

4. Kyselyjen käsittely ja optimointi

(E&N, Ch. 18)

- taustana mm. relaatioalgebra ja hakemistorakenteet

Kyselyn käsittelyn vaiheet:

- käyttöliittymästä tai jostakin ohjelmasta tulee SQL-kysely Q tkhj:lle
- kysely Q selataan ja jäsennetään; tarkistetaan sen oikeellisuus (kaaviotiedot: relaatio- ja sarakenimet jne)
- jäsennetty kysely muunnetaan relaatiolausekkeeksi ja tallennetaan sisäisessä esitysmuodossa, kaavapuuna (kyselypuuna tai -graafina)
- muodostetaan kyselyn kyselysuunnitelma (query plan, execution plan);
 - suunnitelman tekoon liittyy 'optimointia': valitaan vaihtoehtojen välillä (esim. mitä hakemistoja käytetään)
- kyselysuunnitelma suoritetaan (koodingenerointi + suoritus tai tulkinta)
- käyttäjä tai ohjelma saa kyselyn tuloksen

Kyselyn optimoinnin pääperiaatteet:

1. **Heuristinen optimointi:** käytetään yleisiä sääntöjä, joilla yleensä pyritään pienentämään käsittelyyn liittyviä välituloksia (vähentämään levyoperaatioita)

- järjestetään operaatiot sopivasti, esim. valinnat ennen liitoksia,
- käytetään hakemistoa aina kun sellainen on
- yleisyys: ei tarvita tietoja taulujen sisällöistä (eikä järjestelmäkohtaisista asioista)

heuristinen: kokeileva, ei-teoreettinen

- pätee yleensä hyvin
- ei takaa välttämättä optimaalista toimintaa
- 'nyrkkisääntö', enemmänkin

2. **Kustannuslaskentaoptimointi:**

- generoidaan kyselylle vaihtoehtoisia kyselysuunnitelmia
- lasketaan vaihtoehtojen kustannukset (levyhaut, muisti)
- valitaan edullisin tilastotiedot (esim. arvojakaumat) lasketaan aika ajojin; arvioiden tarkkuus vaihtelee

Käytännössä tkhj:n kyselyn optimoija on yhdistelmä kummankin periaatteen mukaisia toimia.

Muunnoksen (optimoinnin) perusteet:

- SQL on kielenä deklaraatiivinen ('mitä', ei 'miten'), relaatioalgebra kertoo operaatioiden järjestyksen
- yleiset säännöt, periaatteet ('heuristics')
- suoritusajaiset tiedot taulujen ominaisuuksista (jopa attribuuttiarvojen jakaumat)

Todellisen (matemaattisen) optimoinnin sijasta kysymys on suorituksen parantamisesta, 'riittävän hyvän' suoritustavan valinnasta: 'reasonably efficient'

- todellinen optimointi olisi laskennallisesti raskasta
- taustatiedot puutteelliset
 - tallennusrakenteiden yksityiskohdat,
 - taulujen sisältö

Kyselyn optimointi liittyy nimenomaan relaatiomallin ideaan ja SQL:n deklaraatiivisuuteen. Oliotietokannoissa ja perinteisissä verkkotietokannoissa kyselyt esitetään 'tarkemmin': olioiden t. tietueiden yhteydet on määritelty, kyselyssä navigointi niiden välillä.

Deklaratiivisuus on kuitenkin suhteellinen asia:

- esimerkiksi kyselyn kirjoitustavalla (osien järjestyksellä) voi käytännössä olla merkitystä
 - käyttäjä optimoi
 - yleensä järjestelmäkohtaista, epävarmaa
 - tavallinen käyttäjä tuntee harvoin perusteita

4.1 Yleisiä periaatteita

Relaatioalgebran operaatiot ja niiden tuloksen koko (relaatiot r ja s, |r| ja |s| riviä):

| | |
|---|------------------|
| valinta $\sigma_F(r)$: ehdon F täyttävät rivit | $\leq r $ |
| projektio $\pi_X(r)$: sarakkeet X | $\leq r $ |
| yhdiste $r \cup s$: jommankumman rivit | $\leq r + s $ |
| leikkaus $r \cap s$: yhteiset rivit | $\min(r , s)$ |
| erotus $r - s$: vain r:ssä olevat | $\leq r $ |

kartesinen tulo $r \times s$:

kaikki attribuuttiarvojen yhdelmät $|r| \cdot |s|$

liitos $r \bowtie s$: karteesisen tulon ehdon F täyttävä osa $\leq |r| \cdot |s|$
(yleensä $\ll |r| \cdot |s|$)

