

Tietokantakyselyjen optimointi relaatiotietokannassa

Olli Rissanen

Kandidaatintutkielma

HELSINGIN YLIOPISTO

Tietojenkäsittelytieteen laitos

Helsinki, 17. huhtikuuta 2013

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		17. huhtikuuta 2013	22
Tiivistelmä — Referat — Abstract			
<p>Tutkielmassa tutustutaan tietokantakyselyjen optimointiin relaatiotietokantojen hallintajärjestelmien osalta sekä optimoinnin vaikutukseen kyselyjen suorituskyyvyssä.</p>			
Avainsanat — Nyckelord — Keywords			
Information systems Query optimization			
Säilytyspaikka — Förvaringsställe — Where deposited			
Muita tietoja — Övriga uppgifter — Additional information			

Sisältö

1 Johdanto	3
2 Taustaluku	5
2.1 Relaatiomalli	5
2.2 Relatioalgebran operaatiot	5
2.3 Optimoijan tavoitteet	6
2.4 Optimoijan historia	7
3 Tietokantakyselyn prosessointi	7
3.1 Kyselyn jäsentäminen	8
3.2 Kyselyn optimointi	9
3.3 Kyselyn suorittaminen	10
4 Kyselysuunnitelmien tuottaminen	10
4.1 Kyselysuunnitelma	11
4.2 Hakualgoritmit	13
4.2.1 Dynaamiset algoritmit	14
4.2.2 Muut algoritmit	14
4.2.3 Algoritmien heuristiikat	14
4.3 Alikyselyt	15
5 Kyselysuunnitelmien kustannusarviointi	16
5.1 Systeemitauluston tilastotieto	17
5.2 Tulosityoukon koon arviointi	17
6 Yhteenveto	19
Lähteet	20

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].

Tietokantaa käytetään tietokannan hallintajärjestelmällä, joka on koelma 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]. Kyselyn optimoijan tavoitteena on minimoida itse optimointiin käytetty aika ja maksimoida optimoinnista saatu hyöty [JK84].

Optimoija toimii arvioimalla mahdolliset menetelmät kyselyn suorittamiseen ja valitsemalla niistä tehokkaimman [SAC⁺79]. Menetelmää kyselyn suorittamiseksi sanotaan kyselysuunnitelmaksi, ja se sisältää sarjan tietokannan relaatioihin kohdistuvia algebrallisia operaatioita, jotka tuottavat tulokseksi halutun vastauksen. Tietokantakyselyä vastaavia kyselysuunnitelmia voi olla useita, sillä kysely voidaan usein esittää monena algebrallisesti toisiaan vastaavana esityksenä [JK84]. Algebrallista operaatiota kohden voi

myös löytyä useita toteutuksia, kuten join-operaatiota toteuttavat 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].

Tutkielmassa keskitymme kyselyn optimoijan toimintaan ja ongelmiin yleisellä tasolla. Jokainen kaupallinen optimoija on osittain erilainen kokonaisuus komponentteja, jotka kuitenkin pohjautuvat samoihin yleisiin malleihin. Yksittäistä hallintajärjestelmää käsiteltäessä mainitaan siitä erikseen. Tutkielman tavoitteena on tutustuttaa lukija optimoijan toiminnan perusteisiin ja optimoijan ajankohtaisiin ongelmiin.

Optimointi on vaikea hakuongelma, 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, jonka 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, tuottamisessa käytettäviä algoritmeja sekä heuristiikkoja epäoptimaalisten kyselysuunnitelmien karsimiseen ja parhaan

kyselysuunnitelman valintaa. Kyselysuunnitelman kustannusarviointi ja siihen liittyvä tilastotiedon käyttö käydään läpi luvussa 5.

