Logiikkapohjainen lähestymistapa RDF-tietolähteen esittämiseksi

Tuomas Räsänen

Tampereen yliopisto
Luonnontieteiden tiedekunta
Tietojenkäsittelytieteiden tutkinto-ohjelma
Pro gradu -tutkielma
Ohjaaja: Marko Junkkari, Kati Iltanen
Maaliskuu 2017

Tampereen yliopisto
Luonnontieteiden tiedekunta
Tietojenkäsittelytieteiden tutkinto-ohjelma
Tekijän Nimi: Tuomas Räsänen
Pro gradu -tutkielma, 64 sivua, 9 liitesivua
Maaliskuu 2017

Semanttinen verkko on internetin laajennos, jossa internetsivuille lisätään tietoa niiden sisällön merkityksestä sellaisessa muodossa, jonka tietokone voi ymmärtää. Tämä tieto esitetään käyttäen RDF-tietomallia ja sen laajennoksia. Semanttisessa verkossa tietolähteet yksilöidään antamalla niille URI-osoitteet. RDF-tietomalli on graafi, joka koostuu näistä URI-osoitteista, niihin liittyvistä arvoista ja niiden välisistä suhteista. RDF-graafi ilmaistaan kolmikkoina, joiden elementit ovat subjekti, predikaatti ja objekti. Subjekti on URI, objekti on URI tai arvo. Predikaatti kertoo subjektin ja objektin suhteen.

Tässä tutkielmassa kehitetään uusi relaatioesitys RDF-tietolähteelle. Lisäksi verrataan kyselyjen tekemistä uudella tavalla esitettyyn tietoon ja kyselyjen tekemistä jo olemassa olevilla ratkaisuilla. Vertailukohteet ovat SPARQL ja SWI-Prolog-kielen valmispredikaatit RDF-tiedon käsittelyyn. Kyselyjä tehdessä tutkitaan sitä, millaisia kyselyjä voidaan toteuttaa ja sitä kuinka vaikea käyttäjän on niitä toteuttaa. Tulokset osoittavat, että kehitetty tapa mahdollistaa sellaisten kyselyjen tekemisen, joihin SPARQL ja SWI-Prolog-kielen valmispredikaatit eivät pysty ilman RDF-mallin laajennuksia. Käytettävyydeltään mikään ratkaisuista ei erotu erityisesti edukseen.

Avainsanat ja -sanonnat: RDF, XML, RDF/XML, Prolog, SPARQL

Sisällys

1. Johdanto

Semanttinen verkko on internetin laajennus, jonka tarkoituksena on lisätä internetsivuille tietoa niiltä löytyvän sisällön merkityksestä. Tämä puolestaan mahdollistaa paljon monipuolisemman toiminnallisuuden verrattuna tavanomaiseen internetiin. [Berners-Lee et al., 2001]

Semanttinen verkko, toisin kuin internet, on suunnattu ensisijaisesti koneille. Tarkoituksena on luoda internetin merkitykselliselle sisällölle sellainen rakenne, jonka avulla tietokone pystyy tietämään, mitä kyseinen sisältö koskee. Tämän pohjalta voidaan kehittää ohjelmia, jotka pystyvät tuottamaan monipuolisempia palveluita käyttäjille. Tällainen ohjelma ei vain tunnista sivuston avainsanoja, vaan myös muuta tietoa sivustolta, kuten kuka kyseisen sisällön on tuottanut tai milloin jokin kauppa on auki. Tällaisen ohjelman toteuttaminen ei vaadi monimutkaista tekoälyä, vaan se perustuu siihen, että tarvittu tieto on sisällytetty www-sivuun itseensä. [Berners-Lee et al., 2001]

Jotta semanttinen verkko voi toimia, täytyy sen sisältämä tieto esittää jollakin soveltuvalla tavalla. Suurin osa perinteisistä tietämyksen esittämisen ratkaisuista ja tietopohjaisista järjestelmistä on sidoksissa tiettyihin aiheisiin ja samaa aihetta käsittelevät sovellukset saattavat esittää asian eri tavalla. Lisäksi mahdollisten kysymysten määrää on rajoitettu, jotta tietokone pystyy vastaamaan niihin luotettavasti. Tiedon lisäksi näillä järjestelmillä on myös järjestelmäkohtaisia sääntöjä, joiden pohjalta ne pystyvät tekemään päättelyjä niille annetusta datasta. Perinteiset ratkaisut pystyvät kuitenkin vastaamaan hyvin oman aihealueensa kysymyksiin. [Berners-Lee et al., 2001] Esimerkki tällaisesta järjestelmästä voisi olla vaikkapa asiantuntijajärjestelmä, joka tunnistaa taudin annettujen oireiden perusteella.

Semanttisen verkon tapauksessa joudutaan kuitenkin tekemään kompromisseja ja hyväksymään, että kaikkeen ei välttämättä saada vastausta. Tämä johtuu siitä, että semanttisen verkon tarkoituksena on kerätä tietoa useista eri lähteistä ja koota niistä käyttäjille hyödyllistä sisältöä. Toisaalta semanttisen verkon kehittäjät ovat pyrkineet luomaan kielen, joka tarjoaa kaikille internetin kehittäjille mahdollisuuden esittää tietolähteensä yhtenäisellä tavalla. Tämän kielen pohjalta semanttisen verkon sovellukset pystyvät tekemään tarvitsemiaan päättelyjä. [Berners-Lee et al., 2001]

Semanttinen verkko esittää tietoresursseja (lähde, joka sisältää tietoa jostakin asiasta tai asioista) ja niiden välisiä suhteita. Jokaisella tietoresurssilla on suhdeverkossa yksikäsitteinen URI-osoite (Unified Resource Identifier), joka identifioi kyseisen tietoresurssin. Lisäksi tarvitaan jokin tapa tai kieli, jolla tieto näistä URI-osoitteista sekä niiden välisistä suhteista merkitään internetsivun koodiin, mihin liittyen käytetyt tietoresurssit ja niiden

suhteet on määriteltävä jonkin ontologian avulla. Ontologia on joukko lauseita, jotka määrittelevät entiteetit ja niiden väliset suhteet. [Berners-Lee et al., 2001]

Alun perin Berners-Lee et al. [2001] ehdottivat tähän tarkoitukseen RDF-tietomallia (Resource Description Framework), joka pystyy kertomaan tietoresurssien URI-osoitteet sekä niiden väliset suhteet. Myöhemmin tälle kehitettiin laajennos OWL (Web Ontology Language), jolla on RDF-tietomallia parempi ilmaisuvoima [Shadbolt et al., 2006]. OWL-tietomallista on kolme ilmaisuvoimaltaan erilaista versiota. OWL-mallin parempi ilmaisuvoima tulee siitä, että se mahdollistaa ontologioiden viittaamisen toisiinsa ja tarjoaa sääntöjä, joiden avulla voidaan tehdä päättelyjä merkittyjen tietoresurssien pohjalta [Shadbolt et al., 2006]. Hieman erilaista koulukuntaa edustaa JSON-LD [Sporny et al., 2014]. Tässä tiedot tietoresursseista annetaan suosittua JSON-formaattia käyttäen. Myös muita esitystapoja löytyy. Tässä työssä kuitenkin keskitytään RDF-tietomallin käsittelyyn.

Vuosien aikana semanttisen verkon teknologioita on hyödynnetty monenlaisissa sovelluksissa. Bernstein et al. [2016] listaavat joukon näitä käyttökohteita. Yli 2,5 miljardia verkkosivua käyttää semanttisen verkon merkintöjä sisältönsä kuvaamiseen yhteisellä sanastolla, monet yhtiöt ja tutkimusryhmät pyrkivät kehittämään tietograafeja (knowledge graphs) datastaan, jotta pystyisivät tarjoamaan parempia palveluita, kaupalliset tietokannanhallintajärjestelmät tarjoavat tukea semanttisen verkon teknologioille, suositusyhtiöt (recommender companies) käyttävät semanttisen verkon teknologioita suositustensa parantamiseen ja Maailman terveysjärjestö WHO kehittää tautiontologiaa kaikkien Yhdistyneiden kansakuntien jäsenvaltioiden käyttöön. Bernstein et al. [2016] kuitenkin huomauttavat, ettei tässä ole koko lista ja mainitsevat, että tulevaisuudessa on mahdollisesti luvassa vielä monimuotoisempia sovelluksia.

Koska semanttisella verkolla on lupaava tulevaisuus, on myös syytä kehittää sille parempia ja monipuolisempia työkaluja. Tässä työssä kehitetään ja esitetään yksi tällainen työkalu. Työssä tutkitaan RDF-tiedon esittämistä XML-relaatioesitystavan [Niemi ja Järvelin, 2006] avulla. Työssä kehitetään XML-relaatioon laajennus, joka mahdollistaa RDF/XML-dokumenttien käännön XML-relaatiomuotoon. Tätä laajennettua XML-relaatiota kutsutaan RDF/XML-relaatioksi. Oletuksena on, että kun tieto esitetään RDF/XML-relaationa, siihen voidaan tehdä joitakin sellaisia hakuja, joita on joko vaikea tai mahdoton toteuttaa muilla tavoilla, käyttämättä RDF-mallin laajennoksia. Tätä väitettä tutkitaan vertaamalla RDF/XML-relaatioon tehtäviä kyselyjä SPARQL-kyselyihin sekä SWI-Prologin RDF-tiedon käsittelyyn tarkoitetuilla predikaateilla tehtäviin kyselyihin. Erityisesti tutkitaan, pystyykö RDF/XML-relaatio suoriutumaan ns. analyyttisistä kyselyistä kilpailijoitaan paremmin tilanteessa, jos on käytössä pelkästään RDF-kuvaus.

Tietomallin esitystapa tarjoaa pohjan sille, että kuinka tietolähteistä voidaan hakea tietoa. SPARQL on syntaksiltaan SQL-kieleen perustuva kyselykieli, joka etsii tietoa RDF-graafeista (graafi voidaan esittää usealla eri tavalla, joilla ei kyselyn tekemisen ja toiminnan kannalta ole eroa) mallinsovituksella, eli antamalla vastauksena ne osat RDF-tietolähteestä, jotka vastaavat SPARQL-kyselyssä annettuja ehtoja [Della Valle and Ceri, 2011]. SWI-Prologin valmispredikaattien pohjalla on järjestelmä, johon RDF-tietolähteet on tallennettu viisiosaisina monikkoina, joiden kolme ensimmäistä alkiota esittävät RDF-kolmikon ja joiden kaksi viimeistä alkiota antavat tietoa kolmikon alkuperästä [Wielemaker, 2016]. Tähän tietoon voidaan sitten tehdä hakuja annettujen valmispredikaattien avulla. Tässä työssä kehitettävässä RDF/XML-relaatiossa tieto puolestaan esitetään kolmikkoina, joiden alkiot kertovat RDF/XML-dokumentissa sijaitsevan tiedon sisällön, tyypin (elementti, attribuutti, URI vai literaaliarvo) ja sijainnin dokumentissa. Nämä relaatiot esitetään SWI-Prolog-muodossa ja lisäksi laaditaan predikaatteja, joilla niihin voidaan tehdä hakuja.

Työ etenee seuraavasti. Luvussa 2 perehdytään tarkemmin RDF-tietomalliin. Luvussa 3 tutustutaan SPARQL kyselykieleen. Luvussa 4 katsotaan, miten RDF-mallia voidaan käsitellä SWI-prologin valmispredikaateilla. Luvussa 5 kehitetään ja esitellään RDF/XML-relaatio. Luvussa 6 tehdään erilaisia kyselyjä tutkittaviin lähestymistapoihin. Luvussa 7 arvioidaan eri ratkaisujen käytettävyyttä ja luvussa 8 vedetään johtopäätökset.

2. RDF-tietoresurssien esittäminen

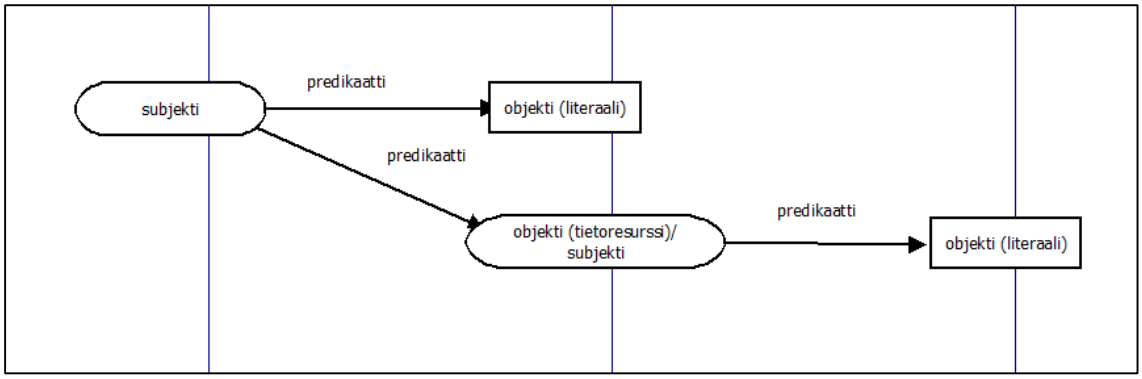
Berners-Leen et al. [2001] mukaan tietoresurssi tarkoittaa mitä tahansa entiteettiä, josta voidaan kerätä tietoa. Tällaisia entiteettejä ovat esimerkiksi WWW-sivut tai niiden osat, erilaiset laitteet tai vaikkapa ihmiset. Tietoresurssien merkitys esitetään semanttisessa verkossa käyttäen RDF-kieltä [Berners-Lee et al., 2001]. RDF esitetään joukkona kolmikkoja. Yksi kolmikko koostuu subjektista, predikaatista ja objektista hieman samaan tapaan kuin puhuttujen kielien lauseet [Berners-Lee et al., 2001]. Subjektilla ja joillakin objekteilla on jokin URI, joka identifioi niiden esittämän tietoresurssin [Berners-Lee et al., 2001]. Tämä muistuttaa tapaa, jolla linkit esitetään internetsivuilla. Internetlinkkiin liittyvä URL (Uniform Resource Locator) on URI-osoitteen erikoistapaus, joka identifioi tietoresurssin sen sijainnin perusteella [Coates et al., 2001]. Objekti voi olla myös jokin tekstimuotoinen arvo [Berners-Lee et al., 2001]. Predikaatit esitetään URI-osoitteina [Berners-Lee et al., 2001].

Uusia subjekteja, objekteja ja predikaatteja voi luoda määrittämällä niille URI-osoitteet [Berners-Lee et al., 2001]. URI varmistaa sen, että dokumentin sisällössä esiintyvä käsite ei ole vain sana dokumentissa, vaan sillä on yksiselitteinen määrite [Berners-Lee et al., 2001]. Kaikki tässä työssä käytetyt URI-osoitteet ovat myös URL-osoitteita. Tämä valinta tehtiin sen vuoksi, että URL-osoitteet ovat erityyppisistä URI-osoitteista parhaiten tunnettuja.

2.1. RDF-tietomalli

RDF-tietomalli voidaan kuvata suunnattuna graafina, jonka kaaret on nimetty [Cyganiak et al., 2014]. RDF-graafista löytyy kolmenlaisia solmuja. Nämä ovat URI, literaali ja tyhjä solmu [Cyganiak et al., 2014]. Graafin visuaalisessa esityksessä nämä solmut esitetään erilaisten kuvioiden avulla.

RDF-graafin voi esittää monella eri tapaa. Tässä työssä käytetään notaatiota [Miller et al., 1997], jossa graafin solmut esitetään soikioina, kaaret nuolina ja literaalit suorakaiteina. Graafin solmut ovat subjekteja ja objekteja, jotka ovat tietoresursseja. Kaaret ovat ominaisuuksia (predikaatit). Suorakaiteet puolestaan ovat ominaisuuden arvoja eli literaaleja.



Kuva 1. Esimerkki RDF-graafista

Kuvassa 1 on esitettynä pieni esimerkki RDF-tietomallista graafi-esityksenä. Tässä esimerkissä on kaksi subjektia, joista toinen on yhden kolmikon objektina. Lisäksi tässä on kaksi literaalia, jotka ovat siis graafin lehtiä.

Huomioitava piirre RDF-tietomallin loogisessa esityksessä on se, että siinä oletetaan maailman olevan avoin. Tämä tarkoittaa sitä, että jokin fakta voi olla tosi, vaikka sitä ei väitettäisikään todeksi. Tämä toimii kontrastina useampien tavanomaisten sovellusten (esim. relaatiomallin) käyttämään suljetun maailman malliin, jossa jokaisen faktan totuus pitää erikseen mainita. [Gandon et al., 2011]

2.2. Erilaiset syntaktiset vaihtoehdot RDF-tietolähteiden esittämiseksi

RDF-tietolähteiden esittämiseksi on muutamia erilaisia syntaktisia vaihtoehtoja. Nämä vaihtoehdot ovat Turtle, N-Triples, Notation 3 ja RDF/XML [Gandon et al., 2011].

Turtle on näistä kaikista yksinkertaisin. Se on kompakti tapa esittää RDF-tietolähteet. N-Triples-esitystapaan verrattuna siinä on vähemmän rajoitteita ja se on yksinkertaisempi kuin Notation 3. Turtle ei rajoitu yksirivisiin lauseisiin ja siinä on erilaisia lyhenteitä, jotka tekevät siitä helpommin ymmärrettävän kuin muista vaihtoehdoista. [Gandon et al., 2011]

N-Triples on Notation 3 -esitystavan ja Turtle-esitystavan alijoukko. Se on yksirivisiin lauseisiin ja tavanomaiseen tekstiin (vrt. perus-Notepad-sovellus) perustuva sarjallistamisformaatti RDF-tietolähteille. N-Triples yksinkertaistaa mallinnusta joillekin sovelluksille, mutta sillä on varsin huono pakkaussuhde (compression ratio), koska se vaatii täyden URI-osoitteen jokaiselle tietoresurssille eikä salli lyhenteitä. [Gandon et al., 2011]

Notation 3 tai lyhemmin N3 tuo tavanomaisen RDF-sarjallistamisen lisäksi lisää ilmaisuvoimaa. Siinä on mukana esimerkiksi muuttujia ja se sallii RDF-faktoihin perustuvien sääntöjen tuottamisen. Lisäksi se mahdollistaa graafien lainaamisen, jotta on mahdollista esittää lauseita lauseista. [Gandon et al., 2011]

Viimeisenä esitystapana on RDF/XML. Tämä on XML-perustainen esitystapa RDF-tietolähteille. Tämä työ keskittyy RDF/XML-esitystavan tarkastelemiseen, koska työssä kehitetään tätä vastaava relaatioesitys, jota voidaan kutsua RDF/XML-relaatioesitykseksi. RDF/XML-esitystapaa käsitellään tarkemmin aliluvussa 2.3.

2.3. RDF/XML-esitystapa

RDF/XML on RDF-tietolähteiden XML-kieleen perustuva esitystapa. Sen pyrkimyksenä on esittää RDF-tietolähde koneen luettavassa muodossa ja internetin pääasiallisen dokumenttien esitystavan, eli XML-kielen, kanssa yhteensopivana. Tämän vuoksi keskenään hyvinkin erilaiset järjestelmät voivat vaihtaa keskenään RDF/XML-muodossa esitettyä tietoa. [Gandon et al., 2011]

2.3.1. XML

XML (Extensible Markup Language) on kuvauskieli, jota voidaan käyttää toisten kielien kuvaamiseen tai spesifikaationa talletettaessa tietoa. XML pystyy tarjoamaan rakenteen kaikentyyppiselle tiedolle ja tämän vuoksi se pystyy kuvailemaan tietoa paljon tarkemmin kuin esimerkiksi tiedon formatointiin tarkoitettu HTML. Tämän ilmaisuvoiman XML perii SGML-kieleltä, mutta ilman niitä monimutkaisuuksia, jotka tekevät SGML:stä niin hankalasti sovellettavan. [Stanek, 2014]

XML tarjoaa käyttäjälle paljon vapauksia. XML-kielessä ei ole ennalta määrättyjä merkintöjä eikä attribuutteja, toisin kuin esimerkiksi HTML-kielessä. Käyttäjä voi lisätä näitä sen mukaan, miten kokee tarpeelliseksi ja päättää siten itse, että miten komponentit liittyvät toisiinsa. [Stanek, 2014]

XML-kielen avulla siis voi antaa konkreettisemman muodon abstrakteille käsitteille. XML mahdollistaa sen, että voidaan määritellä mitä osia johonkin tiettyyn käsitteeseen kuuluu ja miten ne liittyvät toisiinsa. Stanek [2014] käyttää esimerkkinä käsitettä jakelija. Jakelijalla on esimerkiksi nimi, yhteyshenkilö ja osoite. Lisäksi voitaisiin määritellä ja kuvata esimerkiksi jakelijan työntekijät ja tuotteet. [Stanek, 2014]

Dokumenttityyppimäärittely eli DTD (document type definition) antaa tietoa XML-dokumentin rakenteesta. DTD määrittelee säännöt sille, millaisia merkintöjä, elementtejä ja attribuutteja XML-dokumentissa voidaan käyttää. DTD-määrittelyssä voidaan määrittää esimerkiksi se, esiintyykö elementti kerran tai vähintään kerran dokumentissa. DTD-määritelmän avulla tietokoneen on mahdollista päätellä, onko dokumentti muodostettu hyvin. Dokumentin täytyy vastata DTD-määrittelyä, jos sille on sellainen annettu, mutta DTD-määrittelyn antaminen ei ole kuitenkaan pakollista. DTD voi olla osana XML-dokumenttia tai siihen voidaan viitata dokumentista. DTD mahdollistaa rajoitusten antamisen dokumentille, mutta kaikkien yksityiskohtien määrittelyyn siitä ei ole. DTD ei esimerkiksi pysty määrittämään sallittuja arvoja sisällölle. [Stanek 2014]

Toinen tapa, jolla XML-dokumentin rakennetta voi määritellä on XML-skeema. Kuten DTD, XML-skeema kertoo mitä elementtejä ja attribuutteja dokumenteissa voidaan käyttää, ja millaista tietoa ne voivat sisältää. XML-skeema laaditaan XML-skeemakielellä, joka puolestaan on määritelty XML-kielellä. XML-skeema on ilmaisuvoimaisempi ja joustavampi kuin täysin omaa syntaksia ja rakennettaan käyttävä DTD. Lisäksi toisin kuin DTD-määrittelyssä, jossa kaikki elementit ja attribuutit määritellään aina globaalisti (vaikuttaa koko dokumentin alueella), voidaan ne XML-skeemassa määritellä myös paikallisesti (vaikuttaa vain osassa dokumenttia). Tällöin samannimisiä elementtejä tai attribuutteja voidaan käyttää eri konteksteissa. XML-skeema tarjoaa myöskin mahdollisuuden määrittää minkälaista sisältöä jossakin elementissä voi olla ja miten tämä sisältö pitäisi muotoilla; DTD ei tarjoa näitäkään mahdollisuuksia. [Stanek 2014]

XML-nimiavaruudet ovat keskeinen ominaisuus XML-kielessä. Nimiavaruuksien avulla on mahdollista erottaa toisistaan samannimiset elementit ja attribuutit, kun ne viittaavat erityyppisiin tietorakenteisiin. Näin vältytään elementtien ja attribuuttien nimeämisiin liittyviltä ristiriidoilta ja dokumenttia käsittelevät ohjelmat ja ihmiset ymmärtävät mistä kulloinkin on kyse. Nimiavaruudet annetaan dokumenttiin käyttäen kvalitoituja nimiä (qualifed names), jotka annetaan dokumentille muodossa etuliite:lokaali osa. Etuliite kertoo, mikä nimiavaruus on kyseessä ja lokaali osa identifioi tietyn elementin tai attribuutin kyseisessä nimiavaruudessa. Jokaisella nimiavaruudella täytyy myös olla URI. Tämä voidaan antaa määrittelemällä `xmlns`-attribuutti jokaiselle etuliitteen omaavalle elementille. Esimerkkidatassa julistetaan yksi nimiavaruus seuraavasti

```
xmlns:esim="http://www.jokinesimerkkisanasto.fi/#"
```

ja siihen viitataan datassa esimerkiksi seuraavasti:

```
esim:tyyppi.
```

XML-data ei noudata sellaista tarkasti määriteltyä rakennetta kuin relaatiotietokannoissa oleva data. Relaatiotietokannoissa olevaa dataa kutsutaan rakenteiseksi dataksi, juuri sen tarkasti määritellyn rakenteen vuoksi. XML-dokumenteissa data voi olla puolestaan epäsäännöllistä, epätäydellistä ja se voi muuttua vaikeasti ennakoitavilla tavoilla, joten dokumenttien tietoa on mahdoton mallintaa minkään yksittäisen kaavion avulla. Tämän vuoksi XML-data kuvaa itse itsensä, eli sen semanttiset tulkinnat on yhdistetty dokumentin tietoon. Tällaista dataa kutsutaan puolirakenteiseksi. [Niemi ja Järvelin 2006]

2.3.2. RDF/XML

RDF-graafin kuvaaminen XML-kielen avulla tapahtuu seuraavasti. Graafin solmut ja predikaatit kuvataan XML-kielen termeillä, eli elementeillä ja attribuuteilla on nimet sekä elementillä sisältö ja attribuutilla arvo [Gandon and Schreiber, 2014]. URI-osoitteet kuvataan RDF/XML-esitystavassa käyttäen kvalitoituja nimiä, kuten Bray et al. [2009] ovat ne määritelleet. Jokaisella kvalitoidulla nimellä on nimiavaruusnimi, joka on URI ja lyhyt lokaalinimi [Gandon and Schreiber, 2014]; lisäksi on mahdollista tarvittaessa lisätä tämän eteen etuliite.

