hyväksymispäivä arvosana

arvostelija

Puolirakenteisen tiedon kyselykielet

Atte Hinkka

Helsinki 3.5.2010

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Matemaattis-luonnontieteellinen tiedekunta	Tietojenkäsittelytieteen laitos

Tekijä — Författare — Author
Atte Hinkka

Työn nimi — Arbetets titel — Title

Puolirakenteisen tiedon kyselykielet

Oppiaine — Läroämne — Subject
Tietojenkäsittelytiede

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Kandidaatintutkielma	3.5.2010	17 sivua

Tiivistelmä — Referat — Abstract

Puolirakenteinen tieto on tietoa, jolla ei ole skeemaa, mutta se sisältää kuitenkin sisältönsä ohessa riittävästi tietoa rakenteestaan, jotta sitä voidaan kysellä. Esittelen tässä tutkielmassa kaksi puolirakenteisen tiedon kyselykieltä ja käyn vertaillen läpi niiden keskeisimmät ominaisuudet. Erityistä huomiota saavat polkukyselyt, sillä ne mahdollistavat kyselyjen tekemisen rakenteeltaan heterogeenisiin puihin ja verkkoihin: rakenteisiin, joilla puolirakenteista tietoa kuvataan. Käyn läpi polkukyselyiden lisäksi puolirakenteisen tiedon yhtäsuuruus- ja vertailuoperaattorit sekä tiedon päivittämiskeinot.

ACM Computing Classification System (CCS):
H.2.3 [Languages]: *Query languages*

Avainsanat — Nyckelord — Keywords
kyselykielet, puolirakenteinen tieto, Lorel, UnQL, polkulauseke

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — övriga uppgifter — Additional information

Sisältö

1 Johdanto

Puolirakenteinen tieto on skeematonta, epäsäännöllistä, puutteellista ja rakenteeltaan muuttuvaa. Myös sen käyttökohteet ja käyttötavat voivat vaihtua tiedon elinkaaren aikana [Abi97]. Puolirakenteiset tietomallit mahdollistavat juuri tällaisen tiedon tallentamisen ja kyselemisen. Käsittelen tässä tutkielmassa puolirakenteisen tiedon kyselemiseen kehitettyjä kyselykieliä ja ominaisuuksia, jotka tekevät näistä kielistä kykeneviä vastaamaan kyselyihin, vaikka tieto, johon kyselyitä tehdään on rakenteeltaan vaihtelevaa.

Esimerkkejä puolirakenteisista tietokantajärjestelmistä ovat AceDB (A Caenorhabditis elegans database) [STM99], sekä BiBTeX-tietueet [Abi97]. AceDB on malliesimerkki puolirakenteisesta tietokannasta. Se on kehitetty geenien ja muun genetiikkaan liittyvän, tutkimuksen edistyessä rakenteiltaan muuttuvan tiedon tallentamiseen. AceDB:n keskeisiin ominaisuuksiin kuuluu mahdollisuus muuttaa helposti tietokannan skeemaa ja antaa yksittäiselle attribuutille mielivaltainen määrä arvoja. BiBTeX-viittaustietueiden attribuuttijoukkoja taas ei ole formaalisti rajattu, vaan ne joustavat käytön mukaan. Tässä tutkielmassa käsittelen Standfordin yliopiston Lore-projektissa kehitettyä Lorel-kyselykieltä [AQM+97] ja Pennsylvanian yliopistossa ja AT&T:n tutkimuslaboratoriossa kehitettyä UnQL-kyselykieltä [BDHS96]. Muita kokonaisvaltaisempia puolirakenteisen tiedon järjestelmiä ovat muun muassa Florid [LHL+98], joka lähestyy aihetta enemmän olio-ohjelmoinnin näkökulmasta ja Lore-projektin edeltäjä Tsimmis [HMGM97].

Puolirakenteiset tietokannat ovat rakenteeltaan puita tai verkkoja, joiden hahmottamiseen ja kyselemiseen erilaiset, joko polkujen tai tietokannan olioiden prototyyppejä käyttävät kyselyrakenteet ovat sopivia, sillä niiden avulla voidaan ilmaista luontevasti myös epäsäännöllisiä käsitteiden välisiä suhteita. Näistä nostan erityisesti esille *polkukyselyt* (path query). Myös tiedon tyypitys voi olla moninaista, joten puolirakenteisen tietokantajärjestelmän on osattava vertailla eri tietotyypein tallennettuja arvoja toisiinsa. Lisävaikeuksia tuottavat moniarvoiset attribuutit, sillä tietokantajärjestelmän tulee tuottaa järkeviä vastauksia kyselyihin, vaikka attribuutilla onkin yhdessä oliossa yhden sijasta monta arvoa.

Puolirakenteinen tieto oli 1990-luvulla World Wide Webin yleistyessä kuuma tutkimusaihe. WWW:tä ja internet-sivuja yritettiin tuolloin kuvata erilaisin puolirakenteisin mallein [FFLS97, AM99]. Sittemmin vastaava tietokantatutkimus on siirtynyt käsittelemään XML-tiedostoformaattia. Muun muassa Lorel-kielikin on toteu-

tettu myös XML:ää käsittelevään järjestelmään [GMW99]. Samat polkukyselyjen periaatteet pätevät myös XML-maailmassa ja puolirakenteisen tiedon 1990-luvun renessanssi onkin auttanut formalisoimaan polkukyselyt sekä niiden optimointiin tarvittavia tekniikoita uusia sovelluksia varten.