SQL-kyselyn käänös relaatioalgebran operaatioiksi: select-, from- ja where- osien yhdistelmä muodostaa peruskokonaisuuden, ns. kyselylohkon (query block). Sisäkyselyjen tapauksessa kyselylohkot voidaan erottaa (jos osat riippumattomia):

```
Esim.  select lname, fname
        from employee
        where salary >
          (select max(salary)
           from employee
           where dno=5);
```

sisältää kaksi kyselylohkoa, joiden suoritus voidaan suunnitella (optimoida) erikseen.

Vastaavat relaatiolausekkeet:

$$\pi_{\text{name, fname}}(\sigma_{\text{salary} > c}(\text{employee})),$$

$$F_{\text{max salary}}(\sigma_{\text{dno}=5}(\text{employee}))$$

(F on laajennetun relaatioalgebran funktio-operaatio)

Lajittelu (järjestäminen):

- yleinen 'hyvin määritelty' operaatio, jolla on monta käyttötilannetta:
 - tuloksen järjestäminen (order by -kysely)
 - keino parantaa operaation tehokkuutta, ellei ole sopivaa hakemistoa (jonka merkitys usein juuri järjestyksen ylläpitäminen)

Levylajittelu = ulkoinen lajittelu; standardimenetelmä on lomitussajittelu (merge sort):

1° luetaan puskurinkokoon sopiva määrä rivejä levytä ja lajitellaan ne sisäisellä lajittelulla nousevaan järjestykseen - saadaan joukko järjestettyjä rivijonoja (run), jotka viedään peräkkäin levyille

2° lomitetaan jonot yhdeksi järjestetyksi jonoksi peräkkäisissä vaiheissa:

- yhdessä vaiheessa m ($m \geq 2$) jonoa lomitetaan m kertaa yhden syötejonon pituiseksi järjestetyksi jonoksi (vrt. luku 2, s. 25)
- viimeisessä vaiheessa tuloksena on yksi järjestetty jono (yksi levytiedosto)

Lomituksen aikavaativuus on maksimissaan $O(n \log_m n)$, missä n = vaiheen 1 tuloksena oleva jonojen määrä.

Relaatioalgebran operaatioiden suoritus:

- Erillisille operaatioille voidaan johtaa monia suoritusvaihtoehtoja, joiden tehokkuus vaihtelee. valinta, projektiio, liitos
- Tehokkuuden ensisijainen kriteeri: tarvittavien levyoperaatioiden määrä
 - luettaessa operandirelaation osia levytä, ja
 - kirjoitettaessa operaation tulosta (koko lausekkeen kannalta välitulosta) levyille.
- Välituloksen levyille kirjoitus voidaan usein jättää huomiotta: tulosta käytetään 'putkittain' (pipelining) seuraavan operaation syötteenä.
- Levyoperaation koon arviointi: taulun tai välituloksen rivimäärä tai tavumäärä; näistä jaksojen määrä

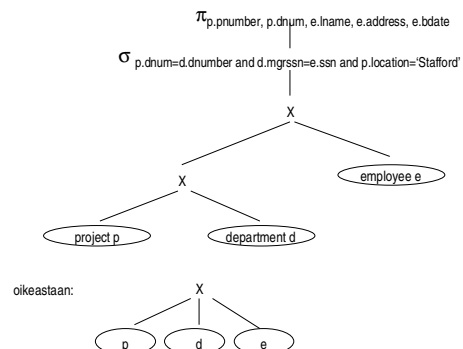
4.2 Heuristinen optimointi

(Kyselylohkon) operaatioiden suoritusjärjestys kuvataan tyypillisesti kyselypuun (kaavapuun) avulla: puun juuri antaa tuloksen, jota muodostetaan aloittamalla puun lehtisolmujen välisistä operaatioista.