Nimiavaruuksia käytetään RDF/XML:ssä yleensä kertomaan, mitä ontologiaa tai sanastoa (sanasto on ontologiaa yksinkertaisempi ja vapaamuotoisempi tapa määritellä käytetyt termit [Dominque et al., 2011]) on käytetty käsitteiden määrittämiseen. Esimerkiksi etuliitteeseen `'esim'` voi liittyä nimiavaruusnimi ”`www.esimerkki.fi`”. Tällöin kvalitoitu nimi `esim:esimerkki` viittaisi osoitteeseen ”`www.esimerkki.fi/esimerkki`”. RDF-mallin käyttöön vaadittu sanasto löytyy nimiavaruusnimestä ”`http://www.w3.org/1999/02/22-rdf-syntax-ns#`”, ja sille asetetaan yleensä etuliitteeksi `rdf`. RDF/XML-dokumentit eivät voi käyttää sellaista XML-nimiavaruutta, jonka nimiavaruusnimi on RDF-nimiavaruusnimi, johon on lisätty useampia kirjaimia. [Gandon and Schreiber, 2014]

2.3.3. Esimerkkisanasto Esim

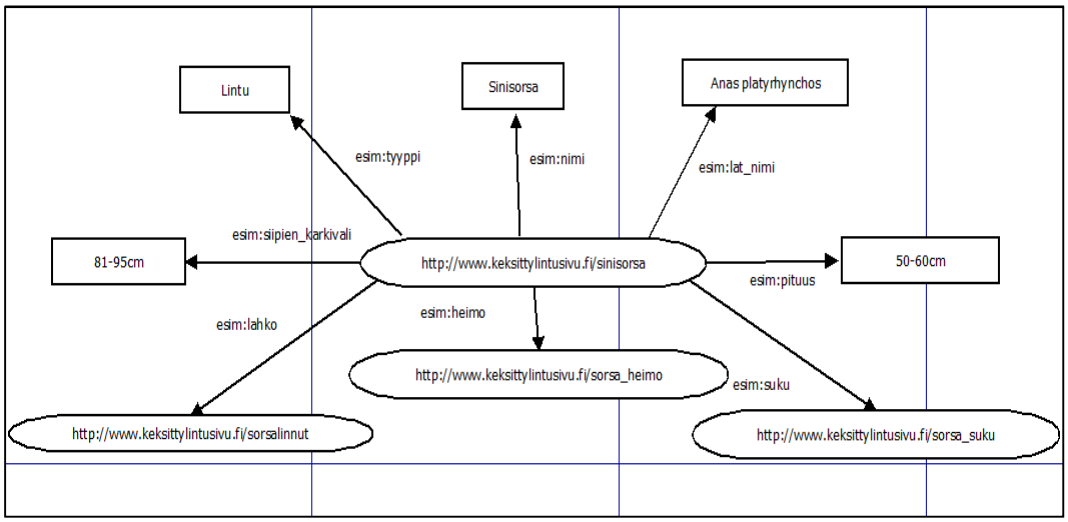
Tämän työ tarpeisiin kehitettiin pieni esimerkkisanasto, josta käytetään aineistossa nimeä `esim` ja jolle annettu URI-osoite on ”`http://www.jokinesimerkkisanasto.fi/#`”, joka on täysin keksitty eli kyseisestä osoitteesta ei oikeasti löydy mitään. Sanastosta löytyy seuraavat määritelmät. `Tyyppi` tarkoittaa tietoresurssin tyyppiä. Se voi olla esimerkiksi `lintu` tai `lahko` yms. `Nimi` tarkoittaa tietoresurssin nimeä ja tämä voi olla lajin nimi tai fiktiivisen hahmon nimi. `Lat_nimi` puolestaan tarkoittaa lajin latinan kielistä nimeä. `Pituus` tarkoittaa tietoresurssin kuvaaman linnun pituutta ja `siipien_karkivali` sen siipien kärkiväliä. `Heimo`, `lahko` ja `suku` ovat kuvatun linnun heimo, lahko ja suku, jotka sille on annettu modernissa eläinten luokittelussa. Myöhemmin lisätään sana `fiktiossa`, joka tarkoittaa kyseisen lajin fiktiossa esiintyvää edustajaa. Liitteessä 5 annetaan sanaston formaalimpi määritelmä.

2.3.4. Esimerkki RDF-tietolähteen XML-esitystavasta

Esimerkissä 1 on kuvattu Sinisorsa RDF/XML-kielellä. Lisäksi siinä on hieman tietoa käytetyistä sanastoista. Kuvassa 2 on esimerkki 1 kuvattuna RDF-graafina. Tässä käytetty RDF/XML-esimerkki on otos liitteessä 1 esitetystä lintudatasta.

Esimerkki 1:

```
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:esim="http://www.jokinesimerkkisanasto.fi/#">
   <rdf:Description rdf:about="http://www.keksittylintu-
sivu.fi/sinisorsa">
         <esim:tyyppi>Lintu</esim:tyyppi>
         <esim:nimi>Sinisorsa</esim:nimi>
         <esim:lat_nimi>Anas platyrhynchos</esim:lat_nimi>
         <esim:pituus>50-60cm</esim:pituus>
         <esim:siipien_karkivali>81-95cm</esim:siipien_karkivali>
         <esim:lahko rdf:resource="http://www.keksittylintu-
sivu.fi/sorsalinnut" />
         <esim:heimo rdf:resource="http://www.keksittylintu-
sivu.fi/sorsa_heimo" />
         <esim:suku rdf:resource="http://www.keksittylintu-
sivu.fi/sorsa_suku" />
   </rdf:Description>
</rdf:RDF>
```



Kuva 2. Esimerkin 1 RDF/XML-dokumentti esitettynä kuvana.

Alkumerkintä `rdf:RDF` kertoo, että kyseessä on RDF/XML-dokumentti. RDF-nimiavaruus kertoo mistä löytyy tavanomaisen RDF-kielen sanasto. `Esim` puolestaan on tätä esimerkkiä varten laadittu esimerkkisanasto.

Seuraava kohta (`rdf:Description rdf:about="http://www.keksittylintu-sivu.fi/sinisorsa`) kertoo, että tässä kohdassa kuvataan sinisorsa. `Rdf:Description` kertoo, että nyt alkaa RDF-kuvaus ja `rdf:about` kertoo mitä tietoresurssia kuvataan. Tämä on RDF-kolmikon subjekti. Kaikki `esim-`alkuiset kohdat kuten `esim:nimi` ja esim:`lahko` ovat predikaatteja. Niiden merkitys on selitetty sanastossa `esim`. Objekti voi olla esitetty joko tämän elementin arvona, jos kyseessä on literaali kuten `<esim:nimi>Sinisorsa</esim:nimi>` kohdassa tai attribuuttina, jos kyseessä on URI kuten kohdassa `<esim:lahko rdf:resource="http://www.keksittylintu-sivu.fi/sorsalinnut" />`. `Rdf:resouce` kertoo, että objektina on annetun URI-osoitteen merkitsemä tietoresurssi.

2.4. RDFS

RDFS eli RDF-skeema on semanttinen laajennos RDF-kielelle. RDFS tarjoaa kevyen tavan kuvailla tietoresursseja (tyyppi ja ominaisuudet) sekä niiden välisiä suhteita. Tämän mahdollistaa RDF-skeeman kyky nimetä ja määritellä RDF-graafien nimeämiseen käytettyjä sanastoja, joita voidaan sitten käyttää tietoresurssien ja niiden välisten suhteiden nimeämiseen. Kun RDFS vielä mahdollistaa URI-osoitteiden antamisen erityyppisille tietoresursseille, voidaan sitä käyttää ontologioiden esittämiseen. Tässä työssä keskitytään kuitenkin vain RDF-datan käsittelyyn, joten RDF-skeemaan perustuvia ratkaisuja ei tässä työssä käytetä. [Gandon et al., 2011]

3. SPARQL-kyselykielen piirteet

SPARQL (Simple Protocol and RDF Query Language) on kyselykieli RDF-tietoresurssien käsittelemiseksi. SPARQL on suunniteltu semanttista verkkoa varten ja se on W3C:n suositus semanttisen verkon kyselykieleksi. Syntaktisesti SPARQL muistuttaa SQL-kieltä ja se hakee tietoa RDF-graafeista mallinsovituksen avulla. [Della Valle and Ceri, 2011]

SPARQL olettaa, että tieto on esitetty RDF-graafina, tietoresurssit URI-osoitteina ja että RDF-literaaleille on määritelty XML-skeeman avulla tietotyyppi (desimaaliluku, päivämäärä jne.) [Della Valle and Ceri, 2011]. Jos tyyppi jätetään määrittelemättä, literaali on automaattisesti tavallista tekstiä (String-tyyppinen muuttuja) [Prud'hommeaux and Seaborne, 2008]. Myös jos literaalin tietotyyppi on kokonaisluku, ei tyyppiä ole pakko erikseen määritellä [Prud'hommeaux and Seaborne, 2008]. Literaalien arvoja voidaan tutkia SPARQL-kielessä mukana olevien XPath-operaattoreiden avulla [Della Valle and Ceri, 2011].

Koska tieto voi olla semanttisessa verkossa useilla eri verkkosivuilla, kieleen sisältyy myös yksinkertainen kommunikaatioprotokolla. Tämän protokollan avulla on mahdollista koota yhteen eri lähteistä löytyvää tietoa järkeväksi kokonaisuudeksi. [Della Valle and Ceri, 2011]

Tässä työssä SPARQL-kielen toimintojen esittelyyn käytetään Java-pohjaista Apache Jena-ohjelmistokehystä [Apache Jena, 2016]. `DESCRIBE`-kyselyn kohdalla esitetään vertailun vuoksi toinen esimerkki käyttäen niin ikään Java-pohjaista Eclipse Rdf4j-ohjelmistokehystä [Eclipse RDF4J, 2016].

3.1. SPARQL-kyselyn rakenne

SPARQL-kyselyssä on viisi osaa. Nämä ovat `PREFIX`-osa, `SELECT`-osa, `FROM`-osa, `WHERE`-osa ja kyselyn loppuun tulevat muokkaimet. Näistä pakollisia ovat `SELECT`- tai kyselyn tyyppi -osa ja `WHERE`-osa, joka määrittelee ehdot sille, mitä haetaan. [Della Valle and Ceri, 2011]

`PREFIX`-osassa määritellään lyhenteitä kyselyssä tarvittaville eli aineistossa esiintyville URI-osoitteille. Näitä voi olla nolla tai enemmän. Yleensä kuitenkin tässä on vähintään RDF-mallin määrittelevän URI-osoitteen lyhenne. [Della Valle and Ceri, 2011]

Pakollisena osana kyselyssä on kyselyn tyypin määrittelevä osuus. Siis se osuus, joka kertoo, onko kyseessä `SELECT`-, `CONSTRUC`-, `ASK`- tai `DESCRIBE`-kysely. Tässä yhteydessä

annetaan myös ne muuttujat, joihin halutaan saada kyselyllä arvot tai joita halutaan tutkia. [Della Valle and Ceri, 2011]

`FROM`-osiossa määritetään mihin dataan haku tehdään. `WHERE`-osiossa annetaan ehdot, joiden perusteella aiemmin annetut muuttujat saavat arvonsa. Toisin sanoen valitaan ne aligraafit, eli osat alkuperäisestä graafista, jotka palautetaan vastauksena. Tämä osio voi vaihdella yksinkertaisesta melko monimutkaiseen ja siinä määritellään ehtoja, jotka vastaukseen kelpaavien kolmikkojen on täytettävä. Kyselyn lopuksi voidaan antaa vielä muokkaimet. Näitä muokkaimia ovat esimerkiksi se, kuinka monta kolmikkoa otetaan tulokseen tai se, miten ne on järjestetty. [Della Valle and Ceri, 2011]

3.2. SPARQL-muuttuja

SPARQL-kielessä muuttuja esitetään siten, että sen nimen edessä on joko `?`- tai `$`-symboli. Muuttujia voivat olla esimerkiksi `?o` tai `$talo`. `?` ja `$` merkit eivät kuitenkaan ole osa muuttujan nimeä vaan esimerkiksi `?a` ja `$a` tarkoittavat samaa muuttujaa [Harris and Seaborne, 2013]. Tässä työssä käytetään SPARQL-kyselyissä muuttajasta merkintää `?`.

Arvoksi muuttujalle tulee kyselyn `WHERE`-osiossa siihen yhdistetty arvo. Jos esimerkiksi käytetään lintudataa ja halutaan muuttujalle `?tyyppi` arvoksi `Lintu`, pitäisi WHERE-osiossa olla ehtona esimerkiksi kolmikko `sinisorsan URI esim:tyyppi ?tyyppi`. Tällöin muuttuja saisi arvokseen `Lintu`. Jos taas haluttaisiin hakea kaikki tietoresurssit ja niiden tyypit, tarvittaisiin kolmikko `?resurssi esim:tyyppi ?tyyppi`. Tässäkin esimerkkinä käytetty lintudata on nähtävissä liitteessä 1.

Muuttujia voidaan hakea myös monimutkaisempien yhdistelyjen kautta. Jos esimerkiksi halutaan tehdä kysely, joka palauttaa lahko-tietoresurssin tyypin sinisorsa-tietoresurssin kautta haettuna, voidaan kyselyn `WHERE`-osiossa antaa seuraavat ehdot

```
<http://www.keksittylintusivu.fi/sinisorsa> esim:lahko ?lahko

ja

 ?lahko esim:tyyppi ?tyyppi.
```

Tässä tilanteessa objekti `?tyyppi` saa arvokseen sorsalintujen lahkoa kuvaavan tietoresurssin tyypin.

3.3. Kyselyn lopputulosten esitysmuodot

SPARQL tarjoaa neljä eri kyselytyyppiä, jotta se voisi vastata hyvin siihen haasteeseen, joka syntyy, kun tietoa julkaistaan verkossa hyödyntäen eri sanastoja. Nämä ovat

SELECT-kysely, jolloin tieto palautetaan taulukkomuodossa, CONSTRUCT-kysely, jolloin tulos palautetaan RDF-graafina, ASK-kysely, jolloin palautetaan totuusarvo sekä DESCRIBE-kysely, joka palauttaa RDF-graafin. [Della Valle and Ceri, 2011]

SELECT- ja CONSTRUCT-kyselyt sopivat tilanteisiin, jolloin tietoa haetaan tutuista kohteista ja tiedetään, mitä sanastoja on käytetty niiden kuvailemiseen. CONSTURCT-kyselyn avulla voi myös muuttaa tietolähteen sanaston toiseksi. Silloin kun tietolähde on tuntemattomampi, DESCRIBE-kysely on paikallaan, sillä se palauttaa halutun resurssin kuvattuna RDF-graafina. ASK-kyselyä voidaan käyttää puolestaan silloin, kun halutaan tietää, pystyykö kyseinen kohde (kolmikkovarasto jne.) vastaamaan haluttuun kyselyyn. [Della Valle and Ceri, 2011]

Kyselyjen esittelyssä käytetään aliluvussa 2.3.3 esitettyä otosta lintudatasta. Kun myöhemmin vaaditaan monimutkaisempia kyselyjä, siirrytään käyttämään koko esimerkkidataa (ks. liite 1).

3.3.1. SELECT-kysely

SELECT-kyselyn ollessa kyseessä tuloksena on joukko kolmikkoja, jotka esiintyvät haun kohteena olleessa graafissa ja sisältävät SELECT-osiossa annetut muuttujat. Muuttujat, jotka ovat graafissa, mutta eivät SELECT-osiossa, jätetään pois lopputuloksesta.

Esimerkki 2:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX esim: <http://www.jokinesimerkkisanasto.fi/#>
SELECT  distinct ?subjekti ?predikaatti ?objekti
WHERE
{?subjekti ?predikaatti ?objekti}
```

Esimerkki 2 esittää SELECT-kyselyn. PREFIX-osa kertoo, mitä sanastoja tai ontologioita on käytetty tutkittavassa RDF-graafissa, jotta ohjelma voi tunnistaa ne. SELECT-osassa määritellään, mitä halutaan hakea. Esimerkissä haetaan kaikki kolmikot, eikä WHERE-osiossa ole määritetty mitään erityisiä ehtoja. Tulos on esitetty kuvassa 3.

subjekti	predikaatti	objekti
<http://www.keksittylintusivu.fi/sinisorsa>	esim:suku	<http://www.keksittylintusivu.fi/sorsa_suku>
<http://www.keksittylintusivu.fi/sinisorsa>	esim:heimo	<http://www.keksittylintusivu.fi/sorsa_heimo>
<http://www.keksittylintusivu.fi/sinisorsa>	esim:lahko	<http://www.keksittylintusivu.fi/sorsalinnut>
<http://www.keksittylintusivu.fi/sinisorsa>	esim:siipien_karkivali	"81-95cm"
<http://www.keksittylintusivu.fi/sinisorsa>	esim:pituus	"50-60cm"
<http://www.keksittylintusivu.fi/sinisorsa>	esim:lat_nimi	"Anas platyrhynchos"
<http://www.keksittylintusivu.fi/sinisorsa>	esim:nimi	"Sinisorsa"
<http://www.keksittylintusivu.fi/sinisorsa>	esim:tyyppi	"Lintu"

Kuva 3. Osa SPARQL-kielen SELECT-kyselyn tulosta.

3.3.2. CONSTRUCT-kysely

CONSTRUCT-kysely palauttaa RDF-graafin. Graafi voi olla esitettynä RDF/XML-muodossa, Turtle-muodossa tai millä tahansa muulla esitystavalla, jonka järjestelmä pystyy tuottamaan. Kuten SELECT-kyselyssäkin, CONSTRUC-kyselyn CONSTRUCT-osassa määritellään muuttujat, jotka tulevat lopputulokseen. Lopputulos on graafi, joka on muodostettu tekemällä unioni kaikkien ehdot täyttävien kolmikkojen välillä. Tässäkin tyypissä muuttujat, jotka eivät vastaa ehtoja, jätetään pois lopputuloksesta. [Della Valle and Ceri, 2011]

Toisena käyttötarkoituksena CONSTRUCT-kyselyllä on uusien kolmikkojen muodostaminen vanhojen pohjalta. Tämä mahdollistaa sen, että tietoa voidaan siirtää yhdestä sanastosta toiseen ja sen että olemassa olevien kolmikkojen pohjalta voidaan tuottaa uutta tietoa. [Della Valle and Ceri, 2011]

Esimerkki 3:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX esim: <http://www.jokinesimerkkisanasto.fi/#>
CONSTRUCT {?subjekti ?predikaatti ?objekti}
WHERE
{?subjekti ?predikaatti ?objekti.
FILTER isLiteral(?objekti)}
```

Esimerkissä 3 PREFIX-kohdat ovat samanlaisia kuin SELECT-kyselyssäkin. Tässäkin haetaan kaikki kolmikot samoin kuin SELECT-kyselyn esimerkissä. Ero SELECT- ja CONTSTRUCT-kyselyillä on vain siinä, missä muodossa kysely palauttaa tuloksen. CONSTRUCT-kysely palauttaa graafin, siinä missä SELECT-kysely palauttaa listan kolmikkoja. Tässä esimerkissä on (toisin kuin SELECT-esimerkissä) vastaukseen kelpuutettu vain objektit, joilla on literaaliarvo, jotta saadaan vastauksesta hieman erilainen verrattuna aiempaan esimerkkiin RDF/XML-datasta. Seuraavassa on esimerkki yllä olevan kyselyn tuottamasta graafista sellaisenaan.

```
<rdf:RDF
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:esim="http://www.jokinesimerkkisanasto.fi/#" >
     <rdf:Description rdf:about="http://www.keksittylintu-
sivu.fi/sinisorsa">
          <esim:tyyppi>Lintu</esim:tyyppi>
          <esim:nimi>Sinisorsa</esim:nimi>
          <esim:lat_nimi>Anas platyrhynchos</esim:lat_nimi>
```

```
        <esim:pituus>50-60cm</esim:pituus>
        <esim:siipien_karkivali>81-95cm</esim:siipien_karkivali>
    </rdf:Description>
</rdf:RDF>
```

3.3.3. ASK-kysely

ASK-kyselyt antavat tosi/epätosi-tyyppisiä vastauksia. Niitä voidaan käyttää, kun halutaan tietää, onko jokin tietoresurssi olemassa tai voidaanko ongelmaan saada ratkaisu käsillä olevan aineiston pohjalta. Kuitenkaan varsinaisesta ratkaisusta ei olla kiinnostuneita. [Della Valle and Ceri, 2011]

Esimerkki 4:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX esim: <http://www.jokinesimerkkisanasto.fi/#>
ASK
{<http://www.keksittylintusivu.fi/sinisorsa>
     esim:tyyppi
     'Lintu'}
```

Esimerkissä 4 PREFIX-kohtien funktio on sama kuin SELECT- ja CONTRUCT-kyselyissä. Kysely esittää kysymyksen aineistoon ja kysyy, onko sinisorsan tyyppi lintu. Vastaukseksi tulee tosi tai epätosi, tässä tapauksessa tosi.

3.3.4. DESCRIBE-kysely

DESCRIBE-kyselyt ovat hyödyllisiä tilanteissa, joissa ei ole riittävästi tietoa käsillä olevasta aineistosta, jotta siihen voitaisiin tehdä muunlaisia kyselyitä. Kysely palauttaa aineistoa kuvaavan RDF-graafin. Erona CONSTRUCT-kyselyyn on se, että CONSTRUCT-kysely palauttaa graafin, joka vastaa kyselyssä annettuja ehtoja. DESCRIBE-kyselyssä puolestaan kysely ei palauta ehtoja vastaavaa graafia vaan graafin, joka kuvaa halutut tietoresurssit. Haun kohteena oleva järjestelmä päättää, millaisen graafin se palauttaa. Tämän vuoksi kaksi eri järjestelmää voi palauttaa erilaisen graafin, vaikka aineisto olisikin muuten sama. [Della Valle and Ceri, 2011]

Esimerkkinä DESCRIBE-kyselystä käytetään kyselyä, joka kuvaa kaikki tietolähteet, joista löytyy predikaatti pituus. Kysely on esitettynä esimerkissä 5.

Esimerkki 5:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX esim: <http://www.jokinesimerkkisanasto.fi/#>
DESCRIBE ?subjekti ?objekti
```

```
WHERE
{?subjekti esim:pituus ?objekti}
```

Vastaus tähän kyselyyn saadaan RDF/XML-muodossa. Jena-ohjelmistokehys palauttaa annettuun kyselyyn vastauksena sinisorsaa kuvaavan tietoresurssin, joka on annettu esimerkissä 1. Jena-ohjelmistokehys siis palauttaa niiden tietolähteiden datan sellaisenaan, joista löytyy halutut kolmikot.