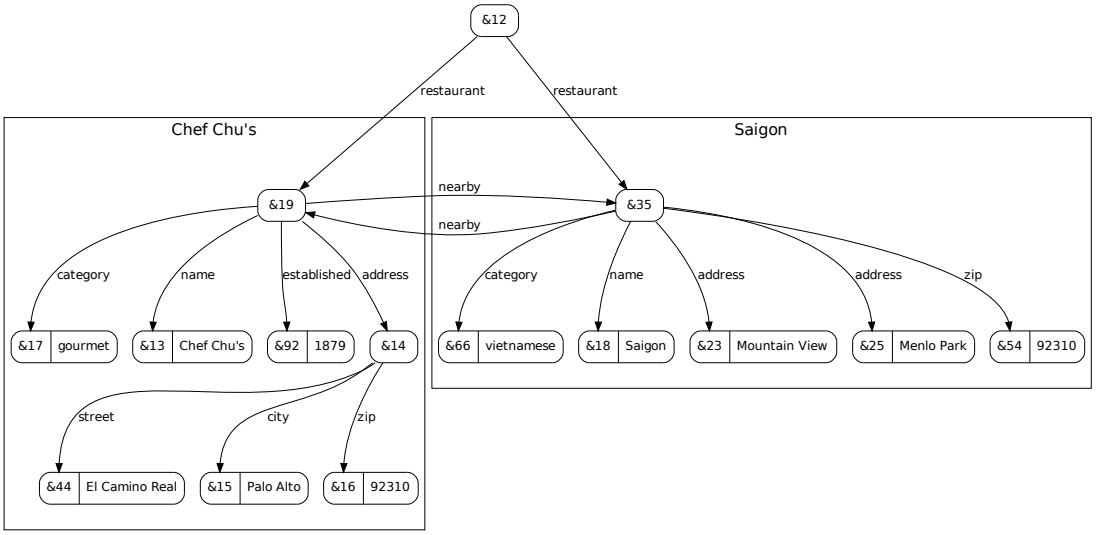
Tässä tutkielmassa käsittelen Lorel- ja UnQL-kieliä sekä näiden tietomalleja esimerkkeinä puolirakenteisista kyselykielistä ja tietojärjestelmistä, sillä ne ovat hyvin formalisoituja ja antavat järkevän kuvan aiheeseen liittyvästä teoriasta. Kohdassa 2 pureudun puolirakenteisen tiedon tietomalliin ja olion käsitteeseen puolirakenteisessa tiedossa. Sen jälkeen käsittelen puolirakenteisen tiedon peruskyselyitä kohdassa 3. Kohdassa 4 kerron puolirakenteisen tiedon vertailu- ja tyyppipäättelymekanismeista ja niiden vaikutuksesta kyselyjen tekemiseen. Muuttujiin ja kyselylausekkeiden eri osien väliseen suhteeseen keskityn kohdassa 5 ja kohdassa 6 käsittelen tapoja päivittää puolirakenteista tietoa.

2 Puolirakenteisen tietokannan oliot

Hyvä esimerkki puolirakenteisesti mallinnetusta tiedosta on Lorel-kyselykieltä käsittelevässä artikkelissa [AQM+97] esitetty ravintolatietokanta (kuva 1). Tietokanta kuvaa ravintoloita ja niihin liittyviä tietoja, eli käytännössä osoitteita, hintaluokkia ja ravintolatyyppejä. Kyseessä on siis erilaisia olioita sisältävä verkko. Lorel-kielen käyttämä *OEM*-tietomalli (Object Exchange Model) soveltuu keveytensä vuoksi hyvin puolirakenteisen tiedon tallentamiseen, sillä tallentamisessa ei tarvitse huolehtia samalla tavalla tiedon heterogeenisyydestä ja rakenteellisesta yhtenäisyydestä, kuin rakenteisemmassa *ODMG*-oliotietokantamallissa (Object Data Management Group) [Abi97, EN00, s. 361–363] ja relaatiomallissa.

Tämän OEM-tietomallia noudattavan verkon oliot voidaan jakaa kahteen päätyyppiin: *atomisiin arvo-olioihin* (atomic object) ja *yhdistelmäolioihin* (complex object), joilla kuvataan monipuolisempia tietotyyppejä. Ravintolat ja niihin liittyvät osoitteet ovat yhdistelmäolioita ja yhdistelmäolioihin nimetyin kaarin liitetyt attribuutit (nimi, katuosoite, puhelinnumero) ovat atomisia arvo-olioita.

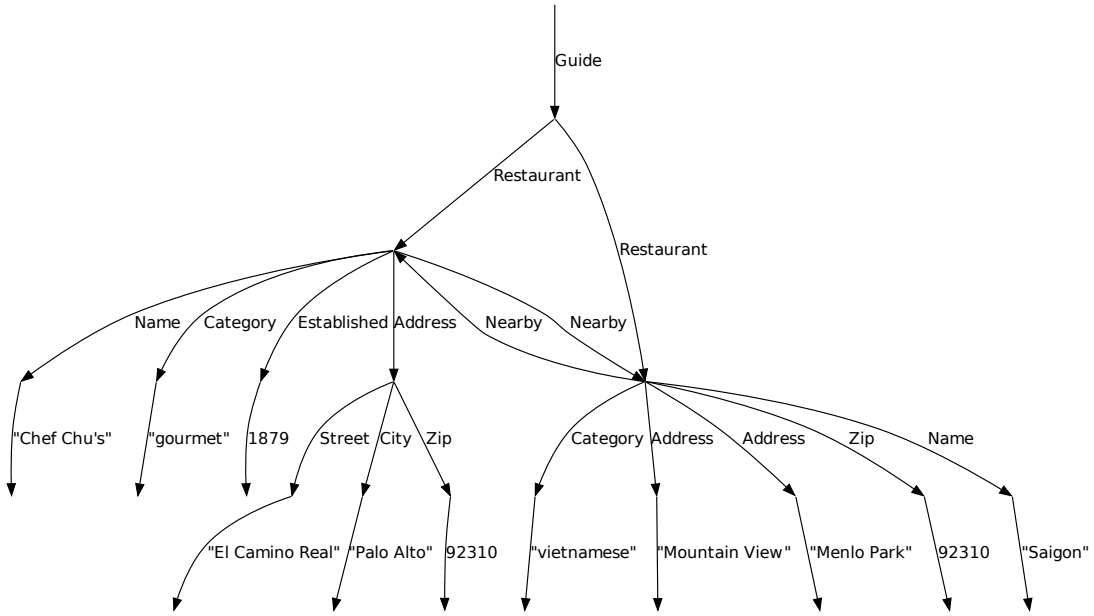
Tämä Lore-järjestelmän variaatio OEM-mallista perustuu aikaisempaan, Tsimmis-projektissa kehitettyyn, alkuperäiseen OEM-tietokantamalliin [PGMW95]. Merkittävin ero näiden kahden eri mallin välillä on tässä tutkielmassa ja Lore-projektissa käytetyn mallin kaarissa sijaitsevat *attribuuttinimet* (edge label) [Abi97].



Kuva 1: Ravintolatietokanta OEM-tietomallilla, &-merkit tarkoittavat olioidentiteettiä, Guide on nimetty solmu (&12). Mukailtu Lorel-artikkelissa [AQM+97] esitetyn tietokannan pohjalta.

