hyväksymispäivä arvosana

arvostelija

Tietokantakyselyjen optimointi relaatiotietokannassa

Olli Rissanen

Kandidaatintutkielma
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Helsinki, 5. toukokuuta 2013

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

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

Tekijä — Författare — Author
Olli Rissanen

Työn nimi — Arbetets titel — Title

Tietokantakyselyjen optimointi relaatiotietokannassa

Oppiaine — Läroämne — Subject
Tietojenkäsittelytiede

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Kandidaatintutkielma	5. toukokuuta 2013	27

Tiivistelmä — Referat — Abstract

Tietokantakyselyn optimointi on prosessi, jossa tietokannan hallintajärjestelmän sisältämä kyselyn optimoija määrittää suorituskykyisimmän tavan kyselyn suorittamiseen. Suorituskykyisimmän suoritustavan löytämiseksi optimoija etsii joukon mahdollisia suoritustapoja, arvioi niiden suorittamiseen kuluvan kustannuksen ja valitsee niistä tehokkaimman. Kyselyn optimointi on NP-täydellinen ongelma, jossa mahdollisten suoritustapojen joukko voi kasvaa erittäin suureksi. Tutkielmassa tutustumme keskeisiin optimoinnin osa-alueisiin relaatiotietokantojen hallintajärjestelmien osalta sekä optimoinnin vaikutukseen kyselyjen suorituskyvyssä. Esittelemme tietokantakyselyn prosessoinnin ja optimoinnin kokonaisuudessaan yleisellä tasolla. Tarkastelemme vaihtoehtoisia tapoja mahdollisten suoritustapojen joukon luomiseen, kuten heuristiikkoja hyödyntävän dynaamisen algoritmin. Lisäksi tutkimme tilastotiedon käyttöä suoritustapojen kustannusarvioinnissa parhaan suoritustavan valitsemiseksi.

ACM Computing Classification System (CCS):
Information systems → Query optimization
Theory of computation → Database query processing and optimization (theory)
Information systems → Query planning

Avainsanat — Nyckelord — Keywords
Tietokantakysely, optimointi, relaatiotietokanta

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Sisältö

1 Johdanto

Tallennettavan tiedon määrä kasvaa jatkuvasti, sillä tietoa kerätään yhä enemmän ja laaja-alaisemmin. Tieto tallennetaan usein tietokantoihin, ja tiedon määrän noustessa tietokannan työtaakka kasvaa. Erityisesti tiedon hakeminen hidastuu tietokannan tiedon määrän kasvaessa, sillä tietokanta joutuu prosessoimaan enemmän tietoa palauttaakseen halutun vastauksen. Jotta tietoa voidaan hyödyntää mahdollisimman tehokkaasti, tarvitaan tiedon hallitsemiseen yhä tehokkaampia työkaluja. Optimoimalla tietokantakyselyjen suoritusta voidaan vaikuttaa suoritettujen operaatioiden määrään sekä muistialueen kokoon ja siten vähentää tietokannan vasteaikaa sekä resurssien käyttöä [MKR12]. Tutkielman tavoitteena on tutustuttaa lukija kyselyn optimointiin yleisellä tasolla sekä sen keskeisiin ongelmiin.

Tietokantaa käytetään tietokannan hallintajärjestelmällä, joka on kokoelma ohjelmia tiedon tallentamiseen, muokkaamiseen, analysoimiseen ja keräämiseen tietokannasta. Hallintajärjestelmää käytetään kyselykielellä, joista esimerkiksi SQL [CAE+76] on suunniteltu relaatiotietokantojen hallintajärjestelmille. Hallintajärjestelmän vastuulla on kyselyn muuttaminen tietokannan ymmärtämään muotoon säilyttäen kyselyn alkuperäinen tarkoitus.

Kyselyn optimointi on toteutettu automaattisena toimenpiteenä hallintajärjestelmän sisältämässä kyselyn optimoijassa, ja kaikista hallintajärjestelmän komponenteista optimoijalla on suurin merkitys tietokannan suorituskykyyn [MKR12]. Jokainen optimoija on osittain erilainen kokonaisuus komponentteja, jotka kuitenkin pohjautuvat samoihin yleisiin malleihin. Kyselyn optimoijan tavoitteena on minimoida itse optimointiin käytetty aika ja maksimoida optimoinnista saatu hyöty [JK84].

Optimoija toimii arvioimalla osajoukon kaikista tavoista kyselyn suorittamiseen ja valitsemalla niistä tehokkaimman [SAC+79]. Suoritustapaa kutsutaan kyselysuunnitelmaksi, ja se sisältää sarjan tietokannan relaatioihin kohdistuvia algebrallisia operaatioita, jotka tuottavat tulokseksi halutun vastauksen. Tietokantakyselyä vastaavia kyselysuunnitelmia on useita, sillä kysely voidaan usein esittää monena algebrallisesti toisiaan vastaavana esityksenä [JK84]. Algebrallista operaatiota kohden voi myös löytyä useita toteutuk-

sia, kuten liitosoperaatiota (join) toteuttavat lomitusliitos ja silmukkaliitos. Kyselysuunnitelma sisältää operaatioiden lisäksi tiedon ne toteuttavista algoritmeista. Saman kyselyn sisältämät mahdolliset kyselysuunnitelmat voivat olla suorituskyvyltään jopa eri suuruusluokassa [Ioa96, Ora13].

Kyselyn optimointi on NP-täydellinen ongelma [IK84], jossa hakualue voi nousta erittäin suureksi. Haasteeksi nousee kyselysuunnitelmien tuottaminen ja niiden suorituskyvyn ennustaminen. Kaikkien mahdollisten kyselysuunnitelmien arvioiminen on usein liian hidasta, joten optimoijan tulee valita pienin mahdollinen hakualue joka pitää sisällään halvimmat suunnitelmat [Cha98]. Suorituskyvyn ennustamisen ja hakualueen rajauksen lisäksi optimoija tarvitsee tehokkaan algoritmin koko hakualueen läpikäymiseen. On epärealistista odottaa kyselyn optimoijan aina löytävän parhaan kyselysuunnitelman, ja onkin tärkeämpää välttää huonoimpia suunnitelmia ja löytää hyvä suunnitelma [RG03].

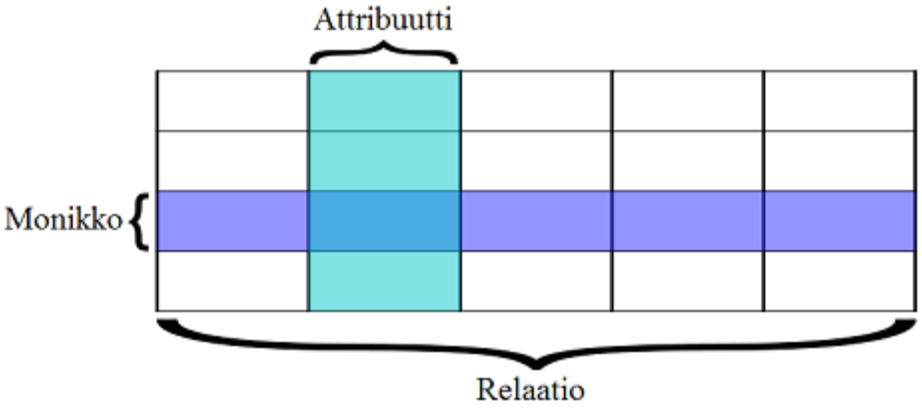
Tutkielman rakenne on seuraava. Luvussa 2 perehdytään optimoijan keskeisiin esitietoihin esittelemällä relaatiomalli ja tutustumalla relaatioalgebran operaatioihin. Lisäksi luvussa esitetään optimoijan tavoitteet, mitattavat resurssit sekä annetaan lyhyt katsaus optimoinnin historiaan. Luku 3 luo yleiskatsauksen tietokantakyselyn prosessointiin hallintajärjestelmän sisällä ja esittelee optimoijan toiminnan pääpiirteittäin kyselyn prosessoinnissa. Kyselysuunnitelman tuottamiseen syvennytään tarkemmin luvussa 4, jossa käsitellään kyselysuunnitelman rakennetta, niiden tuottamiseen käytettäviä algoritmeja sekä heuristiikkoja epäoptimaalisten kyselysuunnitelmien karsimiseen. Kyselysuunnitelmien kustannusarviointi ja siihen liittyvä tilastotiedon käyttö esitellään luvussa 5.