2 Taustaluku

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

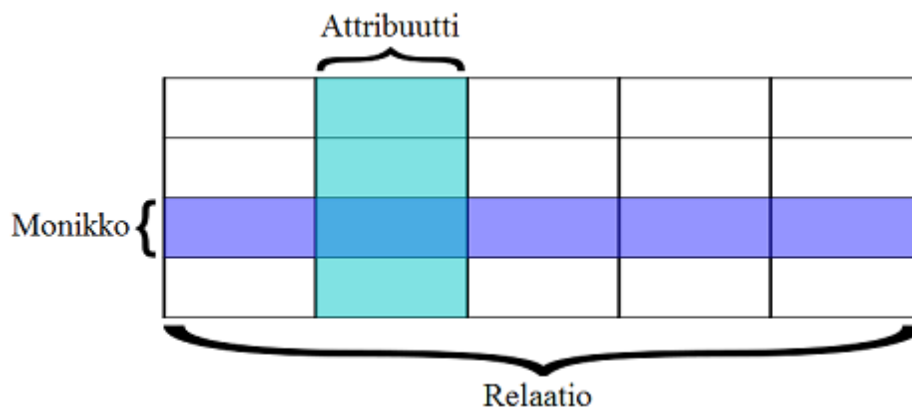
Relaatiotietokanta on relaatiomalliin [Cod70] perustuva tietokanta. Relaatiomallin keskeinen piirre on kaiken datan esittäminen n -paikkaisen karteesisen tulon osajoukkona, ja se tarjoaa deklarativisen menetelmän datan ja kyselyjen määrittämiseen. Relaatiomalli koostuu attribuuteista, monikoista ja relaatioista.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

2.2 Relaatioalgebran operaatiot

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 $B:n$ samoista 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

Kuva 1: Relaatiomalliin perustuva tietokanta



relaation, joka on joukko kaikista relaation monikoista joille pätee valinnan ehto.

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

2.3 Optimoijan tavoitteet

Tietokantakyselyjen optimoinnilla viitataan tietokantakyselyn suorittamiseen mahdollisimman tehokkaasti. Optimoinnin tavoitteena on joko maksimoida suoritussyky annetuilla resursseilla tai minimoida resurssien käyttö. Jarke ja Koch määrittivät artikkelissaan [JK84] mitattaviksi resursseiksi suorittimen ja muistin käytön sekä kommunikointikustannukset. 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 pus-

kurimuistin 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, kun taas paikallisesti hajautetuissa tietokannoissa kaikilla resursseilla on sama painoarvo. 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

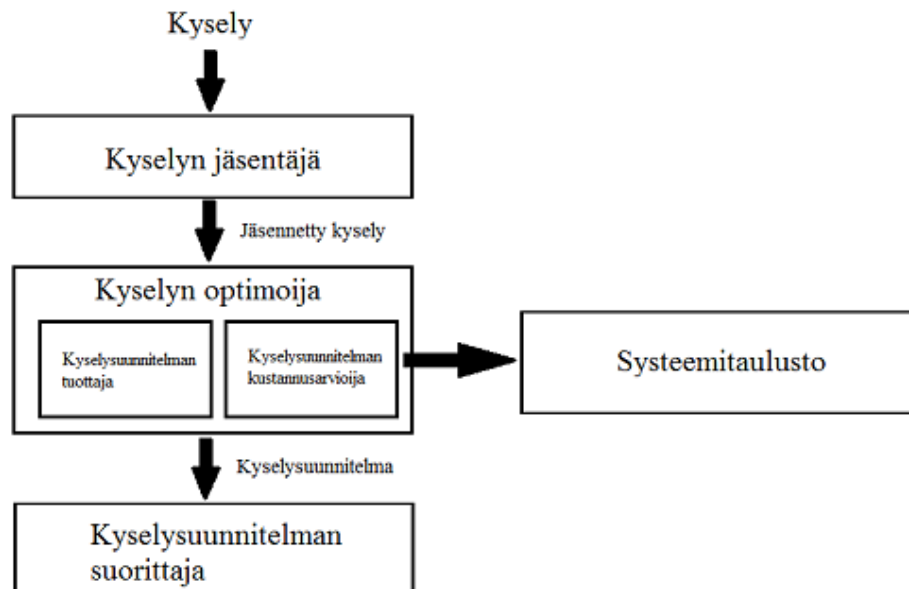
Kun Coddin relaatiomallin pohjalle luotiin sitä toteuttavia kieliä, nousi vuonna 1974 kehitetty SQL nopeasti käytetyimmäksi. SQL:n myötä optimoinnista tuli osa järjestelmän toiminnallisuutta, sillä SQL tarjosi deklarativisuutensa vuoksi useita tapoja toteuttaa kukin kysely. Ennen relaatiomallin käyttöä optimointi tehtiin käsin. Selinger et al. esittelivät artikkelissaan [SAC⁺79] System R-tietokantaohjelmiston OPTIMIZER-komponentin, joka on ensimmäisiä relaatiomalliin pohjautuvia kyselyn optimoijia. OPTIMIZER esittely 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 suorituser-