UnQL-kielen [BDHS96] taustalla oleva tietomalli poikkeaa OEM-tietomallista siten, että se ei sisällä varsinaisia attribuuttisolmuja. Kaikki tieto on tallennettu kaariin, joiden nimet kuvaavat myös niiden arvoja. Malli mahdollistaa yhtä lailla moniarvoiset attribuutit kuin OEM-tietomallikin, mutta johtaa syvempien puiden käyttöön. Malli ei myöskään sisällä olioidentiteetin käsitettä, vaan käytössä ovat erilliset merkkijonoin toteutetut symbolit, jotka kuvaavat tietokannan attribuuttien nimiä ja ovat samalla tietokannan rakennetta kuvaavia elementtejä. UnQL:n tietomalli sisältää symbolien lisäksi myös muita yleisiä tietotyyppejä, kuten merkkijonoja sekä kokonais- ja reaalilukuja [BDHS96]. Kuvassa 2 on esitetty ravintolatietokanta UnQL:n solmuttomalla tietokantamallilla.

Ravintolatietokanta (kuvat 1 ja 2) ei kuitenkaan ole säännöllinen. Se sisältää tyypilliset puolirakenteisen tiedon epäsäännöllisyydet: tietoja (osoitteet) on esitetty eri muodoissa ja attribuuteilla voi olla useampi kuin yksi arvo. Ravintolaolioiden attribuuttijoukkoja ei myöskään ole rajoitettu millään tavalla. Näiden lisäksi ravintoloiden läheisyyttä toisiinsa ilmaisevat *nearby*-kaaret monimutkaistavat kyselyitä.



Kuva 2: Ravintolatietokanta UnQL:n tietomallilla.

3 Polkukyselyt

Tietokannan *verkkoilmentymien* (graph instance) perusteella tehtävät haut [HP93], tässä tutkielmassa erityisesti polkukyselyt, ovat puolirakenteisen tiedon viitekehyksessä tapoja löytää tiedetyllä tavalla toisiinsa liittyviä ja yhdessä esiintyviä olioita tai arvoja. Polkukyselyn pääajatuksena on edetä oliosta toiseen käyttäen nimettyjä kaaria suuntaviittoina. Polkukyselyn polku jaetaan *polunosiin*, joista polku koostuu. Kyselyä suoritettaessa verkossa edetään aina yksi kaari kerrallaan polkua pitkin, kunnes polku on käyty läpi.

3.1 Polkukyselyiden yleinen muoto

Polkukyselyä muodostettaessa on määriteltävä juuri, josta kyselyä lähdetään suorittamaan. Juuri voi olla olio, Lorel-järjestelmän tapauksessa *tietokantanimi* (label) [AQM+97] tai UnQL-järjestelmässä kaari [BDHS96]. Jos tietokanta on rakenteeltaan puu, yleensä kyselyä lähdetään suorittamaan puun juuresta. Käyttäen kuvan 1 ravintolatietokantaa, seuraavassa kyselyssä kyselyn juuri on tietokantanimi *Guide*, joka viittaa olioon &12:

select R **from** *Guide.restaurant* R ja

restaurant on kaari, jota pitkin polku etenee. Kysely palauttaa oliot &19 ja &35, eli kummatkin ravintolat.

Yleisesti polkukyselyt muodostuvat *select*, *from* ja *where*-lauseista, jotka muodostavat yhdessä *select-from-where-lausekkeen*. Sama kyselylausekkeen yleismuoto on käytössä niin Lorel-kielen toteutuksen pohjana olevassa OQL-kielessä (Object Query Language), [AQM+97] kuin SQL-kielessäkin [Cat95, Dat89]. Polkukyselyt voivat OQL- ja SQL-kyselyistä poiketen lisäksi sisältää *polkulausekkeita* (path expression). Lausekkeiden tarkka sisältö riippuu kielestä.

Select-from-where-lausekkeen select-lause kuvaa kyselystä haluttavaa tulosta, from-lauseessa määritellään, mistä oliojoukosta tai juuresta haku on tarkoitus suorittaa ja where-lauseessa suodatetaan from-lauseessa valittua kyselyn tulosten ehdokasjoukkoa kyselyssä määritellyin ehdoin. UnQL-kielessä from-lausetta ei ole, vaan se on korvattu toisella rakenteella.

Polkukyselyt voivat perustua joko pistenotaatiolla erotettujen kaarien muodostamaan polkuun tai yleisempiin *puulausekkeiisiin* (tree expression). Lorel-kielessä polkulauseet muodostetaan pistenotaatiolla [AQM+97], kun taas UnQL-kieli käyttää puulausekkeita. Puulausekkeet voidaan määritellä rekursiivisesti aaltosulkein ilmaistuiksi monikoiksi, jotka pitävät sisällään joko monikoita, symboleita tai arvoja. Monikoiden arvot erotellaan pilkkua käyttäen ja sisäkkäiset monikot muodostavat puun. Monikoita luetaan vasemmalta oikealle ja notaatio "⇒" on viittaus kaaresta (tai monikosta) sen viittaamaan toiseen kaareen, käytännössä siis toiseen monikkoon, symboliin tai arvoon. Monikko on tässä määritelmässä synonyymi kaarelle.

UnQL-kysely

select r **where** $\{Restaurant \Rightarrow \backslash r\} \leftarrow Guide$

palauttaa aikaisemmin mainitun Lorel-esimerkin mukaisesti kaikki tietokannan ravintolat. UnQL-kyselyt koostuvat select- ja where-lauseista. Select-lauseessa määritellään palautettava tulosjoukko ja where-lause koostuu kyselyyn tietoja lisäävistä generaattoreista ($\leftarrow$ *Guide*) ja hakutulosta rajaavista konditionaaleista. Tässä haun juureksi määritellään generaattorissa *Guide*-kaari, eli käytännössä koko tietokanta ja generaattorin täsmääjänä toimivana puulausekkeena toimii monikko, jossa *Restaurant*-kaari johtaa nimeämättömään kaareen, johon sidotaan muuttuja r. Tässä kyselyssä konditionaalia ei ole määritelty.

Attribuuttin arvojen määrän vaihtelevuus on tyypillinen puolirakenteisen tietokan-

nan ominaisuus [Abi97]. Tämä piirre otetaan huomioon myös kyselykielissä. Sekä Lorel- että UnQL-kielten kyselyt etenevät vain olemassaolevia kaaria pitkin. Jos polunosa puuttuu, ei polulla etenemistä jatketa. Esimerkiksi OQL-oliokyselykielessä olioattribuutin puuttuminen johtaisi virheeseen [AQM+97].

