimi, järvi.nimi AS järvinimi, joki.nimi AS
jokinimi, kaupunki.nimi AS kaupunkinimi
FROM kaupunki LEFT JOIN valtio ON kaupunki.valtio_nimi = valtio.nimi LEFT JOIN
sijaitsee ON valtio.nimi = sijaitsee.valtio_nimi LEFT JOIN järvi ON
sijaitsee.järvi_nimi = järvi.nimi LEFT JOIN virtaa ON valtio.nimi =
virtaa.valtio_nimi LEFT JOIN joki ON virtaa.joki_nimi = joki.nimi) AS t1
WHERE (t1.kaupunkinimi IS NOT NULL OR t1.valtionimi IS NOT NULL OR t1.järvinimi IS
NOT NULL OR t1.jokinimi IS NOT NULL) ORDER BY t1.kaupunkinimi
```

Kuva 30. Esimerkki monikkoesityksen kääntämisestä SQL-kyselyksi attribuuttikieliopin avulla. Monikkoesitys on muodostettu XIL-kyselystä SELECT nimi.

Kuvan 30 jäsennys syntyy XIL-kyselyn `SELECT nimi` monikkoesityksestä ⟨⟨⟨{⟨nimi⟩}⟩, ⟨⟩⟩⟩ ja esimerkkinä käytetystä valtiotietokannasta. Kysely sisältää vain SELECT-osan, mikä on mahdollista XIL-kyselykielessä ja voidaan lukea: valitse kaikki nimi-elementit tietolähteestä. Jäsentäminen aloitetaan produktiosta p1 ja tästä siirrytään produktioon p2, koska H-monikko sisältää vain yhden alkion (L-monikon). Attribuutti c saa arvon 2, koska H-monikon viimeisen alkion kuvaamien polkujen viimeinen alkio voi viitata sarakkeeseen. Seuraavaksi siirrytään produktioon p7, koska L-monikon toinen alkio on tyhjä. Produktiosta p7 jatketaan produktioon p8, koska W-monikko sisältää vain yhden alkion (polun). Edelleen jatketaan produktioon p12, koska Ps-joukko sisältää vain yhden alkion (polun solmuille ei ole vaihtoehtoisia esitystapoja). Seuraavaksi siirrytään produktioon p15, koska P-monikon ensimmäinen alkio ei viittaa juureen. Produktiossa p15 t-attribuuttiin asetetaan {⟨1, virtaa⟩, ⟨2, joki⟩, ⟨3, liittyy⟩, ⟨4,järvi⟩, ⟨5,sijaitsee⟩, ⟨6,valtio⟩, ⟨7,kaupunki⟩}, eli kaikki tietokannan taulut, sillä polussa ei esiinny juuritauluihin viittaavaa symbolia. Pc-monikko sisältää vain yhden alkion, joten siirrytään produktioon p16. Jäsennys päättyy lehtiproduktioon p24, jossa siirrytään synteettisiin attribuutteihin. Koska c-attribuutin arvo on 2, asetetaan EN:n qs-attribuuttiin ⟨⟨⟨⟨⟨2,joki⟩,nimi⟩, ⟨⟨4,järvi⟩,nimi⟩,

⟨⟨6,valtio⟩,nimi⟩, ⟨⟨7,kaupunki⟩,nimi⟩⟩, ⟨⟩, ⟨⟩, ⟨⟩⟩⟩-monikko. Säännössä valitaan kaikki tietokannan *nimi*-sarakkeet, koska hierarkia ei ole (vielä) tiedossa. Noustaessa jäsentämisessä ylöspäin, säännöissä p16 ja p15 ei attribuutin arvoja muuteta. Säännössä p12 asetetaan qss-attribuutin alkioksi P:hen liittyvä qs-attribuutti. Myöskään säännöissä p8, p7 ja p2 ei muuteta qss-attribuutin arvoa. Säännössä p1 muodostetaan varsinaiset sanallisessa muodossa olevat SQL-kyselyt. Koska FROM-osaan viittaava monikko on tyhjä, alkuperäisestä kyselystä ei ole muodostunut hierarkiaa. Tällöin pyritään muodostamaan kaikki mahdolliset hierarkiat. Jotta kaikki mahdolliset hierarkiat saadaan muodostettua, tulee jokainen SELECT-osaan viittaavan monikon sisältämistä tauluista valita vuorollaan muodostettavan hierarkian juureksi. Varsinaisia SQL-kyselyitä muodostuu yhtä monta, kuin on eri tauluja. Tämä johtuu siitä, että työssä valitaan aina lyhyin polku ja siinä tapauksessa, että lyhyimpiä polkuja on useita valitaan niistä ensimmäinen. Ensimmäisenä muodostuneessa SQL-kyselyssä on käytetty juurena *joki*-taulua, johon on ensimmäisenä liitetty lyhyimmällä mahdollisessa polulla *järvi*-taulu. Seuraavaksi *joki*-tauluun on liitetty lyhyimmällä mahdollisella polulla *valtio*-taulu ja viimeisenä *kaupunki*-taulu. Kun hierarkia on muodostettu, muodostetaan tästä kysely *string*-funktiossa kuvatulla tavalla. Vastaavasti toimitaan valitsemalla seuraavaksi hierarkian juureksi *järvi*-taulu, tämän jälkeen *valtio*-taulu ja viimeisenä *kaupunki*-taulu.

```
p1 H = ⟨⟨⟨{⟨valtio, nimi⟩}⟩, ⟨⟩⟩⟩
p2 L = ⟨⟨{⟨valtio, nimi⟩}⟩, ⟨⟩⟩, c(L) = 2
p7 W = ⟨{⟨valtio, nimi⟩}⟩, c(W) = c(L)
p8 Ps = {⟨valtio, nimi⟩}, c(Ps) = c(W)
p12 P = ⟨valtio, nimi⟩, c(P) = c(Ps)
p15 Pc = ⟨valtio, nimi⟩, t(Pc) = Tables, c(Pc) = c(P)
p18 EN = valtio, Pbc2 = ⟨nimi⟩, t(EN) = t(Pc1), t(Pc2) = Tables, c(EN) = 1, c(Pc2) =
c(Pc1)

     p22 qs(EN) = ⟨⟨⟨⟩, ⟨⟨⟨6,valtio⟩⟩, ⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩

     p16 EN = nimi, t(EN) = t(Pc), c(EN) = c(Pc)
     p24 qs(EN) = ⟨⟨⟨⟨⟨2,joki⟩,nimi⟩, ⟨⟨4,järvi⟩,nimi⟩, ⟨⟨6,valtio⟩,nimi⟩,
     ⟨⟨7,kaupunki⟩,nimi⟩⟩, ⟨⟩, ⟨⟩, ⟨⟩⟩⟩
     p16 qs(Pc) = qs(EN)

p18 qs(Pc1) = ⟨⟨⟨⟨⟨6,valtio⟩,nimi⟩⟩, ⟨⟨⟨6,valtio⟩, ⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩
p15 qs(P) = qs(Pc)
p12 qss(Ps) = ⟨qs(P)⟩ = ⟨⟨⟨⟨⟨6,valtio⟩,nimi⟩⟩, ⟨⟨⟨6,valtio⟩, ⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩⟩
p8 qss(W) = qss(Ps)
p7 qss(L) = qss(W)
p2 qss(H) = ⟨⟨⟨⟨⟨6,valtio⟩,nimi⟩⟩, ⟨⟨⟨6,valtio⟩, ⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩⟩

p1 string(S) =

SELECT t1.valtionimi
FROM
(SELECT valtio.nimi AS valtionimi FROM valtio) AS t1
WHERE (t1.valtionimi IS NOT NULL)
ORDER BY t1.valtionimi
```

Kuva 31. Toinen esimerkki monikkoesityksen kääntämisestä SQL-kyselyksi attribuuttikieliopin avulla. Monikkoesitys on muodostettu XIL-kyselystä `SELECT valtio/nimi`.

Kuvan 31 jäsennys syntyy XIL-kyselyn `SELECT valtio/nimi` monikkoesityksestä ⟨⟨⟨{⟨valtio, nimi⟩}⟩, ⟨⟩⟩⟩ ja esimerkkinä käytetystä valtiotietokannasta. Kysely tarkoittaa, että valitaan valtioon välittömästi liittyvä *nimi*-solmu. Jäsentäminen tapahtuu produktioissa p1-p15 kuten kuvan 30 esimerkissä. Produktiosta p15 siirrytään kuitenkin produktioon p18. Tähän produktioon siirrytään, koska Pc-monikko sisältää useamman kuin yhden solmun. Tässä tapauksessa Pc-monikko sisältää kaksi solmua. Säännössä EN:n t-attribuuttiin asetetaan Pc1:n t-attribuutin arvo, koska edeltävässä produktiossa määritellyt taulut liittyvät tähän nonterminaaliin. Pc2:n t-attribuuttiin asetetaan *Tables*-

joukko, koska jäsennyksessä aiemmin määräytyneet taulut eivät liity enää tähän nonterminaaliin (polun ensimmäisen solmun jälkeisiin solmuihin). EN:n c-attribuuttiin asetetaan arvo 1, koska vain polun viimeinen solmu voi viitata sarakkeeseen, mutta Pc2:n c-attribuuttiin asetetaan Pc1:n c-attribuutin arvo. Produktiosta p18 muodostuu kaksi haaraa. Ensimmäisessä haarassa jäsennetään polun ensimmäinen solmu (*valtio*) ja siirrytään produktioon p22, koska tämä solmu viittaa tauluun. Seuraavaksi siirrytään käsittelemään synteettisiä attribuutteja. Koska valtio-niminen taulu löytyy joukosta t(EN), qs-attribuuttiin asetetaan monikko ⟨⟨⟨⟩, ⟨⟨⟨6,valtio⟩⟩, ⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩. Toisessa haarassa jatketaan loppupolun jäsentämistä ja siirrytään produktioon p16, koska tämä polku sisältää yhden solmun. Tästä siirrytään lehtiproduktioon p24, koska solmu viittaa sarakkeeseen. Koska EN:n c-attribuutin arvo on 2, asetetaan qs-attribuuttiin monikko ⟨⟨⟨⟨⟨2,joki⟩,nimi⟩, ⟨⟨4,järvi⟩,nimi⟩, ⟨⟨6,valtio⟩,nimi⟩, ⟨⟨7,kaupunki⟩,nimi⟩⟩, ⟨⟩, ⟨⟩, ⟨⟩⟩⟩. Säännössä p16 qs-attribuutin arvo ei muutu, mutta säännössä p18 yhdistetään kahdessa eri haarassa muodostuneet qs-attribuutit. Pc1:n qs-attribuuttiin muodostuu monikko ⟨⟨⟨⟨⟨6,valtio⟩,nimi⟩⟩, ⟨⟨⟨6,valtio⟩, ⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩. Ensimmäiseen alkioon muodostuu vain *valtio*-taulun *nimi*-sarake, koska säännössä sarakkeen tulee liittyä välittömästi polussa aiemmin määriteltyyn tauluun. Säännöissä p15-p2 attribuutit muodostetaan kuten kuvan 30 esimerkissä. Säännössä p1 SQL-kyselyyn muodostuu vain yksi sarake ja yksi taulu.

```
p1 H = ⟨⟨⟨{⟨valtio, //, nimi⟩}⟩, ⟨⟩⟩⟩
p2 L = ⟨⟨{⟨valtio, //, nimi⟩}⟩, ⟨⟩⟩, c(L) = 2
p7 W = ⟨{⟨valtio, //, nimi⟩}⟩, c(W) = c(L)
p8 Ps = {⟨valtio, //, nimi⟩}, c(Ps) = c(W)
p12 P = ⟨valtio, //, nimi⟩, c(P) = c(Ps)
p15 Pc = ⟨valtio, //, nimi⟩, t(Pc) = Tables, c(Pc) = c(P)
p19 EN = valtio, Pbc2 = ⟨nimi⟩, t(EN) = t(Pc1), t(Pc2) = Tables, c(EN) = 1, c(Pc2) =
c(Pc1)

     p22 qs(EN) = ⟨⟨⟨⟩, ⟨⟨⟨6,valtio⟩, ⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩

     p16 EN = nimi, t(EN) = t(Pc), c(EN) = c(Pc)
     p24 qs(EN) = ⟨⟨⟨⟨⟨2,joki⟩,nimi⟩, ⟨⟨4,järvi⟩,nimi⟩, ⟨⟨6,valtio⟩,nimi⟩,
     ⟨⟨7,kaupunki⟩,nimi⟩⟩, ⟨⟩, ⟨⟩, ⟨⟩⟩⟩
     p16 qs(Pc) = qs(EN)

p19 qs(Pc1) = ⟨⟨⟨⟨⟨2,joki⟩,nimi⟩, ⟨⟨4,järvi⟩,nimi⟩, ⟨⟨6,valtio⟩,nimi⟩, ⟨⟨7,kaupunki⟩,nimi⟩⟩,
⟨⟨⟨6,valtio⟩, ⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩
p15 qs(P) = qs(Pc)
p12 qss(Ps) = ⟨qs(P)⟩
p8 qss(W) = qss(Ps)
p7 qss(L) = qss(W)
p2 qss(H) = ⟨⟨⟨⟨⟨⟨2,joki⟩,nimi⟩, ⟨⟨4,järvi⟩,nimi⟩, ⟨⟨6,valtio⟩,nimi⟩, ⟨⟨7,kaupunki⟩,nimi⟩⟩,
⟨⟨⟨6,valtio⟩, ⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩⟩

p1 string(S) =

SELECT t1.valtionimi, t1.järvinimi, t1.jokinimi, t1.kaupunkinimi
FROM
(SELECT valtio.nimi AS valtionimi, järvi.nimi AS järvinimi, joki.nimi AS jokinimi,
kaupunki.nimi AS kaupunkinimi
FROM valtio LEFT JOIN sijaitsee ON valtio.nimi = sijaitsee.valtio_nimi LEFT JOIN
järvi ON sijaitsee.järvi_nimi = järvi.nimi LEFT JOIN virtaa ON valtio.nimi =
virtaa.valtio_nimi LEFT JOIN joki ON virtaa.joki_nimi = joki.nimi LEFT JOIN kaupunki
ON valtio.nimi = kaupunki.valtio_nimi) AS t1
WHERE (t1.valtionimi IS NOT NULL OR t1.järvinimi IS NOT NULL OR t1.jokinimi IS NOT
NULL OR t1.kaupunkinimi IS NOT NULL)
ORDER BY t1.valtionimi
```

Kuva 32. Kolmas esimerkki monikkoesityksen kääntämisestä SQL-kyselyksi attribuuttikieliopin avulla. Monikkoesitys on muodostunut XIL-kyselystä `SELECT valtio//nimi`.

Kuvan 32 jäsennys syntyy XIL-kyselyn SELECT valtio//nimi monikkoesityksestä ⟨⟨⟨{⟨valtio, //, nimi⟩}⟩, ⟨⟩⟩⟩ ja esimerkkinä käytetystä valtiotietokannasta. Kysely tarkoittaa, että valitaan valtion nimi-jälkeläiselementit eli hierarkiasta valtio-solmun alta löytyvät nimi-solmut. Jäsentäminen tapahtuu produktioissa p1-p15, kuten kuvan 30 ja 31 esimerkeissä. Produktiosta p15 siirrytään produktioon p19, koska polun ensimmäinen ja kolmas solmu on erottu //-solmulla. Produktiosta p19 muodostuu kaksi haaraa, joista ensimmäisessä jäsennetään polun ensimmäinen solmu (valtio). Ensimmäisessä haarassa siirrytään produktioon p22, koska solmu viittaa tauluun. Koska valtio-niminen taulu löytyy EN:n t-attribuutista, asetetaan qs-attribuuttiin monikko ⟨⟨⟨⟩, ⟨⟨⟨6,valtio⟩⟩, ⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩. Toisessa haarassa jäsennetään loppupolku ja siirrytään produktioon p16 sekä siitä produktioon p24, koska nimi-solmu viittaa sarakkeeseen. Säännössä p24 qs-attribuuttiin asetetaan kaikki tietokannan nimi-sarakkeet. Säännössä p19 yhdistetään kahdessa haarassa muodostuneet qs-attribuutit. Koska tällä kertaa polussa on osoitettu haettavan valtion jälkeläiselementtejä, asetetaan qs-attribuuttiin kaikki aiemmin muodostuneet sarakkeet. Säännöissä p15-p12 qs-attribuutit muodostetaan kuten aiemmissa esimerkeissä. Säännössä p1 muodostuu SQL-kyselyn sanallinen muoto. SQL-kyselyn FROM-osaan liitetään aiempien vaiheiden kautta muodostunut valtio-taulu ja SELECT-osaan joki.nimi-, järvi.nimi-, valtio.nimi- ja kaupunki.nimi-sarakkeet. Sarakkeita liitettäessä tarkastetaan, että sarakkeen taulu löytyy FROM-osasta. Jos näin ei ole, liitetään sarakkeen taulu FROM-osaan aiemmin muodostuneista tauluista viimeiseen. Tässä esimerkissä tämä viimeinen taulu on valtio-taulu ja joki-, järvi- ja kaupunki-taulut liitetään valtio-tauluun kukin vuorollaan ja lyhyintä mahdollista polkua käyttäen.

```
p1 H = ⟨ ⟨⟨{⟨valtio⟩}⟩, ⟨{⟨järvi, nimi = Näsijärvi⟩}⟩⟩, ⟨⟨{⟨nimi⟩}⟩, ⟨⟩⟩ ⟩
p3 Hu = ⟨⟨⟨{⟨valtio⟩}⟩, ⟨{⟨järvi, nimi = Näsijärvi⟩}⟩⟩⟩, L = ⟨⟨{⟨nimi⟩}⟩, ⟨⟩⟩, c(Hu) = 1, c(L) = 2

     p4 L = ⟨⟨{⟨valtio⟩}⟩, ⟨{⟨järvi, nimi = Näsijärvi⟩}⟩⟩, c(L) = c(Hu)
     p6 W = ⟨{⟨valtio⟩}⟩, I = ⟨{⟨järvi, nimi = Näsijärvi⟩}⟩, c(W) = c(L), c(I) = 3
          p8 Ps = {⟨valtio⟩}, c(Ps) = c(W)
          p12 P = ⟨valtio⟩, c(P) = c(Ps)
          p15 Pc = ⟨valtio⟩, t(Pc) = Tables, c(Pc) = c(P)
          p16 EN = valtio, t(EN) = t(Pc), c(EN) = c(Pc)
          p22 qs(EN) = ⟨⟨⟨⟩, ⟨⟨⟨6,valtio⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩
          p16 qs(Pc) = qs(EN)
          p15 qs(P) = qs(Pc)
          p12 qss(Ps) = ⟨qs(P)⟩
          p8 qss(W) = qss(Ps)

          p10 Ps = {⟨järvi, nimi = Näsijärvi⟩}, c(Ps) = c(I)
          p12 P = ⟨järvi, nimi = Näsijärvi⟩, c(P) = c(Ps)
          p15 Pc = ⟨järvi, nimi = Näsijärvi⟩, t(Pc) = Tables, c(Pc) = c(P)
          p18 EN = järvi, Pc2 = ⟨nimi = Näsijärvi⟩, t(EN) = t(Pc1), t(Pc2) = Tables, c(EN)
          = 1, c(Pc2) = c(Pc1)

               p22 qs(EN) = ⟨⟨⟨⟩, ⟨⟨⟨4,järvi⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩

               p16 EN = 'nimi = Näsijärvi', t(EN) = t(Pc), c(EN) = c(Pc)
               p25 qs(EN) = ⟨⟨⟨⟩, ⟨⟩, ⟨⟨⟨⟨2,joki⟩,nimi,= Näsijärvi⟩, ⟨⟨4,järvi⟩,nimi,=
               Näsijärvi⟩, ⟨⟨6,valtio⟩,nimi,= Näsijärvi⟩, ⟨⟨7,kaupunki⟩,nimi,= Näsijärvi⟩⟩⟩,
               ⟨⟩⟩⟩
               p16 qs(Pc) = qs(EN)

          p18 qs(Pc) = ⟨⟨⟨⟩, ⟨⟨⟨4,järvi⟩,⟨⟩⟩⟩, ⟨⟨⟨⟨4,järvi⟩,nimi,= Näsijärvi⟩⟩⟩, ⟨⟩⟩⟩
          p15 qs(P) = qs(Pc)
          p12 qss(Ps) = ⟨qs(P)⟩
          p10 qss(I) = qss(Ps)
     p6 qss(L) = ⟨⟨⟨⟨⟩, ⟨⟨⟨6,valtio⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟨⟨{⟨⟨6,valtio⟩,nimi⟩}, ⟨⟨⟨4,järvi ⟩,⟨⟩⟩⟩,
     ⟨⟨⟨⟨4,järvi⟩,nimi,= Näsijärvi⟩⟩⟩⟩⟩ ⟩⟩⟩⟩
     p4 qss(Hu) = ⟨⟨⟨⟨⟩, ⟨⟨⟨6,valtio⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟨⟨{⟨⟨6,valtio⟩,nimi⟩}, ⟨⟨⟨6,valtio⟩,⟨5,sijaitsee⟩⟩,
     ⟨⟨5,sijaitsee⟩,⟨4,järvi⟩⟩⟩, ⟨⟨⟨⟨4,järvi⟩,nimi,= Näsijärvi⟩⟩⟩⟩⟩ ⟩⟩⟩⟩


jatkuu
```

```
jatkuu


      p7 W = ⟨{⟨nimi⟩}⟩, c(W) = c(L)
      p8 Ps = {⟨nimi⟩}, c(Ps) = c(W)
      p12 P = ⟨nimi⟩, c(P) = c(Ps)
      p15 Pc = ⟨nimi⟩, t(Pc) = Tables, c(Pc) = c(P)
      p16 EN = nimi, t(EN) = t(Pc), c(EN) = c(Pc)
      p24 qs(EN) = ⟨⟨⟨⟨⟨2,joki⟩,nimi⟩,
      ⟨⟨4,järvi⟩,nimi⟩,⟨⟨6,valtio⟩,nimi⟩,⟨⟨7,kaupunki⟩,nimi⟩⟩, ⟨⟩, ⟨⟩, ⟨⟩⟩⟩
      p16 qs(Pc) = qs(EN)
      p15 qs(P) = qs(Pc)
      p12 qss(Ps) = ⟨qs(P)⟩
      p8 qss(W) = qss(Ps)
      p7 qss(L) = qss(W)

p3 qss(H) = ⟨⟨⟨⟨⟨2,joki⟩,nimi⟩, ⟨⟨4,järvi⟩,nimi⟩,⟨⟨6,valtio⟩,nimi⟩,⟨⟨7,kaupunki⟩,nimi⟩⟩,
⟨⟨⟨6,valtio⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟨⟨{⟨⟨6,valtio⟩,nimi⟩}, ⟨⟨⟨6,valtio⟩,⟨5,sijaitsee⟩⟩,
⟨⟨5,sijaitsee⟩,⟨4,järvi⟩⟩⟩,  ⟨⟨⟨⟨4,järvi⟩,nimi,= Näsijärvi⟩⟩⟩⟩⟩ ⟩⟩⟩⟩