2 Optimoinnin perusteet

Tässä luvussa esitellään relaatiomalli sekä relaatioalgebran operaatiot, jotka ovat optimoinnin keskeisimmät esitiedot. Lisäksi luvussa tutustutaan tarkemmin optimoinnin tavoitteisiin ja sen vaikutuksenalaisiin resursseihin, sekä annetaan lyhyt katsaus optimoinnin historiaan.

2.1 Relaatiomalli

Relaatiotietokanta on relaatiomalliin [Cod70] perustuva tietokanta. Relaatiomallin keskeinen piirre on kaiken datan esittäminen n-paikkaisen karteesisen tulon osajoukkona, ja se tarjoaa deklaratiivisen menetelmän datan ja kyselyjen määrittämiseen. Relaatiomalli koostuu attribuuteista, monikoista ja relaatioista, jotka esitetään kuvassa 1. Matemaattisessa määritelmässä attribuutti on pari joka sisältää attribuutin nimen ja tyypin. Jokaiseen attribuuttiin liittyy sen arvojoukko. Monikko on järjestetty joukko attribuuttien arvoja. Relaatio koostuu otsakkeesta ja sisällöstä, jossa otsake on joukko attribuutteja ja sisältö on joukko monikkoja. Relaation otsake on myös jokaisen monikon otsake. Visuaalisessa esityksissä relaatio on taulukko ja monikko taulukon rivi.



Kuva 1: Relaatiomalliin perustuva tietokanta [Wik08].

2.2 Relaatioalgebra

Relaatioalgebra on relaatiomalliin pohjautuva joukko relaation käsittelyyn tarkoitettuja operaatioita, ja se muodostaa relaatiotietokantojen teoreettisen pohjan. Relaatioalgebran operandit ovat relaatioita, ja sillä on joukko-opin operaatioiden lisäksi omia operaatioita. Tarkastelemme seuraavaksi tärkeimpiä operaatioita.

Liitos (join) on operaatio muotoa $A \bowtie B$, jossa A ja B ovat relaatioita. Liitos

tuottaa relaation, jonka attribuutit ovat relaatiodien attribuuttien yhdiste: Join(rel_1, A op B, rel_2) = [EACH r_1 IN rel_1, EACH r_2 IN rel_2: r_1.A op r_2.B], jossa op on relaatioita vertaileva predikaatti ja rel_1 sekä rel_2 liitettävät relaatiot [JK84].

Valinta (selection) on muotoa $\sigma_{ehto}(A)$, jossa *ehto* on jokin loogisilla operaatioilla $=$, $\neq$, $\leq$, $\geq$, $\wedge$, $\vee$ vertailtavissa oleva ehto. Valinta tuottaa relaation, joka on joukko kaikista relaation monikoista joille pätee valinnan ehto.

Projektio (projection) on muotoa $\pi_{a_1,...a_n}(A)$, jossa $a_1, ..a_n$ on joukko attribuuttien nimiä. Projektio poimii relaatiosta alijoukon joka sisältää vain annetut attribuutit.

2.3 Optimoinnin tavoite

Tietokantakyselyjen optimoinnilla viitataan tietokantakyselyn suorittamiseen mahdollisimman tehokkaasti. Optimoinnin tavoitteena on joko maksimoida suorituskyky annetuilla resursseilla tai minimodia resurssien käyttö. Jarke ja Koch määrittävät mitattaviksi resursseiksi suorittimen ja muistin käytön sekä kommunikointikustannukset [JK84]. Teorey ja Fry huomasivat optimoijan vaikuttavan suorittimen palveluaikaan (service time), suorittimen jonotusaikaan, I/O:n palveluaikaan, I/O:n jonotusaikaan ja lukittumisen viiveeseen [TF82]. Osa tutkimuksista, kuten Kooi:n väitöskirja [Koo80] kuitenkin sivuuttavat suorittimen käytön kokonaan.

Muistin käyttöön liittyvä kustannus on tärkeä tekijä tietokannan suorituskyvyssä. Muistin käyttö jakautuu tallennuskustannukseen sekä ulkomuistiin pääsyn kustannukseen. Tallennuskustannuksella tarkoitetaan ulkomuistin sekä puskurimuistin käyttöä, ja se tulee aiheelliseksi kun muistin käyttö aiheutuu pullonkaulaksi. Kommunikointikustannukset käsittävät tiedon viennin tallennuspaikasta laskentapaikkaan ja edelleen tulosten esityspaikkaan. Ne jakautuvat kommunikaatioväylän käyttökustannukseen ja tiedonsiirrosta aiheutuvaan suorittamisen viiveeseen.

Resurssin merkitys riippuu tietokantatyypistä. Jarke ja Koch erittelivät hajautetuissa tietokannoissa kommunikointikustannukset hallitseviksi kustannuksiksi sekä paikallisissa tietokannoissa kaikki resurssit samanarvoisiksi

[JK84]. Jarken ja Kochin mukaan keskitetyissä tietokannoissa ulkomuistiin pääsyn kustannus ja prosessorin käyttö ovat oleellisia. Tässä tutkielmassa keskitymme erityisesti keskitettyjen tietokantojen optimointiin.

2.4 Optimoinnin historia

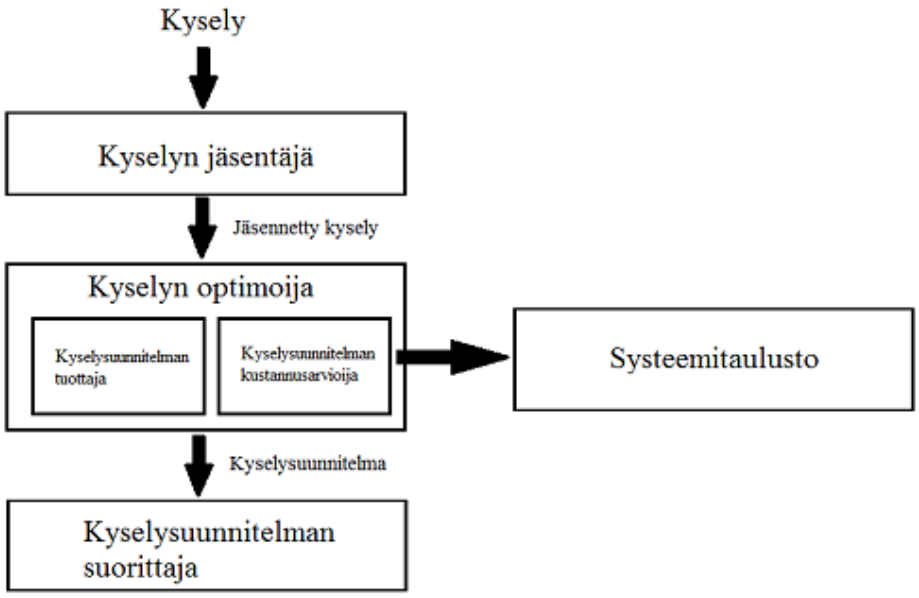
Relaatiomallin myötä optimoinnista tuli osa järjestelmän toiminnallisuutta, sillä malli tarjosi deklaratiivisuutensa vuoksi useita tapoja toteuttaa kukin kysely. Ennen relaatiomallin käyttöä optimointi tehtiin käsin. Selinger ym. esittelivät System R-hallintajärjestelmän OPTIMIZER-komponentin [SAC+79], joka on ensimmäisiä relaatiomalliin pohjautuvia kyselyn optimoijia. OPTIMIZER esitteli dynaamisen hakualgoritmin kyselysuunnitelmien luomiseen, ja se on käytössä hieman muunneltuna käytännössä kaikissa kaupallisissa järjestelmissä [Ioa96].

