Laajennettu relaatiomalli ERDM ja suoraviittauksinen kyselykieli NSQL

Mika Niemelä

Tampereen yliopisto
Tietojenkäsittelyopin laitos
Pro gradu tutkielma
7.4.2003

Tampereen yliopisto
Tietojenkäsittelyopin laitos
Niemelä, Mika Petri: Laajennettu relaatiomalli ERDM ja suoraviittauksinen kyselykieli NSQL
Pro gradu -tutkielma, 87 sivua

Toukokuu 1998

Tiivistelmä

Relaatiomallin ilmaisuvoimassa on joitakin pahoja puutteita, jonka vuoksi on esitetty uusia korvaavia tietokantamalleja. Uudet tietokantamallit eivät kuitenkaan yleensä pysty toteuttamaan kaikkia relaatiomallin kiistattomia hyviä ominaisuuksia. Tässä tutkimuksessa on esitetty relaatiomallin laajennus ERDM, joka poistaa relaatiomallin pahimmat puutteet tuomalla malliin kyvyn esittää rakenteellisia objekteja NF2-mallin keinoin ja mahdollistamalla transitiivisten suhteiden käsittelyn aggregoiden. Tutkimuksessa esitetty laajennettu relaatiomalli säilyttää lisäksi kaikki relaatiomallin hyvät ominaisuudet, kuten deklaratiiviset kyselykielet ja mallin vahvan matemaattisen perustan. Tutkimuksessa on esitetty uusien ominaisuuksien lisääminen relaatiomalliin ja relaatioalgebraan, sekä luotu uusi suoraviittauksinen deklaratiivinen kyselykieli NSQL, jolla laajennetun relaatiomallin käsittely tapahtuu suoraviivaisesti ja intuitiivisesti. Lopuksi on esitetty miten laajennetun relaatiomallin ja NSQL-kielen ominaisuuksia hyödyntäen voidaan malliin helposti lisätä IS-A suhteiden käsittely ja automatisoida osa liitosoperaatioista. Tässä tutkimuksessa esitelty laajennettu relaatiomalli on huomattavasti ilmaisuvoimaisempi kuin perinteinen relaatiomalli. Laajennettu relaatiomalli sisältää useita muun muassa oliotietokantamallissa hyviksi todettuja ominaisuuksia säilyttäen kuitenkin kaikki relaatiomallin kiistattomat hyvät puolet joihin muut tietokantamallit eivät pysty vastaamaan.

Avainsanat: tietokannat, relaatiomalli, NF2, transitiivisulkeuma, kyselykielet, SQL, NSQL, pro gradu –tutkielma.

Sisällysluettelo

Sanastoa

ERDM	Extended Relational Data Model
NSQL	Nested form Structured Query Language. Kehittämäni suoraviivainen strukturoitu kyselykieli NF2-relaatiomallille.
SQL	Structured query language.
SQL/NF	Structured query language/Nested form. SQL-kieli NF2-mallille. Pyrkii noudattamaan perinteisen SQL-kielen notaatiota.
SQL*	SQL-kieli, joka sisältää rekursion.

ASIASANAT: Tietokannat, Relaatiomalli, NF2, Transitiivisulkeuma, Oliotietokantamalli, SQL, OSQL, NSQL

1. JOHDANTO

Relaatiomalli on saavuttanut johtavan aseman tietokantaparadigmana, koska se on ylivoimainen vanhoihin tietokantaparadigmoihin verrattuna. Coddin kehittelemällä relaatiomallilla on kuitenkin useita puutteita, jotka korostuvat erityisesti uudentyyppisillä sovellusaloilla, kuten CAD ja multimedia tietokannoissa, joissa tulisi pystyä esittämään mutkikkaita rakenteellisia objekteja. Lisäksi relaatioalgebralla ja relaatiomalliin de facto standardiksi muodostuneella SQL-tietokannan käsittelykielellä on puutteita, jotka rajoittavat relaatioiden mahdollisia käyttötapoja.

Pahimpia puutteita ovat kyvyttömyys esittää rakenteellisia objekteja ja rajoittunut kyky johdetun tiedon esittämiseen. Näistä puutteista ovat uudet tietokantaparadigmat saaneet hyvän aseen taistelussa parhaan tietokantateknologian asemasta. Tärkein jo tulossa oleva uusi tietokantamalli on oliomalli, joka vastaakin useisiin relaatiomallin puutteisiin. Relaatiomallia on kuitenkin mahdollista laajentaa vastaamaan paremmin nykyajan tarpeita. Tällöin voidaan säilyttää relaatiomallin kiistattomat hyvät puolet ja hyödyntää jo kehitetyt ja käytännössä testatut toteutustekniikat ja optimointitekniikat. Laajennetun relaatiomallin käyttöönotto olemassaolevien relaatiotietokantojen seuraajaksi on myös huomattavasti helpompaa kuin kantojen muuntaminen jonkin kolmannen paradigman mukaiseksi. Laajennettuun relaatiomalliin siirryttäessä ei jouduta tallennetun tiedon radikaaliin uudelleenorganisointiin, mikä olisi välttämätöntä siirryttäessä lähtökohdiltaan erilaiseen tietokantaparadigmaan.

Esittelen tässä tutkimuksessa laajennetun relaatiomallin, jolla on mahdollista esittää rakenteellisia objekteja, ja jonka ilmaisuvoima on huomattavasti tavallista relaatiomallia suurempi. Lisäksi esittelen kehittämäni uuden kyselykielen NSQL (Nested form Structured Query Language), joka on suunniteltu hyödyntämään tehokkaasti ja suoraviivaisesti laajennetun relaatiomallin ominaisuuksia. NSQL-kieli on huomattavasti intuitiivisempi ja helppokäyttöisempi kuin perinteistä SQL:ää mukailevat NF2-mallille esitetyt kyselykielet. Ilmaisuvoimaltaan NSQL-kieli on yhtä vahva kuin SQL lisättynä transitiivisella käsittelyllä. Lopuksi esittelen joitakin lisäominaisuuksia, joita laajennetulla relaatiomallilla ja luomallani kyselykielellä voidaan toteuttaa. Laajennettuun relaatiomalliin voidaan helposti lisätä IS-A suhteiden käsittely, eli oliomallin termistöllä luokkahierarkioiden esittämiskyky. Toinen hyvä lisäominaisuus on liitosoperaation automatisoiminen, joka yksinkertaistaa kyselyiden tekemistä ja tuo useita oliomallin suorien olioviittausten etuja relaatiomalliin. Liitosoperaatio automatisoimalla mahdollistuu jopa koko tietokannan esittämisen loppukäyttäjälle yhtenä NF2-relaationa.

1.1. MIKÄ ON LAAJENNETTU RELAATIOMALLI

Relaatiomallia voi laajentaa monella tavalla, joten esitän mitä tässä tutkimuksessa laajennetulla relaatiomallilla tarkoitetaan. Tässä tutkimuksessa relaatiomallia laajennetaan kahdella eri tavalla. Ensiksi laajennetaan mallin kykyä ekstensionaalisen tiedon, eli tietokantaan fyysisesti tallennetun tiedon esittämiseen mahdollistamalla rakenteellisten objektien esittäminen kannassa niiden rakenne säilyttäen. Toiseksi laa-

jennetaan mallin kykyä johdetun tiedon esittämiseen lisäämällä transitiivisulkeuman laskentakyky.

Rakenteellisten objektien esittäminen mahdollistetaan laajentamalla relaatiomalli NF^2-relaatiomalliksi, jossa relaation attribuuttien atomisuudesta on luovuttu. Tällöin relaation attribuutti voi sisältää arvonaan relaation, joka puolestaan voi sisältää relaation ja niin edelleen. Tämä laajennos mahdollistaa objektin osien esittämisen objektin yhteydessä, tai sen 'sisällä', jolloin tietokannan merkityksen ymmärtäminen on helpompaa ja yleisimmät tietokantahaut yksinkertaistuvat ja tehostuvat. Transitiivisulkeuman läpikäynnin mahdollistaminen taas lisää olennaisesti relaatiomallin kykyä esittää johdettua tietoa, eli tietoa, jota ei ole tietokantaan tallennettu, mutta joka voidaan johtaa tietokantaan fyysisesti tallennetusta tiedosta. Transitiivisulkeuman laskenta lisätään malliin siten, että samalla voidaan kerätä aggregointitietoa. Transitiivisulkeuman avulla voidaan selvittää esimerkiksi ketkä kaikki ovat tietyn henkilön jälkeläisiä, tai mitä eri reittejä voi lentää kahden paikkakunnan välillä, tai mistä osista ja osan osista tuote koostuu. Tällaisiin tietotarpeisiin ei perinteinen relaatiomalli pysty suoraan vastaamaan. Aggregointiominaisuuksien sisällyttäminen transitiivisulkeuman laskentaan tarkoittaa käytännössä, että samalla voidaan selvittää esimerkiksi lentoreitin kokonaispituus tai kertoa, monennessako polvessa jälkeläinen on. Tutkimuksessa esitellyssä laajennetussa relaatiomallissa siis laajennetaan tietorakenteiden esitystapaa ja relaatioalgebraa, sekä luodaan uusi NSQL-kieli, joka pystyy hyödyntämään uudet ominaisuudet tehokkaasti. Näillä laajennoksilla relaatiomalliin saadaan olennaisesti lisää ilmaisuvoimaa ja intuitiivisempi tiedon rakenteen esitystapa. Näin malliin saadaan useita ominaisuuksia, joiden puutteesta sitä on arvosteltu, ja joita hyväksi käyttäen esimerkiksi oliomallia pyritään markkinoimaan. Relaatiomallia näin laajentamalla säilytetään kuitenkin kaikki relaatiomallin vahvuudet, kuten vankka matemaattinen perusta, hyvä tietoriippumattomuus, helppokäyttöinen deklaratiivinen kyselykieli ja tehokkaat optimointitekniikat.

Olennaisin säilytettävä ominaisuus relaatiomallia laajennettaessa on arvoperustaisuus. Relaation riveillä ei ole järjestystä, eikä niihin voi viitata millään suoralla osoittimella vaan relaation rivi on identifioitava avaimen arvojen perusteella. Olio-, verkko- ja hierarkisen-tietokantamallin suoria tietue- tai olio-osoittimia ei relaatiotietokantaan kannata tuoda, koska juuri arvoperustaisuuden ansiota ovat useimmat relaatiomallin vahvuudet, joihin oliomallikaan ei pysty vastaamaan. Arvoperustaisuus on edellytyksenä muun muassa deklaratiivisen kyselykielen luomiselle. Toinen oliomallin ominaisuus, jota ei kannata relaatiomalliin lisätä, on tiedon kapselointi, joka myös estäisi deklaratiivisen kyselykielen luomisen. Muita oliomallin vahvoja puolia on mahdollista lisätä relaatiomalliin varsin kattavasti.

Muita relaatiomallin tärkeitä laajentamiskohteita olisivat aggregointiominaisuuksien lisääminen suoraan relaatioalgebraan, arvojoukko-käsitteen laajentaminen, joka relaatiomallissa on alunperin mukana ollut, mutta jota kaupallisissa sovelluksissa ei ole kunnolla toteutettu, sekä tyhjäarvojen käsittelyn monipuolistaminen. Lisäksi jotkin sovellusalueet vaatisivat relaatiomallin laajentamista todennäköisyysarvoilla ja versiointiominaisuuksilla. Näihin ominaisuuksiin ei kuitenkaan tässä tutkimuksessa puututa.

2. CODDIN RELAATIOMALLI

Coddin kehittämä relaatiomalli [Codd70] on osoittautunut ylivoimaiseksi yleiskäyttöiseksi tietokantaparadigmaksi edeltäjiinsä, esimerkiksi verkkomalliin ja hierarkiseen tietokantamalliin verrattuna. Tärkeimmät syyt tähän ylivoimaisuuteen ovat: relaation visualisoinnin intuitiivisuus taulukkona, mallin yksinkertaisuus ja helppo mallinnettavuus, vankka matemaattinen perusta, deklaratiiviset strukturoidut kyselykielet kuten SQL ja sen luoma standardoitu rajapinta, tehokkaat optimointitekniikat, sekä vahva tietoriippumattomuus.

2.1. RELAATIOMALLIN PERUSTA

2.1.1. Matemaattinen perusta

Relaatiomallin matemaattinen perusta on joukko-opin relaatioteoriassa. Relaatio on karteesisen tulon osajoukko, joka määritellään arvojoukkojen välillä. Oletetaan, että meillä on arvojoukot (domains) D1, D2,...,Dn. Tällöin relaatio R on osajoukko karteesisesta tulosta $D1 \times D2 \times ... \times Dn$, joka on joukko kaikista n-monikoista (n-tuple) (a1,a2,...an) siten, että a1 saa arvonsa arvojoukosta D1, a2 arvojoukosta D2 ja niin edelleen. Relaatio R on siis eräs joukon $P(D1 \times D2 \times ... \times Dn)$ alkio, jossa $P(D1 \times D2 \times ... \times Dn)$ (powerset) on joukko kaikista karteesisen tulon $D1 \times D2 \times ... \times Dn$ mahdollisista osajoukoista. Arvojoukkojen ei tarvitse olla erillisiä. Esimerkin relaatiossa on n attribuuttia, joten relaation R sanotaan olevan astelukua n. Arvojoukko on joko joukko explisiittisesti annettuja arvoja esimerkiksi D1 = {1,2,3} tai yleisesti tunnettu joukko kuten reaalilukujen joukko $D2 = \Re$.

Oletetaan, että meillä on arvojoukot D3 = {Ville, Minna} ja D4 = {15,21,29}. Karteesinen tulo arvojoukkojen D3 ja D4 välillä merkitään $D3 \times D4$ = {(Ville,15), (Ville,21), (Ville,29), (Minna,15), (Minna,21), (Minna,29)}. Relaatio on mikä hyvänsä osajoukko karteesisesta tulosta, joten eräs edellämainittujen arvojoukkojen välinen relaatio on esimerkiksi {(Ville,15),(Minna,21)}, koska se kuuluu joukkoon $P(D3 \times D4)$. Relaatio voi olla myös ääretön, kuten relaatio luonnollisten lukujen ja reaalilukujen joukkojen välillä $R \in P(N \times \Re)$. Arkikielessä relaatiolla ymmärretään kahden joukon tai alkion olemista suhteessa keskenään, mutta matematiikassa relaation voi muodostaa myös yksi arvojoukko $R \in P(D1)$.

Relaatio voidaan visualisoida tauluna, jolloin karteesinen tulo $D3 \times D3 \times D4$ (myös eräs relaatio) voidaan esittää kuten kuvassa kuvio 2.a.

D3	D3	D4
Ville	Ville	15
Ville	Ville	21
VIlle	Ville	29
Ville	Minna	15
Ville	Minna	21
Ville	Minna	29
Minna	Ville	15
Minna	Ville	21
Minna	Ville	29
Minna	Minna	15
Minna	Minna	21
Minna	Minna	29

Kuvio 2.A *Karteesinen tulo D3×D3×D4 tauluna esitettynä.*

Tauluesityksessä on ilmeistä, että jokainen taulun sarake (attribuutti) saa arvonsa tietystä arvojoukosta. Useampi sarake voi saada arvonsa samasta arvojoukosta, joten ainoa tapa viitata tiettyyn sarakkeeseen on sarakkeen järjestysnumero. Joukko-opissa relaation sarakkeilla on siis aina tietty järjestys. On huomattava, että relaation riveillä (tuple) ei ole minkäänlaista järjestystä, vaan ne muodostavat järjestämättömän joukon.

2.1.2. Relaatiomallin suhde matemaattisiin relaatioihin

Relaatiomalli tietokantamallina poikkeaa hieman relaation matemaattisesta esityksestä. Relaatiomallissa relaation sarakkeille annetaan nimet, jolloin sarakkeisiin voidaan viitata nimen perusteella järjestysnumeron asemasta. Arvojoukon nimi ja relaation sarakkeen (attribuutin) nimi siis ovat kaksi eri asiaa. Relaatiomallissa arvojoukon nimi kuvaa yleensä attribuutin tyyppiä (nimet, osoitteet) ja attribuutin nimi saattaa kuvata lisäksi tyypin roolia (isännimi, työntekijänimi, varusmiesnimi saavat kaikki arvonsa arvojoukosta nimet). Useammat attribuutit saattavat saada arvonsa samasta arvojoukosta, mutta attribuuttien nimien (roolien) on oltava erilaiset, kuten esimerkkirelaatiossa kuvio 2.b, joka on osajoukko kuvan kuvio 2.a esittämästä karteesisesta tulosta. Käyttäjän on helpompi muistaa sarakkeen sisältöä kuvaava nimi, esimerkiksi "ikä", kuin sen järjestysnumero relaatiossa. Samalla voidaan loppukäyttäjän ja sovellusohjelman näkökulmasta luopua sarakkeiden järjestyksen merkityksestä ja saavutetaan korkeampi tietoriippumattomuuden taso. Järjestelmän täytyy tietenkin sisäisesti pitää huolta järjestyksestä. Myös relaatioille annetaan nimet, jotta niihin voidaan viitata. Periaatteessa relaation nimi viittaa objektiin tai suhteeseen, jonka attribuutit relaatioon on talletettu, mutta normalisoinnin myötä objekti saattaa hajota useampiin relaatioihin, joille ei ole helppoa keksiä kuvaavia nimiä.

Aviopuolisot

Mies	Vaimo	LiitonIkä
Ville	Minna	15

Kuvio 2.B *Relaatiotaulu tietokannassa.*

Relaation rivit ovat erillisiä, eli samat arvot sisältävä rivi voi esiintyä relaatiotaulussa vain kerran. Tämä tuntuu joukko-opillisessa esityksessä luonnolliselta, mutta tauluesityksessä asian merkitys helposti hämärtyy. Relaatiomallin arvoperustaisuudesta johtuen ei tästä vaatimuksesta kuitenkaan voi luopua, sillä arvoperustaisessa mallissa ainoa tapa erottaa rivit toisistaan ovat rivin attribuuttien arvot. Samat arvot sisältäviä rivejä ei siis pysty mitenkään yksilöimään, eikä niitä siten kannata tallentaa tietokan-

taan. Informaatio perustuu arvojen yhdistelmien tulkintaan eikä moninkertainen tallennus tuo uutta informaatiota.

Joukko-opillisissa relaatioissa ei attribuuttien arvoille ole asetettu mitään rajoituksia, mutta relaatiomallissa attribuuttien arvojen on oltava atomisia. Tämän ehdon täyttävän relaation sanotaan olevan ensimmäisessä normaalimuodossa (1NF) (first normal form). Tällöin attribuutin arvo yhdellä rivillä ei voi olla esimerkiksi joukko tai relaatio vaan ainoastaan atominen arvo. Itse asiassa Codd ei alkuperäisessä relaatiomallin määrittelyssään rajannut attribuuttien arvoja atomisiksi, mutta tämä rajoitus omaksuttiin myöhemmin mukaan relaatiomallin määritelmään. Ensimmäisen normaalimuodon vaatimus tekee mallin hyvin yksinkertaiseksi, ja alaa tuntemattomatkin henkilöt ymmärtävät relaation helposti kaksiulotteisena taulukkona, mikä onkin varsin oikea käsitys asiasta, ja riittää useimpiin käyttötarpeisiin. Tällaiselle yksinkertaiselle mallille on lisäksi ollut helppo kehittää tehokkaita optimointitekniikoita ja implementaatioita. Huono puoli on se, että tieto pirstaloituu useisiin relaatioihin, joita sitten joudutaan liittämään toisiinsa kyselyitä tehtäessä, tämä vaikeuttaa tietokannan merkityksen ymmärtämistä. Samasta syystä ei relaatiotietokannassa voi esittää mutkikkaita rakenteellisia objekteja niiden rakenne säilyttäen, vaan ne pirstaloituvat eri relaatioihin, jolloin niiden rakennetta ja merkitystä on vaikea havaita ja niiden kokoaminen voi olla raskas operaatio.

Relaatiomallin matemaattisen täsmällisestä määrittelystä ja yksinkertaisuudesta johtuen on malliin voitu määrittää tiedonkäsittelykieli relaatioalgebra, jossa on hyvin rajoittunut määrä operaattoreita ja jotka toimivat kaikille relaatioille samalla tavalla, olivatpa relaatiot sitten eksplisiittisiä tai implisiittisiä. Tämä operaatiojoukko on helppo hallita ja sillä saadaan aikaan erittäin ilmaisuvoimainen ja deklaratiivinen kyselykieli.

2.1.3. Relaatiotietokannan rakenteesta

Kyselyn tulokset on tietokantaparadigmasta riippumatta luontevaa esittää tauluna tai relaationa ja tässä relaatiomalli saa ratkaisevan edun muihin tietokantamalleihin verrattuna. Koska relaatiomallissa relaatio on myös tallennetun tiedon luonteva esitysmuoto, voidaan kyselyn tuloksia käyttää suoraan operandeina samoille operaatioille, joilla tietoa käsitellään myös tietokantaan tallennetuissa relaatioissa. Kyselyitä voidaan siten helposti ketjuttaa käyttämällä edellisen kyselyn tulosrelaatiota syötteenä seuraavalle kyselylle. Tilanne ei suinkaan ole tällainen muiden tietokantamallien kohdalla. Oliomalleissa, jolla tarkoitan nyt laajasti kaikkia olioidentiteetiin perustuvia tietokantamalleja (verkko-, hierarkinen- ja varsinainen oliomalli), ei vastaava kyselyiden ketjuttaminen onnistu yhtä helposti, koska tietokantaan tallennetut objektit ovat olioita ja kyselyn tulos esitetään yleensä relaationa, joka ei tue olioidentiteettiä, eikä kyselyn tulosta voi siten käsitellä samalla operaatiojoukolla kuin tietokantaan tallennettuja olioita.

Relaatiomallissa on vain kaksi rakenteellista (compound) tietotyyppiä, järjestetty n-jono ja järjestämätön joukko n-jonoja. Rakenteellinen tietotyyppi on tietotyyppi, joka koostuu määrätystä yhdistelmästä atomisia tai rakenteellisia tietotyyppejä. Tietokannassa tarvitaan tiedon käsittelyyn ainakin kolme eri komentoa: haku, tallennus ja poisto. Yleensä määritellään vielä erikseen hyödyllinen operaatio päivitystä varten, jolloin operaatioita on neljä kappaletta. Nämä operaatiot tarvitaan erikseen jokaista

rakenteellista tietotyyppiä kohden, joten mallissa, jossa on N rakenteellista tietotyyppejä, tarvitaan 4N operaatiota eikä niiden hallitseminen ole käyttäjälle helppoa kun N:n arvoa aletaan kasvattaa, ellei operaatioita saada piilotettua käyttäjältä. Relaatiomallissa, jossa operaatiot tarvitsee määritellä ainoastaan joukolle, riittää neljä muistettavaa komentoa. Abstrakteja tietotyyppejä tukevissa malleissa operaatiojoukko täytyy määritellä erikseen jokaista luotavaa tietotyyppiä varten. Operaatioiden suuren määrän lisäksi ongelmaksi muodostuu silloin se, että vaikka kyselyn tuloksena saataisiinkin olioita, ovat nämä yleensä jotain uutta tyyppiä, eikä niitä voi käyttää suoraan operandeina toisille operaatioille ennen kuin uusille olioille on määritelty operaatiot. Kyselyn tuloksen käyttäminen seuraavan kyselyn syötteenä ei siis ole helposti mahdollista malleissa, jotka tukevat abstrakteja tietotyyppejä.

Redundanssin vähentämiseksi ja epätäsmällisten määrittelyjen poistamiseksi relaatiotietokannasta on ensimmäisen normaalimuodon päälle kehitetty korkeampia normaalimuotoja, joiden ymmärtämistä tarvitaan lähinnä tietokannan rakenteen suunnittelussa. Näistä saatavien kiistattomien hyötyjen vastapainona on tietojen pirstaloituminen yhä useampiin relaatioihin. Käytännössä tärkein normaalimuoto 1NF- määrittelyn jälkeen on Boyce-Codd normaalimuoto. Tässä tutkimuksessa merkinnällä 1NF-relaatio tarkoitetaan tästä eteenpäin relaatiota, joka on vähintään ensimmäisessä normaalimuodossa, eli se voi olla myös esimerkiksi toisessa-, kolmannessa- tai Boyce-Codd normaalimuodossa.

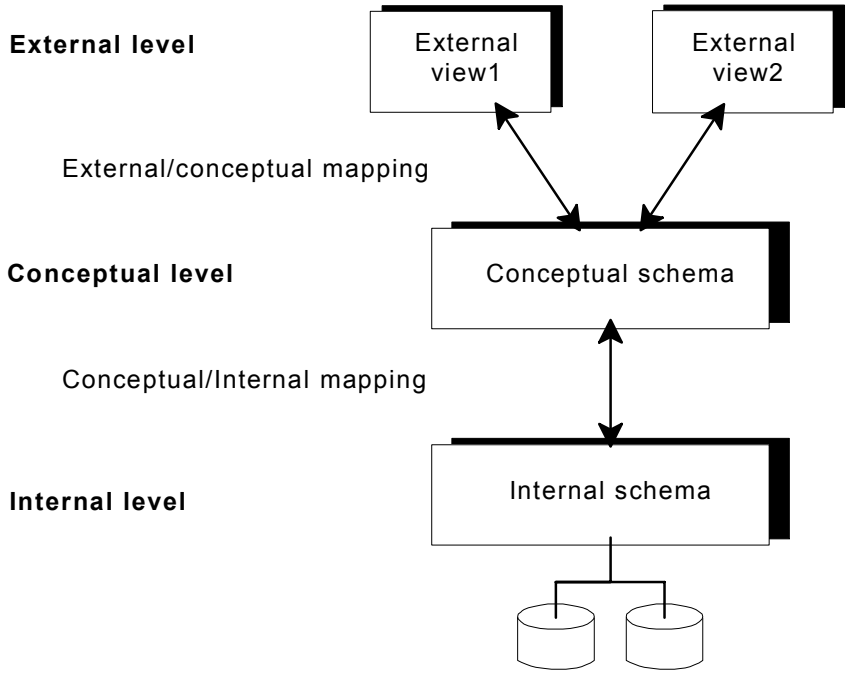
2.1.4. Tietoriippumattomuus

Tietoriippumattomuus edellyttää, että tietokannan fyysistä rakennetta ja käsitteellistä rakennetta tulisi voida muuttaa ilman että muutos näkyy mitenkään loppukäyttäjälle tai sovellusohjelmalle. Tietoa voidaan kuvata hyvin monella tavalla ja tietokoneen tapa esittää tieto bitteinä fyysisessä muistissa on varsin kaukana loppukäyttäjän tavasta käsitellä käsitteitä. Välissä tarvitaan siten useita eri abstraktiotasoja. Tiedonhallintajärjestelmien yhteydessä yleinen tapa on jakaa tietokannan kuvaus kolmelle eri abstraktiotasolle esimerkiksi [ElNa94] ja [Ull88], jotka on kuvattu kuvassa_____ kuvio 2.c.

- *Sisäinen kaavion tasolla* (Internal schema) kuvataan tiedon fyysinen tallennusrakenne ja tiedon saantipolut fyysiseltä tallennusvälineeltä.

- *Käsitekaaviotasolla* (conceptual schema) kuvataan koko tietokannan sisältö piilottaen fyysinen tallennusrakenne. Käsitekaavion tasolla kuvataan tietokannan objektit, tietotyypit, suhteet, operaatiot ja ehdot. Relaatiotietokannan ollessa kyseessä käsitellään käsitekaavio tasolla tauluja, attribuutteja, arvojoukkoja, riippuvuuksia ja ehtoja.

- *Ulkoisella tasolla* tai *näkymätasolla* (External view) kuvataan jonkin käyttäjän tai käyttäjäryhmän tarvitsema osa tietokannasta. Tietokannan rakenne voidaan kuvata myös erinäköisenä kuin se on todellisuudessa käsitekaaviotasolla. Näkymätasolla on mahdollista kuvata myös tietokannan tiedoista implisiittisesti tuotettua tietoa, jota tietokantaan ei fyysisesti ole tallennettu. Relaatiotietokannoissa implisiittistä tietoa voidaan esittää tekemällä kysely, jonka tulosrelaatio näytetään loppukäyttäjälle näkymänä (view). Samalla tekniikalla voidaan muuttaa tietokannan rakennetta loppukäyttäjän näkökulmasta tarkasteltuna.



Kuvio 2.C *Kolmitasoinen tietokantakaavio.*

Tietoriippumattomuudella tarkoitetaan sitä, että tietokantakaaviossa voidaan alemman tason kaaviota muuttaa ilman, että se vaikuttaa ylemmän tason kaavioihin. Kolmitasoisessa mallissa on kahta eri lajia tietoriippumattomuutta:

- *Fyysinen tietoriippumattomuus* tarkoittaa, että sisäistä tietokannan kaaviota (internal schema) voidaan muuttaa ilman että se näkyy käsitteellisellä tasolla (conceptual schema). Tämä tarkoittaa sitä, että tiedon fyysistä tallennusrakennetta voidaan muuttaa esimerkiksi jonosta b-puuksi tai, että relaation tietueiden fyysinen tallennusjärjestys voidaan muuttaa esimerkiksi sukunimen mukaisesta henkilötunnuksen mukaiseksi, ilman että sovellusohjelmia tarvitsee muuttaa tai loppukäyttäjälle näkyisi mitään muutoksia, lukuunottamatta ehkä muutosten aiheuttamaa prosessoinnin tehostumista. Muutokset fyysiseen tallennusrakenteeseen voivat olla tarpeellisia esimerkiksi tietokannan tietosisällön tai käyttötarpeiden muuttuessa ajanmyötä niin, että toinen tietorakenne tai järjestys osoittautuu tehokkaammaksi.

- *Looginen tietoriippumattomuus* tarkoittaa, että käsitteellisen tason kaaviota (conceptual schema) voidaan muuttaa ilman että se näkyy näkymätasolle (external view). Käsitteellisen tason kaaviota voidaan joutua muuttamaan, kun halutaan lisätä kantaan uudentyyppisiä objekteja (relaatioita) tai lisätä uusia ominaisuuksia olemassa oleviin objekteihin (attribuutteja), taikka poistaa objekteja tai ominaisuuksia. Tällaisten muutosten tekeminen ei saisi aiheuttaa muutostarpeita niihin sovellusohjelmiin, joiden tietotarpeissa ei tapahdu muutoksia. Luonnollisesti ob-

jektityyppien tai ominaisuuksien poistaminen vaikuttaa niihin sovellusohjelmiin, jotka käsittelevät poistettavia ominaisuuksia. Lievemmäksi loogisen tietoriippumattomuuden tasoksi voidaan ymmärtää se, että muutokset käsitteellisellä tasolla vaativat muutoksia vain näkymätason ja käsitteellisen tason välisessä konversiossa (external/conceptual mapping), mutta eivät sovellusohjelmissa.

Relaatiomallissa fyysinen tietoriippumattomuus on erittäin hyvä. Tiedon fyysistä tallennusrakennetta voidaan muuttaa täysin vapaasti ilman, että se näkyy mitenkään sovellusohjelmille tai loppukäyttäjille. Looginen tietoriippumattomuuskin on mahdollista saattaa varsin korkealle tasolle näkymiä käyttämällä, mutta näkymien käyttö ei relaatiomallissa ole pakollista, joten looginen tietoriippumattomuus jää helposti hieman heikommalle tasolle. Ilman näkymien käyttöäkin on relaatiomallin looginen tietoriippumattomuus varsin korkealla tasolla; attribuutteja ja relaatioita voi poistaa tai lisätä kantaan ilman, että se vaikuttaa yleisimpien operaatioiden toimintaan sovellusohjelmissa, jotka käsittelevät jäljelle jäävää tietoa. Valinta-, projektio-, karteesinen tulo- ja eri liitosoperaatiot toimivat oikein muutosten jälkeenkin. Sensijaan unioniyhteensopivuutta vaativat operaatiot eivät enää toimi, jos niiden operandirelaatioista toiseen on tehty muutoksia. Tällaisia operaatioita ovat unioni, erotus ja leikkaus. Tosin relaation rakennetta muutettaessa on yleensä muutettava samalla tavalla myös muutettavan relaation kanssa unioniyhteensopivien relaatioiden rakennetta, koska ne kuvaavat yleensä saman objektityypin eri ilmentymäryhmiä ja siten niiden kaikkien ominaisuusjoukko on sama. Näkymiä käyttämällä voidaan myös unioniyhteensopivuuden pysyminen taata käsitetasolla muutoksia tehtäessä. Näkymien avulla voidaan jopa selvitä tilanteista, joissa tarpeellista tietoa siirretään taulusta toiseen (objektista toiseen), tällöin vaaditaan muutoksia käsitetason ja näkymätason väliseen konversioon, mutta sovellusohjelmiin muutoksia ei tarvitse tehdä.

2.1.5. Arvoperustaisuus ja avaimet

Relaatiomalli on arvoperustainen tietokantamalli. Arvoperustaisuudella tarkoitetaan sitä, että objekteihin (relaatiomallissa relaation rivit) viitataan niiden arvon perusteella, eikä objekteilla ole olemassa mitään muuta niitä yksilöivää tunnistetta. Olioidentiteettiin perustuvissa malleissa kuten olio-, verkko- ja hierarkisessa tietokantamallissa on jokaisella oliolla oma yksikäsitteinen järjestelmän ylläpitämä oliotunnisteensa, jonka avulla voidaan tehdä suoria viittauksia olioihin. Olioidentiteettiä käyttävissä tietokantamalleissa ei pystytä luomaan deklaratiivista kyselykieltä [Ull88 s.21], eli sellaista kieltä jolla voidaan kertoa mitä tietoa tietokannasta halutaan hakea sen sijaan, että joudutaan kertomaan tarkalleen miten tieto kannasta löydetään. Loppukäyttäjälle deklaratiivinen kieli on huomattavasti helppokäyttöisempi kuin tavallinen ohjelmointikieli.