3.3.5. Järjestelmien eroista DESCRIBE-kyselyn tapauksessa

Yllä on käytetty Apache Jena-ohjelmistokehystä, joka palauttaa `DESCRIBE`-kyselyn tapauksessa tietolähteet sellaisenaan. Tämä ei kuitenkaan ole ainoa tapa, jolla vastaus voidaan esittää. Tämän vuoksi tehdään sama kysely uudestaan käyttäen Eclipse Rdf4j-ohjelmistokehystä. Kyselyn tulos näyttää seuraavalta:

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

     <rdf:Description
          rdf:about="http://www.keksittylintusivu.fi/sinisorsa">
          <heimo xmlns="http://www.jokinesimerkkisanasto.fi/#"
          rdf:resource="http://www.keksittylintusivu.fi/sorsa_heimo"/>
          <lahko xmlns="http://www.jokinesimerkkisanasto.fi/#"
          rdf:resource="http://www.keksittylintusivu.fi/sorsalinnut"/>
          <lat_nimi xmlns="http://www.jokinesimerkkisanasto.fi/#"
          rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Anas
          platyrhynchos</lat_nimi>
          <nimi xmlns="http://www.jokinesimerkkisanasto.fi/#"
          rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Sini-
          sorsa</nimi>
          <pituus xmlns="http://www.jokinesimerkkisanasto.fi/#"
          rdf:datatype="http://www.w3.org/2001/XMLSchema#string">50-
          60cm</pituus>
          <siipien_karkivali xmlns="http://www.jokinesimerkkisa-
          nasto.fi/#"
          rdf:datatype="http://www.w3.org/2001/XMLSchema#string">81-
          95cm</siipien_karkivali>
```

```
        <suku xmlns="http://www.jokinesimerkkisanasto.fi/#"
        rdf:resource="http://www.keksittylintusivu.fi/sorsa_suku"/>
        <tyyppi xmlns="http://www.jokinesimerkkisanasto.fi/#"
        rdf:data-
        type="http://www.w3.org/2001/XMLSchema#string">Lintu</tyyppi>
    </rdf:Description>

</rdf:RDF>
```

Kuten voidaan huomata, Jena-ohjelmistokehys ja RDF4J-ohjelmistokehys eivät palauta aivan samanlaisia tuloksia. Ensimmäinen huomattava ero on se, että RDF4J palauttaa tuloksen käänteisessä järjestyksessä niin, että `heimo` löytyy ylhäältä ja `tyyppi` alhaalta. Lisäksi RDF4J-ohjelmistokehys ilmaisee predikaattien koko nimiavaruuden (esimerkiksi: `<heimo xmlns="http://www.jokinesimerkkisanasto.fi/#...)` toisin kuin Jena-ohjelmistokehys, joka ilmaisee predikaatin etuliitteen kanssa (`esim:heimo`). Lisäksi RDF4J-ohjelmistokehys, toisin kuin Jena-ohjelmistokehys, kertoo mitä tietotyyppiä kulloinkin käytetään. Esimerkiksi

```
type="http://www.w3.org/2001/XMLSchema#string
```

tarkoittaa tekstityyppistä muuttujaa.

3.4. Esimerkkikysely

Esimerkissä 6 tehdään kysely, jossa valitaan aineistosta kaikkien siinä kuvattujen lintujen lahkot, heimot ja nimet. Huomiotta jätetään linnut, joiden pituus on 12-13,5cm (kirjosieppo). Lisäksi järjestetään tulokset nimen mukaan.

Esimerkki 6:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX esim: <http://www.jokinesimerkkiontologia.fi/#>
SELECT ?lahko ?heimo ?nimi
WHERE
{?lintu esim:tyyppi 'Lintu'.
?lintu esim:lahko ?lahko.
?lintu esim:heimo ?heimo.
?lintu esim:nimi ? nimi.
?lintu esim:pituus ?pituus.
```

```
MINUS {?lintu esim:pituus '12-13,5cm'}}
ORDER BY ?nimi
```

Kyseessä on tavanomainen `SELECT`-kysely, johon on lisätty joitakin toimintoja halutun tuloksen aikaansaamiseksi. Kuten voidaan huomata, `WHERE`-osiossa on useampi ehto valittaville kolmikoille. `MINUS`-operaattorin avulla valitaan, mitä kolmikkoja karsitaan pois tarkastelusta ja `ORDER BY` järjestää tulokset haluttuun järjestykseen. Oletuksena funktiolla on nouseva järjestys A:sta Ö:hön ja se kelpaa tämän esimerkin tarkoitukseen. Tulokseksi saadaan kuvan 4 taulukko. [Harris and Seaborne, 2013]

lahko	heimo	nimi
<http://www.keksittylintusivu.fi/sorsalinnut>	<http://www.keksittylintusivu.fi/sorsa_heimo>	"Laulujoutsen"
<http://www.keksittylintusivu.fi/paivapetolinnut>	<http://www.keksittylintusivu.fi/haukat>	"Saaksi"
<http://www.keksittylintusivu.fi/sorsalinnut>	<http://www.keksittylintusivu.fi/sorsa_heimo>	"Sinisorsa"

Kuva 4. Taulukko kohdan 3.4 kyselyn tuloksista.

4. RDF-tietolähteen käsittely Prolog-kielen valmispredikaateilla

Prolog on logiikkaohjelmointikieli. Prolog-kielestä on tehty useampia toteutuksia, kuten SWI-Prolog ja GNU-Prolog. Tässä työssä käytetään SWI-Prolog-kieltä [SWI-Prolog A, 2016].

Koska Prolog-kielessä käytetään myös termiä predikaatti kuvaamaan ohjelmalle annettuja sääntöjä, tehdään tässä yhteydessä ero Prolog-kielen predikaattien ja RDF-predikaattien välille. Jatkossa RDF-predikaatteja kutsutaan nimellä RDF-predikaatti ja Prologin predikaatteja nimellä predikaatti.

SWI-Prolog sisältää RDF-tiedon käsittelemiseen soveltuvia predikaatteja, jotka pystyvät käsittelemään RDF-kolmikkoja. Tässä työssä käytetään SWI-Prolog-kielen semanttisen verkon kirjastoa SWI-Prolog Semantic Web Library 3.0 [Wielemaker, 2016], josta käytetään jatkossa nimeä Semweb.

Semweb-paketti pitää sisällään kaksi RDF-kolmikkovarastoa, jotka ovat rdf11 ja rdf_db. Molemmat kolmikkovarastot ovat toiminnallisuudeltaan jokseenkin samanlaisia, mutta rdf11 perustuu RDF-kielen uudempaan spesifikaatioon. [Wielemaker, 2016]

Kolmikkovarastoksi valitaan rdf_db-kolmikkovarasto. Tämä versio valittiin, koska rdf11-kolmikkovaraston käytössä esiintyi ongelmia ja tämän työn tarkoituksiin molemmat versiot sopivat yhtä hyvin. Seuraavassa tarkastellaan muutamia hakuun liittyviä valmispredikaatteja semweb-paketista.

4.1. Valmispredikaatit

Rdf_db tarjoaa joukon predikaatteja, joiden avulla voidaan käsitellä RDF-dataa [Wielemaker, 2016]. Tässä esitellään keskeisimpiä. Haku tapahtuu perusmuodossaan `rdf`-predikaatin avulla. `Rdf`-predikaatista on kaksi versiota, joista toisessa on kolme ja toisessa 4 parametria. Nämä ovat `rdf(S, P, O)` ja `rdf(S, P, O, Source)`. `S`, `P` ja `O` tarkoittavat subjektia, RDF-predikaattia ja objektia eli yhtä RDF-kolmikkoa. `Source` puolestaan tarkoittaa graafia, johon haku kohdistuu. Predikaatin arvo on tosi, jos kyseinen kolmikko on olemassa. Subjekti on aina joko tyhjä solmu tai URI, predikaatti on aina URI ja objekti on joko literaali, tyhjä solmu tai URI (se voi myös olla tosi/epätosi-arvo). Jos on annettu myös graafi-parametri, se on URI. [Wielemaker, 2016]

Hakuja voidaan tehdä myös `rdf_has`-predikaatilla. Tämän versiot ovat `rdf_has(S, P, O)` ja `rdf_has(S, P, O, RealPredicate)`. Tämäkin palauttaa toden silloin, kun kyseinen kolmikko on olemassa, mutta ero on siinä, että tämä hyväksyy kaikki RDF-predikaatit, jotka on määritelty RDF-predikaatin `P` alipredikaateiksi. Kuten edelläkin `S`, `P` ja `O` tarkoittavat subjektia, RDF-predikaattia ja objektia. `RealPredicate` on se RDF-

predikaatti, joka määrittelee tämän relaation todeksi, eli jokin RDF-predikaatin `P` alipredikaateista. Koska `rdf_has` hyödyntää RDF-skeeman ominaisuuksia, sitä ei käytetä tässä työssä. [Wielemaker, 2016]

`Rdf_reachable`-predikaatti tutkii, voidaanko objekti saavuttaa subjektista. Tämä predikaatti on muotoa `rdf_reachable(S, P, O)` ja `rdf_reachable(S, P, O, MaxD, D)`. `Rdf_reachable` käyttää `rdf_has`-predikaattia tämän tutkimiseen, sillä se mahdollistaa myös symmetrisyyden ja käänteisyyden (inverse of) tutkimisen. Pidemmässä versiossa on mahdollista tutkia, kuinka monta askelta (`D`) tarvitaan kohteen saavuttamisen. Tämä toteutetaan leveys ensin -hakuna. Lisäksi `MaxD`-muuttujalla voidaan määrittää, kuinka monta askelta voidaan maksimissaan ottaa. Koska tämä predikaatti hyödyntää `rdf_has`-predikaattia, joka puolestaan hyödyntää RDF-skeemaa, sitä ei käytetä tässä työssä. [Wielemaker, 2016]

Näiden lisäksi tarjolla on **`{}`**`(Where)` ja `rdf_where(Where)` -predikaatit. Nämä predikaatit tekevät samat asiat. Näiden predikaattien avulla voidaan tarvittaessa asettaa literaaliarvoille ehtoja, joiden pohjalta rajataan haettavien kolmikkojen joukkoa. [Wielemaker, 2016]

4.2. Esimerkki SWI-Prolog-kielen valmispredikaattien käytöstä

Esimerkissä 7 tehdään lintudataan seuraava kysely:

Esimerkki 7:

```
rdf(Subjekti, Predikaatti, Objekti).
```

Tulokseksi saadaan kolmikkoja muodossa:

```
Subjekti = 'http://www.keksittylintusivu.fi/sinisorsa',
Predikaatti = 'http://www.jokinesimerkkisanasto.fi/#tyyppi',
Objekti = literal('Lintu') ;
Subjekti = 'http://www.keksittylintusivu.fi/sinisorsa',
Predikaatti = 'http://www.jokinesimerkkisanasto.fi/#nimi',
Objekti = literal('Sinisorsa') ;
Subjekti = 'http://www.keksittylintusivu.fi/sinisorsa',
Predikaatti = 'http://www.jokinesimerkkisanasto.fi/#lat_nimi',
Objekti = literal('Anas platyrhynchos') ;
Subjekti = 'http://www.keksittylintusivu.fi/sinisorsa',
Predikaatti = 'http://www.jokinesimerkkisanasto.fi/#pituus',
Objekti = literal('50-60cm') ;
```

```
Subjekti = 'http://www.keksittylintusivu.fi/sinisorsa',
Predikaatti = 'http://www.jokinesimerkkisanasto.fi/#siipien_karkivali',
Objekti = literal('81-95cm') ;
Subjekti = 'http://www.keksittylintusivu.fi/sinisorsa',
Predikaatti = 'http://www.jokinesimerkkisanasto.fi/#lahko',
Objekti = 'http://www.keksittylintusivu.fi/sorsalinnut' ;
Subjekti = 'http://www.keksittylintusivu.fi/sinisorsa',
Predikaatti = 'http://www.jokinesimerkkisanasto.fi/#heimo',
Objekti = 'http://www.keksittylintusivu.fi/sorsa_heimo' ;
Subjekti = 'http://www.keksittylintusivu.fi/sinisorsa',
Predikaatti = 'http://www.jokinesimerkkisanasto.fi/#suku',
Objekti = 'http://www.keksittylintusivu.fi/sorsa_suku'.
```

Kysely palauttaa sinisorsa-tietoresurssin tietoja ilman mitään erityistä suodatusta. Toisin sanoen kyseessä on SPARQL-kielen `SELECT`-kyselyä vastaava kysely. Jos halutaan toteuttaa SPARQL-kielen `ASK`-kyselyä vastaava kysely, niin asetetaan halutut arvot tälle predikaatille (`rdf(subjektin arvo, predikaatin arvo, objektin arvo)` jne.). `CONSTRUCT`-tyyppinen kysely ei vaikuttaisi olevan suoraan mahdollista, mutta predikaatteja hyödyntämällä tai omia predikaatteja määrittelemällä voidaan muodostaa saaduista kolmikoista graafi.

5. RDF/XML-relaatioesitystapa ja sen Prolog-toteutus

Tässä luvussa kehitetään ja esitellään uusi tapa esittää RDF-tietolähde. Kutsutaan tätä tapaa RDF/XML-relaatioksi. RDF/XML-relaatio on Niemen ja Järvelinin [2006] kehittämän XML-relaation laajennus, joka on toteutettu lisäämällä XML-relaation mahdollisuus URI-osoitteiden esittämiseen. XML-relaatiossa on kyse siitä, että jokin XML-dokumentti on muutettu relaatiomuotoon ja RDF/XML-relaatiossa on kyse siitä, että jokin RDF/XML-dokumentti on muutettu relaatiomuotoon. RDF/XML-esitystavan pitäisi tarjota mahdollisuus aineiston monipuolisempaan analysointiin verrattuna SPARQL-kieleen tai SWI-Prologin valmispredikaatteihin. Tässä työssä tutkitaan tämän väitteen paikkansa pitävyyttä.

5.1. XML-relaatio

Niemi ja Järvelin [2006] esittävät tavan muuttaa XML-dokumentti muotoon, joka on yhteensopiva relaatiotietokantojen kanssa. Tätä he kutsuvat XML-relaatioksi. Toisin sanoen Niemi ja Järvelin [2006] osoittavat, että XML-dokumentti voidaan muuttaa XML-relaatioksi ja takaisin menettämättä tietoa.

XML-relaatio on muotoa `D(C, T, I)`. `D` on XML-dokumentin nimi, `C` on jokin komponentti, joka esiintyy dokumentissa `D`. Se voi olla attribuutin tai elementin nimi tai arvo tai arvon yksittäinen osa (esimerkiksi sana arvosta, joka koostuu useista sanoista). `T` on komponentin `C` tyyppi, eli se kertoo, onko kyseessä attribuutin nimi, elementin nimi tai arvo. `I` on indeksi (Dewey-indeksi), joka kertoo komponentin `C` sijainnin dokumentissa `D`. `D` on dokumentin tai relaation nimi, joka voi olla mikä tahansa. [Niemi and Järvelin, 2006]

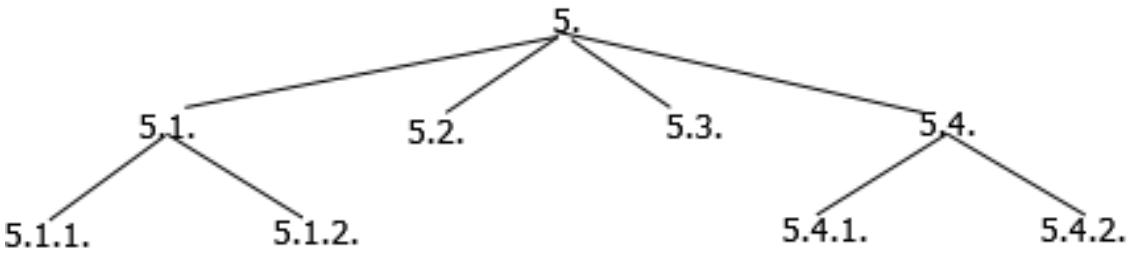
XML-relaatiolla on joitakin eroja relaatiomallin relaatioon verrattuna. XML-relaatio eroaa relaatiomallista siinä, että relaatiomallissa käsitellään rakenteista dataa, kun taas XML-relaatio sisältää puolirakenteista dataa. Lisäksi relaatiomallin relaatiossa voi olla yksi tai useampi attribuutti, mutta XML-relaatiossa niitä on aina kolme (komponentti, tyyppi ja indeksi). Relaatiomallin relaatiossa voidaan myös erottaa selkeästi ekstensionaalinen (olioiden tai entiteettien instanssit) ja intensionaalinen (skeema tai määritelmä) taso toisistaan, sillä relaatiomallin monikot sisältävät vain ekstensionaalista tietoa. XML-relaatiossa komponentit voivat olla sekä ekstensionaalisella että intensionaalisella tasolla. Tasot erotetaan toisistaan komponentin tyypin avulla, siten että elementti- ja attribuuttityyppiset komponentit kuuluvat intensionaaliselle tasolle ja arvotyyppiset komponentit ekstensionaaliselle tasolle. Relaatiomallin relaatiossa avain, joka erottaa monikot toisistaan, koostuu yhdestä tai useammasta attribuutista, mutta XML-relaatiossa indeksi erottaa komponentit ja monikot toisistaan. [Niemi and Järvelin, 2006]

Niemi et al. [2009] esittävät yhdeksi potentiaaliseksi käyttökohteeksi XML-relaatiolle useista XML-tietolähteistä saatavan tiedon harmonisoinnin. Tiedon harmonisointi tarkoittaa sitä, että luodaan eri lähteistä kerätylle tiedolle yhdenmukainen esitystapa.

5.1.1. Dewey-indeksi

Dewey-indeksointi perustuu kirjastoissa käytettyyn Deweyn kymmenjärjestelmään (Dewey Decimal Classifiaction). Pyrkimyksenä Dewey-indeksissä on luokitella aineisto yksinkertaisesti ja yksiselitteisesti siten, että aineiston löytäminen olisi mahdollisimman yksinkertaista. Tämän aikaansaamiseksi indeksinä käytettiin arabialaisianumeroita (tutut symbolit 1,2,3,4,5,6,7,8,9,0), jotka ovat yksinkertaisimpia ihmisen tuntemia symboleja. [Dewey 1922]

Dewey-indeksiä käytetään hierarkioiden kuvaamiseen. Esimerkiksi tähän alilukuun voidaan viitata indeksillä 5.1.1., jota voidaan pitää Dewey-indeksinä. Piste tai pilkku numerojen välissä tarkoittaa, että oikealla puolella oleva luku on suhteessa vasemmalla puolella olevaan. Esimerkiksi luku 5.1.1. on suhteessa lukuun 5.1., joka on suhteessa lukuun 5. Kuvassa 5 on esitetty tämän työn luku 5. puurakenteena. Tämän työn Prolog-esimerkeissä käytetään Prolog-kielen syntaksista johtuen pilkkua, mutta muissa esimerkeissä käytetään pistettä.



Kuva 5. Tämän tutkielman luku 5. esitettynä Dewey-indeksinä.

Arvola et al. [2005] hyödyntävät Dewey-indeksiä kehittämässään XML-dokumentteihin tehtävässä tiedonhakumenetelmässä. Dewey-indeksiä käytetään menetelmässä identifioimaan eri elementit ja osoittamaan niiden väliset suhteet (näiden suhteiden perusteella voi tehostaa tiedonhakua XML-dokumentista). Lisäksi Dewey-indeksin käyttö mahdollistaa dokumenttien uudelleenjärjestelyn. Dewey-indeksiä on myös käytetty esittämään XML-dokumentteja ja niihin tehtäviä kyselyjä relaatiomallin ja SQL-kielen kanssa yhteensopivassa muodossa [Tatarinov et al., 2002].

5.1.2. Rakentaja-algebra XML-relaation esittämiseksi

Niemi ja Järvelin [2006], määrittelevät rakentaja-algebran, jonka avulla on mahdollista muuttaa XML-dokumentti XML-relaatioksi. Algebrassa XML-relaatio koostetaan perusosista ja muodostetaan yhä mutkikkaampi rakenne. He antavat kuusi sääntöä, joiden avulla XML-relaatio voidaan muodostaa. Ensin kuitenkin määritellään muutama operaatio, joita säännöissä käytetään ja sen jälkeen määritellään itse säännöt.

Säännöissä käytetyt operaatiot ovat *len(i), index_transformation(Int, R)* ja *maxfirst(R).* Lisäksi indeksi ilmaistaan merkinnällä *<i|tail>,* jossa *i* on indeksin ensimmäinen elementti ja *tail* tarkoittaa indeksin loppuosaa (jos indeksissä on pelkästään yksi elementti, niin silloin siinä ei ole *tail*-osaa). *Len(i)* laskee indeksin *i* pituuden. *Index_transformation(Int, R)* uudelleenindeksoi relaation *R* kolmikot lisäämällä kokonaisluvun *Int* kunkin kolmikon indeksin ensimmäiseen elementtiin:

index_transformation (int, R) = {(c, t, <i + int | tail>) | (c, t, <i | tail>) ∈ R}.