3 Tietokantakyselyn prosessointi

Tietokannan hallintajärjestelmän suorittama SQL-kyselyn prosessointi sisältää useita vaiheita. Kuva 2 esittää kyselyn optimointikerroksen ja suorituskerroksen hallintajärjestelmän sisällä. Alempana analysoimme komponenttien toimintaa tarkemmin.

3.1 Kyselyn jäsentäminen

Kyselyn prosessointi alkaa kyselyn jäsentäjän suorittamalla kyselyn syntaksin validoinnilla [SAC+79]. Syntaksin validoinnissa jäsentäjä tarkastaa kyselyn lauseopin oikeellisuuden. Osa hallintajärjestelmistä, kuten Oracle 11g, validoi myös samalla semantiikan oikeellisuuden [Ora09]. Semantiikan validoinnissa tarkastetaan objektien aitous, kyselyn yksiselitteisyys, oikeus haettavaan tietoon ja muuttujien tyypin sopivuus sarakkeiden tyyppeihin. Useat hallintajärjestelmät myös tallentavat kyselyt validoinnin jälkeen, jotta samaa kyselyä ei tarvitse jäsentää ja optimoida uudelleen. Yksi esimerkki kyselyjä tallentavasta hallintajärjestelmästä on Oracle Database, jossa tallennuspaikkaa kutsutaan nimellä Shared Pool [Ora05].



Kuva 2: Kyselyn jäsentäminen, optimointi ja suoritus [RG03].

Seuraavaksi jäsentäjä jakaa kyselyn lohkoihin (block) siten, että yhdessä lohkossa on täsmälleen yksi SELECT-lause, yksi FROM-lause ja korkeintaan yksi WHERE-, GROUP BY- ja HAVING-lause [RG03]. Kyselyn mahdollisesti sisältämät alikyselyt muodostavat kukin oman lohkonsa. Alikyselyitä tarkastellaan luvussa 4.3.

Jokainen lohko jäsennetään puuksi, joka on kyselyn algebrallinen esitysmuoto [MJ12]. Puun solmu sisältää yhden operaation kyselyn suorittamiseksi, ja sillä on nolla tai useampi alisolmua joiden ulostuloa (output) käytetään sen syötteenä. Esimerkiksi liitosoperaatiossa solmulla on kaksi alisolmua, joille toteutetaan liitosoperaatio ja järjestysoperaatiolla on yksi alisolmu joka järjestetään. Lehtisolmut ovat solmuja jotka suorittavat hakuja (scan) levyltä ja palauttavat saadut tulokset. Puu suoritetaan lehtisolmuista juureen.

Tämän jälkeen monimutkaiset kyselyt uudelleenkirjoitetaan (rewrite) soveltamalla niihin muutossääntöjä [Ioa96]. Pirahesh ym. tutkivat uudelleenkirjotusta Starburst-hallintajärjestelmän osalta [PHH92]. He määrittävät uudelleenkirjoituksen tavoitteiksi tehdä kyselyistä mahdollisimman deklaratiivisia ja suorittaa luonnollisia heuristiikkoja. Yksi heuristiikka on ope-

raatioiden siirtäminen suoritettavaksi mahdollisimman aikaiseen vaiheeseen, kuten tietokantataulun lukemisen aikana.

3.2 Kyselyn optimointi

Kun kysely on jäsennetty ja mahdollisesti uudelleenkirjoitettu, lähetetään se kyselyn optimoijalle. Kyselyn optimoija hakee tietokannan systeemitaulustoon tallennettua tilastotietoa kyselyyn liittyvistä relaatioista ja relaatioihin liittyvistä saantipoluista (access path). Tilastotietoa käytetään kyselyn suorituskyvyn arviointiin. Saantipolku on menetelmä, jolla taulurivit haetaan taulusta, ja se muodostuu joko tiedoston lukemisesta tai indiksien käytöstä. Selinger et. al. tutkivat kattavasti saantipolkujen kustannusarviointia [SAC+79].

Mikäli kyselyn semantiikkaa ei ole vielä tarkastettu, tarkastaa optimoija kyselyn semantiikan oikeellisuuden sekä kyselyn tyyppien sopivuuden sarakkeisiin ja operaatioihin systeemitauluston tiedon pohjalta [SAC+79]. Optimoija määrittää lohkojen suoritusjärjestyksen ja käy läpi saantipolut jokaisen lohkon FROM-lauseen relaatioiden lukemiseen. Mikäli lohkossa on useampi relaatio, etsii optimoija mahdolliset liitosoperaation suoritusjärjestykset ja suoritustavat. Lohkot liitetään uloimpaan lohkoon predikaatiksi. Alikyselyn liittämistä uloimpaan kyselyyn tutkitaan tarkemmin luvussa 4.3. Optimoija tuottaa mahdollisten kyselysuunnitelmien joukon käyttäen haettuja saantipolkuja, liitosoperaation suoritusjärjetyksiä ja suoritustapoja sekä heuristiikkoja. Yleisiä heuristiikkoja käsitellään luvussa 4.2.3. Optimoija arvioi kyselysuunnitelmien suoritukseen kuluvat resurssit systeemitauluston tilastotiedon pohjalta, ja valitsee suorituskykyisimmän kyselysuunnitelman suoritettavaksi.

3.3 Kyselyn suorittaminen

Seuraavassa vaiheessa kysely suoritetaan käyttämällä optimoijan tuottamaa kyselysuunnitelmaa. Kyselysuunnitelma muutetaan suoritettavaksi konekieleksi ja kyselyn lähteen mukaan joko suoritetaan heti tai tallennetaan muistiin myöhempää suorittamista varten. Kyselysuunnitelman suorittaja lukee tietokantataulut käyttäen kyselysuunnitelman määrittämiä saantipolkuja, ja

suorittaa saaduille relaatioille kyselysuunnitelman määrittämät operaatiot palauttaen lopuksi halutun tuloksen.

4 Kyselysuunnitelmien tuottaminen

Kyselyn optimoijan tulee tuottaa mahdollisimman suorituskykyisen kyselysuunnitelman sisältämä joukko käyttämättä tuottamiseen liikaa resursseja. Tässä luvussa perehdymme ensin kyselysuunnitelman rakenteeseen. Tämän jälkeen tutkimme algoritmeja kyselysuunnitelmien joukon luomiseen, sekä heuristiikkoja epäoptimaalisten kyselysuunnitelmien karsimiseen. Luvun lopuksi tutkimme alikyselyjen suorittamista.