Formaalisti polku on tapa kuvata tietokantaverkon kaaria ja etenemistä verkossa. Polku alkaa juurisolmusta ja päättyy kun jokainen polun osa on kulutettu. Jokaista kuljettua kaarta kohden kuluu aina yksi polun osa. Jokaisella etenemisaskeleella jäljelläolevista ensimmäistä polunosaa verrataan mahdollisiin seuraaviin askeliin eli siitä yhdistelmäoliosta lähteviin kaariin, mihin edellisen etenemisaskeleen jälkeen eteneminen on päätynyt. Kysely etenee jokaiseen kaareen, jonka nimi vastaa polunosaa. Polunosien loppuessa palautetaan olio tai oliojoukko, johon tai joihin eteneminen on päättynyt.

Polkulausekkeet voidaan jakaa kahteen eri kategoriaan: yksinkertaisiin ja yleisiin. Yksinkertainen polkulauseke on eksplisiittisesti kuvattu polku tietokannassa, kun taas yleinen polkulauseke mahdollistaa mielivaltaisen syvien rakenteiden kyselemisen jokeri-ilmauksia ja muita monimutkaisempia polkulausekeoperaattoreita käyttämällä.

3.2 Yksinkertainen polkulauseke

Yksinkertaiset polkulausekkeet sisältävät vain nimettyjä kaaria. Lorel-kielellä yksinkertaiset polkulausekkeet ovat pistein eroteltuja kaarien nimiä sisältäviä lausekkeita, joita voidaan käyttää missä tahansa select-from-where-lausekkeen lauseessa [AQM+97]. UnQL-kielessä kysely tapahtuu where-lauseessa vertailemalla puulausekkeita tietokantaan.

UnQL-kielinen kaikki ravintolat palauttava kysely eroaa Lorel-kielellä tehdystä lähinnä siinä, ettei se sisällä from-lausetta. From-lauseen korvaa generaattori. Ravintolakaariin täsmäävää puulauseketta "$Restaurant \Rightarrow \backslash r$" verrataan *Guide*-kaaresta lähteviin kaariin. Tässä puulausekkeesta esitetään yksinkertainen muoto. Sisäisesti järjestelmä tulkitsee edellisen puulausekkeen samoin kuin aaltosuluin ympäröidynkin eli "$\{Restaurant \Rightarrow \backslash r\}$" ja "$Restaurant \Rightarrow \backslash r$" tarkoittavat samaa.

Lorel-kielistä kyselyä voidaan edelleen rajoittaa where-lauseessa. Seuraava kysely rajoittaa edellisen esimerkin tulosjoukkoa palauttamalla vain ravintolat, joiden nimi on "Chef Chu's":

select R **from** *Guide.restaurant* R

where *R.name* = "Chef Chu's".

Vastaavasti puulausekkeita käyttäen on ravintola-monikkoon lisättävä sisäkkäinen rakenne, joka täsmää *Name*-kaareen ja siitä lähtevaan arvo-kaareen:

select r **where**
$\{Restaurant \Rightarrow \{Name \Rightarrow$ "Chef Chu's"$\}\} \Rightarrow \backslash r \leftarrow Guide.$

Where-lauseen ehtoja on myös mahdollista yhdistää and ja or-operaattorein, kuten SQL-kielessäkin:

select R **from** *Guide.restaurant* R **where** *R.price* = "cheap"
and *R.city* = "Mountain View".

Sama on ilmaistavissa myös puulausekkeiden avulla: ehtojen unioni ilmaistaan samasta kaaresta lähtevien kaarien joukkona:

select r **where** $\{Restaurant \Rightarrow$
$\{Price \Rightarrow$ "Cheap", $City \Rightarrow$ "Mountain View"$\}\} \Rightarrow \backslash r \leftarrow Guide.$

Kuten edellisestä esimerkistä käy ilmi, puulausekkeilla pystytään ilmaisemaan suoraan puita, mutta pisteillä erotelluilla polkulausekkeilla käytännössä vain yksiulotteisia listoja. Edellisessä esimerkissä listoista rakennettiin eräänlainen puu sitomalla listan solmuja muuttujiin ja määrittelemällä niistä lähteviä polkulausekkeita.

Lorel-kielessä yksinkertaisen polkulausekkeen formaali määritelmä kuuluu näin: yksinkertainen polkulauseke on jono Z.l_1....l_n, missä l_1,....,l_n ovat kaarten nimiä ja Z on olion nimi tai muuttujanimi, joka viittaa olioon. Tietokannan polku on jono o_0,l_1,o_1,...,l_n,o_n, missä o_i:t ovat olioita ja jokaista i:tä kohtaan on kaari nimeltään l_i kaarien o_{l-1} ja o_i:n välillä. Lähtien oliosta $Z = o_0$ voi verkossa olla useita polkuja, jotka täsmäävät yksinkertaiseen polkulausekkeeseen $Z.l_1....l_n$ [AQM+97].

Polkulausekkeen täsmääminen tarkoittaa sitä, että olioon on kyselyn juuresta päästävissä kyseistä polkulauseketta käyttäen tai polkulausekkeen polunosan kohdalla sitä, että polunosan nimi voidaan vertailuoperaatiolla todeta samaksi kuin kyseessä olevan verkon kaaren nimi. Puulausekkeita verrataan tietokantaan vastaavalla tavalla ja suurin ero onkin lähinnä se, että kyseessä on kahden puun vertailu toisiinsa. Täsmääminen on määriteltävissä niin, että jokaista puun kaarta kohden löytyy vastaava kaari määritellystä osasta tietokantaa.

Lorel-kielessä kyselyn vastaus on kokoelma OEM-olioita. Käytännössä jokainen kysely tuottaa uuden vastausolion, jonka avulla uusi tulosjoukko ilmaistaan. Uusi vastausolio on siis olioidentiteetillinen ilmentymä, kuten muutkin tietokannan oliot

ja se voidaan tietokantanimeen sitomalla tallentaa pysyvästi tietokantaan. Lorel-järjestelmän määritelmän mukaan oliot, joihin ei ole polkua mistään tietokantanimestä, kerätään roskina. Tämä mahdollistaa uuden olion luomisen jokaiselle vastaukselle. Tietokannassa jo olemassa olleet oliot ovat osa tätä uutta tulosoliota, joka on rakenteeltaan yhdistelmäolio [AQM+97].