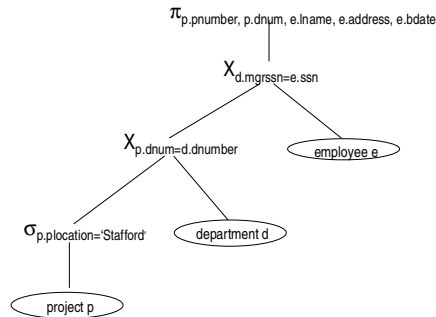
```
Esim.  select pnumber, dnum, lname, address, bdate
        from project, department, employee
        where dnum = dnumber and mgrssn = ssn
        and plocation = 'Stafford';
```

(Staffordin projektit ja vastaavien osastojen johtajat)

Standardimuotoista relaatiolauseketta (projektiio-valinta-tulo) vastaava kyselypuu:



Samaan tulokseen johtava optimoitu kyselypuu:



Tässä

- 1° valinnat suoritetaan mahdollisimman aikaisin
- 2° karteesinen tulo ja sitä seuraava yhtäläisyysvalinta on yhdistetty yhtäläisyysliitokseksi.

Kumpikin muutos pienentää välituloksia. Alkuperäinen muoto on karteesisen tulon muodostamisen takia hyvin tehoton!

Projektiot kannattaa myös yleensä suorittaa aikaisin!

Heuristisen optimoinnin algoritmi:

1. Puretaan konjunktiiiset valinnat säännön 1 mukaan peräkkäisiksi valinnoiksi.
2. Painetaan valinnat niin alas kyselypuussa kuin mahdollista; valinnan vaihdannaisuus: säännöt 3, 4, 6 ja 9.
3. Järjestetään lehtisolmut sääntöihin 5 ja 11 perustuen.
 - Suoritetaan ensin mahdollisimman rajoittavat valinnat:
 - tulos pienin (rivejä tai tavuja)
 - pienin valitsevuus (selektiivisyys)
 - Vältetään karteesisen tulon muodostusta (yleensä). (Karteesinen tulo ei ole suuri, jos toisessa relaatiossa on jopa vain yksi rivi – esimerkiksi edeltävän avainvalinnan takia.)
4. Yhdistetään karteesinen tulo ja seuraava valinta liitokseksi (12).
5. Painetaan projektioperaatiot niin alas puussa kuin mahdollista (säännöt 2, 4, 7).
Huom. Operaatioiden tarvitsemat attribuutit on kuitenkin säilytettävä.
6. Yhdistetään operaatiot (alipuu), jotka voidaan suorittaa yhdellä algoritmilla (esim. liitos, syötteiden luku, tuloksen kirjoitus)

Yleisiä heuristisen optimoinnin muunnoksia:

1. Moniehtoinen valinta peräkkäisiksi:

$$\sigma_{c1 \text{ and } c2 \text{ and } \dots \text{ and } cn}(r) = \sigma_{c1}(\sigma_{c2}(\dots(\sigma_{cn}(r))\dots))$$

2. Vain viimeinen projekti peräkkäisistä tarvitaan.

Vaihdannaisia operaatioita:

3. Peräkkäiset valinnat
4. Valinta ja projekti keskenään (kun projektion attribuutit riittävät valintaan).
5. Karteesinen tulo (X); liitos (tulon, liitoksen osat)
6. Valinta ja liitos (kun valinta kohdistuu vain toisen relaation attribuutteihin) (samoin X)
7. Projekti ja liitos (kunhan projektiossa ei hävitetä liitosattributteja!) (X: vaihto sopii aina)
8. Yhdiste, leikkaus (erotus: ei)
9. Valinta ja joukko-operaatiot
10. Projekti ja yhdiste (leikkaus, erotus: ei)
11. Assosiativisuus: yhdiste, leikkaus, liitos, karteesinen tulo
12. Karteesista tuloa seuraava valinta voidaan muuntaa liitokseksi (edell. esimerkki).
13. Moniosainen valinta- tai liitosehto voidaan muuntaa DeMorganin lakien mukaan:
 - not (c1 and c2) = (not c1) or (not c2)
 - not (c1 or c2) = (not c1) and (not c2)

Valitsevuus (selectivity):

Valitsevuus = tuloksen rivien lukumäärän suhde relaation kaikkien rivien lukumäärään (arvo välillä [0..1]).

Avaimen kohdistuva valintaehto on kaikkein valitsevin; muista ominaisuuksista (attribuuttien arvojakumista) saattaa olla tietoa tietohakemistossa. Kyselytilanteessa käyttäjällä saattaa olla tarkempaa tietoa valitsevuudesta; sitä on vaikea käyttää yleisesti.