p1 string(S) =

SELECT t1.valtionimi, t1.järvinimi, t1.jokinimi, t1.kaupunkinimi
FROM
(SELECT valtio.nimi AS valtionimi, järvi.nimi AS järvinimi, joki.nimi AS jokinimi,
kaupunki.nimi AS kaupunkinimi
FROM valtio LEFT JOIN sijaitsee ON valtio.nimi = sijaitsee.valtio_nimi LEFT JOIN
järvi ON sijaitsee.järvi_nimi = järvi.nimi LEFT JOIN virtaa ON valtio.nimi =
virtaa.valtio_nimi LEFT JOIN joki ON virtaa.joki_nimi = joki.nimi LEFT JOIN kaupunki
ON valtio.nimi = kaupunki.valtio_nimi
WHERE valtio.nimi IN (SELECT valtio.nimi FROM valtio LEFT JOIN sijaitsee ON
valtio.nimi = sijaitsee.valtio_nimi LEFT JOIN järvi ON sijaitsee.järvi_nimi =
järvi.nimi WHERE järvi.nimi = 'Näsijärvi')) AS t1
WHERE (t1.jokinimi IS NOT NULL OR t1.järvinimi IS NOT NULL OR t1.valtionimi IS NOT
NULL OR t1.kaupunkinimi IS NOT NULL)
ORDER BY t1.valtionimi
```

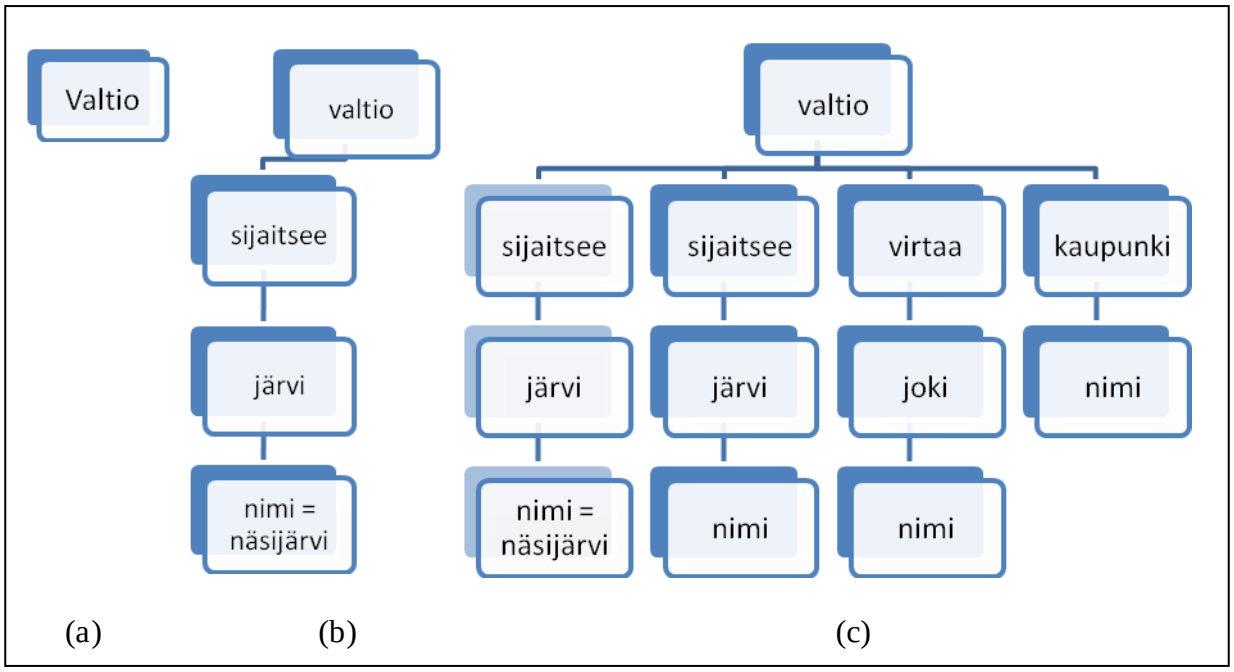
Kuva 33. Neljäs esimerkki monikkoesityksen kääntämisestä SQL-kyselyksi attribuuttikieliopin avulla. Monikkoesitys on muodostunut XIL-kyselystä `SELECT nimi FROM valtio WHERE järvi/nimi = Näsijärvi`.

Kuvan 33 jäsennys syntyy XIL-kyselyn *SELECT nimi FROM valtio* WHERE järvi/nimi = Näsijärvi monikkoesityksestä ⟨⟨⟨{⟨valtio⟩}⟩, ⟨{⟨järvi, nimi = Näsijärvi⟩}⟩⟩, ⟨⟨{⟨nimi⟩}⟩, ⟨⟩⟩⟩ ja esimerkkinä käytetystä valtiotietokannasta. Kysely tarkoittaa, että valitaan valtion *nimi*-jälkeläiselementit sellaisesta valtiosta, jonka järvi-jälkeläiselementin *nimi*-lapsielementti on arvoltaan Näsijärvi. Toisin sanoen etsitään hierarkiasta *valtio*-solmun alla olevia *nimi*-solmuja sellaisesta *valtio*-solmusta, jonka alla *järvi*-solmun *nimi*-solmun arvo on Näsijärvi. Jäsentäminen aloitetaan produktiosta p1, josta siirrytään produktioon p3, koska H-monikko sisältää kaksi L-monikkoa. Hu:n c-attribuutin arvoksi asetetaan 1 ja L:n c-attribuutin arvoksi 2. Produktiosta p3 muodostuu kaksi haaraa, joista ensimmäisessä lähdetään jäsentämään Hu-nonterminaalia (FROM-WHERE) siirtymällä produktioon p4.

Produktiosta p4 siirrytään produktioon p6. Myös produktiosta p6 muodostuu kaksi haaraa, joista ensimmäisessä jäsennetään W-monikko (FROM). Ensimmäisen haaran produktioissa p8-p22 sääntöjen attribuutit muodostetaan kuten aiemmissa esimerkeissä. Toisessa haarassa jäsennetään I-monikko (WHERE). Tämän haaran ensimmäinen produktio on p10, koska I-monikko sisältää vain yhden Ps-joukon (WHERE-osassa ei esiinny OR- tai AND-operaattoreilla yhdistettyjä polkuja). Loppuhaaraan kuuluvat attribuutit muodostetaan kuten kuvan 31 esimerkissä, paitsi produktion p24 sijasta haara päättyy produktioon p25, koska solmu sisältää ehdon. Säännössä p6 liitetään I:n kuvaamat ehdot W:n qss-attribuutin alkioiden alikyselyiksi. Toisin sanoen säännössä ei muuteta W:n qss-attribuutin alkioita, mutta liitetään niihin I:n qss-attribuutin alkiot. Säännössä p4 muodostetaan kyselyn hierarkia. W:n (FROM-osan) osoittama hierarkia on kuvassa 34 hierarkia (a). Kuvan 34 hierarkia (b) kuvaa I:n (WHERE-osan) osoittamien ehtojen muodostamaa hierarkiaa.

Produktiossa p3 muodostuneessa toisessa haarassa jäsennetään L-nonterminaali (SELECT). Produktiot p7-p24 ja niihin liittyvät attribuutit muodostetaan kuten kuvan 31 esimerkissä. Noustaessa takaisin produktioon p3 yhdistetään Hu:n qss-attribuuttin ⟨⟨⟨⟩, x2, x3, x4⟩⟩-alkiot ja L:n qss-attribuuttin ⟨⟨y1, y2, y3, y4⟩⟩-alkiot. Kummassakin qss-attribuutissa on yksi alkio, jolloin myös säännön qss-attribuuttiin muodostuu yksi ⟨⟨z1, z2, z3, z4⟩⟩-alkio. Alkioista x2 ja y2 muodostettu hierarkia asetetaan z2:een. Hierarkiaan muodostuu tässä vaiheessa vain yksi taulu, koska muita ei alkuperäisessä kyselyssä ole. Tämä hierarkia mahdollistaa kaikki y1:ssä olevat sarakkeet, joten ne asetetaan z1:een. Alkio x4 on muodostettu jo aiemmin ja y2:n hierarkia ei voi rajoittaa sitä, joten x4:ää ei muuteta tässä säännössä.

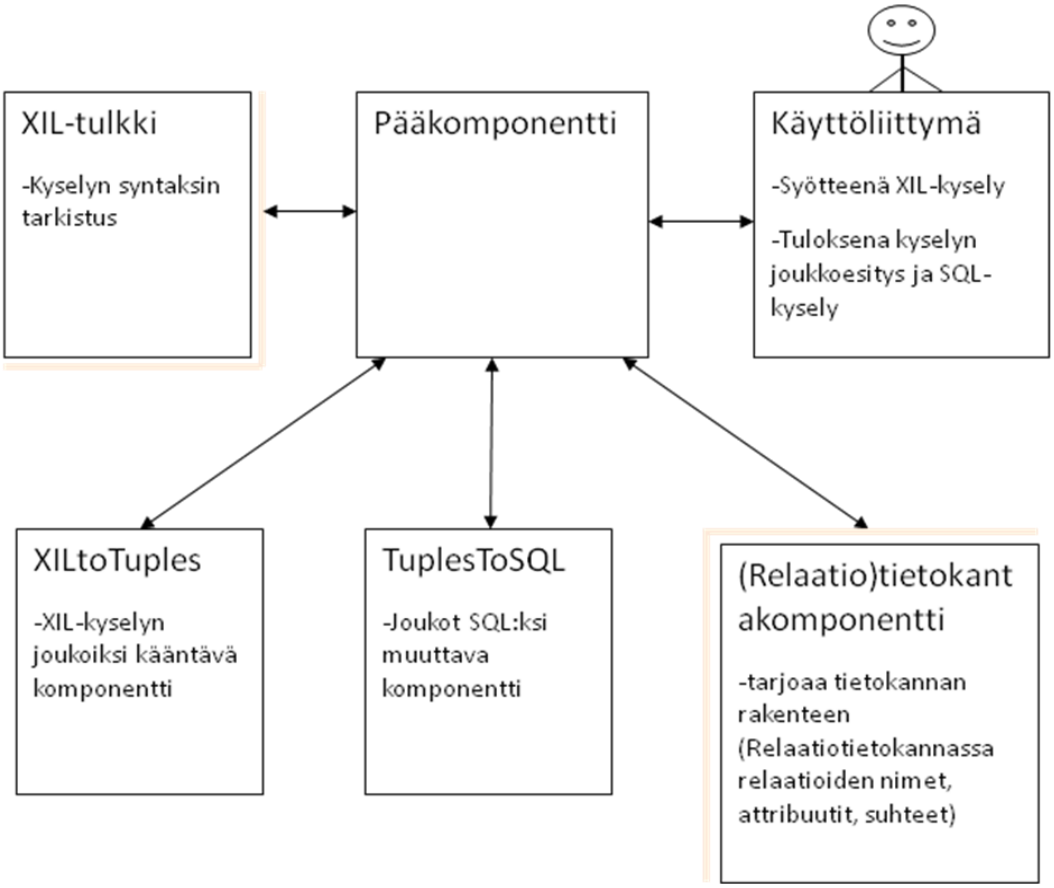
Säännössä p1 muodostetaan SQL-kyselyn SELECT- ja FROM-osa kuten edellä. WHERE-osaan muodostetaan alikysely, joka rajaa kyselyyn vain ne taulun valtiot, jotka täsmäävät alkuperäisen XIL-kyselyn WHERE-osassa olevan ehdon kanssa.



Kuva 34. XIL-kyselystä `SELECT nimi FROM valtio WHERE järvi/nimi = Näsijärvi` vaiheittain muodostuva hierarkia. Hierarkia (a) syntyy kyselyn FROM-osasta, jonka polussa on vain yksi taulu. Hierarkia (b) syntyy kyselyn FROM-WHERE -osasta, jossa FROM-osan hierarkiaan liitetään ehdon määrittävä haara. Hierarkia (c) syntyy koko kyselystä, missä vasemmanpuoleisin haara liittyy WHERE-osaan ja loput haarat liittyvät SELECT-osaan.

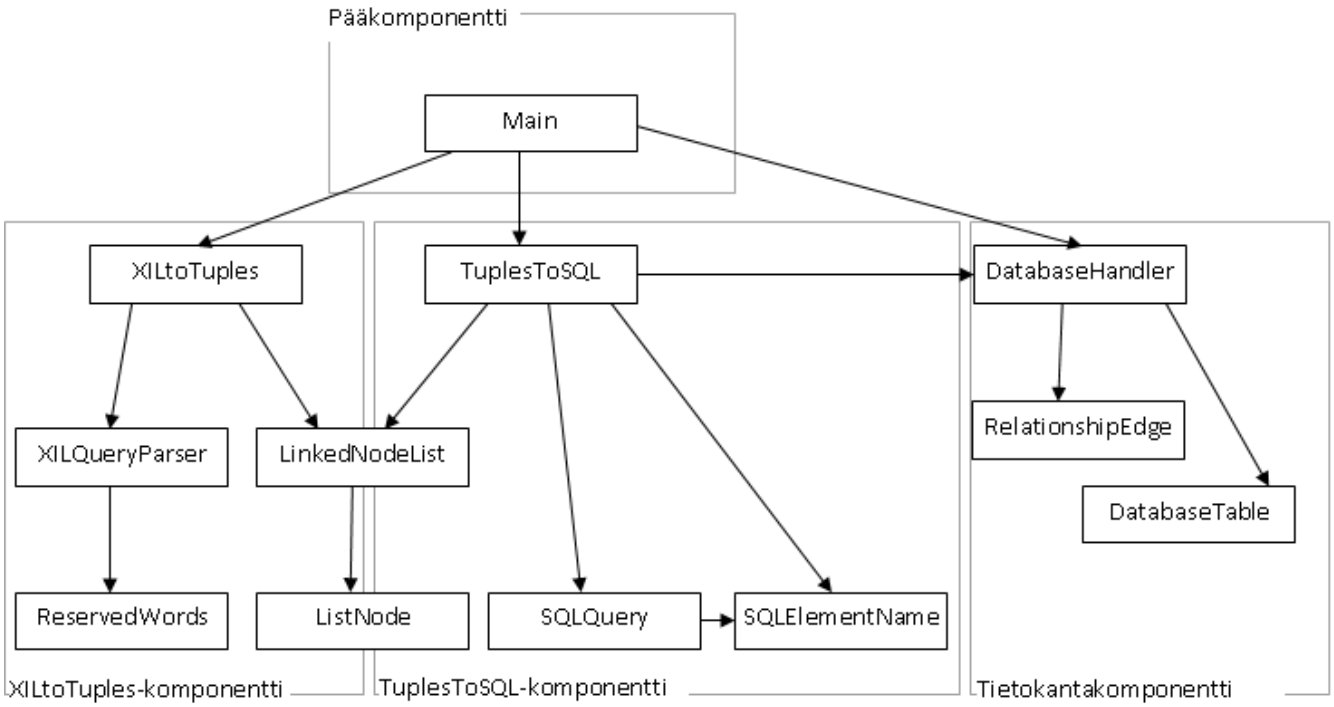
8. XILtoSQL-ohjelman kuvaus

Java-ohjelmointikielellä ja NetBeans IDE -ohjelmointiympäristössä toteutettu XILtoSQL-ohjelma koostuu käyttöliittymästä, pääkomponentista, XIL-tulkista, XILtoTuples- ja TuplesToSQL-komponenteista sekä tietokantakomponentista. Nämä on esitetty kuvassa 35. Käyttäjä käyttää ohjelmaa komentorivipohjaisen käyttöliittymän kautta, johon syötetään XIL-kysely. Vastauksena XIL-kyselyyn ohjelma palauttaa luvussa 7.2. kuvatun kyselyn monikkoesityksen ja SQL-kyselyn.



Kuva 35. XILtoSQL-ohjelman komponentit.

Ohjelman luokat löytyvät komponenteittain jaoteltuna kuvasta 36. Näiden luokkien lisäksi ohjelmassa on käytetty Javan sisään rakennettuja yksinkertaisia tietotyyppi- ja tietorakenneluokkia (java.lang- ja java.util-pakkauksista) sekä JGrapht-kirjastoa (http://jgrapht.org/) tietokannan suhteiden selvittämiseen. Mustareunainen laatikko kuvaa luokkaa, harmaareunainen laatikko kuvaa komponenttia ja nuolet kuvaavat luokkien käyttämiä palveluja. Kuvassa 36 kuvataan vain ne komponentit ja luokat, jotka työssä on itse toteutettu, joten XIL-tulkkia, Javan valmiita luokkia tai JGrapht-kirjaston luokkia ei kuvata.

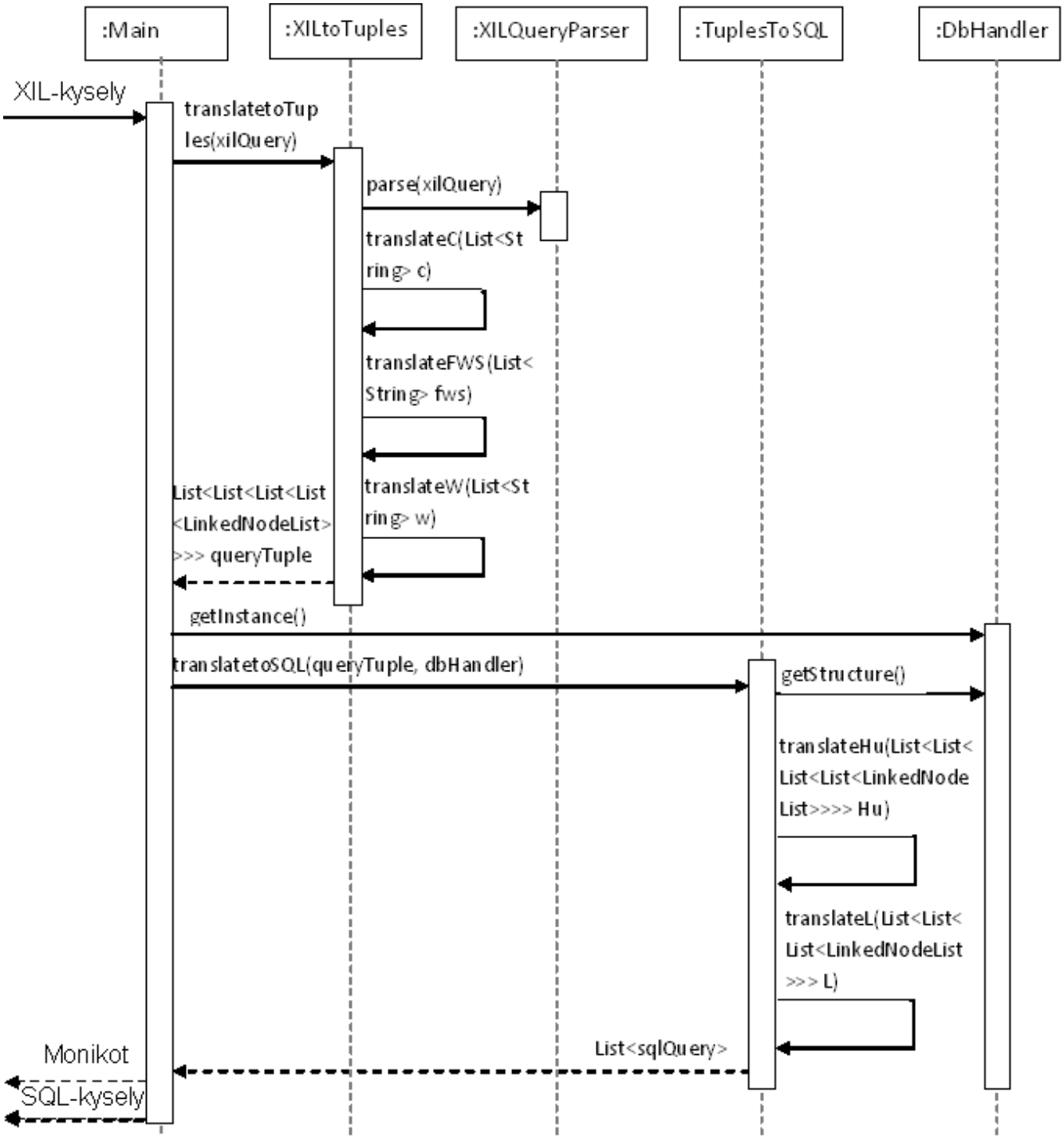


Kuva 36. XILtoSQL-ohjelman luokat komponenteittain.

Kuvan 37 UML sekvenssikaavio kuvaa luokkien välistä vuorovaikutusta esittämällä XIL-kyselyn käännöksen SQL-kyselyksi. Kaikkea vuorovaikutusta ei ole kuvattu kaavion yksinkertaistamiseksi. Sekvenssikaaviossa ei ole kuvattu luokkia ReservedWords, LinkedNodeList, ListNode, SQLQuery, SQLElement, RelationshipEdge ja DatabaseTable, jotta kaavio pysyisi helppolukuisena. Lisäksi ohjelman kulun ymmärtäminen ei vaadi näitä luokkia, koska näiden luokkien päätehtävä on mahdollistaa tarvittavan tiedon tallennus ja käsittely. Sekvenssikaaviossa laatikot, joissa on tekstiä, kuvaavat luokkia. Luokkien alla olevat laatikot kuvaavat luokan elinikää ja nuolet luokkien välistä vuorovaikutusta. Katkoviivalla esitetyt nuolet ovat vaihtoehtoisia ja kuvaavat luokan palauttamaa tietoa. Seuraavaksi tarkastellaan kuvissa esitettyjä luokkia, komponentteja ja näiden vuorovaikutusta yksityiskohtaisemmin.

Pääkomponentti koostuu Main-luokasta, joka hoitaa yhteydenpidon ohjelman komponenttien välillä tai välittää instanssin tarvittavaan luokkaan yhteydenpitoa varten. Pääkomponentille välitetty syntaksiltaan oikeellinen XIL-kysely välitetään XILtoTuples-komponentille, joka kääntää kyselyn yleistetyksi monikkoesitykseksi XILtoTuples-attribuuttikieliopin mukaisesti. XILtoTuples-komponentti koostuu XILtoTuples-, XILQueryParser-, LinkedNodeList-, ReservedWords- ja ListNode-luokasta. XILtoTuples-luokka pohjautuu attribuuttikielioppiin ja käyttää XILQueryParser- ja LinkedNodeList-luokkien tarjoamia palveluja. XILQueryParser jäsentää XIL-kyselyn SELECT-, WHERE- ja FROM-WHERE -osiin ja välittää nämä osat XILtoTuples-luokalle. LinkedNodeList-luokkaa käytetään XIL-kyselystä muodostuneiden polkujen (monikkoesityksen taso 1 ks. kuva 26 sivu 48) tallentamiseen ja

käsittelemiseen ja ListNode-luokkaa polkujen elementtien tallentamiseen ja käsittelemiseen. ReservedWords-luokka sisältää XIL-kyselykielen varatut sanat ja mahdollistaa XIL-kyselyn jäsentämisen SELECT-, WHERE- ja FROM-WHERE -osiin sekä näiden osien sisältämien polkujen jäsentämisen.



Kuva 37. XILtoSQL-ohjelman luokkien välistä vuorovaikutusta kuvaava sekvenssikaavio.

Seuraavaksi XILtoTuples-luokan Main-luokalle palauttama tietorakenne sekä instanssi DatabaseHandler-luokkaan välitetään TuplesToSQL-komponentille. TuplesToSQL-komponentti koostuu TuplesToSQL-, LinkedNodeList-, ListNode-, SQLQuery- ja SQLElementName-luokista. TuplesToSQL-luokka käyttää tietokantakomponenttiin kuuluvan DatabaseHandler-luokan palveluita monikkoesityksen kääntämiseen. DatabaseHandler-luokka sisältää luvussa 7.3.1. kuvatun relaatiotietokannan rakenteen. DatabaseTable-luokkaa käytetään relaatiotietokannan taulujen ja taulujen sarakkeiden tallentamiseen ja käsittelemiseen ja RelationshipEdge-luokkaa taulujen välisten pääavain-vierasavain -suhteiden tallentamiseen ja käsittelemiseen. Kuten jo edellä on sanottu samasta XIL-kyselystä muodostuu aina samanlainen yleistetty kyselyn monikkoesitys, mutta SQL-kysely voi muodostua erilaiseksi käytetystä relaatiotietokannasta ja dynaamisista säännöistä riippuen. LinkedNodeList- ja ListNode-luokat on kuvattu jo edellä. SQLQuery-luokkaa käytetään yleistetystä monikkoesityksestä muodostuneen SQL-kyselyn tallentamiseen ja käsittelemiseen ja SQLElementName-luokkaa kyselyn yksittäisten elementtien (taulut ja sarakkeet) tallentamiseen ja käsittelemiseen.

9. Vertailu ja jatkokehitys

Tässä luvussa palataan osin jo esitettyihin esimerkkikyselyihin tarkoituksena vertailla XIL- ja SQL-kyselyitä samaan tietotarpeeseen. Demonstraatiolla pyritään osoittamaan työssä kehitetyn käännösmenetelmän hyödyllisyyttä, eli kuinka alun perin XML-tiedonhakuun kehitettyä deklaratiivista tiedonhakukieltä voidaan käyttää myös rakenteiseen tietolähteeseen (relaatiotietokanta) kohdistuvissa kyselyissä siten, että hierarkkinen rakenne analysoidaan automaattisesti. Tämä puolestaan vaatii kykyä käsitellä kaavion mahdollisesti tuntemattomia aspekteja, kuten relaatioiden ja attribuuttien nimiä sekä riippuvuuksia. Seuraavilla esimerkkikyselyillä pyritään myös demonstroimaan, kuinka XIL-kysely mahdollistaa huomattavasti yksinkertaisemman kyselyilmauksen verrattuna vastaavaan SQL-esitykseen. Liitteessä 3 on esitelty kymmenen esimerkkikyselyä, joita on työssä jo tarkasteltu mm. käännöksen yhteydessä. Nyt näitä tarkastellaan kielten vertailun näkökulmasta. Lisäksi kyselyiden yhteydessä esitetään XIL-kyselyn tuottama tulos liitteen 2 esimerkkidokumenteista ja SQL-kyselyn tuottama tulos liitteen 1 relaatiotietokannasta.

Kysely 1: Etsitään tietolähteestä kaikki nimi-attribuutit ja koostetaan ne siten, että toisiinsa suhteessa olevat nimet kootaan yhteen. Tällainen XIL-kysely etsii kaikki tietolähteen *nimi*-elementit ja ryhmittelee ne juuren mukaan. XIL-kysely on esitetty kuvassa 38 ja siitä käännetyt SQL-kyselyt kuvissa 39-42.

XIL-kysely