4.1 Kyselysuunnitelma

Kyselysuunnitelma koostuu laajennetusta relaatioalgebrapuusta, jossa jokainen solmu kuvaa algebrallista operaatiota. Solmuun on liitetty tieto käytettävästä hakumetodista tiedon hakemiseen taulusta ja suoritusmetodista operaation suoritukseen. Kyselysuunnitelma sisältää kaikki tiedot jotka hallintajärjestelmä tarvitsee halutun tiedon hakemiseen tietokannasta. Tutkitaan seuraavaa SQL-kyselyä:

SELECT K.nimi

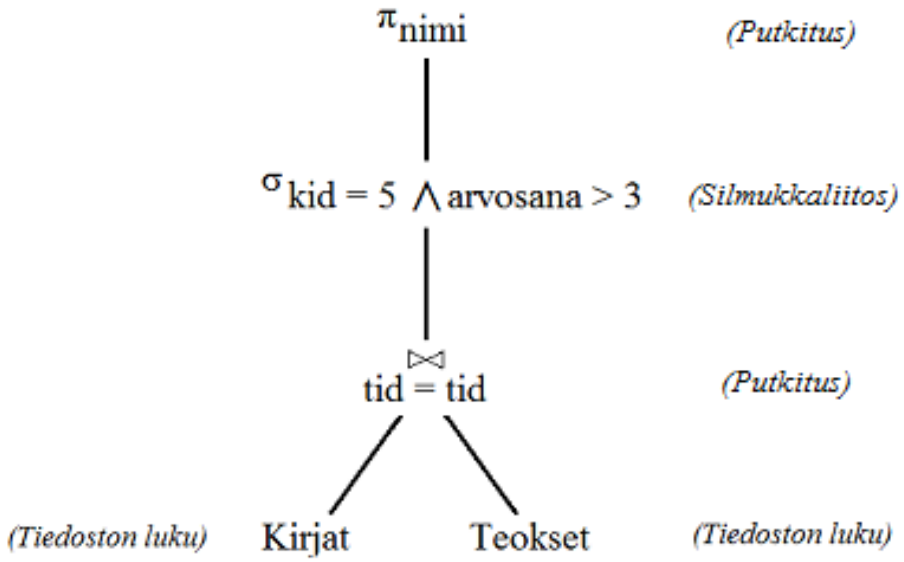
FROM Kirjat K, Teokset T

WHERE K.tid = T.tid AND K.kid = 5 AND T.arvosana > 3

Kysely voidaan tulkita relaatioalgebrassa seuraavasti:

$\pi_{nimi}(\sigma_{kid=5 \wedge arvosana>3}(Kirjat \bowtie_{tid=tid} \text{Teokset}))$

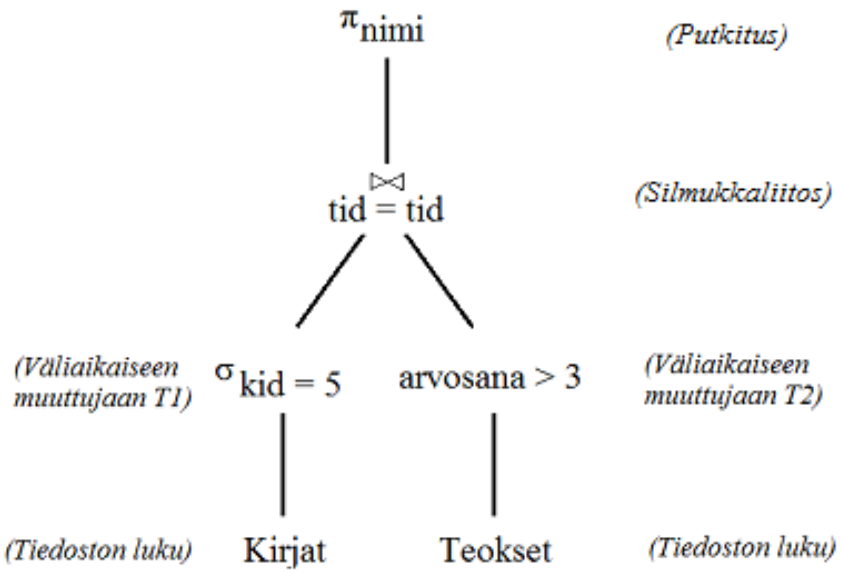
Kysely voidaan suorittaa esimerkiksi käyttäen liitosoperaatiossa silmukkaliitosalgoritmia, jonka jälkeen jokaiselle riville suoritetaan valinta ja projektio. Kyselyä vastaava esimerkkikyselysuunnitelma esitetään kuvassa 3.



Kuva 3: Kyselysuunnitelma esimerkkikyselylle.

liitosoperaatiossa voidaan käyttää hyväksi putkitusta (pipelining) [DSRS01], jossa kukin operaation palauttava monikko lähetetään suorituksen aikana toiseen operaatioon ilman sen tallentamista väliaikaiseen muuttujaan. Ilman putkitusta väliaikaiset tulokset tulee tallentaa väliaikaiseen muuttujaan. Putkitusta käytetään usein peräkkäisten liitosoperaatioiden toteuttamiseen, jolloin jokainen operaation palauttama monikko voidaan syöttää seuraavan operaation parametriksi. Putkitusta voidaan käyttää operaation suorittavan algoritmin salliessa sen.

Usein tehokkaampi tapa kyselyn suorittamiseen on suorittaa valinnat mahdollisimman aikaisessa vaiheessa [RG03]. Esimerkkikyselyssä tauluille suoritetaan kullekin valinta erikseen, joten suorittamalla ne ennen liitosoperaatiota voidaan vähentää yhdistettävien alkioiden määrää. Tehokkaampi kyselysuunnitelma aikaisilla valinnoilla, tallentaen valinnat väliaikaismuuttujiin T1 ja T2, esitetään kuvassa 4. Edelleen tehokkaampi tapa olisi välikaikaiseen muuttujaan tallentamisen sijaan putkittaa valintaoperaatiosta alkiot suoraan liitosoperaatioon.



Kuva 4: Suorituskykyisempi kyselysuunnitelma esimerkkikyselylle.

4.2 Hakualgoritmit

Kyselyn optimoija tarvitsee tehokkaan algoritmin mahdollisten kyselysuunnitelmien luomiseen. Jarke ja Koch määrittelevät kaksi tapaa kyselysuunnitelman valintaan [JK84]. Ensimmäisessä tavassa arvioidaan jokaisen erillisen kyselysuunnitelman kustannus, jolloin hyvä suunnitelma löydetään mutta optimointiin kulunut aika on suuri. Toisessa tavassa kustannus lasketaan inkrementaalisesti niiden rakentamisen aikana, ja suorituskyvyltään huonoksi havaitut kyselysuunnitelmat karsitaan pois. Selinger ym. esittelemä dynaaminen algoritmi on toisen tavan laajennus, jossa algoritmi arvioi operaation ja sen mahdolliset vaikutukset seuraaviin operaatioihin kerrallaan [SAC$^+$79].

Kyselysuunnitelmat sisältävä hakualue voi kasvaa erittäin suureksi kyselyn ollessa laaja. Useat optimoijat käyttävät heuristiikkoja, kuten valinnan suorittamista mahdollisimman aikaisin tai vain tietynlaista liitosoperaatioiden järjestyksen hyväksymistä, pienentääkseen kyselysuunnitelmien määrää. Tavoitteena on pitää hakualue mahdollisimman pienenä, sisältäen kuitenkin optimaalisen suunnitelman [JK84]. Luvussa 4.2.1 tutustutaan tarkemmin dynaamisiin algoritmeihin sekä luvussa 4.2.2 vaihtoehtoisiin algorit-

meihin hakualueen läpikäymiseen, kuten sääntöihin perustuvaan Starburst-hallintajärjestelmälle kehitettyyn algoritmiin [Loh88]. Lopuksi luvussa 4.2.3 käsitellään yleisiä optimoijan käyttämiä heuristiikkoja.