Arvoperustaisuuden takia ei samassa tilassa olevia objekteja voi tallentaa relaatioon useaan kertaan. Tämä on aivan luonnollista, koska niitä ei pystyttäisi erottamaan toisistaan ilman olioidentiteettiä. Relaation attribuuttien on siis sisällettävä kaikissa tilanteissa objektin identifiointiin tarvittavat tiedot. Relaation rivien identifiointiin ei yleensä kuitenkaan tarvita kaikkia attribuutteja vaan siihen riittää osajoukko attribuuteista ja usein jopa yksi attribuutti. Tällaista attribuuttia tai attribuuttijoukkoa kutsutaan avaimeksi. Relaatiomallissa objektiin voidaan viitata avaimen perusteella, kun taas oliomalleissa viittaus tapahtuu oliotunnisteen perusteella. Avaimen on oltava yksikäsitteinen ja minimaalinen, mikä tarkoittaa että:

1. Avaimen arvo identifioi relaation rivin yksikäsitteisesti kaikkien relaation rivien joukossa.
2. Jos avain koostuu useasta attribuutista, ei mitään attribuuttia voi poistaa avaimesta siten että ensimmäinen sääntö olisi yhä voimassa.

Relaatiolla voi luonnollisesti olla useita avaimia, joista yleensä käytännön syistä valitaan yksi pääavaimeksi (primary key). Muita avaimia kutsutaan ehdokasavaimiksi (candidate key) tai vain avaimiksi. Vierasavain (foreign key) tarkoittaa samasta arvojoukosta muodostettua attribuuttijoukkoa jossain muussa relaatiossa, jolloin vierasavaimen arvoa toisen relaation avaimen arvoon vertaamalla voidaan liittää objektit toisiinsa samalla tavalla kuin oliomalleissa viitataan suoraan toisen olion oliotunnisteeseen. Vierasavaimen ei tarvitse identifioida vierastaulun riviä. Esimerkkikuvassa kuvio 2.d Relaation 'Osastot' avaimia ovat 'ono' ja 'onimi', joista 'ono' on valittu pääavaimeksi. Relaation 'Työntekijät' avaimia ovat 'tno' ja 'hetu', joista 'tno' on valittu pääavaimeksi. 'tnimi' ei ole avain, koska on mahdollista, että on olemassa kaksi täsmälleen samannimistä ihmistä. Relaatiossa 'Työntekijät' attribuutti 'ono' on vierasavain, jolla löydetään työntekijän osaston tiedot relaatiosta 'Osastot'.

Osastot

ono	onimi	sijainti
10	tuotanto	Tampere
13	markkinointi	Helsinki
15	kehitys	Tampere

Työntekijät

tno	tnimi	hetu	ono	sukup
115	Pekka	111169-0123	10	m
117	Pirkko	010171-1234	10	n
121	Taisto	020255-2345	10	m
176	Pasi	030359-3456	15	m
187	Leila	040474-4567	15	n

Kuvio 2.D *Esimerkkirelaatiot.*

Avaimen ja vierasavaimen yhteensopivuuden varmistamiseksi ja tietokannan roskaantumisen estämiseksi tarvitaan viite-eheyttä (referential integrity). Viite-eheys tarkoittaa tietokantaan määriteltäviä sääntöjä, joilla varmistetaan, että vierasavainta vastaava pääavain on aina olemassa. Tällöin tietokannan hallintajärjestelmä estää lisäämästä vierasavainta, jota vastaavaa pääavainta ei ole olemassa ja estää tuhoamasta pääavainta, jota vastaavia vierasavaimia on olemassa. Vaihtoehtoisesti tietokannan hallintajärjestelmä voi vyöryttää objektin tuhoamisen myös vierasavaimia sisältäviin relaatioihin. Samoin pääavaimen arvon muuttaminen voidaan estää tai vyöryttää vierasavaimiin. Esimerkkikuvassa relaatioiden välille on luotu viite-eheys-vaatimus siten että jokaiselle työntekijälle oltava hänen 'ono' attribuuttiaan vastaava rivi relaatiossa 'Osastot'.

2.2. RELAATIOALGEBRA

Relaatioalgebra on relaatiomalliin sisältyvä tiedonkäsittelykieli. Relaatioalgebra on hyvin ilmaisuvoimainen kieli, mutta ei kuitenkaan omaa Turingin koneen ilmaisuvoimaa kuten perinteiset ohjelmointikielet. Relaatioalgebraa luotaessa on jouduttu tekemään kompromissi ilmaisuvoiman ja hallittavissa olevan operaatiojoukon välillä. Relaatiomalliin nykyisellään kuuluu, että kyselyt tehdään deklaratiivisella helppokäyttöisellä kyselykielellä kuten SQL ja tietokannan hallintajärjestelmään sisältyvä kyselyn optimoija luo kyselystä tehokkaimman mahdollisen relaatioalgebraoperaatioiden sarjan. Jotta loppukäyttäjät pystyvät hallitsemaan kielen, ja jotta kyselystä on

mahdollista automaattisesti luoda tehokas relaatioalgebraohjelma, täytyy operaatiojoukon olla hyvin rajattu.

Relaatioalgebra on *algebrallinen joukko-oppiin perustuva* relaatiotietokannan hallintakieli. On olemassa myös toinen, täsmälleen saman ilmaisuvoiman omaava kieli, *relaatiokalkyyli*, joka perustuu logiikkaan. Relaatiokalkyylejä on itse asiassa olemassa kaksi eri versiota, *arvojoukkokalkyyli* (*domain relational calculus*) ja *rivikalkyyli* (*tuple relational calculus*). Näiden kolmen vaihtoehtoisen kielen sisältämää ilmaisuvoimaa sanotaan relationaalisesti täydelliseksi ja kaikkien niiden pohjalta rakennettujen kyselykielten tulisi myös olla relationaalisesti täydellisiä, eli omata vähintään sama ilmaisuvoima. Logiikkaan perustuvia kalkyylikieliä voi väittää deklaratiivisemmiksi kuin relaatioalgebraa, koska relaatioalgebrassa operaatioiden suoritusjärjestys on kiinnitetty, mutta logiikkaan perustuvissa kielissä kerrotaan vain mitä tietoja halutaan saada kiinnittämättä lainkaan huomiota suoritusjärjestykseen. Tosin ennen kyselyn suorittamista tietokoneessa, on optimoijan joka tapauksessa luotava ensin tietokoneen sarjalliseen tietojenkäsittelyyn sopiva tehokas järjestyksen sisältävä operaatioiden sarja ja seuraavassa luvussa esiteltävät relaatiomallin päälle rakennetut kyselykielet piilottavat käsittelyjärjestyksen myös algebraan perustuvissa kielissä.

Relaatioalgebran operaatiot voidaan jakaa kahteen joukkoon. Toiseen joukkoon kuuluvat perinteiset joukko-opin operaatiot *unioni*, *leikkaus*, *erotus* ja *karteesinen tulo*. Toisen joukon muodostavat erityisesti relaatiomallia varten kehitetyt operaatiot kuten *valinta*, *projektio* ja *liitos*. Aikaisemmin mainitsin, että relaatiomallissa attribuuttien järjestyksellä ei ole merkitystä käyttäjän näkökulmasta. Relaatioalgebran operaatioista joillekin attribuuttien järjestys on kuitenkin olennaisen tärkeä. Asiasta ei aiheudu ongelmia, koska järjestys voidaan kiinnittää järjestystä vaativan operaation ajaksi. Joukko-opin operaatiot unioni, leikkaus, erotus ja karteesinen tulo ovat täysin riippuvaisia attribuuttien järjestyksestä. Lisäksi unioni, leikkaus ja erotus edellyttävät että niihin osallistuvat relaatiot ovat unioniyhteensopivia, mikä voidaan määritellä seuraavasti:

Määritelmä 2.A. Unioniyhteensopivat relaatiot ovat saman karteesisen tulon osajoukkoja. Toisin sanoen, jos meillä on relaatiot $R(a_1,a_2,...,a_n)$ ja $R(b_1,b_2,...,b_m)$ ja vastaavat arvojoukot ovat $Da_1,Da_2,...,Da_n$ ja $Db_1,Db_2,...,Db_m$, on oltava $n = m$ eli relaatioiden asteluku on sama ja lisäksi $Da_i \subseteq Db_i$ tai $Db_i \subseteq Da_i$ kun $1 \leq i \leq n$, eli vastinattribuuttien arvojoukot ovat samat, tai attribuutin arvojoukko on vastinattribuutin arvojoukon osajoukko.

Relaatioalgebran minimaalinen operaatiojoukko, eli pienin operaatiojoukko, jolla saavutetaan täysi relationaalinen ilmaisuvoima, sisältää vain viisi operaatiota. Nämä ovat valinta, projektio, unioni, erotus ja karteesinen tulo, jotka esittelen seuraavaksi. Kuvassa kuvio 2.g esitetään näiden ja muutaman lisäoperaation toiminta graafisesti.

- *Valinta* (*select*) operaatiota käytetään valitsemaan osajoukko relaation riveistä. Valinta operaattorina käytetään merkkiä σ (sigma) ja operaation käyttö on muotoa:

$$\sigma_{<\text{selection\_condition}>}(<\text{relation\_name}>)$$

Operaation tulosrelaation attribuuttijoukko on sama kuin lähtörelaation <relation_name>. Valintaehto <selection_condition> on Boolean-ehtolause tyyppiä

<clause> <boolean_operator> <clause>

jossa <boolean_operator> on jokin operaattoreista AND, OR tai NOT ja yksittäinen valintaehto <clause> on tyyppiä

<attribute_name> <comparison_op> <constant_value> tai
<attribute_name> <comparison_op> <attribute_name>

jossa <attribute_name> on eräs relaation <relation_name> attribuutin nimi ja <comparison_op> on jokin operaattoreista =, <, ≤, >, ≥ ja ≠.
Valintaoperaatiossa ehtolausetta testataan yksitellen jokaiseen relaation riviin ja valitaan tulokseen ne rivit, joihin ehtolause sopi. Valinta operaatiossa ei siten voi verrata toisiinsa relaation eri rivejä. Valinta on vaihdannainen (kommutatiivinen) eli

$$\sigma_{<cond1>}(\sigma_{<cond2>}(<R>)) = \sigma_{<cond2>}(\sigma_{<cond1>}(<R>))$$

ja sama lopputulos saadaan luonnollisesti aikaan Boolean-lausetta käyttävällä valinnalla

$$\sigma_{<cond1> \text{ AND } <cond2>}(<R>).$$

Valinta vastaa kysymykseen: anna ne relaation R rivit, jotka täyttävät ehdon <ehtolause>.

- *Projektio* operaatiota (*project*) käytetään relaation joidenkin attribuuttien eli sarakkeiden valitsemiseen ja uudelleenjärjestämiseen. Projektio-operaattorina käytetään merkkiä π (pii) ja operaation käyttö on muotoa:

  $$\pi_{<attribute\_list>}(<relation\_name>)$$

  Operaation tulosrelaation attribuutit ovat listassa <attribute_list> luetellut attribuutit siinä järjestyksessä kuin ne listassa ilmenevät. Tulosrelaation rivimäärä on sama tai pienempi kuin lähtörelaatiossa <relation_name>. Jos tulokseen ei tule mukaan mitään lähtörelaation avaimia, on todennäköistä, että tulokseen syntyisi duplikaatteja eli rivejä, joiden kaikki arvot ovat täsmälleen samat. Projektio kuitenkin automaattisesti poistaa duplikaatit jättäen tulosrelaatioon vain yhden samanarvoisista riveistä. Tämä takaa sen, että projektion tulos on aito relaatio, eli joukko rivejä. Projektio ei ole kommutatiivinen.

  Projektio vastaa kysymykseen: anna relaation R sarakkeet <sarakelista>.

- *Unioni* eli *yhdiste* (*union*) merkitään R∪S. Unionin tulosrelaatio sisältää kaikki rivit, jotka ovat joko relaatiossa R tai relaatiossa S. Duplikaattirivit eliminoidaan automaattisesti. Unionin operandirelaatioiden on oltava unioniyhteensopivia. Unionioperaatio ei huomioi attribuuttien nimiä vaan käsittelee relaatioita attri-

buuttien järjestyksen perusteella ja säilyttää niiden järjestyksen tulosrelaatiossa. Unioni on kommutatiivinen ja assosiatiivinen operaatio, eli
$R \cup S = S \cup R$ ja $R \cup (S \cup T) = (R \cup S) \cup T$.

Unioni vastaa kysymykseen: anna kaikki relaatioiden R ja S erilaiset rivit.

- *Erotus* (*difference*) merkitään R - S. Erotuksen tulosrelaatioon tulevat kaikki rivit, jotka ovat relaatiossa R, mutta eivät relaatiossa S. Myös erotus edellyttää operandirelaatioilta unioniyhteensopivuutta ja käsittelee attribuutteja niiden järjestyksen perusteella. Erotus ei ole kommutatiivinen eikä assosiatiivinen.

Erotus vastaa kysymykseen: anna ne R:n rivit joita ei ole S:ssä.

- *Karteesinen tulo* (*cartesian product*) merkitään R×S. Karteesinen tulo yhdistää kaikki relaation R rivit kaikkiin relaation S riveihin. Karteesisen tulon $R(A_1,A_2,...,A_n) \times S(B_1,B_2,...,B_m)$ tulosrelaation n + m attribuuttia ovat $Q(A_1,A_2,...,A_n,B_1,B_2,...,B_m)$ tässä järjestyksessä. Jos relaatiossa R on i_R riviä ja relaatiossa S on i_S riviä sisältää niiden karteesinen tulo R×S $i_R \times i_S$ riviä. Karteesinen tulo käsittelee operandirelaatioitaan attribuuttien järjestyksen mukaan. Karteesinen tulo ei ole kommutatiivinen, koska attribuuttien järjestyksellä on merkitys lopputuloksessa, mutta on assosiatiivinen, koska rivien järjestyksellä ei relaatiossa ole merkitystä.

  Karteesinen tulo vastaa kysymykseen: anna kaikki rivit jotka saadaan aikaan lisäämällä jokainen S:n rivi vuorollaan jokaisen R:n rivin perään.

Havainnollistan seuraavaksi yksinkertaisella esimerkillä, jokaisen relaatioalgebran perusoperaation toimintaa.

R

A	B	C
a	b	c
e	f	g
i	j	k

S

D	E	F
o	p	q
a	b	c

Kuvio 2.E *Esimerkkirelaatiot R ja S, jotka oletetaan unioniyhteensopiviksi.*

R ∪ S

X	Y	Z
a	b	c
e	f	g
i	j	k
o	p	q

R - S

X	Y	Z
e	f	g
i	j	k

R X S

A	B	C	D	E	F
a	b	c	o	p	q
a	b	c	a	b	c
e	f	g	o	p	q
e	f	g	a	b	c
i	j	k	o	p	q
i	j	k	a	b	c

$\pi_{C,A}(R)$

C	A
c	a
g	e
k	i

$\sigma_{B = f}(R)$

A	B	C
e	f	g

Kuvio 2.F *Esimerkkikyselyjen tulokset minimaalisen relaatioalgebraoperaatiojoukon operaatioilla.*

Minimaalisen operaatiojoukon lisäksi on määritelty joukko operaatioita, jotka helpottavat kyselyiden suorittamista, mutta eivät lisää ilmaisuvoimaa. Kaikki nämä ylimääräiset operaatiot on siis mahdollista määritellä minimaalisen operaatiojoukon operaatioiden yhdistelmänä. Hyödyllisiä lisäoperaatioita ovat leikkaus, osamäärä, liitos, luonnollinen liitos ja ulkoliitos jotka esittelen seuraavaksi.

- *Leikkaus* (*intersection*) merkitään R∩S. Leikkauksen tulosrelaatio sisältää rivit, jotka ovat sekä relaatiossa R että relaatiossa S. Leikkaus edellyttää operandirelaatioilta unioniyhteensopivuutta ja käsittelee attribuutteja niiden järjestyksen perusteella. Leikkaus on kommutatiivinen ja assosiatiivinen operaatio.

Leikkaus vastaa kysymykseen: anna ne rivit, jotka ovat sekä R:ssä että S:ssä.

- *Osamäärä* (*quotient*, *division*) merkitään R÷S. Osamäärä operaation lähtörelaation S attribuuttien on oltava aito osajoukko relaation R attribuuteista. Operaation tulosrelaation attribuuttijoukkoon kuuluvat ne R:n attribuutit, joita ei ole relaatiossa S. Jos operandirelaatioiden attribuutit ovat R(A_1, A_2,...,A_n, B_1, B_2,...,B_m) ja S(B_1, B_2,...,B_m) ovat tulosrelaation attribuutit Q(A_1, A_2,...,A_n). Tulosrelaation riveiksi tulevat ne relaation R rivien attribuutteja A1,...,An vastaavat osat, joille löytyy kyseisen rivin B1,...,Bm attribuutteja vastaava korvaus jokaista relaation S riviä kohti. Duplikaatit poistetaan automaattisesti.

Osamäärä vastaa kysymykseen: anna ne R:n osarivit joiden toinen osa vastaa jokaista S:n riviä.

- *Liitos* operaatiota (*join*) käytetään liittämään kahden relaation toisiinsa liittyvät rivit yhdeksi riviksi. Liitos operaattoria merkitään tässä ∞ ja operaation käyttö on muotoa:

$R\infty_{<join\_condition>}S$

Operaation tulosrelaation attribuuttijoukko määräytyy samoin kuin karteesisessa tulossa, eli kaikki R:n ja S:n attribuutit samassa järjestyksessä. Liitosehto <join_condition> määrää millä ehdoilla R:n ja S:n rivit liitetään yhteen ja on muotoa

<condition> {AND <condition>}•

jossa jokainen yksittäinen ehto <condition> on muotoa

$A_i \theta B_j$

jossa θ (theta) on jokin vertailuoperaattoreista =, <, ≤, >, ≥ ja ≠ ja A_i on jokin relaation R attribuutin nimi ja B_j on jokin relaation S attribuutin nimi. Tällaista liitosta kutsutaan myös theta-liitokseksi. Mikäli θ on '=' kutsutaan liitosta equijoiniksi. Mikäli ehtolause ei toteudu rivien välillä tai jos ehdossa käytettävän attribuutin arvona on NULL ei rivejä oteta tulokseen.

Liitos vastaa kysymykseen: liitä jokaiseen R:n riviin ne S:n rivit, jotka toteuttavat annetun ehdon.

- *Luonnollinen liitos* (*natural join*) on equijoin-liitoksen yksinkertaisempi erikoistapaus ja operaation käyttö on yksinkertaisesti muotoa:

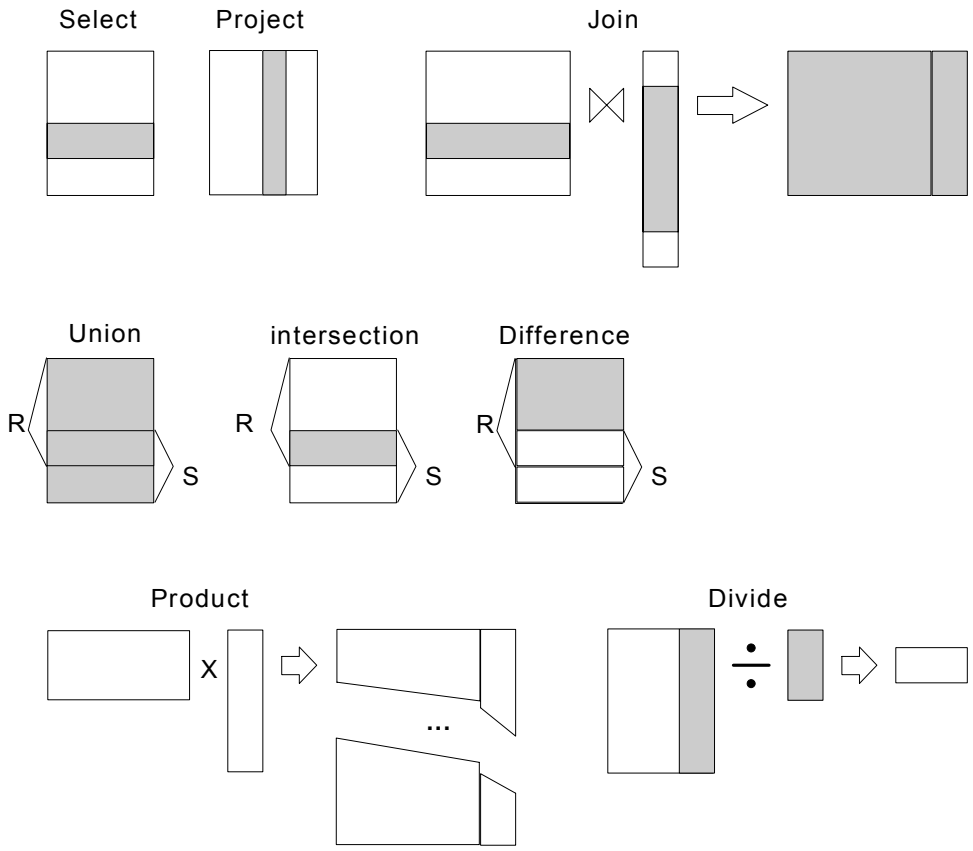
  $R\infty S$

Automaattisena liitosehtona on, että samannimisten attribuuttien arvojen on oltava samat. Luonnollisessa liitoksessa ei tulosrelaatioon oteta mukaan liitosattribuutteja relaatiosta S, jolloin liitosattribuutit eivät kahdennu.

- *Ulkoliitos* (*outerjoin*) eroaa theta-liitoksesta siten, että ulkoliitos säilyttää jommankumman tai kummankin operandirelaation kaikki rivit tulosrelaatiossa vaikka ne eivät täyttäisikään liitosehtoa. Toisesta relaatiosta haettaville attribuuteille annetaan liitosehtoa toteuttamattomilla riveillä arvoksi NULL, joka tarkoittaa, että arvoista ja niiden olemassaolosta ei ole tietoa. Ulkoliitosta merkitään tässä merkinnöillä $>\infty$, $\infty<$ ja $>\infty<$, jotka tarkoittavat vastaavasti vasen ulkoliitos, oikea ulkoliitos ja täydellinen ulkoliitos. Operaation käyttö on muotoa

  $R>\infty_{<join\_condition>}S$ tai $R\infty<_{<join\_condition>}S$ tai $R>\infty<_{<join\_condition>}S$

  Vasemmassa ulkoliitoksessa otetaan aina mukaan kaikki R:n rivit ja ellei S:stä löydy vastin riviä korvataan S:n attribuutit NULL arvoilla. Vastaavasti oikeassa ulkoliitoksessa otetaan mukaan kaikki S:n rivit ja täydellisessä ulkoliitoksessa kaikki rivit sekä R:stä, että S:stä. Muuten ulkoliitos toimii kuten theta-liitos.



Kuvio 2.G *Relaatioalgebran operaatioiden vaikutuksia graafisesti tauluun kuvattuna. Select, project ja join ovat erityisesti relaatiomallia varten luotuja operaatioita, muut operaatiot ovat suoraan matematiikan joukko-opin operaatioita.*

Relaatioalgebran operaatioita voidaan yhdistää kirjoittamalla niitä sisäkkäin tai peräkkäin, jolloin yhden relaatioalgebraoperaation tulos on toisen relaatioalgebraoperaation operandi. Pelkästään valinta- ja projektio-operaatioita yhdistämällä saadaan jo vastauksia useisiin luonnollisiin tietotarpeisiin yhdestä relaatiosta.

Tehdään esimerkkikysely kuvion kuvio 2.d relaatioihin. Kysymykseen, minkä nimiset työntekijät työskentelevät osastolla 15, saamme vastauksen seuraavanlaisella relaatioalgebraoperaatioiden yhdistelmällä:

$$\pi_{tnimi}(\sigma_{ono = 15}(\text{Työntekijät}))$$

Ensin siis valitaan relaatiosta 'Työntekijät' vain rivit, joilla attribuutin 'ono' arvo on 15 ja sen operaation tulosrelaatiosta projisoidaan vain sarake 'tno'.

Jos haluamme tietää nimet ja työntekijänumerot niiltä naisilta, jotka työskentelevät osastolla 10, teemme kyselyn:

$$\pi_{tnimi,\ tno}(\sigma_{sukup = n \wedge ono = 10}(\text{Työntekijät}))$$

Yhdistämällä eri relaatioissa sijaitsevia tietoja pääsemme tekemään huomattavasti monipuolisempia kyselyitä. Jos haluamme tietää ketkä työntekijät käyvät työssä Tampereella, emme löydä tietoa yhdestä relaatiosta, vaan meidän on yhdistettävä kaksi relaatiota seuraavasti:

$$\pi_{tnimi}(\sigma_{sijainti = Tampere}(\text{Osastot} \infty \text{Työntekijät}))$$

Teimme siis ensin luonnollisen liitoksen Osastojen ja Työntekijöiden välillä, jolloin jokainen työntekijä liitettiin osastoon osastonumeron perusteella. Tämän operaation tulosrelaatiosta sitten valittiin Tampereella sijaitsevat osasto-työntekijä-yhdistelmät ja lopuksi projisoitiin tulokseksi työntekijän nimi.

Koska luonnollinen liitos ei kuulu minimaaliseen relaatioalgebraoperaatioiden joukkoon, voidaan se esittää myös minimaalisen operaatiojoukon operaatioiden yhdistelmällä. Esitän seuraavaksi saman tuloksen antavan kyselyn käyttäen vain minimaalisen operaatiojoukon operaatioita. Valintaehdossa erotetaan kahden yhdistetyn relaation samannimiset attribuutit toisistaan kirjoittamalla attribuutin lähtörelaationnimi attribuutin eteen pisteellä erotettuna.

$$\pi_{tnimi}(\sigma_{Osastot.ono = Työntekijät.ono \wedge sijainti = Tampere}(\text{Osastot} \times \text{Työntekijät}))$$

Toinen kyselyn formulointitapa on käyttää apuna välirelaatioita, jolloin mutkikas kysely voidaan suorittaa pienemmissä helpommin hallittavissa osissa. Tehdään edellinen kysely kolmessa osassa tallentaen välitulos aina välirelaatioon:

$$\begin{aligned}
&\text{Temp1} \leftarrow \text{Osastot} \times \text{Työntekijät} \\
&\text{Temp2} \leftarrow \sigma_{Osastot.ono = Työntekijät.ono \wedge sijainti = Tampere}(\text{Temp1}) \\
&\pi_{tnimi}(\text{Temp2})
\end{aligned}$$

Välirelaatioiden avulla on mahdollista myös uudelleennimetä attribuutit. Uudelleennimeäminen tapahtuu luettelemalla attribuuttien uudet nimet siinä järjestyksessä kuin attribuutit kyselyn projektiolauseessa luetellaan. Jos edellisessä esimerkissä valitaan työntekijän nimen lisäksi osaston nimi, jolla hän työskentelee ja nimetään attribuutit saadaan seuraava kysely, jossa 'tnimi' saa uuden nimet 'työntekijä' ja 'onimi' saa uuden nimen 'osasto':

$$\begin{aligned}
&\text{Temp1} \leftarrow \text{Osastot} \times \text{Työntekijät} \\
&\text{Temp2} \leftarrow \sigma_{Osastot.ono = Työntekijät.ono \wedge sijainti = Tampere}(\text{Temp1}) \\
&\text{Tulos(työntekijä, osasto)} \leftarrow \pi_{tnimi,\ onimi}(\text{Temp2})
\end{aligned}$$

Relaatioalgebrasta löytyy tarkempi esitys esimerkiksi lähteistä [ElNa94], [Ull88] tai [Codd90].

2.3. DEKLARATIIVISET KYSELYKIELET

Relaatiomalli itsessään ei sisällä tehokkaita ja helppokäyttöisiä työkaluja tiedonkäsittelyyn, vaan antaa paremminkin perusedellytykset sellaisten luomiselle. Tällaisten

helppokäyttöisten deklaratiivisten tiedonkäsittelykielien olemassaolo mallin päällä on kuitenkin alusta asti ollut eräs mallin vahvuuksia. Relaatiomallin yksinkertaisuus on antanut mahdollisuuden luoda useita erilaisia kyselykieliä, mutta yhteinen nimittäjä kaikille kyselykielille on se, että ne ovat deklaratiivisia (joitain suoraan relaatioalgebraan perustuvia kieliä poislukien) ja ilmaisuvoimaltaan vähintään relationaalisesti täydellisiä. Tietokannanhallintajärjestelmän kyselyn optimoija luo kyselystä tehokkaan relaatioalgebralauseiden sarjan, jolla käyttäjän haluama tulosrelaatio tuotetaan. Lisäksi kaikki kyselykielet sisältävät joitakin ensiarvoisen tärkeitä ominaisuuksia, jotka relaatiomallista puuttuvat kuten aggregointi-funktioita (erimerkiksi summa, lukumäärä, min, max), aritmeettiset operaatiot, sekä komennot tiedon tallennusta, tuhoamista ja muuttamista varten. Edellä mainittujen lisäominaisuuksien vuoksi kyselykielet ovat itse asiassa ilmaisuvoimaisempia kuin mitä relationaalisesti täydelliseltä kyselykieleltä vaaditaan.

Relaatiomallin kyselykielistä de facto standardiksi on muodostunut SQL (Structured Query Language), joka perustuu pitkälti relaatioalgebraan. SQL:n suosion tärkeimmät syyt ovat ISO:n standardointi, kielen helppokäyttöisyys ja ensimmäiset vaikutusvaltaiset tukijat. Toinen kyselykieli, jota esimerkiksi relaatiomallin isä E. F. Codd pitää SQL:ää parempana [Codd90 s.13] on QUEL, joka perustuu rivikalkyyliin. Hyvä esimerkki arvoaluekalkyyliin perustuvasta kielestä on QBE (Query By Example). Näitä kyselykieliä käsitellään tarkemmin lähteissä [Ull88] ja [ElNa94]. Tässä tutkimuksessa keskitytään jatkossa SQL-kieleen ja sen laajentamiseen laajennettuun relaatioalgebraan sopivaksi. Luvussa 5 "nsql - laajennettu sql-kieli" esiteltävä itse luomani kyselykieli NSQL ei noudata SQL:n merkintätapoja, mutta perustuu kuitenkin relaatioalgebraan ja sisältää suurinpiirtein samat operaatiot kuin SQL. NSQL-kieli sisältää lisäksi laajennetun relaatioalgebran vaatimia uusia operaatioita (nest, unnest, closure) ja siitä puuttuu joitain laajennetun relaatiomallin myötä tarpeettomaksi käyviä ilmauksia (group by, having).

SQL kyselyn perusrakenne on tyyppiä:

```
SELECT <attribute_list>
FROM <relation_list>
WHERE <condition>
```

- <attribute_list> sisältää listan attribuuteista, jotka halutaan tulokseen
- <relation_list> sisältää relaatioiden nimet, joita kyselyn suorittamiseen tarvitaan
- <condition> on Boolean ehtolause, joka rajoittaa tulokseen tulevia rivejä

SQL:n perusrakenteesta käytetään jatkossa lyhennystä SFW-kysely (Select From Where).

SQL:n perusrakenne vastaa relaatioalgebran ilmausta:

$$\pi_{<attribute\_list>}(\sigma_{<condition>}(R_1 \times R_2 \times ... \times R_n))$$

missä $R_1, R_2, ..., R_n$ ovat listan <relation_list> alkiot.

Tavallisen SQL-kyselyn muunnos relaatioalgebran lauseiksi tapahtuu siten, että

- SELECT lauseessa luetellut attribuutit projisoidaan RA-kyselyssä π-operaatiolla,
- FROM lauseessa luetelluista relaatioista otetaan karteesinen tulo keskenään ja
- WHERE ehdon ehtolause on suoraan RA:n valintaoperaation σ ehtolause.

Kyselynoptimoijan ei luonnollisestikaan tarvitse tehdä muunnosta näin suoraviivaisesti, vaan käytännössä valinta- ja projektio-lauseita suoritetaan ennen raskaita tulo-operaatioita. Esimerkeissä käytämme kuitenkin suoraviivaista muunnosta SQL:n ja relaatioalgebran välillä yksinkertaisuuden vuoksi ja siksi, että tehokkain RA-operaatiosarja riippuu tietokannan sen hetkisestä sisällöstä, jota me emme tiedä.

SQL-kieli esitellään tarkemmin lähteissä [Ull88] ja [ElNa94]. Tässä tutkimuksessa palataan tarkemmin strukturoituihin kyselykieliin, niiden operaatioihin ja kyselyjen muunnokseen RA-lauseiksi luvussa 5.

3. NF2-LAAJENNUS RELAATIOMALLIIN: NON FIRST NORMAL FORM RELATIONAL MODEL

3.1. NF2-MALLIN OMINAISUUDET

NF2-relaatiomallissa luovutaan attribuuttien arvojen atomisuusvaatimuksesta ja sallitaan attribuutin sisältää arvoinaan relaatioita. NF2-mallia kutsutaan myös lyhenteillä NF^2, ¬1NF ja NFNF (non first normal form). Attribuuttien atomisuudesta luopumista ehdotti ensimmäisenä Makinouchi [Mak77] ja sittemmin mallia on kehitetty edelleen ja yleistetty useiden tutkijoiden toimesta. Itse asiassa Coddin alkuperäisissä papereissa ei attribuuttien atomisuusvaatimusta ollut, mutta siitä tuli pian osa relaatiomallin määritelmää. NF2-malli ei sinänsä lisää relaatiomallin ilmaisuvoimaa, vaan relaatiomallin ja NF2-mallin ilmaisuvoima on täsmälleen sama, minkä Paredaens ja Gucht ovat todistaneet [ParGuc92]. Toisin sanoen kaikki tieto, mikä voidaan esittää NF2-mallilla voidaan esittää myös 1NF-muodossa ja kaikki tieto, mikä kyselyillä voidaan NF2-mallista tuottaa voidaan tuottaa myös 1NF-mallista. NF2-mallin etuja ovat kyky käsitellä rakenteellisia objekteja ja siten kyky esittää tietokanta käyttäjälle helpommin hahmotettavassa muodossa. Reaalimaailmassa objektin osana voi atomisen arvon ohella hyvin luonnollisesti olla joukko arvoja ja 1NF-mallissa tämän kieltäminen on johtanut tiedon hajoamiseen useisiin relaatioihin. Tämän vuoksi taas on jouduttu luomaan keinoja tiedon eheyden ylläpitämiseksi (referential integrity). Relaatioarvoisten attribuuttien salliminen antaa myös lisää mahdollisuuksia rakenteellisten objektien hallinnan siirtämiseksi DBMS:n hoidettavaksi. 1NF-tietokannoissa rakenteellisten objektien hallinta hoidetaan esiprosessoritekniikalla, jolloin DBMS ei ymmärrä tiedon rakennetta, vaan tieto saa merkityksen vasta DBMS:n ulkopuolisessa esiprosessorissa, joka osaa esimerkiksi kuvia käsiteltäessä jakaa pitkän bittijonon kuvan riveiksi ja tietää siten jotain bittien keskinäisestä suhteesta. Tämän tyyppisiä tiedonkäsittelytarpeita on muun muassa CAD-, dokumentti- ja kuva-tietokannoissa. NF2-tietokannan hallitseminen käyttäjälle on helpompaa jo yksin siitä syystä, että erillisten relaatiotaulujen määrä vähenee huomattavasti. Lisäksi yleisimmin tarvitut tiedonhaut oikein suunnitellusta NF2-tietokannasta ovat nopeita, koska objekteja ei tarvitse koota useita relaatioita yhteen liittämällä. Myös yleisesti käytettyjen kyselyiden formulointi on helppoa NF2-mallissa, koska tiedot on tietokannassa valmiiksi ryhmitelty luonnollisten ja yleisimmin käytettyjen ryhmittelyperusteiden mukaisesti. Laajennetussa relaatiomallissa 1NF-relaatio on vain NF2-relaation erikoistapaus.