```
SELECT nimi
```

Tulos

```
<result>
 <valtio>
  <nimi>
   Suomi
  </nimi>
  <nimi>
   Helsinki
  </nimi>
  <nimi>
   Tampere
  </nimi>
  <nimi>
   Näsijärvi
  </nimi>
  <nimi>
   Torniojoki
  </nimi>
  <nimi>
   Muoniojoki
  </nimi>
 </valtio>
 <valtio>
  <nimi>
   Ruotsi
  </nimi>
  <nimi>
   Tukholma
  </nimi>
  <nimi>
   Göteborg
  </nimi>
  <nimi>
   Torniojärvi
  </nimi>
  <nimi>
   Torniojoki
  </nimi>
  <nimi>
   Muoniojoki
  </nimi>
 </valtio>
</result>
```

Kuva 38. XIL-kysely ja kyselyn tulos liitteen 2 esimerkkidokumenteista.

SQL-kysely

```
SELECT t1.jokinimi, t1.järvinimi, t1.valtionimi, t1.kaupunkinimi
FROM
(SELECT joki.nimi AS jokinimi, järvi.nimi AS järvinimi, valtio.nimi AS
valtionimi, kaupunki.nimi AS kaupunkinimi
FROM joki LEFT JOIN liittyy ON joki.nimi = liittyy.joki_nimi LEFT JOIN
järvi ON liittyy.järvi_nimi = järvi.nimi LEFT JOIN virtaa ON joki.nimi
= virtaa.joki_nimi LEFT JOIN valtio ON virtaa.valtio_nimi = valtio.nimi
LEFT JOIN kaupunki ON valtio.nimi = kaupunki.valtio_nimi) AS t1
WHERE (t1.jokinimi IS NOT NULL OR t1.järvinimi IS NOT NULL OR
t1.valtionimi IS NOT NULL OR t1.kaupunkinimi IS NOT NULL)
ORDER BY t1.jokinimi
```

Tulos

jokinimi	järvinimi	valtionimi	kaupunkinimi
Muoniojoki		Suomi	Tampere
Muoniojoki		Suomi	Helsinki
Muoniojoki		Ruotsi	Göteborg
Muoniojoki		Ruotsi	Tukholma
Torniojoki	Torniojärvi	Ruotsi	Göteborg
Torniojoki	Torniojärvi	Suomi	Helsinki
Torniojoki	Torniojärvi	Suomi	Tampere
Torniojoki	Torniojärvi	Ruotsi	Tukholma

Kuva 39. Kuvan 38 XIL-kyselystä käännetty SQL-kysely ja kyselyn tulos, kun hierarkian juurena on käytetty *joki*-taulua.

SQL-kysely

```
SELECT t1.järvinimi, t1.jokinimi, t1.valtionimi, t1.kaupunkinimi
FROM
(SELECT järvi.nimi AS järvinimi, joki.nimi AS jokinimi, valtio.nimi AS
valtionimi, kaupunki.nimi AS kaupunkinimi
FROM järvi LEFT JOIN liittyy ON järvi.nimi = liittyy.järvi_nimi LEFT
JOIN joki ON liittyy.joki_nimi = joki.nimi LEFT JOIN sijaitsee ON
järvi.nimi = sijaitsee.järvi_nimi LEFT JOIN valtio ON
sijaitsee.valtio_nimi = valtio.nimi LEFT JOIN kaupunki ON valtio.nimi =
kaupunki.valtio_nimi) AS t1
WHERE (t1.järvinimi IS NOT NULL OR t1.jokinimi IS NOT NULL OR
t1.valtionimi IS NOT NULL OR t1.kaupunkinimi IS NOT NULL)
ORDER BY t1.järvinimi
```

Tulos

järvinimi	jokinimi	valtionimi	kaupunkinimi
Näsijärvi		Suomi	Tampere
Näsijärvi		Suomi	Helsinki
Torniojärvi	Torniojoki	Ruotsi	Göteborg
Torniojärvi	Torniojoki	Ruotsi	Tukholma

Kuva 40. Kuvan 38 XIL-kyselystä käännetty SQL-kysely ja kyselyn tulos, kun hierarkian juurena on käytetty *järvi*-taulua.

SQL-kysely

```
SELECT t1.valtionimi, t1.järvinimi, t1.jokinimi, t1.kaupunkinimi
FROM
(SELECT valtio.nimi AS valtionimi, järvi.nimi AS järvinimi, joki.nimi AS
jokinimi, kaupunki.nimi AS kaupunkinimi
FROM valtio LEFT JOIN sijaitsee ON valtio.nimi = sijaitsee.valtio_nimi
LEFT JOIN järvi ON sijaitsee.järvi_nimi = järvi.nimi LEFT JOIN virtaa ON
valtio.nimi = virtaa.valtio_nimi LEFT JOIN joki ON virtaa.joki_nimi =
joki.nimi LEFT JOIN kaupunki ON valtio.nimi = kaupunki.valtio_nimi) AS
t1
WHERE (t1.valtionimi IS NOT NULL OR t1.järvinimi IS NOT NULL OR
t1.jokinimi IS NOT NULL OR t1.kaupunkinimi IS NOT NULL)
ORDER BY t1.valtionimi
```

Tulos

valtionimi	järvinimi	jokinimi	kaupunkinimi
Ruotsi	Torniojärvi	Torniojoki	Tukholma
Ruotsi	Torniojärvi	Muoniojoki	Tukholma
Ruotsi	Torniojärvi	Torniojoki	Göteborg
Ruotsi	Torniojärvi	Muoniojoki	Göteborg
Suomi	Näsijärvi	Torniojoki	Helsinki
Suomi	Näsijärvi	Muoniojoki	Helsinki
Suomi	Näsijärvi	Torniojoki	Tampere
Suomi	Näsijärvi	Muoniojoki	Tampere

Kuva 41. Kuvan 38 XIL-kyselystä käännetty SQL-kysely ja kyselyn tulos, kun hierarkian juurena on käytetty *valtio*-taulua.

SQL-kysely

```
SELECT t1.kaupunkinimi, t1.valtionimi, t1.järvinimi, t1.jokinimi
FROM
(SELECT valtio.nimi AS valtionimi, järvi.nimi AS järvinimi, joki.nimi AS
jokinimi, kaupunki.nimi AS kaupunkinimi
FROM kaupunki LEFT JOIN valtio ON kaupunki.valtio_nimi = valtio.nimi
LEFT JOIN sijaitsee ON valtio.nimi = sijaitsee.valtio_nimi LEFT JOIN
järvi ON sijaitsee.järvi_nimi = järvi.nimi LEFT JOIN virtaa ON
valtio.nimi = virtaa.valtio_nimi LEFT JOIN joki ON virtaa.joki_nimi =
joki.nimi) AS t1
WHERE (t1.kaupunkinimi IS NOT NULL OR t1.valtionimi IS NOT NULL OR
t1.järvinimi IS NOT NULL OR t1.jokinimi IS NOT NULL)
ORDER BY t1.kaupunkinimi
```

Tulos

kaupunkinimi	valtionimi	järvinimi	jokinimi
Göteborg	Ruotsi	Torniojärvi	Torniojoki
Göteborg	Ruotsi	Torniojärvi	Muoniojoki
Helsinki	Suomi	Näsijärvi	Muoniojoki
Helsinki	Suomi	Näsijärvi	Torniojoki
Tampere	Suomi	Näsijärvi	Torniojoki
Tampere	Suomi	Näsijärvi	Muoniojoki
Tukholma	Ruotsi	Torniojärvi	Muoniojoki
Tukholma	Ruotsi	Torniojärvi	Torniojoki

Kuva 42. Kuvan 38 XIL-kyselystä käännetty SQL-kysely ja kyselyn tulos, kun hierarkian juurena on käytetty *kaupunki*-taulua.

Johtopäätös: XIL-kyselyllä voidaan ilmaista kysely tuntematta tietolähteen rakennetta. SQL-kyselyssä tietokannan rakenne on tunnettava ja kysely on spesifi tietokannan rakenteelle. Lisäksi kyselyn käännös lisää kyselyn muotoilun joustavuutta. Kohdistettaessa XIL-kysely XML-tietolähteeseen on tietolähteen hierarkia sidottu. Käännettäessä XIL-kysely SQL-kyselyksi tietolähteen hierarkia pyritään sovittamaan kyselyyn. Tästä syystä alkuperäisestä kyselystä on muodostunut neljä SQL-kyselyä. Jokainen *nimi*-sarakkeen sisältävä taulu voi olla hierarkian juuri. Kyselyitä ei synny kuitenkaan tätä enempää, koska työssä on rajattu polut aina lyhyimpiin mahdollisiin polkuihin. Tilanne on luonnollisesti toinen, kun huomioidaan kaikki mahdolliset polut.

Kysely 2: Etsitään valtioihin välittömästi liittyvät nimet. Tällainen XIL-kysely etsii *valtio*-elementin *nimi*-lapsielementit ja ryhmittelee ne tietolähteen juuren mukaan. XIL-kysely on esitetty kuvassa 43 ja siitä käännetty SQL-kysely kuvassa 44.

XIL-kysely

```
SELECT valtio/nimi
```

Tulos

```
<result>
 <valtio>
  <nimi>
   Suomi
  </nimi>
 </valtio>
 <valtio>
  <nimi>
   Ruotsi
  </nimi>
 </valtio>
</result>
```

Kuva 43. XIL-kysely ja kyselyn tulos liitteen 2 esimerkkidokumenteista.

SQL-kysely

```
SELECT t1.valtionimi
FROM
(SELECT valtio.nimi AS valtionimi FROM valtio) AS t1
WHERE (t1.valtionimi IS NOT NULL)
ORDER BY t1.valtionimi
```

Tulos

valtionimi
Ruotsi
Suomi

Kuva 44. Kuvan 43 XIL-kyselystä käännetty SQL-kysely ja kyselyn tulos.

Johtopäätös: Koska haetaan hierarkiassa vain välittömiä seuraajia (eikä SELECT-osassa ole muita polkuja), olisi mahdollista muodostaa SQL-kysely, joka vastaa kompleksisuudeltaan alkuperäistä XIL-kyselyä. Tällainen SQL-kysely on `SELECT nimi FROM valtio`. XILtoSQL-käännös liittää SQL-kyselyyn kuitenkin ehtoja, jotka lisäävät kyselyn kompleksisuutta. Vaikka nämä ehdot eivät ole tarpeellisia juuri tässä kyselyssä, saattavat ne olla tarpeellisia muissa vastaavissa kyselyissä. Kyselytyypissä on huomion arvoista, että polkuesitystä voidaan käyttää myös relaatioiden välillä kuten valtio/kaupunki, jolloin vastaavan SQL-kyselyn kompleksisuus kasvaa. Tämä korostuu polun pituuden kasvaessa. Alkuperäisestä kyselystä on muodostunut vain yksi SQL-kysely, koska hierarkian ensimmäinen taulu on ilmaistu alkuperäisessä kyselyssä yksiselitteisesti.

Kyselyt 3-4: Etsitään kaikki nimet, jotka sisältyvät hierarkiaan, jonka valtio määrää. Tällainen XIL-kysely etsii kaikki *nimi*-elementit *valtio*-elementin alta ja ryhmittelee ne tietolähteen juuren mukaan. XIL-kysely on esitetty kuvassa 45 ja siitä käännetty SQL-kysely kuvassa 46.

XIL-kysely

```
SELECT valtio//nimi          tai       SELECT nimi
                                       FROM valtio
```

Tulos

Näiden kyselyiden tulos on sama kuin kyselyssä 1. Tämä johtuu siitä, että *valtio*-elementti on esimerkkidokumenttien juurielementti.

Kuva 45. XIL-kysely ja kyselyn tulos liitteen 2 esimerkkidokumenteista.

SQL-kysely

```
SELECT t1.valtionimi, t1.järvinimi, t1.jokinimi, t1.kaupunkinimi
FROM
(SELECT valtio.nimi AS valtionimi, järvi.nimi AS järvinimi, joki.nimi AS
jokinimi, kaupunki.nimi AS kaupunkinimi
FROM valtio LEFT JOIN sijaitsee ON valtio.nimi = sijaitsee.valtio_nimi
LEFT JOIN järvi ON sijaitsee.järvi_nimi = järvi.nimi LEFT JOIN virtaa ON
valtio.nimi = virtaa.valtio_nimi LEFT JOIN joki ON virtaa.joki_nimi =
joki.nimi LEFT JOIN kaupunki ON valtio.nimi = kaupunki.valtio_nimi) AS
t1
WHERE (t1.valtionimi IS NOT NULL OR t1.järvinimi IS NOT NULL OR
t1.jokinimi IS NOT NULL OR t1.kaupunkinimi IS NOT NULL)
ORDER BY t1.valtionimi
```

Tulos

valtionimi	järvinimi	jokinimi	kaupunkinimi
Ruotsi	Torniojärvi	Torniojoki	Tukholma
Ruotsi	Torniojärvi	Muoniojoki	Tukholma
Ruotsi	Torniojärvi	Torniojoki	Göteborg
Ruotsi	Torniojärvi	Muoniojoki	Göteborg
Suomi	Näsijärvi	Torniojoki	Helsinki
Suomi	Näsijärvi	Muoniojoki	Helsinki
Suomi	Näsijärvi	Torniojoki	Tampere
Suomi	Näsijärvi	Muoniojoki	Tampere

Kuva 46. Kuvan 45 XIL-kyselystä käännetty SQL-kysely ja kyselyn tulos.

Johtopäätös: Kysely vaatii automaattista tai tapauskohtaista transitiivista prosessointia hierarkian eri tasoilla. XIL-kyselyssä tämä pystytään piilottamaan käyttäjältä, kun taas SQL-kyselyssä käyttäjä on vastuussa transitiivisten suhteiden määrittämisestä – relaatiotietokannassa kysely vaatii hierarkkisten suhteiden analyysin. Kyselyn kompleksisuus riippuu tietokannan mutkikkuudesta. Koska alkuperäisessä kyselyssä on ilmaistu hierarkian ensimmäinen taulu, muodostuu tästä vain yksi SQL-kysely.

Kysely 5: Etsitään kaikki nimet, jotka sisältyvät sellaisen joen määräämään hierarkiaan, joka sisältyy valtion määräämään hierarkiaan. Tällaisessa XIL-kyselyssä etsitään *valtio*-elementin *joki*-jälkeläiselementit ja palautetaan näiden *joki*-elementtien *nimi*-jälkeläiselementit. Tulos ryhmitellään tietolähteen juuren mukaan. XIL-kysely on esitetty kuvassa 47 ja siitä käännetty SQL-kysely kuvassa 48.

XIL-kysely

```
SELECT nimi
FROM joki
FROM valtio
```

Tulos

```
<result>
 <valtio>
  <nimi>
   Torniojoki
  </nimi>
  <nimi>
   Muoniojoki
  </nimi>
 </valtio>
 <valtio>
  <nimi>
   Torniojoki
  </nimi>
  <nimi>
   Muoniojoki
  </nimi>
 </valtio>
</result>
```

Kuva 47. XIL-kysely ja kyselyn tulos liitteen 2 esimerkkidokumenteista.

SQL-kysely

```
SELECT t1.valtionimi, t1.jokinimi, t1.järvinimi
FROM
(SELECT valtio.nimi AS valtionimi, joki.nimi AS jokinimi, järvi.nimi AS
järvinimi
FROM valtio LEFT JOIN virtaa ON valtio.nimi = virtaa.valtio_nimi LEFT
JOIN joki On virtaa.joki_nimi = joki.nimi LEFT JOIN liittyy ON joki.nimi
= liittyy.joki_nimi LEFT JOIN järvi ON liittyy.järvi_nimi = järvi.nimi)
AS t1
WHERE (t1.jokinimi IS NOT NULL OR t1.järvinimi IS NOT NULL)
ORDER BY t1.valtionimi
```

Tulos

valtionimi	jokinimi	järvinimi
Ruotsi	Torniojoki	Torniojärvi
Ruotsi	Muoniojoki	
Suomi	Torniojoki	Torniojärvi
Suomi	Muoniojoki	

Kuva 48. Kuvan 47 XIL-kyselystä käännetty SQL-kysely ja kyselyn tulos.

Johtopäätös: Myös tämä kysely vaatii automaattista tai tapauskohtaista transitiivista prosessointia. Alkuperäisestä kyselystä on muodostunut yksi SQL-kysely, koska hierarkian alku on ilmaistu yksiselitteisesti. XML-tietolähteestä poiketen relaatiotietokannasta palautuvat myös jokeen liittyvien järvien nimet, koska tietokannan suhteet mahdollistavat tämän.

Kysely 6: Etsitään kaikki nimet, jotka sisältyvät sellaisen järven määräämään hierarkiaan, joka sisältyy sellaisen joen määräämään hierarkiaan, joka sisältyy valtion määräämään hierarkiaan. Tällaisessa XIL-kyselyssä etsitään *valtio*-elementin *joki*-jälkeläiselementit, näiden *joki*-elementtien *järvi*-jälkeläiselementit ja palautetaan näiden *järvi*-elementtien *nimi*-jälkeläiselementit juuren mukaan ryhmiteltynä. XIL-kysely löytyy kuvasta 49 ja siitä käännetty SQL-kysely kuvasta 50.

XIL-kysely

```
SELECT nimi
FROM järvi
FROM joki
FROM valtio
```

Tulos

Kysely ei tuota tulosta, koska esimerkkidokumenteissa järvi-elementti ei sisälly joki-elementtiin.

Kuva 49. XIL-kysely ja kyselyn tulos liitteen 2 esimerkkidokumenteista.

SQL-kysely

```
SELECT t1.valtionimi, t1.järvinimi
FROM
(SELECT valtio.nimi AS valtionimi, järvi.nimi AS järvinimi
FROM valtio LEFT JOIN virtaa ON valtio.nimi = virtaa.valtio_nimi LEFT
JOIN joki ON virtaa.joki_nimi = joki.nimi LEFT JOIN liittyy ON joki.nimi
= liittyy.joki_nimi LEFT JOIN järvi ON liittyy.järvi_nimi = järvi.nimi)
AS t1
WHERE (t1.järvinimi IS NOT NULL)
ORDER BY t1.valtionimi
```

Tulos

valtionimi	järvinimi
Ruotsi	Torniojärvi
Suomi	Torniojärvi

Kuva 50. Kuvan 49 XIL-kyselystä käännetty SQL-kysely ja kyselyn tulos.

Johtopäätös: Tämä kysely osoittaa käännöksen joustavuuden verrattuna alkuperäisen kyselyn kohdistamiseen XML-tietolähteeseen. XML-tietolähteessä hierarkia on sidottu ja tällöin alkuperäinen kysely, missä etsitään jokeen liittyvää järveä, ei palauta mitään esimerkkidokumenteista. Sen sijaan kyselyn käännös mahdollistaa alkuperäistä kyselyä vastaavan tuloksen palauttamisen.

Kysely 7: Etsitään kaikki nimet, jotka sisältyvät sellaisen valtion määräämään hierarkiaan, johon sisältyy myös Näsijärvi-niminen järvi. Tällainen XIL-kysely etsii kaikki ehdot täyttävän *valtio*-elementin *nimi*-jälkeläiselementit. XIL-kysely löytyy kuvasta 51 ja siitä käännetty SQL-kysely kuvasta 52.

XIL-kysely

```
SELECT nimi
FROM valtio
WHERE järvi/nimi = Näsijärvi
```

Tulos

```
<result>
 <valtio>                <nimi>
  <nimi>                  Näsijärvi
   Suomi                 </nimi>
  </nimi>                 <nimi>
  <nimi>                  Torniojoki
   Helsinki              </nimi>
  </nimi>                 <nimi>
  <nimi>                  Muoniojoki
   Tampere               </nimi>
  </nimi>                </valtio>
                        </result>
```

Kuva 51. XIL-kysely ja kyselyn tulos liitteen 2 esimerkkidokumenteista.

SQL-kysely