Maxfirst(R) puolestaan laskee niiden indeksien lukumäärän relaatiossa *R*, joiden pituus on 1;

maxfirst(R) = | {(c, t, i)|(c, t, i) ∈ R: len(i)=1} |. [Niemi ja Järvelin, 2006]

Sääntö 1: Olkoon *c* attribuutin tai elementin arvo. Jos käsillä oleva arvo koostuu useammista sanoista, niin *c* viittaa yhteen niistä sanoista. Näissä tapauksissa *c* esitetään XML-relaationa *{(c, 'v', <1>)}*. Kolmikossa v osoittaa, että c on arvo tai arvon osa. [Niemi ja Järvelin, 2006]

Sääntö 2: Attribuuttinimi *an* esitetään XML-relaationa *{(an, 'a', <1>)}*. Kolmikossa a osoittaa, että an on attribuuttinimi. [Niemi ja Järvelin, 2006]

Sääntö 3: Elementtinimi *en* esitetään XML-relaationa *{(en, 'e', <1>)}*. Kolmikossa e osoittaa, että en on elementtinimi. [Niemi ja Järvelin, 2006]

Sääntö 4: Jos R_1 ja R_2 ovat kaksi XML-relaatiota, niin konkatenaatiorakentaja $R_1 <> R_2$ rakentaa XML-relaation $R_1 \cup$ *index_transformation (maxfirst(*R_1*),* R_2*)*. Tässä yhteydessä on syytä huomata, että merkintä <> ei tarkoita tyhjää indeksiä, vaan sitä käytetään merkitsemään konkatenaatiorakentajaa. [Niemi ja Järvelin, 2006]

Sääntö 5: Jos *A* kuvaa attribuuttinimeä XML-relaationa (katso sääntö 2) ja *R* sen sisältöä XML-relaationa, niin attribuuttirakentaja, joka esitetään seuraavasti $A\ \theta\ R$, rakentaa XML-relaation $A\ \cup\ \{(c, 'v', <1 \mid i>)\}\ |(c, 'v', i) \in R\}$. Toisin sanottuna R-relaation jokaisen indeksin pituutta kasvatetaan yhdellä lisäämällä luku '1' jokaisen indeksin ensimmäiseksi komponentiksi. [Niemi ja Järvelin, 2006]

Sääntö 6: Jos *E* kuvaa elementtinimen XML-relaationa (katso sääntö 3) ja *R* sen sisällön XML-relaationa niin elementtirakentaja, joka esitetään $E\ \omega\ R$, rakentaa XML-relaation $E\ \cup\ \{(c, t, <1 \mid i>) \mid (c, t, i) \in R\}$, missä $t \in \{'e', 'a', 'v'\}$. [Niemi ja Järvelin, 2006]

5.2. RDF/XML-relaatio

RDF/XML-relaatio on esitystapa RDF-tiedolle. RDF/XML-relaatio saadaan, kun XML-relaatiota laajennetaan siten, että siihen lisätään URI-tyyppi. Tämä tapahtuu lisäämällä uusi relaatiomuoto jo olemassa olevien muotojen (elementti, attribuutti ja arvo) lisäksi. Tämä lisättävä relaatio on URI. Sen sisältönä on URI-osoite, tyyppi on URI eli u ja indeksi kuten muissakin tyypeissä. URI-relaatio näyttää siis tältä: `D(URI, u, I)`. Lisäksi tämä vaatii uuden säännön lisäämistä XML-relaation rakentaja-algebraan. Tarvittava sääntö perustuu aliluvussa 5.1.2. esitettyyn sääntöön yksi, mutta sillä erolla, että tavanomaisen arvon sijasta annetaan URI-osoite. Sääntö näyttää seuraavalta:

Sääntö 7: Olkoon *c* attribuutin tai elementin arvo, joka on URI-osoite. Esitetään *c* XML-relaationa $\{(c, 'u', <1>)\}$. Kolmikossa *u* osoittaa, että *c* on URI-osoite.

RDF-kolmikko ilmaistaan RDF/XML-relaation avulla seuraavasti. Aluksi annetaan kolmikon subjekti, joka koostuu kolmesta relaatiosta. Ensiksi on tavallinen elementtirelaatio, joka kertoo, että kyseessä on `rdf:description` eli `D(rdf_description, e, indeksi)`, seuraavaksi on attribuutti, joka kertoo, että tästä osoitteesta löytyy halutun subjektin kuvaus eli `D(rdf_about, a, indeksi)` ja lopuksi tulee haluttu osoite `D(URI, u, indeksi)`.

Predikaatti on tavallinen elementti, jos on määritelty käytetyille ontologioille lyhenteet. Jos taas lyhenteitä ei ole määritelty, voidaan käyttää myös URI-tyyppiä. Objektina voi olla elementti, jonka arvona on tavallinen v-tyyppinen arvo tai elementti, jonka `rdf:resource`-attribuutin arvona on URI-osoite.

5.3. RDF/XML-relaatio esimerkki

Muutetaan aliluvun 2.3.4. RDF/XML esimerkki RDF/XML-relaatioksi käyttämällä edellä esitettyä rakentaja-algebraa. Alun nimiavaruudet jätetään pois esimerkin yksinkertaistamiseksi. Aluksi esitetään kaikki tieto yksittäisinä RDF/XML-relaatiomuodossa olevina dokumentteina (kaikkien indeksinä on 1) käyttämällä sääntöjä 1, 2, 3 ja 7 riippuen siitä, onko kyseessä elementti, attribuutti, arvo vai URI.

Haetaan aluksi arvot käyttämällä sääntöä 1. Dokumentista voidaan löytää seuraavat arvot:

V1 = {(Lintu, 'v', <1>)}
V2 = {(Sinisorsa, 'v', <1>)}
V3 = {(Anas platyrhynchos, 'v', <1>)}
V4 = {(50-60cm, 'v', <1>)}
V5 = {(81-95cm, 'v', <1>)}

Seuraavaksi haetaan attribuutit käyttäen sääntöä 2. Löydetään seuraavat attribuutit:

A1 = {(rdf:about, 'a', <1>)}
A2 = {(rdf:resource, 'a', <1>)}

Käytetään sääntöä 3 elementtien hakemiseen. Löydetään seuraavat elementit:

E1 = {(rdf:Description, 'e', <1>)}
E2 = {(esim:tyyppi, 'e', <1>)}
E3 = {(esim:nimi, 'e', <1>)}
E4 = {(esim:lat_nimi, 'e', <1>)}
E5 = {(esim:pituus, 'e', <1>)}
E6 = {(esim:siipien_karkivali, 'e', <1>)}
E7 = {(esim:lahko, 'e', <1>)}
E8 = {(esim:heimo, 'e', <1>)}
E9 = {(esim:suku, 'e', <1>)}

Käytetään vielä sääntöä 7 URI-osoitteiden hakemiseen. Saadaan seuraavat relaatiot:

U1 = {(http://www.keksittylintusivu.fi/sinisorsa, 'u', <1>)}
U2 = {(http://www.keksittylintusivu.fi/sorsalinnut, 'u', <1>)}
U3 = {(http://www.keksittylintusivu.fi/sorsa_heimo, 'u', <1>)}

U4 = {(http://www.keksittylintusivu.fi/sorsa_suku, ’u’, <1>)}

Seuraavaksi kootaan näistä kokonainen dokumentti käyttämällä sääntöjä 4, 5 ja 6. Saadaan seuraava rakentaja-algebran sekvenssi:

E1 ω ((A1 θ U1) <> (E2 ω V1) <> (E3 ω V2) <> (E4 ω V3) <> (E5 ω V4) <> (E6 ω V5) <> (E7 ω ((A2 θ U2)) <> (E8 ω ((A2 θ U3)) <> (E9 ω (A2 θ U4))).

Jaetaan seuraavaksi saatu sekvenssi osiin, jotta sen sisältöä on helpompi tarkastella. Osat ovat seuraavat:

O1 = E9 ω (A2 θ U4)
O2 = E8 ω (A2 θ U3)
O3 = E7 ω (A2 θ U2)
O4 = E6 ω V5
O5 = E5 ω V4
O6 = E4 ω V3
O7 = E3 ω V2
O8 = E2 ω V1
O9 = A1 θ U1
O10 = O9 <> O8 <> O7 <> O6 <> O5 <> O4 <> O3 <> O2 <> O1
O11 = E1 ω O10

Näiden tuottamat relaatiot ovat seuraavat:

O1 = {(esim:suku, ’e’, <1>), (rdf:resource, ’a’, <1. 1>), (http://www.keksittylintusivu.fi/sorsa_suku, ’u’, <1. 1. 1>)}

O2 = {(esim:heimo, ’e’, <1>), (rdf:resource, ’a’, <1. 1>), (http://www.keksittylintusivu.fi/sorsa_heimo, ’u’, <1. 1. 1>)}

O3 = {(esim:lahko, ’e’, <1>), (rdf:resource, ’a’, <1. 1>), (http://www.keksittylintusivu.fi/sorsalinnut, ’u’, <1. 1. 1>)}

O4 = {(esim:siipien_karkivali, ’e’, <1>), (81-95cm, ’v’, <1. 1>)}

O5 = {(esim:pituus, ’e’, <1>), (50-60cm, ’v’, <1. 1>)}

O6 = {(esim:lat_nimi, ’e’, <1>), (Anas platyrhynchos, ’v’, <1. 1>)}

O7 = {(esim:nimi, ’e’, <1>), (Sinisorsa, ’v’, <1. 1>)}

O8 = {(esim:tyyppi, ’e’, <1>), (Lintu, ’v’, <1. 1>)}

O9 = {(rdf:about, ’a’, <1>), (http://www.keksittylintusivu.fi/sinisorsa, ’u’, <1. 1>)}

O10 = {(rdf:about, ’a’, <1>), (http://www.keksittylintusivu.fi/sinisorsa, ’u’, <1. 1>), (esim:tyyppi, ’e’, <2>), (Lintu, ’v’, <2. 1>), (esim:nimi, ’e’, <3>), (Sinisorsa, ’v’, <3. 1>), (esim:lat_nimi, ’e’, <4>), (Anas platyrhynchos, ’v’, <4. 1>), (esim:pituus, ’e’, <5>), (50-60cm, ’v’, <5. 1>), (esim:siipien_karkivali, ’e’, <6>), (81-95cm, ’v’, <6. 1>), (esim:lahko, ’e’, <7>), (rdf:resource, ’a’, <7. 1>), (http://www.keksittylintusivu.fi/sorsalinnut, ’u’, <7. 1. 1>), (esim:heimo, ’e’, <8>), (rdf:resource, ’a’, <8. 1>), (http://www.keksittylintusivu.fi/sorsa_heimo, ’u’, <8. 1. 1>), (esim:suku, ’e’, <9>), (rdf:resource, ’a’, <9. 1>), (http://www.keksittylintusivu.fi/sorsa_suku, ’u’, <9. 1. 1>)}

O11 = {(rdf:Description, ’e’, <1>), (rdf:about, ’a’, <1. 1>), (http://www.keksittylintusivu.fi/sinisorsa, ’u’, <1. 1. 1>), (esim:tyyppi, ’e’, <1. 2>), (Lintu, ’v’, <1. 2. 1>), (esim:nimi, ’e’, <1. 3>), (Sinisorsa, ’v’, <1. 3. 1>), (esim:lat_nimi, ’e’, <1. 4>), (Anas platyrhynchos, ’v’, <1. 4. 1>), (esim:pituus, ’e’, <1. 5>), (50-60cm, ’v’, <1. 5. 1>), (esim:siipien_karkivali, ’e’, <1. 6>), (81-95cm, ’v’, <1. 6. 1>), (esim:lahko, ’e’, <1. 7>), (rdf:resource, ’a’, <1. 7. 1>), (http://www.keksittylintusivu.fi/sorsalinnut, ’u’, <1. 7. 1. 1>), (esim:heimo, ’e’, <1. 8>), (rdf:resource, ’a’, <1. 8. 1>), (http://www.keksittylintusivu.fi/sorsa_heimo, ’u’, <1. 8. 1. 1>), (esim:suku, ’e’, <1. 9>), (rdf:resource, ’a’, <1. 9. 1>), (http://www.keksittylintusivu.fi/sorsa_suku, ’u’, <1. 9. 1. 1>)}

Esimerkissä 8 ilmaistaan tämä vielä Prolog-esityksenä. Dokumentin nimenä on tässä työssä d. Nimi on valittu yksinkertaisesti sen vuoksi, että se on sama, jota on käytetty aiemmin XML-relaation esittelyssä. Dokumentin nimi voisi olla myös esimerkiksi linnut. Lintudata RDF/XML-relaatio-muodossa on esitetty kokonaisuudessaan liitteessä 2. Indeksit toteutetaan Prologissa listoina. Tämän vuoksi indeksit ovat hakasulkeissa, kun RDF/XML-relaatio esitetään Prolog-muodossa.

Esimerkki 8:

```
d(rdf_description, e, [1]).
d(rdf_about, a, [1, 1]).
d("http://www.keksittylintusivu.fi/sinisorsa", u , [1, 1, 1]).
d(esim_tyyppi, e, [1, 2]).
d("lintu", v, [1, 2, 1]).
d(esim_nimi, e, [1, 3]).
d("sinisorsa", v, [1, 3, 1]).
d(esim_latnimi, e, [1, 4]).
d("anas platyrhynchos", v, [1, 4, 1]).
d(esim_pituus, e, [1, 5]).
d("50-60cm", v, [1, 5, 1]).
d(esim_siipien_karkivali, e, [1, 6]).
d("81-95cm", v, [1, 6, 1]).
d(esim_lahko, e, [1, 7]).
d(rdf_resource, a, [1, 7, 1]).
d("http://www.keksittylintusivu.fi/sorsalinnut", u, [1, 7, 1, 1]).
d(esim_heimo, e, [1, 8]).
d(rdf_resource, a, [1, 8, 1]).
d("http://www.keksittylintusivu.fi/sorsa_heimo", u, [1, 8, 1, 1]).
d(esim_suku, e, [1, 9]).
d(rdf_resource, a, [1, 9, 1]).
d("http://www.keksittylintusivu.fi/sorsa_suku", u, [1, 9, 1, 1]).
```

Ensimmäisenä on `rdf_description`, joka kertoo, että kyseessä on tietoresurssin kuvaus. Tämä saa indeksikseen `[1],` joka kertoo sen sijaitsevan dokumentin alussa. Tässä on kyseessä elementti niin kuin kirjain e kertoo ja esimerkistä 1 voidaan nähdä. Tällä on attribuutti (a) `rdf_about`, joka kertoo mitä tietoresurssia kuvataan ja joka saa indeksik-

seen `[1, 1]`, jolloin tiedetään, että se on osa indeksin `[1]` saanutta tietoresurssia. Lopuksi `rdf_about`-attribuutilla on arvona sinisorsan URI ja indeksinä `[1, 1, 1]`, jotta tiedetään että se on `[1, 1]`-kohdan attribuutin arvo. Tämä on siis kolmikon subjekti.

RDF-predikaatit ovat elementtejä. Niin kuin esimerkiksi `esim_nimi`. Tämä on saanut indeksikseen `[1, 3]`, jolloin nähdään, että se liittyy tietoresurssiin `[1]`, muttei kuitenkaan suoraan sen attribuuttiin `[1, 1]`. Elementtiin voi liittyä joko arvo `v` tai URI `u`. Esimerkkinä ovat indeksit `[1, 2, 1]` sekä `[1, 7, 1]` ja `[1, 7, 1, 1]` Ensimmäisessä tapauksessa kyseessä on `v`-tyyppinen arvo. Jälkimmäisessä tapauksessa taas on ensin attribuutti `rdf_resource` indeksillä `[1, 7, 1]`, joka kertoo, mikä tietoresurssi sisältää halutun tiedon. Indeksillä `[1, 7, 1, 1]` taas on haluttu URI, `u`-tyyppinen arvo.

5.4. Esimerkkikyselyjä

Tässä osassa demonstroidaan RDF/XML-relaatiomuodossa esitettyyn dataan tehtäviä hakuja. Toteutetaan ensin tavanomainen poimi ja valitse -kysely ja sen jälkeen analyyttinen kysely, joka tutkii, mitä tietoa on ilmaistu annetussa tietolähteessä eksplisiittisesti eli literaaleina.

5.4.1. Poimi ja valitse -kysely

Toteutetaan poimi ja valitse -kyselyjen tekoon `select`-predikaatti, joka valitsee määritellyt tietolähteet aineistosta. Predikaatti on muotoa `select(S, P, O)`, jossa `S` on kolmikon subjekti, `P` RDF-predikaatti ja `O` objekti. Jos predikaatille määrittelee jokaisen muuttujan, se tutkii, onko kyseistä kolmikkoa olemassa. Jos jättää antamatta muuttujan, niin predikaatti palauttaa ne kolmikot, joissa määritellyt muuttujat esiintyvät. Jos mitään muuttujista ei määritellä, predikaatti palauttaa kaikki kolmikot. Predikaatti on ohjelmoitu seuraavasti:

```
select(S, P, O) :-
   d(O1, v, Oind),
   im_predecessor(Oind, Pind),
   d(P, _, Pind),
   im_predecessor(Pind, Sind),
   append(Sind, [1, 1], Sind_final),
   d(S1, u, Sind_final),
   atom_codes(S, S1),
   atom_codes(O, O1).

select(S, P, O) :-
   d(O1, u, Oind),
```

```
    im_predecessor(Oind, Ind1),
    im_predecessor(Ind1, Ind2),
    d(rdf_about, _, Ind1),
    d(rdf_description, _, Ind2),
    im_predecessor(Ind2, Pind),
    d(P, _, Pind),
    im_predecessor(Pind, Sind),
    append(Sind, [1, 1], Sind_final),
    d(S1, u, Sind_final),
    atom_codes(S, S1),
    atom_codes(O, O1).

select(S, P, O) :-
    d(O1, u, Oind),
    im_predecessor(Oind, Ind1),
    im_predecessor(Ind1, Pind),
    \+ d(rdf_description, _ , Pind),
    d(P, _, Pind),
    im_predecessor(Pind, Sind),
    append(Sind, [1, 1], Sind_final),
    d(S1, u, Sind_final),
    atom_codes(S, S1),
    atom_codes(O, O1).
```

Predikaatista on toteutettu kolme versiota, koska URI- (`u`) ja tavallisten literaali (`v`) -tyyppisten objektien löytäminen vaatii hieman erilaista toiminnallisuutta ja ne määritellään aineistossa hieman eri tavalla. Lisäksi vaaditaan hieman erilaista toiminnallisuutta URI-osoitteiden kanssa toimittaessa riippuen siitä, liittyykö URI `rdf:resource`-merkintään vai `rdf:description`-merkintään. Tämän vuoksi ensimmäisenä haetaan relaatiot, jotka sisältävät objektin, jolloin huomataan, että kummasta tyypistä on kyse ja näin ollen vältetään turhaa työtä ja saadaan koodista hieman tehokkaampaa. `Pind` ja `Oind` tarkoittavat predikaatin ja objektin indeksiä. `Ind1` ja `Ind2` ovat vain välietappeja siirryttäessä objektista predikaattiin. `Sind` on indeksi, joka kertoo, mikä on luetun tietoresurssin ensimmäinen indeksi. `Sind_final`, joka saadaan lisäämällä `Sind`-indeksin perään indeksi `[1, 1]`, on subjektin URI-osoitteen indeksi ja se indeksi, joka palautetaan. Koska Prolog käsittelee tekstiä ASCII-koodina, tarvitaan Prologin `atom_codes`-predikaattia muuttamaan haluttu arvo luettavampaan muotoon, eli `O1` ja `S1` muutetaan arvoiksi `O` ja `S`.

Seuraavaksi haetaan objektia edeltävä RDF-predikaatti nousemalla ylös hierarkiassa Niemen ja Järvelinin [2006] kehittämän `im_predecessor`-predikaatin (ks. liite 3) avulla. Predikaatti `im_predecessor` hakee siis relaation, joka on yhden indeksin käsiteltävänä olevaa relaatiota ylempänä [Niemi and Järvelin, 2006]. URI-osoitteiden tapauksessa katsotaan lisäksi, kumpi tilanne on kyseessä. Tämä tapahtuu katsomalla, että löytyykö kuvauksesta `rdf_description` vai ei. Lopuksi haetaan kolmikon subjekti. Tämä tehdään siten, että haetaan subjektin `rdf_description`-elementin indeksi ja lisätään sen perään indeksi `[1, 1]`, jolloin saadaan otettua subjekti mistä tahansa kohtaa kuvausta.

Lisäksi on huomautettava, että tämä esimerkki (ja kaikki muut tämän työn RDF/XML-relaatio-esimerkit) hakee suoraan relaatioita, joiden nimi on `d`. Tämän työn tarkoitusperiin nähden tämä on täysin riittävä ratkaisu. Jos oltaisiin kehittämässä varsinaiseen käyttöön tarkoitettua järjestelmää, pitäisi koodin olla sellainen, että kaikki relaatiot, nimestä riippumatta, olisivat käsiteltävissä.

Aiempien kohtien `select`-esimerkkiä käyttämällä saadaan esimerkin 9 kysely. Kysely on muotoa

Esimerkki 9:

```
select(Subjekti, Predikaatti, Objekti)
```

ja se palauttaa seuraavan tuloksen:

```
Subjekti = 'http://keksittylintusivu.fi/sinisorsa',
Predikaatti = esim_tyyppi,
Objekti = lintu ;
Subjekti = 'http://keksittylintusivu.fi/sinisorsa',
Predikaatti = esim_nimi,
Objekti = sinisorsa ;
Subjekti = 'http://keksittylintusivu.fi/sinisorsa',
Predikaatti = esim_latnimi,
Objekti = 'anas platyrhynchos' ;
Subjekti = 'http://keksittylintusivu.fi/sinisorsa',
Predikaatti = esim_pituus,
Objekti = '50-60cm' ;
Subjekti = 'http://keksittylintusivu.fi/sinisorsa',
Predikaatti = esim_siipien_karkivali,
Objekti = '81-95cm' ;
Subjekti = 'http://keksittylintusivu.fi/sinisorsa',
```

```
Predikaatti = esim_lahko,
Objekti = 'http://www.keksittylintusivu.fi/sorsalinnut' ;
Subjekti = 'http://keksittylintusivu.fi/sinisorsa',
Predikaatti = esim_heimo,
Objekti = 'http://www.keksittylintusivu.fi/sorsa_heimo' ;
Subjekti = 'http://keksittylintusivu.fi/sinisorsa',
Predikaatti = esim_suku,
Objekti = 'http://www.keksittylintusivu.fi/sorsa_suku' ;
```

Esimerkissä 10 on esitetty toinen poimi ja valitse -esimerkki, joka palauttaa kaikki linnut, joiden nimi on sinisorsa. Kysely on muotoa

Esimerkki 10:

```
select(Lintu, esim_nimi, sinisorsa).
```

Lintu on siis subjekti ja tässä yhteydessä määrittämätön muuttuja, jolloin saamme kaikki linnut joiden nimi (`esim_nimi`-predikaatti) on sinisorsa (objekti, literaali). Tulokseksi saamme:

```
Lintu = 'http://keksittylintusivu.fi/sinisorsa' ;
```

Tuloksena nähdään kyselyn palauttamien lintujen (tässä tapauksessa vain yhden linnun) URI-osoitteet. Muissa tapauksissa URI-osoitteita saattaisi olla useampiakin.

5.4.2. Analyyttinen kysely

Analyyttisissä kyselyissä pyritään analysoimaan haluttua aineistoa. Työn oletuksena on se, että tämä on se osuus, jossa RDF/XML-relaatio erottuu edukseen verrattuna vertailukohteisiin. Tähän vertailuun palataan luvussa 6.

Tässä osiossa annetaan esimerkki analyysista, johon RDF/XML-relaatio sopii erinomaisesti. Kehitetään predikaatti `select_literals`, joka tutkii, millaista tietoa aineistossa on esitetty eksplisiittisesti v-tyyppisinä literaaleina.