UnQL-kielessä puu kuvataan monikkoina, joita voidaan yhdistellä eri muotoon vastausta rakennettaessa. UnQL-kyselyt koostuvat select- ja where-lauseista, joista select-lauseessa kuvataan vastauspuun rakenne. Vastauspuussa voidaan käyttää where-lauseessa määriteltyjä muuttujia. Where-lause koostuu generaattoreista ja konditionaaleista. Generaattorit sisältävät sekä varsinaisen generaattori-ilmauksen "← *Tietokannan nimi*" että sitä edeltävän *täsmääjälausekkeen* (match pattern), jossa voidaan sitoa muuttujia select-lausetta varten [BDHS96]. Generaattoreiden lisäksi where-lause voi sisältää konditionaaleja, esimerkiksi sidottujen muuttujien tyyppejä tarkistavia operaatioita. UnQL:ssä voi käyttää *isempty*-funktiota tarkistamaan, onko muuttujaan sidottu kaarta tai *isstring()*-funktiota tarkistamaan, onko arvona merkkijono vai jokin muu. Tästä on hyötyä heterogeenisen rakenteen kyselemisessä, sillä tietokannan symboliarvoiset rakennekaaret voidaan näin sulkea pois tuloksesta.

Yksinkertainen polkulauseke on hyödyllinen kyselyjen tekemistä helpottava työkalu, mutta heterogeenisen tiedon kyseleminen ei ole niiden avulla mielekästä. Ravintolatietokannan kaksi eri tapaa ilmaista osoitteet tekee osoitteen mukaan hakemisen mahdottomaksi yksinkertaisilla polkulausekkeilla. Tähän ratkaisuna ovat ilmaisuvoimaisemmat yleiset polkulausekkeet.

3.3 Yleinen polkulauseke

Yleinen polkulauseke [AQM+97] eroaa yksinkertaisesta polkulausekkeesta siten, että yleisellä polkulausekkeella voidaan edetä rakenteessa ennalta määräämättömälle syvyydelle polkulausekkeessa määritellyin ehdoin ja se sisältää tapoja ehdollistaa tai toistaa polkulausekkeen askelia. Käytännössä käytössä ovat säännöllisistä lausekkeistakin tutut toisto-operaattorit: tai (|), ? (ehdollisuus), + (yksi tai useampi) ja * (Kleenen tähti). Yleisten polkulausekkeiden periaatteena on tarjota tapa löytää tietokannasta olioita, jotka liittyvät toisiinsa osaksi ennakoiduin tavoin, mutta eivät kuitenkaan aina samalla tavalla.

Lorel-kielessä säännöllisiin lausekkeisiin kuuluvat edellämainittujen lisäksi osaksi polkua evaluoitavat muuttujat, joita merkitään syntaktisesti *unquote()*-funktion avul-

la. Löytääksemme kaikki Mountain Viewin ravintolat millä tahansa syvyydellä puussa suorittaisimme Lorel-kielellä kyselyn

select R **from** *Guide.#.restaurant* R **where** $R.city =$ "Mountain View",

missä "#" on polkuoperaattori, joka täsmää mihin tahansa mielivaltaiseen polkuun. Itse asiassa se on vain syntaktinen apuneuvo, joka tuottaa ilmauksen "(.%)*". Kyselyssä ravintolaoliot tunnistetaan nimenomaan *restaurant*-kaaren avulla, joka siis toimii ainoana olion tyyppimerkintänä. UnQL-kielellä vastaava kysely suoritettaisiin

select $\{Restaurant \Rightarrow r\}$
  where $Restaurant \Rightarrow \backslash r \twoheadleftarrow$ Guide,
    $r \Rightarrow \{City \Rightarrow$ "Mountain View"$\}$,

missä "$Restaurant \Rightarrow \backslash r \twoheadleftarrow Guide$" tarkoittaa samaa kuin
"$\_^* \Rightarrow \{Restaurant \Rightarrow \backslash r\} \leftarrow Guide$". UnQL-kielen konditionaalit ovat käytännössä ehtolauseita joiden toteutuessa kaaren kohdalla se voidaan sisällyttää tulosjoukkoon.

Edellinen "$p \Rightarrow \backslash t \twoheadleftarrow G$"-syntaksi on siis lyhenne yleiselle ilmaukselle ulottaa haku koko puun G alipuulle ja sitoa kaikki symbolista p johtavat kaaret muuttujaan t. Käytännössä haku yrittää täsmätä *Restaurant*-kaarta jokaiseen tietokannan kaareen. Muuttuja r esitellään ensin ensimmäisessä generaattorilauseessa ja sen jälkeen sitä käytetään sekä konditionaalissa että select-lauseessa.

Löytääksemme kaikki ravintolat postinumerossa 92310 tulisi meidän tehdä kysely, joka ottaa huomioon osoitetietojen erilaiset rakenteet, sekä jo edellä huomioidun ravintolaolion mielivaltaisen sijoittumisen tietokannassa. Lorel-kielellä kysely tehtäisiin seuraavasti:

select R **from** *Guide.#.restaurant* R **where** $R(.address)?.zip = 92310$.

Kuten aikaisemmassa kyselyssä, # tarkoittaa mitä tahansa mielivaltaisen syvää polkua. Rakenne "(.address)?" sen sijaan tarkoittaa valinnaista *address*-kaarta, mistä syystä myös ravintolatietokannan (kuva 1) ravintola Chef Chu's täsmää polkulausekkeeseen.

UnQL:ssä kysely on varsin samankaltainen, mutta edellisestä esimerkistä poiketen ei käytetä konditionaalilauseketta, vaan muuttujasta luotua generaattoria:

select $\{Restaurant \Rightarrow r\}$
  where $Restaurant \Rightarrow \backslash r \twoheadleftarrow Guide$,
    $[\hat{}\,Address]? \Rightarrow \{Zip \Rightarrow 92310\} \leftarrow r$.

Yleisten polkulausekkeiden käyttö sisältää myös tiettyjä riskejä, sillä ravintolatie-

tokannan *nearby*-kaaret on otettava huomioon. Jos kysely etenee sokeasti, ei haun tuloksena välttämättä ole sama olio kuin mitä alunperin haettiin. Edellinen haku on hyvä esimerkki siitä, että jokerikaarilla haku saattaisi palauttaa *nearby*-kaarien kautta myös muitakin ravintoloita kuin niitä, joiden postinumero määriteltiin.