```
SELECT t1.valtionimi, t1.järvinimi, t1.jokinimi, t1.kaupunkinimi
FROM
(SELECT valtio.nimi AS valtionimi, järvi.nimi AS järvinimi, joki.nimi AS
jokinimi, kaupunki.nimi AS kaupunkinimi
FROM valtio LEFT JOIN sijaitsee ON valtio.nimi = sijaitsee.valtio_nimi
LEFT JOIN järvi ON sijaitsee.järvi_nimi = järvi.nimi LEFT JOIN virtaa ON
valtio.nimi = virtaa.valtio_nimi LEFT JOIN joki ON virtaa.joki_nimi =
joki.nimi LEFT JOIN kaupunki ON valtio.nimi = kaupunki.valtio_nimi
WHERE valtio.nimi IN (SELECT valtio.nimi FROM valtio LEFT JOIN sijaitsee
ON valtio.nimi = sijaitsee.valtio_nimi LEFT JOIN järvi ON
sijaitsee.järvi_nimi = järvi.nimi WHERE järvi.nimi = 'Näsijärvi')) AS t1
WHERE (t1.jokinimi IS NOT NULL OR t1.järvinimi IS NOT NULL OR
t1.valtionimi IS NOT NULL OR t1.kaupunkinimi IS NOT NULL)
ORDER BY t1.valtionimi
```

Tulos

valtionimi	järvinimi	jokinimi	kaupunkinimi
Suomi	Näsijärvi	Muoniojoki	Tampere
Suomi	Näsijärvi	Torniojoki	Tampere
Suomi	Näsijärvi	Muoniojoki	Helsinki
Suomi	Näsijärvi	Torniojoki	Helsinki

Kuva 52. Kuvan 51 XIL-kyselystä käännetty SQL-kysely ja kyselyn tulos.

Johtopäätös: Sekä alkuperäisen että SQL-kyselyn tulokset vastaavat toisiaan sillä erotuksella, että relaatiotietokannan tuloksessa on toisteisuutta. Tulos voidaan kuitenkin esittää kyselyn tekijälle eri muodossa, jolloin toisteisuus ei haittaa tai se poistuu (esim. tuloksen muuttaminen XML-muotoon).

Kysely 8: Etsitään sellaiset joet, jotka sisältyvät sellaisen järven määräämään hierarkiaan, joka sisältyy valtion määräämään hierarkiaan siten, että jokeen liittyy nimi Torniojoki, järveen liittyy nimi Torniojärvi ja valtioon liittyy nimi Ruotsi. Tällainen XIL-kysely etsii aluksi ehdot täyttävän *valtio*-elementin ehdot täyttävät *järvi*-jälkeläiselementit, seuraavaksi näiden *järvi*-elementtien ehdot täyttävät *joki*-jälkeläiselementit ja lopuksi palauttaa nämä *joki*-elementit juuren mukaan ryhmiteltynä. On syytä huomauttaa, että riittää, kun *valtio*-elementtiin liittyvä ehto on tosi jossakin *valtio*-elementin *nimi*-jälkeläiselementissä. Vastaava tilanne on myös *järvi*- ja *joki*-elementtien kohdalla. XIL-kysely löytyy kuvasta 53 ja siitä käännetty SQL-kysely kuvasta 54.

XIL-kysely

```
SELECT joki
WHERE nimi = Torniojoki
FROM järvi
WHERE nimi = Torniojärvi
FROM valtio
WHERE nimi = Ruotsi
```

Tulos

Kysely ei tuota tulosta, koska esimerkkidokumenteissa joki-elementti ei sisälly järvi-elementtiin.

Kuva 53. XIL-kysely ja kyselyn tulos liitteen 2 esimerkkidokumenteista.

SQL-kysely

```
SELECT t1.valtionimi, t1.jokinimi, t1.jokipituus, t1.jokilaskujoki
FROM
(SELECT valtio.nimi AS valtionimi, joki.nimi AS jokinimi, joki.pituus AS
jokipituus, joki.laskujoki AS jokilaskujoki
FROM valtio LEFT JOIN sijaitsee On valtio.nimi = sijaitsee.valtio_nimi
LEFT JOIN järvi ON sijaitsee.järvi_nimi = järvi.nimi LEFT JOIN liittyy
ON järvi.nimi = liittyy.järvi_nimi LEFT JOIN joki ON liittyy.joki_nimi =
joki.nimi
WHERE valtio.nimi IN (SELECT valtio.nimi FROM valtio LEFT JOIN sijaitsee
ON valtio.nimi = sijaitsee.valtio_nimi LEFT JOIN järvi ON
sijaitsee.järvi_nimi = järvi.nimi LEFT JOIN virtaa ON valtio.nimi =
virtaa.valtio_nimi LEFT JOIN joki ON virtaa.joki_nimi = joki.nimi LEFT
JOIN kaupunki ON valtio.nimi = kaupunki.valtio_nimi WHERE valtio.nimi =
'Ruotsi' OR järvi.nimi = 'Ruotsi' OR joki.nimi = 'Ruotsi' OR
kaupunki.nimi = 'Ruotsi') AND järvi.nimi IN (SELECT järvi.nimi FROM
valtio LEFT JOIN sijaitsee ON valtio.nimi = sijaitsee.valtio_nimi LEFT
JOIN järvi ON sijaitsee.järvi_nimi = järvi.nimi LEFT JOIN liittyy ON
järvi.nimi = liittyy.järvi_nimi LEFT JOIN joki ON liittyy.joki_nimi =
joki.nimi WHERE järvi.nimi = 'Torniojärvi' OR joki.nimi = 'Torniojärvi')
AND joki.nimi IN (SELECT joki.nimi FROM valtio LEFT JOIN sijaitsee ON
valtio.nimi = sijaitsee.valtio_nimi LEFT JOIN järvi ON
sijaitsee.järvi_nimi = järvi.nimi LEFT JOIN liittyy ON järvi.nimi =
liittyy.järvi_nimi LEFT JOIN joki ON liittyy.joki_nimi = joki.nimi WHERE
joki.nimi = 'Torniojoki')) AS t1
WHERE (t1.jokinimi IS NOT NULL OR t1.jokipituus IS NOT NULL OR
t1.jokilaskujoki IS NOT NULL)
ORDER BY t1.valtionimi
```

Tulos

valtionimi	jokinimi	jokipituus	jokilaskujoki
Ruotsi	Torniojoki	510	

Kuva 54. Kuvan 53 XIL-kyselystä käännetty SQL-kysely ja kyselyn tulos.

Johtopäätös: Samoin kuin kyselyt 3 ja 4 tämä kysely osoittaa käännöksen joustavuuden. XML-tietolähteessä hierarkia on sidottu ja tällöin alkuperäinen kysely, missä etsitään järveen liittyvää jokea, ei palauta mitään esimerkkidokumenteista. Kyselyn käännös mahdollistaa kuitenkin alkuperäistä kyselyä vastaavan tuloksen palauttamisen.

Kysely 9: Etsitään sellaiseen jokeen välittömästi liittyvät nimet ja pinta-alat ja/tai pinta-alueet, jotka sisältyvät sellaisen järven määräämään hierarkiaan, joka sisältyy valtion määräämään hierarkiaan. Lisäksi järven määräämään hierarkiaan tulee sisältyä Torniojärvi-arvoinen nimi-solmu tai pinta-ala-solmu, jonka arvo on suurempi kuin 5000. XIL-kysely etsii aluksi *valtio*-elementin *järvi*-jälkeläiselementit, jotka täsmäävät annettuun ehtoon. Seuraavaksi etsitään näiden *järvi*-elementtien *joki*-jälkeläiselementit, joilla on *nimi*-lapsielementti ja *järvi*-elementtien *pinta-ala*- tai *pinta-alue*-jälkeläiselementit. XIL palauttaa löydetyt *nimi*- ja *pinta-ala*- tai *pinta-alue*-elementit juuren mukaan ryhmiteltynä, jos ryhmässä on ainakin yksi *nimi*-elementti ja yksi *pinta-ala*- tai *pinta-alue*-elementti. XIL-kysely löytyy kuvasta 55 ja siitä käännetty SQL-kysely kuvasta 56.

XIL-kysely

```
SELECT joki/nimi, pinta-ala|pinta-alue
FROM valtio//järvi
WHERE nimi = Torniojärvi OR pinta-ala > 5000
```

Tulos

Kysely ei tuota tulosta, koska esimerkkidokumenteissa joki-elementti ei sisälly järvi-elementtiin. Jos kuitenkin kyselyn SELECT-osasta poistettaisiin ensimmäinen polku, tuottaisi kysely seuraavan tuloksen:

```
<result>
 <valtio>
  <pinta-ala>
   330 000
  </pinta-ala>
 </valtio>
</result>
```

Kuva 55. XIL-kysely ja kyselyn tulos liitteen 2 esimerkkidokumenteista.

SQL-kysely

```
SELECT t1.valtionimi, t1.jokinimi, t2.järvipinta-ala
FROM
(SELECT valtio.nimi AS valtionimi, sijaitsee.valtio_nimi AS
sijaitseevaltio_nimi, sijaitsee.järvi_nimi AS sijaitseejärvi_nimi,
järvi.nimi AS järvinimi, joki.nimi AS jokinimi
FROM valtio LEFT JOIN sijaitsee ON valtio.nimi = sijaitsee.valtio_nimi
LEFT JOIN järvi ON sijaitsee.järvi_nimi = järvi.nimi LEFT JOIN liittyy
ON järvi.nimi = liittyy.järvi_nimi LEFT JOIN joki ON liittyy.joki_nimi =
joki.nimi
WHERE järvi.nimi IN (SELECT järvi.nimi FROM valtio LEFT JOIN sijaitsee
ON valtio.nimi = sijaitsee.valtio_nimi LEFT JOIN järvi ON
sijaitsee.järvi_nimi = järvi.nimi LEFT JOIN liittyy ON järvi.nimi =
liittyy.järvi_nimi LEFT JOIN joki ON liittyy.joki_nimi = joki.nimi WHERE
joki.nimi = 'Torniojärvi' OR järvi.nimi = 'Torniojärvi') OR järvi.nimi
IN (SELECT järvi.nimi FROM valtio LEFT JOIN sijaitsee On valtio.nimi =
sijaitsee.valtio_nimi LEFT JOIN järvi ON sijaitsee.järvi_nimi =
järvi.nimi WHERE järvi.pinta-ala > 5000)) AS t1
INNER JOIN
(SELECT valtio.nimi AS valtionimi, sijaitsee.valtio_nimi AS
sijaitseevaltio_nimi, sijaitsee.järvi_nimi AS sijaitseejärvi_nimi,
järvi.nimi AS järvinimi, järvi.pinta-ala AS järvipinta-ala
FROM valtio LEFT JOIN sijaitsee On valtio.nimi = sijaitsee.valtio_nimi
LEFT JOIN järvi ON sijaitsee.järvi_nimi = järvi.nimi
WHERE järvi.nimi IN (SELECT järvi.nimi FROM valtio LEFT JOIN sijaitsee
On valtio.nimi = sijaitsee.valtio_nimi LEFT JOIN järvi ON
sijaitsee.järvi_nimi = järvi.nimi LEFT JOIN liittyy ON järvi.nimi =
liittyy.järvi_nimi LEFT JOIN joki ON liittyy.joki_nimi = joki.nimi WHERE
joki.nimi = 'Torniojärvi' OR järvi.nimi = 'Torniojärvi') OR järvi.nimi
IN (SELECT järvi.nimi FROM valtio LEFT JOIN sijaitsee On valtio.nimi =
sijaitsee.valtio_nimi LEFT JOIN järvi ON sijaitsee.järvi_nimi =
järvi.nimi WHERE järvi.pinta-ala > 5000)) AS t2
ON t1.valtionimi = t2.valtionimi AND t1.sijaitseevaltio_nimi =
t2.sijaitseevaltio_nimi AND t1.sijaitseejärvi_nimi =
t2.sijaitseejärvi_nimi AND t1.järvinimi = t2.järvinimi
WHERE ((t1.jokinimi IS NOT NULL) AND (t2.järvipinta-ala IS NOT NULL))
ORDER BY t1.valtionimi
```

Tulos

valtionimi	jokinimi	järvipinta-ala
Ruotsi	Torniojoki	330000

Kuva 56. Kuvan 55 XIL-kyselystä käännetty SQL-kysely ja kyselyn tulos.

Johtopäätös: Joustavuus hierarkian muodostuksessa mahdollistaa käännetyn kyselyn tuottaman tuloksen.

Kysely 10: Etsitään kaikki nimet, jotka sisältyvät sellaisen joen määräämään hierarkiaan, joka sisältyy joko sellaisen sijaitsee-solmun tai sellaisen virtaa-solmun määräämään hierarkiaan, joka sisältyy valtion määräämään hierarkiaan. XIL-kysely etsii aluksi *valtio*-elementtien *sijaitsee*- tai *virtaa*-lapsielementit ja näiden elementtien *joki*-jälkeläiselementit. Lopuksi XIL-kysely palauttaa näiden *joki*-elementtien *nimi*-

jälkeläiselementit juuren mukaan ryhmiteltynä. XIL-kysely löytyy kuvasta 57 ja siitä käännetyt SQL-kyselyt kuvissa 58 ja 59.

XIL-kysely

```
SELECT nimi
FROM valtio/sijaitsee|virtaa//joki
```

Tulos

Kysely ei tuota tulosta, koska esimerkkidokumenteista ei löydy *sijaitsee*- tai *virtaa*-elementtejä.

Kuva 57. XIL-kysely ja kyselyn tulos liitteen 2 esimerkkidokumenteista.

SQL-kysely

```
SELECT t1.valtionimi, t1.jokinimi
FROM
(SELECT valtio.nimi AS valtionimi, joki.nimi AS jokinimi
FROM valtio LEFT JOIN sijaitsee ON valtio.nimi = sijaitsee.valtio_nimi
LEFT JOIN järvi ON sijaitsee.järvi_nimi = järvi.nimi LEFT JOIN liittyy
ON järvi.nimi = liittyy.järvi_nimi LEFT JOIN joki ON liittyy.joki_nimi =
joki.nimi) AS t1
WHERE (t1.jokinimi IS NOT NULL) ORDER BY t1.valtionimi
```

Tulos

valtionimi	jokinimi
Ruotsi	Torniojoki

Kuva 58. Kuvan 57 XIL-kyselystä käännetty SQL-kysely ja kyselyn tulos, kun vaihtoehtoisista solmuista on valittu *sijaitsee*.

SQL-kysely