Valitsevuuden arvioita (relaatiossa r monikko):

- yhtäsuuruuskyselyssä avaimelle 1/r
- yhtäsuuruuskyselyssä attribuutille, jolla on i eri arvoa, 1/i (oletus tasaisesta jakaumasta)

Tuloksessa on vastaavasti 1 tai r/i riviä.

Esim.

$$\sigma_{lname='Smith' \text{ and } dno=5}(employee):$$

Attribuutin lname valitsevuus varmaan pienempi, siis:

$$\sigma_{dno=5}(\sigma_{lname='Smith'}(employee))$$

(hakemistojen osuus myöhemmin)

4.3 Valintaoperaation toteutus

- useita vaihtoehtoisia tapoja riippuen
 - saantipoluista (hakemistoista),
 - valintaehdoista (F)

1° tiedoston läpikäynti eli lineaarinen haku:
luetaan tiedoston kaikki sivut ja testataan
valintaehto joka tietueelle
(perusvaihto: soveltuu aina, mutta on hidas)
esim. $\sigma_{\text{Iname} = \text{Max}}(\text{employee})$

2° binäärihaku: soveltuu attribuutin A mukaan
järjestetylle tiedostolle, kun valintaehto on muotoa
 $A = v$ tai $v1 \leq A \leq v2$ ($v1$ ja $v2$ vakioita)
esim. $\sigma_{\text{ssn}=123456789}(\text{employee})$, jos monikot
ssn-järjestyksessä

3° hakemiston käyttö: mikä tahansa hakemisto (ISAM,
hajautus, B+ -puu) soveltuu täsmällisen
valintaehdon $A = v$ toteutukseen

4° järjestykseen perustuva hakemisto (ISAM, B+ -puu)
attribuutille A tai attribuuttiyhdistelmälle (A, B, ...)
soveltuu täsmällisen ehdon lisäksi
- osaväliehdolle $v1 \leq A \leq v2$
- likimääräiselle ehdolle like v, kun v sisältää
attribuuttiarvon alkuosan

Hakemiston valintaan vaikuttaa myös, onko se ryvästävä
(järjestävä).

Jos esim. on ryvästävä dno-hakemisto ja (ei-ryvästävä)
Iname-hakemisto, kannattaa valitsevuudesta huolimatta
ehkä käyttää dno-hakemistoa.
- voi olla esim. 5 sivullista osaston 5 työntekijöitä, mutta
10 erikseen haettavaa Smith-nimistä

Jopa koko tiedoston läpikäynti ryvästysjärjestyksessä
peräkkäin voi olla tehokkaampaa kuin ei-ryvästävän
hakemiston käyttö, jos valintaehdon täyttäviä monikkoja
on paljon.

Funktion käyttö kyselyssä saattaa rajoittaa hakemiston
käyttöä:

Esim.

```
select * from employee
where upper(Iname) = 'SMITH'
```


ei käytä Iname-hakemistoa,


```
select * from employee
where Iname = upper('Smith')
```


käyttää.

Yleistys: Valintaehdon rinnalla voi olla konjunkttiivinen
lisäehto: esim. $A = v$ and G (**riippumatta G:stä**). Ehdon
G voimassaolo testataan jokaiselle valitulle tietueelle.

Jos on hakemistoja, on valittava valitsevuudeltaan paras:

Esim. employee: ssn-, Iname- ja dno-hakemistot:
 $\sigma_{\text{ssn}=123456789}$ and $\text{dno}=5(\text{employee})$ ssn-hakemistolla

$\sigma_{\text{Iname}='Smith'}$ and $\text{dno}=5(\text{employee})$ Iname-
hakemistolla

Jos ei ole Iname-hakemistoa, dno-hakemistolla.

Tietueosoitteiden käyttö: oletetaan, että
- valintaehdossa on konjunkttiivisesti yhdistetyt
attribuutit A ja B,
- kummallekin attribuutille on erillinen oheishakemisto,
- hakemistossa on tietueosoitteet (eikä vain jakso-
osoitteita).

Tällöin voidaan muodostaa hakemistojen avulla
tietueosoinlistojen leikkaus ja vasta tässä (pienessä?)
leikkausjoukossa olevat tietueet haetaan käsittelyyn:
- muihin attribuutteihin kohdistuvat ehdot testataan ja
muodostetaan tulos

Disjunkttiivisesti yhdistetyt valintaehdot, esim.