Nykyisissä suurissa tietokannoissa on jopa satoja eri tauluja. Pystyäkseen hallitsemaan kantaa tehokkaasti ja oikein tulisi käyttäjän muistaa kaikkien taulujen nimet, tietosisällöt ja tarkoitukset ja lisäksi pystyä hahmottamaan taulujen suhteet toisiinsa. On huomattavasti helpompaa hallita esimerkiksi kymmenen NF2-relaation nimet ja tietosisältö, kuin vastaava informaatio lukuisina 1NF-relaatioina, sillä jokaisen NF2-relaation rakenne koostuu luonnollisista reaalimaailman objekteista ja niiden osista. Tietojen keskinäiset suhteet ilmenevät NF2-relaatiosta hyvin intuitiivisesti, kuten käy ilmi tarkastelemalla kuvan kuvio 3.a NF2-relaatiota ja kuvan kuvio 3.b 1NF-esitystä samoista tiedoista. NF2-relaatiota piirrettäessä tulisi upotettu taulu piirtää kehyksineen ja attribuutin nimineen, jokaisen rivin jokaiseen relaatioarvoiseen sa-

rakkeeseen. Tilankäytöllisistä syistä upotettujen taulujen rakenneinformaatio on kuvasta jätetty pois.

Yritys

ono	onimi	sijainti	Työntekijät									Tuotteet	
			tyno	tynimi	hetu	esimies	palkka	Kurssit		Lapset		*tuno*	tunimi
								kuno	pvm	*lnimi*	synt		
10	tuotanto	tampere	14	erkki	111169-0123	0	24000	11 15 21	2.3.81 5.8.91 1.9.90	mari veli	93 90	776 668 445	vitkain vatkain vemputin
			25	mira	010171-1234	14	19000	11 31	4.5.91 2.2.95	jaakko	96		
			65	heikki	020255-2345	25	13000	11	4.4.97	viljo saara	89 90		
20	markkinointi	helsinki	59	sirkka	030359-3456	14	14000	11 42	4.5.91 7.9.95			112 227 265	mainos tarjous mtutk
30	laskutus	tampere	34	paavo	040471-4567	1	18000	15	5.8.91	kati ilona	82 84	344 321	lasku karhu
			78	heli	050578-5678	34	7000			sepe jope	91 95		

Kuvio 3.A *Yritys esitettynä yhtenä NF2-relaationa.*

Osastot

ono	onimi	sijainti
10	tuotanto	tampere
20	markkinointi	helsinki
30	laskutus	tampere

Työntekijät

tyno	tynimi	hetu	ono	esimies	palkka
14	erkki	111169-0123	10	1	24000
25	mira	010171-1234	10	14	19000
65	heikki	020255-2345	10	25	13000
59	sirkka	030359-3456	20	14	14000
34	paavo	040474-4567	30	1	18000
78	heli	050578-5678	30	34	7000

Lapset

tyno	*lnimi*	synt
14	mari	93
14	veli	90
25	jaakko	96
65	viljo	89
65	saara	90
34	kati	82
34	ilona	84
78	sepe	91
78	jope	95

Tuotteet

tuno	osasto	tunimi
776	10	vitkain
668	10	vatkain
445	10	vemputin
112	20	mainos
227	20	tarjous
265	20	mtutk
344	30	lasku
321	30	karhu

TtKurssit

tyno	*kuno*	pvm
14	11	2.3.81
14	15	5.8.91
14	21	1.9.90
25	11	4.5.91
25	31	2.2.95
65	11	4.4.97
59	11	4.5.91
59	42	7.9.95
34	15	5.8.91

Kuvio 3.B *Yritys esitettynä perinteisellä relaatiomallilla.*

Esitän seuraavaksi sääntöpohjaisen määritelmän NF2-relaatiolle, jollaista käytetään lähteessä [RoKoBa87].

Määritelmä 3.A: *NF2-relaatio*

Tietokantakaavio koostuu joukosta sääntöjä, jotka ovat muotoa $R_j \leftarrow (R_{j1}, R_{j2},...,R_{jn})$. R_j on relaatio ja R_{ji} on attribuutti, kun $1 \leq i \leq n$. Attribuutti on relaatioarvoinen (rakenteellinen) jos se esiintyy jonkin säännön vasemmalla puolella, muussa tapauksessa se on atominen.

Esimerkin kuvio 3.a ja kuvio 3.c NF2-tietokannan relaatiokaavio sääntöinä ilmaistuna:

Yritys ← (ono, onimi, sijainti, Työntekijät, Tuotteet)
Työntekijät ← (tyno, tynimi, hetu, esimies, palkka, Kurssit, Lapset)
Tuotteet ← (tuno, tunimi)
Kurssit ← (kuno, pvm)
Lapset ← (lnimi, synt)
Kurssi ← (kuno, kunimi, sisältö)

1NF-relaatiomallia käsiteltäessä esitettiin normaalimuotoja, joiden tarkoituksena oli vähentää redundanssia, eli samojen tietojen toistamista tietokannassa. Jos laskemme tallennettujen arvojen määrän edellisissä kahdessa esimerkkitietokannassamme, huomaamme että 1NF-esityksessä on arvoja kaikkiaan 123 kpl ja NF2-esityksessä 90 kpl, eli tässä tapauksessa 1NF-esitykseen on tallennettu peräti 37% enemmän arvoja kuin NF2-esityksessä ja tämä kaikki on redundanttia tietoa, koska tietokannat esittävät täsmälleen saman tietosisällön. 1NF-tietokannassa on toistettava tietoa, jota tarvitaan toisiinsa liittyvien tietojen yhteenliittämiseen. NF2-mallissa näyttäisi olevan vähemmän redundanssia, kuin 1NF-mallissa, eli ensimmäisen normaalimuodon vaatimus on tuonut relaatiomalliin huomattavan redundanssin, jota edelleen normalisoimalla on sittemmin pyritty pienentämään.

Myös NF2-mallissa joudutaan joissakin tapauksissa tieto jakamaan useampiin relaatioihin. Esimerkiksi many-to-many suhteissa redundanssi kasvaisi valtavasti, jos tällaisessa suhteessa olevista objekteista toinen upotettaisiin toisen sisälle. Klassinen esimerkki many-to-many suhteesta on oppilas-kurssi suhde. Edellisessä esimerkissäkin työntekijä-relaatioon on upotettu vain kurssin avaintieto ja kyseisen kurssin ja oppilaan suhteeseen liittyvää tietoa. Pelkästään kurssiin liittyvät tiedot täytyy sekä 1NF, että NF2-mallissa tallentaa erilliseen relaatioon, jotta näitä tietoja ei jouduta toistamaan jokaisen kurssin suorittaneen oppilaan kohdalla. Relaatio 'Kurssi' on esitetty kuvassa kuvio 3.c.

Kurssi

kuno	kunimi	sisältö
11	sql perusteet	opetellaan...
15	johtaminen 1	uudet johtamis...
21	olioteknologia 1	peusteet olio...
31	corba 1	corba rajapin...
42	markkina tutk.	tilastollisten...

Kuvio 3.C *Relaatio Kurssi sisältyy yritystietokannan NF2 ja 1NF esityksiin.*

3.2. TIETOKANTAKAAVION UUDELLEENMUOTOILU

Kun olemme määritelleet NF2-mallin täytyy meidän seuraavaksi määritellä operaatiot mallin käsittelyyn. Matematiikan relaatioteoriassa, johon relaatiomalli perustuu, ei attribuuttien atomisuusvaatimusta ole olemassa, vaan se on lisätty tietokantamalliin myöhemmin. Niinpä relaatiomallin matemaattinen perusta ei muutu olennaisesti 1NF-mallia NF2-malliksi laajennettaessa, vaan sama operaatiojoukko käy NF2-mallinkin käsittelyyn. Operaatiot on toki määriteltävä uudelleen ja näitä formaaleja joukko-opillisia määritelmiä löytyy lähteistä [RoKoSi88], [ScSc86], [MoNgEm96], [FisTho83]. Relaatioalgebraan tarvitaan 1NF-mallista tuttujen operaatioiden lisäksi kaksi uutta operaatiota relaation rakenteen muokkaamiseen. *Nest* (ν) pakkaa joukon attribuutteja alirelaatioksi ja *unnest* (μ) purkaa alirelaation attribuutit alirelaa-

tiohierarkiassa edellisen tason relaation attribuuteiksi. Nest ja Unnest purkavat ja pakkaavat yhden alirelaatiotason kerrallaan. Kirjallisuudessa on esitetty myös operaatiot *Unpack* ja *Pack*, joista Unpack purkaa koko alirelaatiohierarkian 1NF-relaatioksi ja Pack pakkaa relaation takaisin alkuperäiseen muotoonsa. Näiden operaatioiden avulla on kätevää määritellä relaatioalgebran operaatioita NF2-mallille helposti ymmärrettävässä muodossa, mutta muuten ne eivät ole kovin käyttökelpoisia, koska koko NF2-relaation purkaminen ja takaisin pakkaaminen käytännössä olisi hyvin raskasta. Lisäksi pack-operaatio on varsin huonosti määritelty ja sitä voi käyttää vain yhdessä unpack-operaation kanssa, koska pack-operaatio saa tiedot relaation rakenteesta unpack-operaatiolta. Pack- ja unpack-operaatioita ei tässä tutkimuksessa oteta lainkaan mukaan relaatioalgebraan.

Määritellään seuraavaksi uudet relaatioalgebran operaatiot nest ja unnest.

- *Nest* (ν) muokkaa relaatiokaaviota seuraavasti. Olkoon meillä relaatiokaavio $R(A_1,...,A_n,B_1,...,B_m)$, jossa on siis n + m attribuuttia. Operaatio $\nu_{RB = \{B1,...,Bn\}}(R)$ tuottaa tällöin relaatiokaavion $R_2(A_1,...,A_n, RB(B_1,...,B_m))$, eli pakkaa attribuutit $B_1,...,B_m$ alirelaatioksi RB. Kun relaatiokaavion R mukainen relaatio r pakataan relaatiokaavion R_2 mukaiseksi relaatioksi r_2 ovat tulosrelaation rivit

$r_2 = \{t| \exists t' \in r \wedge t[A_1,...,A_n] = t'[A_1,...,A_n] \wedge t[RB] = \{t_b[B_1,...,B_m]| t_b \in r \wedge t_b[A_1,...,A_n] = t'[A_1,...,A_n]\}\}$.

- *Unnest* (μ) muokkaa relaatiokaaviota seuraavasti. Olkoon meillä relaatiokaavio $R(A_1,...,A_n, RB(B_1,...,B_m))$. Operaatio $\mu_{RB}(R)$ tuottaa relaatiokaavion $R_2(A_1,...,A_n,B_1,...,B_m)$, eli purkaa alirelaation RB attribuutit relaation R_2 attribuuteiksi. Kun relaatiokaavion R mukainen relaatio r puretaan relaatiokaavion R_2 mukaiseksi relaatioksi r_2 ovat tulosrelaation rivit

$r_2 = \{t| \exists t' \in r \wedge t[A_1,...,A_n] = t'[A_1,...,A_n] \wedge t[B_1,...,B_m] \in t'[RB]\}$

Pakataan esimerkin vuoksi kuvan kuvio 3.d 'Osastot'-relaation attribuutit 'tyno', 'tynimi', 'hetu', 'esimies' ja 'palkka' alirelaatioksi 'Työntekijät' RA-lauseella:

$$\text{Yritys} = \nu_{\text{Työntekijät} = \{\text{tyno, tynimi, hetu, esimies, palkka}\}}(\text{Osastot})$$

Tulosrelaatio Yritys on esitetty kuvassa kuvio 3.e.

Osastot

ono	onimi	sijainti	*tyno*	tynimi	hetu	esimies	palkka
10	tuotanto	tampere	14	erkki	111169-0123	1	24000
10	tuotanto	tampere	25	mira	010171-1234	14	19000
10	tuotanto	tampere	65	heikki	020255-2345	25	13000
20	markkinointi	helsinki	59	sirkka	030359-3456	14	14000
30	laskutus	tampere	34	paavo	040474-4567	1	18000
30	laskutus	tampere	78	heli	050578-5678	34	7000

Kuvio 3.D *Yrityksen osastot ja työntekijät 1NF-relaatioina.*

Yritys

ono	onimi	sijainti	Työntekijät				
			tyno	tynimi	hetu	esimies	palkka
10	tuotanto	tampere	14	erkki	111169-0123	0	24000
			25	mira	010171-1234	14	19000
			65	heikki	020255-2345	25	13000
20	markkinointi	helsinki	59	sirkka	030359-3456	14	14000
30	laskutus	tampere	34	paavo	040471-4567	1	18000
			78	heli	050578-5678	34	7000

Kuvio 3.E *Yrityksen osastot ja osastojen työntekijät NF2-esityksenä.*

Vastaavasti voimme purkaa kuvion kuvio 3.e relaation 'Yritys' kuvan kuvio 3.d relaatioksi 'Osastot' komennolla:

$$\text{Osastot} = \mu_{\text{Työntekijät}}(\text{Yritys})$$

Kyselyn optimointitekniikat saavat NF2-mallissa uusia piirteitä ja niitä on tutkittu esimerkiksi lähteessä [Pää95].

3.3. PNF-NORMAALIMUOTO (PARTITIONED NORMAL FORM)

Yleisesti nest- ja unnest- operaatiot eivät ole toistensa käänteisoperaatioita, joten on syytä esitellä uusi normaalimuoto rajoittamaan NF2-relaation rakennetta siten, että nest- ja unnest-operaatioista saadaan toisilleen käänteisiä. Tällaisen rajoituksen määrittää PNF-normaalimuoto joka määritellään esimerkiksi lähteessä [RoKoSi88]. Esimerkistä kuvio 3.f huomaamme, että ottamalla suoraviivaisesti unionin kahdesta NF2-relaatiosta saamme helposti tulokseksi NF2-relaation, jolle nest ja unnest eivät ole käänteisiä operaatioita. Purkamalla relaatiosta YritysA∪YritysB alirelaatio 'Työntekijät' ja sen jälkeen pakkaamalla se takaisin, seuraavasti

$$\nu_{\text{Työntekijät} = \{\text{tyno}\}}(\mu_{\text{Työntekijät}}(\text{YritysA} \cup \text{YritysB})),$$

saadaan kuvion kuvio 3.g mukainen relaatio, eikä se ole sama kuin lähtörelaatio. PNF-muotoisessa relaatiossa sitävastoin nest ja unnest ovat toistensa käänteisoperaatioita ja kuvion kuvio 3.g esittämä relaatio onkin PNF-muodossa.

Määritelmä 3.B *PNF-relaatio*

Merkitköön A atomisten attribuuttien joukkoa ja X relaatioarvoisten attribuuttien joukkoa. Tällöin relaatio R(A,X) on PNF-muodossa, jos
1. kahdella relaation rivillä ei ole saman arvoista riviä attribuuttijoukossa A ja
2. kaikilla riveillä kaikki joukon X alirelaatiot ovat PNF-muodossa.

Toisin sanoen relaation atomisten attribuuttien joukon tulee muodostaa relaation superavain, oli kyseessä sitten ylimmän tason relaatio tai jokin alirelaatio.

YritysA

ono	onimi	Työntekijät
		tyno
10	tuotanto	14 25
20	markkinointi	59

YritysB

ono	onimi	Työntekijät
		tyno
10	tuotanto	332
20	markkinointi	445 442 449

YritysA∪YritysB

ono	onimi	Työntekijät
		tyno
10	tuotanto	14 25
20	markkinointi	59
10	tuotanto	332
20	markkinointi	445 442 449

Kuvio 3.F *Kahden eri yrityksen yhdistäminen unionilla. Lähtörelaatiot ovat PNF-muodossa, mutta tulosrelaatio ei ole.*

YritysA∪YritysB

ono	onimi	Työntekijät
		tyno
10	tuotanto	14 25 332
20	markkinointi	59 445 442 449

Kuvio 3.G *PNF-muodossa oleva NF2-relaatio, jonka tietosisältö on sama kuin edellisen esimerkin tulosrelaation.*

Määritelmä 3.C

PNF-muotoiseen relaatioon kohdistettu unnest operaatio tuottaa aina PNF-muotoisen relaation [RoKoSi88].

Määritelmä 3.D

Nest-operaation tulosrelaatio on PNF-muodossa jos PNF-muotoisen lähtörelaation pakkaamatta jäävät atomiset attribuutit määräävät yksikäsitteisesti pakkaamatta jäävät relaatioarvoiset attribuutit. Eli PNF-relaatioon $R(A_1,...,A_k,A_{k+1},...,A_n, B_1,...B_l,B_{l+1},...,B_m)$, jossa attribuutit $A_1,...,A_n$ ovat atomisia ja attribuutit $B_1,...,B_m$ ovat relaatioarvoisia, kohdistetun nest-operaation tulosrelaatio

$R_2 = \nu_{B0 = \{Ak+1,...An,Bl+1,...,Bm\}}$ (R) on PNF-muodossa, jos $A_1,...,A_k \rightarrow B_1,...,B_l$.

4. TRANSITIIVISEN PROSESSOINNIN LISÄÄMINEN RELAATIOMALLIIN

Perinteinen relaatiomalli ei tue millään tavalla rekursiivista määrittelyä, mikä on paha puute mallin ilmaisuvoimassa. Eräs useimmin tarvittu rekursiivinen määrittely liittyy transitiivisulkeuman läpikäymiseen aggregoiden, joka antaa vastauksen esimerkiksi seuraaviin tietotarpeisiin:

- mitä tarvikkeita tarvitaan tuotteen kokoonpanoon (tuotteen räjäytyskuva)
- mikä on lyhin reitti kaupunkien välillä (nopein, suurikapasiteettisin, halvin)
- ovatko CAD kuvan virtapiirin kaksi osaa yhteydessä toisiinsa
- anna henkilön kaikki jälkeläiset

4.1. TRANSITIIVINEN PROSESSOINTI OHJELMOINTI- JA KYSELYKIELISSÄ

Transitiivisulkeuman laskenta voidaan ohjelmointikielessä esittää kolmella vaihtoehtoisella tavalla: rekursiivisesti, iteratiivisesti tai erikoisoperaattorilla. Mannino ja Shapiro ovat vertailleet näitä erityyppisiä graafin läpikäynti algoritmeja eri tietokantaparadigmoissa [ManSha90]. Kuvassa kuvio 4.a on listattu eri esittämistapojen hyviä ja huonoja puolia Manninon ja Shapiron mukaan.

Metodi	Hyvät puolet	Huonot puolet
Erikois operaattori	Kyselyn formulointi helppoa, Algoritmin valinta optimoijalle	Ilmaisuvoima rajoittunut
Iteraatio	Helppo implementoida	Pitkä esitystapa, Algoritmin valinta käyttäjälle
Rekursio	Tiivis esitystapa	Optimoijan konvertoitava iteratiiviseksi tai tehoton, Algoritmin valinta käyttäjälle.

Kuvio 4.A *Eri tapoja hallita transitiivisulkeuma kyselykielissä.*

Lähestymistavaksi transitiiviseen prosessointiin olen valinnut erikoisoperaattorin käytön, eli transitiivisulkeuman prosessointi tapahtuu yhdellä korkean tason relaatioalgebraoperaatiolla ja vastaavasti yhdellä SQL-lauseella. Huonona puolena tässä lähestymistavassa on rajoittunut ilmaisuvoima, mutta operaattori on kuitenkin helppo saada vastaamaan yleisiin transitiivisiin kyselytarpeisiin. Erikoisoperaattorin käyttö on mielestäni ainoa mahdollinen tapa toteuttaa aggregoiva transitiivisulkeuman laskenta relaatiomallissa siten, että uusi ominaisuus saadaan integroitua relaatiomalliin muiden operaatioiden tavoin. Jo vaatimus siitä, että relaatiomallin kyselykielen tulee olla deklaratiivinen karsii pois mahdollisuuden käyttää iteraatiota tai rekursiota, koska niiden käyttö ei koskaan ole deklaratiivista vaan käyttäjän on hallittava iteraatio tai rekursio ja huolehdittava niiden lopetusehdoista. Lisäksi kaikki muut relaatioalgebran operaatiot ottavat syötteeksi yhden tai kaksi relaatiota ja antavat tuloksena yhden relaation, joten uuden operaation on oltava samantyyppinen, jotta laajennetussakin relaatioalgebrassa voidaan minkä hyvänsä operaation tuloste antaa syötteeksi mille hyvänsä toiselle operaatiolle.

Erikoisoperaattorin tärkeimmäksi eduksi iteraatioon ja rekursioon verrattuna asetan erikoisoperaattorin deklaratiivisuuden ja siten kyselyn formuloinnin helppouden. Tietokantakyselykielen tulee olla intuitiivisesti ymmärrettävä, jotta loppukäyttäjä

pystyy formuloimaan nopeasti 'ad hoc' kyselyitä. Iteraatiota käytettäessä vaaditaan käyttäjältä ohjelmointitaitoja ja oikeaa ratkaisua ei aina saada edes suoraviivaisesti ohjelmoiden vaan voidaan joutua käyttämään aika eksoottisia algoritmeja väärien vastauksien karsimiseksi. Rekursiota käytettäessä taas vaaditaan loppukäyttäjältä jopa logiikkaohjelmointitaitoa ja rekursiivisen ohjelman määritteleminen onkin kokemattomalle lähes mahdoton tehtävä.

Toinen vahva etu erikoisoperaattorilla on se, että optimoija valitsee algoritmin kyselyn suorittamiseen. Tällöin voidaan hyödyntää järjestelmässä olevia tilastotietoja kannan sen hetkisestä sisällöstä tehokkaan algoritmin valintaan ja taitamatonkin loppukäyttäjä saa aikaan tehokasta koodia. Iteraatiossa ja rekursiossa algoritmin valinta on loppukäyttäjän tehtävä. Transitiivisulkeuman laskevan erikoisoperaattorin lisääminen relaatiomalliin vaatii luonnollisesti optimointimenetelmien kehittämistä, jotta optimoija osaa tuottaa tehokasta koodia.

Rekursiivisella lähestymistavalla SQL-kielen ilmaisuvoiman kasvattamiseen on huomattavia hyviä puolia ja varsinkin rekursion hyvin hallitsevat logiikkaohjelmoijat kannattavat rekursion esittämistä SQL-kielessäkin samantyyppisillä rakenteilla kuin loogiset lausekkeet logiikkaohjelmoinnissa. Tämäntyyppistä esitystä käytetään SQL* kielessä [KoyCai93] ja SQL*/NR kielessä [NgQua95]. Tällä lähestymistavalla voidaan saavuttaa jopa Turingin koneen ilmaisuvoima ja esitystapa on periaatteessa deklaratiivinen. Käytännössä nykytekniikalla käyttäjän tulee kuitenkin tietää lauseiden suoritusjärjestys ja ymmärtää rekursio varsin hyvin. Logiikkaohjelmointia hallitsemattomalle tämä lähestymistapa on vaikeasti ymmärrettävä, eikä siten sovellu korkean tason kyselykieleksi.

Rekursion lisääminen malliin voitaisiin perustellusti tehdä myös näkymien (view) kautta, kuten Eder esittää [Eder90]. Tämä sopii hyvin tavalliseen relaatiomalliin, koska view-mekanismi on siinä ennestäänkin ainut tapa esittää johdettua tietoa. Tällöin näkymän määrittely on ohjelmoijan tehtävä ja loppukäyttäjä voi käyttää näkymää kuten mitä tahansa relaatiota. Mielestäni on suositeltavaa, että ammattilaiset esittävät usein tarvittavat 'rekursiiviset' relaatiot näkyminä loppukäyttäjille, koska yksinkertaistettukaan rekursion ymmärtäminen ei ole loppukäyttäjälle helppoa. Mielestäni ei ole kuitenkaan syytä rajoittaa rekursion käyttöä pelkästään näkymien yhteyteen, koska rekursiivisia "ad hoc" kyselytarpeitakin esiintyy. Transitiivisulkeuman laskennan tulee olla niin yksinkertainen operaatio, että kokeneempi loppukäyttäjä pystyy siitä selviytymään. Näkymäksi luotu transitiivisulkeumarelaatio voidaan myös materialisoida, mistä voi olla usein hyötyä, koska sulkeuman laskeminen voi olla hyvin raskas operaatio. Transitiivisulkeuma, jota tarvitaan usein, kannattaa joissain tapauksissa luoda materialisoiduksi näkymäksi.

4.2. TRANSITIIVISEN PROSESSOINNIN LISÄYS NF2-RELAATIOMALLIIN

Transitiivista käsittelyä on esitetty lisättäväksi relaatiomalliin ja sen kyselykieliin usealla eri tavalla muun muassa lähteissä [DarAgr93], [Agr87], [Eder90], [KoyCai93], [NieJär92/1], [NieJär92/2], [AhaYao93]. Tässä tutkimuksessa käytettävä transitiivisulkeuman laskentaoperaattori pohjautuu Rakesh Agrawalin esittämään relaatioalgebran lisäoperaatioon α (alpha) [Agr87], sekä Shaul Darin ja Rakesh Agrawalin sen pohjalta edelleen kehittämään relaatioalgebraan ja myös SQL-kieleen määriteltyyn CLOSURE-operaatioon [DarAgr93]. Molemmat operaatiot on määritelty

1NF-relaatiomalliin. Olen tehnyt operaation toimintaan ja esitykseen joitakin muutoksia sovittaessani sen NF2-malliin ja NSQL-kieleen saadakseni operaation hyödyntämään paremmin tämän tietomallin uusia ominaisuuksia. Jopa operaation ilmaisuvoima on kasvanut sovitettaessa se uuteen tietomalliin. Olen myös muuttanut operaation syntaksia helpommin ymmärrettäväksi.

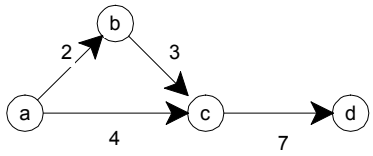
Määritän ensin rakenteen transitiivirelaatiolle, jollaiseksi kutsun relaatiota, josta transitiivisulkeuma voidaan tuottaa. Transitiivirelaatio on muotoa R(S, T, L), jossa attribuutteja S ja T kutsun sulkeuma-attribuuteiksi. Nämä attribuutit ovat pakollisia, sillä transitiivisulkeuma lasketaan niiden kesken. S ja T saavat arvonsa samasta arvojoukosta ja ne voivat olla myös attribuuttijoukkoja. L on attribuuttijoukko, jonka attribuutteja kutsun särmä-attribuuteiksi. Joukon L attribuutit ovat optionaalisia. Kuten kuvasta kuvio 4.b huomaamme, relaatio R(S,T,L) voidaan esittää graafina G, jossa jokainen relaation R rivi esittää graafin särmää pisteestä s pisteeseen t piirrettynä attribuuteista S ja T ja joukon L attribuutit esittävät särmän ominaisuuksia (esimerkiksi pituus, kapasiteetti...). Transitiivisulkeuman laskeminen tarkoittaa graafin kaikkien reittien etsimistä, joten se voidaan ymmärtää myös särmien piirtämisenä kaikkien pisteiden välille, joista pääsee kulkemaan toisiinsa muita särmiä pitkin.

Määrittelen seuraavaksi eräitä käsitteitä, joita transitiivisulkeuman prosessoinnissa ilmenee:

- E(s,t) = särmä pisteestä s pisteeseen t.
- P(u,v) = reitti pisteestä u pisteeseen v, muodostuu järjestetystä joukosta särmiä $\{E_k(s,t)\}$, jossa k tarkoittaa särmän järjestysnumeroa reitillä ja saa arvoja 1,...,n. Nyt voidaan merkitä $E_k.s$ ja $E_k.t$, jotka tarkoittavat reitin P järjestyksessä k:nen särmän attribuutteja s ja t. Tällöin $E_1.s = u$, $E_k.t = E_{k+1}.s$ ja $E_n.t = v$, eli edellisen särmän t on aina seuraavan särmän s.
- $\psi(u,v)$ = reittijoukko pisteestä u pisteeseen v, eli $\{P_k(u,v)\}$. Kahden pisteen välillä voi luonnollisesti olla useampia eri reittejä.

R

src	dest	dist
a	b	2
a	c	4
b	c	3
c	d	7



Kuvio 4.B *Transitiivirelaatio ja sen graafi esitys.*

Kuvan kuvio 4.b transitiivirelaatiossa särmät E(s,t) ovat suoraan tallennettuina lähtörelaatioon R. Pisteestä a pisteeseen d löytyy reitti P(a,d) = {(a,b,2),(a,c,4),(c,d,7)} ja toinen reitti P(a,d) = {(a,c,4),(c,d,7)}. Yhdessä nämä reitit muodostavat reittijoukon ψ(a,d) = {{(a,b,2),(a,c,4),(c,d,7)}, {(a,c,4),(c,d,7)}}.

Yksinkertaisimmillaan transitiivisulkeuman laskenta kohdistetaan relaatioon, jossa on vain attribuutit S ja T, eikä laskennalle aseteta mitään ehtoja. Tällöin saadaan tulokseksi relaatio, joka ilmaisee, minkä pisteiden välillä on olemassa reitti. Agrawalin esittämä alpha-operaatio (α) suorittaa tämän toiminnon esimerkin kuvio 4.b relaatiolle yksikertaisesti komennolla α(R). Tämän komennon tulosrelaatio on kuvassa kuvio 4.c, josta näemme suoraan, että a:sta pääsee pisteisiin b,c ja d, että b:stä pääsee pis-

teisiin c ja d, ja että c:stä pääsee vain pisteeseen d. Esimerkkimme lähtörelaatiossa oli sulkeuma-attribuuttien lisäksi myös joukon L attribuutti 'dist'. Sitä ei huomioida peruskyselyn tulosjoukossa, koska sitä ei käytetty kyselyssä.

α(R)

src	dest
a	b
a	c
a	d
b	c
b	d
c	d

Kuvio 4.C *laajennetun relaatioalgebralauseen α(R) tulos.*

Algoritmina esitettynä transitiivisulkeuma voidaan yksinkertaisimmillaan laskea seuraavanlaisella iteratiivisella algoritmilla, jossa jokainen iteraatiokierros lisää tulosrelaatioon polut, joiden pituus on yhtä pidempi, kuin edellisellä kierroksella löytyneillä poluilla. R_0 on lähtörelaatio, josta transitiivisulkeuma lasketaan ja R on tulosrelaatio johon transitiivisulkeumaa kerätään. Algoritmissa merkintä $R.a_2$ tarkoittaa liitoksen (∞) ensimmäisen operandirelaation (R) järjestyksessä toista attribuuttia. Vastaavasti merkintä $R_0.a_1$ tarkoittaa ristitulon toisen operandirelaation R_0 järjestyksessä ensimmäistä attribuuttia.

$R = R_0$
DO

$$R = R \cup \pi_{R.a_1, R_0.a_2}(R \infty_{R.a_2 = R_0.a_1} R_0)$$

WHILE "R changes"

DO-WHILE rakenteen sisällä oleva lauseke voidaan ilmaista myös lauseella:

$$<a,b> \in R \wedge <b,c> \in R_0 \Rightarrow <a,c> \in R$$

Usein halutaan tietää reitistä muutakin kuin että se on olemassa. Esimerkkirelaatiosta on luonnollista haluta tietää reitin kokonaispituus, ja tällöin tarvitaan aggregointiominaisuuksia transitiivisulkeumaa laskettaessa. Jos laskemme esimerkkirelaatiosta reitin särmien pituuksien summan saamme kuvan kuvio 4.d tulosrelaation. Mukaan tulee nyt useita eri reittejä kahden pisteen välillä, koska aggregointitieto eroaa reiteillä, vaikka päätepisteet ovat samat.

α(R)

src	dest	SUM(dist)
a	b	2
a	c	4
a	c	5
a	d	11
a	d	12
b	c	3
b	d	10
c	d	7

Kuvio 4.D *Transitiivisulkeuma aggregoiden reitin kokonaispituus.*

Havainnollistan seuraavaksi transitiivisulkeumaoperaation käyttöä käytännön esimerkillä: oletetaan että meillä on kuvan kuvio 4.e relaatio, jossa on tallennettuna tiet eri kaupunkien välillä ja niiden pituudet. Haluamme tietää lyhimmän tien Tampereel-

ta Turkuun. Tietoa ei suoraan taulusta löydy, koska siellä ovat vain vierekkäiset suoran tieosuuden yhdistämät kaupungit. Esimerkkitauluumme kohdistetulla aggregoivalla transitiivisulkeuman laskentaoperaatiolla saamme vastaukseksi yhden rivin, joka ilmaisee lyhimmän reitin pituuden Tampereelta Turkuun (kuvio 4.f).

Tiestö

lähtö	määräp	km
Tampere	Nokia	20
Tampere	Valkeakoski	40
Valkeakoski	Hämeenlinna	40
Vantaa	Helsinki	20
...	...	...