Predikaatti `select_literals` toimii kuten aliluvun 5.4.1. `select`-predikaatti, jonka muuttujia ei ole määritetty, eli se palauttaa kaikki kolmikot. Tulosta suodatetaan kuitenkin siten, että otetaan mukaan vain v-tyyppiset objektit. Predikaatista tulee seuraavanlainen:

```
select_literals(S, P, O) :-
```

```
    d(O1, v, [H|T]),
    im_predecessor([H|T], Ind1),
    d(P, _, Ind1),
    d(S1, u, [H, 1, 1]),
    atom_codes(S, S1),
    atom_codes(O, O1).
```

Predikaatti antaa tulokseksi kolmikkoja. Esimerkissä 11 tehdään kysely `select_literals`-predikaattia käyttäen. Kysely näyttää seuraavalta:

Esimerkki 11:

```
select_literals(Subjekti, Predikaatti, Objekti).
```

Tuloksena saadaan seuraavat kolmikot:

```
Subjekti = 'http://keksittylintusivu.fi/sinisorsa',
Predikaatti = esim_tyyppi,
Objekti = lintu ;
Subjekti = 'http://keksittylintusivu.fi/sinisorsa',
Predikaatti = esim_nimi,
Objekti = sinisorsa ;
Subjekti = 'http://keksittylintusivu.fi/sinisorsa',
Predikaatti = esim_latnimi,
Objekti = 'anas platyrhynchos' ;
Subjekti = 'http://keksittylintusivu.fi/sinisorsa',
Predikaatti = esim_pituus,
Objekti = '50-60cm' ;
Subjekti = 'http://keksittylintusivu.fi/sinisorsa',
Predikaatti = esim_siipien_karkivali,
Objekti = '81-95cm' ;
```

Kuten voidaan huomata, yksikään objekti ei ole URI-osoite, eli kysely on palauttanut pelkästään `v`-tyyppisiä literaaleja.

6. Eri lähestymistapojen vertailu

Tässä luvussa laaditaan lintudataan muutama kysely ja katsotaan, miten ne voidaan toteuttaa eri lähestymistavoilla silloin, kun kysely on toteutettavissa kyseisellä lähestymistavalla. Erityisesti tutkitaan tämän työn oletusta siitä, että RDF/XML-relaation avulla olisi, toisin kuin muilla vertailun lähestymistavoilla, mahdollista toteuttaa analyyttinen kyselytyyppi.

Testataan kolme erilaista kyselytyyppiä. Nämä ovat poimi ja valitse -kyselyt, aggregointikyselyt sekä analyyttiset kyselyt. Testataan, miten nämä voidaan toteuttaa tarkasteltavilla lähestymistavoilla ja katsotaan, mitä näiden toteuttaminen vaatii käyttäjältä ja onko kyseinen toiminto ylipäätään toteutettavissa kyseisellä lähestymistavalla.

6.1. Poimi ja valitse -kysely

Poimi ja valitse -kyselyissä haetaan aineistosta jotakin tiettyä kolmikkoa tai tiettyjä kolmikkoja. Tämä tyyppi on tietokantoihin tehtävistä kyselyistä kaikkein yksinkertaisin ja se löytyy todennäköisesti kaikista tietokannoista. Näissä kyselyissä valitaan aineistosta ne osat, jotka vastaavat kyselyssä annettuja ehtoja. Katsotaan, miten kyselytyypin voi toteuttaa valituissa esimerkeissä. Ensimmäisessä esimerkkikyselyssä haetaan lintudatasta (liite 1) kaikki sorsalintujen lahkoon kuuluvat linnut. Toisessa esimerkkikyselyssä tehdään hieman monimutkaisempi kysely, jossa valitaan kaikki subjektit, joiden tyyppi on lintu ja pituus on 52-60cm. Lopuksi tehdään vielä monimutkaisempi kysely, jossa valitaan tietoresurssit, joiden latinan kielinen nimi on ilmaistu joko `lat_nimi` tai `latNimi` -RDF-predikaatilla.

6.1.1. Poimi ja valitse -kysely SPARQL-kielellä

Poimi ja valitse -kyselyjä voidaan SPARQL-kielessä toteuttaa `SELECT`- ja `CONSTRUCT`-kyselytyypeillä. Tässä yhteydessä valitaan `SELECT`-kysely, sillä se on näistä kahdesta kenties tunnetumpi ja ero `CONSTRUCT`-kyselyyn on lähinnä kyselyjen palauttaman datan esittämisessä. SPARQL-kyselyksi tulee esimerkin 12 kysely. Kysely on seuraavanlainen:

Esimerkki 12:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX esim: <http://www.jokinesimerkkisanasto.fi/#>"
SELECT  distinct ?lintu
WHERE
{?lintu esim:lahko <http://www.keksittylintusivu.fi/sorsalinnut>}
```

Kyselyssä siis valitaan linnut (`SELECT distinct ?lintu`), joiden lahkona (`esim:lahko`) on sorsalinnut (`'http://www.keksittylintusivu.fi/sorsalinnut'`).

`Distinct`-lisäys varmistaa, ettei samaa lintua listata useampaan kertaan, joskin tässä datassa ei sitä välttämättä tarvittaisi.

Tämä on toteutettavissa SPARQL-kielellä helposti, mikä on odotettua, sillä tämäntyyppiset kyselyt ovat kaikkein tavanomaisin kyselytyyppi. Toisekseen tällainen toiminnallisuus on niin keskeinen osa kaikkia tietokantajärjestelmiä, että on vaikea kuvitella kyselykieltä ilman sitä.

Toisessa esimerkissä annetaan `WHERE`-osioon hieman monimutkaisemmat ehdot, jotta saadaan haluttu tulos. Kysely on annettu esimerkissä 13. Kysely on seuraavanlainen:

Esimerkki 13:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX esim: <http://www.jokinesimerkkisanasto.fi/#>
SELECT  ?subjekti
WHERE
{?subjekti esim:tyyppi 'Lintu' .
?subjekti esim:pituus '52-60cm'}
```

Tämä on siis toteutettavissa varsin yksinkertaisesti. Tulokseksi tulee:

```
-----------------------------------------
| subjekti                              |
=========================================
| <http://www.keksittylintusivu.fi/saaksi> |
-----------------------------------------
```

Viimeisessä esimerkissä voidaan hyödyntää `UNION`-avainsanaa, joka toteuttaa unioni-operaation. Unioni-operaatiolla SPARQL pystyy yhdistämään kahden kyselyn tuottamat tulokset yhdeksi vastaukseksi. Toisin sanoen se vertaa haun kohteena olevia kolmikkoja kahteen tai useampaan kyselyssä annettuun määrittelyyn ja kelpuuttaa vastaukseksi kaikki kolmikot, jotka vastaavat ainakin yhtä määrittelyä. [Della Valle and Ceri, 2011]

Esimerkissä 14 tehdään seuraavanlainen kysely:

Esimerkki 14:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX esim: <http://www.jokinesimerkkisanasto.fi/#>"
SELECT  ?subjekti
WHERE {
{?subjekti esim:lat_nimi ?objekti .}
UNION
```

```
{?subjekti esim:latNimi ?objekti .}
}
```

Voidaan huomata, että kyselyssä yhdistetään kahden eri määritelmän tuottamat vastaukset toisiinsa. Ensin haetaan kolmikot, joista löytyy RDF-predikaatti `lat_nimi` ja sitten ne kolmikot, joista löytyy RDF-predikaatti `latNimi`. Vastaus on seuraavanlainen:

```
-------------------------------------------------
| subjekti                                      |
=================================================
| <http://www.keksittylintusivu.fi/sorsalinnut>  |
| <http://www.keksittylintusivu.fi/kirjosieppo>  |
| <http://www.keksittylintusivu.fi/saaksi>       |
| <http://www.keksittylintusivu.fi/sinisorsa>    |
| <http://www.keksittylintusivu.fi/laulujoutsen> |
-------------------------------------------------
```

Kuten voidaan huomata, vastauksessa ovat melkein kaikkien liitteessä 1 annetun lintudatan tietoresurssien URI-osoitteet pois lukien joutsenten suvun URI-osoite. Tämä johtuu siitä, että joutsenten sukua kuvaavassa tietoresurssissa latinankielistä nimeä osoittava RDF-predikaatti on nimeltään `nimi_latinaksi`, eikä sitä oltu annettu kyselynmäärittelyssä.

6.1.2. Poimi ja valitse -kysely SWI-Prolog-kielen valmispredikaateilla

SWI-Prolog-kielen valmispredikaateilla tavallinen poimi ja valitse -kysely on toteutettavissa varsin yksinkertaisesti. Ensimmäinen esimerkki voidaan toteuttaa esimerkissä 15 annetulla lauseella

Esimerkki 15:

```
rdf(Lintu, 'http://www.jokinesimerkkiontologia.fi/#lahko',
'http://www.keksittylintusivu.fi/sorsalinnut').
```

`Rdf-`merkintä ennen sulkua kertoo, että käytetään `rdf`-predikaattia (SWI-prologin valmispredikaatti). Tämän jälkeen sulkujen sisällä olevat argumentit ovat RDF-kolmikon subjekti, RDF-predikaatti ja objekti tässä järjestyksessä. Tulokseksi saadaan seuraava:

```
Lintu = 'http://www.keksittylintusivu.fi/sinisorsa' ;
Lintu = 'http://www.keksittylintusivu.fi/laulujoutsen'.
```

Toinen esimerkkikysely voidaan toteuttaa yksinkertaisimmin yhdistämällä kaksi yksinkertaisempaa poimi ja valitse -kyselyä esimerkissä 16 esitetyllä tavalla. Kysely on seuraavanlainen:

Esimerkki 16:

```
rdf(Subjekti, 'http://www.jokinesimerkkisanasto.fi/#tyyppi', literal('Lintu')), rdf(Subjekti, 'http://www.jokinesimerkkisanasto.fi/#pituus', literal('52-60cm')).
```

Lopputulokseksi saadaan odotetusti

```
Subjekti = 'http://www.keksittylintusivu.fi/saaksi'.
```

SWI-Prologilla voidaan yhdessä kyselyssä SPARQL-kielen unionia vastaava toiminto toteuttaa siten, että määritellään kyselyyn useampi tapa, joilla vastauksia etsitään. Nämä tavat erotetaan toisistaan puolipisteellä. Esimerkissä täytyy etsiä ne kolmikot, joissa on RDF-predikaattina joko `lat_nimi` tai `latNimi`, joten käytetään kahta `rdf`-predikaattia, jotka on määritetty hakemaan juuri näitä kolmikkoja. Saadaan esimerkin 17 kysely, joka on seuravanlainen:

Esimerkki 17:

```
rdf(Subjekti, 'http://www.jokinesimerkkisanasto.fi/#lat_nimi', _);
rdf(Subjekti, 'http://www.jokinesimerkkisanasto.fi/#latNimi', _).
```

Se palauttaa vastauksen:

```
Subjekti = 'http://www.keksittylintusivu.fi/sinisorsa' ;
Subjekti = 'http://www.keksittylintusivu.fi/saaksi' ;
Subjekti = 'http://www.keksittylintusivu.fi/kirjosieppo' ;
Subjekti = 'http://www.keksittylintusivu.fi/sorsalinnut' ;
Subjekti = 'http://www.keksittylintusivu.fi/laulujoutsen'.
```

Vastauksesta voi huomata, että ensin SWI-Prolog on hakenut ensimmäisenä määritellyn predikaatin tuottamat vastaukset (sinisorsa, sääksi, kirjosieppo ja sorsalinnut) ja seuraavaksi jälkimmäisen tuottamat vastaukset (laulujoutsen).

6.1.3. Poimi ja valitse -kysely RDF/XML-relaatiolla

RDF/XML-relaatiossa poimi ja valitse -kysely on toteutettavasti samalla tavalla kuin SWI-Prolog-kielen valmispredikaateilla. Tarvitaan vain predikaatti, joka hakee halutut tiedot. Tämä predikaatti on annettu esimerkissä 18.

Esimerkki 18:

```
select(Lintu, esim_lahko, "http://www.keksittylintusivu.fi/sorsalin-
nut")
```

Esimerkissä 18 annetun predikaatin toiminta on samanlainen kuin SWI-Prolog-kielen valmispredikaattien `rdf`-predikaatinkin. Erot tulevat esiin koodissa.

Toisenkin esimerkin kohdalla voidaan hyödyntää samaa ajatusta kuin SWI-Prolog-kielen valmispredikaattien kohdalla. Saadaan esimerkissä 19 annettu kysely

Esimerkki 19:

```
select(Subjekti, esim_tyyppi, lintu), select(Subjekti, esim_pituus,
'52-60cm'),
```

joka palauttaa tulokseksi sääksen URI-osoitteen.

Myös kolmas esimerkki voidaan toteuttaa samalla tavoin kuin SWI-Prolog-kielen valmispredikaateilla. Saadaan esimerkissä 20 annettu kysely

Esimerkki 20:

```
select(Subjekti, esim_lat_nimi, _); select(Subjekti, esim_latNimi, _),
```

joka palauttaa saman vastauksen kuin SWI-Prolog-kielen valmispredikaateilla toteutettu versio. Nämä samanlaisuudet johtuvat erityisesti siitä, että RDF/XML-relaation toiminnallisuus on tässä työssä toteutettu SWI-Prolog-kielellä.

6.2. Aggregointikysely

Aggregointikyselyt ovat kyselyjä, joissa vastaukseen kootaan uutta informaatiota käytettävissä olevan datan pohjalta. Tämä voisi olla esimerkiksi lukujen summaamista. Tämän toiminnallisuuden testaamiseksi tehdään kysely, joka laskee kuinka monta lintua lintudata sisältää. Kysely laskee, kuinka monta sellaista kolmikkoa löytyy, jolla on predikaattina `esim:tyyppi` ja objektina `lintu`. Toisena esimerkkinä tehdään kysely, joka tuottaa vastaukseksi kaikki tietolähteiden tyypit ja niiden lukumäärät lintudatassa.

6.2.1. Aggregointikysely SPARQL-kielellä

SPARQL-kielen uudemmassa SPARQL 1.1-versiossa tuetaan aggregaatteja. Nämä funktiot ovat `COUNT` (instanssien lukumäärä), `SUM` (numeeristenarvojen yhteenlasku), `MIN` (pienin arvo), `MAX` (suurin arvo), `AVG` (keskiarvo), `GROUP_CONCAT` (tekstityyppisten arvojen yhdistäminen) ja `SAMPLE` (palauttaa satunnaisen arvon sille annetusta arvojoukosta). Lisäksi tietoa voidaan ryhmitellä `GROUP BY` -funktiolla. [Harris and Seaborne, 2013]

Kun halutaan laskea tietolähteet, joiden tyyppinä on `Lintu`, käytetään `COUNT`-funktiota. Kyselyssä tehdään muuten tavallinen `SELECT`-kysely, mutta `COUNT`-funktion avulla lasketaan ehtoa vastaavien kolmikkojen lukumäärä. Kysely on esitetty esimerkissä 21. Kyselystä tulee seuraavanlainen:

Esimerkki 21:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX esim: <http://www.jokinesimerkkisanasto.fi/#>
SELECT(COUNT(?subjekti) as ?lkm)
WHERE
{?subjekti esim:tyyppi 'Lintu'}
```

Kyselyn tulokseksi saadaan seuraava tulostaulu:

```
| lkm |
=======
| 4  |
```

Toisessakin esimerkissä käytetään tyyppien laskemiseen `COUNT`-funktiota. Lisäksi ryhmitellään tyypit `GROUP BY` -funktion avulla, jolloin saadaan valittua tyypit aineistoon vain yhden kerran. Tämä kysely on annettu esimerkissä 22. Kyselystä tulee seuraavanlainen:

Esimerkki 22:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX esim: <http://www.jokinesimerkkisanasto.fi/#>
SELECT ?tyyppi (COUNT(?tyyppi) as ?lkm)
WHERE
{?subjekti esim:tyyppi ?tyyppi }
GROUP BY ?tyyppi
```

Tulokseksi saadaan:

```
=================
| "Lahko" | 1   |
| "Lintu" | 4   |
| "Suku"  | 1   |
-----------------
```

6.2.2. Aggregointikysely SWI-Prolog-kielen valmispredikaateilla

SWI-Prolog-kielen valmispredikaateilla aggregointia ei voi suorittaa RDF-tietolähteiden käsittelyyn tarkoitetun kirjaston pohjalta suoraan, mutta SWI-Prolog tarjoaa kirjaston, joka mahdollistaa aggregoinnin. Täältä valitaan aggregointia varten predikaatti

```
aggregate_all(Template, Goal, Result).
```

`Template` kertoo, mitä aggregointifunktiota käytetään, `Goal` mitä tuloksia pitää aggregoida ja `Result` palauttaa lopputuloksen. [O'Keefe and Wielemaker, 2016].

Kyselyksi tulee esimerkissä 23 annettu kysely:

Esimerkki 23:

```
aggregate_all(count, rdf(Subjekti, 'http://www.jokinesimerkkisa-
nasto.fi/#tyyppi', literal('Lintu')), Lkm).
```

Kuten SPARQL esimerkissä, tässäkin valitaan tavallisella poimintakyselyllä ehdot täyttävät kolmikot ja lasketaan niiden lukumäärä, joka on 4. Aggregointikysely on toteutettavissa valmispredikaattien avulla, mutta nyt joudutaan jo turvautumaan muiden kirjastojen apuun.

Myös monimutkaisempi esimerkki, jossa etsitään kaikki tyypit ja niiden lukumäärät, voidaan toteuttaa SWI-Prolog-kielen valmispredikaateilla. Kuitenkin jos halutaan saada aikaiseksi täysin SPARQL-version vastausta vastaava tulos, joudutaan tekemään hieman monimutkaisempi kysely, jotta saadaan poistetuksi ratkaisut, joita Prolog tarjoaa useampaan kertaan (duplikaatit). Kyselyssä käytetään `findall`-predikaattia [SWI-Prolog D, 2016], joka kerää kaikki halutun kyselyn (`(rdf(..),aggregate_all(…))` tuottamat arvot halutuille muuttujille (`Tyyppi`, `Lkm`) listaan `Tyypit`. Lopuksi käytetään `sort`-predikaattia, joka lajittelee halutun listan ja poistaa siitä samalla duplikaatit [SWI-Prolog E, 2016]. Kyselyssä itsessään etsitään ensin kaikki tyypit `rdf`-predikaatin avulla ja sen jälkeen lasketaan `aggregate_all`-predikaatin avulla, kuinka monta kertaa kyseinen tyyppi löytyy aineistosta. Kysely on annettu esimerkissä 24 ja se on seuraavanlainen:

Esimerkki 24:

```
findall((Tyyppi, Lkm),
   (rdf(_, 'http://www.jokinesimerkkisanasto.fi/#tyyppi',
        literal(Tyyppi)),aggregate_all(count, rdf(_,
        'http://www.jokinesimerkkisanasto.fi/#tyyppi', lit-
        eral(Tyyppi)), Lkm)),
    Tyypit),
sort(Tyypit, Tyypit_ei_duplikaatteja).
```

Vastaukseksi saadaan ensin lista `Tyypit`, jossa on duplikaatit ja lista `Tyypit_ei_duplikaatteja`, josta ne on poistettu. Näistä jälkimmäinen on haluttu vastaus.

```
Tyypit = [('Lintu', 4), ('Lintu', 4), ('Lintu', 4), ('Lintu', 4),
('Suku', 1), ('Lahko', 1)],
Tyypit_ei_duplikaatteja = [('Lahko', 1), ('Lintu', 4), ('Suku', 1)].
```

Toisaalta jos duplikaatit eivät haittaa, voidaan tehdä paljon yksinkertaisempi kysely käyttämällä vain `rdf` ja `aggregate_all`-predikaattien yhdistelmää esimerkissä 25 esitetyllä tavalla. Kysely on seuraavanlainen:

Esimerkki 25:

```
rdf(_, 'http://www.jokinesimerkkisanasto.fi/#tyyppi',
literal(Tyyppi)),
aggregate_all(count, rdf(_, 'http://www.jokinesimerk-
kisanasto.fi/#tyyppi', literal(Tyyppi)), Lkm)
```

6.2.3. Aggregointikysely RDF/XML-Relaatiolla

Myös RDF/XML-relaatioon tehtävissä aggregointikyselyissä voidaan hyödyntää samaa SWI-Prolog-kielen `aggregate_all`-predikaattia, jota käytettiin SWI-Prolog-kielen valmispredikaattien kohdalla. Tällöin saadaan kyselyksi esimerkissä 26 annettu kysely.

Esimerkki 26:

```
aggregate_all(count, select(Subjekti, esim_tyyppi, "lintu"), Lkm).
```

Tämä kysely on muuten sama kuin esimerkki 23, mutta `rdf`-predikaatti on korvattu aliluvun 5.3.1. `select`-predikaatilla. Vastaukseksi saamme `Lkm=4`, niin kuin kuuluukin.

Kun verrataan RDF/XML-relaatioon tehtyä kyselyä SWI-Prolog-kielen valmispredikaateilla tehtyyn, huomataan että RDF/XML-relaation versio on syntaksiltaan hieman yksinkertaisempi.

Myös monimutkaisemmassa esimerkissä, jossa haetaan tyypit ja niiden lukumäärät, on mahdollista saada haluttu tulos samantyyppisellä kyselyllä kuin SWI-Prolog-kielen valmispredikaattien tapauksessa. Tämä kysely on, kuten esimerkistä 24 voi huomata, kuitenkin monimutkainen, joten RDF/XML-relaation tapauksessa osa toiminnallisuutta voidaan kyselyn yksinkertaistamiseksi siirtää omaan predikaattiin. Tähän toteutetaan `findall_without_duplicates` predikaatti (toteutus liitteessä 4), joka on käytännössä esimerkin 24 kysely omana predikaattinaan. Tällöin voidaan jättää varsinaisesta kyselystä `sort`-predikaatti pois. Huomioitava on myös, ettei `findall_without_duplicates` ole varsinaisesti RDF/XML-relaatioon liittyvä predikaatti vaan se toimii monien erilaisten kyselyjen ja aineistojen kanssa (esimerkiksi SWI-Prologin valmispredikaateilla). Kysely on esitetty esimerkissä 27 ja se on seuraavanlainen:

Esimerkki 27:

```
findall_without_duplicates((Tyyppi, Lkm), (select(_, esim_tyyppi, Tyyppi),aggregate_all(count, select(_, esim_tyyppi, Tyyppi), Lkm)), Tyypit).
```

Vastaukseksi saadaan:

```
Tyypit = [(lahko, 1), (lintu, 4), (suku, 1)].
```

Voidaan huomata, että nyt `Tyypit`-lista sisältää suoraan halutun tuloksen.

6.3. Analyyttinen kysely

Analyyttisessä kyselyssä pyritään analysoimaan annettua tietoa. Tästä on esimerkkinä aliluvussa 5.4. annettu kysely (esimerkki 11), jossa katsottiin, mikä on tietolähteessä eksplisiittisesti annettu `v`-tyyppinen tietosisältö. Seuraavana esimerkkinä on kysely, jossa tutkitaan, että onko jokin tieto saavutettavissa käsillä olevasta tietolähteestä. Tästä tilanteesta voi olla kaksi versiota. Ensimmäisenä on tilanne, jossa etsittävä tieto on osana subjektin kuvausta, kuten esimerkkidokumentissa fiktiivisen sinisorsan nimi `Repe`.

```
<rdf:Description rdf:about="http://www.keksittylintu-
sivu.fi/sinisorsa">
     <esim:tyyppi>Lintu</esim:tyyppi>
     .
```

```
    .
    .
  <esim:lahko  rdf:resource="http://www.keksittylintusivu.fi/sorsalin-
  nut" />
    .
    .
    .
    <esim:fiktiossa>
         <rdf:Description rdf:about="http://www.keksittylintu-
         sivu.fi/repe">
              <esim:nimi>Repe</esim:nimi>
         </rdf:Description>
    </esim:fiktiossa>
</rdf:Description>
```

Toinen tilanne on puolestaan sellainen, jossa etsittävä tieto sijaitsee toisen tietolähteen kuvauksessa. Esimerkkinä tästä voi olla sinisorsan lahkon latinankielinen nimi, joka näyttää RDF/XML-muodossa seuraavalta.

