Tietokantakyselyjen optimointi relaatiotietokannassa

Olli Rissanen

Kandidaatintutkielma
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Helsinki, 27. huhtikuuta 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	27. huhtikuuta 2013	24

Tiivistelmä — Referat — Abstract

Tietokantakyselyn optimointi on kriittistä relaatiotietokannan suorituskyvyn kannalta. Tutkielmassa tutustutaan tietokantakyselyjen optimointiin relaatiotietokantojen hallintajärjestelmien osalta sekä optimoinnin vaikutukseen kyselyjen suorituskyvyssä. Kyselyn optimointia tarkastellaan vaikeana hakuongelmana, ja tutkitaan olemassa olevia ratkaisuja ongelman ratkaisemiseen.

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

Avainsanat — Nyckelord — Keywords
Tietokantakysely, optimointi, relaatiotietokanta

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Sisältö

1 Johdanto

Modernit järjestelmät lisäävät jatkuvasti tietokantojen työtaakkaa tiedon määrän kasvaessa. Jotta tiedosta saadaan mahdollisimman paljon irti, tarvitaan tiedon hallitsemiseen yhä tehokkaampia työkaluja. Tietokannan suorituskyky on tärkeää koko järjestelmän suorituskyvyn osalta, sillä tiedon lukeminen massamuistista on hidasta verrattuna rekistereiden tai välimuistin käyttöön. 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 optimoijan toimintaan yleisellä tasolla sekä optimoinnin 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 hallintojärjestelmän 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 menetelmistä kyselyn suorittamiseen ja valitsemalla niistä tehokkaimman menetelmän [SAC+79]. Menetelmää kyselyn suorittamiseksi sanotaan 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 toteutuksia, kuten join-operaatiota toteut-

tavat merge join, hash join ja nested loop join. 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 hakeminen 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].

Luku 2 sisältää esitiedot kyselyn optimoijan toiminnalle, joiden lisäksi luvussa perehdytään optimoijan historiaan ja sen tavoitteisiin. Luku 3 tiivistää tietokantakyselyn prosessoinnin 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.

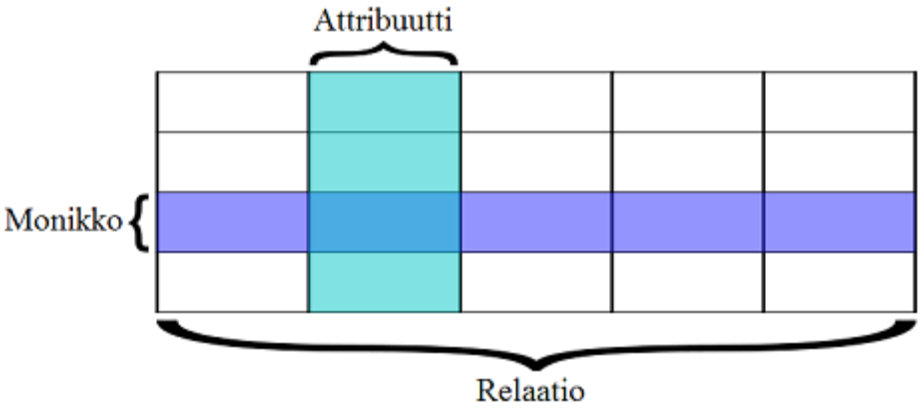
2 Taustaluku

Tässä luvussa käydään läpi relaatiomallin käsite sekä relaatioalgebran operaatiot, jotka ovat optimoijan keskeisimmät esitiedot. Lisäksi luvussa tutustutaan tarkemmin optimoijan tavoitteisiin ja sen vaikutuksen alaisiin resursseihin, sekä annetaan lyhyt katsaus optimoijan 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.



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, joka on kaikkien kombinaatioiden joukko A:n ja B:n yhteisistä attribuuteista.

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 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 [TF82] huomasivat optimoijan vaikuttavan suorittimen palveluaikaan (service time), suorittimen jonotusaikaan, I/O:n palveluaikaan, I/O:n jonotusaikaan ja lukittumisen viiveeseen. Osa tutkimuksista, kuten Kooi:n väitöskirja [Koo80] kuitenkin sivuuttavat suorittimen käytön kokonaan.