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

Seuraavaksi jäsentäjä jakaa kyselyn lohkoihin (block) siten, että yhdessä lohossa 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 kappaleessa 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 alisolmu joiden ulostuloa (output) käytetään sen syötteenä. Esimerkiksi join-operaatiossa solmulla on kaksi alisolmuja, joille toteutetaan join-operaatio ja sort-operaatiolla on yksi alisolmu joka järjestetään. Lehtisolmut ovat solmuja jotka suorittavat hakuja (scan) levyiltä ja palauttavat saadut tulokset. Puu suoritetaan lehtisolmuista juureen.

Tämän jälkeen monimutkaiset kyselyt uudelleenkirjoitetaan (rewrite) soveltamalla niihin muutossääntöjä [Ioa96]. Pirahesh et al. tutkivat uudelleenkirjoitusta Starburst-hallintajärjestelmän osalta artikkelissa [PHH92], ja määrittivät uudelleenkirjoituksen tavoitteiksi tehdä kyselyistä mahdollisimman deklarativisia ja suorittaa luonnollisia heuristiikkoja, kuten predikaattien siirtämistä mahdollisimman aikaiseen vaiheeseen.

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. Tilastotietoa käytetään kyselyn suorituskyvyn arviointiin. Saantipolku on menetelmä jolla taulurivit haetaan taulusta, ja se koostuu joko tiedoston lukemisesta tai indeksistä ja siihen liittyvistä valinnoista. Selinger et. al. tutkivat kattavasti saantipolkujen kustannusarviointia

artikkelissaan [SAC⁺79].

Mikäli kyselyn semantiikkaa ei ole vielä tarkastettu, tarkastaa optimoi- ja kyselyn semantiikan oikeellisuuden ja kyselyn tyyppien sopivuuden sarakkeisiin ja operaatioihin systeemitaulun tiedon pohjalta. (Selvennä tästä kappaleen loppuun) Optimoija määrittää lohkojen suoritusjärjestyksen ja prosessoi jokaisen lohkon FROM-lauseeseen liittyvät relaatiot. Mikäli loh- kossa on useampi relatio, arvioi optimoija join-operaation permutaatioiden suorituskyvyt ja valitsee tehokkaimman algoritmin operaation suorittami- seen. Optimoija valitsee jokaiselle lohkolle tehokkaimman saantipolun ja rakentaa niistä kyselysuunnitelman [SAC⁺79]. (koko kappaleen viite)

3.3 Kyselyn suorittaminen

Seuraavassa vaiheessa kysely suoritetaan käyttämällä optimoijan tuotta- maa kyselysuunnitelmaa. Suorittamisessa kyselysuunnitelma muutetaan suo- ritettavaksi konekieleksi ja kyselyn lähteen mukaan joko suoritetaan tai tallennetaan muistiin myöhempää suorittamista varten.

4 Kyselysuunnitelmien tuottaminen

Kyselyn optimoijan tulee tuottaa mahdollisimman hyvän kyselysuunnitelman sisältämä joukko käyttämättä tuottamiseen liikaa resursseja. Kappale on jaettu kolmeen osa-alueeseen, joista aluksi perehdymme kyselysuunnitelman rakenteeseen. Tämän jälkeen tutkimme algoritmeja kyselysuunnitelmien joukon luomiseen, sekä heuristiikkoja epäoptimaalisten kyselysuunnitelmien karsimiseen. Kappaleen 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 relaatio-operaation suoritukseen. 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:

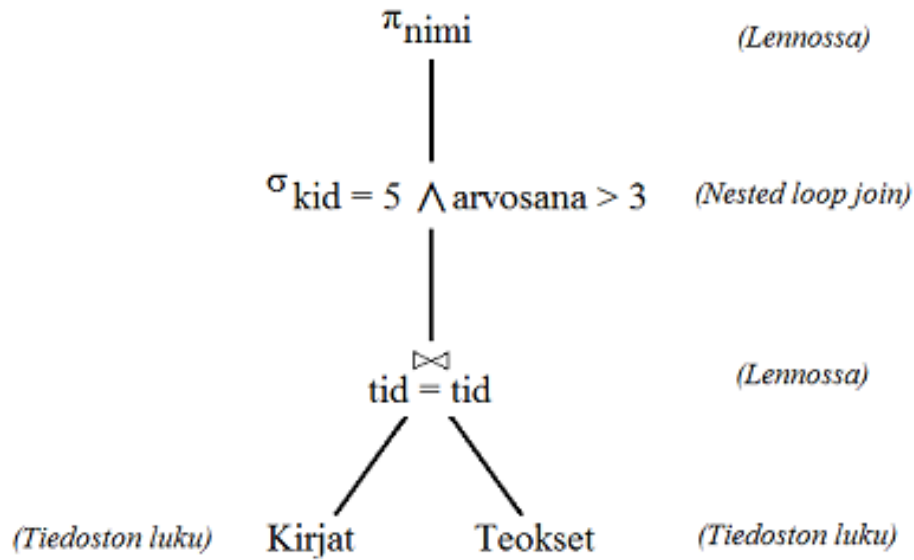
$$\pi_{nimi}(\sigma_{kid=5 \wedge arvosana > 3}(Kirjat \bowtie_{tid=tid} Teokset))$$

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 materialisoida. Putkitusta käytetään usein peräkkäisten join-operaatioiden toteuttamiseen, jolloin jokainen operaation palauttava operandi voidaan syöttää seuraavan operaation parametriksi. Putkitusta voidaan käyttää operaation suorittavan algoritmin salliessa sen.

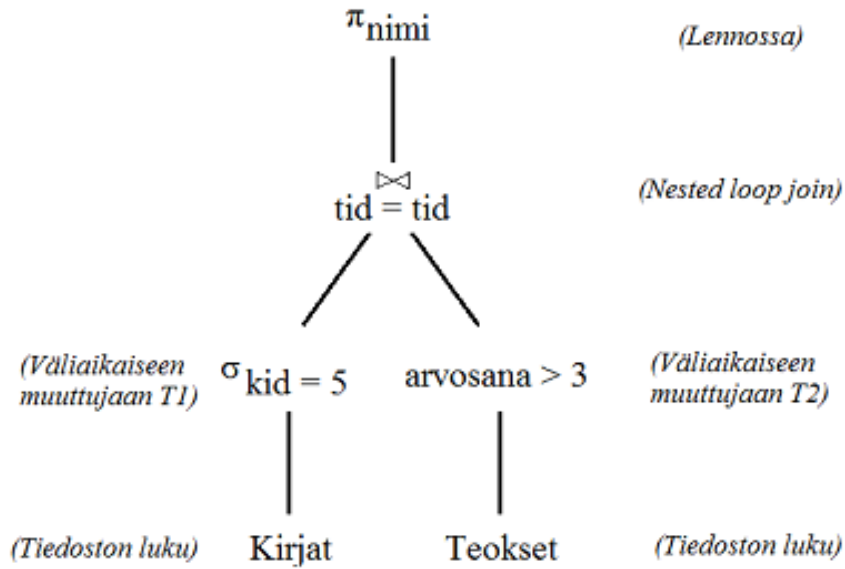
Usein tehokkaampi tapa kyselyn suorittamiseen on suorittaa valinnat mahdollisimman aikaisessa vaiheessa [RG03]. Esimerkkikyselyssä tauluille suoritetaan kullekin valinta erikseen, joten suorittamalla ne ennen join-operaatiota voidaan vähentää yhdistettävien alkioden määrää. Tehokkaampi kyselysuunnitelma aikaisilla valinnoilla, tallentaen valinnat väliaikaismuuttujiin T1 ja

Kuva 3: Kyselysuunnitelma erimerkkikyselylle



T2, esitetään kuvassa 4.