```
<rdf:Description rdf:about="http://www.keksittylintusivu.fi/sorsalin-
nut">
    <esim:tyyppi>Lahko</esim:tyyppi>
    <esim:lat_nimi>Anseriformes</esim:lat_nimi>
</rdf:Description>
```

Tutkitaan tätä analyyttisyyttä kahdella esimerkillä. Ensin tutkitaan, voidaanko nimi `Repe` saavuttaa tietolähteestä sinisorsa, ja sen jälkeen, voidaanko nimi `Anseriformes` saavuttaa tietolähteestä sinisorsa.

Tiedon saavutettavuus eri tietolähteistä ei ole ainoa tapa, jolla aineistoa voidaan analysoida. Tämän vuoksi testataan eri lähestymistapoja myös käyttäen aliluvussa 5.4.2. tutkittua esimerkkiä, joka katsoo, mitä eksplisiittisesti ilmaistua materiaalia aineistosta löytyy.

Kolmanneksi tehdään vielä hieman monimutkaisempi kysely. Tässä kyselyssä katsotaan, mitä kaikkia literaaleja voidaan saavuttaa tietolähteestä laulujoutsen siten, että aineistosta tiedetään vain laulujoutsen-tietolähteen URI-osoite.

6.3.1. Analyyttinen kysely SPARQL-kielellä

Tietojen saavutettavuutta voidaan analysoida tiettyyn pisteeseen asti ASK-kyselyn avulla. Tehdään siis haku, jossa kysytään, onko olemassa kolmikko, jonka subjekti on sinisorsa ja jonka objekti on jossakin toisessa kolmikossa subjektina, ja että jälkimmäisen kolmikon RDF-predikaattina on nimi ja objektina Repe. Tämä kysely on annettu esimerkissä 28 ja se on seuraavanlainen:

Esimerkki 28:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX esim: <http://www.jokinesimerkkisanasto.fi/#>
ASK
{<http://www.keksittylintusivu.fi/sinisorsa> esim:fiktiossa ?objekti.
?objekti esim:nimi 'Repe'};
```

Haluttu kysely on siis toteutettavissa, mutta jotta se voidaan toteuttaa, käytettävissä on oltava riittävästi tietoa tietolähteen rakenteesta, sillä jos ei tiedettäisi, että tietolähteessä on rakenne, jossa sinisorsalla on RDF-predikaatti fiktiossa, kyselyä ei voitaisi muodostaa. Tätä tietoa voidaan hankkia esimerkiksi DESCRIBE-kyselyn avulla.

Vastaavasti myös sorsalintujen lahkon latinankielistä nimeä voidaan etsiä ASK-kyselyn avulla. Laaditaan kysely, jossa katsotaan, että onko olemassa kolmikko, jonka subjekti on sinisorsa, ja jonka objekti on jossakin toisessa kolmikossa subjektina, ja että jälkimmäisen kolmikon RDF-predikaattina on lat_nimi ja objektina Anseriformes. Saatuun kyselyyn pätevät samat olettamukset kuin aikaisempaankin esimerkkiin. Kysely on esitetty esimerkissä 29 ja se näyttää seuraavalta:

Esimerkki 29:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX esim: <http://www.jokinesimerkkisanasto.fi/#>
ASK
{<http://www.keksittylintusivu.fi/sinisorsa> esim:lahko ?objekti.
?objekti esim:lat_nimi 'Anseriformes'};
```

Tietojen eksplisiittisyyttä voidaan testata SPARQL-kielestä löytyvän isLiteral-suodattimen avulla [Harris and Seaborne, 2013] Kyseinen suodatin siis palauttaa arvon ainoastaan silloin, kun haluttu arvo on literaali. Saadaan esimerkin 30 kysely, joka on seuraavanlainen:

Esimerkki 30:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX esim: <http://www.jokinesimerkkisanasto.fi/#>
SELECT ?subjekti ?predikaatti ?objekti
WHERE
{?subjekti ?predikaatti ?objekti .
FILTER isLiteral(?objekti)}
```

Tulokseksi saadaan kuvan 6 mukainen taulukko.

subjekti	predikaatti	objekti
<http//www.keksittylintusivu.fi/repe>	esim:nimi	"Repe"
<http://www.keksittylintusivu.fi/laulujoutsen>	esim:siipienKarkivali	"205-235cm"
<http://www.keksittylintusivu.fi/laulujoutsen>	esim:pituus	"140-160cm"
<http://www.keksittylintusivu.fi/laulujoutsen>	esim:latNimi	"Cygnus cygnus"
<http://www.keksittylintusivu.fi/laulujoutsen>	esim:nimi	"Laulujoutsen"
<http://www.keksittylintusivu.fi/laulujoutsen>	esim:tyyppi	"Lintu"
<http://www.keksittylintusivu.fi/joutsenet>	esim:nimi_latinaksi	"Cygnus"
<http://www.keksittylintusivu.fi/joutsenet>	esim:tyyppi	"Suku"
<http://www.keksittylintusivu.fi/saaksi>	esim:siipien_karkivali	"152-167cm"
<http://www.keksittylintusivu.fi/saaksi>	esim:pituus	"52-60cm"
<http://www.keksittylintusivu.fi/saaksi>	esim:lat_nimi	"Pandion haliaetus"
<http://www.keksittylintusivu.fi/saaksi>	esim:nimi	"Saaksi"
<http://www.keksittylintusivu.fi/saaksi>	esim:tyyppi	"Lintu"
<http://www.keksittylintusivu.fi/sinisorsa>	esim:siipien_karkivali	"81-95cm"
<http://www.keksittylintusivu.fi/sinisorsa>	esim:pituus	"50-60cm"
<http://www.keksittylintusivu.fi/sinisorsa>	esim:lat_nimi	"Anas platyrhynchos"
<http://www.keksittylintusivu.fi/sinisorsa>	esim:nimi	"Sinisorsa"
<http://www.keksittylintusivu.fi/sinisorsa>	esim:tyyppi	"Lintu"
<http://www.keksittylintusivu.fi/kirjosieppo>	esim:pituus	"12-13,5cm"
<http://www.keksittylintusivu.fi/kirjosieppo>	esim:lat_nimi	"Ficedula hypoleuca"
<http://www.keksittylintusivu.fi/kirjosieppo>	esim:nimi	"Kirjosieppo"
<http://www.keksittylintusivu.fi/kirjosieppo>	esim:tyyppi	"Lintu"
<http://www.keksittylintusivu.fi/sorsalinnut>	esim:lat_nimi	"Anseriformes"
<http://www.keksittylintusivu.fi/sorsalinnut>	esim:tyyppi	"Lahko"

Kuva 6. Lintudatan eksplisiittisesti ilmaistu tieto haettuna SPARQL-kielen toimintojen avulla.

Haluttu eksplisiittisesti ilmaistu tieto löytyy kohdasta `objekti`. Tämän kaltainen kysely on varsin helposti toteutettavissa SPARQL-kielellä.

Kolmas esimerkkikysely on toteutettavissa jossain määrin kahden aiemman esimerkin pohjalta. Koska tässä esimerkissä oletetaan, että aineistoa ei tunneta, tehdään kyselystä sellainen, joka etsii laulujoutsen-tietolähteeseen suoraan kuvatut literaalit (muiden arvojen löytäminen vaatisi aineiston tuntemista, jotta voitaisiin tietää, mitä muita ehtoja näiden arvojen löytäminen vaatii). Tehdään esimerkin 31 kysely, joka näyttää seuraavalta:

Esimerkki 31:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

```
PREFIX esim: <http://www.jokinesimerkkisanasto.fi/#>
SELECT ?predikaatti ?objekti
WHERE
{<http://www.keksittylintusivu.fi/laulujoutsen> ?predikaatti ?objekti .
FILTER isLiteral(?objekti)}
```

Vastaukseksi tulee:

```
-----------------------------------------
| predikaatti           | objekti         |
=========================================
| esim:siipienKarkivali | "205-235cm"     |
| esim:pituus           | "140-160cm"     |
| esim:latNimi          | "Cygnus cygnus" |
| esim:nimi             | "Laulujoutsen"  |
| esim:tyyppi           | "Lintu"         |
-----------------------------------------
```

Vastauksesta voidaan huomata, että löydetään kaikki literaalit, jotka liittyvät suoraan laulujoutsen-tietolähteeseen, mutta ne, jotka pitäisi hakea toisen URI-osoitteen takaa, jäävät löytymättä.

6.3.2. Analyyttinen kysely SWI-Prolog-kielen valmispredikaateilla

SWI-Prolog-kielen valmispredikaateilla on mahdollista arvioida tietolähteen saavutettavuutta käsillä olevasta tietolähteestä `rdf_reachable`-predikaatin avulla. Kuitenkin `rdf_reachable`-predikaatin käyttö edellyttää, että tieto on ilmaistu RDF-skeeman avulla, kuten luvussa 4 mainittiin. Koska tässä työssä ei käytetä mitään RDF-mallin lisäosia, RDF-skeeman ja `rdf_reachable`-predikaatin käyttö ei ole vaihtoehto. Myöskin luomalla omia predikaatteja olisi mahdollista tutkia tietolähteiden saavutettavuutta. Koska tässä yhteydessä tutkitaan vain SWI-Prolog-kielen valmispredikaatteja, ei tämäkään vaihtoehto ole käytettävissä. Näiden seikkojen pohjalta voidaan todeta, että SWI-Prolog-kielen valmispredikaateilla ei voi tehdä tämänkaltaista analyyttista kyselyä RDF-dataan.

Eksplisiittisen tiedon etsiminen onnistuu SWI-Prolog-kielen valmispredikaateilla siten, että mainitaan, että haetaan halutut objektit literaaleina. Literaalit on määritelty SWI-Prolog-kielen semweb-kirjastossa `literal(arvo)` [Wielemaker, 2016]. Tehdään esimerkin 32 kysely.

Esimerkki 32:

```
rdf(Subjekti, Predikaatti, literal(Objekti)).
```

Seuraavassa on osa kyselyn palauttamasta tuloksesta:

```
Subjekti = 'http://www.keksittylintusivu.fi/sinisorsa',
Predikaatti = 'http://www.jokinesimerkkisanasto.fi/#tyyppi',
Objekti = 'Lintu' ;
Subjekti = 'http://www.keksittylintusivu.fi/sinisorsa',
Predikaatti = 'http://www.jokinesimerkkisanasto.fi/#nimi',
Objekti = 'Sinisorsa' ;
Subjekti = 'http://www.keksittylintusivu.fi/sinisorsa',
Predikaatti = 'http://www.jokinesimerkkisanasto.fi/#lat_nimi',
Objekti = 'Anas platyrhynchos' ;
Subjekti = 'http://www.keksittylintusivu.fi/sinisorsa',
Predikaatti = 'http://www.jokinesimerkkisanasto.fi/#pituus',
Objekti = '50-60cm' ;
Subjekti = 'http://www.keksittylintusivu.fi/sinisorsa',
Predikaatti = 'http://www.jokinesimerkkisanasto.fi/#siipien_karkivali',
Objekti = '81-95cm' ;
Subjekti = 'http://www.keksittylintusivu.fi/repe',
Predikaatti = 'http://www.jokinesimerkkisanasto.fi/#nimi',
Objekti = 'Repe' ;
```

Kuten voidaan huomata kysely, antaa objektina vain literaaliarvoja, joten tämänlainen analyyttinen kysely voidaan toteuttaa SWI-Prolog-kielen valmispredikaateilla.

Kolmas esimerkkikysely on toteutettavissa SWI-Prolog-kielen valmispredikaateilla samaan pisteeseen asti kuin sen sai toteutettua SPARQL-kielen avulla. Esimerkissä 33 tehdään muuten vastaava kysely kuin aiemmassakin esimerkissä, mutta lisätään subjektiksi laulujoutsen-tietolähde. Saadaan kysely:

Esimerkki 33:

```
rdf('http://www.keksittylintusivu.fi/laulujoutsen', Predikaatti, literal(Objekti)).
```

Vastaukseksi tulee:

```
Predikaatti = 'http://www.jokinesimerkkisanasto.fi/#tyyppi',
Objekti = 'Lintu' ;
Predikaatti = 'http://www.jokinesimerkkisanasto.fi/#nimi',
Objekti = 'Laulujoutsen' ;
```

```
Predikaatti = 'http://www.jokinesimerkkisanasto.fi/#latNimi',
Objekti = 'Cygnus cygnus' ;
Predikaatti = 'http://www.jokinesimerkkisanasto.fi/#pituus',
Objekti = '140-160cm' ;
Predikaatti = 'http://www.jokinesimerkkisanasto.fi/#siipienKarkivali',
Objekti = '205-235cm'
```

Kuten tuloksesta voi nähdä, tämä kysely antaa saman tuloksen kuin SPARQL-versiokin.

6.3.3. Analyyttinen kysely RDF/XML-relaatiolla

RDF/XML-relaation avulla tiedon saavutettavuuden tutkiminen onnistuu helposti tutkimalla indeksejä. Tämän toteuttaminen vaatii kahden asian tutkimista. Ensinnäkin on tutkittava, että löytyykö haluttu tieto käsillä olevan tietolähteen kuvauksesta itsestään, kuten tilanteessa, jossa haetaan fiktiivisen sinisorsan nimi (`repe`). Tällöin täytyy tarkistaa, että sisältyykö subjektina toimivan tietolähteen indeksi (elementin `rdf_description` indeksi, joka aloittaa kolmikon) objektina toimivan tietolähteen indeksiin. Toinen tilanne on se, jossa tutkitaan, löytyykö haluttu tieto jostain toisesta tietolähteestä. Tällöin on tutkittava, että löydetäänkö etsittävä tieto seuraamalla URI-osoitteiden muodostamaa polkua tietolähteestä toiseen. Tehdään `reachable`-predikaatti tähän tarkoitukseen:

```
reachable(S, O) :-
     atom_codes(S, S1),
     d(S1, _, Sind),
     d(O1, v, Oind),
     atom_codes(O, O1),
     im_predecessor(Sind, Sind2),
     im_predecessor(Sind2, Sind_final),
     append(Sind_final, _, Oind).

reachable(S, O) :-
     atom_codes(S, S1),
     d(S1, _, Sind),
     d(O1, u, Oind),
     atom_codes(O, O1),
     im_predecessor(Sind, Sind2),
     im_predecessor(Sind2, Sind_final),
     append(Sind_final, _, Oind).
```

```
reachable(S, O) :-
     atom_codes(S, S1),
     d(S1, _, Sind),
     d(S2, u, S2ind),
     S1 \= S2,
     im_predecessor(Sind, Sind2),
     im_predecessor(Sind2, Sind_final),
     append(Sind_final, _, S2ind),
     atom_codes(S3, S2),
     reachable(S3, O).
```

Tästä predikaatista on kolme versiota, joista kaksi ensimmäistä tutkivat, löytyykö haluttu tieto (ensimmäisessä predikaatissa literaali, toisessa URI) käsillä olevasta tietolähteestä itsestään. Kolmas predikaatti puolestaan tutkii, löytyykö käsillä olevasta tietolähteestä URI, jonka kautta voitaisiin lähteä etsimään tietoa jostakin toisesta tietolähteestä.

Kahdessa ensimmäisessä predikaatissa jokaisen subjektin kohdalla pitää etsiä sitä edeltävän `rdf_description`-elementin indeksi `Sind_final`. `Sind_final` on se indeksi, joka on tietolähteen hierarkiassa ylimpänä ja on siten yhteinen kaikille tietolähteen komponenteille. Täten, jos tavoiteltava tieto eli objekti on osana tietolähdettä, niin sen indeksin täytyy alkaa `Sind_final`-indeksillä. Kun `Sind_final` on löydetty, tutkitaan että sisältyykö se objektin indeksiin `Oind` siten, että `Sind_final` on `Oind`-indeksin alku. Tämä voidaan testata SWI-Prolog-kielessä valmiiksi toteutetulla `append`-predikaatilla, joka tutkii, että saadaanko `Oind`-lista tuotettua, kun `Sind_final`-listan perään lisätään tietyt arvot [SWI-Prolog B, 2016]. Jos tämä onnistuu, haluttu tieto voidaan saavuttaa.

Kolmas predikaatti toimii muuten samalla tavoin kuin muutkin, mutta siinä etsitään objektin sijasta uusi subjekti (`Subjekti2`, jonka indeksi on `S2ind`) valitsemalla käsillä olevasta tietolähteestä jokin URI. Seuraavaksi pitää katsoa, että uusi subjekti ei ole sama kuin alkuperäinen subjekti. Tämä tapahtuu `\=` merkinnän avulla, joka on SWI-Prolog-kielen termien erilaisuutta ilmaiseva merkki [SWI-Prolog C, 2016]. Kolmantena erona on se, että lopuksi kutsutaan uudestaan `reachable`-predikaattia uusilla parametreilla (`Subjekti2` ja `Objekti`), jotta voidaan tutkia, löytyykö haluttu tieto seuraavasta tietolähteestä. Jos ei löydy, niin valitaan tietolähteestä URI ja lähdetään etsimään haluttua tietoa tämän URI-osoitteen osoittamasta tietolähteestä (kyseessä on syvyyssuuntainen haku). Näin voidaan käydä RDF-graafia läpi, kunnes haluttu tieto löytyy (jos sitä voi ylipäätään löytää). Kaikissa kolmessa predikaatissa esiintyvää `atom_codes`-predikaattia käytetään

muuttamaan tekstiä luettavasta muodosta ASCII-koodiksi ja takaisin aina tarpeen mukaan.

Testataan predikaattia tekemällä kyselyt, jotka tutkivat voidaanko tietolähteestä sinisorsa saavuttaa literaalit `'repe'` ja `'anseriformes'`. Muodostetaan esimerkkien 34 ja 35 kyselyt:

Esimerkki 34:
```
reachable('http://keksittylintusivu.fi/sinisorsa', 'repe')
```

ja

Esimerkki 35:
```
reachable('http://keksittylintusivu.fi/sinisorsa', 'anseriformes'),
```

jotka palauttavat tuloksena arvon `true`. Annettujen arvojen täytyy olla '-merkkien sisällä, jotta `atom_codes`-predikaatti voi ottaa ne vastaan. Tässä huomataan, että tarvitsee vain tietää halutut tietolähteet, toisin kuin SPARQL-kielen tapauksessa.

Eksplisiittinen tiedonhaku RDF/XML-relaation avulla on toteutettu aliluvussa 5.4.2., jossa tehtiin tätä tarkoitusta varten `select_literals`-predikaatti. Kyseisessä predikaatissa periaatteessa riittää, että otetaan mukaan kaikki relaatiot, joiden tyyppinä on `v` eli literaali, joskin selkeyden vuoksi lisätään muuta tietoa hieman kontekstiksi. Seuraavaksi annetaan osa esimerkissä 36 esitetyn kyselyn

Esimerkki 36:
```
select_literals(Subjekti, Predikaatti, Objekti)
```

tuloksesta:

```
Subjekti = 'http://keksittylintusivu.fi/sinisorsa',
Predikaatti = esim_tyyppi,
Objekti = lintu ;
Subjekti = 'http://keksittylintusivu.fi/sinisorsa',
Predikaatti = esim_nimi,
Objekti = sinisorsa ;
Subjekti = 'http://keksittylintusivu.fi/sinisorsa',
Predikaatti = esim_latnimi,
Objekti = 'anas platyrhynchos' ;
Subjekti = 'http://keksittylintusivu.fi/sinisorsa',
Predikaatti = esim_pituus,
```

```
Objekti = '50-60cm' ;
Subjekti = 'http://keksittylintusivu.fi/sinisorsa',
Predikaatti = esim_siipien_karkivali,
Objekti = '81-95cm' ;
Subjekti = 'http://keksittylintusivu.fi/sinisorsa',
Predikaatti = esim_nimi,
Objekti = repe ;
```

Kolmas esimerkkikysely on toteutettavissa RDF/XML-relaation avulla kokonaisuudessaan. Kyselyssä täytyy ensin hakea kaikki tieto, johon `laulujoutsen`-tietolähteestä voidaan päästä. Sen jälkeen katsotaan `select_literals`-predikaatin avulla, että onko kyseinen objekti literaali ja mikä sitä edeltävä RDF-predikaatti on. Lopuksi vielä laitetaan koko kysely `findall_without_duplicates`-predikaatin sisään, jolloin päästään eroon duplikaateista. Kyselyksi tulee esimerkissä 37 annettu kysely, joka on seuraavanlainen:

Esimerkki 37:

```
findall_without_duplicates((Predikaatti, Objekti), (reacha-
ble('http://www.keksittylintusivu.fi/laulujoutsen', Objekti), se-
lect_literals(_, Predikaatti, Objekti)), Tulokset).
```

Vastaukseksi tulee:

```
Tulokset = [(esim_latNimi, 'cygnus cygnus'), (esim_lat_nimi, anserifor-
mes), (esim_nimi, laulujoutsen), (esim_nimi_latinaksi, cygnus),
(esim_pituus, '140-160cm'), (esim_siipienkväli, '205-235cm'),
(esim_tyyppi, lahko), (esim_tyyppi, lintu), (esim_tyyppi, suku)].
```

Nyt voidaan huomata, että tuloslista sisältää myös joutsenet- ja sorsalinnut-tietolähteiden sisältämiä tietoja. Muista versioista poikkeava järjestys johtuu siitä, että `findall_without_duplicates` lajittelee tulokset.

7. Käytettävyyden arviointia

Lähdemateriaalin pohjalta valittiin joitakin kriteereitä kyselykielten käytettävyyden arvioimiseksi, jotta eri lähestymistapoja voidaan verrata. Ensimmäinen kriteeri on, kuinka helposti kyselyjä voidaan kirjoittaa ja lukea [Reisner, 1981]. Myös kyselykielen proseduraalisuus eli se, kuinka paljon kyselyä tehdessä kerrotaan kielelle siitä, miten tulos saavutetaan, on perinteinen mittari [Welty and Stemple, 1981]. Weltyn ja Stemplen [1981] tekemien kokeiden mukaan vaikeiden kyselyiden teko on helpompaa proseduraalisilla kielillä kuin ei-proseduraalisilla kielillä, joista voidaan käyttää myös nimitystä deklaratiivinen kieli [Lassila, 2013]. Deklaratiivisissa kielissä määritellään haluttu tulos. Tässä työssä tehty vertailu perustuu vain kielten ja kyselyiden analyysiin, ja on siten vain suuntaa-antava. Kattavamman arvioinnin tekeminen vaatisi käyttäjäkokeita. Arviointiin valittiin sellaiset mittarit, jotka sopivat tässä työssä tehtyyn analyysiin.

7.1. Kyselyiden kirjoittaminen

Kyselyiden kirjoittamisessa kyse on siitä, kuinka helppoa käyttäjän on laatia kyselyitä kyseisellä kyselykielellä. Toisin sanoen voidaan kysyä, että kuinka paljon ajatustyötä kyselyn laatiminen tietyllä kielellä teettää. Jarke ja Vassiliou [1985] esittivät, että yksi tähän vaikuttavista tekijöistä on kielen syntaktisien rakenteiden määrä. Tämä tarkoittaa sitä, kuinka paljon kielessä on erilaisia muistettavia avainsanoja ja muita kielioppisääntöjä, kuten se missä järjestyksessä haluttujen toimintojen tulee kyseisessä kyselyssä esiintyä. Tämä on hyvä mittari, koska sen avulla on helppo arvioida kielten toiminnallisuutta. Tässä työssä tätä arvioidaan siten, että mitä vähemmän avainsanoja ja syntaktisia rakenteita täytyy muistaa, sen parempi.