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

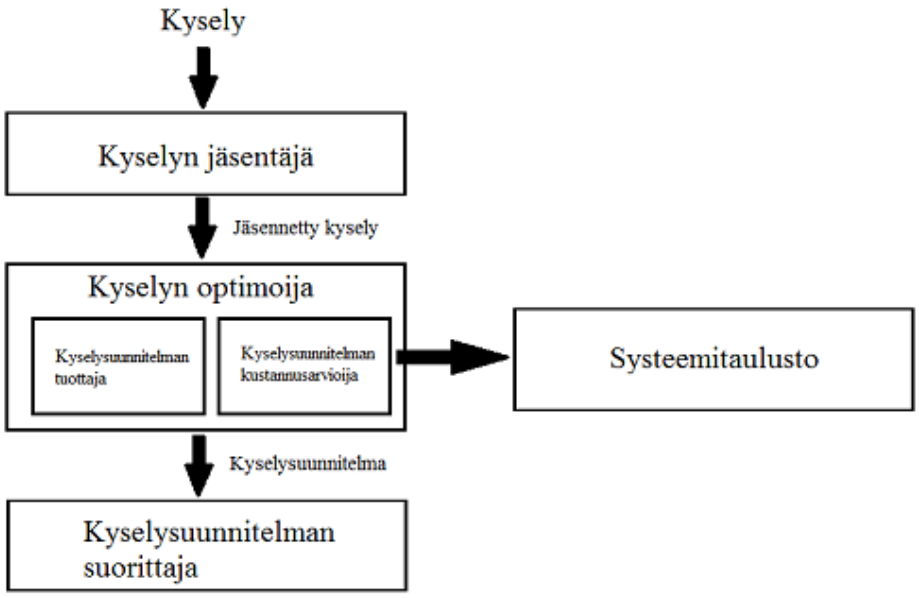
2.4 Optimoijan historia

Relaatiomallin myötä optimoinnista tuli osa järjestelmän toiminnallisuutta, sillä se tarjosi deklaratiivisuutensa vuoksi useita tapoja toteuttaa kukin kysely. Ennen relaatiomallin käyttöä optimointi tehtiin käsin. Selinger et al. 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.

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



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].

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 join-operaatiossa solmulla on kaksi alisolmua, joille toteutetaan join-operaatio ja sort-operaatiolla 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 et al. tutkivat uudelleenkirjotusta Starburst-hallintajärjestelmän osalta artikkelissa [PHH92]. He määrittivät uudelleenkirjoituksen tavoitteiksi tehdä kyselyistä mahdollisimman deklaratiivisia ja suorittaa luonnollisia heuristiikkoja. Yksi heuristiikka on operaatioiden siirtäminen suoritettavaksi mahdollisimman aikaisessa vaiheessa, 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 artikkelissaan [SAC+79].

Mikäli kyselyn semantiikkaa ei ole vielä tarkastettu, tarkastaa optimoija kyselyn semantiikan oikeellisuuden ja kyselyn tyyppien sopivuuden sarakkeisiin ja operaatioihin systeemitaulun tiedon pohjalta. Optimoija määrittää lohkojen suoritusjärjestyksen ja käy läpi saantipolut jokaisen lohkon FROM-lauseen relaatioiden lukemiseen. Mikäli lohkossa on useampi relaatio, arvioi optimoija join-operaation järjestyksen ja suoritustapojen suorituskyvyt ja valitsee tehokkaimman algoritmin operaatioiden suorittamiseen. Optimoija valitsee jokaiselle lohkolle vähiten resursseja käyttävän saantipolun ja rakentaa niistä kyselysuunnitelman [SAC+79].

3.3 Kyselyn suorittaminen

Seuraavassa vaiheessa kysely suoritetaan käyttämällä optimoijan tuottamaa kyselysuunnitelmaa. Suorittamisessa kyselysuunnitelma muutetaan suoritettavaksi konekieleksi ja kyselyn lähteen mukaan joko suoritetaan tai tallennetaan muistiin myöhempää suorittamista varten.