```
SELECT t1.valtionimi, t1.jokinimi, t1.järvinimi
FROM
(SELECT valtio.nimi AS valtionimi, joki.nimi AS jokinimi, järvi.nimi AS
järvinimi
FROM valtio LEFT JOIN virtaa ON valtio.nimi = virtaa.valtio_nimi LEFT
JOIN joki ON virtaa.joki_nimi = joki.nimi LEFT JOIN liittyy ON joki.nimi
= liittyy.joki_nimi LEFT JOIN järvi ON liittyy.järvi_nimi = järvi.nimi)
AS t1
WHERE (t1.jokinimi IS NOT NULL OR t1.järvinimi IS NOT NULL)
ORDER BY t1.valtionimi
```

Tulos

valtionimi	jokinimi	järvinimi
Ruotsi	Torniojoki	Torniojärvi
Ruotsi	Muoniojoki	
Suomi	Torniojoki	Torniojärvi
Suomi	Muoniojoki	

Kuva 59. Kuvan 57 XIL-kyselystä käännetty SQL-kysely ja kyselyn tulos, kun vaihtoehtoisista solmuista on valittu *virtaa*.

Johtopäätös: XIL-kysely ei ole tuottanut tulosta esimerkkidokumenteista. Tilanteissa, joissa esimerkkitietokannasta ei löydy alkuperäisessä kyselyssä annettua solmua, myöskään käännettyä kyselyä ei synny. Esimerkkitietokannasta kuitenkin löytyivät kummatkin vaihtoehtoiset solmun nimet, joten alkuperäisestä kyselystä muodostui kaksi SQL-kyselyä.

Yllä olevien XIL-kyselyiden relationaalinen tulkinta perustuu lyhyimmän polun löytämiseen relaatiotietokannasta. Solmut voivat kuitenkin olla keskenään assosiaatiossa monen eri polun kautta – varsinkin laajassa tietokannassa. Jatkotutkimukseksi jääkin erilaisten assosiaatiopolkujen käsittely sekä näiden täsmäävyyden arviointi. Yksi mielenkiintoinen tapa tutkia tätä olisi tarjota assosiaatiopolut käyttäjälle jossakin oletetussa relevanssijärjestyksessä. Alustavan tutkimuksen perusteella lyhyin polku tarjoaa parhaan vastauksen. Tällä ei kuitenkaan saavuteta välttämättä haettua tulosta, jolloin eräs ratkaisu tähän voisi olla relevanssilajittelun soveltaminen datakeskeisiin tuloksiin. Löydetyn polun pituus voisi olla yksi tekijä tässä analyysissä. Toisin sanoen tiedonhaussa yleisesti käytettyä relevanssilajittelua voitaisiin soveltaa datakeskeiseen tiedonhakuun uusin perustein.

Tutkielmassa käsitelty XIL-kyselykieli on ensisijaisesti suunnattu XML-tiedonhakuun sekä relevanttien osien koostamiseen tuloksessa. Tässä työssä on kuitenkin keskitytty vain XIL-kyselykielen rakenteellisiin ominaisuuksiin, eli eksplisiittiseen ja implisiittiseen hierarkian tulkintaan relaatiotietokannoissa. Varsinaisten tiedonhakupiirteiden mukaan ottaminen mahdollistaisi tiedonhakuominaisuudet myös relaatiotietokannoissa. Näin sama kysely voitaisiin tehdä sekä XML- ja relaatiomuotoisiin tietolähteisiin. Esitetty tutkimus antaa tälle valmiin lähtökohdan. Muita jatkotutkimus- ja jatkokehitysaiheita ovat muun muassa SQL-kyselyn tuloksena olevan taulun muuttaminen XML-muotoon.

10. Yhteenveto

Tässä tutkimuksessa on osoitettu, että relaatiotietokannan hierarkian automaattinen muodostaminen on mahdollista. Avain tähän on tietomallien tuntemus. Jotta yhteen tietomalliin (relaatiomalliin) pohjautuvaa tietoa (dataa) voitaisiin tulkita kuten toiseen tietomalliin (XML-tietomalliin) pohjautuvaa tietoa, on kummankin tietomallin tavat organisoida ja käsitellä tietoa tunnettava. Relaatiotietokannasta voidaan muodostaa hierarkia tiukan tai symmetrisen tulkinnan mukaisesti. Tiukka tulkinta mahdollistaa juuri- ja lehtitaulujen määrittämisen relaatiotietokannasta. Symmetrinen tulkinta mahdollistaa muun muassa hierarkian sovittamisen relaatiotietokantaan niissäkin tilanteissa, joissa kyselyn tekijä ei tunne tietokannan rakennetta. Relaatiotietokannan hierarkkinen tulkinta mahdollistaa XML-kyselykielellä XIL tehtyjen kyselyjen kääntämisen SQL:ksi niin, että alkuperäisen kyselyn semantiikka säilyy. Työssä on pyritty osoittamaan käännösmenetelmän hyödyllisyys, eli kuinka käännösmenetelmä mahdollistaa helpomman kyselyn muotoilun poistamalla kyselyn tekijän vastuuta tietokannan spesifistä tuntemuksesta. XIL-kyselyn käännös SQL-kyselyksi mahdollistaa myös joustavamman tavan muotoilla kysely, kuin tilanne, jossa XIL-kysely kohdistetaan XML-muotoiseen tietoon. Tämä johtuu luonnollisesti siitä, että XML-muotoisessa tiedossa hierarkia on sidottu ja kyselyn pitää vastata tätä hierarkiaa.

Viiteluettelo

[Benedikt *et al.*, 2002] Michael Benedikt, Chee Yong Chan, Wenfei Fan, Rajeev Rastogi, Shihui Zheng and Aoying Zhou, DTD-directed publishing with attribute translation grammars. In: *Proc. of VLDB* **28** (2002), 838-849.

[Beyer *et al*., 2005] Kevin Beyer, Roberta J. Cochrane, Vanja Josifovski, Jim Kleewein, George Lapis, Guy Lohman, Bob Lyle, Fatma Özcan, Hamid Pirahesh, Normen Seemann, Tuong Truong, Bert Van der Linden, Brian Vickery, Chun Zhang, System RX: one part relational, one part XML. In: *Proc. of SIGMOD* (2005), 347-358.

[Bohannon *et al.*, 2002a] Philip Bohannon, Juliana Freire, Prasan Roy, Jérôme Siméona and Bell Laboratories, From XML schema to relations: a cost-based approach to XML storage. In: *Proc. of ICDE* **18** (2002), 64-75.

[Bohannon *et al.*, 2002b] Philip Bohannon, Sumit Ganguly, Henry F. Korth, P. P. S. Narayan and Pradeep Shenoy, Optimizing view queries in ROLEX to support navigable tree results. In: *Proc. of VLDB* **28** (2002), 119-130.

[Bunge, 1967] Mario Bunge, *Scientific Research I, The Search for Systems, Volume 3/1*, Springer-Verlag, 1967.

[Chamberlin and Boyce, 1974] Donald D. Chamberlin and Raymond F. Boyce, SEQUEL: A structured English query language. In: *Proc. of the 1974 ACM SIGFIDET (SIGMOD) Workshop on Data Description*, 249-264.

[Chen *et al.*, 2002] Yi Chen, Susan B. Davidson and Yifeng Zheng, Constraint preserving XML storage in relations. In: *Proc. of WebDB* (2002).

[Chen, 1976] Peter Pin-shan Chen, The Entity-Relationship Model: Toward a unified view of data. *TODS* **1** (March 1976), 9-36.

[Codd, 1970] E. F. Codd, A relational model of data for large shared data banks. *Comm. ACM* **13**, 6 (June 1970), 377-387.

[Date, 1989] C. J. Date, *A Guide to the SQL Standard*. Addison-Wesley, 1989.

[David, 1992] Michael M David, Advanced capabilities of the outer join. ACM *SIGMOD Record* **21**, 1 (March 1992), 65-70.

[David, 2003] Michael M David, ANSI SQL hierarchical processing can fully integrate native XML. ACM *SIGMOD Record* **32**, 1 (March 2003), 41-46.

[DBTG, 1971] Report of the CODASYL Database Task Group, *ACM*, (April 1971).

[Dehaan *et al.*, 2003] David Dehaan, David Toman, Mariano P. Consens, M. Tamer Özsu, A comprehensive XQuery to SQL translation using dynamic interval encoding. In: *Proc. of SIGMOD* (2003), 623-634.

[Deutsch and Tannen, 2003] Alin Deutsch and Val Tannen, MARS: a system for publishing XML from mixed and redundant storage. In: *Proc. of VLDB* **29** (2003), 201-212.

[Deutsch *et al.*, 1999] Alin Deutsch, Mary Fernandez and Dan Suciu, Storing semistructured data with STORED. In: *Proc. of SIGMOD* (1999), 431-442.

[DOM, 1998] Vidur Apparao, Steve Byrne, Mike Champion, Scott Isaacs, Ian Jacob, Arnaud Le Hors, Gavin Nicol, Jonathan Robie, Robert Sutor, Chris Wilson and Lauren Wood (eds.), Document Object Model (DOM) Level 1 Specification Version 1.0, W3C Recommendation 1 October, 1998. http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/. Viitattu 10.11.2011

[DOM, 2000] Arnaud Le Hors, Philippe Le Hégaret, Lauren Wood, Gavin Nicol, Jonathan Robie, Mike Champion Steve and Byrne (eds.), Document Object Model (DOM) Level 2 Core Specification Version 1.0, W3C Recommendation 13 November, 2000. http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113. Viitattu 10.11.2011

[Eisenberg and Melton, 2002] Andrew Eisenberg and Jim Melton, SQL/XML is making good progress. *ACM SIGMOD Record* **31**, 2 (June 2002), 101-108.

[Elmasri and Navathe, 1989] Ramez Elmasri and Shamkant B. Navathe, *Fundamentals of Database Systems*. Benjamin Cummings, 1989.

[Elmasri and Navathe, 1994] Ramez Elmasri and Shamkant B. Navathe, *Fundamentals of Database Systems*. Benjamin Cummings, 1994.

[Elmasri and Navathe, 2004] Ramez Elmasri and Shamkant B. Navathe, *Fundamentals of Database Systems*. Addison-Wesley, 2004.

[Fernández *et al.*, 2002] Mary Fernández, Atsuyuki Morishima and Dan Suciu, Efficient evaluation of XML middle-ware queries. *ACM SIGMOD Record* **30**, 2 (June 2001), 103-114.

[Fernández *et al.*, 2000] Mary Fernández, Wang-Chiew Tan and Dan Suciu, SilkRoute: trading between relations and XML. *Computer Networks: The International Journal of Computer and Telecommunications Networking* **33**, 1-6 (June 2000), 723-745.

[Florescu and Kossmann, 1999] Daniela Florescu and Donald Kossmann, Storing and querying XML data using an RDBMS. *Bulletin of the Technical Committee on Data Engineering* **22**, 3 (Sept. 1999), 27-34.

[Fuhr and Großjohann, 2001] Norbert Fuhr and Kai Großjohann, XIRQL: A query language for information retrieval in XML documents. In: *Proc. of SIGIR* **24** (2001), 172-180.

[Goldfarb and Prescod, 1988] Charles F. Goldfarb and Paul Prescod, *The Xml Handbook*. Prentice Hall, 1988.

[Grust, 2002] Torsten Grust, Accelerating XPath location steps. In: *Proc. of SIGMOD* (2002), 109-120.

[Hammer and McLeod, 1981] Micheal Hammer and Dennis McLeod, Database description with SDM: A semantic database model. *TODS* **6**, 3 (Sept. 1981), 351-386.

[Hongwei *et al.*, 2002] Sun Hongwei, Zhang Shusheng, Zhou Jingtao and Wang Jing, Constraints-preserving mapping algorithm from XML-schema to relational schema. In: *Engineering and Deployment of Cooperative Information Systems* (2002), 193-207.

[Infoset, 2004] John Cowan and Richard Tobin, XML Information Set (Second Edition), W3C Proposed Recommendation 4 February 2004, http://www.w3.org/TR/2004/REC-xml-infoset-20040204. Viitattu 10.11.2011.

[Jain *et al.*, 2002] Sushant Jain, Ratul Mahajan and Dan Suciu, Traslating XSLT programs to efficient SQL queries. *Proceedings of the 11th international conference on World Wide Web* **11** (2002), 616-626.

[Junkkari, 2005] Marko Junkkari, PSE: An object-oriented representation for modeling and managing part-of relationship. *JIIS* **25** (2005), 131-157.

[Junkkari, 2007] Marko Junkkari, A Concept-Oriented Data Modeling and Query Language Approach to Next Generation Information Systems. University of Tampere, Dept. of Computer Science, Report **A-2007-2**, Feb. 2007.

[Junkkari *et al.*, 2006] Marko Junkkari, Paavo Arvola and Jaana Kekäläinen, Grammatical Approach to XML Information Retrieval Query Languages. University of Tampere, Dept. of Computer Science, Report **A-2006-5**, May 2006.

[Järvelin and Niemi, 1999] Kalervo Järvelin and Timo Niemi, Integration of complex objects and transitive relationships for information retrieval. *Information Processing & Management* **35**, 5 (Sept. 1999), 655-678.

[Klettke and Meyer, 2000] Meike Klettke and Holger Meyer, XML and object-relational database systems - enhancing structural mappings based on statistics. In: *ACM SIGMOD Workshop on the Web and Databases* (2000), 151-170.

[Knuth, 1968] Donald E. Knuth, Semantics of context free languages. *Math. Syst. Theory* **2** (1968), 127-145.

[Krishnamurthy *et al.*, 2004] Rajasekar Krishanamurthy, Raghav Kaushik and Jeffrey F. Naughton, XML-to-SQL Query Translation Literature: The State of the Art and Open Problems, *Springer*, 2004.

[Krishnamurthy *et al*., 2003] Rajasekar Krishanamurthy, Venkatesan T. Chakaravarthy and Jeffrey F. Naughton, On the difficulty of finding optimal relational decompositions for XML workloads: a complexity theoretic perspective. In: *Proc. of ICDT* **9** (2003), 270-284.

[Krishnaprasad *et al.*, 2004] Muralidhar Krishnaprasad, Zhen Hua Liu, Anand Manikutty, James W. Warner, Vikas Arora and Susan Kotsovolos, Query rewrite for XML in Oracle XML DB. In: *Proc. of VLDB* **30** (2004), 1134-1145.

[Lalmas, 2009] Mounia Lalmas, *XML Retrieval*. Morgan & Claypool, 2009.

[Lecluce *et al.*, 1988] Cristophe Lecluce, Philippe Richard and Fernando Velez, O2, an Object-Oriented Data Model. *ACM SIGMOD Record* **17**, 3 (June 1988), 424-433.

[Lee and Chu, 2000] Dongwon Lee and Wesley W. Chu, Constraints-preserving transformation from XML document type deffinition to relational schema. In: *Proc. of ER* **19** (2000), 323-338.

[Li *et al.*, 2003] Chengkai Li, Philip Bohannon and P. P. S. Narayan, Composing XSL transformations with XML publishing views. In: *Proc. of SIGMOD* (2003), 515-526.

[Li and Moon, 2001] Quanzhong Li and Bongki Moon, Indexing and querying XML data for regular path expressions. In: *Proc. of VLDB* **27** (2001), 361-370.

[LINQ, 2010] LINQ (Language-Integrated Query) Visual Studio 2010. http://msdn.microsoft.com/en-us/library/bb397926.aspx. Viitattu 09.01.2012

[Liu, 1999] Mengchi Liu, Deductive database languages: problems and solutions. *ACM Computing Surveys* **31**, 1 (March 1999), 27-62.

[Mani and Lee, 2002] Murali Mani and Dongwon Lee, XML to relational conversion using theory of regular tree grammars. In: *Proc. of VLDB 2002 Workshop EEXTT and CAiSE 2002 Workshop DTWeb* (2002), 81-103.

[Manolescu *et al.*, 2001] Ioana Manolescu, Daniela Florescu and Donald Kossmann, Answering XML queries on heterogeneous data sources. In: *Proc. of VLDB* **27** (2001), 241-250.

[Niemi *et al.*, 2002] Timo Niemi, Marko Junkkari and Kalervo Järvelin, Relational Deductive Object-Oriented Modeling (DROOM) Approach for Finding, Representing and Integrating Application-Specific Concepts. *IJSEKE* **12** (2002), 415-451.

[Niemi *et al.*, 2009] Timo Niemi, Turkka Näppilä and Kalervo Järvelin, A relational data harmonization approach to XML. *JIS* **35**, 5 (Oct. 2009), 571-601.

[Paakki, 1995] Jukka Paakki, Attribute grammar paradigms - a high-level methodology in language implementation. *ACM Computing Surveys* **27**, 2 (June 1995), 196-255.

[Pal *et al*., 2004] Shankar Pal, Istvan Cseri, Oliver Seeliger, Gideon Schaller, Leo Giakoumakis and Vasili Zolotov, Indexing XML data stored in a relational database. In: *Proc. of VLDB* **30** (2004), 1146-1157.

[Runapongsa and Patel, 2002] Kanda Runapongsa and Jignesh M. Patel, Storing and querying XML data in object-relational DBMSs. In: *Proc. of EDBT* (2002), 266-285.

[Salminen and Tompa, 2001] Airi Salminen and F. W. Tompa, Requirements for XML document database systems. *DocEng* (2001), 85-94.

[Samet, 1981] P. A. Samet ed., *Query Languages - a Unified Approach*. Heyden & Son Ltd., The British Computer Society, Cambridge, 1981.

[Schmidt *et al.*, 2000] Albrecht Schmidt, Martin L. Kersten, Menzo Windhouver and Florian Waas, Efficient relational storage and retrieval of XML documents. In: *Proc. of WebDB* **3** (2000), 137-150.

[Shanmugasundaram *et al.*, 2001a] Jayavel Shanmugasundaram, Jerry Kiernan, Eugene J. Shekita, Catalina Fan and John Funderberk, Querying XML views on relational data. In: *Proc. of VLDB* **27** (2001), 261-270.

[Shanmugasundaram *et al.*, 2001b] Jayavel Shanmugasundaram, Eugene J. Shekita, Jerry Kiernan, Rajasekar Krishnamurthy, Efstratios Viglas, Jeffrey F. Naughton and Igor Tatarinov, A general technique for querying XML documents using a relational database system. *ACM SIGMOD Record* **30**, 3 (Sept. 2001), 20-26.

[Shanmugasundaram *et al.*, 2000] Jayavel Shanmugasundaram, Eugene J. Shekita, Rimon Barr, Michael Carey, Bruce Lindsay, Hamid Pirahesh and Berthold Reinwald, Efficiently publishing relational data as XML documents. *The VLDB Journal* **10**, 2-3 (2001), 133-154.

[Shanmugasundaram *et al.*, 1999] Jayavel Shanmugasundaram, Kristin Tufte, Chun Zhang, Gang He, David J. DeWitt and Jeffrey F. Naughton, Relational databases for querying XML documents: limitations and opportunities. In: *Proc. of VLDB* **25** (1999), 302-314.

[Tatarinov *et al.*, 2002] Igor Tatarinov, Stratis D. Viglas, Kevin Beyer, Jayavel Shanmugasundaram, Eugene Shekita and Chun Zhang, Storing and querying ordered XML using a relational database system. In: *Proc. of SIGMOD* (2002), 204-215.

[Teubner *et al.*, 2003] Torsten Grust, Maurice van Keulen and Jens Teubner, Staircase join: teach a relational DBMS to watch its (axis) steps. In: *Proc. of VLDB* **29** (2003), 524-535.

[Tsichritzis and Klug, 1978] Dennis Tsichritzis and Anthony C. Klug, *The ANSI/X3/SPARC DBMS Framework*. AFIPS Press, 1978.

[Ullman, 1988] Jeffrey D. Ullman, *Principles of Database and Knowledge-Base Systems, Volume I: Classical Database Systems*. Computer Science Press, Rockville, 1988.

[XDM, 2010] Anders Berglund, Mary Fernández, Ashok Malhotra, Jonathan Marsh, Marton Nagy and Norman Walsh, Xquery 1.0 and XPath 2.0 Data Model (XDM) (Second Edition), W3C Recommendation 14 December 2010. http://www.w3.org/TR/2010/REC-xpath-datamodel-20101214/. Viitattu 10.11.2011

[XML, 2008] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler and François Yergeau, Extensible Markup Language (XML) 1.0 W3C Recommendation 26 November 2008 http://www.w3.org/TR/REC-xml/ viitattu 03.11.2011

[XPath, 1999] James Clark and Steve DeRose. XML Path Language (XPath) Version 1.0. W3C Recommendation 16 November 1999, http://www.w3.org/TR/xpath/ viitattu 03.11.2011

[XQuery, 2010] Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Johathan Robie and Jérôme Siméon, XQuery 1.0: An XML Query Language (Second Edition) W3C Recommendation 14 December 2010, http://www.w3.org/TR/xquery/ viitattu 03.11.2011

[Yoshikawa *et al.*, 2001] Masatoshi Yoshikawa, Toshiyuki Amagasa, Takeyuki Shimura and Shunsuke Uemura, XRel: a path-based approach to storage and retrieval of XML documents using relational databases. *TOIT* **1**, 1 (Aug. 2001), 110-141.

[Zhang *et al.*, 2001] Chun Zhang, Jeffrey F. Naughton, David DeWitt, Qiong Luo and Guy Lohman, On supporting containment queries in relational database management systems. In: *Proc. of SIGMOD* (2001), 425-436.

Liite 1

Valtiotietokanta relaatiotietokannan ilmentyminä

valtio

nimi	valtiomuoto	väkiluku
Suomi	tasavalta	5 391 699
Ruotsi	perustuslaillinen_monarkia	9 408 028

kaupunki

nimi	väkiluku	valtio_nimi
Helsinki	591 892	Suomi
Tampere	213 645	Suomi
Tukholma	810 120	Ruotsi
Göteborg	500 197	Ruotsi

järvi

nimi	pinta-ala
Näsijärvi	254 640
Torniojärvi	330 000

joki

nimi	pituus	laskujoki
Torniojoki	510	NULL
Muoniojoki	230	Torniojoki

sijaitsee

valtio_nimi	järvi_nimi
Suomi	Näsijärvi
Ruotsi	Torniojärvi

virtaa

valtio_nimi	joki_nimi
Ruotsi	Torniojoki
Suomi	Torniojoki
Ruotsi	Muoniojoki
Suomi	Muoniojoki

liittyy

joki_nimi	järvi_nimi
Torniojoki	Torniojärvi

Liite 2

Valtiotietokanta XML-dokumentin esiintyminä

```
<valtio>
  <nimi>Suomi</nimi>
  <valtiomuoto>tasavalta</valtiomuoto>
  <väkiluku>5 391 699</väkiluku>
  <kaupunki>
    <nimi>Helsinki</nimi>
    <väkiluku>591 892</väkiluku>
  </kaupunki>
  <kaupunki>
    <nimi>Tampere</nimi>
    <väkiluku>213 645</väkiluku>
  </kaupunki>
  <järvi>
    <nimi>Näsijärvi</nimi>
    <pinta-ala>254 640</pinta-ala>
  </järvi>
  <joki>
    <nimi>Torniojoki</nimi>
    <pituus>510</pituus>
    <liittyy>Torniojärvi</liittyy>
  </joki>
  <joki>
    <nimi>Muoniojoki</nimi>
    <pituus>230</pituus>
    <yhtyy>Torniojoki</yhtyy>
  </joki>
</valtio>

<valtio>
  <nimi>Ruotsi</nimi>
  <valtiomuoto>perustuslaillinen monarkia</valtiomuoto>
  <väkiluku>9 408 028</väkiluku>
  <kaupunki>
    <nimi>Tukholma</nimi>
    <väkiluku>810 120</väkiluku>
  </kaupunki>
```

```
  <kaupunki>
    <nimi>Göteborg</nimi>
    <väkiluku>500 197</väkiluku>
  </kaupunki>
  <järvi>
    <nimi>Torniojärvi</nimi>
    <pinta-ala>330 000</pinta-ala>
  </järvi>
  <joki>
    <nimi>Torniojoki</nimi>
    <pituus>510</pituus>
    <liittyy>Torniojärvi</liittyy>
  </joki>
  <joki>
    <nimi>Muoniojoki</nimi>
    <pituus>230</pituus>
    <yhtyy>Torniojoki</yhtyy>
  </joki>
</valtio>
```

LIITE 3

Jäsennysesimerkkejä XIL-kyselyiden kääntämisestä SQL-kyselyiksi luvussa 7 esitettyjen attribuuttikielioppien avulla.

(1)

```
SELECT nimi

p1 S = SELECT nimi
p2 C = SELECT nimi
p6 TP = nimi
p8 P = nimi
p10 FP = nimi
p14 PP = nimi
p15 EN = nimi
p24 pt(EN) = {⟨nimi⟩}
p15 pt(PP) = pt(EN)
p14 pt(FP) = pt(PP)
p10 pt(P) = pt(FP)
p8 pt(TP) = pt(P)
p6 wt(C) = ⟨pt(TP) ⟩
p2 ht(S) = ⟨⟨wt(C), ⟨⟩⟩ = ⟨⟨⟨{⟨nimi⟩}⟩, ⟨⟩⟩
p1 ht(Q) = ht(S) = ⟨⟨⟨{⟨nimi⟩}⟩, ⟨⟩⟩

⟨⟨⟨{⟨nimi⟩}⟩, ⟨⟩⟩

p1 H = ⟨⟨⟨{⟨nimi⟩}⟩, ⟨⟩⟩
p2 L = ⟨⟨{⟨nimi⟩}⟩, ⟨⟩⟩, c(L) = 2
p7 W = ⟨{⟨nimi⟩}⟩, c(W) = c(L)
p8 Ps = {⟨nimi⟩}, c(Ps) = c(W)
p12 P = ⟨nimi⟩, c(P) = c(Ps)
p15 Pc = ⟨nimi⟩, t(Pc) = Tables, c(Pc) = c(P)
p16 EN = nimi, t(EN) = t(Pc), c(EN) = t(Pc)
p24 qs(EN) = ⟨⟨⟨⟨2,joki⟩,nimi⟩, ⟨⟨4,järvi⟩,nimi⟩, ⟨⟨6,valtio⟩,nimi⟩, ⟨⟨7,kaupunki⟩,nimi⟩⟩, ⟨⟩, ⟨⟩,
⟨⟩⟩⟩
p16 qs(Pc) = qs(EN)
p15 qs(P) = qs(Pc)
p12 qss(Ps) = ⟨qs(P)⟩
p8 qss(W) = qss(Ps)
p7 qss(L) = qss(W)
p2 qss(H) = ⟨⟨⟨⟨⟨2,joki⟩,nimi⟩, ⟨⟨4,järvi⟩,nimi⟩, ⟨⟨6,valtio⟩,nimi⟩, ⟨⟨7,kaupunki⟩,nimi⟩⟩, ⟨⟩, ⟨⟩,
⟨⟩⟩⟩⟩

p1 string(S) =
```

```
SELECT t1.jokinimi, t1.järvinimi, t1.valtionimi, t1.kaupunkinimi
FROM (SELECT joki.nimi AS jokinimi, järvi.nimi AS järvinimi, valtio.nimi AS valtionimi,
kaupunki.nimi AS kaupunkinimi
FROM joki LEFT JOIN liittyy ON joki.nimi = liittyy.joki_nimi LEFT JOIN järvi ON
liittyy.järvi_nimi = järvi.nimi LEFT JOIN virtaa ON joki.nimi = virtaa.joki_nimi LEFT
JOIN valtio ON virtaa.valtio_nimi = valtio.nimi LEFT JOIN kaupunki ON valtio.nimi =
kaupunki.valtio_nimi) AS t1
WHERE (t1.jokinimi IS NOT NULL OR t1.järvinimi IS NOT NULL OR t1.valtionimi IS NOT NULL
OR t1.kaupunkinimi IS NOT NULL) ORDER BY t1.jokinimi

SELECT t1.järvinimi, t1.jokinimi, t1.valtionimi, t1.kaupunkinimi
FROM (SELECT järvi.nimi AS järvinimi, joki.nimi AS jokinimi, valtio.nimi AS valtionimi,
kaupunki.nimi AS kaupunkinimi
FROM järvi LEFT JOIN liittyy ON järvi.nimi = liittyy.järvi_nimi LEFT JOIN joki ON
liittyy.joki_nimi = joki.nimi LEFT JOIN sijaitsee ON järvi.nimi = sijaitsee.järvi_nimi
LEFT JOIN valtio ON sijaitsee.valtio_nimi = valtio.nimi LEFT JOIN kaupunki ON valtio.nimi
= kaupunki.valtio_nimi) AS t1
WHERE (t1.järvinimi IS NOT NULL OR t1.jokinimi IS NOT NULL OR t1.valtionimi IS NOT NULL
OR t1.kaupunkinimi IS NOT NULL) ORDER BY t1.järvinimi

SELECT t1.valtionimi, t1.järvinimi, t1.jokinimi, t1.kaupunkinimi
FROM (SELECT valtio.nimi AS valtionimi, järvi.nimi AS järvinimi, joki.nimi AS jokinimi,
kaupunki.nimi AS kaupunkinimi
FROM valtio LEFT JOIN sijaitsee ON valtio.nimi = sijaitsee.valtio_nimi LEFT JOIN järvi ON
sijaitsee.järvi_nimi = järvi.nimi LEFT JOIN virtaa ON valtio.nimi = virtaa.valtio_nimi
LEFT JOIN joki ON virtaa.joki_nimi = joki.nimi LEFT JOIN kaupunki ON valtio.nimi =
kaupunki.valtio_nimi) AS t1
WHERE (t1.valtionimi IS NOT NULL OR t1.järvinimi IS NOT NULL OR t1.jokinimi IS NOT NULL
OR t1.kaupunkinimi IS NOT NULL) ORDER BY t1.valtionimi

SELECT t1.kaupunkinimi, t1.valtionimi, t1.järvinimi, t1.jokinimi
FROM (SELECT valtio.nimi AS valtionimi, järvi.nimi AS järvinimi, joki.nimi AS jokinimi,
kaupunki.nimi AS kaupunkinimi
FROM kaupunki LEFT JOIN valtio ON kaupunki.valtio_nimi = valtio.nimi LEFT JOIN sijaitsee
ON valtio.nimi = sijaitsee.valtio_nimi LEFT JOIN järvi ON sijaitsee.järvi_nimi =
järvi.nimi LEFT JOIN virtaa ON valtio.nimi = virtaa.valtio_nimi LEFT JOIN joki ON
virtaa.joki_nimi = joki.nimi) AS t1
WHERE (t1.kaupunkinimi IS NOT NULL OR t1.valtionimi IS NOT NULL OR t1.järvinimi IS NOT
NULL OR t1.jokinimi IS NOT NULL) ORDER BY t1.kaupunkinimi
```

HUOM. Koska kyselyyn ei jäsennysvaiheessa muodostunut hierarkiaa (yksikäsitteistä juurta), muodostetaan yhtä monta SQL-kyselyä, kuin SELECT-osaan liittyvistä tauluista on mahdollista generoida hierarkioita.

(2)

```
SELECT valtio/nimi

⟨⟨⟨{⟨valtio, nimi⟩}⟩, ⟨⟩⟩⟩

p1 H = ⟨⟨⟨{⟨valtio, nimi⟩}⟩, ⟨⟩⟩⟩
p2 L = ⟨⟨{⟨valtio, nimi⟩}⟩, ⟨⟩⟩, c(L) = 2
p7 W = ⟨{⟨valtio, nimi⟩}⟩, c(W) = c(L)
p8 Ps = {⟨valtio, nimi⟩}, c(Ps) = c(W)
p12 P = ⟨valtio, nimi⟩, c(P) = c(Ps)
p15 Pc = ⟨valtio, nimi⟩, t(Pc) = Tables, c(Pc) = c(P)
p18 EN = valtio, Pbc2 = ⟨nimi⟩, t(EN) = t(Pc1), t(Pc2) = Tables, c(EN) = 1, c(Pc2) =
c(Pc1)

      p22 qs(EN) = ⟨⟨⟨⟩, ⟨⟨⟨6,valtio⟩⟩, ⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩

      p16 EN = nimi, t(EN) = t(Pc), c(EN) = c(Pc)
      p24 qs(EN) = ⟨⟨⟨⟨⟨2,joki⟩,nimi⟩, ⟨⟨4,järvi⟩,nimi⟩, ⟨⟨6,valtio⟩,nimi⟩, ⟨⟨7,kaupunki⟩,nimi⟩⟩,
      ⟨⟩, ⟨⟩, ⟨⟩⟩⟩
      p16 qs(Pc) = qs(EN)

p18 qs(Pc1) = ⟨⟨⟨⟨⟨6,valtio⟩,nimi⟩⟩, ⟨⟨⟨6,valtio⟩, ⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩
p15 qs(P) = qs(Pc)
p12 qss(Ps) = ⟨qs(P)⟩ = ⟨⟨⟨⟨⟨⟨6,valtio⟩,nimi⟩⟩, ⟨⟨⟨6,valtio⟩, ⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩⟩
p8 qss(W) = qss(Ps)
p7 qss(L) = qss(W)
p2 qss(H) = ⟨⟨⟨⟨⟨⟨6,valtio⟩,nimi⟩⟩, ⟨⟨⟨6,valtio⟩, ⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩⟩

p1 string(S) =

SELECT t1.valtionimi
FROM
(SELECT valtio.nimi AS valtionimi FROM valtio) AS t1
WHERE (t1.valtionimi IS NOT NULL)
ORDER BY t1.valtionimi
```

(3)