Jarke ja Vassiliou [1985] esittävät myös toisen kätevän mittarin, jonka avulla kyselyn kirjoittamista on hyvä arvioida. Tämä mittari on se, kuinka paljon rutiininomaista työtä kyselyn laatiminen vaatii. Tätä voidaan Jarken ja Vassiloun [1985] mukaan mitata esimerkiksi sillä, kuinka monta näppäimen painallusta kyselyn kirjoittaminen vaatii, ja tämän yksinkertaisuuden vuoksi se sopii mittariksi. Oletetaan, että vähemmän näppäimen painalluksia on parempi. Jarke ja Vassiliou [1985] tarjoavat myös muita mittareita kyselykielten vertailuun, mutta ne eivät juurikaan sovellu tämän työn tarkoituksiin, sillä niiden tutkiminen vaatisi käyttäjäkokeita ja RDF/XML-relaatioon perustuvan järjestelmän tulisi olla paljon pidemmälle kehitetty.

Edellisen luvun kyselyesimerkeistä nähdään, että SPARQL on syntaksiltaan monimutkaisin. Se vaatii myös selkeästi eniten näppäinten painalluksia. Tämä voidaan huomata, kun katsotaan esimerkkejä, joissa SPARQL vaatii useamman rivin tekstiä ja monia avainsanoja, kun taas SWI-Prolog-kielen valmispredikaateilla ja RDF/XML-relaation ratkaisuissa selvitään yhdellä predikaatilla tai yhdistelemällä muutamaa. Poikkeuksena

tähän voidaan pitää aggregointikyselyjä, joissa myös SWI-Prolog-kielen valmispredikaattien ja RDF/XML-relaation vaatimat syntaksit ovat varsin monimutkaisia. Näin on erityisesti jälkimmäisessä esimerkissä, jossa joudutaan muokkaamaan vastausta mielekkäämpään muotoon. Tässä kohdassa SPARQL vaikuttaa selkeämmältä.

Kun verrataan SWI-Prolog-kielen valmispredikaatteja ja RDF/XML-relaatiota, huomataan, että RDF/XML-relaatio on yksinkertaisempi. Osaltaan tämä johtuu siitä, että RDF/XML-relaation tapauksessa oletetaan, että kun RDF/XML-dokumentti muutetaan RDF/XML-relaatioksi, annetaan käytetylle ontologialle tai sanastolle lyhenne, kuten tässä työssä sanastolle on annettu lyhenteeksi esim. Jos tätä toiminnallisuutta ei toteuteta, joudutaan predikaatiksikin antamaan koko URI.

Toisaalta kyselyn loppupäässä, silloin kun tutkitaan literaaleja (esimerkit 32 ja 36), joudutaan valmispredikaattien kohdalla antamaan erikseen maininta, että kyseessä on literaali (`literal(Arvo)`), kun taas RDF/XML-relaatiossa näin ei tarvitse tehdä koska se on huomattavissa helposti relaatiosta (monikko (komponentti) on tyyppiä `v`). Näistä syistä RDF/XML-relaatio on yksinkertaisempi kuin SWI-Prolog-kielen valmispredikaatit, ja kyselyn kirjoittaminen on sen avulla helpompaa kuin SWI-Prolog-kielen valmispredikaateilla, joskaan ei automaattisesti helpompaa kuin SPARQL-kielen avulla.

7.2. Kyselyiden luettavuus

Reisner [1981] määrittelee kyselyiden luettavuuden siten, että kuinka helposti ne voidaan kääntää luonnolliselle kielelle. Tässä vertailussa arvioidaan kielien ulkoasua ja syntaksia sen mukaan, miten ne taipuvat englanniksi.

SPARQL-kielen etuna on sen avainsanat `SELECT`, `WHERE` jne. Nämä auttavat jäsentämään kyselyä ja näin jakavat sen osiin, joiden avulla on helppo hahmottaa, mitä missäkin kohtaa tapahtuu. Toisaalta varsinkin `WHERE`-osiossa esiintyvät ehdot saattavat olla aika monimutkaisia, kuten esimerkissä 6.

SWI-Prolog-kielen valmispredikaateissa ja RDF/XML-relaatioon perustuvassa ratkaisussa yksinkertaisen poimi ja valitse -kyselyn toteuttaminen onnistuu yhdellä lauseella. Sen tulkintakin on varsin yksinkertainen: valitse kolmikot, jotka sopivat annettuihin muuttujiin. Tässä kohtaa se on kompaktiutensa ja selkeytensä vuoksi parempi kuin SPARQL-kielen avainsanoihin perustuva ratkaisu. Kuitenkin monimutkaisemmissa kyselyissä joudutaan yhdistelemään useampia poimi ja valitse -kyselyjä, jotta saadaan aikaiseksi sama tulos kuin yhdellä hieman monimutkaisella SPARQL-kyselyllä. Tämä saattaa tuntua Prologiin tottumattomista käyttäjistä vaikealta, erityisesti kun Prolog-ratkaisuissa pitää oikean tuloksen saamiseksi yhdistellä saatuja tuloksia.

RDF/XML-relaatioon perustuva ratkaisu on luettavampi kuin SWI-Prolog-kielen valmispredikaatteihin perustuva ratkaisu. Tämä johtuu siitä, että tavanomaiseen poimi ja

valitse -kyselyyn predikaatin nimeksi on valittu `select`, joka kuvaa termiä `rdf` paremmin sen mitä kyseinen predikaatti tekee. Lisäksi valmispredikaattien kanssa joudutaan literaalien kohdalla esimerkiksi lisäämään rakenne `literal(Arvo)` (ks. esim: esimerkki 33) kertomaan, että kyseessä on eksplisiittinen arvo. Tämä lisää käyttäjän hämmennystä

Aggregointikyselyssä jokaisella kolmella lähestymistavalla annetaan sama tieto. Kyselyssä siis kerrotaan, että mitä tietoa aggregoidaan ja millä funktiolla tämä aggregointi tehdään. SWI-Prologin valmispredikaateilla ja RDF/XML-relaatioratkaisussa annetaan yksi lause, kun taas SPARQL-kyselyssä kysely jaetaan avainsanojen avulla. Tässä SPARQL-kyselyn avainsanat selkeyttävät kyselyä enemmän verrattuna muihin ratkaisuihin. Esimerkiksi SPARQL kertoo selkeästi, että ehtoja vastaavat kolmikot lasketaan muuttujaan lkm, mutta SWI-Prolog-ratkaisuissa `aggregate_all`-predikaattia tuntematon käyttäjä saattaa joutua tekemään enemmän ajatustyötä asian oivaltamiseksi. Vielä selkeämmin tämä näkyy monimutkaisemmassa aggregointiesimerkissä, jossa itse aggregointi pitää sijoittaa `findall`-predikaatin sisään ja duplikaatit on poistettava lajittelulla (esimerkki 24). SPARQL-kielellä tämä onnistuu käyttämällä `GROUP BY` -operaatiota. Lisäksi SPARQL ei vaadi käytetyissä esimerkeissä esittämään asioita sisäkkäisinä toimintoina samaan tapaan kuin muut ratkaisut vaativat. Tämä selkeyttää kyselyn lukemista. Toisaalta monimutkaisemmissa SPARQL-kyselyissä myös sisäkkäisten kyselyiden teko on mahdollista [Harris and Seaborne, 2013].

Kun analysoidaan sitä, kuinka hyvin jokin tietolähde voidaan saavuttaa toisesta tietolähteestä, RDF/XML-relaatio on selkeästi helpoiten luettava. Kysely on muotoa `reachable(Tietolähde1, Tietolähde2)`. SPARQL-kielinen versio ei ole yhtä selkeä eikä se edes pysty tuottamaan samanlaisia tuloksia (ei löydä kaikkia tietoja). SPARQL-kielessä pitää muotoilla kysely kaikkine osineen, kun taas RDF/XML-relaatio ratkaisussa riittää yksi lause.

Kun analysoidaan eksplisiittisesti annettua tietoa, erot eivät ole aivan yhtä selkeitä. Kaikissa versioissa oikeastaan toteutetaan tavallinen poimi ja valitse -kysely, sillä muutoksella, että objektin pitää olla aina literaali. SPARQL-kielessä tämä määritellään lisäämällä kyselyn loppuun ehto, jonka perusteella mukaan otetaan vain objektit, jotka ovat literaaleja. Valmispredikaateilla annetaan kyselyn objekti muodossa `literal(Objekti)`, joka kertoo, että valitaan vain literaali-muotoiset objektit. RDF/XML-relaation tapauksessa predikaatti `select_literals(S, P, O)` kertoo jo nimessään, että valitaan vain literaalit, joskin tässä toisin kuin SPARQL-kielessä ja SWI-Prolog-kielen valmispredikaateissa se, että kyseessä on objekti saattaa jäädä hieman epäselväksi. Selvimmin ehto on määritelty SWI-Prolog-kielen valmispredikaateissa, sillä niissä se on mainittu suoraan `rdf`-predikaatin muuttujana.

Kolmannessa analyysiesimerkissä, jossa tutkitaan mitä literaaleja tietolähteestä voidaan saavuttaa, SPARQL ja SWI-Prolog-kielen valmispredikaatit ovat paljon selkeämpiä kuin RDF/XML-relaatioon tehtävät haut. Tässä täytyy kuitenkin huomata, että ainoastaan RDF/XML-relaatioon perustuva lähestymistapa löytää kaikki halutut tiedot.

Huomautettava on, että nämä ovat vain tutkimuksen tekijän omia arvioita luettavuudesta. Jotta tästä voitaisiin saada luotettavampaa tietoa, olisi toteutettava kunnollisia käyttäjätestejä.

7.3. Proseduraalisuus

Welty ja Stemple [1981] määrittelevät proseduraalisuuden siten, että proseduraalinen kieli määrittelee ne askeleet, joiden pohjalta tulos tulee saavuttaa. Ei-proseduraalinen eli deklaratiivinen kieli puolestaan kuvaa halutun tuloksen. Tutkimuksessaan Welty ja Stemple [1981] tutkivat hypoteesia, jonka mukaan käyttäjät kirjoittavat vaikeita kyselyitä paremmin proseduraalisen kielen avulla. Myös heidän tutkimustuloksensa tukivat tätä hypoteesia. Weltyn ja Stemplen [1981] mukaan helpoissa kyselyissä ei kielen tyypillä ollut merkittävää vaikutusta.

Welty ja Stemple [1981] määrittelevät yhtälön, jonka avulla proseduraalisuutta voidaan arvioida ja tässä työssä sitä käytetään vertailtavien kielten proseduraalisuuden arviointiin. Yhtälö on seuraavanlainen:

```
PM = (sidottujen muuttujien määrä/sallittujen muuttujien sidontajärjes-
tysten määrä) + (suoritettujen operaatioiden määrä/sallittujen operaa-
tioiden suoritusjärjestysten määrä).
```

PM tarkoittaa proseduraalisuusarvoa (procedurality metric). Ensimmäisten sulkeiden sisällä olevassa laskussa jaetaan kyselyssä määriteltävien muuttujien määrä kielen syntaksin sallimien muuttujien määrittelyjärjestysten määrällä. Tähän summataan toisten sulkeiden sisällä olevan laskun tulos, jossa jaetaan kyselyssä suoritettavien operaatioiden määrä kielen operaatioiden suorittamiselle sallimien järjestysten määrällä. Seuraavassa lasketaan kaikille luvussa 6 esiintyneille esimerkeille proseduraalisuusarvo ja lasketaan laskettujen arvojen pohjalta jokaiselle kielelle proseduraalisuusarvojen keskiarvo. Näin saadaan arvio siitä, mikä kieli on proseduraalisin.

SPARQL-kielen poimi ja valitse -esimerkeissä ei sidota muuttujiin mitään arvoja. Lisäksi koko `SELECT`-kysely toimii operaationa. Koska kahdessa ensimmäisessä esimerkissä (12 ja 13) on vain yksi `SELECT`-kysely niin operaatioille sallittuja järjestyksiä voi olla vain yksi. Joten SPARQL-kielen kahden ensimmäisen poimi ja valitse -kyselyn proseduraalisuusarvo on:

```
0 + 1 = 1.
```

Kolmannen (esimerkki 14) poimi ja valitse -kyselyn `WHERE`-osiossa annettiin kaksi eri vaihtoehtoa, joiden palauttamat kolmikot yhdistettiin tulokseen unioni-operaation avulla. Tälläkään kertaa yhtään muuttujaa ei sidota mihinkään arvoon, mutta nyt tehdään kolme operaatiota. Vaikka kysely onkin ilmaistu vain yhtenä `SELECT`-kyselynä, siinä tehdään oikeastaan kaksi kyselyä, joista toinen tehdään ennen `UNION`-avainsanaa annettuihin ehtoihin ja toinen sen jälkeen annettuihin ehtoihin. Mahdollisia järjestyksiä operaatioille on kaksi, koska vastauksen kannalta ei ole väliä, kumpi ehto annetaan ensin. Siis tämän kyselyn proseduraalisuusarvoksi tulee:

```
0 + 3/2 = 1,5
```

SPARQL-kielen ensimmäisessä aggregointikyselyssä (esimerkki 21) tapahtuu yksi muuttujan sidonta kohdassa `COUNT(?s) as lkm`, jossa löydettyjen kolmikkojen summa lasketaan muuttujaan lkm. Tämä voidaan tehdä vain yhdessä järjestyksessä. Operaatioita kyselyssä on kolme `SELECT`-operaatio, `COUNT`-operaatio ja muuttujan sidonta. Mahdollisia järjestyksiä näille on yksi, joten aggregointikyselyn arvoksi tulee:

```
1/1 + 3/1 = 4.
```

Vastaavasti toisessakin aggregointikyselyssä (esimerkki 22) tehdään sama muuttujan sidonta kuin aiemminkin (`COUNT(?s) as lkm`). Muutenkin operaatiot ovat samat, mutta lopussa suoritetaan uutena vielä `GROUP BY` -operaatio. Taaskin mahdollisia järjestyksiä on yksi. Proseduraalisuusarvoksi tulee siis:

```
1/1 + 4/1 = 5.
```

Ensimmäisessä SPARQL-kielen analyyttisten kyselyiden esimerkissä (esimerkki 28, esimerkki 29 jätetään pois koska se on proseduraalisuuden tutkimisen kannalta samanlainen kuin 28) muuttujia ei sidota ja suoritetaan vain yksi operaatio (kuten poimi ja valitse -esimerkeissäkin), joten se saa arvokseen 1. Myöskään toisessa esimerkissä (30) ei sidota yhtään muuttujaa. Siinä kuitenkin suoritetaan kaksi operaatiota `SELECT` ja FILTER, jotka voivat esiintyä vain yhdessä järjestyksessä, joten arvoksi saadaan:

```
0 + 2/1 = 2.
```

Kolmannessa analyyttisessä SPARQL-esimerkissä (31) suoritetaan samat operaatiot ja muuttujien sidonnat kuin toisessakin, joten sen proseduraalisuusarvoksi tulee myös 2.

Lopuksi lasketaan saatujen tulosten keskiarvo. SPARQL-kielen proseduraalisuusarvo on siis käytettyjen esimerkkien mukaan:

```
(1 + 1 + 1,5 + 4 + 5 + 1 + 2 + 2) / 8 = 2,19.
```

Seuraavaksi tutkitaan SWI-Prolog-kielen valmispredikaattien proseduraalisuutta. Valmispredikaattien ensimmäisessä poimi ja valitse -esimerkissä (15), ei tehdä yhtään muuttujan sidontaa ja suoritetaan vain yksi operaatio, joten se saa arvokseen 1 samoin kuin SPARQL-kielen vastaava esimerkki. Toisessakaan esimerkissä (16) ei sidota muuttujia, mutta suoritetaan kaksi operaatiota, joiden järjestyksellä ei ole väliä, joten proseduraalisuusarvoksi saadaan:

```
0 + 2/2 = 1.
```

Kolmannessa kyselyssä (esimerkki 17) suoritetaan kaksi poimi ja valitse -kyselyä, joiden järjestyksellä ei ole väliä. Muuttujia ei tässäkään sidota, joten arvoksi tulee 1, kuten aiemmassa esimerkissäkin.

Ensimmäisessä aggregointikyselyssä (esimerkki 23) suoritetaan yksi muuttujan sidonta (yhteen laskettu tulos muuttujaan `Lkm`). Lisäksi suoritetaan kolme operaatiota, valinta, lasku ja muuttujan sidonta, joiden järjestykseen ei pysty vaikuttamaan. Näin ollen myös SWI-Prolog-kielen valmispredikaatit saavat ensimmäisessä aggregointikyselyssä arvokseen 4. Toisessa aggregointikyselyssä (esimerkki 24) suoritetaan kolme muuttujan sidontaa. Ensinnäkin `aggregate_all`-predikaatin suorittama lasku sidotaan muuttujaan `Lkm`, toiseksi `findall`-predikaatin löytämät tulokset sidotaan muuttujaan `Tyypit` ja lopuksi `sort`-predikaatti sitoo lajitellun ja duplikaatittoman `Tyypit`-listan muuttujaan `Tyypit_ei_duplikaatteja`. Muuttujien sidonnalle on yksi mahdollinen järjestys. Suoritettavia operaatioita on 4. Nämä operaatiot ovat tyypin valinta, tyypin esiintymien lasku, löydettyjen tyyppien ja lukumäärien sijoittaminen `Tyypit`-listaan ja `Tyypit`-listan käsittely `sort`-predikaatilla. Operaatioilla on yksi mahdollinen suoritusjärjestys. Näin ollen proseduraalisuusarvoksi saadaan:

```
3/1 + 4/1 = 7
```

SWI-Prolog-kielen valmispredikaattien kummassakaan analyyttisessa esimerkissä (32, 33) ei suoriteta muuttujien sidontaa ja suoritetaan yksi operaatio, joten proseduraalisuusarvoksi tulee 1. Kun lasketaan keskiarvo, saadaan SWI-Prolog-kielen valmispredikaattien proseduraalisuus arvoksi:

```
(1 + 1 + 1 + 4 + 7 + 1 + 1) / 7 = 2,29.
```

RDF/XML-relaatioratkaisun arvot ovat poimi ja valitse -kyselyissä (esimerkit 18, 19 ja 20) samat kuin SWI-Prolog-kielen valmispredikaateillakin. Ensimmäisessä aggregointiesimerkissä (26) arvo on sama kuin SWI-Prolog-kielen valmispredikaateilla, koska käytetään samaa predikaattia. Toisessa aggregointikyselyssä (esimerkki 27) `sort`-predikaatti on siirretty `findall_without_duplicates`-predikaatin toteutukseen, mutta muuten kyseessä on samanlainen kysely. Tässä esimerkissä on yksi sidottava muuttuja vähemmän ja yksi operaatio vähemmän. Arvoksi tulee siis:

```
2/1 + 3/1 = 5.
```

Kahden analyyttisen esimerkin (34 ja 36 (35 jätetään pois samankaltaisuuden vuoksi)) proseduraalisuusarvo on sama. Kummassakaan ei suoriteta muuttujien sidontaa ja molemmissa on vain yksi operaatio, joten kummankin esimerkin proseduraalisuusarvoksi tulee 1. Kolmannessa kyselyssä (esimerkki 37) suoritetaan yksi muuttujien sidonta, kun löydetyt RDF-predikaatit ja objektit sijoitetaan listaan `Tulokset`. Operaatioita kyselyssä on kolme. Ensin etsitään subjektista saavutettava muuttuja, toiseksi katsotaan, että onko se literaali ja kolmanneksi sijoitetaan se listaan. Tälle on yksi mahdollinen järjestys. Arvoksi tulee siis:

```
1/1 + 3/1 = 4
```

Kun lasketaan arvot yhteen, saadaan XML/RDF-relaation proseduraalisuusarvoksi:

```
(1 + 1 + 1 + 4 + 5 + 1 + 1 + 4) / 8 = 2,25.
```

Laskettujen arvojen perusteella erot tutkittujen ratkaisujen proseduraalisuudessa ovat varsin pieniä, kun huomioidaan että Weltyn ja Stemplen [1981] tutkimien kielien proseduraalisuusarvo vaihteli välillä 3,5 – 66,0. Proseduraalisuusarvojen pieni ero johtuu erityisesti siitä, että tässä työssä käytetyt esimerkit ovat suhteellisen yksinkertaisia.

7.4. Keskustelua

Kaikki lähestymistavat pystyvät tekemään hakuja varsin monipuolisesti. SPARQL-lähestymistavan vahvuuksia ovat erityisesti sen monipuoliset kyselytyypit `SELECT`, `CONSTRUCT`, `ASK` ja `DESCRIBE`, joista on vielä mahdollista tehdä varsin monimutkaisia. Lisäksi on huomattava, että tässä työssä esiteltiin vain pieni osuus SPARQL-kielen toiminnallisuudesta. SPARQL mahdollistaakin monipuolisen RDF-tiedon käsittelyn. Lisäksi SPARQL-kielen ja SQL-kielen syntaktinen samankaltaisuus edesauttaa kielen käyttöön ottamista.

RDF/XML-relaatioon tehdyt kyselyt ja SWI-Prolog-kielen valmispredikaatit olivat keskenään varsin samankaltaisia, kun asiaa tarkastellaan käyttäjän kannalta. Poimi ja valitse -kyselyissä sekä aggregointikyselyissä nämä ovat lähes samanlaisia, joskin muutama predikaatti on nimetty eri tavalla (tämä ei ole pakollista, mutta predikaatit haluttiin nimetä kuvaavammin kuin SWI-Prolog-kielessä on tehty). Vahvuutena näillä on SPARQL-kieleen verrattuna se, että nämä kyselyt voi ilmaista paljon tiiviimmin. Kuitenkin kun kyselyt monimutkaistuvat, saattaa SPARQL-kielen jäsentely helpottaa kyselyn hahmottamista ja tulkintaa.

RDF/XML-relaatio erottuu edukseen analyyttisissä kyselyissä, joissa se tarjoaa sellaisia mahdollisuuksia, joita ei ollut muissa vaihtoehdoissa. Kun analysoidaan sitä, voiko jonkin tietolähteen saavuttaa käsillä olevasta tietolähteestä, tarvitsee vain katsoa, että alkavatko haluttujen resurssien indeksit samalla luvulla tai jos tietoa ei löydetä suoraan tietolähteestä, siirrytään tekemään sama tutkimus toiseen tietolähteeseen, jonka URI löytyi ensimmäisestä tietolähteestä. SPARQL-kielessä tähän tarvittaisiin melko monimutkainen kysely ja valmispredikaateilla tarvittaisiin RDF-mallin laajennus.

Muuten mikään lähestymistavoista ei varsinaisesti erottunut edukseen muihin verrattuna. Joissain tilanteissa SPARQL on selkeämpi kuin vaihtoehdot ja toisinaan taas SWI-Prologin valmispredikaatit tai RDF/XML-relaatio on parempi. Kuitenkin voitaisiin sanoa, että yksinkertaisissa kyselyissä RDF/XML-relaatio sekä SWI-Prolog-kielen valmispredikaatit ovat yksinkertaisuutensa vuoksi käytettävyydeltään SPARQL-kieltä parempia, mutta monimutkaisissa kyselyissä SPARQL on kenties parempi käyttää. Poikkeuksena tähän ovat analyyttiset kyselyt, joissa RDF/XML-relaatio pystyy tekemään sellaisiakin hakuja, joihin SPARQL tai SWI-Prolog-kielen valmispredikaatit eivät kykene. Lisäksi RDF/XML-relaatio tarjoaa eksplisiittistä tietoa tutkivassa kyselyssä mahdollisuuden toteuttaa kyseinen kysely helpommin kuin SPARQL, joskaan ei välttämättä helpommin kuin valmispredikaatit.