4 Kyselysuunnitelmien tuottaminen

Kyselyn optimoijan tulee tuottaa mahdollisimman suorituskykyisen kyselysuunnitelman sisältämä joukko käyttämättä tuottamiseen liikaa resursseja. Tässä luvussa perehdymme ensiks 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:

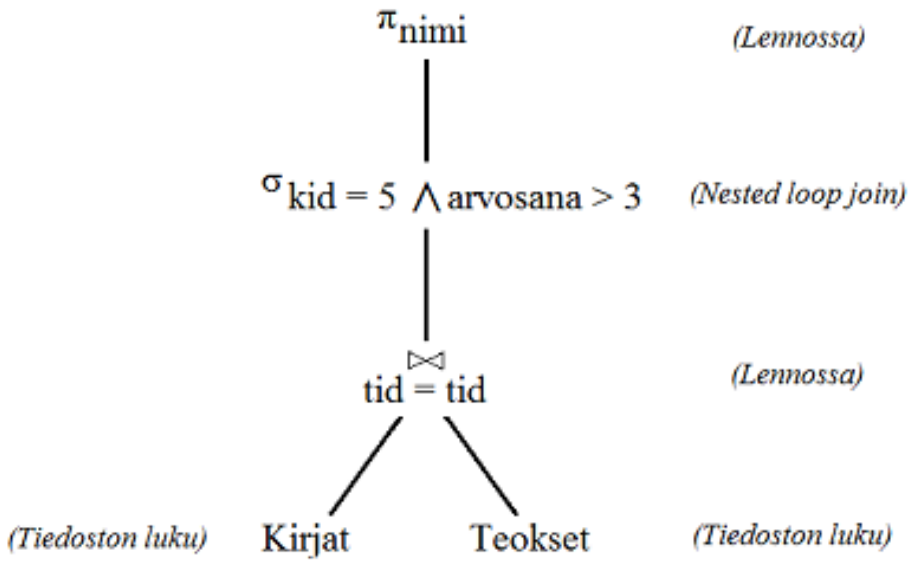
$\pi_{nimi}(\sigma_{kid=5 \wedge arvosana>3}(Kirjat \bowtie_{tid=tid} \text{Teokset}))$

Kysely voidaan suorittaa esimerkiksi käyttäen join-operaatiossa nested loop join-algoritmia, jonka jälkeen jokaiselle riville suoritetaan valinta ja projektio. Kyselyä vastaava esimerkkikyselysuunnitelma esitetään kuvassa 3.

Join-operaatiossa voidaan käyttää hyväksi putkitusta (pipelining) [DSRS01], jossa operandi putkitetaan suorituksen aikana toiseen operaatioon ilman väliaikaista taulua tuloksen tallentamiseen, lennossa. Ilman putkitusta väliaikaiset tulokset tulee tallentaa väliaikaiseen muuttujaan. Putkitusta käytetään usein peräkkäisten join-operaatioiden toteuttamiseen, jolloin jokainen operaation palauttama operandi 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 join-operaatiota

Kuva 3: Kyselysuunnitelma erimerkkikyselylle.