Syklisten rakenteiden kohdalla, saman solmun tullessa toisen kerran vastaan polulla eteneminen pysäytetään sekä Lorel- että UnQL-kielissä. Vaikka tietokannat ovatkin äärellisiä verkkoja, eivät verkosta löytyvät polut tietokannan syklien takia välttämättä ole äärellisiä. Ainoana ravintolatietokannan esimerkkinä tästä toimivat juurikin *nearby*-kaaret, joita seuraamalla voisi polkukysely jatkua loputtomiin, mikäli sitä eivät järjestelmät katkaisisi.

4 Vertailuoperaatiot ja tyyppipäättely

Puolirakenteisen tiedon luonteeseen kuuluvat epäloogisesti ja vaihtelevin tietotyypein kuvatut arvot. Käytännön esimerkkejä tietokantajärjestelmälle ongelmallisista vertailuoperaatioista ovat esimerkiksi desimaalilukuna tallennettujen postinumeroiden vertaaminen numeerisesti määriteltyyn kyselyyn tai merkkijonona tallennetun iän vertaaminen kokonaislukuun. Toisaalta ongelmallisia voivat olla eri oliotyyppien, eli käytännössä yhdistelmäolioiden ja atomisten olioiden mielekäs vertaaminen toisiinsa. Sen lisäksi on varmistettava, että atomisen olion vertaaminen varsinaiseen arvoon onnistuu, eli käytännössä automaattisesti hakea olion arvo, eikä verrata sen identiteettiä vertailtavaan arvoon.

Lorel-kielessä on kaksi eri vertailuoperaattoria: = ja ==. Yksinkertaistaen voidaan sanoa, että = vertailee olioidentiteettejä ja == vertailee arvoja. Kielen määritelmä tuntee neljä erillistä oliotyyppiä, joiden avulla eri vertailuoperaatiot määritellään: arvot, atomiset oliot, oliojoukot ja yhdistelmäoliot [AQM+97]. Yleisesti ottaen Lorelin arvovertailu toimii kuten SQL-kielen yhtäsuuruusvertailu. Lorelin tyyppipakotusmekaniikka kuitenkin mahdollistaa erityyppisten arvojen vertailun mielekkäästi toisiinsa.

Lorel-kielen =-vertailuoperaattoria tarvitaan vain, jos halutaan vertailla kahden olion identiteettiä toisiinsa. Esimerkiksi jos halutaan tietää, viittaavatko kahden ravintolan *price*-kaaret samaan hintaa kuvaavaan olioon, vai eivät. Kyseinen käyttötapaus voi olla tietokannan ylläpitäjälle olennainen, kun taas varsinaisessa kyselykäytössä se on turha. UnQL:n kaarilla ei ole olioidentiteettiä, eli yhdellä yhtäsuuruusmerkillä suoritettu yhtäsuuruusvertailu on aina arvojen vertailu, eikä erillistä

identiteettivertailua olioidentiteettien puuttuessa ole [BDHS96].

Puolirakenteisen tiedon kontekstissa mielenkiintoisimpia vertailuoperaatioita ovat yksittäisten olioiden tai arvojen vertailu joukkoihin (taulukko 1). Nämä mahdollistavat moniarvoisten attribuuttien kyselemisen. Yksittäisen olion tai sen arvon ja joukon yhtäsuuruuskysely tapahtuu muuttamalla kysely vastaamaan kysymykseen: onko olio joukossa? Muuten vertailuoperaatiot pyrkivät aina tyyppipakotuksen ja olioiden arvojen selvittämisen avuilla vertailemaan olioiden arvoja toisiinsa.

	arvo	atominen olio	oliojoukko	yhdistelmäolio
arvo	tyyppi-pakotus	hae arvo	onko olemassa joukossa, =	epätosi
atominen olio		olioidentiteetti-vertailu	onko olemassa joukossa, =	epätosi
oliojoukko			joukkojen yhtäsuuruus	epätosi
yhdistelmä-olio				oliovertailu, =

Taulukko 1: Oliotyyppien pakotussäännöt Lorel-kielessä [AQM+97].

Lore-järjestelmässä arvojen vertailua ei lopeteta kesken, jos ne eivät ole samaa tyyppiä, kuten perinteisesti ohjelmointikielissä, vaan ne yritetään ennalta määrättyjen sääntöjen mukaan muuttaa samaan muotoon. Ohjesääntönä onkin, että kyselyn ja vertailuoperaation tulisi tuottaa mielekäs tulos riippumatta tietokannan tyypityksestä [AQM+97]. Lorelin tyyppimuunnokset toimivat seuraavasti: jos verrattavina ovat merkkijono tai kokonaisluku ja reaaliluku, muunnetaan ensiksi mainittu reaaliluvuksi. Jos arvot ovat samaa tyyppiä, ei muunnoksia suoriteta, vaan verrataan arvoja toisiinsa. Jos vertailua ei voida suorittaa, palauttaa vertailuoperaatio kielteisen vastauksen, ei virhettä.

Näiden lisäksi Lore-järjestelmässä on mahdollista käyttää Unix-komentona tuttua *grep*-funktiota merkkijonoja vertailtaessa. Vastaavasti UnQL:ssä kaarien symbolinimiin voidaan käyttää yleistä säännöllisten lausekkeiden syntaksia. Esimerkiksi Lorelin grep(”Movie”)-kutsua vastaava saavutettaisiin UnQL:ssä säännöllistä lauseketta ”[.*Movie.*]” käyttämällä.

UnQL sisältää myös tavan tunnistaa tietotyyppejä toisistaan where-lauseen ehtoja sekä puiden mielivaltaisella syvyydellä suoritettavaa *uudelleenjärjestelyä* (restructuring) varten [BDHS96]. Uudelleenjärjestely on käytännössä heterogeenisten tulospuiden luomista uudestaan homogeeniseksi hakutulokseksi. Esimerkkejä tällaisista funktioista ovat jo aikaisemmin mainitut *isstring()* ja *isempty()* -funktiot.

5 Muuttujien esittely

Muuttujia käytetään puolirakenteisen tiedon kyselykielissä kuten muissakin kyselykielissä. Muuttujia käytettäessä on kuitenkin ymmärrettävä kyselyn sisäinen semantiikka. Select-lauseessa ei voida viitata muuttujiin, joita ei ole esitelty from- tai where-lauseissa. Tyypillisesti muuttujat sidotaan polkulausekkeiden palauttamiin arvoihin ja niitä käytetään sen jälkeen yhdessä tai useammassa, muussa tai samassa kyselylausekkeen lauseista. Muuttujat voivat myös toimia polkulausekkeen juurina [AQM+97]. Seuraava Lorel-kielinen kysely sitoo from-lauseessa ravintolan muuttujaan R ja sen jälkeen sekä from- että where-lauseessa käyttää sitä osana polkulauseketta:

select R, N **from** *Guide.restaurant* R, *R.name* N **where** *R.price* = "cheap".