Kuvio 4.E *Relaatio teistä eri kaupunkien välillä.*

Answer

lähtö	määräp	minpit
Tampere	Turku	170km

Kuvio 4.F *Aggregoivan transitiivisulkeuman tulos, joka ilmaisee lyhimmän reitin Tampereelta Turkuun.*

Transitiivisulkeumaa laskettaessa täytyy tulosrelaation jokaista riviä kohden luoda järjestetty joukko särmistä, joiden kautta reitti kuljetaan. Tätä tietoa tarvitaan sulkeumaa generoitaessa ja aggregointitietoa laskettaessa. Esitetyissä transitiivisulkeuman laskentaoperaatioissa tätä joukkoa ei yleensä ole voinut käyttää sulkeuman generoinnin jälkeen, koska sitä ei voi 1NF-relaatiomallissa tallentaa. NF2-mallin ominaisuuksia voidaan hyödyntää transitiivisulkeumaoperaatiossa ja tallettaa reitin särmäjoukko alirelaatioksi 'Path' (katso kuva kuvio 4.g). Polku on järjestetty joukko särmiä, mutta koska en halua lisätä relaatiomallin rakenteellisten tietotyyppien määrää, lisään polun särmille järjestysnumeron attribuuttiin 'ano' (arc number), jonka järjestelmä generoi sulkeumaa laskiessaan. Näin saadaan särmien järjestysinformaatio tallennettua ja relaatiomallin rakenteellisten tietotyyppien määrää ei tarvitse kasvattaa. 'Path'-alirelaatioon tallennetaan oletusarvoisesti kaikkien reitille kuuluvien särmien kaikki attribuutit. Koska NF2-mallissa voidaan nyt polun särmäjoukkoa tutkia jälkikäteen ja vertailla eri polkujen särmäjoukkoja toisiinsa, saadaan operaatiosta ilmaisuvoimaisempi kuin 1NF-mallissa. Esimerkiksi risteäviä reittejä ei operaatiolla voi 1NF-mallissa etsiä, koska reitin generoinnin jälkeen ei sen rakentamiseen käytettyjä särmiä enää voi tutkia. Nyt on mahdollista etsiä kaikki reitit esimerkiksi Helsingin ja Tampereen välillä ja kaikki reitit Turun ja Lahden välillä ja sen jälkeen tutkia mitkä reitit risteävät ja millä paikkakunnilla.

NF2-relaatiomallissa PNF-normaalimuodon vaatimus aiheuttaa ongelmia transitiivisulkeuman tulosrelaation kanssa, koska kahden pisteen välillä voi olla kaksi eri reittiä (riviä), joiden atomisten attribuuttien arvot ovat samat. Kuitenkaan reitit eivät ole samat, jos ne on kuljettu eri särmien kautta, minkä näkee 'Path'-alirelaation särmälistasta. Tämän vuoksi lisään sulkeuman tulosrelaatioon attribuutin 'pno' (path number), joka erottaa juoksevalla numerosarjalla toisistaan eri reitit samojen pisteiden välillä, vaikka kaikki atomiset attribuutit olisivat samoja. NF2-mallissa on mahdollista käyttää tulosrelaatiota 'Path'-listoineen jatkoprosessointiin, joten eri reitit täytyy pystyä tallentamaan. NF2-malli perusmuodossaan kykeneekin tallentamaan eri reitit ilman mitään lisäyksiä, koska relaation rivit ovat eriävät, jos niiden kaikkien attribuuttien arvot, atomisten ja relaatioarvoisten, eivät ole samat. PNF-

normaalimuoto kuitenkin vaatii, että atomisten attribuuttien on kyettävä yksilöimään rivi.

TC(R)

src	dest	tot_dist	*pno*	Path			
				ano	src	dest	dist
a	b	2	1	1	a	b	2
a	c	4	1	1	a	c	4
a	c	5	2	1 2	a b	b c	2 3
a	d	11	1	1 2	a c	c d	4 7
a	d	12	2	1 2 3	a b c	b c d	2 3 7
b	c	3	1	1	b	c	3
b	d	10	1	1 2	b c	c d	3 7
c	d	7	1	1	c	d	7

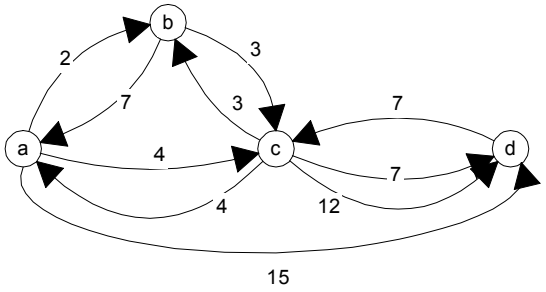
Kuvio 4.G *Transitiivisulkeumarelaatio, jossa laskettu polkujen lisäksi polun pituus.*

Syklejä sisältävässä graafissa on mahdollista, että päädytään päättymättömään prosessointiin. Siksi on huolehdittava, että prosessointi päättyy myös syklisessä verkossa. Estämme proseduurin joutumisen päättymättömään prosessointiin vaatimuksella, että saman särmän kautta voi kulkea vain kerran, eli reittiä luotaessa ei reitin 'Path'-alirelaatioon voi lisätä samaa riviä (särmää), joka siellä jo on.

Syklisessä graafissa (kuvio 4.h) voi saman solmun kautta olla mahdollista kulkea eri särmiä pitkin. Yleensä ollaan kuitenkin kiinnostuneista vain suorista reiteistä, eikä silmukoita tekevistä, koska silmukoita sisältävät reitit eivät yleensä ole nopeimpia, suurikapasiteettisimpia yms. Esimerkiksi reiteistä a:sta c:hen on suoraan särmän a-c kautta kulkeva reitti varmasti lyhyempi, kuin silmukan sisältävä reitti a-b-a-c. Silmukoiden salliminen saattaisi lisäksi johtaa odottamattomaan tulokseen, koska huonoimpiakin reittejä etsittäessä tarkoitetaan yleensä huonointa reittiä, joka ei sisällä syklejä. Esimerkiksi etsittäessä huonokapasiteettisinta reittiä kahden pisteen välillä ei tarkoitettane, että lähtöpisteen kautta voi kulkea useita kertoja. Kaikkien silmukoiden laskeminen syklisessä graafissa saattaa moninkertaistaa transitiivisulkeuman laskennan raskauden, eikä silmukoita sisältävistä reiteistä yleensä olla kiinnostuneita joten estämme silmukoiden läpikäymisen syklisessä graafissa oletusarvoisesti. Tämä tapahtuu estämällä saman solmun läpi kulkeminen kahteen kertaan. Joskus kuitenkin on tarvetta sallia syklien läpikäyminen, esimerkiksi poikkeaminen jollakin välipaikkakunnalla suorimmalta reitiltä. Tämän käyttäjä voi erikseen sallia kyselyssä (katso kappale 4.2.1). Tällöinkään ei saman särmän kautta voi kulkea kuin kerran, joten päättymättömään prosessointiin ei jouduta.

R

src	dest	dist
a	b	2
b	a	7
a	c	4
c	a	4
a	d	15
b	c	3
c	b	3
c	d	7
d	c	7
c	d	12



Kuvio 4.H *Transitiivirelaatio ja sen graafiesitys syklisestä verkosta.*

NF2-mallissa on mahdollista transitiivisulkeuman tulosrelaatiota uudelleenmuotoilemalla ryhmitellä reittejä ja siten lisätä tuloksen havainnollisuutta. Haluan antaa käyttäjälle mahdollisuuden vaikuttaa tulosrelaation rakenteeseen jo sulkeumaa tuotettaessa, mutta en toisaalta halua monimutkaistaa transitiivisulkeumaoperaatiota. Ratkaisuksi esitän, että tulosrelaation rakenne määräytyy lähtörelaation rakenteen mukaan, jolloin käyttäjä näkee lähtörelaation rakenteesta millaisen tulosrelaation hän tulee saamaan. Järjestelmä lisää automaattisesti ainoastaan 'Path'-alirelaation alimmalle upotustasolle. Katso kuvan kuvio 4.i lähtörelaatio ja kuvan kuvio 4.j vastaava tulosrelaatio. Käyttäjä voi nyt vaikuttaa tulosrelaation rakenteeseen muotoilemalla transitiivisulkeumaoperaatiolle syötteeksi annettavan lähtörelaation rakenteen haluamakseen ennen transitiivisulkeumaoperaatiota. Tällä tavalla transitiivisulkeumaoperaation syntaksiin ei tarvita lisäominaisuuksia, vaan relaatiokaavion muotoiluun voidaan käyttää ennestään tuttuja relaatiokaavion uudelleenmuotoilukomentoja.

R

src	Dests	
	dest	Properties
		dist
a	b	2
	c	4
b	c	3
c	d	7

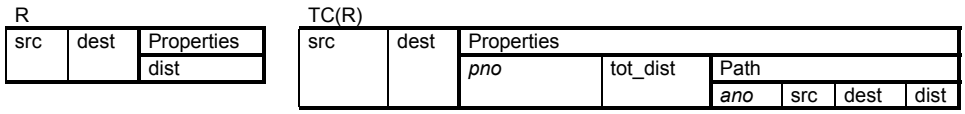
Kuvio 4.I *Sulkeuman lähtörelaatio NF2-muodossa.*

TC(R)

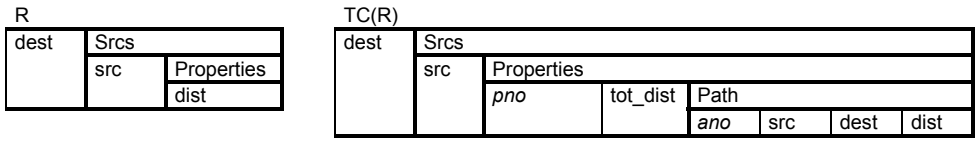
src	Dests						
	dest	Properties					
		pno	tot_dist	Path			
				ano	src	dest	dist
a	b	1	2	1	a	b	2
	c	1	4	1	a	c	4
		2	5	1 2	a b	b c	2 3
	d	1	11	1 2	a c	c d	4 7
		2	12	1 2 3	a b c	b c d	2 3 7
b	c	1	3	1	b	c	3
	d	1	10	1 2	b c	c d	3 7
c	d	1	7	1	c	d	7

Kuvio 4.J *Sulkeuman tulosrelaatio, kun lähtörelaatio on NF2-muodossa.*

Samalla tavalla muotoutuu tulosrelaatio erilaisten lähtörelaatiokaavioiden mukaan seuraavasti.



Kuvio 4.K *Tulosrelaation rakenteen muotoutuminen lähtörelaation mukaan.*



Kuvio 4.L *Tulosrelaation rakenteen muotoutuminen lähtörelaation mukaan.*

Kuten kuviosta huomaamme sulkeuma attribuuttien järjestys voidaan vaihtaa, jolloin nähdään suoraan mistä lähtöpisteistä tiettyyn päätepisteeseen päästään. Aiemmin on aina tutkittu mihin päätepisteisiin jostakin lähtöpisteestä päästään.

4.2.1. Transitiivisulkeuma operaattori PATHS

Koska NF2-mallissa transitiivisulkeuman tulosrelaation 'Path'-alirelaatiota voidaan käsitellä aivan normaaleilla laajennetun relaatioalgebran lauseilla, eivät transitiivisulkeumaan liittyvät valinta- ja aggregointi- operaatiot poikkea tavasta tehdä vastaavat operaatiot mille hyvänsä NF2-relaatiolle. Poikkeuksena on kuitenkin polun peräkkäisten särmien mukaan valinta, joka vaatii polun särmien käsittelyä järjestettynä joukkona. Tämä operaatio onkin upotettu relaatioalgebran transitiivisulkeuman laskevan operaation PATHS sisään. Esitän kuitenkin transitiivisulkeuman laskentaoperaation kokonaisuutena, jossa kaikki valinta ja aggregointi suoritetaan operaation yhteydessä, jolloin operaation ominaisuuksista ja ilmaisuvoimasta saa hyvän käsityksen.

Yleinen transitiivisulkeuman laskentaoperaatio on muotoa:

$$TC = \pi_A \, \delta AGG_{Y,Z} \, \sigma CON_{X,\xi} \, PATHS_{\lambda,\, LOOPS} \, (R[S,T,L])$$

R on lähtörelaatio, jota kutsun transitiivirelaatioksi. PATHS on polkujen määrittelyoperaatio, eli varsinainen transitiivisulkeuman laskeva operaatio. Lisäksi käsittelen tässä samassa yhteydessä transitiivisulkeuman aggregoinnin ja tuloksen muotoilun, joihin tarvitaan seuraavia operaatioita. CON on polun särmien aggregointioperaatio. AGG on polkujoukkojen aggregointioperaatio. π on projektio-operaatio ja λ, ξ, σ, ja δ ovat valintaoperaatioita särmien, polkujen ja polkujoukkojen mukaan. LOOPS määrittelee syklisyyden käsittelyä. S ja T ovat sulkeuma-attribuutit tai attribuuttijoukot, L on optionaalisten särmä-attribuuttien joukko, X on polulta aggregoitava attribuutti, Y on polkujoukosta aggregoitava attribuutti, Z on polkujoukkojen ryhmittelyattribuutti ja A on projisoitava-attribuuttijoukko. Pakollisia osia transitiivisulkeumaa laskettaessa ovat operaattori PATHS ja relaation R attribuutit S ja T.

Käsittelen seuraavaksi eri operaatioiden toiminnan. Esimerkkirelaationa käytän kuvan kuvio 4.m transitiivirelaatiota.

Lennot

lähtöp	määräp	lasema	masema	lähtöaika	tuloaika	hinta	yhtiö
Tampere	Helsinki	Pirkkala	Vantaa	7:00	7:20	500	Finnair
Helsinki	Lontoo	Vantaa	Gatwick	8:10	10:50	4200	Finnair
Lontoo	Helsinki	Gatwick	Vantaa	11:50	14:20	3000	Finnair
Helsinki	Lontoo	Vantaa	Stansted	15:55	18:20	4400	BA
Lontoo	Newyork	Standsted	Newyork	20:00	6:00	6500	BA
...	...	...	...	...	...	...	...

Kuvio 4.M *Lentoreitit sisältävä relaatio.*

- ***PATHS***-operaattori suorittaa transitiivisulkeuman laskennan, eli luo kaikki reitit sisältävän reittijoukon ψ sulkeumapredikaatissa λ kerrottujen sulkeuma-attribuuttien perusteella. Esimerkkirelaatiosta voimme laskea yleisen transitiivisulkeuman lähtö- ja määräpaikkojen välillä, jolloin tulos sisältää kaikki lentoreitit eri lentokenttien välillä.

- *Sulkeumapredikaatti* λ määrittää miten särmistä luodaan polkuja ja suorittaa valintaa polkujen mukaan. Sulkeumapredikaatti sisältää kahdenlaisia ehtoja, joista ensimmäinen tyyppi kohdistuu reitin peräkkäisiin särmiin. Yhden näistä ehdoista on oltava *primäärisulkeumaehto*, joka on yhtäsuuruusehto sulkeuma-attribuuttien välillä, ja ilmaisee minkä attribuuttien välillä sulkeuma lasketaan. Mikäli transitiiviattribuutit koostuvat attribuuttijoukoista S ja T, esitetään tässä lauseessa yhtäsuuruusehto jokaista joukkojen S ja T attribuuttiparia kohden. Esimerkkirelaatiossa primäärisulkeumaehto voisi olla:

  määräp = NEXT lähtöp

Tällöin kaikki lentoreitit eri kaupunkien välillä hakeva transitiivikysely on:

$$\mathrm{PATHS}_{\text{määräp = NEXT lähtöp}}(\text{Lennot})$$

eli reitin seuraavan lennon lähtöasema on edellisen lennon määräasema. Lisäksi voidaan esittää vapaaehtoisia ehtoja peräkkäisille reiteille kuten:

$$\mathrm{PATHS}_{(\text{määräp = NEXT lähtöp}) \text{ AND } (\text{tuloaika < NEXT lähtöaika - 1})}(\text{Lennot})$$

eli edellisen lennon täytyy saapua määräasemalle vähintään tuntia ennen jatkolennon lähtöaikaa.

Toinen vapaaehtoinen ehtotyyppi sulkeumapredikaatissa rajoittaa sulkeumaan osallistuvien reittien joukkoa. Voisimme esimerkiksi vaatia, että reitti ei kulje Lontoon kautta ehdolla 'määräp ≠ Lontoo', jolloin kysely saa muodon:

$$\mathrm{PATHS}_{(\text{määräp = NEXT lähtöp}) \text{ AND } (\text{tuloaika < NEXT lähtöaika - 1}) \text{ AND } (\text{määräp} \neq \text{Lontoo})}(\text{Lennot})$$

- *Syklisyydenmäärittelyvakio LOOPS* määrittää syklisyyden käsittelyä verkossa. Mikäli vakiota LOOPS ei anneta, reitillä ei syklisessäkään graafissa sallita saman solmun läpi kulkemista kahteen kertaan. Tämä tarkistetaan primäärisulkeumaehdossa määriteltyjen attribuuttien perusteella siten, että merkintää '= NEXT' seuraavalla attribuutilla voi olla sama arvo vain kerran ydellä polulla. Mikäli vakio

LOOPS annetaan, sallitaan saman solmun läpi kulkeminen useita kertoja eri särmiä pitkin, mutta samaa särmää ei silti voi kulkea kahteen kertaan. Ilman LOOPS-vakion määrittelyä on transitiivisulkeuman laskenta syklisessä verkossa huomattavasti tehokkaampaa, kuin LOOPS-vakion kanssa. Jos on tarvetta sallia saman solmun läpi kulkeminen useaan kertaan määritellään LOOPS-vakio seuraavasti:

$PATHS_{määräp = NEXT\ lähtöp,\ LOOPS}$ (Lennot)

- ***CON***-operaattori aggregoi erikseen jokaiselle polulle tietoa polun särmistä polkujoukolle ψ. CON on uusi NF2-mallin aggregointi-operaatio, joka kohdistetaan relaatioon ja saadaan vastaukseksi yksi arvo. CON-operaatioita voidaan suorittaa useita peräkkäin $CON_1,...,CON_k$, joista jokainen CON_i on kiinnitetty attribuuttiin $Xi \in L$ ja tuottaa uuden aggregoidun attribuutin PL_i. Xi ilmaistaan muodossa '$R_i.L_i$', jossa R_i ilmaisee alirelaation nimen ja L_i attribuutin, jonka arvo aggregoidaan. Transitiivisulkeumaoperaation yhteydessä R_i on alirelaatio 'Path'. Laajennetussa relaatiomallissa 'path' on varattu sana, ja se viittaa PATHS-operaation yhteydessä automaattisesti luotuun 'Path'-relaatioon. Tällä aggregointioperaatiolla voimme laskea esimerkiksi lennon kokonaishinnan jokaiselle lentoreitille P_j seuraavasti:

  yhthinta := $SUM_{path.hinta}$ $PATHS_{(määräp = NEXT\ lähtöp)}$(Lennot)

- *Särmänvalintapredikaatilla* ξ voidaan määrätä, että CON_i suoritetaan vain niille särmille, joille valinta ehto ξ_i on totta. Esimerkiksi voimme rajoittaa lennon hinnan laskennan tietyn lentoyhtiön reitteihin:

  yhtfinnair := $SUM_{path.hinta,\ yhtiö = Finnair}$ $PATHS_{(määräp = NEXT\ lähtöp)}$(Lennot)

- *Reitinvalintaoperaatiolla* σ valitaan osajoukko ψ' kaikkien polkujen joukosta ψ. Valintaoperaatio voidaan kohdistaa sulkeuma-attribuutteihin ja polun aggregoituihin attribuutteihin. Voidaan esimerkiksi valita reitit Tampereelta Lontooseen, joiden yhteishinta on alle 4000 mk seuraavasti:

  $\sigma_{lähtöp = Tampere,\ määräp = Lontoo,\ yhthinta < 4000}$
  yhthinta := $SUM_{path.hinta}$ $PATHS_{(määräp = NEXT\ lähtöp)}$(Lennot)

- ***AGG***-operaattori aggregoi tietoa polkujoukoille joukossa ψ'. Operaatiot $AGG_1,...,AGG_m$ tuottavat jokainen yhden arvon joukosta arvoja. Operaatio AGG_i kohdistetaan attribuuttiin $Y_i \in \{S, T, PL\}$ ja se luo uuden aggregoidun attribuutin PSL_i, eli tämä on perinteinen aggregointi-operaatio, joka laskee NF1-relaation jostakin attribuutista aggregoidun arvon uuteen attribuuttiin. Transitiivisulkeuman yhteydessä AGG operaation huomionarvoinen piirre on, että sillä voidaan aggregoida polkujen attribuutteja, jotka on ensin aggregoitu särmien attribuuteista. Voimme esimerkiksi etsiä halvimman lennon Tampereelta Lontooseen:

  minhinta := $MIN_{yhthinta}$ $\sigma_{lähtöp = Tampere,\ määräp = Lontoo}$
  yhthinta := $SUM_{path.hinta}$ $PATHS_{(määräp = NEXT\ lähtöp)}$(Lennot)

- *Z on ryhmittelyattribuutti*, joka voidaan haluttaessa liittää AGG-operaattoriin. Tällöin polut ryhmitellään ensin attribuutin Z_i arvojen perusteella ja aggregointi suoritetaan erikseen jokaiselle ryhmälle. Voimme esimerkiksi laskea minimihinnan erikseen eri Lontoon kentille:

  $\text{minhinta} := \text{MIN}_{\text{yhthinta, masema}}\, \sigma_{\text{lähtöp = Helsinki, määräp = Lontoo, yhthinta < 4000}}$
  $\text{yhthinta} := \text{SUM}_{\text{path.hinta}}\, \text{PATHS}_{(\text{määräp = NEXT lähtöp})}(\text{Lennot})$

- ***Polkujoukkojenvalintaoperaattorilla*** δ voimme valita osajoukon ψ'' joukon ψ' poluista silloin kun attribuutilla Z on operaation AGG yhteydessä ryhmitelty joukkoja. Voimme esimerkiksi valita minimihintaiset reitit vain niille Lontoon kentille, joille minimihinta alittaa 3000 mk kun olemme ensin etsineet lennot Tampereelta Lontooseen, ryhmitelleet ne kenttien mukaan ja laskeneet minimihinnan jokaiselle kentälle:

  $\delta_{\text{minhinta < 3000}}\, \text{minhinta} := \text{MIN}_{\text{yhthinta, masema}}\, \sigma_{\text{lähtöp = Tampere, määräp = Lontoo}}$
  $\text{yhthinta} := \text{SUM}_{\text{path.hinta}}\, \text{PATHS}_{(\text{määräp = NEXT lähtöp})}(\text{Lennot})$

- Projektio-operaattorilla π voimme rajoittaa polkujoukkojen ψ'' attribuutit attribuuttijoukossa A lueteltuihin. Voisimme esimerkiksi ottaa tulokseen mukaan vain määränpääkentän ja minimihinnan:

  $\pi_{\text{masema, minhinta}}$
  $\delta_{\text{minhinta < 3000}}\, \text{minhinta} := \text{MIN}_{\text{yhthinta, masema}}\, \sigma_{\text{lähtöp = Tampere, määräp = Lontoo}}$
  $\text{yhthinta} := \text{SUM}_{\text{path.hinta}}\, \text{PATHS}_{(\text{määräp = NEXT lähtöp})}(\text{Lennot})$

Seuraavalla esimerkkikyselyllä saadaan aikaan kuvan kuvio 4.e transitiivirelaatiosta kuvan kuvio 4.f tulosrelaatio.

Esimerkki 4.A

$\pi_{\text{lähtö, määräp, minpit}}\, \text{minpit} := \text{MIN}_{\text{yhtpit}}\, \sigma_{\text{lähtöp = Tampere, määräp = Turku}}\, \text{yhtpit} := \text{SUM}_{\text{Path.km}}$
$\text{PATHS}_{\text{määräp = lähtöp}}(\text{Tiestö})$

Tässä kappaleessa olen keskittynyt transitiivisulkeumaoperaation syntaksin ja semantiikan esittelyyn, operaation erilaisia käyttötapoja ja esimerkkikyselyitä on esitetty lisää luvussa 5.

4.3. TRANSITIIVI SULKEUMAN TEHOKAS TOTEUTUS

Yleisen transitiivisulkeuman tuottaminen saattaa olla hyvinkin raskas operaatio. Lisäämällä aggregointi ja kuljettujen särmien tallentaminen sen kuormittavuus kasvaa entisestään. Transitiivisulkeuman tehokkaasta toteuttamisesta on tehty runsaasti tutkimuksia. Tehokkaammilla algoritmeilla saadaan huomattavasti tehostettua sulkeuman laskentatehokkuutta. Lisäksi optimoijan ei tarvitse yleensä laskea koko transitiivisulkeumaa, vaan kyselyä tutkimalla voidaan käyttää eritilanteissa erilaisia algoritmeja ja tuottaa vain tarvittava osajoukko sulkeumasta. Eräs mahdollisuus tehostaa transitiivisulkeuman tuottamista on materialisoida se, jolloin laskentakuorma siirtyy kyselyistä tietokannan päivittämisen yhteyteen. Erilaisia tehokkaita transitiivisul-

keuman ja sen osajoukkojen toteutuksia tutkitaan muun muassa seuraavissa artikkeleissa [AgDaJa90], [CrNo89], [DarAgJa91], [GuhYu94], [HanLu89], [IoRaWi93], [KaIoCa92], [LuLeeHa94], [RoHeDaMa86], [Teu96]. Rekursion materialisointia tutkitaan esimerkiksi artikkeleissa [GuhYu92], [Jag90].

Vaikka 'Path'-relaatio esitetään NF2-mallissa relaationa, ei sitä tarvitse sulkeuman prosessoinnin aikana käsitellä relaationa, vaan reittiä on huomattavasti tehokkaampaa ylläpitää prosessoinnin aikana järjestettynä joukkona ja muuntaa se vasta lopuksi relaatioksi. Oman lisänsä NF2-mallissa laskenta- ja tilatehokkuuteen tuo 'Path'-listan tuominen käyttäjälle NF2-relaationa, koska tämä lisää tulosrelaation kokoa merkittävästi. Transitiivirelaatio (lähtörelaatio), voi sisältää paljonkin tarpeettomia attribuutteja ja jopa alirelaatioita, jolloin 'Path'-relaation koko voi kasvaa eksponentiaalisesti. Siksi NF2-mallissa on hyvä antaa käyttäjälle mahdollisuus päättää, haluaako hän 'Path'-relaation tuotettavaksi, ja jos haluaa, niin mitkä attribuutit 'Path'-relaatioon tulisi tallentaa. Mielenkiintoinen jatkotutkimuskohde transitiivisulkeuman tuottamisen tehokkuudesta NF2-mallissa on tutkia, milloin 'Path'-alirelaatio kannattaa tuottaa kaikkine attribuutteineen, ja milloin voitaisiin ottaa mukaan vain avainattribuutit, joiden avulla tarvittaessa löydetään kaikki reittiin liittyvä tieto.

5. NSQL - LAAJENNETTU SQL-KIELI

NF2-mallille on tehty joitakin ehdotuksia strukturoiduksi kyselykieleksi. Roth, Korth ja Batory ovat esittäneet 1987 selkeän kyselykielen SQL/NF [RoKoBa87], joka perii rakenteensa varsin suoraviivaisesti ISO:n standardoimasta SQL-kielestä. SQL/NF-kieleen on lisätty NF2-relaatioiden käsittely sisäkkäisillä kyselyillä ja relaatiokaavion uudelleenmuotoiluoperaatioilla, sekä korjattu joitakin SQL-kielen virheellisyyksiä. Huomattavasti suoraviivaisemman tavan kyselyiden suorittamiseen NF2-tietokannasta ovat esittäneet Niemi ja Järvelin 1993 [NieJär93], jotka sallivat nest- ja unnest-operaatioiden käytön suoraan yli useiden alirelaatiotasojen ja vielä täysin käyttäjältä piilotetusti. Mielenkiintoisen erikoisalan kyselykielen TQL ovat esittäneet Sack-Davis, Kent ja Ramamohanarao tekstitietokannoille luotuun Atlas-hybriditietokantajärjestelmäänsä [SaKeRa95].

Joitakin transitiivisulkeuman laskentaan kykeneviä 1NF-mallin kieliä on myös kehitetty. SQL-kieleen parhaiten sopiva ratkaisu on Darin ja Agrawalin 1993 esittämä SQL-kielen lisäoperaatio CLOSURE [DarAgr93]. Tämä erikoisoperaattori ei juuri vaadi rekursion ymmärtämistä ja antaa tuloksenaan relaation, joten se sopii hyvin SQL-kieleen. Koymen ja Cai ovat esittäneet 1993 SQL*-kielen [KoyCai93], joka on hyvin ilmaisuvoimainen, mutta vaatii käyttäjältä rekursion ymmärtämistä ja on siten vaikeaselkoisempi. Operaatio-orientautuneen ratkaisun transitiivisulkeuman laskentaan antavat Niemi ja Järvelin 1991 [NieJär92/2], jotka tarjoavat käyttäjälle valmiin joukon yleisesti tarvittavia transitiivisulkeumaan liittyviä operaatioita. Ederin esityksessä [Eder90] transitiivisulkeuma esitetään view-mekanismin yhteydessä, mutta se ei sisällä kunnon aggregointia. Myös Ahadin ja Yaon esittämä RQL-kieli kykenee rekursion esittämiseen [AhaYao93]. Salmisen, Järvelinin ja Niemen esittämä CQL-kieli [SalJärNie95] on QBE-tyyppinen kieli, joka käsittelee is-a suhteita transitiivisulkeuman avulla käyttäjälle näkymättömästi

NF2-mallia ja transitiivisulkeuman laskentakyvyn yhdistäviä kyselykieliä ei juuri ole olemassa. Ng ja Quaraeen ovat yhdistäneet nämä ominaisuudet SQL*/NR -kielessä [NgQua95], jossa on yhdistetty kielien SQL/NF ja SQL* ominaisuudet.

Olemassa olevat NF2-mallin ja aggregoivan transitiivisulkeuman yhdistävät kielet eivät mielestäni ole tarpeeksi intuitiivisia ja helppokäyttöisiä, joten olen luonut uuden kyselykielen NSQL, joka on intuitiivisesti ymmärrettävä ja suorakäyttöinen kieli. Myös relaatiokaavion muunnosoperaatiot tapahtuvat NSQL-kielessä hyvin suoraviivaisesti ja helposti. Yleensä relaatiokaavionmuunnosoperaatiot kyselykielissä ovat hyvin vaikeasti hahmotettavissa poislukien Niemen ja Järvelinin hyvin suoraviivainen esitys [NieJär93]. Ilmaisuvoimaltaan NSQL vastaa SQL-kieltä lisättynä aggregoivalla transitiivisulkeumaoperaatiolla ja NF2-mallin ominaisuuksilla. NSQL-kieleen on sisällytetty parhaita puolia kolmesta edellämainitusta tutkimuksesta [RoKoBa87], [NieJär93], ja [DarAgr93]. NSQL-kieli on kuitenkin uusi itsenäinen kyselykieli, jonka esitystapa eroaa täysin edellämainituista esityksistä. NSQL-kielessä on myös piirteitä, joita ei löydy mistään edellämainituista esityksistä.

5.1. NSQL-KIELEN YLEISISTÄ OMINAISUUKSISTA

NSQL-kieli kykenee perinteisen SQL:n ominaisuuksien lisäksi NF2-relaatioiden käsittelyyn ja rakenteen uudelleenmuotoiluun, sekä transitiivisulkeuman tuottamiseen aggregointitietoa laskien. NSQL-kielen kehittelyssä lähtökohtana on ollut, että NF2-mallia käytettäessä on pystyttävä osoittamaan suoraviivaisesti ja intuitiivisesti syvälläkin alirelaatiohierarkiassa sijaitsevaan attribuuttiin. Tämä tapahtuu pistenotaatiolla, josta esimerkkinä seuraava kysely esimerkkirelaatiosta kuvio 3.a

Esimerkki 5.A. Anna kaikki päivämäärät, joina työntekijät ovat suorittaneet kursseja.

```
Yritys.Työntekijät.Kurssit.pvm
```

Answer

pvm
2.3.81
5.8.91
1.9.90
4.5.91
2.2.95
4.4.97
7.9.95

Kuvio 5.A *Vastausrelaatio kyselyyn* `Yritys.Työntekijät.Kurssit.pvm`

Tulosrelaatiosta kuvio 5.a näemme, että saimme vastaukseksi joukon päivämääriä. Lähtörelaation rakenne ei kuitenkaan ollut näin yksinkertainen, joten pistenotaatio ei tehnyt pelkästään projektiota vaan se sisälsi myös relaatiokaavion uudelleenmuotoiluoperaation unnest. Tässä yksinkertaisessa esimerkissä suoritettiin relaatiokaavion uudelleenmuotoilua jo useaan kertaan täysin intuitiivisesti osoittaen pistenotaatiolla relaatiokaavion sisältä attribuutti jonka arvot halutaan tulokseen.

Vastaava kysely perinteisen SQL:n tyyppisellä ei rekursiiviseen NF2-algebraan perustuvalla NF2-kielellä olisi:

```
SELECT pvm
FROM   UNNEST (UNNEST (Yritys)
                ON Työntekijät)
       ON Kurssit;
```

Tämän tutkimuksen tietokantamallissa oletetaan yhden NF2-relaation kaikkien attribuuttien nimien olevan yksikäsitteisiä, vaikka ne sijaitsisivat eri alirelaatioissa. Siten viittaaminen mihin hyvänsä attribuuttiin NF2-relaation eri hierarkia tasoilla voitaisiin periaatteessa tehdä antamalla pelkästään attribuutin nimi. Attribuutin nimi on käytännössä kuitenkin lyhenne, josta suurissa tietokannoissa on hyvin vaikea päätellä, mitä tietoa attribuutti sisältää arvoinaan. Kun kyselyssä kirjoitetaan alirelaatiohierarkia näkyviin saadaan automaattisesti informaatiota attribuutin merkityksestä, koska relaatiokaavion alirelaatioiden nimet yhdessä attribuutin nimen kanssa kertovat tarkalleen, mitä tietoa attribuutissa sijaitsee. Esimerkki tapauksessa haetaan yrityksen työntekijöiden käymien kurssien päivämäärät, mikä ilmenee suoraan kyselystä. Toisin on suoran viittauksen 'pvm', 'kpvm' tai 'ttkpvm' kanssa, jollaisia tietokannoissa attribuuttien niminä yleisesti käytetään. Tiedonhallintajärjestelmän kyselyeditori voi kyllä automaattisesti täydentää relaatiokaaviopolut attribuutin nimen perusteella, jol-

loin kyselyä kirjoitettaessa voi kirjoittaa lyhyesti päärelaation ja attribuutin nimen ja järjestelmä täydentää oikean viittausketjun.