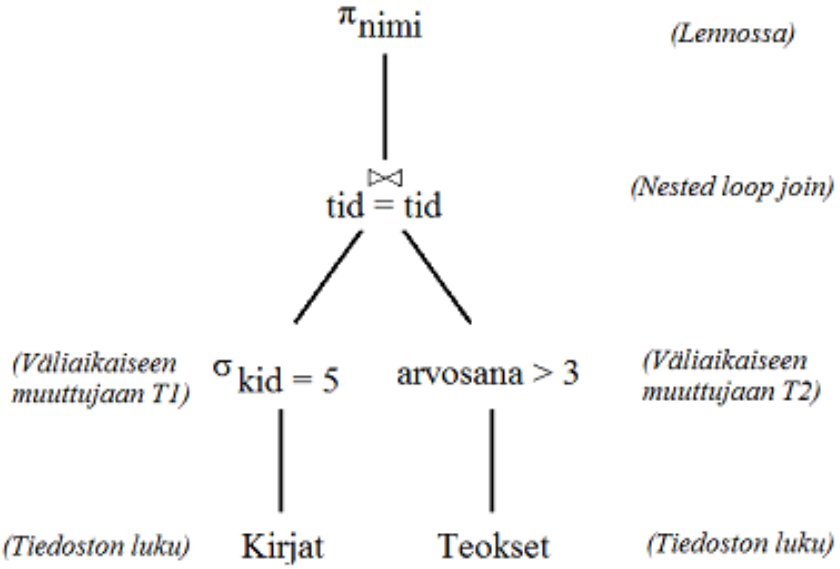


voidaan vähentää yhdistettävien alkioiden määrää. Tehokkaampi kyselysuunnitelma aikaisilla valinnoilla, tallentaen valinnat väliaikaismuuttujiin T1 ja T2, esitetään kuvassa 4. Edelleen tehokkaampi tapa olisi välikaikaiseen muuttujaan tallentamisen sijaan putkittaa valinta-operaatiosta alkiot suoraan join-operaatioon.

4.2 Hakualgoritmit

Kyselyn optimoija tarvitsee tehokkaan algoritmin mahdollisten kyselysuunnitelmien luomiseen. Jarke ja Koch määrittelevät artikkelissaan [JK84] kaksi tapaa kyselysuunnitelman valintaan. 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 et al. esittelemä dynaaminen algoritmi [SAC$^+$79] on toisen tavan laajennus, jossa algoritmi arvioi operaation ja sen mahdolliset vaikutukset seuraaviin operaatioihin kerrallaan.

Kuva 4: Suorituskykyisempi kyselysuunnitelma erimerkkikyselylle.



Kyselysuunnitelmat sisältävä hakualue voi kasvaa erittäin suureksi kyselyn ollessa laaja. Useat optimoijat käyttävät heuristiikkoja, kuten valinnan suorittamista mahdollisimman aikaisin tai vain tietynlaista join-operaatioiden järjestyksen hyväksymistä, pienentääkseen kyselysuunnitelmien määrää. Optimoija voi näin karsia tarkasteltavien kyselysuunitelmien 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 algoritmeihin 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 et al. System R:lle esittelemä algoritmi on dynaaminen karsiva ja tyhjentävä (exhaustive) hakualgoritmi. Dynaamista algoritmia pidetään standardina algoritmina optimoijalle, ja usean hallintajärjestelmän hakualgoritmi pohjautuu dynaamiseen algoritmiin sisältäen kuitenkin hieman variaatioita. Algoritmi rakentaa kaikki mahdolliset join-puut käyttämällä heuristiikkoja

ja karsii alioptimaalisiksi tiedetyt puut. Selinger et al. esittelemän algoritmin käyttämät heuristiikat ovat predikaattien ja valintojen suorittaminen mahdollisimman aikaisin ja niiden putkitus, karteesisten tulojen välttäminen sekä pelkästään vasemmalta syvien puiden käyttö join-operaation suorituksessa. Heuristiikkoja tutkitaan tarkemmin luvussa 4.2.3. Algoritmi huomioi join-operaation suorittamiseen vain nested loop join- ja merging scans-algoritmit, joiden Selinger et al. huomasivat lähes aina tuottavan optimaalisen tai lähes optimaalisen suorituskyvyn muille paitsi erittäin pienille relaatioille.

4.2.2 Muut algoritmit

Dynaamisten algoritmien kilpailijoita ovat satunnaistetut algoritmit, kuten Iterative Improvement [NSS86], jossa algoritmi pyrkii dynaamisten algoritmien kyselysuunnitelmien tuottamisen sijaan rakentamaan yhden mahdollisimman hyvän suunnitelman. Algoritmi käy satunnaisesti läpi verkon solmuja, jossa jokainen solmu kuvaa yhtä kyselysuunnitelmaa johon on liitetty sen kustannus. Iterative Improvement valitsee vain siirtoja alamäkeen, jossa kohdesolmun kustannus on pienempi kuin lähtösolmun, ja lopuksi palauttaa reitin varrelta löydetyn pienimmän kustannuksen sisältävän solmun.