Sama kysely, joka palauttaa tuloksenaan sekä halvat ravintolat, että niiden nimet voidaan Lorel-kielellä muotoilla myös {}-syntaksia käyttäen. Tässä syntaktisessa rakenteessa aaltosulkeiden sisällä oleva muuttujanimi sidotaan sitä polulla edeltävään olioon:

select R, N **from** *Guide.restaurant*$\{R\}$*.name* N **where** *R.price* = "cheap".

Aaltosuljesyntaksin avulla pystytään palauttamaan yksi polun osa ja tämä sopiikin useimpiin käyttötapauksiin. Pidempien polkujen selvittämiseen yleisiä polkulausekkeita käytettäessä aaltosuljesyntaksi on soveltumaton. Niiden selvittämistä varten Lorel-kieli sisältää @-syntaksirakenteen, jolla pystytään palauttamaan mielivaltaisen pitkiä, yleensä lähinnä #-operaattorilla käsiteltäviä polkuja:

select R, P **from** *Guide.#@P.restaurant*$\{R\}$*.address.zip* Z **where** $Z = 90210$.

Tämä kysely palauttaisi muuttujassa R ravintolat, joiden postinumero on 90210 ja muuttujassa P polut, joiden päästä ravintolat löytyvät.

UnQL-kielessä muuttuja asetetaan siihen, missä normaalisti olisi puulauseke tai sen osa ja muuttujaan sidotaan puulausekkeeseen täsmänneiden monikoiden joukko.

Seuraavassa esimerkissä muuttujaan r sidotaan niiden kaarien joukko, josta lähtee kaaret *Price* ja *City*, joista vastaavasti lähtevät kaaret ”Cheap” ja ”Mountain View”:

select r **where** $\{Restaurant \Rightarrow$
$\{Price \Rightarrow$ ”Cheap”, $City \Rightarrow$ ”Mountain View”$\} \Rightarrow \backslash r\} \leftarrow Guide.$

Muuttujia käytetään UnQL-kielessä myös vastauspuiden luomiseen. Seuraavassa esimerkissä luodaan monimutkaisempi vastauspuu, johon asetetaan jokaiseen tietokannasta löydettyyn ravintola-puuhun täsmänneet ja niiden arvoihin sidotut muuttujat n ja c:

select $\{Restaurant \Rightarrow \{Name \Rightarrow n, Category \Rightarrow c\}\}$
where $Restaurant \Rightarrow \{Name \Rightarrow \backslash n, Category \Rightarrow \backslash c\} \twoheadleftarrow Guide.$

6 Päivitykset

Lorel-kielessä olioiden arvojen päivittäminen tapahtuu *update-from-where-päivityslausekkeella* [AQM+97]. Päivityslausekkeen osina voidaan käyttää polkulausekkeita, kuten select-from-where-lausekkeessakin. Käytännössä päivityslauseke muodostuu esimerkiksi seuraavasti:

update $R.address.zip := 92120$
from $Guide.restaurant\{R\}.name\ N$
where $N =$ ”Chef Chu’s”.

Päivityslausekkeen rakenne muistuttaa monilta osiltaan select-from-where-lauseketta. Näiden ainoa ero onkin niiden ensimmäisissä osissa. Siinä missä select-from-where-lausekkeen select-lauseessa määritellään kyselyssä palautettavat muuttujat, update-from-where-lausekkeen update-lauseessa päivitetään vastaavaa muuttujaa.

Edellä esitetty yksinkertainen esimerkki päivittää atomisen postinumeroa kuvaavan arvo-olion arvon toiseen. Atomista numeerista arvoa voi muuttamisen lisäksi inkrementoida tai dekrementoida. Yhdistelmäolioiden päivitysoperaatiot ovat vähän atomisia olioita vastaavia: attribuutteja voidaan lisätä ja poistaa ja niille voidaan lisätä arvoja, sekä poistaa niitä. Näin kyetään muuttamaan moniarvoisiakin attribuutteja. Kaupunkiattribuutin lisääminen ravintolalle tapahtuisi seuraavasti:

update $R.address.city \mathrel{+}=$ ”Menlo Park”
from $Guide.restaurant\{R\}.name\ N$

where N = "Chef Chu's",

missä ensin haetaan ravintolaolio muuttujaan R minkä jälkeen edetään polkulausekkeella ravintolan osoitekaarta. Sen jälkeen luodaan uusi atominen arvo-olio, johon tallennetaan merkkijono "Menlo Park", minkä jälkeen lisätään kaupunkikaari edellä selvitettyyn osoite-olioon.

Atomisten arvojen inkrementointi on erikoistapaus päivityslausekkeen syntaksissa. Atominen olio on ensin sidottava tietokantanimeen ja sen jälkeen tapausta voidaan käsitellä kuin attribuutin lisäämisenä. Esimerkiksi tietokantanimeen *Established* sidottua vuosilukua voitaisiin toteuttaa seuraavasti:

update *Established* += 1.

Jos sen sijaan käyttäisimme syntaksia

update *R.established* += 1
from *Guide.restaurant R, R.name N*
where N = "Chef Chu's",

lisäisi lauseke tietokantaan uuden arvon ja tekisi toisen *established*-kaaren yhdistelmäoliosta arvo-olioon.

UnQL-kieli ei sisällä vastaavia tapoja päivittää tietokannan sisältöä, mutta se sisältää *traverse*-lausekkeen, jolla on mahdollista tehdä monimutkaisia uudelleenjärjestelyjä myös syklisiin verkkoihin. *Traverse* on kuitenkin myös kyselylause ja se perustuu toteutukseltaan tilakoneista tuttuihin ϵ-kaariin. *Traverse* on erityisen ilmaisuvoimainen siksi, että sillä voi ehdollistaa paikalliset muutokset erilaisten paikallisten verkon läpikäyntiehtojen mukaan [BDHS96].