Perinteisen 1NF SQL-kielen kyselyssä, jonka rakenne on muotoa:

```
SELECT <attribute_list>
FROM <relation_list>
WHERE <condition>
```

Relaatioiden nimet <relation_list> ja attribuuttilista <attribute_list> erotetaan toisistaan select- ja from-lauseisiin. NF2-mallissa myös attribuutti voi olla relaatio, joten relaation nimien ja attribuutin nimien kirjoittaminen yhteen listaan on luonnollinen ratkaisu.

NSQL-kielen kysely yksinkertaisimmillaan on tyyppiä:

```
<query_expression> := <relation_name>(<attribute_list>)
                      WHERE <condititon>
```

missä <attribute_list> voi sisältää atomisten attribuuttien ja relaatioarvoisten attribuuttien nimiä, sekä projektiolauseita.

```
<attribute_list> :=   {<atomic_attribute_name>}•,
                      {<subrelation>}•,
                      {<subrelation>(<attribute_list>)}•
```

Luomieni uuden tyyppisten suoraviivaisten attribuuttiviittauksien ja suoraviivaisten relaatiokaavion muunnosoperaatioiden, sekä NF2-mallia hyödyntävän transitiivisulkeumaoperaation lisäksi NSQL-kielessä on seuraavia parannuksia SQL-kieleen, jonka tyyppisiä ehdotuksia on esitetty lähteissä [RoKoBa87] ja [Date84].

- Vaikeaselkoiset GROUP BY - HAVING kyselyt on kokonaan poistettu NF2-mallissa tarpeettomina, koska NF2-mallissa yleensä tarvittavat ryhmittelyt ovat tietokantaan sisäänrakennettuna ja relaatiokaavion uudelleenmuotoiluoperaatioilla voidaan toteuttaa kaikki muut tarvittavat ryhmittelyt.
- UNION voidaan tehdä suoraan relaatioille, eikä vain SFW-lauseille, mikä lyhentää kyselylauseita ja antaa selkeämmän kuvan operaation tarkoituksesta.
- Aggregointifunktiot kohdistetaan relaatioihin (tai kyselyihin) eikä attribuutteihin.
- Kun NF2-mallissa aggregointifunktio kohdistetaan alirelaatioon puretaan tulosrelaatiota automaattisesti yhdellä tasolla unnest-operaatiolla, jotta aggregointifunktion tuottama yksirivinen relaatio eliminoituu ja korvautuu attribuutilla.
- Kaikkiin attribuutteihin viittaavaan * symbolin sijasta käytetään lausetta ALL
- Lauseella ALL BUT voidaan ilmaista ne attribuutit, joita ei haluta mukaan projektioon.

Käymme seuraavaksi systemaattisesti läpi NSQL-kielen ominaisuudet esimerkkien avulla. Esitän vertailun vuoksi osan esimerkeistä myös perinteisen SQL:n tyyppisellä kielellä, jossa noudatetaan SQL:n merkintätapoja mahdollisimman pitkälle ja kirjoitetaan nest- ja unnest-ilmaisut explisiittisesti näkyviin. Nimitän tätä kieltä SQLX:ksi. Lisäksi esitän kyselyt laajennetulla relaatioalgebralla, ellei relaatioalgebraesitys ole täysin ilmeinen edellisen esimerkin perusteella. Merkitsen aina esimerkkikyselyn

eteen kielen, jolla esitys on tehty muodossa 'NSQL:' ,'SQL:', 'SQLX:' tai 'RA:'. Esimerkkikyselyt tehdään kuvassa kuvio 3.b esitellystä 1NF-tietokannasta ja kuvan kuvio 3.a NF2-tietokannasta, ellei toisin mainita.

5.2. PERUSKYSELYT 1NF-RELAATIOISTA

1NF-relaatio on vain NF2-relaation erikoistapaus, jossa ei ole yhtään relaatioarvoista attribuuttia. Siten 1NF-relaatioiden käsittely ei poikkea mitenkään NF2-relaatioiden käsittelystä. Koska kyseessä on uusi kyselykieli, esitän aluksi peruskyselyitä 1NF-mallin relaatioista jotta lukija pääsee sisään kielen esitystapaan ja siirryn vasta sitten NF2-relaatioiden tuomiin ominaisuuksiin, jossa NSQL-kielen edut tulevat kunnolla esiin. Käytän puolipistettä (;) kyselyn loppumerkkinä.

PROJEKTIO 1NF-RELAATIOSTA:

Projektio tapahtuu kertomalla relaation nimi ja sen perässä sulkujen sisällä projisoitavat attribuutit. Samalla attribuuttien järjestys voidaan vaihtaa.

Esimerkki: Anna kuvan kuvio 3.b 1NF-tietokannasta työntekijöiden palkat ja nimet.

```
NSQL:     Työntekijät(palkka, tynimi);

SQL:      SELECT palkka, tynimi
          FROM Työntekijät;

RA:       π palkka,tynimi (Työntekijät);
```

VALINTA 1NF-RELAATIOSTA:

Valinta tapahtuu WHERE lauseella. SELECT ja FROM lauseita ei tarvita, vaan kerrotaan vain minkä relaation rivit halutaan ja annetaan rajoitusehto WHERE lauseessa.

Esimerkki: Anna kuvan kuvio 3.b 1NF-tietokannasta työntekijät jotka työskentelevät osastolla 10.

```
NSQL:     Työntekijät
          WHERE ono = 10;

SQL:      SELECT *
          FROM Työntekijät
          WHERE ono = 10;

RA:       σ ono = 10 (Työntekijät);
```

Koko relaation valitseminen ilman projektiota ja valintaa on NSQL-kielellä hyvin suoraviivaista.

```
NSQL:     Työntekijät;

SQL:      SELECT *
          FROM Työntekijät;
```

```
RA:       Työntekijät;
```

LIITOS 1NF-RELAATIOSTA:

Karteesinen tulo lasketaan kyselyssä lueteltujen relaatioiden välillä. Liitoksessa liitosattribuutit määritellään WHERE-lauseessa.

Esimerkki: Hae kuvan kuvio 3.b 1NF-tietokannasta osastot ja niiden työntekijät.

```
NSQL:     Työntekijät, Osastot
          WHERE Työntekijät.ono = Osastot.ono;

SQL:      SELECT *
          FROM Työntekijät, Osastot
          WHERE Työntekijät.ono = Osastot.ono;

RA:       Työntekijät∞<Työntekijät.ono = Osastot.ono> Osastot;
```

5.3. PROJEKTIO

Seuraavaksi käsitellään yleisiä kyselyitä esimerkin kuvio 3.a NF2-tietokannasta. Projektio tapahtuu hyvin suoraviivaisesti myös syvemmältä NF2-relaatiosta sulkunotaatiolla.

Esimerkki: Hae yrityksen osaston nimi ja sen työntekijöiden nimet ja palkat, sekä työntekijöiden lasten nimet.

```
NSQL:     Yritys(onimi, Työntekijät(tynimi, palkka, Lapset(lnimi)));

RA:       ν Työntekijät = (tynimi, palkka, Lapset) (ν Lapset = (lnimi)
          (π onimi, tynimi, palkka, lnimi (μ Lapset (μ Työntekijät (Yritys)))));
```

Answer

onimi	Työntekijät		
	tynimi	palkka	Lapset
			lnimi

Kuvio 5.B *tulosrelaation relaatiokaavio.*

Projektiolauseessa itse asiassa lueteltiin suoraan lopputulokseen haluttu relaatiokaavion osa sulkunotaatiota käyttäen. Tämä tapa on huomattavasti intuitiivisempi, kuin perinteistä SQL-kieltä mukailevan syntaksin mukainen alikyselyihin perustuva kysely:

```
SQLX:     NEST
               (NEST
                    (SELECT onimi, tynimi, palkka, lnimi
                     FROM UNNEST
                               (UNNEST Yritys
                                ON Työntekijät)
                          ON Lapset)
                ON lnimi AS Lapset)
          ON tynimi, palkka, Lapset AS Työntekijät;
```

Mikäli projektiolauseessa ilmaistaan lehtisolmuna alirelaation nimi, otetaan alirelaatio mukaan tulokseen kaikkine attribuutteineen. Mukaan tulevat siis myös kaikki relaatioarvoiset attribuutit sisältäen koko alirelaatiohierarkian.

Esimerkki: Hae yrityksen osaston nimi ja sen työntekijät kaikkine tietoineen.

```
NSQL:     Yritys(onimi, Työntekijät);
```

Answer

onimi	Työntekijät								
	tyno	tynimi	hetu	esimies	palkka	Kurssit		Lapset	
						kuno	pvm	lnimi	synt

Kuvio 5.C *Tulosrelaation relaatiokaavio.*

```
SQLX:     SELECT onimi, Työntekijät
          FROM Yritys;
```

RA: $\pi_{\text{onimi, Työntekijät}}$ (Yritys);

RELAATIOIDEN JA ATTRIBUUTTIEN UUDELLEEN NIMEÄMINEN AS-LAUSEELLA

Projisoinnin yhteydessä voidaan attribuutit ja alirelaatiot uudelleen nimetä AS-lauseella perinteisen SQL-kielen mukaisesti. AS-lauseen edellä kerrotaan attribuutin tai relaation nimi lähtörelaatiossa ja AS-lauseen perässä kerrotaan nimi tulosrelaatiossa.

```
NSQL: Yritys(onimi AS osasto_nimi, Työntekijät AS Työläiset);
```

Answer

osasto_nimi	Työläiset								
	tyno	tynimi	hetu	esimies	palkka	Kurssit		Lapset	
						kuno	pvm	lnimi	synt

Kuvio 5.D *Tulosrelaation relaatiokaavio.*

```
SQLX:     SELECT onimi AS osasto_nimi, Työntekijät AS Työläiset
          FROM Yritys;
```

RA: ANSWER(osasto_nimi, Työläiset) ← $\pi_{\text{onimi, Työntekijät}}$ (Yritys);

5.4. VALINTA

Tietokantakyselyissä NF2-relaatioista täytyy pystyä asettamaan ehtoja myös alirelaatioiden riveille. Joskus näillä ehdoilla halutaan karsia alirelaatioin rivejä, mutta joskus taas halutaan näiden ehtojen perusteella karsia rivit ylimmältä relaatiotasolta lähtien. Eli halutaanko kaikki oppilaat ja heidän suorittamassa ehdot täyttävät kurssit, vai halutaanko vain ne oppilaat, joilla on tietyt ehdot täyttäviä kursseja. Jälkimmäisessä tapauksessa on vielä jäljellä vaihtoehdot halutaanko valituilta oppilailta kaikki kurssit vai vain ehdot täyttävät kurssit.

Valintaoperaatio tapahtuu WHERE lauseella, mutta NF2-relaatiossa se voi kohdistua eri upotustasoille. Valitaan aluksi uloimman tason mukaan.

Esimerkki: Hae Yrityksen osastonimi ja osaston työntekijöiden nimet osastolta 10.

```
NSQL:     Yritys(onimi, Työntekijät(tynimi))
          WHERE ono = 10;
```

Answer

onimi	Työntekijät
	tynimi
tuotanto	erkki
	mira
	heikki

Kuvio 5.E *Tulosrelaatio.*

Yrityksen osastotasolle (juuritaso) kohdistettu WHERE-ehto ottaa mukaan kaikki tiedot ehdon täyttäviltä osastoilta, eli myös ehdon täyttävien osastojen alirelaatiot kokonaisuudessaan. Kyselyn ensimmäisellä rivillä projisoidaan mukaan vain osastonimi ja Työntekijän nimi attribuutit.

Sama kysely esitettynä perinteistä SQL:ää mukaillen ja relaatioalgebralla

```
SQLX:     NEST
               (SELECT onimi, tynimi
               FROM UNNEST Yritys ON Työntekijät
               WHERE ono = 10
          ON tynimi AS Työntekijät;
```

RA: $\nu_{\text{Työntekijät = (tnimi)}}\ (\pi_{\text{onimi,tnimi}}\ (\sigma_{\text{ono = 10}}\ (\mu_{\text{Työntekijät}}\ (\text{Yritys}))));$

Rajoitetaan seuraavaksi alirelaation rivejä.

Esimerkki: Hae osastoittain työntekijät, jotka tienaavat yli 15000.

```
NSQL:     Yritys(onimi, Työntekijät)
          WHERE Työntekijät.palkka > 15000;
```

RA: $\nu_{\text{Työntekijät = (tyno,tynimi,hetu,esimies,palkka,Kurssit,Lapset)}}$
$(\pi_{\text{onimi, tyno, tynimi, hetu, esimies, palkka, Kurssit, Lapset}}$
$(\sigma_{\text{palkka > 15000}}(\mu_{\text{Työntekijät}}(\text{Yritys}))));$

Answer

onimi	Työntekijät								
	tyno	tynimi	hetu	esimies	palkka	Kurssit		Lapset	
						kuno	pvm	lnimi	synt
tuotanto	14	erkki	111169-0123	0	24000	11 15 21	2.3.81 5.8.91 1.9.90	mari veli	93 90
	25	mira	010171-1234	14	19000	11 31	4.5.91 2.2.95	jaakko	96
laskutus	34	paavo	040471-4567	1	18000	15	5.8.91	kati ilona	82 84

Kuvio 5.F *Tulosrelaatio.*

Mukaan otetaan vain ne alirelaation 'Työntekijät' rivit, jotka täyttävät ehdot. Ehdot täyttäviltä alirelaation 'Työntekijät' riveiltä mukaan tulevat kaikki attribuutit, myös relaatioarvoiset kaikkine alirelaatioineen. Osastoista jäävät pois ne, joilla ei ole ehdot täyttäviä työntekijöitä. Sama kysely perinteistä SQL:ää mukaillen vaatii jälleen relaatiokaavion uudelleenmuotoilua:

```
SQLX:     NEST
              (SELECT onimi, tyno, tynimi, hetu, esimies, palkka,
                      Kurssit, Lapset
               FROM UNNEST Yritys
                    ON Työntekijät
               WHERE palkka > 15000)
          ON tyno, tynimi, hetu, esimies, palkka, Kurssit, Lapset
          AS Työntekijät;
```

Jos halutaan säilyttää osaston tiedot, vaikka osastolla ei olisikaan ehdot täyttäviä työntekijöitä, ilmaistaan tämä NSQL-kielellä PRESERVE-predikaatilla. Työntekijä-relaatio saa tällöin arvokseen tyhjän relaation niille osastoille joilla ei ollut ehdot täyttäviä työntekijöitä.

Esimerkki: Hae Yrityksen osastonimet ja osaston ne työntekijät, jotka ansaitsevat yli 15000. Säilytä tuloksessa kaikki osastot.

```
NSQL:     Yritys(onimi, Työntekijät WHERE palkka > 15000)
          WHERE Työntekijät.palkka > 15000
          PRESERVE Yritys;
```

PRESERVE-ilmauksen perässä luetellaan ne relaatiotasot, joilta halutaan säilyttää kaikki rivit huolimatta ehtolauseesta. Tässä tapauksessa säilytetään relaation 'Yritys' kaikki rivit, vaikka alirelaatiot saisivat NULL-arvoja.

Answer

onimi	Työntekijät								
	tyno	tynimi	hetu	esimies	palkka	Kurssit		Lapset	
						kuno	pvm	lnimi	synt
tuotanto	14	erkki	111169-0123	0	24000	11 15 21	2.3.81 5.8.91 1.9.90	mari veli	93 90
	25	mira	010171-1234	14	19000	11 31	4.5.91 2.2.95	jaakko	96
markkinointi									
laskutus	34	paavo	040471-4567	1	18000	15	5.8.91	kati ilona	82 84

Kuvio 5.G *Tulosrelaatio, jossa osasto 20 on säilytetty tuloksessa, vaikka siellä ei ole yhtään ehdot täyttävää työntekijää.*

Relaatioalgebralla ilmaistuna kyselyä ei voida toteuttaa aivan suoraviivaisesti. Jaan relaatioalgebrakyselylauseen pienempiin osiin, jotta sitä on helpompi tulkita.

RA:

$$\text{vastaus} \leftarrow \pi_{\text{ono,onimi,sijainti,Työntekijät}} (\nu_{\text{Työntekijät = (tyno,tynimi,hetu,esimies,palkka,Kurssit,Lapset}} (\text{valitut});$$

$$\text{valitut} \leftarrow \text{kaikki\_osastot} >\infty_{\text{<ono = ttono>}} \text{valitut\_työntekijät};$$

$$\text{kaikki\_osastot} \leftarrow \pi_{\text{ono,onimi,sijainti}} (\mu_{\text{Työntekijät}} (\text{Yritys}));$$

$$\text{valitut\_työntekijät}(_{\text{ttono,tyno,tynimi,hetu,esimies,Kurssit,Lapset}}) \leftarrow \pi_{\text{ono,tyno,tynimi,hetu,esimies,Kurssit,Lapset}} (\sigma_{\text{palkka > 15000}} (\mu_{\text{Työntekijät}} (\text{Yritys})));$$

Lopuksi sama kysely esitettynä perinteistä SQL:ää mukaillen. Olen lisännyt SQLX-kieleen OUTERJOIN-operaattorin, jotta kielellä voi suorittaa suoraviivaisesti vasemman ulkoliitoksen. Outerjoin-operaattorin perässä ilmaistaan, minkä attribuuttien suhteen operaattoria edeltävä ja seuraava relaatio, tai SFW-lauseen tulos liitetään yhteen.

SQLX:

```
NEST
  SELECT onimi, tyno, tynimi,
         hetu, esimies, palkka, Kurssit, Lapset
  FROM  ((SELECT onimi,
          FROM Yritys)
         OUTERJOIN (onimi = onimi2)
         (SELECT onimi AS onimi1, tyno, tynimi,
                 hetu, esimies, palkka, Kurssit, Lapset
          FROM UNNEST Yritys
               ON Työntekijät
          WHERE palkka > 15000))
ON tyno, tynimi, hetu, esimies, palkka, Kurssit, Lapset
AS Työntekijät;
```

Joskus halutaan rajoittaa relaation rivejä ylemmältä tasolta saakka jonkin alirelaation sisällön perusteella karsimatta kuitenkaan alirelaation rivejä vastauksesta.

Esimerkki: Hae yrityksen niiden osastojen nimet, joilla jokin työntekijä ansaitsee yli 15000 ja osastojen kaikki työntekijät.

```
NSQL:    Yritys(onimi, Työntekijät)
         WHERE EXISTS (Työntekijät.palkka > 15000);

SQLX:    SELECT onimi, Työntekijät
         FROM Yritys
         WHERE ono IN (SELECT ono
                       FROM UNNEST Yritys
                            ON Työntekijät
                       WHERE palkka > 15000);
```

IN-merkintä tarkoittaa vertailuoperaatiota, jonka vasen operandi on atominen arvo ja oikea operandi on joukko. Vertailuoperaatio on tosi jos vasen operandi sisältyy oikean operandin esittämään joukkoon.

RA: $\text{tulos} \leftarrow \pi_{\text{onimi,Työntekijät}} (\text{osastot} \infty_{<\text{ono = ttono}>} (\text{Yritys}))$;
$\text{osastot(ttono)} \leftarrow \pi_{\text{ono}} (\sigma_{\text{palkka > 15000}} (\mu_{\text{Työntekijät}}(\text{Yritys}))$;

Answer

onimi	Työntekijät								
	tyno	tynimi	hetu	esimies	palkka	Kurssit		Lapset	
						kuno	pvm	*lnimi*	synt
tuotanto	14	erkki	111169-0123	0	24000	11 15 21	2.3.81 5.8.91 1.9.90	mari veli	93 90
	25	mira	010171-1234	14	19000	11 31	4.5.91 2.2.95	jaakko	96
	65	heikki	020255-2345	25	13000	11	4.4.97	viljo saara	89 90
laskutus	34	paavo	040471-4567	1	18000	15	5.8.91	kati ilona	82 84
	78	heli	050578-5678	34	7000			sepe jope	91 95

Kuvio 5.H *Vastausrelaatio jossa on ne osastot, joilla on yli 15000 ansaitseva työntekijä.*

Sama tulos saadaan myös seuraavalla kyselyllä, jossa käytetään aggregointifunktiota MAX kohdistettuna relaatioon, kuten NSQL-kielessä voi tehdä. Aggregointifunktioiden käyttö esitellään tarkemmin myöhemmin.

```
NSQL:    Yritys(onimi, Työntekijät)
         WHERE MAX(Työntekijät.palkka) > 15000;
```

Sama kysely perinteisen SQL-kielen merkintätapaa mukaillen:

```
SQLX:    SELECT onimi, (SELECT * FROM Työntekijät)
         FROM Yritys
         WHERE 15000 < (SELECT MAX(palkka)
                        FROM Työntekijä);
```

Laajennetulla relaatioalgebralla esitettynä operaatiosarja on seuraava:

RA: $\pi_{\text{onimi,Työntekijät}} (\sigma_{\text{max\_pm > 15000}}$
$(\text{max\_p} := \text{MAX}_{\text{Työntekijät.palkka}} (\text{Yritys})))$;

5.5. JOUKKO-OPILLISET OPERAATIOT

Union voidaan NSQL-kielessä asettaa myös relaatioiden väliin eikä vain kyselylauseiden väliin, kuten SQL-kielessä. Unioniin osallistuvien relaatioiden on oltava unioniyhteensopivia ja ne voivat olla myös NF2-relaatioita. Käytetään esimerkissä relaatioita 'Rel1' ja 'Rel2', jotka voivat olla mitä hyvänsä unioniyhteensopivia relaatioita.

```
NSQL:     Rel1 UNION Rel2;

SQL:      SELECT *
          FROM Rel1
          UNION
          SELECT *
          FROM Rel2;

RA:       Rel1 ∪ Rel2;
```

Jos UNION-operaation tulosrelaatiosta halutaan tehdä projektiota ja valintaa, käy se helposti seuraavasti.

```
NSQL:     (Rel1 UNION Rel2)(attrib1, attrib2)
          WHERE attrib3 = 10;
```

Esimerkissä projisoidaan unionin tulosrelaatiosta attribuutit attrib1 ja attrib2 ja tehdään valinta attribuutin attrib3 arvon mukaan. Kyselyn optimoijan tehtävä on päättää, tehdäänkö projektio ja valinta todellisuudessa ennen vai jälkeen unionin.

Sama kysely perinteisellä SQL-kielellä.

```
SQL:      SELECT attrib1, attrib2
          FROM Rel1
          WHERE attrib3 = 10
          UNION
          SELECT attrib1, attrib2
          FROM Rel2
          WHERE attrib3 = 10;

RA:       π attrib1, attrib2 σ attrib3 = 10 (Rel1 ∪ Rel2);
```

Erotus toimii samalla tyylillä kuin unioni.

```
NSQL:     (Rel1 DIFFERENCE Rel2)(attrib1, attrib2)
          WHERE attrib3 = 10;

SQL:      SELECT attrib1, attrib2
          FROM (SELECT *
                FROM Rel1
                DIFFERENCE
                SELECT *
                FROM Rel2)
          WHERE attrib3 = 10;

RA:       π attrib1, attrib2 σ attrib3 = 10 (Rel1 - Rel2);
```

5.6. TULOSRELAATION RAKENTEEN UUDELLEENMUOTOILU

5.6.1. Alirelaatioiden purkaminen

Alirelaatioiden purkaminen, eli unnest-lauseen käyttö on NSQL-kielessä hyvin intuitiivista. Se tapahtuu viittaamalla pistenotaatiolla, jolloin samalla suoritetaan projektio.

Esimerkki: Anna kaikkien työntekijöiden nimet.

Tehdään kysely esimerkin vuoksi ensin ilman alirelaatioiden purkamista:

```
NSQL:     Yritys(onimi, (Työntekijät(tynimi)));
```

Tässä kyselyssä suoritetaan vain projektiota ja saamme tulokseen työntekijät osastoittain ryhmiteltynä. Haluamme tulokseen kuitenkin vain työntekijöiden nimet ilman ryhmittelyä, joten osoitamme pistenotaatiolla suoraan attribuuttiin 'tynimi', jolloin relaatiotasot puretaan.

```
NSQL:     Yritys.Työntekijät.tynimi;
```

Answer

tynimi
erkki
mira
heikki
sirkka
paavo
heli

Kuvio 5.1 *Tulosrelaatio.*

Suoritetaan seuraavaksi sama kysely suoraan perinteisestä SQL:stä laajennetulla kielellä, jossa käyttäjän täytyy ensin muotoilla lähtörelaatio nest- ja unnest-operaatioilla.

```
SQLX:     SELECT tynimi
          FROM UNNEST Yritys ON Työntekijät;
```

RA: $\pi_{\text{tynimi}}\ (\mu_{\text{Työntekijät}}(\text{Yritys}))$;

Jos viitataan suoraan useamman alirelaatiotason läpi, pysyy NSQL-kielen ilmaisu yhä täysin intuitiivisena, kun taas unnest-operaatioiden explisiittisesti näkyviin kirjoittamiseen perustuva perinteisempi ratkaisu alkaa olla varsin vaikeaselkoinen. Haetaan esimerkiksi kaikki työntekijöiden lasten nimet.

```
NSQL:     Yritys.Työntekijät.Lapset.lnimi;

SQLX:     SELECT lnimi
          FROM UNNEST(UNNEST Yritys
                      ON Työntekijät)
               ON Lapset;
```

```
RA:     π1nimi (μLapset (μTyöntekijät(Yritys)));
```

Pistenotaatio ja sulkujen käyttö voidaan yhdistää, jolloin voidaan purkaa relaatiotasoja ja projisoida samalla useampia attribuutteja ilman, että tarvitsee kirjoittaa koko alirelaatiorakenne näkyviin jokaiselle attribuutille erikseen.

```
NSQL:   Yritys.Työntekijät.(tynimi, palkka);

SQLX:   SELECT tynimi, palkka
        FROM UNNEST Yritys
             ON Työntekijät;

RA:     πtynimi, palkka (μTyöntekijät(Yritys));
```

Answer

tynimi	palkka
erkki	24000
mira	19000
heikki	13000
sirkka	14000
paavo	18000
heli	7000

Kuvio 5.J *Tulosrelaatio.*

Sulku- ja pistenotaatioita voi kombinoida vapaasti, jolloin voidaan purkaa relaatioita alirelaatiohierarkian eri tasoilta ja suorittaa samalla projisointia. Suoritetaan seuraavaksi relaatiotasojen purkua esimerkin vuoksi vielä alirelaatiohierarkian keskeltä.

Esimerkki: Hae yrityksen osastonimi, sekä työntekijän nimi ja palkka 1NF-muodossa.

```
NSQL:   Yritys(onimi, Työntekijät.(tynimi, palkka));

SQLX:   SELECT onimi, tynimi, palkka
        FROM UNNEST Yritys
             ON Työntekijät;

RA:     πonimi, tynimi, palkka (μTyöntekijät(Yritys));
```

Answer

onimi	tynimi	palkka
tuotanto	erkki	24000
tuotanto	mira	19000
tuotanto	heikki	13000
markkinointi	sirkka	14000
laskutus	paavo	18000
laskutus	heli	7000

Kuvio 5.K *Tulosrelaatio.*

ALL-ilmauksella voi viitata kaikkiin relaation attribuutteihin. Tästä on hyötyä esimerkiksi relaatiotasoja purettaessa, jos emme halua projisoida mitään pois viimeiseltä alirelaatiotasolta.

Esimerkki: Hae yrityksen osastonimi ja osastolla käydyt kurssit 1NF-muodossa.

```
NSQL:     Yritys(onimi, Työntekijät.Kurssit.(ALL));

SQLX:     SELECT onimi, kuno, pvm
          FROM UNNEST (UNNEST Yritys
                       ON Työntekijät)
               ON Kurssit;
```

RA: $\pi_{onimi,\ kuno,\ pvm}\ (\mu_{Kurssit}\ (\mu_{Työntekijät}(Yritys)))$;

Answer

onimi	kuno	pvm
tuotanto	11	2.3.81
tuotanto	15	5.8.91
tuotanto	21	1.9.90
tuotanto	11	4.5.91
tuotanto	31	2.2.95
tuotanto	11	4.4.97
markkinointi	11	4.5.91
markkinointi	42	7.9.95
laskutus	15	5.8.91

Kuvio 5.L

Haetaan seuraavaksi samasta relaatiosta kaikki oppilaat, jotka ovat suorittaneet tietyn kurssin. Tämän kyselyn tulosta ei kuvan kuvio 3.a NF2-relaatiosta nähdä suoraan koska se on suunniteltu olettaen, että yleensä ollaan kiinnostuneita siitä mitä kursseja tietyt oppilaat ovat suorittaneet. Kyseessä on kuitenkin many-to-many suhde oppilaiden ja kurssien välillä ja on luonnollista, että joskus on tarpeita tutkia suhdetta toiseenkin suuntaan. Nyt siis halutaan selvittää ketkä oppilaat ovat suorittaneet tietyn kurssin.

```
NSQL:     Yritys.Työntekijät.tynimi
          WHERE Yritys.Työntekijät.Kurssit.kuno = 15;

SQLX:     SELECT tynimi
          FROM UNNEST (UNNEST Yritys
                       ON Työntekijät)
               ON Kurssit
          WHERE kuno = 15;
```

RA: $\pi_{tynimi}\ (\sigma_{kuno\ =\ 15}\ (\mu_{Kurssit}\ (\mu_{Työntekijät}(Yritys))))$;

NSQL-esityksessä viitataan where-ehdossa kyselyn lähtörelaatioon rivien karsimiseksi ja suoritetaan relaatiokaavion uudelleenmuotoilu helposti pistenotaatiolla. Perinteisemmällä tavalla ilmaistuna käyttäjä joutuu muotoilemaan relaation ensin uudelleen ja hahmottamaan mielessään tämän uuden relaatiokaavion, johon hän sitten where-ehdon kohdistaa.

5.6.2. Alirelaatioiden luominen

Relaatiokaavioon voidaan lisätä uusia relaatiotasoja yksinkertaisesti kirjoittamalla upotettava relaatio tai relaation luova kysely haluttuun paikkaan formuloitavan kyselyn tulosrelaatiokaaviota. Lisättävän relaation nimen edessä kerrotaan &-merkillä, että tähän kohtaan lisätään toinen relaatio ja että määrittely lähtee liikkeelle juurirelaatioiden tasolta.

Esimerkki: Pakkaa kuvan kuvio 3.b 1NF tietokannasta työntekijärelaatio osastorelaation sisään.

```
NSQL:     Osastot(ALL, &Työntekijät)
          WHERE Osastot.ono = Työntekijät.ono;
```

Answer

ono	onimi	sijainti	Työntekijät				
			tyno	tynimi	hetu	esimies	palkka
10	tuotanto	tampere	14 25 65	erkki mira heikki	111169-0123 010171-1234 020255-2345	0 14 25	24000 19000 13000
20	markkinointi	helsinki	59	sirkka	030359-3456	14	14000
30	laskutus	tampere	34 78	paavo heli	040471-4567 050578-5678	1 34	18000 7000

Kuvio 5.M *Tulosrelaatio.*

Relaation lisääminen toisen sisään ei ole suoraviivaisesti NEST-operaatio, vaan kyselyn optimoijan tehtäväksi jää määritellä tarvittava unnest-, nest- ja valintaoperaatioiden sarja. Loppukäyttäjä vain ilmaisee, mihin kohtaan relaatiokaaviota hän haluaa jonkin alirelaation lisätä. NSQL-kieli edellyttää siten perinteisten optimointitekniikoiden laajentamista, jotta NSQL-ilmauksen perusteella pystytään luomaan laajennetunrelaatioalgebran lause. Mikäli relaatioiden, alirelaatioiden ja attribuuttien nimet olisivat yksikäsitteisiä koko tietokannassa, ei &-merkintää välttämättä tarvittaisi lainkaan, ellei haluta näyttää selvästi, missä kohdassa tapahtuu relaation upotus toisen sisään projektion sijasta. Kun suoritetaan liitosoperaatio ja upotus yhdessä, järjestelmä poistaa automaattisesti liitosattribuutit upotettavasta relaatiosta. Esimerkkitapauksessa 'ono'-attribuutin 'Työntekijät'-relaatiosta. Vastaava relaatioalgebraoperaatioiden sarja on:

```
RA:       ν Työntekijät = (tyno,tynimi,hetu,esimies,palkka) ((Osastot)∞(Työntekijät));
```

ja vastaava kysely perinteisemmällä laajennetulla SQL:llä esitettynä on:

```
SQLX:     SELECT ono, onimi, sijainti, Työntekijät
          FROM Osastot, (NEST Työntekijät
                         ON tyno, tynimi, hetu, esimies, palkka
                         AS Työntekijät)
          WHERE Osastot.ono = Työntekijät.ono;
```

Halutessamme käsitellä vain yhtä relaatiota sisentämällä joitain sen attribuutteja uudeksi alirelaatioksi, voimme kirjoittaa saman lähtörelaation useampaan kertaan tulosrelaatioon käyttäen &-notaatiota. Tämä vastaa suoraan relaatioalgebran nest-operaatiota silloin, kun sisennystä tehdään vain yhden tason verran. NSQL-kielessä on tosin mahdollista sisentää kerralla useita relaatiotasoja. Yhden relaation käsittely ei eroa ulkopuolisen relaation upottamisesta muuten kuin, että liitosehtoja ei tarvita. Otetaan esimerkkirelaatioksi 1NF-relaatio 'Osastot', johon on liitetty 1NF-relaatio 'Tuotteet' siten, että lopputuloskin on 1NF-muodossa. Esimerkkirelaation relaatiokaavio on:

Osastot2

ono	onimi	sijainti	tuno	tunimi

Kuvio 5.N *Esimerkkirelaation relaatiokaavio.*

Esimerkki: Pakkaa relaation 'Osastot2' attribuutit 'tuno' ja 'tunimi' alirelaatioksi 'Tuotteet'

```
NSQL:     Osastot2(ono, onimi, sijainti,
                   &Osastot2(tuno, tunimi) AS Tuotteet);
```

Answer

ono	onimi	sijainti	Tuotteet	
			tuno	tunimi

Kuvio 5.O *Tulosrelaation relaatiokaavio.*

Relaation pakkaaminen NF2-muotoon ja toisen relaation liittäminen toisen sisään eivät NSQL-kielessä juurikaan eroa toisistaan, koska liitoksessa tapahtuu samalla pakkaaminen hyvin intuitiivisesti. Edellisen kyselyn SQLX ja RA-vastine ovat seuraavat:

```
SQLX:     NEST Osastot2
          ON tuno, tunimi
          AS Tuotteet;
```

RA: $\nu_{\text{Tuotteet = (tno, tunimi)}}$ (Osastot2);

Käännetään seuraavaksi ylirelaatio-alirelaatio suhde toisinpäin, eli käännetään NF2-relaation hierarkiatasot. Tätä tarvitaan usein many-to-many suhteissa kun halutaan viitata suhteessa toisinpäin. NF2-esimerkkirelaatiossamme oletetaan, että yleensä halutaan tietää mitä kursseja kukakin työntekijä on suorittanut. Tämän tiedon esimerkkirelaatiosta näkeekin suoraan. Jos kuitenkin halutaan tutkia ketkä työntekijät ovat suorittaneet tietyn kurssin, täytyy relaatiokaavion hierarkiatasot kääntää toisinpäin.