```
SELECT valtio//nimi

p1 S = SELECT valtio//nimi
p2 C = SELECT valtio//nimi
p6 TP = valtio//nimi
p8 P = valtio//nimi
p10 FP = valtio//nimi
p14 PP = valtio//nimi
p17 EN = valtio, T = //nimi
     p24 pt(EN) = {⟨valtio⟩}

     p19 PP = nimi
     p15 EN = nimi
     p24 pt(EN) = {⟨nimi⟩}
     p15 pt(PP) = pt(EN)
     p19 pt(T) = {⟨//, nimi⟩}
p17 pt(PP) = {⟨valtio, //, nimi⟩}
p14 pt(FP) = pt(PP)
p10 pt(P) = pt(FP)
p8 pt(TP) = pt(P)
p6 wt(C) = ⟨pt(TP) ⟩ = ⟨{⟨valtio, //, nimi⟩}⟩
p2 ht(S) = ⟨⟨wt(C), ⟨⟩⟩ = ⟨⟨⟨{⟨valtio, //, nimi⟩}⟩, ⟨⟩⟩
p1 ht(Q) = ht(S) = ⟨⟨⟨⟨{⟨valtio, //, nimi⟩}⟩, ⟨⟩⟩

⟨⟨⟨{⟨valtio, //, nimi⟩}⟩, ⟨⟩⟩

p1 H = ⟨⟨⟨{⟨valtio, //, nimi⟩}⟩, ⟨⟩⟩
p2 L = ⟨⟨{⟨valtio, //, nimi⟩}⟩, ⟨⟩⟩, c(L) = 2
p7 W = ⟨{⟨valtio, //, nimi⟩}⟩, c(W) = c(L)
p8 Ps = {⟨valtio, //, nimi⟩}, c(Ps) = c(W)
p12 P = ⟨valtio, //, nimi⟩, c(P) = c(Ps)
p15 Pc = ⟨valtio, //, nimi⟩, t(Pc) = Tables, c(Pc) = c(P)
p19 EN = valtio, Pbc2 = ⟨nimi⟩, t(EN) = t(Pc1), t(Pc2) = Tables, c(EN) = 1, c(Pc2) =
c(Pc1)

     p22 qs(EN) = ⟨⟨⟨⟩, ⟨⟨⟨6,valtio⟩, ⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩

     p16 EN = nimi, t(EN) = t(Pc), c(EN) = c(Pc)
     p24 qs(EN) = ⟨⟨⟨⟨⟨2,joki⟩,nimi⟩, ⟨⟨4,järvi⟩,nimi⟩, ⟨⟨6,valtio⟩,nimi⟩, ⟨⟨7,kaupunki⟩,nimi⟩⟩,
     ⟨⟩, ⟨⟩, ⟨⟩⟩
     p16 qs(Pc) = qs(EN)

p19 qs(Pc1) = ⟨⟨⟨⟨⟨2,joki⟩,nimi⟩, ⟨⟨4,järvi⟩,nimi⟩, ⟨⟨6,valtio⟩,nimi⟩, ⟨⟨7,kaupunki⟩,nimi⟩⟩,
⟨⟨⟨6,valtio⟩, ⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩
p15 qs(P) = qs(Pc)
```

p12 qss(Ps) = ⟨qs(P)⟩
p8 qss(W) = qss(Ps)
p7 qss(L) = qss(W)
p2 qss(H) = ⟨⟨⟨⟨⟨2,joki⟩,nimi⟩, ⟨⟨4,järvi⟩,nimi⟩, ⟨⟨6,valtio⟩,nimi⟩, ⟨⟨7,kaupunki⟩,nimi⟩⟩,
⟨⟨⟨6,valtio⟩, ⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩

p1 string(S) =

```
SELECT t1.valtionimi, t1.järvinimi, t1.jokinimi, t1.kaupunkinimi
FROM
(SELECT valtio.nimi AS valtionimi, järvi.nimi AS järvinimi, joki.nimi AS jokinimi,
kaupunki.nimi AS kaupunkinimi
FROM valtio LEFT JOIN sijaitsee ON valtio.nimi = sijaitsee.valtio_nimi LEFT JOIN järvi ON
sijaitsee.järvi_nimi = järvi.nimi LEFT JOIN virtaa ON valtio.nimi = virtaa.valtio_nimi
LEFT JOIN joki ON virtaa.joki_nimi = joki.nimi LEFT JOIN kaupunki ON valtio.nimi =
kaupunki.valtio_nimi) AS t1
WHERE (t1.valtionimi IS NOT NULL OR t1.järvinimi IS NOT NULL OR t1.jokinimi IS NOT NULL
OR t1.kaupunkinimi IS NOT NULL)
ORDER BY t1.valtionimi
```

(4)

```
SELECT nimi
FROM valtio

p1 S = SELECT nimi FROM valtio
p3 C = nimi FWS = FROM valtio
      p6 TP = nimi
      p8 P = nimi
      p10 FP = nimi
      p14 PP = nimi
      p15 EN = nimi
      p24 pt(EN) = {⟨nimi⟩}
      p15 pt(PP) = pt(EN)
      p14 pt(FP) = pt(PP)
      p10 pt(P) = pt(FP)
      p8 pt(TP) = pt(P)
      p6 wt(C) = ⟨pt(TP) ⟩ = ⟨{⟨nimi⟩}⟩

      p27 FW = FROM valtio
      p29 F = FROM valtio
      p31 P = valtio
      p10 FP = valtio
      p14 PP = valtio
      p15 EN = valtio
      p24 pt(EN) = {⟨valtio⟩}
      p15 pt(PP) = pt(EN)
      p14 pt(FP) = pt(PP)
      p10 pt(P) = pt(FP)
      p31 wt(F) = ⟨pt(P) ⟩ = ⟨{⟨valtio⟩}⟩
      p29 lt(FW) = ⟨wt(F), ⟨⟩⟩ = ⟨⟨{⟨valtio⟩}⟩, ⟨⟩⟩
      p27 ht(FWS) = ⟨lt(FW) ⟩ = ⟨⟨⟨{⟨valtio⟩}⟩, ⟨⟩⟩⟩
p3 ht(S) = ⟨⟨⟨{⟨valtio⟩}⟩, ⟨⟩⟩, ⟨⟨{⟨nimi⟩}⟩,⟨⟩⟩⟩
p1 ht(Q) = ht(S)

⟨ ⟨⟨{⟨valtio⟩}⟩, ⟨⟩⟩, ⟨⟨{⟨nimi⟩}⟩, ⟨⟩⟩ ⟩


p1 H = ⟨ ⟨⟨{⟨valtio⟩}⟩, ⟨⟩⟩, ⟨⟨{⟨nimi⟩}⟩, ⟨⟩⟩ ⟩
p3 Hu = ⟨⟨⟨{⟨valtio⟩}⟩, ⟨⟩⟩⟩, L = ⟨⟨{⟨nimi⟩}⟩, ⟨⟩⟩, c(Hu) = 1, c(L) = 2
      p4 L = ⟨⟨{⟨valtio⟩}⟩, ⟨⟩⟩, c(L) = c(Hu)
      p7 W = ⟨{⟨valtio⟩}⟩, c(W) = c(L)
      p8 Ps = {⟨valtio⟩}, c(Pc) = c(W)
      p12 P = ⟨valtio⟩, c(P) = c(Ps)
      p15 Pc = ⟨valtio⟩, t(Pc) = Tables, c(Pc) = c(P)
      p16 EN = valtio, t(EN) = t(Pc), c(EN) = c(Pc)
      p22 qs(EN) = ⟨⟨⟩, ⟨⟨⟨6,valtio⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩
```

```
      p16 qs(Pc) = qs(EN)
      p15 qs(P) = qs(Pc)
      p12 qss(Ps) = ⟨qs(P)⟩
      p8 qss(W) = qss(Ps)
      p7 qss(L) = qss(W)
      p4 qss(Hu) = ⟨⟨⟨⟨⟩, ⟨⟨⟨6,valtio⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩

      p7 W = ⟨{⟨nimi⟩}⟩, c(W) = c(L)
      p8 Ps = {⟨nimi⟩}, c(Ps) = c(W)
      p12 P = ⟨nimi⟩, c(P) = c(Ps)
      p15 Pc = ⟨nimi⟩, t(Pc) = Tables, c(Pc) = c(P)
      p16 EN = nimi, t(EN) = t(Pc), c(EN) = c(Pc)
      p24 qs(EN) = ⟨⟨⟨⟨⟨2,joki⟩,nimi⟩, ⟨⟨4,järvi⟩,nimi⟩,⟨⟨6,valtio⟩,nimi⟩,⟨⟨7,kaupunki⟩,nimi⟩⟩, ⟨⟩,
      ⟨⟩, ⟨⟩⟩⟩
      p16 qs(Pc) = qs(EN)
      p15 qs(P) = qs(Pc)
      p12 qss(Ps) = ⟨qs(P)⟩
      p8 qss(W) = qss(Ps)
      p7 qss(L) = qss(W)
p3 qss(H) = ⟨⟨⟨⟨⟨⟨2,joki⟩,nimi⟩, ⟨⟨4,järvi⟩,nimi⟩,⟨⟨6,valtio⟩,nimi⟩,⟨⟨7,kaupunki⟩,nimi⟩⟩,
⟨⟨⟨6,valtio⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩

p1 string(S) =

SELECT t1.valtionimi, t1.järvinimi, t1.jokinimi, t1.kaupunkinimi
FROM
(SELECT valtio.nimi AS valtionimi, järvi.nimi AS järvinimi, joki.nimi AS jokinimi,
kaupunki.nimi AS kaupunkinimi
FROM valtio LEFT JOIN sijaitsee ON valtio.nimi = sijaitsee.valtio_nimi LEFT JOIN järvi ON
sijaitsee.järvi_nimi = järvi.nimi LEFT JOIN virtaa ON valtio.nimi = virtaa.valtio_nimi
LEFT JOIN joki ON virtaa.joki_nimi = joki.nimi LEFT JOIN kaupunki ON valtio.nimi =
kaupunki.valtio_nimi) AS t1
WHERE (t1.valtionimi IS NOT NULL OR t1.järvinimi IS NOT NULL OR t1.jokinimi IS NOT NULL
OR t1.kaupunkinimi IS NOT NULL)
ORDER BY t1.valtionimi
```

HUOM. Tämän kohdan (4) ja kohdan 3 XIL-kyselyt vastaavat semanttisesti toisiaan ja niistä muodostuneet SQL-kyselyt vastaavat toisiaan sekä syntaksiltaan että semantiikaltaan.

(5)

```
SELECT nimi
FROM joki
FROM valtio
⟨ ⟨⟨{⟨valtio⟩}⟩, ⟨⟩⟩, ⟨⟨{⟨joki⟩}⟩, ⟨⟩⟩, ⟨⟨{⟨nimi⟩}⟩, ⟨⟩⟩ ⟩

p1 H = ⟨ ⟨⟨{⟨valtio⟩}⟩, ⟨⟩⟩, ⟨⟨{⟨joki⟩}⟩, ⟨⟩⟩, ⟨⟨{⟨nimi⟩}⟩, ⟨⟩⟩ ⟩
p3 Hu = ⟨ ⟨⟨{⟨valtio⟩}⟩, ⟨⟩⟩, ⟨⟨{⟨joki⟩}⟩, ⟨⟩⟩⟩, L = ⟨⟨{⟨nimi⟩}⟩, ⟨⟩⟩, c(Hu) = 1, c(L) = 2

      p5 Hu2 = ⟨ ⟨⟨{⟨valtio⟩}⟩, ⟨⟩⟩⟩, L = ⟨⟨{⟨joki⟩}⟩, ⟨⟩⟩, c(Hu2) = c(Hu1), c(L) = c(Hu1)
            p4 L = ⟨⟨{⟨valtio⟩}⟩, ⟨⟩⟩, c(L) = c(Hu)
            p7 W = ⟨{⟨valtio⟩}⟩, c(W) = c(L)
            p8 Ps = {⟨valtio⟩}, c(Ps) = c(W)
            p12 P = ⟨valtio⟩, c(P) = c(Ps)
            p15 Pc = ⟨valtio⟩, t(Pc) = Tables, c(Pc) = c(P)
            p16 EN = valtio, t(EN) = t(Pc), c(EN) = c(Pc)
            p22 qs(EN) = ⟨⟨⟨⟩, ⟨⟨⟨6,valtio⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩
            p16 qs(Ptc) = qs(EN)
            p15 qs(P) = qs(Pc)
            p12 qss(Ps) = ⟨qs(P)⟩
            p8 qss(W) = qss(Ps)
            p7 qss(L) = qss(W)
            p4 qss(Hu) = ⟨⟨⟨⟩, ⟨⟨⟨6,valtio⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩⟩

            p7 W = ⟨{⟨joki⟩}⟩, c(W) = c(L)
            p8 Ps = {⟨joki⟩}, c(Ps) = c(W)
            p12 P = ⟨joki⟩, c(P) = c(Ps)
            p15 Pc = ⟨joki⟩, t(Pc) = Tables, c(Pc) = c(P)
            p16 EN = joki, t(EN) = t(Pc), c(EN) = c(Pc)
            p22 qs(EN) = ⟨⟨⟨⟩, ⟨⟨⟨2,joki⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩
            p16 qs(Pc) = qs(EN)
            p15 qs(P) = qs(Pc)
            p12 qss(Ps) = ⟨qs(P)⟩
            p8 qss(W) = qss(Ps)
            p7 qss(L) = ⟨⟨⟨⟩, ⟨⟨⟨2,joki⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩⟩
      p5 qss(Hu1) = ⟨⟨⟨⟩, ⟨⟨⟨6,valtio⟩,⟨1,virtaa⟩⟩, ⟨⟨1,virtaa⟩,⟨2,joki⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩⟩

      p7 W = ⟨{⟨nimi⟩}⟩, c(W) = c(L)
      p8 Ps = {⟨nimi⟩}, c(Ps) = c(W)
      p12 P = ⟨nimi⟩, c(P) = c(Ps)
      p15 Pc = ⟨nimi⟩, t(Pc) = Tables, c(Pc) = c(P)
      p16 EN = nimi, t(EN) = t(Pc), c(EN) = c(Pc)
      p24 qs(EN) = ⟨⟨⟨⟨⟨2,joki⟩,nimi⟩, ⟨⟨4,järvi⟩,nimi⟩,⟨⟨6,valtio⟩,nimi⟩,⟨⟨7,kaupunki⟩,nimi⟩⟩, ⟨⟩,
      ⟨⟩, ⟨⟩⟩⟩
      p16 qs(Pc) = qs(EN)
      p15 qs(P) = qs(Pc)
```

p12 qss(Ps) = ⟨qs(P)⟩
p8 qss(W) = qss(Ps)
p7 qss(L) = qss(W)

p3 qss(H) = ⟨⟨⟨⟨⟨2,joki⟩,nimi⟩, ⟨⟨4,järvi⟩,nimi⟩⟩, ⟨⟨⟨6,valtio⟩,⟨1,virtaa⟩⟩,
⟨⟨1,virtaa⟩,⟨2,joki⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩⟩

p1 string(S) =

```
SELECT t1.valtionimi, t1.jokinimi, t1.järvinimi
FROM
(SELECT valtio.nimi AS valtionimi, joki.nimi AS jokinimi, järvi.nimi AS järvinimi
FROM valtio LEFT JOIN virtaa ON valtio.nimi = virtaa.valtio_nimi LEFT JOIN joki On
virtaa.joki_nimi = joki.nimi LEFT JOIN liittyy ON joki.nimi = liittyy.joki_nimi LEFT JOIN
järvi ON liittyy.järvi_nimi = järvi.nimi) AS t1
WHERE (t1.jokinimi IS NOT NULL OR t1.järvinimi IS NOT NULL)
ORDER BY t1.valtionimi
```

(6)

```
SELECT nimi
FROM järvi
FROM joki
FROM valtio
⟨ ⟨⟨{⟨valtio⟩}⟩, ⟨⟩⟩, ⟨⟨{⟨joki⟩}⟩, ⟨⟩⟩, ⟨⟨{⟨järvi⟩}⟩, ⟨⟩⟩, ⟨⟨{⟨nimi⟩}⟩, ⟨⟩⟩ ⟩

p1 H = ⟨⟨⟨{⟨valtio⟩}⟩, ⟨⟩⟩, ⟨⟨{⟨joki⟩}⟩, ⟨⟩⟩, ⟨⟨{⟨järvi⟩}⟩, ⟨⟩⟩, ⟨⟨{⟨nimi⟩}⟩, ⟨⟩⟩⟩
p3 Hu = ⟨⟨⟨{⟨valtio⟩}⟩, ⟨⟩⟩, ⟨⟨{⟨joki⟩}⟩, ⟨⟩⟩, ⟨⟨{⟨järvi⟩}⟩, ⟨⟩⟩⟩, L = ⟨⟨{⟨nimi⟩}⟩, ⟨⟩⟩, c(Hu) = 1,
c(L) =2

      p5 Hu2 = ⟨⟨⟨{⟨valtio⟩}⟩, ⟨⟩⟩, ⟨⟨{⟨joki⟩}⟩, ⟨⟩⟩⟩, L = ⟨⟨{⟨järvi⟩}⟩, ⟨⟩⟩, c(Hu2) = c(Hu1), c(L)
      = c(Hu1)
            p5 Hu2 = ⟨⟨{⟨valtio⟩}⟩, ⟨⟩⟩, L = ⟨⟨{⟨joki⟩}⟩, ⟨⟩⟩, c(Hu2) = c(Hu1), c(L) = c(Hu1)
                  p4 L = ⟨⟨{⟨valtio⟩}⟩, ⟨⟩⟩, c(L) = c(Hu)
                  p7 W = ⟨{⟨valtio⟩}⟩, c(W) = c(L)
                  p8 Ps = {⟨valtio⟩}, c(Ps) = c(W)
                  p12 P = ⟨valtio⟩, c(P) = c(Ps)
                  p15 Pc = ⟨valtio⟩, t(Pc) = Tables, c(Pc) = c(P)
                  p16 EN = valtio, t(EN) = t(Pc), c(EN) = c(Pc)
                  p22 qs(EN) = ⟨⟨⟩, ⟨⟨⟨6,valtio⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩
                  p16 qs(Pc) = qs(EN)
                  p15 qs(P) = qs(Pc)
                  p12 qss(Ps) = ⟨qs(P)⟩
                  p8 qss(W) = qss(Ps)
                  p7 qss(L) = qss(W)
                  p4 qss(Hu) = ⟨⟨⟨⟩, ⟨⟨⟨6,valtio⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩

                  p7 W = ⟨{⟨joki⟩}⟩, c(W) = c(L)
                  p8 Ps = {⟨joki⟩}, c(Ps) = c(W)
                  p12 P = ⟨joki⟩, c(P) = c(Ps)
                  p15 Pc = ⟨joki⟩, t(Pc) = Tables, c(Pc) = c(P)
                  p16 EN = joki, t(EN) = t(Pc), c(EN) = c(Pc)
                  p22 qs(EN) = ⟨⟨⟩, ⟨⟨⟨2,joki⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩
                  p16 qs(Pc) = qs(EN)
                  p15 qs(P) = qs(Pc)
                  p12 qss(Ps) = ⟨qs(P)⟩
                  p8 qss(W) = qss(Ps)
                  p7 qss(L) = qss(W)
            p5 qss(Hu1) = ⟨⟨⟨⟩, ⟨⟨⟨6,valtio⟩,⟨1,virtaa⟩⟩,⟨⟨1,virtaa⟩,⟨2,joki⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩

            p7 W = ⟨{⟨järvi⟩}⟩, c(W) = c(L)
            p8 Ps = {⟨järvi⟩}, c(Ps) = c(W)
            p12 P = ⟨järvi⟩, c(P) = c(Ps)
            p15 Pc = ⟨järvi⟩, t(Pc) = Tables, c(Pc) = c(P)
            p16 EN = järvi, t(EN) = t(Pc), c(EN) = c(Pc)
```

p22 qs(EN) = ⟨⟨⟩, ⟨⟨⟨4,järvi⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩
p16 qs(Pc) = qs(EN)
p15 qs(P) = qs(Pc)
p12 qss(Ps) = ⟨qs(P)⟩
p8 qss(W) = qss(Ps)
p7 qss(L) = qss(W)

p5 qss(Hu1) = ⟨⟨⟨⟩, ⟨⟨⟨6,valtio⟩,⟨1,virtaa⟩⟩, ⟨⟨1,virtaa⟩,⟨2,joki⟩⟩,
⟨⟨3,liittyy⟩,⟨4,järvi⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩⟩

p7 W = ⟨{⟨nimi⟩}⟩, c(W) = c(L)
p8 Ps = {⟨nimi⟩}, c(Ps) = c(W)
p12 P = ⟨nimi⟩, c(P) = c(Ps)
p15 Pc = ⟨nimi⟩, t(Pc) = Tables, c(Pc) = c(P)
p16 EN = nimi, t(EN) = t(Pc), c(EN) = c(Pc)
p24 qs(EN) = ⟨⟨⟨⟨⟨2,joki⟩,nimi⟩, ⟨⟨4,järvi⟩,nimi⟩,⟨⟨6,valtio⟩,nimi⟩,⟨⟨7,kaupunki⟩,nimi⟩⟩, ⟨⟩,
⟨⟩, ⟨⟩⟩⟩
p16 qs(Pc) = qs(EN)
p15 qs(P) = qs(Pc)
p12 qss(Ps) = ⟨qs(P)⟩
p8 qss(W) = qss(Ps)
p7 qss(L) = qss(W)

p3 qss(H) = ⟨⟨⟨⟨⟨4,järvi⟩,nimi⟩⟩, ⟨⟨⟨6,valtio⟩,⟨1,virtaa⟩⟩, ⟨⟨1,virtaa⟩,⟨2,joki⟩⟩,
⟨⟨3,liittyy⟩,⟨4,järvi⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩⟩

```
p1 string(S) =
SELECT t1.valtionimi, t1.järvinimi
FROM
(SELECT valtio.nimi AS valtionimi, järvi.nimi AS järvinimi
FROM valtio LEFT JOIN virtaa ON valtio.nimi = virtaa.valtio_nimi LEFT JOIN joki ON
virtaa.joki_nimi = joki.nimi LEFT JOIN liittyy ON joki.nimi = liittyy.joki_nimi LEFT JOIN
järvi ON liittyy.järvi_nimi = järvi.nimi) AS t1
WHERE (t1.järvinimi IS NOT NULL) ORDER BY t1.valtionimi
```

(7)
SELECT nimi
FROM valtio
WHERE järvi/nimi = Näsijärvi
⟨ ⟨⟨{⟨valtio⟩}⟩, ⟨{⟨järvi, nimi = Näsijärvi⟩}⟩⟩, ⟨⟨{⟨nimi⟩}⟩, ⟨⟩⟩ ⟩

p1 H = ⟨ ⟨⟨{⟨valtio⟩}⟩, ⟨{⟨järvi, nimi = Näsijärvi⟩}⟩⟩, ⟨⟨{⟨nimi⟩}⟩, ⟨⟩⟩ ⟩
p3 Hu = ⟨⟨⟨{⟨valtio⟩}⟩, ⟨{⟨järvi, nimi = Näsijärvi⟩}⟩⟩⟩, L = ⟨⟨{⟨nimi⟩}⟩, ⟨⟩⟩, c(Hu) = 1, c(L) = 2

p4 L = ⟨⟨{⟨valtio⟩}⟩, ⟨{⟨järvi, nimi = Näsijärvi⟩}⟩⟩, c(L) = c(Hu)
p6 W = ⟨{⟨valtio⟩}⟩, I = ⟨{⟨järvi, nimi = Näsijärvi⟩}⟩, c(W) = c(L), c(I) = 3
p8 Ps = {⟨valtio⟩}, c(Ps) = c(W)
p12 P = ⟨valtio⟩, c(P) = c(Ps)
p15 Pc = ⟨valtio⟩, t(Pc) = Tables, c(Pc) = c(P)
p16 EN = valtio, t(EN) = t(Pc), c(EN) = c(Pc)
p22 qs(EN) = ⟨⟨⟩, ⟨⟨⟨6,valtio⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩
p16 qs(Pc) = qs(EN)
p15 qs(P) = qs(Pc)
p12 qss(Ps) = ⟨qs(P)⟩
p8 qss(W) = qss(Ps)

p10 Ps = {⟨järvi, nimi = Näsijärvi⟩}, c(Ps) = c(I)
p12 P = ⟨järvi, nimi = Näsijärvi⟩, c(P) = c(Ps)
p15 Pc = ⟨järvi, nimi = Näsijärvi⟩, t(Pc) = Tables, c(Pc) = c(P)
p18 EN = järvi, Pc2 = ⟨nimi = Näsijärvi⟩, t(EN) = t(Pc1), t(Pc2) = Tables, c(EN) = 1, c(Pc2) = c(Pc1)

p22 qs(EN) = ⟨⟨⟩, ⟨⟨⟨4,järvi⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩

p16 EN = ’nimi = Näsijärvi’, t(EN) = t(Pc), c(EN) = c(Pc)
p25 qs(EN) = ⟨⟨⟩, ⟨⟩, ⟨⟨⟨⟨2,joki⟩,nimi,= Näsijärvi⟩, ⟨⟨4,järvi⟩,nimi,= Näsijärvi⟩, ⟨⟨6,valtio⟩,nimi,= Näsijärvi⟩, ⟨⟨7,kaupunki⟩,nimi,= Näsijärvi⟩⟩⟩, ⟨⟩⟩⟩
p16 qs(Pc) = qs(EN)

p18 qs(Pc) = ⟨⟨⟩, ⟨⟨⟨4,järvi⟩,⟨⟩⟩⟩, ⟨⟨⟨⟨4,järvi⟩,nimi,= Näsijärvi⟩⟩⟩, ⟨⟩⟩⟩
p15 qs(P) = qs(Pc)
p12 qss(Ps) = ⟨qs(P)⟩
p10 qss(I) = qss(Ps)
p6 qss(L) = ⟨⟨⟨⟨⟩, ⟨⟨⟨6,valtio⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟨⟨{⟨⟨6,valtio⟩,nimi⟩}, ⟨⟨⟨4,järvi ⟩,⟨⟩⟩⟩, ⟨⟨⟨⟨4,järvi⟩,nimi,= Näsijärvi⟩⟩⟩⟩⟩ ⟩⟩⟩⟩
p4 qss(Hu) = ⟨⟨⟨⟨⟩, ⟨⟨⟨6,valtio⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟨⟨{⟨⟨6,valtio⟩,nimi⟩}, ⟨⟨⟨6,valtio⟩,⟨5,sijaitsee⟩⟩, ⟨⟨5,sijaitsee⟩,⟨4,järvi⟩⟩⟩, ⟨⟨⟨⟨4,järvi⟩,nimi,= Näsijärvi⟩⟩⟩⟩⟩ ⟩⟩⟩⟩

p7 W = ⟨{⟨nimi⟩}⟩, c(W) = c(L)

```
p8 Ps = {⟨nimi⟩}, c(Ps) = c(W)
p12 P = ⟨nimi⟩, c(P) = c(Ps)
p15 Pc = ⟨nimi⟩, t(Pc) = Tables, c(Pc) = c(P)
p16 EN = nimi, t(EN) = t(Pc), c(EN) = c(Pc)
p24 qs(EN) = ⟨⟨⟨⟨⟨2,joki⟩,nimi⟩, ⟨⟨4,järvi⟩,nimi⟩,⟨⟨6,valtio⟩,nimi⟩,⟨⟨7,kaupunki⟩,nimi⟩⟩, ⟨⟩,
⟨⟩, ⟨⟩⟩⟩
p16 qs(Pc) = qs(EN)
p15 qs(P) = qs(Pc)
p12 qss(Ps) = ⟨qs(P)⟩
p8 qss(W) = qss(Ps)
p7 qss(L) = qss(W)

p3 qss(H) = ⟨⟨⟨⟨⟨⟨2,joki⟩,nimi⟩, ⟨⟨4,järvi⟩,nimi⟩,⟨⟨6,valtio⟩,nimi⟩,⟨⟨7,kaupunki⟩,nimi⟩⟩,
⟨⟨⟨6,valtio⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟨⟨{⟨⟨6,valtio⟩,nimi⟩}, ⟨⟨⟨6,valtio⟩,⟨5,sijaitsee⟩⟩,
⟨⟨5,sijaitsee⟩,⟨4,järvi⟩⟩⟩,  ⟨⟨⟨⟨4,järvi⟩,nimi,= Näsijärvi⟩⟩⟩⟩⟩ ⟩⟩⟩⟩

p1 string(S) =

SELECT t1.valtionimi, t1.järvinimi, t1.jokinimi, t1.kaupunkinimi
FROM
(SELECT valtio.nimi AS valtionimi, järvi.nimi AS järvinimi, joki.nimi AS jokinimi,
kaupunki.nimi AS kaupunkinimi
FROM valtio LEFT JOIN sijaitsee ON valtio.nimi = sijaitsee.valtio_nimi LEFT JOIN järvi ON
sijaitsee.järvi_nimi = järvi.nimi LEFT JOIN virtaa ON valtio.nimi = virtaa.valtio_nimi
LEFT JOIN joki ON virtaa.joki_nimi = joki.nimi LEFT JOIN kaupunki ON valtio.nimi =
kaupunki.valtio_nimi
WHERE valtio.nimi IN (SELECT valtio.nimi FROM valtio LEFT JOIN sijaitsee ON valtio.nimi =
sijaitsee.valtio_nimi LEFT JOIN järvi ON sijaitsee.järvi_nimi = järvi.nimi WHERE
järvi.nimi = 'Näsijärvi')) AS t1
WHERE (t1.jokinimi IS NOT NULL OR t1.järvinimi IS NOT NULL OR t1.valtionimi IS NOT NULL
OR t1.kaupunkinimi IS NOT NULL)
ORDER BY t1.valtionimi
```

(8)

```
SELECT joki
WHERE nimi = Torniojoki
FROM järvi
WHERE nimi = Torniojärvi
FROM valtio
WHERE nimi = Ruotsi
```

⟨ ⟨⟨{⟨valtio⟩}⟩, ⟨{⟨nimi = Ruotsi⟩}⟩⟩, ⟨⟨{⟨järvi⟩}⟩, ⟨{⟨nimi = Torniojärvi⟩}⟩⟩, ⟨⟨{⟨joki⟩}⟩, ⟨{⟨nimi = Torniojoki⟩}⟩⟩ ⟩

p1 H = ⟨ ⟨⟨{⟨valtio⟩}⟩, ⟨{⟨nimi = Ruotsi⟩}⟩⟩, ⟨⟨{⟨järvi⟩}⟩, ⟨{⟨nimi = Torniojärvi⟩}⟩⟩, ⟨⟨{⟨joki⟩}⟩, ⟨{⟨nimi = Torniojoki⟩}⟩⟩ ⟩
p3 Hu = ⟨⟨⟨{⟨valtio⟩}⟩,⟨{⟨nimi = Ruotsi⟩}⟩⟩,⟨⟨{⟨järvi⟩}⟩,⟨{⟨nimi = Torniojärvi⟩}⟩⟩⟩, L = ⟨⟨{⟨joki⟩}⟩, ⟨{⟨nimi = Torniojoki⟩}⟩⟩, c(Hu) = 1, c(L) = 2

p5 Hu2 = ⟨⟨⟨{⟨valtio⟩}⟩,⟨{⟨nimi = Ruotsi⟩}⟩⟩⟩, L = ⟨⟨{⟨järvi⟩}⟩,⟨{⟨nimi = Torniojärvi⟩}⟩⟩, c(Hu2) = c(Hu1), c(L) = c(Hu1)

p4 L = ⟨⟨{⟨valtio⟩}⟩,⟨{⟨nimi = Ruotsi⟩}⟩⟩, c(L) = c(Hu)
p6 W = ⟨{⟨valtio⟩}⟩, I = ⟨{⟨nimi = Ruotsi⟩}⟩, c(W) = c(L), c(I) = 3
p8 Ps = {⟨valtio⟩}, c(Ps) = c(W)
p12 P = ⟨valtio⟩, c(P) = c(Ps)
p15 Pc = ⟨valtio⟩, t(Pc) = Tables, c(Pc) = c(P)
p16 EN = valtio, t(EN) = t(Pc), c(EN) = c(Pc)
p22 qs(EN) = ⟨⟨⟩, ⟨⟨⟨6,valtio⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩
p16 qs(Pc) = qs(EN)
p15 qs(P) = qs(Pc)
p12 qss(Ps) = ⟨qs(P)⟩
p8 qss(W) = qss(Ps)

p10 Ps = {⟨nimi = Ruotsi⟩}, c(Ps) = c(I)
p12 P = ⟨nimi = Ruotsi⟩, c(P) = c(Ps)
p15 Pc = ⟨nimi = Ruotsi⟩, t(Pic) = Tables, c(Pc) = c(P)
p16 EN = ’nimi = Ruotsi’, c(EN) = c(Pc)
p25 qs(EN) = ⟨⟨⟩, ⟨⟩, ⟨⟨⟨⟨2,joki⟩,nimi,= Ruotsi⟩, ⟨⟨4,järvi⟩,nimi,= Ruotsi⟩, ⟨⟨6,valtio⟩,nimi,= Ruotsi⟩, ⟨⟨7,kaupunki⟩,nimi,= Ruotsi⟩⟩⟩, ⟨⟩⟩⟩
p16 qs(Pc) = qs(EN)
p15 qs(P) = qs(Pc)
p12 qss(Ps) = qs(P)
p10 qss(I) = qss(Ps)
p6 qss(L) = ⟨⟨⟨⟩, ⟨⟨⟨6,valtio⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟨⟨{⟨⟨6,valtio⟩,nimi⟩}, ⟨⟩, ⟨⟨⟨⟨2,joki⟩,nimi,= Ruotsi⟩, ⟨⟨4,järvi⟩,nimi,= Ruotsi⟩, ⟨⟨6,valtio⟩,nimi,= Ruotsi⟩, ⟨⟨7,kaupunki⟩,nimi,= Ruotsi⟩⟩⟩⟩ ⟩⟩⟩

```
      p4 qss(Hu) = ⟨⟨⟨⟩, ⟨⟨⟨6,valtio⟩,⟨⟩⟩⟩, ⟨⟩,  ⟨⟨⟨{⟨⟨6,valtio⟩,nimi⟩}, ⟨⟨⟨6,valtio⟩,⟨⟩⟩⟩,
      ⟨⟨⟨⟨2,joki⟩,nimi,= Ruotsi⟩, ⟨⟨4,järvi⟩,nimi,= Ruotsi⟩, ⟨⟨6,valtio⟩,nimi,= Ruotsi⟩,
      ⟨⟨7,kaupunki⟩,nimi,= Ruotsi⟩⟩⟩⟩⟩ ⟩⟩⟩⟩

      p6 W = ⟨{⟨järvi⟩}⟩, I = ⟨{⟨nimi = Torniojärvi⟩}⟩, c(W) = c(L), c(I) = 3
            p8 Ps = {⟨järvi⟩}, c(Ps) = c(W)
            p12 P = ⟨järvi⟩, c(P) = c(Ps)
            p15 Pc = ⟨järvi⟩, t(Pc) = Tables, c(Pc) = c(P)
            p16 EN = järvi, t(EN) = t(Pc), c(EN) = c(Pc)
            p22 qs(EN) = ⟨⟨⟨⟩, ⟨⟨⟨4,järvi⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩
            p16 qs(Pc) = qs(EN)
            p15 qs(P) = qs(Pc)
            p12 qss(Ps) = ⟨qs(P)⟩
            p8 qss(W) = qss(Ps)

            p10 Ps = {⟨nimi = Torniojärvi⟩}, c(Ps) = c(I)
            p12 P = ⟨nimi = Torniojärvi⟩, c(P) = c(Ps)
            p15 Pc = ⟨nimi = Torniojärvi⟩, t(Pic) = Tables, c(Pc) = c(P)
            p16 EN = 'nimi = Torniojärvi', c(EN) = c(Pc)
            p25 qs(EN) = ⟨⟨⟨⟩, ⟨⟩, ⟨⟨⟨⟨2,joki⟩,nimi,= Torniojärvi⟩, ⟨⟨4,järvi⟩,nimi,=
            Torniojärvi⟩, ⟨⟨6,valtio⟩,nimi,= Torniojärvi⟩, ⟨⟨7,kaupunki⟩,nimi,=
            Torniojärvi⟩⟩⟩, ⟨⟩⟩⟩
            p16 qs(Pc) = qs(EN)
            p15 qs(P) = qs(Pc)
            p12 qss(Ps) = ⟨qs(P)⟩
            p10 qss(I) = qss(Ps)
      p6 qss(Lu) = ⟨⟨⟨⟩, ⟨⟨⟨4,järvi⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟨⟨{⟨⟨4,järvi⟩,nimi⟩}, ⟨⟩, ⟨⟨⟨⟨2,joki⟩,nimi,=
      Torniojärvi⟩, ⟨⟨4,järvi⟩,nimi,= Torniojärvi⟩, ⟨⟨6,valtio⟩,nimi,= Torniojärvi⟩,
      ⟨⟨7,kaupunki⟩,nimi,= Torniojärvi⟩⟩⟩⟩⟩ ⟩⟩⟩⟩

p5 qss(Hu1) = ⟨⟨⟨⟩, ⟨⟨⟨6,valtio⟩,⟨5,sijaitsee⟩⟩, ⟨⟨5,sijaitsee⟩,⟨4,järvi⟩⟩⟩, ⟨⟩,
⟨⟨⟨{⟨⟨6,valtio⟩,nimi⟩}, ⟨⟨⟨6,valtio⟩,⟨⟩⟩⟩, ⟨⟨⟨⟨2,joki⟩,nimi,= Ruotsi⟩, ⟨⟨4,järvi⟩,nimi,=
Ruotsi⟩, ⟨⟨6,valtio⟩,nimi,= Ruotsi⟩, ⟨⟨7,kaupunki⟩,nimi,= Ruotsi⟩⟩⟩⟩⟩,
⟨⟨{⟨⟨4,järvi⟩,nimi⟩}, ⟨⟨⟨6,valtio⟩,⟨5,sijaitsee⟩⟩, ⟨⟨5,sijaitsee⟩,⟨4,järvi⟩⟩⟩,
⟨⟨⟨⟨2,joki⟩,nimi,= Torniojärvi⟩, ⟨⟨4,järvi⟩,nimi,= Torniojärvi⟩⟩⟩⟩⟩ ⟩⟩⟩⟩

p6 W = ⟨{⟨joki⟩}⟩, I = ⟨{⟨nimi = Torniojoki⟩}⟩, c(W) = c(L), c(I) = 3
      p8 Ps = {⟨joki⟩}, c(Ps) = c(W)
      p12 P = ⟨joki⟩, c(P) = c(Ps)
      p15 Pc = ⟨joki⟩, t(Pc) = Tables, c(Pc) = c(P)
      p16 EN = joki, t(EN) = t(Pc), c(EN) = c(Pc)
      p22 qs(EN) = ⟨⟨⟨⟩, ⟨⟨⟨2,joki⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩
      p16 qs(Pc) = qs(EN)
      p15 qs(P) = qs(Pc)
      p12 qss(Ps) = ⟨qs(P)⟩
      p8 qss(W) = qss(Ps)
```

p10 Ps = {⟨nimi = Torniojoki⟩}, c(Ps) = c(I)
p12 P = ⟨nimi = Torniojoki⟩, c(P) = c(Ps)
p15 Pc = ⟨nimi = Torniojoki⟩, t(Pic) = Tables, c(Pc) = c(P)
p16 EN = 'nimi = Torniojoki', c(EN) = c(Pc)
p25 qs(EN) = ⟨⟨⟨⟩, ⟨⟩, ⟨⟨⟨⟨2,joki⟩,nimi,= Torniojoki⟩, ⟨⟨4,järvi⟩,nimi,= Torniojoki⟩,
⟨⟨6,valtio⟩,nimi,= Torniojoki⟩, ⟨⟨7,kaupunki⟩,nimi,= Torniojoki⟩⟩⟩, ⟨⟩⟩⟩
p16 qs(Pc) = qs(EN)
p15 qs(P) = qs(Pc)
p12 qss(Ps) = ⟨qs(P)⟩
p10 qss(I) = qss(Ps)
p6 qss(Lu) = ⟨⟨⟨⟨⟩, ⟨⟨⟨2,joki⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟨⟨{⟨⟨2,joki⟩,nimi⟩}, ⟨⟩, ⟨⟨⟨⟨2,joki⟩,nimi,=
Torniojoki⟩, ⟨⟨4,järvi⟩,nimi,= Torniojoki⟩, ⟨⟨6,valtio⟩,nimi,= Torniojoki⟩,
⟨⟨7,kaupunki⟩,nimi,= Torniojoki⟩⟩⟩⟩⟩ ⟩⟩⟩⟩

p3 qss(H) = ⟨⟨⟨⟨⟩, ⟨⟨⟨6,valtio⟩,⟨5,sijaitsee⟩⟩, ⟨⟨5,sijaitsee⟩,⟨4,järvi⟩⟩, ⟨⟨4,järvi⟩,⟨3,liittyy⟩⟩,
⟨⟨3,liittyy⟩,⟨2,joki⟩⟩⟩, ⟨⟩, ⟨⟨⟨{⟨⟨⟨6,valtio⟩,nimi⟩}, ⟨⟨⟨6,valtio⟩,⟨⟩⟩⟩, ⟨⟨⟨⟨2,joki⟩,nimi,= Ruotsi⟩,
⟨⟨4,järvi⟩,nimi,= Ruotsi⟩, ⟨⟨6,valtio⟩,nimi,= Ruotsi⟩, ⟨⟨7,kaupunki⟩,nimi,= Ruotsi⟩⟩⟩⟩⟩,
⟨⟨{⟨⟨4,järvi⟩,nimi⟩}, ⟨⟨⟨6,valtio⟩,⟨5,sijaitsee⟩⟩, ⟨⟨5,sijaitsee⟩,⟨4,järvi⟩⟩⟩, ⟨⟨⟨⟨2,joki⟩,nimi,=
Torniojärvi⟩, ⟨⟨4,järvi⟩,nimi,= Torniojärvi⟩⟩⟩⟩⟩, ⟨⟨{⟨⟨2,joki⟩,nimi⟩}, ⟨⟨⟨6,valtio⟩,⟨5,sijaitsee⟩⟩, ⟨⟨5,sijaitsee⟩,⟨4,järvi⟩⟩, ⟨⟨4,järvi⟩,⟨3,liittyy⟩⟩,
⟨⟨3,liittyy⟩,⟨2,joki⟩⟩⟩, ⟨⟨⟨⟨2,joki⟩,nimi,= Torniojoki⟩⟩⟩⟩⟩ ⟩⟩⟩⟩

```
p1 string(S) =
SELECT t1.valtionimi, t1.jokinimi, t1.jokipituus, t1.jokilaskujoki
FROM
(SELECT valtio.nimi AS valtionimi, joki.nimi AS jokinimi, joki.pituus AS jokipituus,
joki.laskujoki AS jokilaskujoki
FROM valtio LEFT JOIN sijaitsee On valtio.nimi = sijaitsee.valtio_nimi LEFT JOIN järvi ON
sijaitsee.järvi_nimi = järvi.nimi LEFT JOIN liittyy ON järvi.nimi = liittyy.järvi_nimi
LEFT JOIN joki ON liittyy.joki_nimi = joki.nimi
WHERE valtio.nimi IN (SELECT valtio.nimi FROM valtio LEFT JOIN sijaitsee ON valtio.nimi =
sijaitsee.valtio_nimi LEFT JOIN järvi ON sijaitsee.järvi_nimi = järvi.nimi LEFT JOIN
virtaa ON valtio.nimi = virtaa.valtio_nimi LEFT JOIN joki ON virtaa.joki_nimi = joki.nimi
LEFT JOIN kaupunki ON valtio.nimi = kaupunki.valtio_nimi WHERE valtio.nimi = 'Ruotsi' OR
järvi.nimi = 'Ruotsi' OR joki.nimi = 'Ruotsi' OR kaupunki.nimi = 'Ruotsi') AND järvi.nimi
IN (SELECT järvi.nimi FROM valtio LEFT JOIN sijaitsee ON valtio.nimi =
sijaitsee.valtio_nimi LEFT JOIN järvi ON sijaitsee.järvi_nimi = järvi.nimi LEFT JOIN
liittyy ON järvi.nimi = liittyy.järvi_nimi LEFT JOIN joki ON liittyy.joki_nimi =
joki.nimi WHERE järvi.nimi = 'Torniojärvi' OR joki.nimi = 'Torniojärvi') AND joki.nimi IN
(SELECT joki.nimi FROM valtio LEFT JOIN sijaitsee ON valtio.nimi = sijaitsee.valtio_nimi
LEFT JOIN järvi ON sijaitsee.järvi_nimi = järvi.nimi LEFT JOIN liittyy ON järvi.nimi =
liittyy.järvi_nimi LEFT JOIN joki ON liittyy.joki_nimi = joki.nimi WHERE joki.nimi =
'Torniojoki')) AS t1
WHERE (t1.jokinimi IS NOT NULL OR t1.jokipituus IS NOT NULL OR t1.jokilaskujoki IS NOT
NULL)ORDER BY t1.valtionimi
```

(9)

```
SELECT joki/nimi, pinta-ala|pinta-alue
FROM valtio//järvi
WHERE nimi = Torniojärvi OR pinta-ala > 5000