$\sigma_{\text{dno}=5 \text{ or } \text{salary}>30000}(\text{employee})$:
tulos on erillisillä ehdoilla saatavien tulosten yhdiste.
Optimointimahdollisuuksia vähän; yksikin ehdossa
esiintyvä hakemistoton attribuutti pakottaa selaamaan
relaation läpi. (Disjunkttiiviset ehdot eivät ole yhtä yleisiä
kuin konjunkttiiviset.)

4.4 Projektio - operaation toteutus

Tulokseen kuuluvat kaikki monikot, paitsi jos projektiossa
syntyisi duplikaatteja. Luonnollinen toteutustapa on koko
tiedoston läpikäynti ja projektioehdon täyttävien rivien
tulostaminen.

Jos projektiosarakkeet sisältävät taulun avaimen,
duplikaatteja ei synny.

SQL-kyselyssä duplikaatteja ei välttämättä haluta
poistaa:

$\text{vrt. select } x \text{ from } r$ vs. $\text{select distinct } x \text{ from } r$

Duplikaattien poisto on epätriviaali toimenpide:

- naiivi ratkaisu on verrata jokaista tulosrivää jokaiseen
jo tulokseen kirjoitettuun ($O(n^2)$)!
- normaali tapa on lajitella tulosrivit: duplikaatit tulevat
peräkkäin ja muut paitsi ensimmäinen voidaan helposti
poistaa ($O(n \log n)$)
- myös hajautus käyttökelpoinen: duplikaatit osuvat
välttämättä samaan kotisoluun, joten vertailtavaa on
vähän

Projektio voidaan usein yhdistää edeltävän operaation,
esim. liitoksen, tuloksen muodostamiseen.

4.5 Liitosoperaation toteutus

- tuloksen ('neliöllinen') koko ja operaation yleisyys
=> liitos on tehokkuuden kannalta tärkein operaatio

Yleisin liitostyyppi on viiteavaimeen perustuva rivien yhdistäminen, esimerkiksi työntekijän osaston nimi:

```
select lname, dname from employee, department
where lname='Smith' and dno=dnumber;
```

Tarkastellaan yleisesti kahden relaation liitosta (two-way join).

Neljä perusmenetelmää:

- 1) sisäkkäiset silmukat (nested-loop join),
- 2) hakemistoliitos (indexed join, single-loop join),
- 3) järjestämisliitos (lomituliitos; sort-merge join),
- 4) hajautusliitos (hash join).

Tarkastellaan vain yhtäläisyysliitosta ja oletetaan seuraavassa:

- liitossarakkeet ovat A ja B: $r \bowtie_{A=B} s$
- puskuritilaa on käytettävissä $M+1$ sivulle

Pienempi relaatioista (jaksojen määrässä) kannattaa valita ulompaan silmukkaan.

Esim. employee 2000 jaksoa
department 10 jaksoa
puskuritila: $M+1 = 7$ jaksoa

- joko $2000 + 10 \lceil 2000/5 \rceil = 2000 + 10 \cdot 400 = 6000$
tai $10 + 2000 \lceil 10/5 \rceil = 10 + 2000 \cdot 2 = 4010$
jakson lukua

Valitsevuudella ei ole merkitystä, kun ei ole hakemistoja.

Jos tulos voidaan putkittaa seuraavaan operaatioon, levyille kirjoitusta ei erikseen tarvita.

Jos puskuritilaa on riittävästi (ulompi relaatio kokonan puskurissa), levylukuja on $m + n$ ("one-pass join"); yleensä enemmän.

Sisäkkäiset silmukat on aina periaatteessa sopiva menetelmä, mutta voi olla tehoton.

1° Sisäkkäiset silmukat

Jokaista riviä verrataan jokaiseen muuhun ja muodostetaan tulosrivi, jos liitosehto on voimassa.

Jaksojen lukeminen voidaan minimoida, kun toisen relaation rivit luetaan vain kerran (ulommassa silmukassa) ja toisen relaation rivejä verrataan samalla kertaa kaikkiin puskurissa oleviin riveihin. Puskuritilan määrä jakaa ulomman silmukan relaation (r) lukemisen (yleensä) useaan erään; erän koko on $M-1$ sivua, ja erien lukumäärä $\lceil m/(M-1) \rceil$.

Jokaista r:n erää kohti luetaan läpi koko relaatio s.