4.2.1 Dynaamiset algoritmit

Selinger ym. System R:lle esittelemä algoritmi on dynaamisesti karsiva (pruning) ja tyhjentävä (exhaustive) hakualgoritmi [SAC+79]. Dynaamista algoritmia pidetään standardina algoritmina optimoijalle, ja usean hallintajärjestelmän hakualgoritmi pohjautuu dynaamiseen algoritmiin sisältäen kuitenkin hieman variaatioita. Algoritmi rakentaa kaikki mahdolliset liitospuut käyttämällä heuristiikkoja ja karsii alioptimaalisiksi tiedetyt puut. Selinger ym. esittelemän algoritmin käyttämät heuristiikat ovat predikaattien ja valintojen suorittaminen mahdollisimman aikaisin ja niiden putkitus, karteesisten tulojen välttäminen sekä pelkästään vasemmalta syvien puiden käyttö liitosoperaation suorituksessa. Heuristiikkoja tutkitaan tarkemmin luvussa 4.2.3. Algoritmi huomioi liitosoperaation suorittamiseen vain silmukkaliitos ja lomitusliitosalgoritmit, joiden Selinger ym. huomasivat lähes aina tuottavan optimaalisen tai lähes optimaalisen suorituskyvyn muille paitsi erittäin pienille relaatioille.

Dynaaminen algoritmi toimii käyden jokaiselle kyselyn relaatiolle läpi jokaisen saantipolun, käyttäen mahdollisia indeksejä sekä tiedoston lukua. Algoritmi arvioi saantipolut, ja valitsee relaatioille suorituskykyisimmän saantipolun. Algoritmi pyrkii valitsemaan järjestetyn tulosjoukon palauttavan saantipolun, ellei järjestämätön tulosjoukko ole tehokkaampi. Järjestetty tulosjoukko mahdollistaa järjestettyä operandia vaativan lomitusliitosalgoritmin käytön liitosoperaation suorittamiseen. Seuraavaksi algoritmi käy läpi kaikki tavat relaatioiden liittämiseen käyttäen aiemmin saatuja saantipolkuja ja heuristiikkoja, sekä karsii epäoptimaalisiksi huomatut tavat. Algoritmi arvioi edellisen askeleen tuottamat kyselysuunnitelmat ja palauttaa suorituskykyisimmän, jota käytetään kyselyn suorittamiseen.

4.2.2 Muut algoritmit

Dynaamisten algoritmien kilpailijoita ovat satunnaistetut algoritmit, kuten Iterative Improvement [SG88]. Algoritmi pyrkii dynaamisten algoritmien kyselysuunnitelmien tuottamisen sijaan käymään läpi pienen määrän kyselysuunnitelmia ja löytävän niistä parhaan. Algoritmi toimii tila-avaruudessa, jossa yksi tila kuvaa yhtä kyselysuunnitelmaa johon on liitetty sen kustannus. Ttilasta voi siirtyä sen naapuritiloihin. Tilaa, jonka kustannus on pienempi kuin sen ympäröivien tilojen kustannus sanotaan paikalliseksi minimiksi. Algoritmi valitsee satunnaisesti aloitustilan, ja suorittaa siirtoja pienemmän kustannuksen sisältävään tilaan kunnes se löytää paikallisen minimin. Kun algoritmi löytää paikallisen minimin, valitsee se satunnaisesti uuden lähtötilan ja etsii uuden paikallisen minimin kunnes lopetusehto täyttyy. Algoritmi palauttaa lopuksi pienimmän kustannuksen sisältävän tilan löydetyistä paikallisista minimeistä.

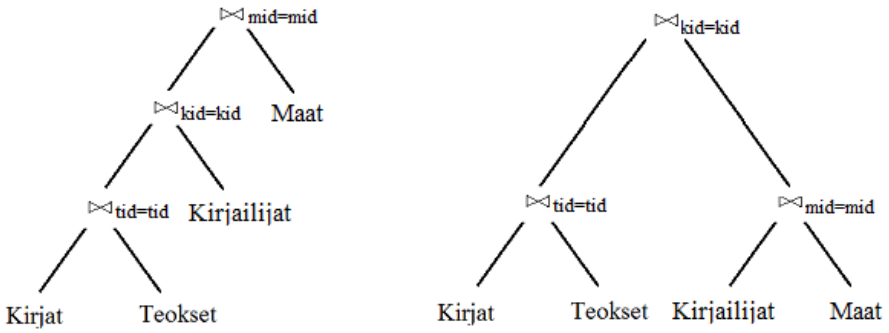
Satunnaisten algoritmien lisäksi geneettisiä algoritmeja, kuten Bennet ym. esittelemää algoritmia [BFI91], on sovellettu parhaan kyselysuunnitelman tuottamiseen. Geneettinen algoritmi koostuu kromosomeista, jotka ovat ratkaisuita ongelmaan ja vertailtavissa keskenään niiden kunnon (fitness) mukaan. Geneettinen algoritmi simuloi biologista ilmiötä, jossa kromosomeja verrataan keskenään niiden kunnon mukaan ja niistä luodaan jälkeläinen joka perii vanhemmilta ominaisuuksia. Vertailu tapahtuu iteraatioissa, joissa heikommat kromosomit karsiutuvat pois ja suorituskykyisemmät jatkavat seuraavaan iteraatioon. Bennet ym. algoritmissa kromosomit ovat binääripuita, joiden solmuissa on tieto kunkin operaation kustannuksesta. Suorituskykyisimmät jälkeläiset selviytyvät jatkoon, ja lopputuloksena saadaan suorituskykyinen ratkaisu. Bennet ym. huomasivat algoritmin löytävän 4-6 liitosoperaatiota sisältävässä kyselyssä aina optimaalisen ratkaisun, mutta suuremmilla kyselyillä ratkaisun suorituskyvyn heikkenevän.

4.2.3 Algoritmien heuristiikat

Optimoija voi suorittaa usean peräkkäisen liitosoperaation liittämisen monella eri tavalla. Ono ja Lohman tutkivat liitosoperaation suorittamisjärjestysten

vaikutusta tarkemmin, ja huomasivat liitosoperaation järjestyksen olevan hallitseva kustannus kyselyn optimoinnissa [OL90].

Tutkitaan kyselyä $Kirjat \bowtie Teokset \bowtie Kirjailijat \bowtie Maat$. Vasemmalta syväksi (left-deep) puuksi kutsutaan tapaa liittää kyselyt seuraavasti: $(((Kirjat \bowtie Teokset) \bowtie Kirjailijat) \bowtie Maat)$, jossa oikea lapsi on aina kantataulu. Lehtevässä puussa myös oikea lapsi voi olla liitosoperaation tulos: $(((Kirjat \bowtie Teokset) \bowtie (Kirjailijat \bowtie Maat)$. Kuva 5 havainnollistaa vasemmalta syvät ja lehtevät puut. Optimoijat käyttävät usein pelkästään vasemmalta syviä puita, sillä eri tapoja liittää liitosoperaatioita on liitosoperaatioiden määrän kasvaessa erittäin paljon. Käyttämällä vain vasemmalta syviä puita voidaan pienentää hakualuetta [IK91], ja vasemmalta syvät puut mahdollistavat putkituksen. Oikea lapsisolmu eli liitosoperaation sisempi taulu täytyy aina materialisoida eli tallentaa väliaikaiseen muuttujaan, sillä se täytyy käydä läpi jokaiselle ulomman taulun eli vasemman lapsisolmun monikolle [RG03].



Kuva 5: Vasemmalta syvät ja lehtevät puut.

Ioannidis ja Cha Kang tutkivat vasemmalta syvien puiden ja lehtevien (bushy) puiden suorituskykyeroja, ja huomasivat sekä lehtevien puiden että vasemmalta syvien puiden samanaikaisen käytön olevan helpompaa ja sen tuottavan parempia tuloksia kuin vain vasemmalta syvien puiden käytön [IK91]. Lehtevä puu mahdollistaa suuremman joukon mahdollisia kyselysuunnitelmia, joten se usein sisältää vasemmalta syviin puihin verrattuna huonompia suunnitelmia mutta usein myös optimaalisen suunnitelman.