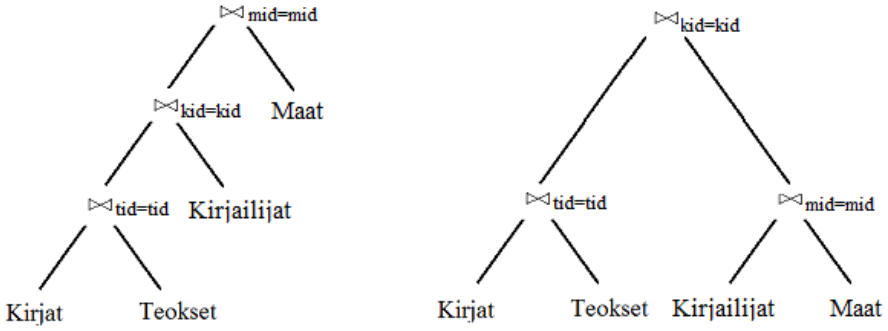
Satunnaisten algoritmien lisäksi geneettiset algoritmit, kuten Bennet et al. esittelemä algoritmi [BFI91], ovat tutkimuksen kohteena. Geneettinen algoritmi koostuu kromosomeista, jotka ovat ratkaisuita ongelmaan ja vertailtavissa keskenään niiden kunnon (fitness) mukaan. Geneettinen algoritmi simuloi biologista ilmiötä, jossa kromosomeja verrataan keskenään niiden kunnon mukaan ja niistä luodaan jälkeläinen joka perii vanhemmilta ominaisuuksia. Heikommat kromosomit karsiutuvat pois ja suorituskykyisemmät jatkavat seuraavaan sukupolveen. Bennet et al. algoritmissa kromosomit ovat binääripuita, joiden solmuissa on tieto kunkin operaation kustannuksesta. Suorituskykyisimmät jälkeläiset selviytyvät jatkoon, ja lopputuloksena saadaan optimaalinen ratkaisu.

4.2.3 Algoritmien heuristiikat

Optimoija voi suorittaa usean peräkkäisen join-operaation liittämisen monella eri tavalla. Ono ja Lohman tutkivat join-operaation suorittamisjärjestysten vaikutusta tarkemmin artikkelissaan [OL90], ja huomasivat join-operaation järjestyksen olevan hallitseva kustannus kyselyn optimoinnissa.

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. Lehtevä puu taas on puu, jossa myös oikea lapsi voi olla join-operaation 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 join-operaatioita on join-operaatioiden määrän kasvaessa erittäin paljon. Käyttämällä vain vasemmalta syviä puita voidaan pienentää hakualuetta [IK91], ja vasemmalta syvät puut mahdollistavat putkituksen. Oikea lapsisolmu eli join-operaation sisempi taulu täytyy aina materialisoida, 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 artikkelissaan [IK91] left-deep 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. 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 et al. esittelivät artikkelissaan [LMS94] tavan siirtää predikaatteja lohkojen välillä, jolloin predikaatit voidaan suorittaa yhä aiemmin ja siten vähentää väliaikaisen tulosjoukon kokoa. 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 join-operaation suorittaminen hidastuu.

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

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 [BFI91] käyttö parhaiden heuristiikkojen määrittämiseen.

4.3 Alikyselyt

Kyselyn predikaatin operandin ollessa toinen kysely puhutaan alikyselystä. Won Kim esitteli strategian [Kim82], jossa alikysely suoritetaan ennen ulompaa kyselyä ja saatu tulos liitetään ulomman kyselyn predikaatiksi. Kim jakoi alikyselyt viiteen luokkaan niiden sisältämien predikaattien mukaan. Tutkimme seuraavaksi näistä viidestä luokasta kahta luokkaa, joissa alikysely ei sisällä join-operaatiota 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 join-operaation viitaten ulompaan kyselyyn, joudutaan join-operaatio suorittamaan ennen alikyselyn suorittamista. Alikyselyn paluuarvo voidaan kuitenkin edelleen liittää ulomman kyselyn predikaatiksi.

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 järjestyksestä ja koosta.

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 et al. tutkivat kattavasti tilastotiedon käyttöä kyselyn optimoinnissa artikkelissaan [MCS88]. Heidän mukaan tarkka tilastotiedon tallentaminen on tärkeää kyselyn optimoinnille, mutta liian täsmällinen tallentaminen aiheutuu rasitteeksi.