Kuva 4: Toinen kyselysuunnitelma erimerkkikyselylle



Valinnan suorittaminen mahdollisimman aikaisin ei kuitenkaan aina tuo-

ta suorituskykyisintä tulosta. [RG03] Mikäli taulut on indeksoitu, voi valinta kadottaa tärkeät indeksit jolloin join-operaation suorittaminen hidastuu. Paras kyselysuunnitelma riippuu siten täysin taulujen koosta ja niiden indeksoinnista. Edelleen tehokkaampi tapa olisi väliaikaisen muuttujan käytön sijaan putkittaa valinta-operaatiosta alkiot suoraan join-operaatioon.

4.2 Hakualgoritmit

Kyselyn optimoija tarvitsee tehokkaan algoritmin mahdollisten kyselysuunnitelmien luomiseen. Jarke ja Koch määrittelivät 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.

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 kyselysuunnitelmien määrää. Tavoitteena on pitää hakualue mahdollisimman pienenä, sisältäen kuitenkin optimaalisen suunnitelman [JK84]. Luvussa tutustutaan tarkemmin dynaamisiin algoritmeihin sekä vaihtoehtoihin algoritmeihin hakualueen läpikäymiseen, kuten sääntöihin perustuvaan Starburst-hallintajärjestelmälle kehitettyyn algoritmiin [Loh88]. Luvun loppuksi 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 ja karsii alioptimaalisiksi tiedetyt puut.

4.2.2 Muut algoritmit

Dynaamisia algoritmeja on kritisoitu tästä ja tästä [ähän?]

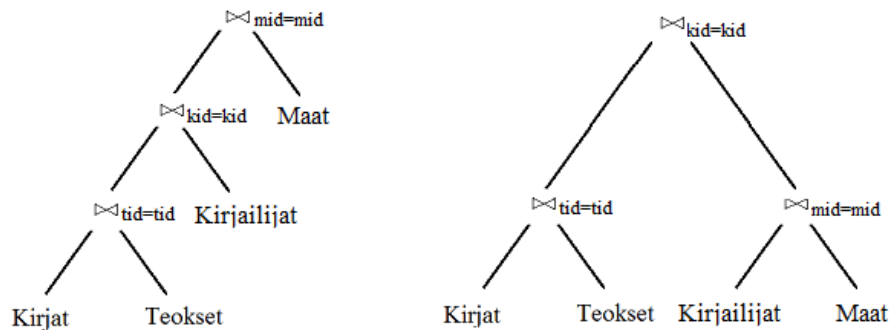
4.2.3 Algoritmien heuristiikat

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. 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 merkittävästi. [ä tutkiva artikkeli?] Toinen syy on vasemmalta syvien puiden mahdollistama putkitus. 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



Ionaddis tutki artikkelissaan [IK91] left-deep puiden ja lehtevien (bushy) puiden suorituskykyeroja, ja huomasi lehtevien puiden optimoinnin olevan helpompaa ja sen tuottavan parempia tuloksia.

4.3 Alikyselyt

Kyselyn predikaatin operandin ollessa toinen kysely puhutaan alikyselystä. Alikyselyt suoritetaan..

Alikysely voi palauttaa yhden arvon tai listan arvoja riippuen sen operandista. Tutkitaan seuraavaa kyselyä:

```

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

```

Esimerkkikyselyssä alikysely palauttaa yhden arvon, AVG(hinta). Tämä

arvo lisätään ylikyselyyn predikaatiksi niinkuin se olisi ollut osa ylempää kyselyä. Esimerkkitapauksessa keskiarvon ollessa 10 predikaatti on ”hinta = 10”.

Mikäli alikysely palauttaa useamman arvon, kuten seuraavassa tapauksessa:

```
SELECT nimi
FROM Kirjat
WHERE hinta IN (SELECT hinta
FROM Kirjat
WHERE genre = 'scifi')
```

käytetään alikyselyn palauttamaa listaa ylikyselyn predikaattina. Esimerkkikyselyn palauttaessa listan (5,15) vastaava ylikyselin predikaatti on ”hinta IN (5,15).

5 Kyselysuunnitelmien kustannusarviointi

Kyselysuunnitelmien kustannusarvioinnissa optimoijan tulee arvioida kyselysuunnitelman operaatioiden suorittamiseen kuluva aika sekä niiden tulosjoukon 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. Kappaleessa 5.1 tutustutaan tarkemmin systeemitaulustoon, ja kappaleessa 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 mahtavuus 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.