7 Yhteenveto

Puolirakenteinen tieto on rakenteeltaan heterogeenistä ja osin puutteellista. Sitä kuvataan yleisesti puina ja verkkoina, joiden kyselemiseen erityisesti polkukyselyt ovat käytännöllisiä. Tiedon puolirakenteinen ilmaiseminen tarkoittaa käytännössä skeeman puuttumista, mikä johtaa erilaisiin ongelmiin kyselyn muodostamisessa. Miten ilmaista deklaratiivisesti kysely, joka sisältää ehdollisia ja vaihtelevan syviä rakenteita? Miten vertailla eri muodoissa olevaa tietoa? Tietokannan entiteetin attribuuttien lukumäärä on yleensä määrätty. Miten ominaisuuden määräämättömyys muuttaa tiedon kyselemistä ja miten arvojen päivittäminen määritellään niin syntaktisesti

kuin semanttisestikin?

Lore-tutkimusprojektissa luotu Lorel-kyselykieli [AQM+97] ja Bunemanin & kumppaneiden kehittämä UnQL [BDHS96] ovat yrityksiä ratkaista puolirakenteisen tiedon kyselemiseen liittyviä ongelmia. Lorelin lähestymistapa on syntaktisesti helppo ymmärtää, jos tuntee ohjelmointikielissä yleisesti käytetyn tavan aksessoida olioattribuutteja pistenotaation avulla. Lorel-kielen pistenotaation etuna on se, että pistenotaatio toimii samalla tavalla niin kysely- kuin päivityslausekkeessakin. UnQL taas pohjautuu aaltosulkeilla ilmaistaviin monikoihin ja niistä muodostuviin puulausekkeisiin. Puulausekkeille ei löydy heti vastaavaa syntaktista vastinetta ohjelmointikielissä, joten ne eivät ole yhtä intuitiivisia ymmärtää. Niillä kuitenkin kyetään ilmaisemaan puurakenteita lyhyemmin, kuin Lorel-kielen vastaavilla polkulausekeyhdistelmillä.

Yksinkertaiset polkulausekkeet tarjoavat helpon ja yksinkertaisen tavan aksessoida syviä rakenteita, mikäli tiedon rakenne on tiedossa, vaikka itse rakenne olisikin monimutkainen. Yleiset polkulausekkeet mahdollistavat polkuelementtien toiston ja ehdollistamisen, eli tekevät mahdolliseksi heterogeenisten rakenteiden kyselemisen ja tietokannan rakenteen kartoittamisen. Erilaiset monimutkaisemmat vertailuoperaatiot ja olioattribuuttien arvojoukkojen älykäs käsittelu tekevät yksinkertaisten kyselyjen suorittamisesta mahdollista ilman selviä oletuksia tiedon tallennustavasta.

Puolirakenteista tietoa on tutkittu Lore-järjestelmän ja UnQL-kielen lisäksi Ludäscherin ja kumppaneiden toimesta muun muassa Florid-järjestelmässä [LHL+98]. Sittemmin polkukyselyitä on toteutettu XML-dokumenteissa ja myös Lorel-kielestä on versio, jolla pystytään käsittelemään XML:ää [GMW99].

Puolirakenteiset tietomallit mahdollistavat rakenteeltaan muuttuvan ja heterogeenisen tiedon mallintamisen. Puolirakenteisen tiedon kyselykielet kykenevät antamaan syvien jokerikyselyjen, erilaisten ehdollistamiskeinojen ja näiden avulla suoritettavien polkuvalintojen avulla järkeviä vastauksia kyselyihin, joiden vastaukset tuottaisivat vain virheitä tyypillisessä tietokannassa. Puolirakenteisen tiedon kyselykielet ratkaisevat selviä ongelmia. Ne myös tekevät heterogeenisen tiedon kanssa työskentelystä huomattavasti helpompaa kuin mitä se on rakenteisissa järjestelmissä ja näiden järjestelmien kyselykielissä.

Lähteet

Abi97 Abiteboul, S., Querying semi-structured data. *International Conference on Database Theory (ICDT)*, 1997, sivut 1–18.

AM99 Arocena, G. ja Mendelzon, A., WebOQL: Restructuring documents, databases, and webs. *Theory and Practice of Object Systems*, 5,3(1999), sivut 127—-141.

AQM+97 Abiteboul, S., Quass, D., McHugh, J., Widom, J. ja Wiener, J., The Lorel query language for semistructured data. *International Journal on Digital Libraries*, 1,1(1997), sivut 68–88.

BDHS96 Buneman, P., Davidson, S., Hillebrand, G. ja Suciu, D., A query language and optimization techniques for unstructured data. *ACM SIGMOD Record*, 25,2(1996), sivut 505—-516.

Cat95 Cattell, R. G. G., Object databases and standards. *Advances in Databases, 13th British National Conference on Databases, BNCOD 13, Manchester, United Kingdom, July 12-14, 1995, Proceedings*, 1995, sivut 1–11.

Dat89 Date, C. J., *A guide to the SQL standard (2nd ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

EN00 Elmasri, R. ja Navathe, S. *Concepts for Object-Oriented Databases*, luku 11. Addison-Wesley, kolmas painos, 2000.

FFLS97 Fernandez, M. F., Florescu, D., Levy, A. Y. ja Suciu, D., A query language for a web-site management system. *SIGMOD Record*, 26,3(1997), sivut 4–11.

GMW99 Goldman, R., McHugh, J. ja Widom, J., From semistructured data to XML: Migrating the Lore data model and query language. *WebDB (Informal Proceedings)*, 1999, sivut 25–30.

HMGM97 Hammer, J., McHugh, J. ja Garcia-Molina, H., Semistructured data: The TSIMMIS experience. *Proceedings of the First East-European Workshop on Advances in Databases and Information Systems*, 1997.

HP93 Hidders, J. ja Paredaens, J., Goal, a graph-based object and association language. *CISM - Advances in Database Systems*, 1993, sivut 247–265.

LHL+98 Ludäscher, B., Himmeröder, R., Lausen, G., May, W. ja Schlepphorst, C., Managing semistructured data with FLORID: A deductive object-oriented perspective. *Information Systems*, 23,8(1998), sivut 589—-613.

PGMW95 Papakonstantinou, Y., Garcia-Molina, H. ja Widom, J., Object exchange across heterogeneous information sources. *Proceedings of the 11th International Conference on Data Engineering (ICDE'95)*, (1995).

STM99 Stein, L. D. ja Thierry-Mieg, J., AceDB: A genome database management system. *Computing in Science & Engineering*, 1,3(1999), sivut 44–52.