⟨⟨⟨{⟨valtio, //, järvi⟩}⟩, ⟨{⟨nimi=Torniojärvi⟩} OR {⟨pinta-ala > 5000⟩}⟩⟩,
 ⟨⟨{⟨joki, nimi⟩}, {⟨pinta-ala⟩, ⟨pinta-alue⟩}⟩, ⟨⟩⟩ )

p1 H = ⟨⟨⟨{⟨valtio, //, järvi⟩}⟩, ⟨{⟨nimi=Torniojärvi⟩} OR {⟨pinta-ala > 5000⟩}⟩⟩,
 ⟨⟨{⟨joki, nimi⟩}, {⟨pinta-ala⟩, ⟨pinta-alue⟩}⟩, ⟨⟩⟩ )
p3 Hu = ⟨⟨⟨{⟨valtio, //, järvi⟩}⟩, ⟨{⟨nimi=Torniojärvi⟩} OR {⟨pinta-ala > 5000⟩}⟩⟩⟩, L =
⟨⟨{⟨joki, nimi⟩}, {⟨pinta-ala⟩, ⟨pinta-alue⟩}⟩, ⟨⟩⟩, c(Hu) = 1, c(L) = 2
     p4 L = ⟨⟨{⟨valtio, //, järvi⟩}⟩, ⟨{⟨nimi=Torniojärvi⟩} OR {⟨pinta-ala > 5000⟩}⟩⟩, c(L) =
c(Hu)
     p6 W = ⟨{⟨valtio, //, järvi⟩}⟩, I = ⟨{⟨nimi=Torniojärvi⟩} OR {⟨pinta-ala > 5000⟩}⟩,
     c(W) = c(L), c(I) = 3

          p8 Ps = {⟨valtio, //, järvi⟩}, c(Ps) = c(W)
          p12 P = ⟨valtio, //, järvi⟩, c(P) = c(Ps)
          p15 Pc = ⟨valtio, //, järvi⟩, t(Pc) = Tables, c(Pc) = c(P)
          p19 EN = valtio, Pc2 = ⟨järvi⟩, t(EN) = t(Pc1), t(Pc2) = Tables, c(EN) = 1,
          c(Pc2) = c(Pc1)

               p22 qs(EN) = ⟨⟨⟩, ⟨⟨⟨6,valtio⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩

               p16 EN = järvi, t(EN) = t(Pc), c(EN) = c(Pc)
               p22 qs(EN) = ⟨⟨⟩, ⟨⟨⟨4,järvi⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩
               p16 qs(Pc) = qs(EN)

          p19 qs(Pc) = ⟨⟨⟩, ⟨⟨⟨6,valtio⟩,⟨⟩⟩, ⟨⟨4,järvi⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩
          p15 qs(P) = qs(Pc)
          p12 qss(Ps) = ⟨qs(Pc)⟩
          p8 qss(W) = qss(Ps)

          p11 Ps = {⟨nimi=Torniojärvi⟩}, I2 = ⟨{⟨pinta-ala > 5000⟩}⟩, c(Ps) = c(I1), c(I2)
          = c(I1)
               p12 P = ⟨nimi=Torniojärvi⟩, c(P) = c(Ps)
               p15 Pc = ⟨nimi=Torniojärvi⟩, t(Pc) = Tables, c(Pc) = c(P)
               p16 EN = ’ nimi=Torniojärvi’, c(EN) = c(Pc)
               p25 qs(EN) = ⟨⟨⟩, ⟨⟩, ⟨⟨⟨⟨2,joki⟩,nimi,= Torniojärvi⟩, ⟨⟨4,järvi⟩,nimi,=
               Torniojärvi⟩, ⟨⟨6,valtio⟩,nimi,= Torniojärvi⟩, ⟨⟨7,kaupunki⟩,nimi,=
               Torniojärvi⟩⟩⟩, ⟨⟩⟩⟩
               p16 qs(Pc) = qs(EN)
               p15 qs(P) = qs(Pc)
               p12 qss(Ps) = ⟨qs(P)⟩
```

p10 Ps = {⟨pinta-ala > 5000⟩}, c(Ps) = c(I)
p12 P = ⟨pinta-ala > 5000⟩, c(P) = c(Ps)
p15 Pc = ⟨pinta-ala > 5000⟩, t(Pc) = Tables, c(Pc) = c(P)
p16 EN = ’ pinta-ala > 5000’, c(EN) = c(Pc)
p25 qs(EN) = ⟨⟨⟨⟩, ⟨⟩, ⟨⟨⟨⟨4,järvi⟩, pinta-ala, > 5000⟩⟩⟩, ⟨⟩⟩⟩
p16 qs(Pc) = qs(EN)
p15 qs(P) = qs(Pc)
p12 qss(Ps) = ⟨qs(P)⟩
p10 qss(I) = qss(Ps)

p11 qss(I1) = ⟨⟨⟨⟩, ⟨⟩, ⟨⟨⟨⟨2,joki⟩,nimi,= Torniojärvi⟩, ⟨⟨4,järvi⟩,nimi,= Torniojärvi⟩, ⟨⟨6,valtio⟩,nimi,= Torniojärvi⟩, ⟨⟨7,kaupunki⟩,nimi,= Torniojärvi⟩⟩⟩, ⟨⟩⟩⟩, OR, ⟨⟨⟨⟩, ⟨⟩, ⟨⟨⟨⟨4,järvi⟩, pinta-ala, > 5000⟩⟩⟩, ⟨⟩⟩⟩⟩

p6 qss(L) = ⟨⟨⟨⟩, ⟨⟨⟨6,valtio⟩,⟨⟩⟩, ⟨⟨4,järvi⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟨⟨{⟨⟨4,järvi⟩,nimi⟩}, ⟨⟩, ⟨⟨⟨⟨2,joki⟩,nimi,= Torniojärvi⟩, ⟨⟨4,järvi⟩,nimi,= Torniojärvi⟩, ⟨⟨6,valtio⟩,nimi,= Torniojärvi⟩, ⟨⟨7,kaupunki⟩,nimi,= Torniojärvi⟩⟩⟩⟩, OR, ⟨{⟨⟨4,järvi⟩,nimi⟩}, ⟨⟩, ⟨⟨⟨⟨4,järvi⟩, pinta-ala, > 5000⟩⟩⟩⟩⟩ ⟩⟩⟩⟩

p4 qss(Hu) = ⟨⟨⟨⟩, ⟨⟨⟨6,valtio⟩,⟨5,sijaitsee⟩⟩, ⟨⟨5,sijaitsee⟩,⟨4,järvi⟩⟩⟩, ⟨⟩, ⟨⟨⟨{⟨⟨4,järvi⟩,nimi⟩}, ⟨⟨⟨6,valtio⟩,⟨5,sijaitsee⟩⟩, ⟨⟨5,sijaitsee⟩,⟨4,järvi⟩⟩⟩, ⟨⟨⟨⟨2,joki⟩,nimi,= Torniojärvi⟩, ⟨⟨4,järvi⟩,nimi,= Torniojärvi⟩⟩⟩⟩, OR, ⟨{⟨⟨4,järvi⟩,nimi⟩}, ⟨⟨⟨6,valtio⟩,⟨5,sijaitsee⟩⟩, ⟨⟨5,sijaitsee⟩,⟨4,järvi⟩⟩⟩, {⟨⟨4,järvi⟩, pinta-ala, > 5000⟩⟩⟩⟩⟩ ⟩⟩⟩⟩

p7 W = ⟨{⟨joki, nimi⟩}, {⟨pinta-ala⟩, ⟨pinta-alue⟩}⟩, c(W) = c(L)
p9 Ps = {⟨joki, nimi⟩}, W2 = ⟨{⟨pinta-ala⟩, ⟨pinta-alue⟩}⟩, c(Ps) = c(W1), c(W2)=c(W1)
p12 P = ⟨joki, nimi⟩, c(P) = c(Ps)
p15 Pc = ⟨joki, nimi⟩, t(Pbc) = Tables, c(Pc) = c(P)
p18 EN = joki, Pc2 = ⟨nimi⟩, t(EN) = t(Pc1), t(Pc1) = Tables, c(EN) = 1, c(Pc2) = c(Pc1)
p22 qs(EN) = ⟨⟨⟨⟩, ⟨⟨⟨2,joki⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩

p16 EN = nimi, t(EN) = t(Pc), c(EN) = c(Pc)
p24 qs(EN) = ⟨⟨⟨⟨⟨2,joki⟩,nimi⟩, ⟨⟨4,järvi⟩,nimi⟩, ⟨⟨6,valtio⟩,nimi⟩, ⟨⟨7,kaupunki⟩,nimi⟩⟩, ⟨⟩, ⟨⟩, ⟨⟩⟩⟩
p16 qs(Pc) = qs(EN)

p18 qs(Pc1) = ⟨⟨⟨⟨⟨2,joki⟩,nimi⟩⟩, ⟨⟨⟨2,joki⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩
p15 qs(P) = qs(Pc)
p12 qss(Ps) = ⟨qs(P)⟩

p8 Ps = {⟨pinta-ala⟩, ⟨pinta-alue⟩}, c(Ps) = c(W)
p13 P = ⟨pinta-ala⟩, Ps2 = {⟨pinta-alue⟩}, c(P) = c(Ps1), c(Ps2) = c(Ps1)
p15 Pc = ⟨pinta-ala⟩, t(Pc) = Tables, c(Pc) = c(P)
p16 EN = pinta-ala, t(En) = t(Pc), c(EN) = c(Pc)

```
                p24 qs(EN) = ⟨⟨⟨⟨⟨4,järvi⟩,pinta-ala⟩⟩, ⟨⟩,⟨⟩, ⟨⟩⟩⟩
                p16 qs(Pc) = qs(EN)
                p15 qs(P) = qs(Pc)

                p12 P = ⟨pinta-alue⟩, c(P) = c(Ps)
                p15 Pc = ⟨pinta-alue⟩, t(Pc) = Tables, c(Pc) = c(P)
                p16 EN = pinta-alue, t(EN) = t(Pc), c(EN) = c(Pc)
                p24 qs(EN) = ⟨⟨⟨⟩, ⟨⟩, ⟨⟩, ⟨⟩⟩⟩
                p16 qs(Pc) = qs(EN)
                p15 qs(P) = qs(Pc)
                p12 qss(Ps) = ⟨qs(P)⟩
           p13 qss(Ps1) = ⟨⟨⟨⟨⟨⟨4,järvi⟩,pinta-ala⟩⟩, ⟨⟩,⟨⟩, ⟨⟩⟩⟩⟩
           p8 qss(W) = qss(Ps)
      p9 qss(W1) = ⟨⟨⟨⟨⟨⟨2,joki⟩,nimi⟩⟩, ⟨⟨⟨2,joki⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩, ⟨⟨⟨⟨⟨4,järvi⟩,pinta-ala⟩⟩,⟨⟩,⟨⟩,⟨⟩⟩⟩⟩
      p7 qss(L) = qss(W)

p3 qss(H) =
⟨⟨⟨⟨⟨⟨2,joki⟩,nimi⟩⟩, ⟨⟨⟨6,valtio⟩,⟨5,sijaitsee⟩⟩, ⟨⟨5,sijaitsee⟩,⟨4,järvi⟩⟩,
⟨⟨4,järvi⟩,⟨3,liittyy⟩⟩, ⟨⟨3,liittyy⟩,⟨2,joki⟩⟩⟩, ⟨⟩, ⟨⟨⟨{⟨⟨4,järvi⟩,nimi⟩},
⟨⟨⟨6,valtio⟩,⟨5,sijaitsee⟩⟩, ⟨⟨5,sijaitsee⟩,⟨4,järvi⟩⟩⟩, ⟨⟨⟨⟨2,joki⟩,nimi,= Torniojärvi⟩,
⟨⟨4,järvi⟩,nimi,= Torniojärvi⟩⟩⟩⟩, OR, ⟨{⟨⟨4,järvi⟩,nimi⟩}, ⟨⟨⟨6,valtio⟩,⟨5,sijaitsee⟩⟩,
⟨⟨5,sijaitsee⟩,⟨4,järvi⟩⟩⟩, ⟨⟨⟨⟨4,järvi⟩, pinta-ala, > 5000⟩⟩⟩⟩⟩ ⟩⟩⟩,
⟨⟨⟨⟨⟨4,järvi⟩,pinta-ala⟩⟩, ⟨⟨⟨6,valtio⟩,⟨5,sijaitsee⟩⟩, ⟨⟨5,sijaitsee⟩,⟨4,järvi⟩⟩⟩, ⟨⟩,
⟨⟨⟨{⟨⟨4,järvi⟩,nimi⟩}, ⟨⟨⟨6,valtio⟩,⟨5,sijaitsee⟩⟩, ⟨⟨5,sijaitsee⟩,⟨4,järvi⟩⟩⟩, ⟨⟨⟨⟨2,joki⟩,nimi,=
Torniojärvi⟩, ⟨⟨4,järvi⟩,nimi,= Torniojärvi⟩⟩⟩⟩, OR, ⟨{⟨⟨4,järvi⟩,nimi⟩},
⟨⟨⟨6,valtio⟩,⟨5,sijaitsee⟩⟩, ⟨⟨5,sijaitsee⟩,⟨4,järvi⟩⟩⟩, ⟨⟨⟨⟨4,järvi⟩, pinta-ala, > 5000⟩⟩⟩⟩⟩ ⟩⟩⟩⟩


p1 string(S) =
SELECT t1.valtionimi, t1.jokinimi, t2.järvipinta-ala
FROM
(SELECT valtio.nimi AS valtionimi, sijaitsee.valtio_nimi AS sijaitseevaltio_nimi,
sijaitsee.järvi_nimi AS sijaitseejärvi_nimi, järvi.nimi AS järvinimi, joki.nimi AS
jokinimi
FROM valtio LEFT JOIN sijaitsee ON valtio.nimi = sijaitsee.valtio_nimi LEFT JOIN järvi ON
sijaitsee.järvi_nimi = järvi.nimi LEFT JOIN liittyy ON järvi.nimi = liittyy.järvi_nimi
LEFT JOIN joki ON liittyy.joki_nimi = joki.nimi
WHERE järvi.nimi IN (SELECT järvi.nimi FROM valtio LEFT JOIN sijaitsee ON valtio.nimi =
sijaitsee.valtio_nimi LEFT JOIN järvi ON sijaitsee.järvi_nimi = järvi.nimi LEFT JOIN
liittyy ON järvi.nimi = liittyy.järvi_nimi LEFT JOIN joki ON liittyy.joki_nimi =
joki.nimi WHERE joki.nimi = 'Torniojärvi' OR järvi.nimi = 'Torniojärvi') OR järvi.nimi IN
(SELECT järvi.nimi FROM valtio LEFT JOIN sijaitsee On valtio.nimi = sijaitsee.valtio_nimi
LEFT JOIN järvi ON sijaitsee.järvi_nimi = järvi.nimi WHERE järvi.pinta-ala > 5000)) AS t1
INNER JOIN
```

```
(SELECT valtio.nimi AS valtionimi, sijaitsee.valtio_nimi AS sijaitseevaltio_nimi,
sijaitsee.järvi_nimi AS sijaitseejärvi_nimi, järvi.nimi AS järvinimi, järvi.pinta-ala AS
järvipinta-ala
FROM valtio LEFT JOIN sijaitsee On valtio.nimi = sijaitsee.valtio_nimi LEFT JOIN järvi ON
sijaitsee.järvi_nimi = järvi.nimi
WHERE järvi.nimi IN (SELECT järvi.nimi FROM valtio LEFT JOIN sijaitsee On valtio.nimi =
sijaitsee.valtio_nimi LEFT JOIN järvi ON sijaitsee.järvi_nimi = järvi.nimi LEFT JOIN
liittyy ON järvi.nimi = liittyy.järvi_nimi LEFT JOIN joki ON liittyy.joki_nimi =
joki.nimi WHERE joki.nimi = 'Torniojärvi' OR järvi.nimi = 'Torniojärvi') OR järvi.nimi IN
(SELECT järvi.nimi FROM valtio LEFT JOIN sijaitsee On valtio.nimi = sijaitsee.valtio_nimi
LEFT JOIN järvi ON sijaitsee.järvi_nimi = järvi.nimi WHERE järvi.pinta-ala > 5000)) AS t2
ON t1.valtionimi = t2.valtionimi AND t1.sijaitseevaltio_nimi = t2.sijaitseevaltio_nimi
AND t1.sijaitseejärvi_nimi = t2.sijaitseejärvi_nimi AND t1.järvinimi = t2.järvinimi
WHERE ((t1.jokinimi IS NOT NULL) AND (t2.järvipinta-ala IS NOT NULL))
ORDER BY t1.valtionimi
```

(10)

```
SELECT nimi
FROM valtio/sijaitsee|virtaa//joki

⟨ ⟨⟨{⟨valtio,sijaitsee,//,joki⟩,⟨valtio,virtaa,//,joki⟩}⟩, ⟨⟩⟩, ⟨⟨{⟨nimi⟩}⟩, ⟨⟩⟩ ⟩

p1 H = ⟨ ⟨⟨{⟨valtio,sijaitsee, //,joki⟩,⟨valtio,virtaa, //,joki⟩}⟩, ⟨⟩⟩, ⟨⟨{⟨nimi⟩}⟩, ⟨⟩⟩ ⟩
p3 Hu = ⟨⟨⟨{⟨valtio,sijaitsee, //,joki⟩,⟨valtio,virtaa, //,joki⟩}⟩,⟨⟩⟩⟩, L = ⟨⟨{⟨nimi⟩}⟩, ⟨⟩⟩,
c(Hu) = 1, c(L) = 2
     p4 L = ⟨⟨{⟨valtio,sijaitsee, //,joki⟩,⟨valtio,virtaa, //,joki⟩}⟩, ⟨⟩⟩, c(L) = c(Hu)
     p7 W = ⟨{⟨valtio,sijaitsee, //,joki⟩,⟨valtio,virtaa, //,joki⟩}⟩, c(W) = c(L)
     p8 Ps = {⟨valtio,sijaitsee, //,joki⟩,⟨valtio,virtaa, //,joki⟩}, c(Ps) = c(W)
     p13 P = ⟨valtio,sijaitsee, //,joki⟩, Pst = {⟨valtio,virtaa, //,joki⟩}, c(P) = c(Ps)
          p15 Pc = ⟨valtio,sijaitsee,//,joki⟩, t(Ptc) = Tables, c(Pc) = c(P)
          p18 EN = valtio, Pc2 = ⟨sijaitsee, //,joki⟩, t(EN) = t(Pc1), t(Pc2) = Tables,
          c(EN) = 1, c(Pc2) = c(Pc1)
               p22 qs(EN) = ⟨⟨⟩, ⟨⟨⟨6,valtio⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩

               p19 EN = sijaitsee, Pc2 = ⟨joki⟩, t(EN) = t(Pc1), t(Pc2) = Tables, c(EN) =
               1, c(Pc2) = c(Pc1)
                    p22 qs(EN) = ⟨⟨⟩, ⟨⟨⟨5,sijaitsee⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩

                    p16 EN = joki, t(EN) = t(Pc), c(EN) = c(Pc)
                    p22 qs(EN) = ⟨⟨⟩, ⟨⟨⟨2,joki⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩
                    p16 qs(Pc) = qs(EN)
               p19 qs(Pc) = ⟨⟨⟩, ⟨⟨⟨5,sijaitsee⟩,⟨⟩⟩, ⟨⟨2,joki⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩
          p18 qs(Pc1) = ⟨⟨⟩, ⟨⟨⟨6,valtio⟩,⟨5,sijaitsee⟩⟩, ⟨⟨5,sijaitsee⟩,⟨⟩⟩, ⟨⟨2,joki⟩,⟨⟩⟩⟩,⟨⟩,⟨⟩⟩
          p15 qs(P) = qs(Pc)

          p12 P = ⟨valtio,virtaa,//,joki⟩
          p15 Pc = ⟨valtio,virtaa,//,joki⟩, t(Ptc) = Tables
          p18 EN = valtio, Pc2 = ⟨virtaa,//,joki⟩, t(EN) = t(Pc1), t(Pc2) = Tables, c(EN)
          = 1, c(Pc2) = c(Pc1)
               p22 qs(EN) = ⟨⟨⟩, ⟨⟨⟨6,valtio⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩

               p19 EN = virtaa, Pc2 = ⟨joki⟩, t(EN) = t(Pc1), t(Pc2) = Tables, c(EN) = 1,
               c(Pc2) = c(Pc1)
                    p22 qs(EN) = ⟨⟨⟩, ⟨⟨⟨1,virtaa⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩

                    p16 EN = joki, t(EN) = t(Pc), c(EN) = c(Pc)
                    p22 qs(EN) = ⟨⟨⟩, ⟨⟨⟨2,joki⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩
                    p16 qs(Pc) = qs(EN)
               p19 qs(Pc) = ⟨⟨⟩, ⟨⟨⟨1,virtaa⟩,⟨⟩⟩, ⟨⟨2,joki⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩
          p18 qs(Pc1) = ⟨⟨⟩, ⟨⟨⟨6,valtio⟩,⟨1,virtaa⟩⟩, ⟨⟨1,virtaa⟩,⟨⟩⟩, ⟨⟨2,joki⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩
          p15 qs(P) = qs(Pc)
          p12 qss(Ps) = ⟨qs(P)⟩
```

p13 qss(Ps1) = ⟨⟨⟨⟩, ⟨⟨⟨6,valtio⟩,⟨5,sijaitsee⟩⟩, ⟨⟨5,sijaitsee⟩,⟨⟩⟩, ⟨⟨2,joki⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩,
⟨⟨⟩, ⟨⟨⟨6,valtio⟩,⟨1,virtaa⟩⟩, ⟨⟨1,virtaa⟩,⟨⟩⟩, ⟨⟨2,joki⟩,⟨⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩
p8 qss(W) = qss(Ps)
p7 qss(L) = qss(W)
p4 qss(Hu) = ⟨⟨⟨⟩, ⟨⟨⟨6,valtio⟩,⟨5,sijaitsee⟩⟩,
⟨⟨5,sijaitsee⟩,⟨4,järvi⟩⟩,⟨⟨4,järvi⟩,⟨3,liittyy⟩⟩,⟨⟨3,liittyy⟩,⟨2,joki⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩,
⟨⟨⟩, ⟨⟨⟨6,valtio⟩,⟨1,virtaa⟩⟩,⟨⟨1,virtaa⟩,⟨2,joki⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩

p7 W = ⟨{⟨nimi⟩}⟩, c(W) = c(L)
p8 Ps = {⟨nimi⟩}, c(Ps) = c(W)
p12 P = ⟨nimi⟩, c(P) = c(Ps)
p15 Pc = ⟨nimi⟩, t(Pc) = Tables, c(Pc) = c(P)
p16 EN = nimi, t(EN) = t(Pc), c(EN) = c(Pc)
p24 qs(EN) =
⟨⟨⟨⟨2,joki⟩,nimi⟩,⟨⟨4,järvi⟩,nimi⟩,⟨⟨6,valtio⟩,nimi⟩,⟨⟨7,kaupunki⟩,nimi⟩⟩,⟨⟩,⟨⟩,⟨⟩⟩⟩
p16 qs(Pc) = qs(EN)
p15 qs(P) = qs(Pc)
p12 qss(Ps) = ⟨qs(P)⟩
p8 qss(W) = qss(Ps)
p7 qss(L) = qss(W)

p3 qss(H) = ⟨⟨⟨⟨⟨2,joki⟩,nimi⟩⟩, ⟨⟨⟨6,valtio⟩,⟨5,sijaitsee⟩⟩,
⟨⟨5,sijaitsee⟩,⟨4,järvi⟩⟩,⟨⟨4,järvi⟩,⟨3,liittyy⟩⟩,⟨⟨3,liittyy⟩,⟨2,joki⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩,
⟨⟨⟨⟨2,joki⟩,nimi⟩, ⟨⟨4,järvi⟩,nimi⟩⟩, ⟨⟨⟨6,valtio⟩,⟨1,virtaa⟩⟩,⟨⟨1,virtaa⟩,⟨2,joki⟩⟩⟩, ⟨⟩, ⟨⟩⟩⟩⟩

p1 string(S) =

```
SELECT t1.valtionimi, t1.jokinimi
FROM
(SELECT valtio.nimi AS valtionimi, joki.nimi AS jokinimi
FROM valtio LEFT JOIN sijaitsee ON valtio.nimi = sijaitsee.valtio_nimi LEFT JOIN järvi ON
sijaitsee.järvi_nimi = järvi.nimi LEFT JOIN liittyy ON järvi.nimi = liittyy.järvi_nimi
LEFT JOIN joki ON liittyy.joki_nimi = joki.nimi) AS t1
WHERE (t1.jokinimi IS NOT NULL)
ORDER BY t1.valtionimi

SELECT t1.valtionimi, t1.jokinimi, t1.järvinimi
FROM
(SELECT valtio.nimi AS valtionimi, joki.nimi AS jokinimi, järvi.nimi AS järvinimi
FROM valtio LEFT JOIN virtaa ON valtio.nimi = virtaa.valtio_nimi LEFT JOIN joki ON
virtaa.joki_nimi = joki.nimi LEFT JOIN liittyy ON joki.nimi = liittyy.joki_nimi LEFT JOIN
järvi ON liittyy.järvi_nimi = järvi.nimi) AS t1
WHERE (t1.jokinimi IS NOT NULL OR t1.järvinimi IS NOT NULL)
ORDER BY t1.valtionimi
```