Bruno et al. tarkastelivat tutkimuksessaan [BC02] tilastotiedon laajentamista väliaikaisiin tuloksiin, ja he huomasivat ratkaisun tuottavan parempia kyselysuunnitelmia kuin pelkän tietokannan taulun tilastotiedon käytön.

5.2 Tulostuloksen koon arviointi

Operaation kustannus riippuu syötteen koosta. Tutkitaan seuraavaa tapausta:

```
SELECT attribuutit
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 systemitaulustoon tallennettua tilastotietoa. Mikäli relaatiolle ei ole tilastotietoa saatavilla, oletetaan relaation olevan pieni ja käytetään määriteltyä vakiota lasketun kertoimen sijaan [SAC⁺79]. 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 kaavalla $F = \frac{1}{MAX(NKeys(I1), NKeys(I2))}$ 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.

sarake > arvo

Vähennyskerroin voidaan arvioida kaavalla $F = \frac{\text{suurin avain} - \text{arvo}}{\text{suurin avain} - \text{pienin avain}}$. Mikäli sarake ei ole aritmeettinen tai arvoa ei tiedetä käytetään vakiona arvoa 1/3.

sarake IN (lista arvoja)

$F = \text{koko(lista)} \times (\text{sarake} = \text{arvo})$

6 Yhteenveto

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*, sivut 263–274. ACM, 2002.
- [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*, sivut 34–43. ACM, 1998.
- [Cod70] Codd, Edgar F: *A relational model of data for large shared data banks*. Communications of the ACM, 13(6):377–387, 1970.
- [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*, sivut 59–70. ACM, 2001.
- [IK91] Ioannidis, Yannis E ja Younkyung Cha Kang: *Left-deep vs. bushy trees: An analysis of strategy spaces and its implications for query optimization*. Teoksessa *ACM SIGMOD Record*, nide 20, sivut 168–177. ACM, 1991.

- [Ioa96] Ioannidis, Yannis E: *Query optimization*. ACM Computing Surveys (CSUR), 28(1):121–123, 1996.
- [JK84] Jarke, Matthias ja Jurgen Koch: *Query optimization in database systems*. ACM Computing surveys (CsUR), 16(2):111–152, 1984.
- [Koo80] Kooi, Robert Philip: *The optimization of queries in relational databases*. 1980.
- [Loh88] Lohman, Guy M: *Grammar-like functional rules for representing query optimization alternatives*, nide 17. ACM, 1988.
- [MCS88] Mannino, Michael V, Paicheng Chu ja Thomas Sager: *Statistical profile estimation in database systems*. ACM Computing Surveys (CSUR), 20(3):191–221, 1988.
- [MJ12] Mahajan, S.M. ja V.P. Jadhav: *An analysis of execution plans in query optimization*. Teoksessa *Communication, Information Computing Technology (ICCICT), 2012 International Conference on*, sivut 1–5, 2012.
- [MKR12] Mor, Jyoti, Indu Kashyap ja RK Rathy: *Analysis of Query Optimization Techniques in Databases*. International Journal, 47, 2012.
- [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.

- [Ora09] Oracle: *Oracle Database Online Documentation 11g Release 1*, 2009. http://docs.oracle.com/cd/B28359_01/server.111/b28318/sqlplsql.htm.
- [Ora13] Oracle: *MySQL 5.0 Reference Manual*, 2013. http://docs.oracle.com/cd/E17952_01/refman-5.0-en/controlling-optimizer.html.
- [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, R. ja J. Gehrke: *Database Management Systems*. McGraw-Hill international editions: Computer science series. McGraw-Hill Education, 2003, ISBN 9780072465631. <http://books.google.fi/books?id=JSVhe-WLGZ0C>.
- [SAC⁺79] Selinger, P Griffiths, Morton M Astrahan, Donald D Chamberlin, Raymond A Lorie ja Thomas G Price: *Access path selection in a relational database management system*. Teoksessa *Proceedings of the 1979 ACM SIGMOD international conference on Management of data*, sivut 23–34. ACM, 1979.
- [TF82] Teorey, Toby J ja James P Fry: *Design of database structures*. Prentice Hall Professional Technical Reference, 1982.