Artikkelissa tilastollinen profiili jaettiin 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 [Cha98], joissa tieto jaetaan lohkoihin. Optimoija voi tällöin ladata muistiin vain kulloinkin relevantit lohkot.

Bruno et al. tarkastelivat tutkimuksessaan tilastotiedon laajentamista väliaikaisiin tuloksiin, ja he huomasivat 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 et al. 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 et al. määrittävät artikkelissaan [SAC$^+$79] laskukaavoja vähennyskertoimien laskemiseen. He myös määrittivät oletuskertoimet relaatioille, joille arvioinnin kannalta olennaista tilastotietoa ei ole saatavilla. Oletetaan seuraavat tiedot:

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

F = vähennyskerroin

$sarake = arvo$
Tämän tyyppiselle ehdolle vähennyskerroin voidaan arvioida kaavalla $F = \frac{1}{NKeys(I)}$, jos sarakkeessa on indeksi kyseiselle relaatiolle. [SAC+79] 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.

Join-predikaatin suorittamisessa merkittävää kustannusarvioinnin kannalta on sen suorittava algoritmi sekä join-predikaattien järjestys. Nested loop join- sekä merging scans-algoritmit 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 optimoijan tehtävä on tuottaa mahdollisten kyselysuunnitelmien joukko, arvioida niiden kustannus tilastotiedon pohjalta ja valita niistä paras. Kustannus koostuu kyselysuunnitelman suorittamiseen käytetyistä resursseista.

Kyselysuunnitelmien tuottamisessa kaikkien suunnitelmien läpikäyminen on kuitenkin usein liian hidasta, joten kyselysuunnitelmien joukkoa pyritään karsimaan heuristiikkojen avulla. Heuristiikkojen käyttäminen ei ole triviaalia, sillä toisinaan paras kyselysuunnitelma karsitaan heuristiikan takia.

Kyselysuunnitelmien kustannusarviossa ajankohtainen ongelma on tilastotiedon optimaalinen tallentaminen. Liian tarkka tilastotiedon tallentaminen rasittaa tietokannan käyttöä, mutta auttaa kyselysuunnitelmien optimoinnissa.

Lähteet

[BC02] Bruno, Nicolas ja Surajit 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, Kristin, Michael C. Ferris ja Yannis E. 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.D., M.M. Astrahan, K.P. Eswaran, P. P. Griffiths, R.A. Lorie, J. W. Mehl, P. Reisner ja B.W. 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, Surajit: *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. F.: *A relational model of data for large shared data banks.* Commun. ACM, 13(6):377–387, kesäkuu 1970, ISSN 0001-0782. `http://doi.acm.org/10.1145/362384.362685`.

[DSRS01] Dalvi, Nilesh N., Sumit K. Sanghai, Prasan 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`.

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

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

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

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

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

[Koo80] Kooi, Robert Philip: *The optimization of queries in relational databases.* 1980. AAI8109596.

[LMS94] Levy, Alon Y., Inderpal Singh Mumick ja Yehoshua 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, Guy M.: *Grammar-like functional rules for representing query optimization alternatives.* SIGMOD Rec., 17(3):18–27, kesäkuu 1988, ISSN 0163-5808. http://doi.acm.org/10.1145/971701.50204.

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

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

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

[NSS86] Nahar, Surendra, Sartaj Sahni ja Eugene Shragowitz: *Simulated annealing and combinatorial optimization.* Teoksessa *Proceedings of the 23rd ACM/IEEE Design Automation Conference*, DAC '86, sivut 293–299, Piscataway, NJ, USA, 1986. IEEE Press, ISBN 0-8186-0702-5. http://dl.acm.org/citation.cfm?id=318013.318059.

[OL90] Ono, Kiyoshi ja Guy M 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, Hamid, Joseph M Hellerstein ja Waqar 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, Raghu ja Johannes Gehrke: *Database Management Systems.* McGraw-Hill, Inc., New York, NY, USA, 3 painos, 2003, ISBN 0072465638, 9780072465631.

[SAC+79] Selinger, P. Griffiths, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie ja T. G. 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`.

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