Esimerkki: Etsi NF2-relaatiosta 'Yritys' kurssit 11, 15 tai 21 suorittaneiden työntekijöiden nimet. Muotoile tulos NF2-muotoon.

```
NSQL: Yritys.Työntekijät.Kurssit(kuno, &Yritys.Työntekijät(tynimi))
      WHERE Yritys.Työntekijät.Kurssit.kuno IN (11, 15, 21);
```

Esimerkissä käytetty IN-vertailuoperaatio palauttaa tosi-arvon, jos sen vasemmanpuoleisen atomisen operandin arvo sisältyy oikean puoleisen joukko-arvoisen operandin arvoihin.

Tulosrelaatio on tällöin seuraava:

Answer

kuno	Työntekijät
	tynimi
11	erkki mira heikki sirkka
15	erkki paavo
21	erkki

Kuvio 5.P *Tulosrelaatio.*

Vastaava relaatioalgebralause on:

```
RA:     π kuno, Työntekijät (σ kuno = 11 OR kuno = 15 OR kuno = 21
        (ν Työntekijät = (tynimi) (μ Kurssit (μ Työntekijät (Yritys)))));
```

Perinteisemmällä laajennetulla SQL-kielellä ilmaistuna kysely on seuraava:

```
SQLX:   SELECT kuno, Työntekijät
        FROM NEST (UNNEST (UNNEST Yritys
                           ON Työntekijät)
                  ON Kurssit)
             ON tynimi AS Työntekijät
        WHERE kuno IN (11, 15 ,21);
```

Valintaoperaatiossa esiteltiin PRESERVE-ilmauksella, jolla pystyttiin säilyttämään jonkin relaatiotason kaikki rivit vaikka rivin jokin alirelaatio saisi valinnan tuloksena NULL-arvoisen alirelaation. Relaatiota toiseen upotettaessa saatamme jälleen tarvita PRESERVE-operandia. Jos haluamme säilyttää kaikki arvot toisesta relaatiosta liitoksen yhteydessä, eli suorittaa ulkoliitoksen, ilmaistaan tämä PRESERVE-predikaatilla. Suoritamme tämän kappaleen ensimmäisen kyselyn uudelleen, mutta varmistamme nyt, että tulokseen tulevat kaikki osastot, vaikkei niillä olisi työntekijöitä.

Esimerkki: Pakkaa kuvan kuvio 3.b työntekijärelaatio osastorelaation sisään.

```
NSQL:   Osastot(ALL, &Työntekijät)
        WHERE Osastot.ono = Työntekijät.ono
        PRESERVE Osastot;

SQLX:   SELECT ono, onimi, sijainti, Työntekijät
        FROM Osastot
             OUTERJOIN WITH (Osastot.ono = Työntekijät.ono)
             (NEST Työntekijät
              ON tyno, tynimi, hetu, esimies, palkka
              AS Työntekijät);

RA:     ν Työntekijät = (tyno,tynimi,hetu,esimies,palkka)
        ((Osastot)>∞ Osastot.ono = Työntekijät.ono (Työntekijät));
```

5.7. ALIAS NIMIEN KÄYTTÖ

Alias nimiä tarvitaan sellaisten kyselyiden formuloinnissa, joissa halutaan käyttää kopioita samasta relaatiosta. ALIASE-lauseella voidaan määritellä samasta relaatiosta kaksi ilmentymää.

Esimerkki: Luo pareja samalla paikkakunnalla sijaitsevista yrityksen osastonumeroista.

```
NSQL:   Eka.ono, Toka.ono
        ALIASE eka  = Yritys,
               toka = Yritys
        WHERE Eka.sijainti = Toka.sijainti
              AND Eka.ono < Toka.ono.
```

```
RA:     π Eka.ono, Toka.ono (σ Eka.ono < Toka.ono (Eka ∞ Eka.ono = Toka.ono Toka));
        Eka ← Yritys;
        Toka ← Yritys;
```

Aliase-lause on yleinen aliasnimen määrittävä lause, joten siinä voi antaa alias-nimen myös jollekin useaan kertaan käytettävälle kyselylauseelle. Aliase-lauseen vaikutusalue on kysely, jossa se ilmenee projisointirivin jälkeen. Jos samaa alias-nimeä halutaan käyttää useammassa kyselyssä, sallitaan NSQL-kielessä aliase-lause sijoittaa myös yksinään NSQL-lauseen ulkopuolelle. Tällöin lähetettäessä useamman NSQL-kyselyn sarjaa DBMS:n kyselyngeneroijalle, voidaan niissä kaikissa käyttää aliasnimiä, jotka on määritelty kyselysarjan alussa. Kyselysarjan eteen yksittäisten kyselyiden ulkopuolelle sijoitetun aliase-lauseen vaikutusalue on seuraavaan commit-lauseeseen saakka. Seuraavassa esimerkissä alias-lauseet ovat voimassa molemmille NSQL-kyselyille.

```
ALIASE alias1 = <kyselylause>,
       alias2 = <kyselylause>;

<NSQL-kysely1>;
<NSQL-kysely2>;
COMMIT;
```

Tämäntyyppistä aliase-lausetta apurelaatioiden käytön mahdollistamiseksi on erittäin hyödyllinen relaatiomallin kyselykielessä. Lause vastaa relaatioalgebran lauseita kirjoitettaessa olevaa mahdollisuutta jakaa monimutkainen kysely lyhyemmiksi relaatioalgebran lauseiksi käyttäen välirelaatioita. Tällainen automaattinen väli- tai apurelaation luova lause on varsin tarpeellinen tietokantakyselyitä tehtäessä, koska välirelaatioiden avulla on helpompi hallita mutkikkaiden kyselyiden formulointia kuin yrittämällä luoda kysely, joka tekee asian kerralla. Lisäksi nykyiset kyselyn optimoijat eivät aina selviä kunnialla kovin mutkikkaasta kyselystä. Käytännössä siis apurelaatioita tarvitaan jatkuvasti, mutta ne joudutaan yleensä luomaan tietokantaan ja huolehtimaan siten myös niiden tuhoamisesta. NSQL-kielessä apurelaation voi luoda aliase-lauseella ja järjestelmä huolehtii sen tuhoamisesta, kun kysely vahvistetaan valmistuneeksi commit-toiminnolla.

5.8. AGGREGOINTIFUNKTIOT

Aggregointi-operaatiot kohdistetaan relaatioihin eikä attribuutteihin, kuten SQL-kielessä. Tällä tavalla aggregointioperaatioitakin on helppo upottaa kyselyiden sisälle ja kohdistaa suoraan alirelaatioihin. Aggregointioperaatiot myös purkavat automaattisesti yhden relaatiotason relaatiosta, johon ne kohdistetaan, jolloin tulokseksi ei tule relaatiota vaan atominen arvo.

Esimerkki: Hae Yrityksen osastonimi ja osaston työntekijöiden keskipalkka.

```
NSQL:    Yritys(onimi, AVERAGE(Työntekijät.palkka) AS keskipalkka);
```

Answer

onimi	keskipalkka
tuotanto	18667
markkinointi	14000
laskutus	12500

Kuvio 5.Q *Tulosrelaatio.*

Laajennetulla relaatioalgebralla kysely on:

```
RA:     π onimi,keskipalkka (keskipalkka := AVG Työntekijät.palkka (YRITYS));
```

Perinteistä SQL-kieltä mukaillen sama kysely on:

```
SQLX:   SELECT onimi, AVERAGE(palkka) AS keskipalkka
        FROM UNNEST Yritys ON Työntekijät;
```

Aggregointi voidaan kohdistaa NSQL-kielessä relaatioon, joten aggregointi suoritetaan kerralla kaikille operandirelaation attribuuteille. Olkoon meillä relaatio Myynti(myyjä, tammi, helmi, maalis, huhti, touko, kesä, heinä, elo, syys, loka, marras, joulu)), joka esittää myyjien myynnin kuukausittain. Saamme kuukausittaisen kokonaismyynnin helposti seuraavalla kyselyllä:

```
NSQL:  SUM(Myynti(tammi, helmi, maalis, huhti, touko, kesä,
                  heinä, elo, syys, loka, marras, joulu));
```

tai vielä lyhyemmin:

```
NSQL:  SUM(Myynti(ALL BUT myyjä));
```

Aggregointifunktion käyttö myös ehtolauseessa on huomattavasti selkeämpää kun aggregointia voi suorittaa relaatiolle. Tätä operaatiota käytettiinkin jo valintaa käsittelevässä kappaleessa, mutta otetaan tässä vielä toinen esimerkki.

Esimerkki: Hae yrityksen niiden osastojen nimet ja työntekijät, joilla työntekijöiden keskipalkka on yli 15000.

```
NSQL:   Yritys(onimi, Työntekijät)
        WHERE AVERAGE(Työntekijät.palkka) > 15000;

RA:     π onimi,Työntekijät (σ avg_p > 15000
        (avg_p := AVG Työntekijät.palkka (Yritys)));

SQLX:   SELECT onimi, (SELECT * FROM Työntekijät)
        FROM Yritys
        WHERE 15000 < (SELECT AVERAGE(palkka)
                       FROM Työntekijä);
```

5.9. TRANSITIIVISULKEUMA

Transitiivisulkeuman laskentaa varten lisätään NSQL-kieleen CLOSURE-predikaatti, jonka toiminta ja liittäminen muihin NSQL-kyselylauseisiin esitellään seuraavaksi. Closure-predikaatti saa lähtörelaatiokseen yhden transitiivirelaation ja antaa tuloksenaan relaation, johon transitiivisulkeuman tulos on tallennettu. Täten closure-predikaattia voi vapaasti yhdistellä muihin NSQL-kyselylauseisiin. Käsitte-

len kuitenkin tässä yhteydessä erikseen yleisimpiä transitiivisulkeuman liitoksia muihin operaatioihin kahdesta syystä: ensinnäkin aggregoivan transitiivisulkeuman ilmaisuvoimasta ja käyttötarpeista saa siten hyvän kuvan ja toisekseen muita NSQL-lauseita sopivasti closure-predikaattiin liittämällä voidaan huomattavasti tehostaa transitiivisulkeuman tuottamista. Transitiivisulkeuman laskentatehokkuuteen on hyvä kiinnittää huomiota enemmän kuin yleensä kyselylauseissa, koska täydellisen transitiivisulkeuman tuottaminen voi olla äärimmäisen raskas operaatio.

Käytämme tässä esimerkkinä seuraavaa lentoreitit sisältävää relaatiota, jossa sulkeuma-attribuutit ovat lähtö ja määränpää kaupunkeja kuvaavat attribuutit 'lähtöp' ja 'määräp'. Myös lentoasemia kuvaavia attribuutteja 'lasema' ja 'masema' voidaan käyttää sulkeuma-attribuutteina. Attribuutit 'lähtöaika' ja 'tuloaika' täyttävät myös sulkeuma-attribuuttien määritelmän, mutta niiden käyttämiselle ensisijaisena sulkeuma-ehtona on aika vaikeaa keksiä järkevää käyttöä.

Lennot

lähtöp	määräp	lasema	masema	lähtöaika	tuloaika	hinta	yhtiö
Tampere	Helsinki	Pirkkala	Vantaa	7:00	7:20	500	Finnair
Helsinki	Lontoo	Vantaa	Gatwick	8:10	10:50	4200	Finnair
Lontoo	Helsinki	Gatwick	Vantaa	11:50	14:20	3000	Finnair
Helsinki	Lontoo	Vantaa	Stansted	15:55	18:20	4400	BA
Lontoo	Newyork	Standsted	Newyork	20:00	6:00	6500	BA
Helsinki	Tukholma	Helsinki	Tukholma	10:00	10:50	3400	SAS
Tukholma	Lontoo	Tukholma	Lontoo	12:20	14:20	3600	SAS
...	...	...	...	...	...	...	...

Kuvio 5.R *Lentoreitit sisältävä relaatio.*

Lähdetään liikkeelle tuottamalla aluksi täydellinen transitiivisulkeuma ilman aggregointia, mikä on transitiivisulkeumaoperaation perustyyppi.

Esimerkki: Minkä paikkakuntien välille löytyy lentoreitti.

```
NSQL:     CLOSURE määräp = NEXT lähtöp OF Lennot;
```

Transitiivisulkeumanlaskentalauseessa kerrotaan closure-predikaatin perässä primäärisulkeumaehdossa attribuutit, joiden perusteella polun peräkkäiset särmät liitetään toisiinsa. Tämä ilmaistaan muodossa:

edellisen_särmän_loppupiste = NEXT seuraavan_särmän_alkupiste

Tämä luetaan: edellisen särmän loppupiste on seuraavan särmän alkupiste. 'Lennot'-relaatiosta siis: edellisen lennon pääteasema on seuraavan lennon lähtöasema. Lauseen lopussa kerrotaan vielä mistä relaatiosta sulkeuma lasketaan, eli tässä tapauksessa relaatiosta 'Lennot'.

lähtöp	määräp	pno	Path								
			ano	lähtöp	määräp	lasema	masema	lähtöaika	tuloaika	hinta	yhtiö
Tampere	Helsinki	1	1	Tampere	Helsinki	Pirkkala	Vantaa	7:00	7:20	500	Finnair
Tampere	Lontoo	1	1 2	Tampere Helsinki	Helsinki Lontoo	Pirkkala Vantaa	Vantaa Gatwick	7:00 8:10	7:20 10:50	500 4200	Finnair Finnair
Tampere	Lontoo	2	1 2	Tampere Helsinki	Helsinki Lontoo	Pirkkala Vantaa	Vantaa Stansted	7:00 15:55	7:20 18:20	500 4400	Finnair BA
Tampere	Lontoo	3	1 2 3	Tampere Helsinki Tukholma	Helsinki Tukholma Lontoo	Pirkkala Helsinki Tukholma	Vantaa Tukholma Lontoo	7:00 10:00 12:20	7:20 10:50 14:20	500 3400 3600	Finnair SAS SAS
Tampere	Newyork	1	1 2 3	Tampere Helsinki Lontoo	Helsinki Lontoo Newyork	Pirkkala Vantaa Standsted	Vantaa Gatwick Newyork	7:00 8:10 20:00	7:20 10:50 6:00	500 4200 6500	Finnair Finnair BA
Tampere	Newyork	2	1 2 3	Tampere Helsinki Lontoo	Helsinki Lontoo Newyork	Pirkkala Vantaa Standsted	Vantaa Stansted Newyork	7:00 15:55 20:00	7:20 18:20 6:00	500 4400 6500	Finnair BA BA
Tampere	Newyork	3	1 2 3 4	Tampere Helsinki Tukholma Lontoo	Helsinki Tukholma Lontoo Newyork	Pirkkala Helsinki Tukholma Standsted	Vantaa Tukholma Lontoo Newyork	7:00 10:00 12:20 20:00	7:20 10:50 14:20 6:00	500 3400 3600 6500	Finnair SAS SAS BA
Tampere	Tukholma	1	1 2	Tampere Helsinki	Helsinki Tukholma	Pirkkala Helsinki	Vantaa Tukholma	7:00 10:00	7:20 10:50	500 3400	Finnair SAS
...	...	...	...	...	...	...	...	...	...	...	...

Kuvio 5.S *Täydellisen transitiivisulkeuman tulosrelaatio. Kuvassa on vain osa tuloksesta. Lähtörelaation esiinkirjoitetusta osasta on tähän tulosrelaatioon kirjoitettu näkyviin Tampereelta lähtevät reitit.*

Tulosrelaatioon tulee jokaisesta polusta tieto sulkeuma-attribuuteista, eli tässä tapauksessa 'lähtöp' ja 'määräp'. Järjestelmä generoi ylimääräisen attribuutin 'pno' (path number), joka erottaa juoksevalla numerolla toisistaan samojen pisteiden väliset eri polut. Lisäksi järjestelmä generoi relaatioarvoisen attribuutin 'Path', joka sisältää täydelliset tiedot polun rakenteesta, eli kaikki polkuun kuuluvat särmät (yksittäiset lennot). Särmien järjestys on olennainen tieto, joten järjestelmä generoi 'Path'-alirelaatioon vielä lisäattribuutin 'ano' (arc number), joka ilmaisee juoksevalla numerosarjalla polun särmien järjestyksen.

Vastaava kysely laajennetulla relaatioalgebralla (katso kappale 4.2.1):

```
RA:       PATHS_määräp = NEXT lähtöp(Lennot);
```

Oletusarvoisesti transitiivisulkeumaa laskettaessa ei sallita syklejä reitin varrella, eli saman solmun läpi kulkemista useampaan kertaan. Tämä tarkistetaan primäärisulkeumaehdon attribuuttien perusteella. Edellisessä esimerkissämme attribuutilla 'lähtöp' ei saa olla samaa arvoa kahdella samaan polkuun kuuluvalla särmällä. Näin on mahdollista etsiä reitit jostakin pisteestä itseensä, mutta tällaista silmukkaa ei voi esiintyä reitin varrella. Tämä tehostaa huomattavasti transitiivisulkeuman laskentaa karsimalla pois reittejä, joista ei olla kiinnostuneita. Näitä reittejä ei aina edes saa esiintyä tuloksessa, sillä vaikka etsittäisiin huonointa reittiä, tarkoitetaan yleensä kuitenkin huonointa reittiä, joka ei sisällä syklejä. Jos kuitenkin halutaan sallia silmukat laskettaessa transitiivisulkeumaa syklisestä graafista, voidaan tämä sallia merkinnällä LOOPING closure-predikaatin perässä. Tämä voi olla tarpeellista, jos halutaan esimerkiksi etsiä reittejä, joiden on kuljettava tietyn välietapin kautta vaikka se edellyttäisi silmukan tekemistä.

```
NSQL:     CLOSURE LOOPING määräp = NEXT lähtöp OF Lennot;
```

RA: $PATHS_{määräp = NEXT\ lähtöp,\ LOOPS}$(Lennot);

Transitiivisulkeuman laskentaa voi helposti keventää tilankäytön osalta projisoimalla tarpeettomia attribuutteja pois transitiivisulkeuman laskennan yhteydessä. Varsinkin NF2-mallissa, jossa relaatiolla voi olla myös relaatioarvoisia attribuutteja, kannattaa karsia pois turhat alirelaatiohierarkiat, jottei niitä jouduta toistamaan 'Path'-alirelaatiossa. Vaikka seuraavassa kyselyssä lasketaan ensin koko transitiivisulkeuma ja projisoidaan sitten tulosrelaatiosta vain lähtö ja määräpaikka, ei kyselyä todellisuudessa näin kannata suorittaa, vaan kyselyn optimoija pyrkii projisoimaan lähtörelaatiota etukäteen.

```
NSQL:     (CLOSURE määräp = NEXT lähtöp OF Lennot)(lähtöp,määräp);
```

RA: $\pi_{lähtöp,määräp}$ ($PATHS_{määräp = NEXT\ lähtöp}$ (Lennot));

Closure

lähtöp	määräp
Tampere	Helsinki
Tampere	Lontoo
Tampere	Newyork
Tampere	Tukholma
Helsinki	Lontoo
...	...

Kuvio 5.T *Tulosrelaatio.*

Nyt tuloksena saadaan relaatio, josta nähdään vain minkä paikkakuntien väliltä löytyy lentoyhteys.

Koska NF2-mallissa pystytään 'Path'-listaa käsittelemään transitiivisulkeuman tuottamisen jälkeenkin, onnistuu reittien karsiminen ja polkujen aggregoiminen normaaleilla NSQL-kielen lauseilla ilman erikoisoperaatioita. Peräkkäisten särmien mukaan asetettavat ehdot ja polun särmien aggregointi sisällytetään kuitenkin CLOSURE-operaation sisään, koska:

1. peräkkäisten reittien käsittely vaatii reitin käsittelyä järjestettynä joukkona, eikä siten ole helppoa relaatiomallin puitteissa, vaan kannattaa suorittaa erikoisoperaattorin sisällä,
2. käyttäjälle on luonnollisempaa määritellä haluttujen reittien särmien ehtoja sulkeumaoperaatiota määritellessään, kuin karsia vääriä reittejä kaikkien reittien joukosta,
3. raskas transitiivisulkeuman laskentaoperaatio voidaan optimoida huomattavasti tehokkaammin, kun sulkeumaa tuotettaessa tiedetään mahdollisimman paljon ehtoja mukaan tulevista särmistä,
4. särmien aggregoinnin yhteydessä voidaan helposti asettaa aggregointia koskevia ehtoja ja ryhmittelyjä.

5.9.1. Valinta särmien tai solmujen perusteella

Primäärisulkeumaehdon lisäksi voimme luetella muitakin ehtoja polun särmille tai solmuille. Voimme suorittaa reittien karsimista yksittäisten särmien tai solmujen arvojen perusteella antamalla closure-lauseessa primäärisulkeumaehdon lisäksi lisäehtoja seuraavasti.

Esimerkki: Etsi kaikki kaupungit, jotka ovat yhteydessä toisiinsa lentoreitillä, joka ei kulje Helsingin kautta.

```
NSQL:     CLOSURE määräp = NEXT lähtöp
                  AND määräp ≠ 'Helsinki' OF Lennot;
```

RA: $PATHS_{(määräp = NEXT lähtöp)\ AND\ (määräp \neq 'Helsinki')}(Lennot);$

Esimerkkikyselyssä on pakollisen primäärisulkeumaehdon lisäksi määritelty, että särmän määränpää kaupunki ei saa olla Helsinki, jolloin reitti ei voi kulkea Helsingin kautta.

Polun särmiä voidaan karsia myös peräkkäisten särmien tietojen perusteella. Tällöin tarvitaan jälleen jo primäärisulkeuma ehdosta (= NEXT) tuttua NEXT-ilmausta, mutta nyt sen yhteyteen liitetään jokin muu vertailuoperaattori.

Esimerkki: Etsi kaikki kaupungit, joiden välillä on lentoyhteys siten, että välilaskuilla on vähintään tunti vaihtoaikaa. Edellisen lennon saapumisaika on siis vähintään tuntia ennen jatkolennon lähtöaikaa.

```
NSQL:     (CLOSURE määräp = NEXT lähtöp
                   AND tuloaika ≤ NEXT lähtöaika - 1
           OF Lennot);
```

RA: $PATHS_{(määräp = NEXT lähtöp)\ AND\ (tuloaika \leq NEXT\ lähtöaika - 1)}(Lennot);$

Transitiivisulkeumaa tuotettaessa voidaan NEXT-predikaatilla viitata peräkkäisiin särmiin ja asettaa siten ehtoja peräkkäisten särmien attribuuttien arvoille. Tämä ominaisuus sulkeuman laskennan yhteyteen sijoitettuna helpottaa kyselyiden formulointia huomattavasti, koska relaatiomalli ei muuten ole tehokas järjestetyn informaation käsittelyssä.

5.9.2. Valinta päätepisteiden perusteella

Seuraavaksi suoritetaan valinta polun päätepisteiden perusteella. Tämä operaatio on tavallinen valintaoperaatio, eikä kuulu transitiivisulkeumanlaskentaoperaation sisään. Transitiivisulkeumaoperaation valintaehdot kannattaa kuitenkin kirjoittaa aina esiin sulkeuman laskennan yhteyteen (myös apurelaatioita käytettäessä), koska näin sulkeuman laskenta tehostuu huomatavasti. Valintaehtoja sulkeuman laskennassa hyväksikäyttämällä optimoija pystyy yleensä jättämään kokonaan laskematta suurimman osan turhista reiteistä.

Esimerkki: Etsi lentoreitit Tampereen ja Lontoon välillä.

```
NSQL:     CLOSURE määräp = NEXT lähtöp OF Lennot
          WHERE lähtöp = 'Helsinki'
                AND määräp = 'Lontoo';
```

RA: $\sigma_{lähtöp = Helsinki,\ määräp = Lontoo}\ (PATHS_{määräp = NEXT\ lähtöp}(Lennot));$

Closure

lähtöp	määräp	pno	Path								
			ano	lähtöp	määräp	lasema	masema	lähtöaika	tuloaika	hinta	yhtiö
Helsinki	Lontoo	1	1	Helsinki	Lontoo	Vantaa	Gatwick	8:10	10:50	4200	Finnair
Helsinki	Lontoo	2	1	Helsinki	Lontoo	Vantaa	Stansted	15:55	18:20	4400	BA
Helsinki	Lontoo	3	1 2	Helsinki Tukholma	Tukholma Lontoo	Helsinki Tukholma	Tukholma Lontoo	10:00 12:20	10:50 14:20	3400 3600	SAS SAS
...	...	...	...	...	...	...	...	...	...	...	...

Kuvio 5.U *Tulosrelaatio lennoista Helsingistä Lontooseen, jolloin tulosjoukko on murto-osa täydellisestä sulkeumasta.*

5.9.3. Polun aggregointitiedon kerääminen

Polun aggregointi transitiivisulkeumaoperaation sisällä mahdollistaa kokoomatietojen keräämisen polulle polun särmäjoukosta. Tällainen tieto on esimerkiksi polun kokonaispituus. Polun aggregointitiedot määritellään closure-lauseen yhteydessä WITH-lauseella, jonka perässä kerrotaan uuden aggregointitiedon sisältävän attribuutin nimi ja määritellään aggregointioperaatiot ja aggregoitavat tiedot muodossa :

```
WITH <aggregated_attr_name>
     = <aggregate_operator>(Path.<arc_attribute_name>)
```

jossa <aggregated_attr_name> on uusi attribuutti, johon aggregointitieto kerätään, <aggregate_operator> on jokin aggregointifunktioista (MIN, MAX, FIRST, LAST, COUNT, SUM, AVG, PRODUCT). 'Path' on NSQL-kielessä varattu sana, ja viittaa closure-operaation luomaan polun särmät sisältävään 'Path'-alirelaatioon. <arc_attribute_name> on 'Path'-alirelaation attribuutin nimi, josta aggregointitieto kerätään.

Esimerkki: Mihin Helsingistä pääsee, ja millä kokonaishinnalla.

```
NSQL:     (CLOSURE määräp = NEXT lähtöp OF Lennot
           WITH kokhinta = SUM(Path.hinta)
           WHERE lähtöp = 'Helsinki')(määräp, kokhinta);
```

RA: $\sigma_{\text{lähtöp = Helsinki}}$ (yhthinta := $\text{SUM}_{\text{path.hinta}}$ (PATHS$_{\text{(määräp = NEXT lähtöp)}}$(Lennot)));

Closure

määräp	yhthinta
Lontoo	4200
Lontoo	4400
Newyork	10700
Tukholma	3400
...	...

Kuvio 5.V

Primäärisulkeuma-attribuutit tulevat mukaan automaattisesti transitiivisulkeuman tulosrelaation alimmalle relaatiotasolle ja kertovat reitin lähtö- ja päätepisteen. Usein on tarvetta saada mukaan myös muita attribuutteja päätepisteistä. Lisäämme transitiivisulkeuman yhteyteen aggregointifunktiot FIRST ja LAST, jotka palauttavat vastaavasti ensimmäisen ja viimeisen särmän attribuuttien arvoja.

Esimerkki: Mitä maksaa lento Tampereelta Lontooseen ja mikä on yhteyden lähtöaika Tampereelta ja tuloaika Lontooseen.

```
NSQL:    (CLOSURE määräp = NEXT lähtöp OF Lennot
          WITH  kokhinta = SUM(Path.hinta),
                lähtöaika = FIRST(Path.lähtöaika),
                tuloaika = LAST(Path.tuloaika)
          WHERE lähtöp = 'Tampere' AND määräp = 'Lontoo');
```

RA: $\sigma_{\text{lähtöp = Helsinki, määräp = Lontoo}}$ (kokhinta := $\text{SUM}_{\text{path.hinta}}$ (lähtöaika := $\text{FIRST}_{\text{path.lähtöaika}}$ (tuloaika := $\text{LAST}_{\text{Path.tuloaika}}$ (PATHS$_{\text{(määräp = NEXT lähtöp)}}$(Lennot)))));

Closure

lähtöp	määräp	kok hinta	lähtö aika	tulo aika	pno	Path					
						ano	lähtöp	määräp	lähtöaika	tuloaika	...
Tampere	Lontoo	4700	7:00	10:50	1	1 2	Tampere Helsinki	Helsinki Lontoo	7:00 8:10	7:20 10:50	... ...
Tampere	Lontoo	4900	7:00	18:20	2	1 2	Tampere Helsinki	Helsinki Lontoo	7:00 15:55	7:20 18:20	... ...
Tampere	Lontoo	7500	7:00	14:20	3	1 2 3	Tampere Helsinki Tukholma	Helsinki Tukholma Lontoo	7:00 10:00 12:20	7:20 10:50 14:20	... ...
...	...	...	...	...	...	...	...	...	...	...	...

Kuvio 5.W *Tulosrelaatio. (Relaatiosta puuttuu tilasyistä rivien lisäksi joitain Path-alirelaation attribuutteja).*

Polkua aggregoitaessa voidaan aggregointifunktiolle antaa ehto, jolloin aggregoinnissa huomioidaan vain ehdon täyttävät särmät. Ehto annetaan aggregointifunktion perässä WHERE-lauseella, jossa rajoitetaan polun aggregointiin osallistuvia särmiä.

Esimerkki: Etsi Helsinki-Kööpenhamina lennot ja kerro kokonaislentoaika, sekä erikseen lentoaika Finnairin koneilla ja SAS:n koneilla.

```
NSQL:    CLOSURE lähtöp = NEXT määräp OF Lennot
         WITH kokaika = SUM(Path.tuloaika - Path.lähtöaika),
              finaika = SUM(Path.tuloaika - Path.lähtöaika)
                        WHERE yhtiö = 'Finnair',
              sasaika = SUM(Path.tuloaika - Path.lähtöaika)
                        WHERE yhtiö = 'SAS'
         WHERE lähtöp = 'Helsinki'
           AND määräp = 'Kööpenhamina';
```

RA: $\sigma_{\text{lähtöp = Helsinki, määräp = Kööpenhamina}}$ (kokaika := $\text{SUM}_{\text{path.tuloaika - Path.lähtöaika}}$ (finaika := $\text{SUM}_{\text{(path.tuloaika - Path.lähtöaika), yhtiö = Finnair}}$ (sasaika := $\text{SUM}_{\text{(path.tuloaika - Path.lähtöaika), yhtiö = SAS}}$ (PATHS$_{\text{(määräp = NEXT lähtöp)}}$(Lennot)))));

5.9.4. Polkujoukkojen aggregointi

Kun halutut polut on löydetty, saatetaan olla kiinnostuneita yksittäisten polkujen sijasta polkujoukon yhteisistä ominaisuuksista tai ääriarvoista, jolloin täytyy kerätä aggregointitietoa polkujoukosta. Tämä tapahtuu tavallisella NSQL-kielen aggregointioperaatiolla closure-lauseen perässä.

Esimerkki: Etsi halvin lentoreitti Helsingistä Kööpenhaminaan.

```
NSQL:     (CLOSURE määräp = NEXT lähtöp OF Lennot
          WITH kokhinta = SUM(Path.hinta)
          WHERE lähtöp = 'Helsinki'
            AND määräp = 'Kööpenhamina')
          (lähtöp, määräp, MIN(kokhinta) AS minhinta);
```

RA: $\pi_{\text{lähtöp,määräp,minhinta}}(\text{minhinta} := \text{MIN}_{\text{kokhinta}}$
$(\sigma_{\text{lähtöp = Helsinki, määräp = Kööpenhamina}}$
$(\text{kokhinta} := \text{SUM}_{\text{path.hinta}}\ (\text{PATHS}_{\text{(määräp = NEXT lähtöp)}}(\text{Lennot}))));$

Closure

lähtöp	määräp	minhinta
Helsinki	Kööpenhamina	2100

Kuvio 5.X

Edellisessä esimerkissä etsittiin halvin lento tietyllä linjalla. Usein on tarvetta ryhmitellä polkuja ja laskea aggregointitietoa erikseen jokaiselle polkuryhmälle. Perinteisellä SQL-kielellä ryhmittely tapahtuisi "GROUP BY"-lauseella, mutta NF2-mallia käsittelevästä NSQL-kielestä olen jättänyt kyseisen lauseen pois tarpeettomana ja ryhmittely suoritetaan sensijaan relaatiokaaviota uudelleenmuotoilemalla.