8. Johtopäätökset

RDF-tiedon käsittelyyn on olemassa monenlaisia välineitä. Tässä työssä kehitettiin XML-relaatioon [Niemi and Järvelin, 2006] perustuva esitystapa RDF/XML-relaatio. Tätä lähestymistapaa verrattiin sitten SPARQL-kieleen ja SWI-Prolog-kielen RDF-kirjaston valmispredikaatteihin.

Vertailussa huomattiin, että RDF/XML-relaatio tarjoaa joitakin etuja vaihtoehtoihin nähden. Näitä olivat yksinkertaisempi syntaksi kuin SPARQL-kielessä. Kehittämällä ilmaisutavan päälle kunnollinen kyselykieli, tätä puolta aiheesta voidaan vielä korostaa.

Lisäksi lähestymistapa tarjoaa mahdollisuuksia tehdä sellaisia analyyttisiä kyselyjä aineistoon, joita on joko mahdoton tai kovin hankala tehdä vertailukohteissa. RDF/XML-relaation rakenteesta johtuen nämä ovat taasen yksinkertaisia toteuttaa sen avulla. Lisäksi siitä on mahdollista tehdä käyttäjäystävällisempi suunnittelemalla siinä käytettävä kyselykieli (oikea kieli tai predikaatit) sellaiseksi.

RDF/XML-relaatiossa on potentiaalia uuden RDF-kolmikkovaraston pohjaksi. Toisaalta samoja tekniikoita voi hyödyntää myös pelkän XML-relaation avulla muuhunkin kuin RDF-tietolähteisiin.

On kuitenkin huomioitava, että tässä työssä tutkimusta tehtiin vain RDF-dataan perustuen. Kun käyttöön otettaisiin lisäksi vielä RDFS ja OWL, niin SPARQL-kielen ja Prologin valmispredikaattien toiminnallisuus kasvaisi. Tällöin on mahdollista, että XML/RDF-relaatiolla ei olisi enää sitä etua, joka sillä pelkän RDF-datan kanssa on. Mielenkiintoista olisikin tutkia, miten RDF/XML-relaatio vertautuisi vertailukohteisiin siinä tilanteessa.

Viiteluettelo

Apache Jena. 2016. *Apache Jena*. https://jena.apache.org/. Retrieved 26.11.2016.

Paavo Arvola, Marko Junkkari and Jaana Kekäläinen. 2005. Generalised contextualization method for XML information retrieval. In: *Proc. of the 14th ACM international conference on Information and knowledge management (CIKM'05)*, 20-27.

Tim Berners-Lee, James Hendler and Ora Lassila. 2001. The semantic web. *Scientific American* 284, 5, 34–43.

Abraham Bernstein, James Hendler and Natalya Noy. 2016. A new look at the semantic web. *Communications of the ACM* 59, 9, 35–37.

Tim Bray, Dave Hollander, Andrew Layman, Richard Tobin and Henry S. Thompson. 2009. *Namespaces in XML 1.0 (Third Edition)*. https://www.w3.org/TR/xml-names/. Retrieved 26.11.2016.

Tony Coates, Dan Connolly, Diana Dack, Leslie Daigle, Ray Denenberg, Martin Dürst, Paul Grosso, Sandro Hawke, Renato Iannella, Graham Klyne, Larry Massinter, Michael Mealling, Mark Needleman and Norman Walsh. 2001. *URIs, URLs, and URNs: Clarifications and Recommendations 1.0*. https://www.w3.org/TR/uri-clarification/. Retrieved 25.11.2016.

Richard Cyganiak, David Wood and Markus Lanthaler. 2014. *RDF 1.1 Concepts and Abstract Syntax*. https://www.w3.org/TR/rdf11-concepts/. Retrieved 28.1.2016.

Eclipse RDF4J. 2016. *RDF4J*. http://rdf4j.org/. Retrieved 26.11.2016.

Emanuele Della Valle and Stefano Ceri. 2011. Querying the semantic web: SPARQL. In: John Dominque, Dieter Fensel and James A. Hendler (eds.), *Handbook of Semantic Web Technologies.* Springer, 299-365.

Melvil Dewey. 1922. *Decimal Classification and Relativ Index for Libraries and Personal Use in Arranjing for Immediate Reference Books, Pamflets, Clippings, Pictures, Manuscript Notes and Other Material*. Lake Placid Club, N.Y.: Forest Press.

John Dominque, Dieter Fensel and James A. Hendler. 2011. Introduction to the semantic web technologies. In: John Dominque, Dieter Fensel and James A. Hendler (eds.), *Handbook of Semantic Web Technologies.* Springer, 3-43.

Fabien L. Gandon, Reto Krummenacher, Sung-Kook Han and Ioan Toma. 2011. Semantic annotation and retrival: RDF. In: John Dominque, Dieter Fensel and James A. Hendler (eds.), *Handbook of Semantic Web Technologies.* Springer, 117-157.

Fabien Gandon and Guus Schreiber. 2014. *RDF 1.1 XML Syntax*. https://www.w3.org/TR/rdf-syntax-grammar/. Retrieved 8.2.2016.

Steve Harris and Andy Seaborne. 2013. SPARQL 1.1 Query Language. https://www.w3.org/TR/sparql11-query/. Retrieved 4.7.2016.

Matthias Jarke and Yannis Vassiliou. 1985. A framework for choosing a database query language. *Computing Surveys* 17, 3, 313–340.

Matti Lassila. 2013. *Kahden XML-kyselykielen vertaileva käyttäjätutkimus.* Pro gradu -tutkielma. Informaatiotieteiden yksikkö, Tampereen yliopisto.

Eric Miller, Bob Schloss, Ora Lassila and Ralph R. Swick. 1997. *Resource Description Framework (RDF) Model and Syntax*. https://www.w3.org/TR/WD-rdf-syntax-971002/. Retrieved 28.1.2016.

Timo Niemi and Kalervo Järvelin. 2006. *Another look at XML*. Report A-2006-1. University of Tampere, Dept. of Computer Science.

Timo Niemi, Turkka Näppilä and Kalervo Järvelin. 2009. A relational data harmonization approach to XML. *Journal of Information Science* 35, 5, 571–601.

Richard O'Keefe and Jan Wielemaker. 2016. *A.1 library(aggregate): Aggregation operators on backtrackable predicates.* http://www.swi-prolog.org/pldoc/doc_for?object=section(2,%27A.1%27,swi(%27/doc/Manual/aggregate.html%27)). Retrieved 1.9.2016.

Eric Prud'hommeaux and Andy Seaborne. 2008. *SPARQL Query Language for RDF*. https://www.w3.org/TR/rdf-sparql-query/. Retrieved 23.11.2016.

Phyllis Reisner. 1981. Human factors studies in database query languages: a survey and assessment. *Computing Surveys* 13, 1, 13–31.

Nigel Shadbolt, Wendy Hall and Tim Berners-Lee. 2006. The semantic web revisited. *IEEE Intelligent Systems*, 21, 3, 96-101.

Manu Sporny, Dave Longley, Greg Kellogg, Markus Lanthaler and Niklas Lindström. 2014. *JSON-LD 1.0*. https://www.w3.org/TR/json-ld/. Retrived 26.11.2016.

William Stanek. 2014. *XML, DTDs, Schemas: The Personal Trainer (2)*. Stanek & Associates.

SWI-Prolog A. 2016. *SWI-Prolog's features.* http://www.swi-prolog.org/features.html. Retrieved 26.11.2016.

SWI-Prolog B. 2016. *Library(lists): List manipulation*. http://www.swi-prolog.org/pldoc/man?section=lists. Retrieved 12.9.2016.

SWI-Prolog C. 2016. *Comparison and Unification of Terms*. http://www.swi-prolog.org/pldoc/man?section=compare. Retrieved 6.12.2016.

SWI-Prolog D. 2016. *Predicate findall/3*. http://www.swi-prolog.org/pldoc/man?predicate=findall/3. Retrieved 30.12.2016.

SWI-Prolog E. 2016. *Predicate sort/2*. http://www.swi-prolog.org/pldoc/man?predicate=sort/2. Retrieved 30.12.2016.

Igor Tatarinov, Stratis D. Viglas, Kevin Beyer, Jayavel Shanmugasundaram, Eugene Shekita and Chun Zhang. 2002. Storing and querying ordered XML using a relational database system. In: *Proc. of the 2002 ACM SIGMOD International Conference on Management of Data*, 204-215.

Charles Welty and David W. Stemple. 1981. Human factors comparison of a procedural and a nonprocedural query language. *ACM Transactions on Database Systems* 6, 4, 626–649.

Jan Wielemaker. 2016. *SWI-Prolog Semantic Web Library 3.0*. http://www.swi-prolog.org/pldoc/doc_for?object=section(%27packages/semweb.html%27). Retrieved 7.7.2016.

Liite 1: Lintudata RDF/XML-muodossa

```
<?xml version="1.0"?>
<rdf:RDF
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:esim="http://www.jokinesimerkkisanasto.fi/#">
      <rdf:Description rdf:about="http://www.keksittylintusivu.fi/sinisorsa">
            <esim:tyyppi>Lintu</esim:tyyppi>
            <esim:nimi>Sinisorsa</esim:nimi>
            <esim:lat_nimi>Anas platyrhynchos</esim:lat_nimi>
            <esim:pituus>50-60cm</esim:pituus>
            <esim:siipien_karkivali>81-95cm</esim:siipien_karkivali>
            <esim:lahko rdf:resource="http://www.keksittylintusivu.fi/sorsalinnut" />
            <esim:heimo rdf:resource="http://www.keksittylintusivu.fi/sorsa_heimo" />
            <esim:suku rdf:resource="http://www.keksittylintusivu.fi/sorsa_suku" />
            <esim:fiktiossa>
                  <rdf:Description rdf:about="http//www.keksittylintusivu.fi/repe">
                        <esim:nimi>Repe</esim:nimi>
                  </rdf:Description>
            </esim:fiktiossa>
      </rdf:Description>
      <rdf:Description rdf:about="http://www.keksittylintusivu.fi/saaksi">
            <esim:tyyppi>Lintu</esim:tyyppi>
            <esim:nimi>Saaksi</esim:nimi>
            <esim:lat_nimi>Pandion haliaetus</esim:lat_nimi>
            <esim:pituus>52-60cm</esim:pituus>
            <esim:siipien_karkivali>152-167cm</esim:siipien_karkivali>
            <esim:lahko   rdf:resource="http://www.keksittylintusivu.fi/paivapetolinnut"
/>
            <esim:heimo rdf:resource="http://www.keksittylintusivu.fi/haukat" />
            <esim:suku rdf:resource="http://www.keksittylintusivu.fi/pandion" />
      </rdf:Description>
      <rdf:Description rdf:about="http://www.keksittylintusivu.fi/laulujoutsen">
            <esim:tyyppi>Lintu</esim:tyyppi>
            <esim:nimi>Laulujoutsen</esim:nimi>
            <esim:latNimi>Cygnus cygnus</esim:latNimi>
```

```
            <esim:pituus>140-160cm</esim:pituus>
            <esim:siipienKarkivali>205-235cm</esim:siipienKarkivali>
            <esim:lahko rdf:resource="http://www.keksittylintusivu.fi/sorsalinnut" />
            <esim:heimo rdf:resource="http://www.keksittylintusivu.fi/sorsa_heimo" />
            <esim:suku rdf:resource="http://www.keksittylintusivu.fi/joutsenet" />
      </rdf:Description>
      <rdf:Description rdf:about="http://www.keksittylintusivu.fi/kirjosieppo">
            <esim:tyyppi>Lintu</esim:tyyppi>
            <esim:nimi>Kirjosieppo</esim:nimi>
            <esim:lat_nimi>Ficedula hypoleuca</esim:lat_nimi>
            <esim:pituus>12-13,5cm</esim:pituus>
            <esim:lahko rdf:resource="http://www.keksittylintusivu.fi/varpuslinnut" />
            <esim:heimo rdf:resource="http://www.keksittylintusivu.fi/siepot" />
            <esim:suku rdf:resource="http://www.keksittylintusivu.fi/kirjosiepot" />
      </rdf:Description>
      <rdf:Description rdf:about="http://www.keksittylintusivu.fi/joutsenet">
            <esim:tyyppi>Suku</esim:tyyppi>
            <esim:nimi_latinaksi>Cygnus</esim:nimi_latinaksi>
      </rdf:Description>
      <rdf:Description rdf:about="http://www.keksittylintusivu.fi/sorsalinnut">
            <esim:tyyppi>Lahko</esim:tyyppi>
            <esim:lat_nimi>Anseriformes</esim:lat_nimi>
      </rdf:Description>
</rdf:RDF>
```

Liite 2: Lintudata RDF/XML-relaationa

```
d(rdf_description, e, [1]).
d(rdf_about, a, [1, 1]).
d("http://www.keksittylintusivu.fi/sinisorsa", u , [1, 1, 1]).
d(esim_tyyppi, e, [1, 2]).
d("lintu", v, [1, 2, 1]).
d(esim_nimi, e, [1, 3]).
d("sinisorsa", v, [1, 3, 1]).
d(esim_lat_nimi, e, [1, 4]).
d("anas platyrhynchos", v, [1, 4, 1]).
d(esim_pituus, e, [1, 5]).
d("50-60cm", v, [1, 5, 1]).
d(esim_siipien_karkivali, e, [1, 6]).
d("81-95cm", v, [1, 6, 1]).
d(esim_lahko, e, [1, 7]).
d(rdf_resource, a, [1, 7, 1]).
d("http://www.keksittylintusivu.fi/sorsalinnut", u, [1, 7, 1, 1]).
d(esim_heimo, e, [1, 8]).
d(rdf_resource, a, [1, 8, 1]).
d("http://www.keksittylintusivu.fi/sorsa_heimo", u, [1, 8, 1, 1]).
d(esim_suku, e, [1, 9]).
d(rdf_resource, a, [1, 9, 1]).
d("http://www.keksittylintusivu.fi/sorsa_suku", u, [1, 9, 1, 1]).
d(esim_fiktiossa, e, [1, 10]).
d(rdf_description, e, [1, 10, 1]).
d(rdf_about, a, [1, 10, 1, 1]).
d("http://www.keksittylintusivu.fi/repe", u, [1, 10, 1, 1, 1]).
d(esim_nimi, e, [1, 10, 1, 2]).
d("repe", v, [1, 10, 1, 2, 1]).
d(rdf_description, e, [2]).
d(rdf_about, a, [2, 1]).
d("http://keksittylintusivu.fi/saaksi", u , [2, 1, 1]).
d(esim_tyyppi, e, [2, 2]).
d("lintu", v, [2, 2, 1]).
d(esim_nimi, e, [2, 3]).
```

```
d("saaksi", v, [2, 3, 1]).
d(esim_lat_nimi, e, [2, 4]).
d("pandion haliaetus", v, [2, 4, 1]).
d(esim_pituus, e, [2, 5]).
d("52-60cm", v, [2, 5, 1]).
d(esim_siipien_karkivali, e, [2, 6]).
d("152-167cm", v, [2, 6, 1]).
d(esim_lahko, e, [2, 7]).
d(rdf_resource, a, [2, 7, 1]).
d("http://www.keksittylintusivu.fi/paivapetolinnut", u, [2, 7, 1, 1]).
d(esim_heimo, e, [2, 8]).
d(rdf_resource, a, [2, 8, 1]).
d("http://www.keksittylintusivu.fi/haukat", u, [2, 8, 1, 1]).
d(esim_suku, e, [2, 9]).
d(rdf_resource, a, [2, 9, 1]).
d("http://www.keksittylintusivu.fi/pandion", u, [2, 9, 1, 1]).
d(rdf_description, e, [3]).
d(rdf_about, a, [3, 1]).
d("http://www.keksittylintusivu.fi/laulujoutsen", u , [3, 1, 1]).
d(esim_tyyppi, e, [3, 2]).
d("lintu", v, [3, 2, 1]).
d(esim_nimi, e, [3, 3]).
d("laulujoutsen", v, [3, 3, 1]).
d(esim_latNimi, e, [3, 4]).
d("cygnus cygnus", v, [3, 4, 1]).
d(esim_pituus, e, [3, 5]).
d("140-160cm", v, [3, 5, 1]).
d(esim_siipien_karkivali, e, [3, 6]).
d("205-235cm", v, [3, 6, 1]).
d(esim_lahko, e, [3, 7]).
d(rdf_resource, a, [3, 7, 1]).
d("http://www.keksittylintusivu.fi/sorsalinnut", u, [3, 7, 1, 1]).
d(esim_heimo, e, [3, 8]).
d(rdf_resource, a, [3, 8, 1]).
d("http://www.keksittylintusivu.fi/sorsa_heimo", u, [3, 8, 1, 1]).
d(esim_suku, e, [3, 9]).
d(rdf_resource, a, [3, 9, 1]).
```

```
d("http://www.keksittylintusivu.fi/joutsenet", u, [3, 9, 1, 1]).
d(rdf_description, e, [4]).
d(rdf_about, a, [4, 1]).
d("http://keksittylintusivu.fi/kirjosieppo", u , [4, 1, 1]).
d(esim_tyyppi, e, [4, 2]).
d("lintu", v, [4, 2, 1]).
d(esim_nimi, e, [4, 3]).
d("kirjosieppo", v, [4, 3, 1]).
d(esim_lat_nimi, e, [4, 4]).
d("fidecula hypoleuca", v, [4, 4, 1]).
d(esim_pituus, e, [4, 5]).
d("12-13,5cm", v, [4, 5, 1]).
d(esim_lahko, e, [4, 6]).
d(rdf_resource, a, [4, 6, 1]).
d("http://www.keksittylintusivu.fi/varpuslinnut", u, [4, 6, 1, 1]).
d(esim_heimo, e, [4, 7]).
d(rdf_resource, a, [4, 7, 1]).
d("http://www.keksittylintusivu.fi/siepot", u, [4, 7, 1, 1]).
d(esim_suku, e, [4, 8]).
d(rdf_resource, a, [4, 8, 1]).
d("http://www.keksittylintusivu.fi/kirjosiepot", u, [4, 8, 1, 1]).
d(rdf_description, e, [5]).
d(rdf_about, a, [5, 1]).
d("http://www.keksittylintusivu.fi/joutsenet", u , [5, 1, 1]).
d(esim_tyyppi, e, [5, 2]).
d("suku", v, [5, 2, 1]).
d(esim_nimi_latinaksi, e, [5, 3]).
d("cygnus", v, [5, 3, 1]).
d(rdf_description, e, [6]).
d(rdf_about, a, [6, 1]).
d("http://www.keksittylintusivu.fi/sorsalinnut", u , [6, 1, 1]).
d(esim_tyyppi, e, [6, 2]).
d("lahko", v, [6, 2, 1]).
d(esim_lat_nimi, e, [6, 3]).
d("anseriformes", v, [6, 3, 1]).
```

Liite 3: Im_predecessor-predikaatin toteutus

```
im_predecessor(Ind, []) :-
  length(Ind, 1).

im_predecessor(Ind, Ind1) :-
  d(_, _, Ind),
  reverse(Ind, [H|T]),
  reverse(T, Ind1).
```

`Im_predecessor`-predikaatti hakee indeksin (`ind`) edeltäjän (`ind1`), joka on yhden askeleen `ind`-indeksiä ylempänä. Esimerkiksi, jos `ind` on `[1, 5]` niin sen edeltäjä on `[1]`. Predikaatissa on kaksi osaa. Ylempi osa kertoo sen, että silloin kun indeksissä on yksi numero, sillä ei ole edeltäjää. Tällöin predikaatti palauttaa tyhjän listan (`[]`). Alemmassa versiossa tutkitaan ensin, sisältääkö jokin relaatio kyseisen indeksin. Jos sisältää, niin se hakee edeltäjän (`Ind1`). Edeltäjä (`Ind1`) haetaan kääntämällä `ind`-lista ympäri, ottamalla käännetyn listan häntä (`T`) ja kääntämällä se takaisin. Näin saadaan alkuperäinen indeksi (`Ind`) ilman sen viimeistä numeroa (`H`).

Liite 4: Findall_without_duplicates-predikaatin toteutus

```
findall_without_duplicates(Variables, Query, List) :-
     findall(Variables, Query, Results),
     sort(Results, List).
```

Predikaatille kerrotaan muuttujat (`Variables`), jotka etsitään kyselyllä (`Query`) ja kerätään listaan (`List`). Predikaatti hakee aluksi `findall`-predikaatilla kaikki mahdolliset tulokset halutuilla muuttujilla ja kyselyllä listaan `Results`. Seuraavaksi se käyttää `sort`-predikaattia lajittelemaan `Results`-listan ja karsimaan siitä pois duplikaatit. Näin se muodostaa lopullisen vastauslistan (`List`).

Liite 5: Esimerkkisanaston muodollisempi esitys

Tyyppi:
Erottaa erityyppiset tietoresurssit toisistaan. Saa arvokseen tekstimuotoisen muuttujan. Tyyppi voi olla lintu, suku tai lahko. Tyyppi voisi olla myös heimo, mutta sen tyyppistä tietoresurssia ei ole käytetty tässä työssä, joten sitä ei määritellä.

Lintu:
Kun tyyppinä on lintu, se kertoo, että määriteltävä tietoresurssi kuvaa lintua. Lintu voi saada seuraavat muuttujat:

Nimi:
Arvoksi tulee teksti, joka kertoo tietoresurssin kuvaaman linnun nimen.
Lat_nimi:
Arvoksi tulee teksti, joka kertoo tietoresurssin kuvaaman linnun latinankielisen nimen.
Pituus:
Arvoksi tulee teksti, joka kertoo tietoresurssin kuvaaman linnun pituuden.
Siipien_karkivali:
Arvoksi tulee teksti, joka kertoo tietoresurssin kuvaaman linnun siipien kärkivälin.
Lahko:
Arvoksi tulee joko URI-osoite, joka viittaa tietoresurssin kuvaaman linnun lahkoa kuvaavaan tietoresurssiin, tai sitä seuraa lahko-tietoresurssin kuvaus sisällytettynä linnun tietoresurssin kuvaukseen.
Suku:
Arvoksi tulee joko URI-osoite, joka viittaa tietoresurssin kuvaaman linnun sukua kuvaavaan tietoresurssiin, tai sitä seuraa suku-tietoresurssin kuvaus sisällytettynä linnun tietoresurssin kuvaukseen.
LatNimi:
Arvoksi tulee teksti, joka kertoo tietoresurssin kuvaaman linnun latinankielisen nimen.
Fiktiossa:
Arvoksi tulee joko URI-osoite, joka viittaa fiktiiviseen lintuun, joka edustaa tietoresurssin kuvaamaa lintulajia, tai fiktiivisen linnun tietoresurssi on kuvattu alkuperäisen tietoresurssin yhteydessä.

Lahko:
Kun tyyppinä on lahko, se kertoo, että kuvattava tietoresurssi kuvaa jotakin eläinlahkoa. Lahko voi saada seuraavat muuttujat:

Lat_nimi:
Arvoksi tulee teksti, joka kertoo tietoresurssin kuvaaman lahkon latinankielisen nimen.

Suku:
Kun tyyppinä on suku, se kertoo, että kuvattava tietoresurssi kuvaa jotakin lintusukua. Suku voi saada seuraavat muuttujat:

Nimi_latinaksi:
Arvoksi tulee teksti, joka kertoo tietoresurssin kuvaaman suvun latinankielisen nimen.