Yleinen optimoijien käyttämä heuristiikka on myös predikaattien suorittaminen samaan aikaan relaatioiden ensimmäisen haun yhteydessä. Predikaateista projektio pyritään suorittamaan suoraan muiden operaatioiden tulosjoukolle. Levy ym. esittelivät tavan siirtää predikaatteja lohkojen välillä, jolloin predikaatit voidaan suorittaa yhä aiemmin ja siten vähentää väliaikaisen tulosjoukon kokoa [LMS94]. He myös esittelivät aritmeettisesti vertailtavissa olevien predikaattien lisäksi tavan siirtää negatiivisia predikaatteja, kuten NOT EXISTS ja EXCEPT. Predikaattien suorittaminen mahdollisimman aikaisin ei kuitenkaan aina tuota suorituskykyisintä tulosta [RG03]. Mikäli taulut on indeksoitu, voi valinta kadottaa tärkeät indeksit jolloin liitosoperaation suorittaminen hidastuu.

Tärkeä heuristiikka on lisäksi karteesisten tulojen välttäminen liitosoperaation suorituksessa mikäli kysely ei erityisesti vaadi niitä. Heuristiikkaa toteuttava algoritmi tuottaa liitosoperaation suoritukseen vain vaihtoehdot jotka eivät sisällä karteesista tuloa. Chaudhuri havaitsi karteesisten tulojen välttämisen toisinaan johtavan huonompaan suorituskykyyn [Cha98].

Laajennetuissa optimoijissa heuristiikkoja voidaan vaihtaa kyselykohtaisesti [Cha98], kuten toisinaan sallimalla lehtevien puiden läpikäyminen tai karteesisten tulojen käyttäminen. On kuitenkin erittäin vaikeaa määritellä heuristiikkojen vaikutukset etukäteen. Yksi ratkaisu ongelmaan on geneettisten algoritmien käyttö parhaiden heuristiikkojen määrittämiseen [BFI91].

4.3 Alikyselyt

Kyselyn predikaatin operandin ollessa toinen kysely puhutaan alikyselystä. Won Kim esitteli strategian, jossa alikysely suoritetaan ennen ulompaa kyselyä ja saatu tulos liitetään ulomman kyselyn predikaatiksi [Kim82]. Kim jakoi alikyselyt viiteen luokkaan niiden sisältämien predikaattien mukaan. Tutkimme seuraavaksi näistä viidestä luokasta kahta luokkaa, joissa alikysely ei sisällä liitosoperaatiota joka viittaa ulompaan kyselyyn. Tällaisessa tapauksessa alikysely voidaan suorittaa omana lohkonaan.

SELECT nimi

```
FROM Kirjat
WHERE hinta =
    (SELECT AVG(hinta)
    FROM Kirjat)
```

Esimerkkikyselyssä alikyselyn palauttama arvo lisätään ulompaan kyselyyn predikaatiksi niinkuin se olisi ollut osa ulompaa kyselyä. Esimerkkitapauksessa AVG(hinta) palauttaessa arvon 10 predikaatiksi saadaan "hinta = 10". Alikysely voi myös palauttaa joukon monikkoja, kuten seuraavassa tapauksessa:

```
SELECT nimi
FROM Kirjat
WHERE tid IS IN
    (SELECT tid
    FROM Teokset
    WHERE arvosana > 3)
```

Tällöin ulompaan kyselyyn liitetään predikaatti "tid IS IN X", jossa X on alikyselyn palauttama joukko monikoita. Mikäli alikysely sisältää liitosoperaation joka viittaa ulompaan kyselyyn, joudutaan liitosoperaatio suorittamaan ennen alikyselyn suorittamista. Alikyselyn paluuarvo voidaan kuitenkin edelleen liittää ulomman kyselyn predikaatiksi. Kim esitteli strategiat ulommasta kyselystä riippuvaisten alikyselyjen suorittamiseen kolmessa seuraavassa luokassaan. Yhteistä näille kolmelle luokalle on joidenkin operaatioiden suorittaminen ennen itse alikyselyn suorittamista.

5 Kyselysuunnitelmien kustannusarviointi

Kyselysuunnitelmien kustannusarvioinnissa optimoijan tulee arvioida kyselysuunnitelmien operaatioiden suorittamiseen kuluva aika sekä niiden tulosjou-

kon koko. Kyselysuunnitelman kustannus on sen sisältämien operaatioiden kustannusten summa, ja operaatioiden kustannukseen vaikuttaa olennaisesti sen operandien koko. Yksittäisten operaatioiden kustannus arvioidaan käyttämällä systeemitaulustoon tallennettua tilastotietoa operandeihin liittyvistä tiedoista, kuten indekseistä ja koosta. Systeemitaulusto koostuu tauluista ja näkymistä (view), ja sisältää kaiken tiedon mitä tietokanta tietää itsestään.

Optimoija käyttää kustannusarvointiin joko heuristisia arvioita tai kustannusmallia (cost model) [JK84], joista tutkimme tarkemmin kustannusmallia. Kustannusmallissa optimoija arvioi jokaisen operaation kustannuksen ja tulosjoukon koon, sillä operaation tulosjoukko on usein seuraavan operaation parametri. Kustannusmallin tuottaman kustannusarvion perusteella voidaan valita paras kyselysuunnitelma. Luvussa 5.1 tutustutaan tarkemmin systeemitaulustoon tallennettavaan tilastotietoon, ja luvussa 5.2 käsitellään systeemitauluston tilastotiedon pohjalta tehtävää kyselysuunnitelman kustannusarviointia.

5.1 Systeemitauluston tilastotieto

Systeemitaulusto sisältää tietokannan tilastollisen profiilin. Tilastotiedon avulla pyritään arvioimaan predikaattien ja operaatioiden kustannuksia käyttämällä tilastotietoa niiden operandeista. Mannino ym. tutkivat kattavasti tilastotiedon käyttöä kyselyn optimoinnissa [MCS88]. Heidän mukaan tarkka tilastotiedon tallentaminen on tärkeää kyselyn optimoinnille, mutta liian täsmällinen tallentaminen aiheutuu rasitteeksi.

Mannino ym. jakavat tilastollisen profiilin neljään kategoriaan: (1) keskitietoihin kuten moodi, mediaani ja keskiarvo, (2) hajontaan kuten varianssi sekä pienin ja suurin arvo, (3) kokoon kuten kardinaliteetti ja eri arvojen lukumäärä sekä (4) frekvenssin jakaantumiseen kuten normaalius ja tasaisuus. Systeemitaulusto sisältää myös jokaisesta taulusta tiedot sen nimestä ja tiedoston nimestä, attribuuttien tyypeistä ja nimistä, indekseistä ja niiden nimistä ja hakuavaimista sekä yleisistä tiedoista kuten sivukoosta ja puskurimuistin koosta. Hallintajärjestelmät tyypillisesti tallentavat tilastotiedon systeemitaulustoon histogrammeihin, joissa tieto jaetaan lohkoihin [Cha98].

Optimoija voi tällöin ladata muistiin vain kulloinkin relevantit lohkot.