Esimerkki: Etsi halvimmat lentoreitit Helsingistä muihin paikkoihin.

```
NSQL:     (CLOSURE Määräpt.määräp = NEXT lähtöp
           OF (Lennot(lähtöp, &Lennot(lähtöp, &Lennot(hinta)
                                              AS Hinnat)
                              AS Määräpt))
           WITH kokhinta = SUM(Path.hinta)
           WHERE lähtöp = 'Helsinki')
          (Määräpt.(määräp, MIN(Hinnat.kokhinta) AS minkokhinta));
```

Esimerkissä uudelleenmuotoillaan relaatio 'Lennot' ennen transitiivisulkeuman tuottamista, jolloin tulosrelaatio saadaan NF2-muotoon, katso kuva kuvio 5.y. Tulosrelaatio on nyt valmiiksi ryhmitelty lähtöpaikkakuntien mukaan ja lopuksi on helppoa etsiä reittien minimihinnat.

Answer1

lähtöp	Määräpt	
	määräp	Hinnat
		hinta
Tampere	Helsinki	500
Helsinki	Lontoo	4200 4400
	Tukholma	3400
Lontoo	Helsinki Newyork	3000 6500
Tukholma	Lontoo	3600
...	...	...

Kuvio 5.Y *Lähtörelaatio muotoilun jälkeen ennen transitiivisulkeuman laskemista.*

Closure

lähtöp	Määräpt						
	määräp	Hinnat					
		kokhinta	pno	Path			
				ano	lähtöp	määräp	hinta
Helsinki	Lontoo	4200	1	1	Helsinki	Lontoo	4200
		4400	2	1	Helsinki	Lontoo	4400
		7000	3	1 2	Helsinki Tukholma	Tukholma Lontoo	3400 3600
		...	...	...	...	...	...
	Tukholma	3400	1	1	Helsinki	Tukholma	3400
		...	...	...	...	...	...
	Newyork	10700	1	1 2	Helsinki Lontoo	Lontoo Newyork	4200 6500
		10900	2	1 2	Helsinki Lontoo	Lontoo Newyork	4400 6500
		13500	3	1 2 1	Helsinki Tukholma Lontoo	Tukholma Lontoo Newyork	3400 3600 6500
		...	...	...	...	...	...
	...	...	...	...	...	...	...

Kuvio 5.Z *Transitiivisulkeuman tulosrelaatio ennen polkujoukkojen aggregointia, jossa etsitään halvimmat polut eri määräpaikkoihin.*

Answer

määräp	minkokhinta
Lontoo	4200
Tukholma	3400
Newyork	10700
...	...

Kuvio 5.AA *Tulosrelaatio halvimmista kokonaishinnoista Helsingistä muihin kaupunkeihin.*

Seuraavassa sama kysely relaatioalgebralla. Algebralause on muotoiltu suoraan NSQL-kyselyn vaiheita seuraten ja vaatii siten huomattavaa optimointia.

RA: $\pi_{määräp,minkokhinta}$ (minkokhinta := $MIN_{Hinnat.kokhinta}$ ($\mu_{Määräpt}$ ($\nu_{Määräpt = \{määräp,Hinnat\}}$ ($\nu_{Hinnat = \{kokhinta,pno,Path\}}$ ($\sigma_{lähtöp = Helsinki}$ (kokhinta := $SUM_{path.hinta}$ (μ_{Hinta} ($\mu_{Määräpt}$ (PATHS$_{(Määräpt.määräp = NEXT\ lähtöp)}$ ($\nu_{Määräpt = \{määräp,Hinnat\}}$ ($\nu_{Hinnat = \{hinta\}}$ ($\pi_{lähtöp,määräp,hinta}$(Lennot)))))))))))));

Lopuksi voimme vielä valita vain tietyn ehdon täyttävät rivit lisäämällä ehtolauseen edellisen valinnan yhteyteen.

Esimerkki: Etsi halvimmat lentoreitit Helsingistä muihin paikkoihin. Lentoreitin yhteishinnan täytyy olla alle 3000 markkaa.

NSQL:

```
(CLOSURE Määräpt.määräp = NEXT lähtöp
 OF (Lennot(lähtöp, &Lennot(lähtöp, &Lennot(hinta)
                                      AS Hinnat)
                     AS Määräpt))
 WITH kokhinta = SUM(Path.hinta)
 WHERE lähtöp = 'Helsinki')
(Määräpt.(määräp, MIN(Hinnat.kokhinta) AS minkokhinta))
WHERE minkokhinta < 3000;
```

RA:

$$\pi_{\text{määräp,minkokhinta}}\ (\sigma_{\text{minhinta} < 3000}\ (\text{minkokhinta} := \text{MIN}_{\text{Hinnat.kokhinta}}\ (\mu_{\text{Määräpt}}\ (\nu_{\text{Määräpt} = \{\text{määräp,Hinnat}\}}\ (\nu_{\text{Hinnat} = \{\text{kokhinta,pno,Path}\}}\ (\sigma_{\text{lähtöp} = \text{Helsinki}}\ (\text{kokhinta} := \text{SUM}_{\text{path.hinta}}\ (\mu_{\text{Hinta}}\ (\mu_{\text{Määräpt}}\ (\text{PATHS}_{(\text{Määräpt.määräp} = \text{NEXT lähtöp})}\ (\nu_{\text{Määräpt} = \{\text{määräp,Hinnat}\}}\ (\nu_{\text{Hinnat} = \{\text{hinta}\}}\ (\pi_{\text{lähtöp,määräp,hinta}}(\text{Lennot}))))))))))))));$$

6. LIITOKSEN AUTOMATISOINTI: SUORAT OLIO-VIITTAUKSET ILMAN OLIOIDENTITEETTIÄ

6.1. ONGELMALLISIA SUHDETYYPPEJÄ

Ongelmana NF2-mallissa on joissain tapauksissa redundanssin kasvaminen upotettaessa tietoja alirelaatioihin. Pahin ongelma ovat many-to-many suhteet, josta hyvä esimerkki on oppilas-kurssi suhde, katso kuva kuvio 6.a. Sama oppilas on suorittanut useita kursseja ja vastaavasti tietyn kurssin on suorittanut suuri joukko oppilaita. Redundanssi, eli samojen arvojen useampaan kertaan tallentaminen, lisääntyy tällaisessa tapauksessa huomattavasti, jos kaikki kurssia koskevat tiedot upotetaan alirelaatioksi relaatioon 'Oppilas'. Tämä johtuu siitä, että kaikki kurssiin liittyvä informaatio jouduttaisiin toistamaan jokaisen kurssin suorittaneen oppilaan kohdalla. Samoin käy, jos oppilaan tiedot upotetaan alirelaatioksi relaatioon 'Kurssit'. Tällaisessa tapauksessa on siis luotava kaksi erillistä relaatiota. Tämä aiheuttaa sen, että myös NF2-mallia käytettäessä joudutaan kyselyissä tekemään liitoksia sellaistenkin relaatioiden välillä, joita tarvitaan usein yhdessä. NF2-mallissa liitosten tekeminen on käyttäjälle vielä vaikeammin hahmotettavissa oleva operaatio kuin tavallisessa relaatiomallissa. Varsinkin many-to-many suhteissa tietojen haku 'väärään suuntaan', eli esimerkissämme oppilaiden haku kursseittain, on peruskäyttäjälle vaikeaselkoista perinteisellä SQL:n tyyppisellä kielellä.

Oppilas

hetu	nimi	osoite	Kurssit		
			kuno	pvm	arvosana
111169-0123	erkki	kulmakatu 16	11 15 21	2.3.81 5.8.91 1.9.90	2+ 1+ 3
010171-1234	mira	kirkkokuja 6	11 31	4.5.91 2.2.95	3 2-
020255-2345	heikki	lokintaival 7	11	4.4.97	1+
030359-3456	sirkka	jokipolku 5	11 42	4.5.91 7.9.95	2 2-
040474-4567	paavo	ahmajotos 6	15	5.8.91	2+
050578-5678	heli	ahtotie 5			

Kurssi

kuno	Kunimi	Sisältö
11	sql perusteet	opetellaan...
15	johtaminen 1	uudet johtamis...
21	olioteknologia 1	peusteet olio...
31	corba 1	corba rajapin...
42	markkina tutk.	tilastollisten...

Kuvio 6.A *Oppilas - Kurssit suhde esitettynä NF2-relaationa.*

Ongelmat lisääntyvät entisestään silloin, kun tiedonhaun tehokkuus on hyvin tärkeää molempiin suuntiin many-to-many suhteessa. Esimerkin kuvio 6.a tapauksessa tietyn oppilaan suorittamat kurssit löytyvät tehokkaasti ja helposti, mutta jos pitäisi pystyä

yhtä tehokkaasti selvittämään, ketkä oppilaat ovat suorittaneet tietyn kurssin, ei tietokannan rakenne ole paras mahdollinen. Perinteinen 1NF-relaatiomallin mukainen esitystapa (kuvio 6.b) parantaa tehokkuutta kaksisuuntaisessa tiedonhaussa, mutta samalla menetetään NF2-mallin tuomat edut.

Oppilas

hetu	nimi	osoite
111169-0123	erkki	kulmakatu 16
010171-1234	mira	kirkkokuja 6
020255-2345	heikki	lokintaival 7
030359-3456	sirkka	jokipolku 5
040474-4567	paavo	ahmajotos 6
050578-5678	heli	ahtotie 5

Kurssit

kuno	Kunimi	Sisältö
11	sql perusteet	opetellaan...
15	johtaminen 1	uudet johtamis...
21	olioteknologia 1	peusteet olio...
31	corba 1	corba rajapin...
42	markkina tutk.	tilastollisten...

OSuoritukset

hetu	kuno
111169-0123	11
111169-0123	15
111169-0123	21
010171-1234	11
010171-1234	31
020255-2345	11
030359-3456	11
030359-3456	42
040474-4567	15

Kuvio 6.B *Oppilas - Kurssit suhde esitettynä perinteisesti 1NF-relaatioina.*

Toinen vastaavaan tehokkuusongelmaan johtava tapaus on, jos samaan objektityyppiin viitataan useista eri objekteista. Esimerkiksi Henkilö-relaatio voi olla tällainen. Yrityksessä henkilötiedot on usein luonnollisinta upottaa osaston sisälle, mutta jos henkilöön joudutaan usein viittaamaan esimerkiksi tauluista 'Ammattiyhdistys', 'Päivähoitokerho', 'Valtuusto', 'Palokunta', ei henkilötietojen hakeminen syvältä yritysrelaation sisältä ole yhtä tehokasta ja helppoa kuin erillisestä henkilötaulusta. Niinpä tällaiset tiedot voidaan joutua jossakin tapauksessa viemään erilliseen tauluun kuvan kuvio 6.c mukaisesti ja joudutaan näin suorittamaan yhä useammin liitosoperaatioita, myös NF2-mallissa.

Yritys

ono	onimi	sijainti	Työntekijät	
			tyno	esimies
10	tuotanto	tampere	14 25 65	0 14 25
20	markkinointi	helsinki	59	14
30	laskutus	tampere	34 78	1 34

Valtuusto

ryhmä	Valtuutetut	Projektit	
	tyno	pno	tehtävä
Talous	14 25	1 2	lainavakuuk... keskipitkän....
Kehitys	14 34	7	tuotannon...

Palokunta

ryhmä	onimi	sijainti	Palomiehet	
			tyno	esimies
1	tuotanto	tampere	34 25	 34
2	markkinointi laskutus	helsinki tampere	78 65	34 78

Työntekijät

tyno	tynimi	hetu	esimies	palkka	Kurssit		Lapset	
					kuno	pvm	lnimi	synt
14	erkki	111169-0123	0	24000	11 15 21	2.3.81 5.8.91 1.9.90	mari veli	93 90
25	mira	010171-1234	14	19000	11 31	4.5.91 2.2.95	jaakko	96
65	heikki	020255-2345	25	13000	11	4.4.97	viljo saara	89 90
59	sirkka	030359-3456	14	14000	11 42	4.5.91 7.9.95		
34	paavo	040471-4567	1	18000	15	5.8.91	kati ilona	82 84
78	heli	050578-5678	34	7000			sepe jope	91 95

Kuvio 6.C *Henkilötietoja joudutaan liittämään useisiin eri relaatioihin, jolloin niiden tallentaminen erilliseen relaatioon voi olla perusteltua.*

Ihanteellinen tapa esittää esimerkiksi yrityksen tiedot olisi luonnollisesti yksi NF2-relaatio 'Yritys', joka sisältää kaiken yrityksen toimintaan liittyvän informaation yleisimmin tarvittavassa suhdejärjestyksessä ilman mitään liitostarpeita. Käytännössä kuitenkin many-to-many suhteita täytyy usein pystyä tutkimaan tehokkaasti eri suuntiin ja objektit eivät ole aina hierarkisessa suhteessa toisiinsa vaan suhteet muodostavat monimutkaisen verkon, jota yksi NF2-relaatio ei voi esittää.

Esitän ongelman ratkaisuksi liitosoperaation automatisointia NSQL-kielellä NF2-mallissa. Tällä tavalla fyysisen NF2-mallin lisäksi pystyn esittämään virtuaalisia NF2-suhteita niissä tapauksissa, mistä tavallisen NF2-mallin keinoin ei selvitä hyvin. Tällä tavalla pystyn toteuttamaan myös many-to-many suhteiden haut kumpaankin suuntaan vähintään samalla tehokkuudella kuin perinteisessä relaatiomallissa ja suhteiden esittäminen käyttäjälle on yhtä intuitiivista kuin tavallisissa NF2-relaatioissa. Käyttäjälle liitosten tekeminen ei näy mitenkään, joten tietokanta on itse asiassa mahdollista rakentaa siten, että käyttäjälle koko tietokanta näyttää yhdeltä NF2-relaatiolta. Itse asiassa päästään vielä pidemmälle, koska relaatiokaavio voi nyt sisältää syklejä, eli relaatio voi sisältää attribuuttinaan itsensä. Tämä mahdollistaa myös rajoitetun transitiivisten suhteiden läpikäynnin halutulle tasolle ilman erikoisoperaattoria. Tässä virtuaalirelaatiossa pääsee suoraan kulkemaan kumpaankin suuntaan many-to-many suhteita, ja jopa silmukat ovat mahdollisia täysin intuitiivisesti. Itse asiassa tämä ominaisuus tuo relaatiomalliin samoja ominaisuuksia, jotka oliomallissa saavutetaan olioidentiteetin ja sen mahdollistamien suorien viittausten avulla. Arvoperustaisessa relaatiomallissa on kuitenkin mahdollista luoda deklaratiivinen kyselykieli toisin kuin oliopohjaisessa mallissa.

6.2. TIEDONMÄÄRITTELYKIELEN LAAJENTAMINEN JA ARVOJOUKKO-KÄSITE

Liitän automaattisen liitosoperaation arvojoukko-käsitteeseen (domain), joka relaatiomallissa on alusta saakka ollut mukana, vaikkei sitä kaupallisissa tietokantasovelluksissa ole kovin hyvin tuettu. Arvojoukko-käsite määrittää relaatiomallissa arvojoukon, josta attribuutin saavat arvojaan. Arvojoukko voi rajoittaa arvojoukon esimerkiksi kokonaisluvuiksi ja lisäksi esimerkiksi arvoalueeseen 1-90. Arvoaluetarkistuksen lisäksi arvojoukko-käsite antaa käsitteellisellä tasolla arvokasta tietoa siitä piirrettävistä attribuuteista, koska samasta arvojoukosta arvonsa saavien attribuuttien

tiedetään edustavat saman objektiluokan ilmentymiä. Lisäksi arvojoukon nimestä selviää mistä reaalimaailman objektityypistä on kysymys, joskin attribuuttien nimeämisellä pyritään samaan.

Teen joitakin lisäyksiä tiedonmäärittelykieleen, jotta saan käyttöön arvojoukko-käsitteen ja voin valmistella automaattisen liitoksen käyttöönottoa. Tiedonmäärittelykieltä tässä tutkimuksessa ei enempää käsitellä, mutta hyvä esitys NF2-mallin tiedonmäärittelykielestä löytyy esimerkiksi lähteestä [RoKoBa86]. Esitän tässä tutkimuksessa vain arvojoukko-määrittelyn ja relaationluontilauseessa tarvittavan lisäyksen tiedonmäärittelykieleen.

Lisätään tiedonmäärittelykieleen arvojoukon määrittelylause, jossa määritellään arvojoukon nimi ja sen tietotyyppi, sekä erilaisia arvoalue tarkistuksia. Esitän seuraavaksi arvojoukon määrittelyn muutaman esimerkin avulla.

```
CREATE DOMAIN HenkilöTunnus(
          DESCRIPTION("Suomen kansalaisen henkilötunnus"),
          Char(11),
          APPEARANCE(iiiiii-iiic),
          VALUECHECK(VALUE)
          );

CREATE DOMAIN RekisteriNumero(
          DESCRIPTION("Suomessa rekisteröidyn auton rekisterinumero")
          Char(7),
          APPEARANCE(ccc-iii)
          );

CREATE DOMAIN AutoMerkki(
          DESCRIPTION("Automerkki"),
          Char(20),
          SQLCHECK(VALUE IN KaikkiAutomerkit)
          );
```

'HenkilöTunnus' on arvojoukko, joka kertoo, että kyseessä on henkilötunnus ja lisäksi määrää arvon 11 merkkiä pitkäksi merkkijonoksi ja määrittelee ulkoasun sisältävän kuusi kokonaislukua väliviivan, kolme kokonaislukua ja yhden merkin. Lisäksi kutsutaan ulkoista ohjelmaa VALUECHECK, joka tarkistaa tarkistusmerkistä, että arvo on sallittu henkilötunnus. Arvojoukossa 'AutoMerkki' tarkistetaan lisäksi NSQL-lauseella, että arvo on jokin relaatiossa 'KaikkiAutomerkit' esiintyvistä arvoista. Erilaisia arvon tarkistuksia voidaan arvojoukon määrittelyyn sisällyttää enemmänkin, mutta en tässä keskity enempää niihin. Olennaista tämän tutkimuksen kannalta on, että meillä on nyt käytössä arvojoukko, joka kertoo käsitteellisellä tasolla, että kaikki tästä arvojoukosta arvonsa saavat attribuutit edustavat täsmälleen samaa objektijoukkoa. Kun esimerkiksi arvojoukko 'HenkilöTunnus' on määritelty ja määritellään tietokantaan uusi attribuutti, jonka arvojen kuuluu olla henkilötunnuksia, määritellään uuden attribuutin tyypiksi arvojoukko 'HenkilöTunnus'. Katso seuraavan esimerkin attribuuttia 'käyttöoikeus'.

```
CREATE TABLE Auto(
          merkki           AutoMerkki,
          rekno            RekisteriNumero,
          käyttöoikeus     HenkilöTunnus
          );
```

Attribuuttien tietotyyppi määritellään siis tietyn arvojoukon mukaiseksi, jolloin siihen liittyy automaattisesti jokin perustietotyypeistä (char, int,...) ja tietyt arvoalue tarkistukset, sekä käsitteellinen tieto siitä mikä reaalimaailman objekti on kyseessä. Relaatiosta 'Auto' tiedämme nyt, että se sisältää attribuuttiensa arvoina automerkin, rekisterinumeron ja käyttöoikeiden haltijan henkilötunnuksen, mitä ei attribuuttien nimistä voi varmasti tietää.

Nyt voimme määritellä mistä jonkin arvojoukon yksilöimät objektit löytyvät, eli missä relaatiossa esimerkiksi henkilötunnuksen yksilöimät objektit eli henkilöt sijaitsevat. Teemme tämän relaatiokaavion luonnin yhteydessä määreellä DOMAINMAINTABLE, joka määrittää tässä yhteydessä kerrotun arvojoukon yksilöimien objektien sijaitsevan tässä yhteydessä mainitussa taulussa.

```
CREATE TABLE Henkilö(
          nimi         Nimi,
          sukupuoli    Sex,
          hetu         HenkilöTunnus         DOMAINMAINTABLE
          );
```

Määre 'DOMAINMAINTABLE' arvojoukosta 'HenkilöTunnus' arvonsa saavan attribuutin 'hetu' perässä kertoo, että arvojoukon 'HenkilöTunnus' yksilöimät tiedot löytyvät relaatiosta 'Henkilö'. Nyt on mahdollista liittää automaattisesti kaikkialla muualla tietokannassa esiintyvät arvojoukon 'HenkilöTunnus' arvot relaatioon 'Henkilö'. Yhdelle arvojoukolle voidaan luonnollisesti määritellä vain yksi päärelaatio.

Vastaavasti relaatio 'Auto' sisältää arvojoukon 'RekisteriNumero' yksilöimät objektit, joten lisäämme relaatioon vastaavan määrittelyn.

```
CREATE TABLE Auto(
          merkki         AutoMerkki,
          rekno          RekisteriNumero    DOMAINMAINTABLE,
          käyttöoikeus   HenkilöTunnus
          );
```

Nyt voimme automaattisesti sijoittaa relaation 'Auto' atomisen attribuutin tilalle jos attribuutti saa arvonsa arvojoukosta 'RekisteriNumero'.

NSQL-kielen pistenotaatio mahdollistaa nyt automaattisen liitos- ja nest- operaatioiden käytön seuraavasti.

Esimerkki: Hae autojen käyttäjien nimet.

```
NSQL:     Auto.käyttöoikeus.nimi;
```

Kaikissa tietokannan relaatioissa, joissa esiintyy henkilötunnus, voimme kuvitella sen tilalla olevan relaatio Henkilö. Loppukäyttäjä näkee siis relaation 'Auto' seuraavanlaisena.

Auto

merkki	rekno	käyttöoikeus		
		nimi	sukupuoli	hetu
Audi	aaa-111	Mikko Pekka	m m	111155-111a 222261-222b
Samara	zzz-666	Pirkko Minna	n n	300369-333c 200473-444d
...	...	...	...	...

Kuvio 6.D

Edellisen esimerkin kyselyä, jossa haettiin vain nimi, vastaa seuraava SQL-kysely.

```
SQL:      SELECT nimi
          FROM Auto, Henkilö
          WHERE Auto.omistaja = Henkilö.Hetu;
```

Jos haluamme kaikki autojen käyttäjien henkilötiedot NF2-muodossa voimme käyttää apuna ilmausta ALL.

```
NSQL:     Auto(merkki, rekno, käyttöoikeus(ALL));
```

Tällöin saamme vastaukseksi kuvassa kuvio 6.d olevan relaation. Vastaavan relaation luominen perinteistä SQL:ää mukailevalla kielellä olisi

```
SQLX:     SELECT  Auto.merkki, Auto.rekno, Henkilö.käyttöoikeus
          FROM Auto, (NEST Henkilö ON nimi, sukupuoli
                      AS käyttöoikeus)
          WHERE Auto.käyttöoikeus = Henkilö.hetu;
```

Kun liitostaulu on määritelty arvojoukon määrittelyn yhteydessä, toimii liitos suoraan myös kyselyjen tulosrelaatioista, koska tulosrelaatioiden attribuutit perivät arvojoukkonsa lähtörelaatioista ja tieto arvojoukon päätaulusta on itse tietokannassa, eikä kyselyn tulosrelaatiossa. Kun ylläpitäjä tai käyttäjä luo uusia tauluja, joissa käytetään vaikkapa henkilötunnusta, toimii liitosoperaatio automaattisesti ilman mitään määrittelytarvetta, koska liitosehto on määrätty jo henkilötietotaulua luotaessa.

6.3. ESIMERKKEJÄ AUTOMAATTISESTA LIITOKSESTA NSQL-KIELELLÄ

6.3.1. Tavanomaiset liitokset - olioidentiteetin korvaus

Edellisen kappaleen esimerkeissä tehtiin tavallisia liitoksia ja relaatiokaavion uudelleenmuotoiluja. Tämä vastaa käyttäjän kannalta olioidentiteetin mahdollistamia suoria viittauksia. Näin myös relaatiomallissa on mahdollista viitata samaan objektiin suoraan useista eri objekteista aivan kuin viitattava objekti sisältyisi viittaavaan objektiin sen aliobjektina, tai relaatiomallin tapauksessa alirelaationa. Tällaista viittaustapaa NF2-mallikaan ei perusmuodossaan tarjoa, koska relaatio voi sisältyä alirelaationa vain yhteen päärelaatioon.

Kun lisäämme kuvan kuvio 3.a esimerkkitietokantaan arvojoukko-käsitteet, automatisoituu esimerkiksi viittaus alirelaation 'Yritys(Työntekijät(Kurssit))' attribuutista 'kuno' relaatioon 'Kurssi'.

Esimerkki: Anna osaston 10 työntekijöiden käymien kurssien nimet.

```
NSQL:     Yritys.Työntekijät.Kurssit.kuno.kunimi
          WHERE osasto = 10;
```

Esimerkki: Anna osastoittain työntekijät, jotka ovat suorittaneet kurssin, jonka kuvaukseen sisältyy sana 'sql'.

```
NSQL:     Yritys(onimi,(Työntekijät(tynimi))
          WHERE Yritys.Työntekijät.Kurssit.kuno.sisältö = "*sql*";
```

6.3.2. Transitiivisten suhteiden suoraviivainen tutkiminen

Liitoksen automatisoiminen mahdollistaa automaattisesti myös rajoittuneen transitiivisten suhteiden tutkimisen täysin intuitiivisesti ilman erikoisoperaattoria tai liitoksista huolehtimista.

Esimerkiksi kuvan kuvio 6.e työntekijärelaatiossa attribuutit 'tyno' ja 'esimies' edustavat samaa arvojoukkoa ja relaatio 'Työntekijät' on kyseisen arvojoukon päätaulu attribuutin 'tyno' mukaan. Relaatiokaavio on määritelty seuraavasti. (Oletamme kaikki arvojoukot määritellyiksi).

```
CREATE TABLE Työntekijät(
          tyno         TyöntekijäNumero      DOMAINMAINTABLE,
          tynimi       HenkilönNimi,
          hetu         HenkilöTunnus,
          ono          OsastoNumero,
          esimies      TyöntekijäNumero,
          palkka       Palkka
          );
```

Työntekijät

tyno	tynimi	hetu	ono	esimies	palkka
14	erkki	111169-0123	10	1	24000
25	mira	010171-1234	10	14	19000
65	heikki	020255-2345	10	25	13000
59	sirkka	030359-3456	20	14	14000
34	paavo	040474-4567	30	1	18000
78	heli	050578-5678	30	34	7000

Kuvio 6.E *Työntekijärelaatio.*

Nyt voimme etsiä Työntekijän esimiehen seuraavasti.

Esimerkki: Hae Miran esimiehen nimi.

```
NSQL:     Työntekijät.esimies.tynimi
          WHERE tynimi = mira;
```

Vastaukseksi saamme relaation

Answer

tynimi
erkki

Kuvio 6.F

Voimme yhtähelposti kulkea läpi useammankin tason transitiivisessa ketjussa. Otetaan esimerkkirelaatioksi relaatio 'Sukupuu'.

Sukupuu

hetu	nimi	sukup	Lapset
			lhetu
111169-0123	erkki	m	221289-111a 070791-222b
111168-333c	minna	n	221289-111a 070791-222b
030345-121a	Veikko	m	111169-0123
040422-444f	Marja	n	030345-121a 050544-212b 060651-332h
...	...	...	...

Kuvio 6.G *Relaatio sukupuu.*

Attribuutit 'hetu' ja 'lhetu' saavat nyt arvonsa samasta arvojoukosta ja 'Sukupuu' on arvojoukon päärelaatio. Voimme nyt osoittaa suoraan lapseen, lapsenlapseen tai lapsenlapsenlapseen eli osoittaa suoraan läpi usean transitiivisen tason.

Esimerkki: Hae Marjan lapsenlapsenlapsien nimet

```
NSQL:     Sukupuu.Lapset.lhetu.Lapset.lhetu.Lapset.lhetu.nimi
          WHERE nimi = 'marja';
```

Kyselyn tulkinta voi olla helpompaa, jos käytetään apuna sulkeita transitiivisuuden tasojen erottelemiseen.

```
NSQL:     Sukupuu.(Lapset.lhetu).(Lapset.lhetu).(Lapset.lhetu).nimi
          WHERE nimi = 'marja';
```

Jos halutaan tutkia koko sukupuu tai halutaan aggregointitietoa, tarvitaan luonnollisesti CLOSURE-operaatiota, mutta automaattisen liitosoperaation kautta saadaan hyvin helposti vastaus joihinkin triviaaleihin kysymyksiin transitiivisista suhteista.

6.3.3. Many-to-many suhteiden kulkeminen kahteen suuntaan

Automaattisen liitoksen avulla päästään many-to-many-suhteissa kulkemaan kätevästi siihen suuntaan, mikä tietokantakaaviossa on määritetty yleensä tarvittavaksi. Esimerkkirelaation kuvio 6.a oppilas-kurssi suhteessa päästään automaattisesti tutkimaan kurssin oppilaita, mutta toisinpäin liitoksen automatisointi ei toimi, vaan joudutaan turvautumaan vaikeaselkoiseen relatiokaavion uudelleenmuotoiluoperaatioon ja itse määriteltyyn liitokseen. Jotta pystymme kulkemaan suhdetta yhtähelposti myös toiseen suuntaan, lisäämme myös kurssiin tiedon kurssin suorittaneesta oppilaasta seuraavasti.

Oppilas

hetu	nimi	osoite	Kurssit		
			kuno	pvm	arvosana
111169-0123	erkki	kulmakatu 16	11 15 21	2.3.81 5.8.91 1.9.90	2+ 1+ 3
010171-1234	mira	kirkkokuja 6	11 31	4.5.91 2.2.95	3 2-
020255-2345	heikki	lokintaival 7	11	4.4.97	1+
030359-3456	sirkka	jokipolku 5	11 42	4.5.91 7.9.95	2 2-
040474-4567	paavo	ahmajotos 6	15	5.8.91	2+
050578-5678	heli	ahtotie 5			

Kurssi

kuno	kunimi	sisältö	Suorittaneet
			hetu
11	sql perusteet	opetellaan...	111169-0123 010171-1234 020255-2345 030359-3456
15	johtaminen 1	uudet johtamis...	111169-0123 040474-4567
21	olioteknologia 1	peusteet olio...	111169-0123
31	corba 1	corba rajapin...	010171-1234
42	markkina tutk.	tilastollisten...	030359-3456

Kuvio 6.H *Oppilas - Kurssit suhde esitettynä NF2-relaationa siten, että myös kurs-seille löydetään suoraviivaisesti kurssin suorittaneet oppilaat.*

Relaatioiden luontilauseet ja niihin sisältyvät arvojoukkojen päätaulumäärittelyt:

```
CREATE TABLE Oppilas(
          hetu         HenkilöTunnus          DOMAINMAINTABLE,
          nimi         HenkilönNimi,
          osoite       Osoite,
          Kurssit(
              kuno           KurssiNumero,
              pvm            Päivämäärä,
              arvosana       KurssinArvosana
              )
          );

CREATE TABLE Kurssi(
          kuno         KurssiNumero           DOMAINMAINTABLE,
          kunimi       KurssiNimi,
          sisältö      KurssinKuvaus
          Suorittaneet(
              oppilas         HenkilöTunnus
              )
          );
```

Huonona puolena tässä lisäyksessä on, että redundanssi kasvaa ja tallennusoperaatio tulee raskaammaksi, koska oppilaalle kurssisuoritusta lisättäessä joudutaan lisäksi viemään oppilaan henkilötunnus 'Kurssit'-relaatioon. Jos tietojen tutkiminen molempiin suuntiin many-to-many suhdetta on yleistä ja jos kirjoitusoperaatio vielä on harvinainen lukuoperaatioon verrattuna, on tämän lisäyksen tekeminen kuitenkin kan-

nattavaa. Tietokannassa tulee nyt relaatioon 'Oppilas.Kurssit' lisätä tapahtumaan reagoiva laukaisin, joka suorittaa automaattisesti lisäykset, poistot ja päivitykset relaatioon 'Kurssit.Suorittaneet' ja estää kokonaan relaation 'Kurssit.Suorittaneet' modifiointi muuta kautta. Näiden lisäysten jälkeen on tiedonhaku todella helppoa myös toiseen suuntaan many-to-many-suhdetta.

Esimerkki: Hae kurssin 'sql perusteet' suorittaneiden oppilaiden nimet.

```
NSQL:     Kurssi.Suorittaneet.oppilas.nimi
          WHERE kunimi = 'sql perusteet';
```

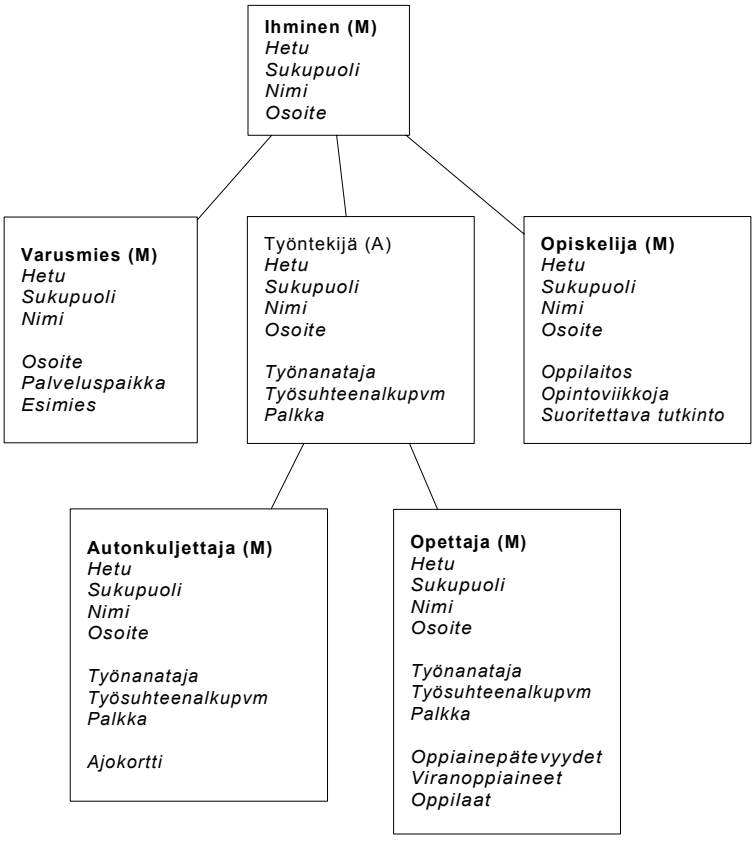
Voimme nyt kulkea suhdetta myös erisuuntiin täysin vapaasti, jota hyödynnämme seuraavassa esimerkissä.