Yksi puskurisivu varataan toiselle relaatiolle (sisempi silmukka), yksi tulosrelaation kirjoittamiseen.

```
for jokainen r:n erä <P1, ..., PM-1> do
  Br,i := bufferfix(Pi), i = 1, ..., M-1;
for jokainen s:n sivu Q do
  Bs := bufferfix(Q);
  muodosta kaikki liitosmonikot tu, missä
  t ∈ Br,1 ∪ ... ∪ Br,M-1, u ∈ Bs ja t(A) = u(B);
  bufferunfix(Bs);
  bufferunfix(Br,i), i = 1, ..., M-1; % erä valmis
end.
```

Levyhakuja kertyy relaatioiden lukemisesta

$m + n \cdot \lceil m/(M-1) \rceil$, jos $|r| = m$ ja $|s| = n$ (jaksoa).

2° Hakemistoliitos

Jos edes toiselle relaatioista (esim. s) on liitosattribuuttiin perustuva hakemisto, operaatio voidaan suorittaa edellistä tehokkaammin:

- toinen relaatio (esim. r) käydään läpi sivu sivulta (ja rivi riviltä)
- toisen relaation (s) vastinrivit haetaan hakemiston avulla

$(r \bowtie_{A=B} s)$

```
for jokainen r:n sivu P do % operoi levyllä
  Br := bufferfix(P);
  for jokainen Br:n monikko t do % puskurissa
    for jokainen s:n sivu Q, jolla on % hakemistosta
      ehdon t.A = B täyttäviä monikoita do
        Bs := bufferfix(Q);
        muodosta kaikki liitosmonikot tu, missä
        u ∈ Bs ja t.A = u.B;
        bufferunfix(Bs);
        bufferunfix(Br);
      end.
```

Algoritmissa on vain yksi levyhakuihin liittyvä silmukka (uloin; 'single-loop join'). Toinen silmukka kohdistuu sivun läpikäyntiin (ei uusia levyhakuja) ja sisin relaation s B-hakemiston käyttöön (levyhakuja likimain vakiomäärä, esim. ylivuotoketjuista riippuen).

Hakemiston valinta hakemistoliitoksessa?

Esim. employee 10000 riviä, 2000 jaksoa;
ssn-hakemisto, tasoja 4
department 50 riviä, 10 jaksoa
mgrssn-hakemisto, tasoja 2

1° haetaan jokainen employee-sivu ja käytetään mgrssn-hakemistoa

$$2000 + 10000 \cdot (2+1) = 32000 \text{ levyhakua}$$

2° haetaan jokainen department-sivu ja käytetään ssn-hakemistoa

$$10 + 50 \cdot (4+1) = 260 \text{ levyhakua}$$

Silmukassa kokonaan käsiteltäväksi valitaan relaatio, jossa on vähemmän jaksoja.

Toinen tekijä: liitososallistuvuus (join selection factor) = liitokseen osallistuvien rivien osuus relaatiossa.

Tässä jokainen department-rivi osallistuu liitokseen eli relation department liitososallistuvuus on 1; employee-relaatiolle vastaavasti 50/10000 = 0.005. Suuri liitososallistuvuus: hakemiston teho huono eli tällainen relaatio uloimmaksi.

Yleisesti tarvitaan siis

$$m + n_r \cdot d \text{ levyhakua, missä}$$

m = läpikäytävän relaation jaksojen määrä,

n_r = saman relaation rivien määrä ja

d = "hakemistopolun" pituus (tarvittaessa myös hakemiston perustaso ja mahdolliset ylivuotoketjut).

Esim. employee, department

- lomitus $2000 + 10 = 2010$ jaksohakua

- järjestäminen ($M=6$):

$$2000 \lceil \log_6 2000 \rceil + 10 \lceil \log_6 10 \rceil$$

- yhteensä

$$10000 + 20 + 2000 + 10 = 12030 \text{ jaksohakua}$$

Lomituksen $m + n$ vaatii, että puskuritila (M) riittää kaikkien tietyllä liitosattribuutin arvolla varustettujen tietueiden pitämiseen samanaikaisesti puskurissa (ainakin toisessa relaatiossa).

Usein liitosattribuutti on toisen relaation avain (esimerkiksi viiteavaimeen perustuva liitos).

Käyttämällä tätä hyväksi lomitus vaatii vain 3 jakson puskuritilan.