Bruno ym. tarkastelevat tilastotiedon laajentamista väliaikaisiin tuloksiin, ja ovat huomanneet ratkaisun tuottavan parempia kyselysuunnitelmia kuin pelkän tietokannan taulun tilastotiedon käytön [BC02]. Ilman tilastotiedon käyttöä väliaikaisissa tuloksissa voi virhe kustannusarvioinnissa edetä operaatiolta operaatiolle ja johtaa tehottoman suunnitelman valintaan. Bruno ym. esittämä ratkaisu tallentaa tilastotietoa operaatioista saatuihin tulosjoukkoihin. Koko tilastotiedon tallentaminen väliaikaisiin tuloksiin on kuitenkin liian hidasta, joten he esittelivät algoritmin, joka valitsee verrattaen pienen osajoukon saatavilla olevasta tilastotiedosta tallennettavaksi väliaikaisiin tuloksiin.

5.2 Tulosjoukon koon arviointi

Predikaatin kustannus riippuu sen syötteen koosta ja järjestyksestä. Tutkitaan seuraavaa tapausta:

SELECT attribuuttit

FROM relaatiot

WHERE ehto 1 $\wedge$ ehto 2 $\wedge$ ... $\wedge$ ehto n

Kyselyn palauttamien monikkojen maksimimäärä on relaatioiden karteesinen tulo. Jokainen WHERE-ehto harventaa monikkojen määrää. WHERE-ehdon vaikutusta tulosjoukon kokoon voidaan mallintaa lisäämällä jokaiseen ehtoon vähennyskerroin, joka on oletettu suhde lähtöjoukosta tulosjoukkoon vain kyseisen ehdon osalta. Tulosjoukon koko voidaan siten arvioida kertomalla maksimijoukko vähennyskertoimien tulolla [RG03].

WHERE-lauseen ehtojen kertoimia voidaan laskea hyödyntämällä systeemitaulustoon tallennettua tilastotietoa. Selinger ym. määrittävät laskukaavoja vähennyskertoimien laskemiseen [SAC$^+$79]. He myös määrittävät oletuskertoimet relaatioille, joille arvioinnin kannalta olennaista tilastotietoa ei ole saatavilla. Oletetaan seuraavat tiedot:

$NKeys(I)$ = eri avainten lukumäärä indeksissä I
F = vähennyskerroin

Tutkimme seuraavaksi kahta Selinger ym. määrittelemää laskukaavaa:

$sarake = arvo$
Tämän tyyppiselle ehdolle vähennyskerroin voidaan arvioida kaavalla $F = \frac{1}{NKeys(I)}$, jos sarakkeessa on indeksi kyseiselle relaatiolle. Ilman indeksiä kyselyn optimoija käyttää kiinteää arvoa vähennyskertoimen arvioimiseen, joka esimerkiksi System R-relaatiotietokantaohjelmassa on 1/10.

$sarake1 = sarake2$
Tässä tapauksessa voidaan vähennyskerroin arvioida käyttäen kaavaa $F = \frac{1}{MAX(NKeys(I1),NKeys(I2))}$ jos kummassakin sarakkeessa on indeksi. Lisäksi oletetaan, että jokaisesta pienemmän indeksin arvoa vastaa arvo toisesta indeksistä. Mikäli vain toisessa sarakkeessa on indeksi, voidaan kustannus laskea aiemmalla kaavalla käyttäen indeksin omaavaa saraketta. Mikäli kummassakaan sarakkeessa ei ole indeksiä, arvioidaan arvoksi 1/10.

liitospredikaatin suorittamisessa merkittävää kustannusarvioinnin kannalta on sen suorittava algoritmi sekä liitospredikaattien järjestys. Silmukkaliitos sekä lomitusliitosalgoritmit ovat yleisesti käytettyjä, ja jälkimmäinen vaatii liitettävien relaatioiden olevan järjestyksessä liitettävien attribuuttien mukaan. Optimoijan tulee tällöin ottaa arvioinnissa huomioon relaation järjestämiseen kuluvat resurssit. Mahdolliset indeksit vaikuttavat myös olennaisesti algoritmien kustannuksiin.

6 Yhteenveto

Kyselyn optimoijalla on tietokannan hallintajärjestelmän komponenteista suurin merkitys tietokannan suorituskykyyn. Optimoijan tehtävä on löytää mahdollisimman tehokas tapa suorittaa kysely. Optimoija käy läpi joukon kyselysuunnitelmia kyselyn suorittamiseen, arvioi niiden kustannuksen tilas-

totiedon pohjalta ja valitsee niistä parhaan. Kustannus koostuu kyselysuunnitelman suorittamiseen käytetyistä resursseista.

Esittelimme kyselyn optimoijan sijainnin hallintajärjestelmän sisällä sekä tietokantakyselyn prosessoinnin yleisellä tasolla. Kyselysuunnitelmien tuottamisessa kaikkien suunnitelmien läpikäyminen on usein liian hidasta, joten huomioitavien kyselysuunnitelmien joukkoa pyritään rajaamaan. Tutkimme yleisiä hakualgoritmeja kyselysuunnitelmien joukon tuottamiseen, joista dynaamiset algoritmit käyttävät heuristiikkoja kyselysuunnitelmien joukon rajaamiseen. Heuristiikkojen käyttäminen ei ole triviaalia, sillä toisinaan paras kyselysuunnitelma karsitaan heuristiikan takia.

Kyselysuunnitelmien kustannusarviossa keskeinen ongelma on tilastotiedon optimaalinen tallentaminen. Liian tarkka tilastotiedon tallentaminen rasittaa tietokannan käyttöä, mutta auttaa kyselysuunnitelmien optimoinnissa. Tilastotiedon käyttö väliaikaisten tulosten arviointiin on tärkeää, sillä ilman sen käyttöä arviointivirhe etenee operaatiolta operaatiolle ja voi johtaa tehottoman kyselysuunnitelman valitsemiseen.

Sivusimme pintapuolisesti muita hakualgoritmeja, kuten geneettistä algoritmia. Myös alikyselyn suorittamisen osalta tutkimme vain kahta tapausta, ohittaen monimutkaisemmat alikyselyt. Jätimme tutkimatta suurimman osan tilastotiedon käytön teoriasta, esitellen aihetta yksinkertaisin esimerkein.

Optimoijan toiminta on suurimmaksi osaksi piilotettu tietokannan käyttäjältä, mutta useat hallintajärjestelmät mahdollistavat optimoijan ohjaamisen vihjeillä (hint). Usein vihjeillä voi määrittää käytettäviä indeksejä, hakualueen syvyyttä sekä joissain tapauksessa myös kytkeä kyselysuunnitelmien karsinnan pois päältä. Indeksin määrittäminen manuaalisesti mahdollistaa optimoijan käyttävän vain yhtä saantipolkua relaatiolle, ja näin vähentää luotavien kyselysuunnitelmien määrää. Manuaalinen indeksien määrittäminen on kuitenkin herättänyt myös vastustusta. Indeksien määrittämistä on pidetty optimoijan tehtävänä ja rasittavan turhaan tietokannan käyttäjää.

Kyselyn optimoinnin tutkimus on yhä laajamittaista. Viimeaikaiset tutkimukset käsittelevät grafiikkaprosessorin sekä rinnakkaisohjelmoinnin hyödyntämistä kyselyn optimoinnissa [HM12, WLMO11], mutta myös vaihtoehtoisia

algoritmeja dynaamiselle algoritmille [SC11]. Muita tutkittavia alueita ovat koneoppimisen käyttö suorituskykyisten kyselysuunnitelmien tunnistamiseen [BBC+12] sekä optimaalinen tilastotiedon käyttö [HILM09].

Lähteet

[BBC+12] Borkar, V., Y. Bu, M. Carey, J. Rosen, N. Polyzotis, T. Condie, M. Weimer ja R. Ramakrishnan: *Declarative systems for large-scale machine learning.* Bulletin of the Technical Committee on Data Engineering, 35(2):24, 2012.