Esimerkki: Hae kurssin 'sql perusteet' suorittaneiden oppilaiden kaikki suoritetut kurssit.

```
NSQL:     Kurssi.Suorittaneet.oppilas.Kurssit.kuno.kunimi
          WHERE kunimi = 'sql perusteet';
```

7. IS-A SUHTEIDEN ELI LUOKKAHIERARKIAN LISÄÄMINEN LAAJENNETTUUN RELAATIOMALLIIN

Perinteinen relaatiomalli ei kykene IS-A suhteiden esittämiseen. Olio-ohjelmointikielissä ja siten myös olioperustaisessa tietokantamallissa IS-A suhteiden esittäminen toteutetaan luokkahierarkian avulla perimispuun kautta. IS-A suhteen avulla voidaan jokin objektiryhmä jaotella pienempiin ryhmiin antaen aliryhmille lisää ominaisuuksia, jotka ovat vain tietylle aliryhmälle ominaisia. IS-A suhdetta käyttäen voidaan kaikkien aliryhmien objekteihin kuitenkin viitata kerralla yliryhmän kautta niin kauan kuin käsitellään vain kaikkien ryhmien yhteisiä ominaisuuksia. Esimerkiksi ihmiset voidaan jaotella ryhmiin ammateittain (postinkantaja, autonkuljettaja,...) ja määritellä ammattikohtaisia ominaisuuksia. Jokainen ammattilainen on (is-a) kuitenkin myös ihminen ja yliluokassa 'Ihminen' on määritelty eri aliluokkien yhteiset ominaisuudet, eli tässä tapauksessa kaikkien ihmisten yhteiset ominaisuudet. Jos käsitellään vain ihmisten yhteisiä ominaisuuksia, voidaan IS-A suhteen avulla viitata kerralla kaikkien ammattiryhmien ihmisiin. Laajennettuun relaatiomalliin IS-A suhteet on helppoa lisätä nyt kun käytettävissä on transitiivisulkeuman laskentaoperaatio.



Kuvio 7.A *IS-A hierarkia.*

Kuvassa kuvio 7.a on IS-A hierarkia ihmisestä. Kaikilla ihmisillä on attribuutit 'hetu', 'sukupuoli', 'nimi' ja 'osoite', siis myös varusmiehillä, autonkuljettajilla jne, koska nekin ovat ihmisiä IS-A hierarkian mukaisesti. Oliomallissa IS-A hierarkian objektit ovat olioluokkia, mutta relaatiomallissa ne voidaan toteuttaa suoraan relaationa. Kutsun hierarkiassa alempana olevia relaatioita ylempänä olevien aliluokkarelaatioiksi erottaakseni ne NF2-mallin osasuhteessa käytettävästä alirelaatiokäsitteestä. Ylempänä hierarkiassa oleva relaatio on alempana olevan yliluokkarelaatio. Jos esimerkissämme ihminen on varusmies, työntekijä tai opiskelija, ei häntä tallenneta relaatioon 'Ihminen' vaan esimerkiksi opiskelija tallennettaisiin relaatioon 'Opiskelija', joka on relaation 'Ihminen' aliluokkarelaatio. Aliluokkarelaatioissa on kaikki samat attribuutit kuin yliluokkarelaatiossakin ja lisäksi joitain uusia attribuutteja, jotka kuvaavat aliluokkarelaation objektien niitä ominaisuuksia mitä yliluokkarelaatioilla ei ole. Aliluokkarelaation objektijoukko on siis erikoistus yliluokkarelaation objektijoukosta. Kuviossa relaation nimen perässä oleva kirjain A tarkoittaa, että kyseessä on abstrakti relaatio, eikä se voi sisältää arvoja. Abstrakti relaatio vain määrittelee joitain attribuutteja aliluokkarelaatioidensa käyttöön ja antaa mahdollisuuden viitata aliluokkarelaatioihin ryhmänä. Kirjain M tarkoittaa, että relaatioon voi tallentaa arvoja.

Jos haluamme käsitellä opettajia, voimme tehdä kyselyn relaatioon 'Opettaja' ja käsitellä vain kyseiseen relaatioon tallennettuja objekteja. Jos sen sijaan haluaisimme käsitellä kaikkia ihmisiä, kohdistaisimme kyselymme relaatioon 'Ihminen', tällöin tietokannanhallintajärjestelmämme kohdistaisi kyselyn automaattisesti relaation 'Ihminen' lisäksi kaikkiin sen aliluokkarelaatioihin. Tällaisessa kyselyssä voimme käyttää vain relaation 'Ihminen' attribuutteja, koska ne ovat yhteisiä kaikille sen aliluokkarelaatioille.

7.1. IS-A SUHTEIDEN TOTEUTUS LAAJENNETUSSA RELAATIOMALLISSA

IS-A suhteiden lisäämisen vaatimat muutokset laajennettuun relaatiomalliin ovat varsin vähäisiä. IS-A suhteiden lisäys voidaan tehdä lähes täysin laajennetun relaatiomallin omia ominaisuuksia hyödyntäen. Tiedon määrittelykieleen vaaditaan pieni lisäys ja tiedonhallintajärjestelmän vastuulle lisätään IS-A hierarkiasta huolehtiminen. Kyselyn optimointia on luonnollisesti mietittävä uudelleen, jotta IS-A hierarkiaan kohdistuvista kyselyistä saadaan mahdollisimman tehokkaita, mutta mitään varsinaisesti uusia optimointiongelmia ei tule koska IS-A hierarkian mukaan ottaminen tapahtuu union-operaatiolla, jonka optimoinnista mainittiinkin jo NSQL-kielen joukko-operaatioita määriteltäessä.

Määrittelen aluksi lisäykset tiedonmäärittelykieleen. Määrittelen luontilauseet osalle esimerkkikuvan IS-A hierarkian relaatioista esimerkissä esimerkki 7.a. Uuden relaation yliluokkarelaatio kerrotaan relaationluontilauseen perässä määreellä SUPERCLASS. Luotava relaatio perii automaattisesti kaikki attribuutit yliluokkarelaatioltaan, joten perittäviä attribuutteja ei tarvitse määritellä uudestaan jokaisen aliluokkarelaation kohdalla. Uuden relaation määrittelyn yhteydessä kerrotaan vain attribuutit jotka uuteen relaation halutaan yliluokkarelaation attribuuttien lisäksi. CREATE TABLE lauseen perässä voidaan käyttää ilmausta ABSTRACT, joka määrää uuden relaation abstraktiksi relaatioksi, johon ei voi tallentaa arvoja. Tällaisen relaation tehtävä on määritellä joitain aliluokkarelaatioiden yhteisiä ominaisuuksia ja mahdollistaa kaikkiin aliluokkarelaatioihin viittaaminen kerralla. Esimerkiksi relaatio 'Työntekijä' on tällainen. Nyt voimme viitata kaikkiin työntekijöihin kerralla.

Emme kuitenkaan halua ilmentymiä relaatioon 'Työntekijä', koska jokaisen työntekijän täytyy kuulua johonkin ammattiryhmään, ja siten se tulee tallentaa kyseisen ammattiryhmän relaatioon. Ellei relaatiolle ole määritelty mitään yliluokkarelaatiota, sijaitsee relaatio ylimmällä tasolla IS-A hierarkiassa tai sitten se ei kuulu mihinkään IS-A hierarkiaan.

```
CREATE TABLE Ihminen(
        hetu            HenkilöTunnus,
        sukupuoli       Sex,
        nimi            HenkilönNimi,
        osoite          Osoite
        );

CREATE TABLE ABSTRACT Työntekijä(
        työnantaja              TyönantajanNimi,
        työsuhteenalkupvm       Päivämäärä,
        palkka                  Palkka
        ) SUPERCLASS Ihminen;

CREATE TABLE Opettaja(
        oppiainepätevyydet      Oppiaine,
        viranoppiaineet         Oppiaine,
        oppilaat                Henkilötunnus
        ) SUPERCLASS Työntekijä;
```

Esimerkki 7.A *IS-A suhteiden määrittely tiedonmäärittelykielessä.*

IS-A suhteessa olevilla relaatioilla on oltava sama pääavain, koska niihin tallennetaan saman objektityypin ilmentymiä, eikä niitä siten voi olla kuin yksi koko IS-A hierarkiassa. Siten tiedonhallintajärjestelmän on ylläpidettävä IS-A hierarkiassa oleville relaatioille yhteistä pääavainindeksiä. Esimerkkimme IS-A hierarkiassa 'hetu' on rivin koko hierarkiassa yksilöivä avain.

Kun relaation relaatiokaaviota muutetaan, tekee järjestelmä automaattisesti samat muutokset kaikkiin muutetun relaation aliluokkarelaatioihinkin. Jos lisätään attribuutti relaatioon, lisätään vastaava attribuutti myös kaikkiin tämän relaation alapuolella IS-A-puussa sijaitseviin relaatioihin ja jos poistetaan attribuutti, poistetaan se samaten koko aliluokkarelaatiopuusta.

IS-A suhteen ylläpito käytännössä edellyttää yhden lisärelaation perustamista tiedonhallintajärjestelmän systeemitauluihin. Nimeän IS-A-hierarkiaa ylläpitävän systeemitaulun 'Classhierarchy'-relaatioksi. Aina kun luodaan uusi relaatio, jolle on määritelty SUPERCLASS-määrettä käyttäen yliluokkarelaatio, viedään 'Classhierarchy'-relaatioon tieto tästä. Relaatio 'Classhierarchy' on transitiivirelaatio, joka sisältää yliluokkarelaatio-aliluokkarelaatio pareja. Esimerkkimme IS-A-hierarkian 'Classhierarchy'-relaatio on esitetty kuvassa kuvio 7.b.

Classhierarchy

Super	Subs
	sub
Ihminen	Varusmies Työntekijä Opiskelija
Työntekijä	Autonkuljettaja Opettaja

Kuvio 7.B *Systeemirelaatio joka kuvaa IS-A suhteet.*

Nyt luokkahierarkian käyttö kyselyissä on hyvin suoraviivaista. Kun johonkin relaatioon kohdistetaan kysely, laskee järjestelmä transitiivisulkeuman systeemirelaatiosta 'Classhierarchy' ja löydettyään kaikki kyselyssä käytetyn relaation suorat ja epäsuorat aliluokkarelaatiot, kohdistaa saman kyselyn kaikkiin aliluokkarelaatioihin. Kyselyjen tulokset liitetään lopuksi toisiinsa union-operaatiolla. Havainnollistan IS-A hierarkian käyttöä seuraavilla esimerkeillä.

Esimerkki: Hae kaikkien naisopettajien nimet, osoitteet ja oppiainepätevyydet.

```
NSQL:     Opettaja(nimi, osoite, oppiainepätevyydet)
          WHERE sukupuoli = 'n';
```

RA: $\pi_{\text{nimi, osoite}}\ (\sigma_{\text{sukupuoli = 'n'}}\ (\text{Opettaja}))$;

Transitiivisulkeuma $\sigma_{\text{super = Opettaja}}\ \text{CLOSURE}_{\text{Subs.sub = NEXT super}}$ (Classhierarchy) palauttaa tyhjän vastauksen, joten kysely suoritettiin vain relaatioon 'Opettaja'.

Esimerkki: Hae kaikkien naisten nimet ja osoitteet.

```
NSQL:     Ihminen(nimi, osoite)
          WHERE sukupuoli = 'n';
```

RA: $\pi_{\text{nimi, osoite}}\ (\sigma_{\text{sukupuoli = 'n'}}\ (\text{ihminen}))$

Nyt järjestelmän generoima transitiivisulkeuma

$$\sigma_{\text{super = Ihminen}}\ \text{CLOSURE}_{\text{Subs.sub = NEXT super}}\ (\text{Classhierarchy});$$

palauttaa vastaukseksi seuraavat relaatioiden nimet.

Answer

sub
Varusmies
Työntekijä
Opiskelija
Autonkuljettaja
Opettaja

Kuvio 7.C *Vastausrelaatio tiedonhallintajärjestelmän tekemään IS-A suhteiden selvityskyselyyn.*

Järjestelmä generoi nyt kyselyn relaation 'Ihminen' lisäksi kaikkiin kuvan kuvio 7.c materialisoituihin relaatioihin, eli kaikkiin relaation 'Ihminen' materialisoituihin aliluokkarelaatioihin. Vastaukset liitetään toisiinsa union-operaatiolla seuraavasti.

```
RA:     π nimi, osoite (σ sukupuoli = 'n' (ihminen))
        UNION π nimi, osoite (σ sukupuoli = 'n' (Ihminen))
        UNION π nimi, osoite (σ sukupuoli = 'n' (Varusmies))
        UNION π nimi, osoite (σ sukupuoli = 'n' (Opiskelija))
        UNION π nimi, osoite (σ sukupuoli = 'n' (Autonkuljettaja))
        UNION π nimi, osoite (σ sukupuoli = 'n' (Opettaja));
```

Esimerkissämme käytimme 1NF-relaatioita, mutta IS-A suhteet toimivat aivan samoin myös NF2-relaatioille. Mikäli kyselyssä valitaan relaation kaikki attribuutit tai NF2-relaation kyseessä ollessa koko alirelaatio, lisää tiedonhallintajärjestelmä automaattisesti kyselyyn projektio-operaation, joka rajoittaa kyselyn vain niihin attribuutteihin, jotka sijaitsevat siinä relaatiossa, johon kysely ensisijaisesti kohdistetiin.

Esimerkki: Hae ihmiset.

```
NSQL:   Ihminen
```

Järjestelmän IS-A hierarkian perusteella generoima RA-kysely on:

```
RA:     Ihminen
        UNION π hetu,sukupuoli,nimi,osoite (Ihminen)
        UNION π hetu,sukupuoli,nimi,osoite (Varusmies)
        UNION π hetu,sukupuoli,nimi,osoite (Opiskelija)
        UNION π hetu,sukupuoli,nimi,osoite (Autonkuljettaja)
        UNION π hetu,sukupuoli,nimi,osoite (Opettaja);
```

Mitä hyvänsä relaatioalgebran lausetta yliluokkarelaatioon kohdistettaessa, suoritetaan operaatio koko aliluokkarelaatiopuuhun. Tämä koskee siten myös closure-operaatiota, joka laskee transitiivisulkeuman tällöin koko aliluokkarelaatiopuusta.

8. LAAJENNETUN RELAATIOMALLIN ARVIOINTIA JA VERTAILUA MUIHIN TIETOKANTAMALLEIHIN

8.1. KYSELYKIELTEN ILMAISUVOIMA

Laajennettu relaatiomalli on saanut lisää ilmaisuvoimaa transitiivisulkeumaoperaation ansiota ja pystyy siten paremmin pitämään paikkansa taistelussa tietokantaparadigmojen paremmuudesta. Oliomalleja vastaan laajennettu relaatiomalli on varsin vahvoilla, koska se säilyttää arvoperustaisuuden ja deklaratiivisen kyselykielen. Laajennetunkaan relaatiomallin ilmaisuvoima ei kuitenkaan ole sama kuin perinteisten ohjelmointikielten, jotka pystyvät Turingin koneen ilmaisuvoimaan. Tulevaisuuden ratkaisuna tietokantakieleksi ovatkin hyvin vahvoilla logiikkaan perustuvat kielet, joissa yhdistyy deklaratiivisuus ja Turingin koneen ilmaisuvoima. Nykyisiä johtavia tietokantamalleja vertaillen voidaan sanoa, että tällöin yhdistyvät oliomallin kyselykielen ilmaisuvoima ja relaatiomallin kielten deklaratiivisuus. Huomionarvoinen yksityiskohta tulevaisuutta ajatellen on, että logiikkaohjelmointikielet ovat relationaalisia kieliä, joten relaatiotietokannat on huomattavasti helpompi muuntaa Turingin koneen ilmaisuvoimaisen tiedonhallintakielen omaaviksi logiikkatietokannoiksi kuin esimerkiksi oliotietokannoiksi. Saattaa olla mahdollista jopa lisätä logiikkaohjelmointikieli suoraan relaatiotietokantaparadigmaan, jolloin vanhojen tietokantojen rakennetta ei tarvitse muuttaa lainkaan. Eräs yritys yhdistää relaatiotietokantamalli ja logiikkaohjelmointikieli on LDL-järjestelmä (Logic Data Language) [ChiGam90].

Ongelmana Turingin koneen ilmaisuvoiman ja kyselykielen helppokäyttöisyyden yhdistämisessä on, että täyden ilmaisuvoiman omaava kieli tulee väistämättä niin monimutkaiseksi, että vaikka se periaatteessa olisikin deklaratiivinen ei sen hallitseminen ole kuitenkaan helppoa, eikä missään tapauksessa loppukäyttäjän taitojen ulottuvissa.

Yleisin logiikkaohjelmointikieli Prolog paljastaa logiikkaohjelmoinnin nykyiset heikkoudet. Nykyiset tietokoneet eivät pysty tehokkaasti ajamaan täysin deklaratiivisia aitoja logiikkaohjelmia. Niinpä logiikkaohjelmointikielet, kuten Prolog, eivät ole deklaratiivisia, vaan niiden ohjelmilla on tarkkaan määrätty suoritusjärjestys, ja logiikkaohjelman teko vaatii siten vahvaa ohjelmointitaitoa. Logiikkaohjelmointikielten nykyisten deklaratiivisesti rajoittuneiden toteutustenkaan tehokkuus ei yllä lähellekään perinteisiä ohjelmointikieliä, joten tämäkin aiheuttaa omat rajoitteensa kielten yleistymiselle.

Tulevaisuudessa tekniikan kehitys kuitenkin ratkaissee nämä ongelmat, ja jo juuri tulossa oleva keskusmuistitietokantoihin siirtyminen tasoittaa tehokkuusongelmaa logiikkaohjelmoinnin hyväksi. Aidosti deklaratiivisen logiikkaohjelmointikielen kehittäminen vaatii vielä huimaa kehitystä tekniikankin puolella, mutta viimeistään DNA-tietokoneet tai kvanttitietokoneet mahdollistanevat sellaisen luomisen. Aidosti deklaratiivista logiikkaohjelmointikieltä ei siis aivan heti pystytä kehittämään. Uskonkin, että lähitulevaisuuden yleiskäyttöisessä tietokantamallissa on logiikkaan perustuva melko deklaratiivinen logiikkaohjelmointikieli ja lisäksi yksinkertainen täysin dekla-

ratiivinen tiedonhallintakieli, jolla yleisimmät 'ad hoc' -kyselytarpeet voidaan tyydyttää.

8.2. TIETOKANTAOBJEKTIEN KÄSITTELY

NF2-malli mahdollistaa rakenteellisten objektien esittämisen relaatiotietokannassa ja transitiivisulkeumaoperaation avulla saadaan relaatiomalliin lisättyä myös IS-A suhteiden esittäminen, joten relaatiomallin käytettävyys paranee oleellisesti näiden yleensä oliomalliin liitettävien piirteiden myötä. Kun automaattisella liitosoperaatiolla vielä saadaan käyttäjän ulottuville oliomallista tuttu suoraviivainen viittaaminen toiseen objektiin, kaventuu oliomallin etumatka varsin pieneksi. Kun relaatiomalli vielä mahdollistaa tietokantaoperaatioiden ketjuttamisen toisiinsa yksinkertaisten tietotyyppiensä ansiosta ja tarjoaa käyttöön tehokkaat optimointitekniikat ja yleiskäyttöisten tietokantojen tehokkaan hallinnan, vaikuttaa tämä paradigma hyvin vahvalta.

Oliotietokantaparadigmalle jää vielä joitain hyödyllisiä ominaisuuksia joihin relaatiomalli ei pysty vastaamaan. Oliotietokantojen olioidentiteetti, abstraktit tietotyypit ja tiedon kapselointiominaisuus eivät ole helposti lisättävissä relaatioparadigmaan sen hyötyjä menettämättä, joten olioparadigma on näitä ominaisuuksia vaativilla erityisaloilla varsin vahva paradigma myös tietokantaparadigmana. Esimerkiksi IS-A suhteiden täydellinen hyödyntäminen vaatisi tiedon kapselointia ja polymorfismia, jotta erityyppisiä objekteja voisi käsitellä tehokkaasti yhdessä. Oliotietokantoihin integroidut tehokkaat oliokielet antavat oliotietokantamallille etulyöntiaseman joillakin äärimmäistä tehokkuutta vaativilla sovellusaloilla, kuten teletekniikassa. Oletettavaa onkin, että sekä olio-, että relaatioparadigmoille on markkinansa, joskin yleiskäyttöisissä tietokannoissa relaatiomalli vaikuttaa laajennettuna varsin vahvalta. Erilaisiin laajennettuihin relaatiomalleihin on jo lisätty, ja tullaan jatkossa yhä enemmän lisäämään myös näitä oliomallin relaatiomalliin huonosti sopivia ominaisuuksia, jolloin eri paradigmojen parhaita puolia saadaan liitettyä yhteen. Kutsutaanko tällaista hybridi-mallia sitten relaatiomalliksi vai oliomalliksi on makuasia.

9. YHTEENVETO

9.1. TULOKSET

Olen tässä tutkimuksessa laajentanut relaatiomallia mallin ilmaisuvoiman lisäämiseksi, koska se on osoittautunut puutteelliseksi varsinkin uusilla sovellusaloilla. Tällä hetkellä varteenotettavin haastaja relaatiomallille on oliotietokantamalli, joten olen tutkimuksen edetessä verrannut laajennettua relaatiomallia erityisesti oliotietokantamalliin ja pyrkinyt tuomaan relaatiomalliin niitä ominaisuuksia, jotka ovat osoittautuneet käyttökelpoisiksi oliomallin yhteydessä. Olen laajentanut relaatiomallin NF2-tietokantamalliksi, jolloin mahdollistuu rakenteellisesti mutkikkaiden objektien esittäminen relaatiomallissa. Seuraavaksi olen lisännyt relaatiomalliin aggregoivan transitiivisulkeumanlaskentaoperaation lisäämällä relaatioalgebraan uuden operaation PATHS, joka hoitaa kyseisen tehtävän. Transitiivisulkeuman laskenta on integroitu täysin relaatiomalliin, joten PATHS-operaatiota voi yhdistellä vapaasti muihin relaatioalgebra-operaatioihin. Lisäetuna aiempiin esityksiin transitiivisen käsittelyn lisäämisestä relaatiomalliin on, että lisättäessä transitiivinen käsittely NF2-malliin olen voinut antaa koko transitiivisen polun reitti-informaation käytettäväksi kyselyn tulosrelaatiossa ja saanut täten vielä lisää ilmaisuvoimaa operaatioon.

Olen kehittänyt laajennetulle relaatiomallille uuden suoraviittauksisen kyselykielen NSQL, jolla kyselyiden tekeminen NF2-mallista on huomattavasti helpompaa kuin perinteiseen SQL-kieleen perustuvilla kyselykielillä, joiden käyttämistä sisäkkäisistä kyselyistä tulee hyvin vaikeaselkoisia. NSQL-kielen pistenotaatiolla päästään relaatiomallissa osoittamaan relaatioita ja attribuutteja yhtä suoraviivaisesti kuin oliomallissa on totuttu osoittamaan olioita ja niiden ominaisuuksia. Lopuksi olen vielä automatisoinut liitosoperaation laajennetun relaatiomallin ja NSQL-kielen päälle, jolloin suoria viittauksia voidaan tehdä myös erillisten relaatioiden välillä ja tiedonhallintajärjestelmä huolehtii relaatioiden liitoksista taustalla käyttäjältä piilotettuna. Viimeisenä lisäyksenä olen tuonut malliin IS-A suhteiden käsittelyn, joka oliomallissa hoidetaan luokkahierarkian avulla.

Kaikki laajennokset relaatiomalliin olen tehnyt siten, että relaatiomallin perusteet ja siten sen kaikki hyvät puolet on säilytetty. Tärkeimmät säilyttämäni ominaisuudet ovat arvoperustaisuus, yksinkertaiset helposti hallittavat tietotyypit, sekä se että relaatioalgebran kaikki operaatiot saavat syötteekseen yhden tai kaksi relaatiota ja tuottavat yhden relaation. Arvoperustaisuuden ansiosta voidaan luoda deklaratiivisia kyselykieliä, kuten tässä tutkimuksessa esitelty NSQL. Yksinkertaisten tietotyyppiensä ansiosta malli on saatu pidettyä tehokkaana ja helppokäyttöisenä. Relaatioalgebran operaatioiden tuloksen ollessa aina relaatio, voidaan kyselyitä linkittää vapaasti yhteen, jolloin saadaan helposti suoritettua monimutkaisiakin kyselyitä. Relaatiomallin perusteiden pysyessä ennallaan saadaan hyödynnettyä myös vanhat tehokkaat ja testatut optimointimenetelmät, tosin uuden mallin vaatimin laajennoksin, sekä säilytettyä relaatiomallin suosion perustana oleva vankka matemaattinen perusta ja mallin luonteva visualisointi taulukkoina.

Tämän tutkimuksen mukaisesti laajennettuna relaatiomalliin saadaan lisättyä sellaisia ominaisuuksia, jotka lisäävät mallin käytettävyyttä ja vähentävät siten tarvetta ottaa käyttöön uusia tietokantamalleja kuten oliomallia. Kun lisäksi huomioidaan se valtava työmäärä, joka vanhojen tietokantojen muuntaminen johonkin erilaiseen paradigmaan vaatii, osoittautunee tämän tutkimuksen mukainen laajennettu relaatiomalli varmasti hyväksi vaihtoehdoksi uuden ilmaisuvoimaisemman teknologian ja vanhan turvallisen ja edullisen teknologian yhdistäjänä.

9.2. JATKOTUTKIMUSKOHTEITA

Mielenkiintoisia jatkotutkimuskohteita avautuu paljon tämän tutkimuksen aihepiiristä. Useita tähän tutkimukseen liittyviä asioita on tutkittu jo aiemmin, mutta eri ominaisuuksien lisääminen samaan malliin tuo aina uusia ongelmia ominaisuuksien tehokkaaksi yhdistämiseksi ja uusien ominaisuuksien täydelliseksi hyödyntämiseksi. Mielenkiintoisia jatkotutkimuskohteita ovat muunmuassa tiedon kapseloinnin ja polymorfismin lisääminen relaatiomalliin, koska näiden avulla saataisiin IS-A suhteiden käsittelyynkin huomattavasti lisää ilmaisuvoimaa. IS-A hierarkian läpikäyntiä voidaan tehostaa kehittämällä optimointitekniikoita, siten että union-operaatiot tehdään tehokkuuden kannalta oikeassa vaiheessa kyselyä. Transitiivisulkeuman tehokasta toteuttamista on jo paljon tutkittukin, mutta aggregoivan transitiivisulkeuman tehokas toteuttaminen NF2-mallissa ja optimointi yhdessä muiden relaatio-operaatioiden kanssa on yhä mielenkiintoinen tutkimuskenttä. Erittäin lupaava tutkimusalue on logiikkaohjelmointikielen yhdistäminen relaatiomalliin Turingin koneen ilmaisuvoiman aikaansaamiseksi relaatiomallin sisään.

Lähdeluettelo:

[Agr87] Agrawal. Alpha: an extension of relational algebra to express a class of recursive queries. IEEE Data Engineering conference. Los Angeles. s. 580-590, (1987).

[AgDaJa90] R. Agrawal, S.Dar, V. Jagadish. Direct transitive closure algorithms: design and performance evaluation. ACM Transactions on database systems. vol. 15, no. 3. s. 427-458, (1990).

[AhaYao93] R. Ahad, B. Yao. RQL: A recursive query language. IEEE Transactions on knowledge and data engineering. vol. 5, no. 3. s. 451-461, (1993).

[Amer84] X3H2-84.2. Relational database language, American national standards committee on computer and information processing, (1984).

[BanRam86] F. Bancilhon, R. Ramakrishnan. An amateurs introduction to recursive query processing strategies. Proceedings: ACM SIGMOD 1986 international conference on management of data. s. 16, (1986).

[ChiGam90] D. Chimenti, R. Gamboa, R. Krishnamurthy, S. Naqvi, S. Tsur, C. Zaniolo. The LDL system prototype. IEEE Transactions on knowledge and data engineering. vol. 2, no. 1. s. 76-89, (1990).

[Codd70] E. F. Codd, A Relational model of data for large shared databanks, Communications of the ACM, Vol.13, no. 6, s. 377-387, (1970).

[Codd90] E. F. Codd, The relational model for database management: version2, Addison-Wesley publishing company, (1990).

[CrNo89] I. Cruz, T. Norvell. Aggregative closure: an extension of transitive closure. IEEE Fifth international conference on data engineering. s. 284, (1989).

[DarAgr93] S. Dar, R. Agrawal. Extending SQL with generalized transitive closure. IEEE Transactions on knowledge and data engineering, vol. 5, no. 5. s. 799-812, (1993).

[DarAgrJag91] S. Dar, R. Agrawal, H. Jagadish. Optimization of generalized transitive closure queries. Proceedings: IEEE Seventh international conference on data engineering. s. 345, (1991).

[Date84] C. Date, A critique of the SQL database language. Association of computer machinery, SIGMOD 14, s. 8-54, (1984).

[Eder90] J. Eder. Extending SQL with general transitive closure and extreme value selections. IEEE Transactions on knowledge and data engineering. vol. 2, no. 4. s. 381-390, (1990).

[ElNa94] R. Elmashri, S. Navathe. Fundamentals of database systems. The Benjamin/Cummings publishing company, (1994).

[GuhYu92] K. Guh, C. Yu. Efficient management of materialized transitive closure in centralized and parallel environments. IEEE Transactions on knowledge and data engineering. vol. 4, no. 4. s. 371-380, (1992).

[GuhYu94] K. Guh, C. Yu. Efficient query processing for a subset of linear recursive binary rules. IEEE Transactions on knowledge and data engineering. vol. 6, no. 5. s. 835-849, (1994).

[HanLu89] J. Han, W. Lu. Asynchronous chain recursion. IEEE Transactions on knowledge and data engineering. vol. 1, no. 2. s. 185-195, (1989).

[IoRaWi93] Y. Ioannis, R. Ramakrishnan, L. Winger. Transitive closure algorithms based on graph traversal. ACM Transactions on database systems. vol. 8, no. 3. s. 512-576, (1993).

[Jag90] H. Jagadish. Acompression technique to materialize transitive closure. ACM Transactions on database systems. vol. 15, no. 4. s. 558-598, (1990).

[KaIoCa92] R. Kabler, Y.Ioannis, M.Carey. Performance evaluation of algorithms for transitive closure. Information systems. vol. 17, no. 5. s. 415-441, (1992).

[KoyCai93] K. Koymen, Q. Cai. SQL*: A recursive SQL. Information systems. vol. 18, no. 2. s. 121-128, (1993).

[LuLeeHa94] W. Lu, D. Lee, J. Han. A study on the structure of linear recursion. IEEE Transactions on knowledge and data engineering. vol. 6, no. 5. s. 723-737, (1994).

[Mak77] A. Makinouchi, A consideration on normal form of not-necessarily-normalized relations in the relational data model, Proc. 3rd int. conf. on very large databases, Tokio, s. 447-453, (1977).

[ManSha90] M. Mannino, L. Shapiro. Extensions to query languages for graph traversal problems. IEEE Transactions on knowledge and data engineering. vol. 2, no. 3. s. 1-2, (1990).

[MoNgEm96] W. Mok, Y. Ng, D. Embley. A normal form for precisely characterizing redundancy in nested relations. ACM Transaction on database systems. vol. 21, no. 1. s. 77-106, (1996).

[NgQua95] Y. Ng, N. Qaraeen. Data retrieval and aggregates in SQL*/NR Proc. of the 6th international conference on information systems and management of data, CISMOD95, (1995).

[NieJär92/1] T. Niemi, K Järvelin. Operation-oriented query language approach for recursive queries - Part1. Functional definition. Information systems, vol. 17, no. 1. s. 49-75, (1992).

[NieJär92/2] T. Niemi, K Järvelin. Operation-oriented query language approach for recursive queries - Part2. Prototype implementation and its integration with relational databases. Information systems, vol. 77, no. 1. s. 77-106, (1992).

[NieJär93] T. Niemi, K Järvelin. A straightforward NF^2 relational interface with applications in information retrieval. Information processing & management, vol. 31, no. 2. s. 215-231, (1995).

[ParGuc92] J. Paredaens, D. Gucht. Converting nested algebra expressions into flat algebra expressions. ACM Transactions on database systems. vol. 17, no. 1. s. 65-93, (1992).

[Pää95] K. Pääkkönen, Kyselyn optimointi NF2-relaatiomallissa, Pro gradu -tutkielma, Tampereen yliopisto, tetojenkäsittelyopin laitos, (1995).

[RoHeDaMa86] A. Rosenthalm S. Heiler, U. Dayal, F. Manola. Traversal recursion: A practical approach to supporting recursive applications. Proceedings: ACM SIGMOD 1986 international conference on management of data. s. 166, (1986).

[RoKoSi88] M. Roth, H. Korth, A. Silberschatz. Extended algebra and calculus for nested relational databases. ACM Transactions on database systems. vol. 13, no. 4. s. 389-417, (1988).

[RoKoBa87] M. Roth, H. Korth, D. Batory. SQL/NF: A query language for non 1NF relational databases. Information systems. vol. 12, no. 1. s. 99-114, (1987).

[SaKeRa95] R. Sack-Davis, A. Kent, K. Ramamohanarao, J. Thom, J. Zobel. Atlas: A Nested relational database system fot text applications. IEEE Transactions on knowledge and data engineering. vol. 7, no. 3. s. 454-469, (1995).

[SalJärNie95] A. Salminen, K. Järvelin, T. Niemi. CQL: a QBE-like visual query language with recursion for data classification. Computer Science and information systems reports, working papers wp-32, University of Jyväskylä, (1995).

[ScSc86] H. Schek, M Scholl. The relational model with relation-valued attributes. Inform systems, vol. 11, No. 2. s. 137-147, (1986).

[Teu96] J. Teuhola. Path signatures: A way to speed up recursion in relational databases. IEEE Transactions on knowledge and data engineering. vol. 8, no. 3, (1996).

[Ull88] J. Ullman. Principles of database and knowledge-base systems, vol. 1. Computer science press, (1988).