[BC02] Bruno, N. ja S. Chaudhuri: *Exploiting statistics on query expressions for optimization.* Teoksessa *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, SIGMOD '02, sivut 263–274, New York, NY, USA, 2002. ACM, ISBN 1-58113-497-5. http://doi.acm.org/10.1145/564691.564722.

[BFI91] Bennett, K., M. Ferris ja Y. Ioannidis: *A Genetic Algorithm for Database Query Optimization.* Teoksessa *In Proceedings of the fourth International Conference on Genetic Algorithms*, sivut 400–407. Morgan Kaufmann Publishers, 1991.

[CAE+76] Chamberlin, D., M. Astrahan, K. Eswaran, P. Griffiths, R. Lorie, J. Mehl, P. Reisner ja B. Wade: *SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control.* IBM Journal of Research and Development, 20(6):560–575, 1976, ISSN 0018-8646.

[Cha98] Chaudhuri, S.: *An overview of query optimization in relational systems.* Teoksessa *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, PODS '98, sivut 34–43, New York, NY, USA, 1998. ACM, ISBN 0-89791-996-3. http://doi.acm.org/10.1145/275487.275492.

[Cod70] Codd, E.: *A relational model of data for large shared data banks.* Commun. ACM, 13(6):377–387, 1970, ISSN 0001-0782. http://doi.acm.org/10.1145/362384.362685.

[DSRS01] Dalvi, N., S. Sanghai, P. Roy ja S. Sudarshan: *Pipelining in multi-query optimization*. Teoksessa *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '01, sivut 59–70, New York, NY, USA, 2001. ACM, ISBN 1-58113-361-8. http://doi.acm.org/10.1145/375551.375561.

[HILM09] Haas, P., I. Ilyas, G. Lohman ja V. Markl: *Discovering and exploiting statistical properties for query optimization in relational databases: A survey*. Statistical Analysis and Data Mining, 1(4):223–250, 2009.

[HM12] Heimel, M. ja V. Markl: *A First Step Towards GPU-assisted Query Optimization*. Teoksessa *The Third International Workshop on Accelerating Data Management Systems using Modern Processor and Storage Architectures, Istanbul, Turkey*, sivut 1–12, 2012.

[IK84] Ibaraki, T. ja T. Kameda: *On the optimal nesting order for computing N-relational joins*. ACM Trans. Database Syst., 9(3):482–502, 1984, ISSN 0362-5915. http://doi.acm.org/10.1145/1270.1498.

[IK91] Ioannidis, Y. ja Y. Kang: *Left-deep vs. bushy trees: an analysis of strategy spaces and its implications for query optimization*. SIGMOD Rec., 20(2):168–177, 1991, ISSN 0163-5808. http://doi.acm.org/10.1145/119995.115813.

[Ioa96] Ioannidis, Y..: *Query optimization*. ACM Comput. Surv., 28(1):121–123, 1996, ISSN 0360-0300. http://doi.acm.org/10.1145/234313.234367.

[JK84] Jarke, M. ja J. Koch: *Query Optimization in Database Systems*. ACM Comput. Surv., 16(2):111–152, 1984, ISSN 0360-0300. http://doi.acm.org/10.1145/356924.356928.

[Kim82] Kim, W.: *On optimizing an SQL-like nested query.* ACM Trans. Database Syst., 7(3):443–469, 1982, ISSN 0362-5915. http://doi.acm.org/10.1145/319732.319745.

[Koo80] Kooi, R.: *The optimization of queries in relational databases.* väitöskirja, Case Western Reserve University, Cleveland, OH, USA, 1980. AAI8109596.

[LMS94] Levy, A., I. Mumick ja Y. Sagiv: *Query Optimization by Predicate Move-Around.* Teoksessa *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, sivut 96–107, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc., ISBN 1-55860-153-8. http://dl.acm.org/citation.cfm?id=645920.672839.

[Loh88] Lohman, G.: *Grammar-like functional rules for representing query optimization alternatives.* SIGMOD Rec., 17(3):18–27, 1988, ISSN 0163-5808. http://doi.acm.org/10.1145/971701.50204.

[MCS88] Mannino, M., P. Chu ja T. Sager: *Statistical profile estimation in database systems.* ACM Comput. Surv., 20(3):191–221, 1988, ISSN 0360-0300. http://doi.acm.org/10.1145/62061.62063.

[MJ12] Mahajan, S. ja V. Jadhav: *An analysis of execution plans in query optimization.* Teoksessa *2012 International Conference on Communication, Information & Computing Technology (ICCICT)*, sivut 1–5. IEEE, 2012.

[MKR12] Mor, J., I. Kashyap ja R. Rathy: *Article: Analysis of Query Optimization Techniques in Databases.* International Journal of Computer Applications, 47(15):6–12, June 2012.

[OL90] Ono, K.i ja G. Lohman: *Measuring the complexity of join enumeration in query optimization.* Teoksessa *Proceedings of the 16th International Conference on Very Large Data Bases*, sivut 314–325. Morgan Kaufmann Publishers Inc., 1990.

[Ora05] Oracle: *Understanding Shared Pool Memory Structures*, syyskuu 2005. http://www.oracle.com/technetwork/database/focus-areas/manageability/ps-s003-274003-106-1-fin-v2-128827.pdf, |1.4.2013|.

[Ora09] Oracle: *Oracle Database Online Documentation 11g Release 1*, 2009. http://docs.oracle.com/cd/B28359_01/server.111/b28318/sqlplsql.htm, |1.4.2013|.

[Ora13] Oracle: *MySQL 5.0 Reference Manual*, 2013. http://docs.oracle.com/cd/E17952_01/refman-5.0-en/controlling-optimizer.html, |1.4.2013|.

[PHH92] Pirahesh, H., J. Hellerstein ja W. Hasan: *Extensible/rule based query rewrite optimization in Starburst.* Teoksessa *International Conference on Management of Data: Proceedings of the 1992 ACM SIGMOD international conference on Management of data*, nide 2, sivut 39–48, 1992.

[RG03] Ramakrishnan, R. ja J. Gehrke: *Database Management Systems.* McGraw-Hill, Inc., New York, NY, USA, 3 painos, 2003, ISBN 0072465638, 9780072465631.

[SAC+79] Selinger, P., M. Astrahan, D. Chamberlin, R. Lorie ja T. Price: *Access path selection in a relational database management system.* Teoksessa *Proceedings of the 1979 ACM SIGMOD international conference on Management of data*, SIGMOD '79, sivut 23–34, New York, NY, USA, 1979. ACM, ISBN 0-89791-001-X. http://doi.acm.org/10.1145/582095.582099.

[SC11] Sevinç, E. ja A. Coşar: *An evolutionary genetic algorithm for optimization of distributed database queries.* The Computer Journal, 54(5):717–725, 2011.

[SG88] Swami, A. ja A. Gupta: *Optimization of large join queries.* SIGMOD Rec., 17(3):8–17, 1988, ISSN 0163-5808. http://doi.acm.org/10.1145/971701.50203.

[TF82] Teorey, T. ja J. Fry: *Design of Database Structures*. Prentice Hall Professional Technical Reference, 1982, ISBN 0132000970.

[Wik08] Wikipedia: *Relational database terminology*, joulukuu 2008. `https://en.wikipedia.org/wiki/File:Relational_database_terms.svg`, |1.4.2013|.

[WLMO11] Wu, S., F. Li, S. Mehrotra ja B. Ooi: *Query optimization for massively parallel data processing*. Teoksessa *Proceedings of the 2nd ACM Symposium on Cloud Computing*, SOCC '11, sivut 12:1–12:13, New York, NY, USA, 2011. ACM, ISBN 978-1-4503-0976-9. `http://doi.acm.org/10.1145/2038916.2038928`.