Lomituksen perusalgoritmi on tiedostojen luvun tahdistuksen osalta sopiva myös järjestämiseen (lajitteluun), kun avainvertailujen yhteydessä suoritettavia toimenpiteitä muutetaan:

if $r(i).A > s(j).A$ then vie $s(j)$ tulokseen ja aseta $j:=j+1$

if $r(i).A <= s(j).A$ then vie $r(i)$ tulokseen ja aseta $i:=i+1$

- vastaavasti duplikaattien poisto

(ja peräkkäistiedoston päivitys ja ...)

3° Järjestämisliitos (lomitusliitos)

Lomitusalgoritmi sopii hyvin liitoksessa tarvittavaan rivien väliseen vertailuun, mikäli kumpikin relaatio on järjestyksessä liitosattribuutin mukaan:

- luetaan kumpikin relaatio läpi attribuuttiarvojen mukaan tahdistetusti ja muodostetaan tulosrivit

Perusalgoritmi:

(merk. $r(i)$ on relaation r i:s tietue jne)

$i:=j:=1$;

while $i \leq m$ and $j \leq n$ do

if $r(i).A > s(j).B$ then $j:=j+1$

else if $r(i).A < s(j).B$ then $i:=i+1$

else

muodosta tulosrivi $\langle r(i), s(j) \rangle$;

$i:=i+1$; $j:=j+1$

end;

Välttämätön laajennos:

- samaan A- tai B-arvoon voi liittyä useita vastinrivejä

Jos r tai s ei ole aluksi järjestyksessä, se täytyy järjestää.

Järjestämisen kustannus:

$$2m \lceil \log_m m \rceil + 2n \lceil \log_m n \rceil \text{ jaksohakua}$$

Lomituksen kustannus:

$$m + n \text{ jaksohakua (+ mahdollisesti tuloksen kirjoitus)}$$

Myös ulkoliitos (outer join) voidaan laskea lomitusalgoritmia hieman muuntamalla.

Esim. vasen ulkoliitos

$$r \text{] } > < \mid_{A=B} S$$

if $r(i).A > s(j).B$ then $j:=j+1$

else if $r(i).A < s(j).B$ then

muodosta tulosrivi $\langle r(i), \text{null}, \text{null}, \dots \rangle$

ja aseta $i:=i+1$

else ... (liitoksen normaali tulosrivi)

Ulkoliitos voidaan laskea myös muilla kuin lomitusliitoksella. Kun toisen relaation kaikki monikot tulevat tulokseen, tämän relaation läpikäynti valitaan ulompaan silmukkaan menetelmässä 1° ja ainoaksi silmukaksi menetelmässä 2°.

Ulkoliitos voidaan laskea myös hajautusliitoksella (menetelmä 4°) tai peräkkäisillä relaatioalgebraoperaatioilla:

- lasketaan tavallinen liitos, esim. $t1 \leftarrow r \text{] } > < \mid s$

- lasketaan $t2 \leftarrow \pi_x(r) - \pi_x(t1)$

(liitoksen kuulumattomat monikot)

- jatketaan $t2$:n monikkoja null-arvoilla

- vasen ulkoliitos saadaan: $t1 \cup t2$

- tässä yhdisteessä ei ole duplikaatteja (ei tarvitse tarkistaa)

4° hajautusliitos (hash join)

Jompikumpi tai molemmat liitosrelaatioista hajautetaan liitosattribuutin perusteella -> helpotetaan vastinrivien löytämistä.

Useita variaatioita, esim. seuraavasti:

1. hajautetaan relaation r rivit (eli ositetaan relaatio r kotisolujen määrän ilmaiseisiin osiin $r_0, r_1, r_2, \dots, r_{B-1}$)
2. käydään läpi relaatio s : lasketaan jokaiselle riville $s(i)$ samalla hajauttimella sen kotiosoite $h_i (= r_k$ jollakin $k:n$ arvolla) kaikki rivin $s(i)$ vastinrivit saadaan kotisolusta r_k

Menetelmä on sitä tehokkaampi, mitä suurempi osa vaiheessa 1 hajautetuista riveistä voidaan pitää puskureissa koko vaiheen 2 ajan.

→ kannattaa valita vaiheessa 1 pienempi relaatio, joka ehkä mahtuu jopa kokonaan puskuriiin

Em. hajautusliitoksen idea on lähellä hakemistoliitosta: vaiheen 2 silmukka vastaa ainoa relaation läpikäyntiä, vaiheessa 1 muodostettu hajautusrakenne tavallaan hakemistoa.

Arvio esimerkkitapaukselle:

10 + 2000 jaksonlukua, jos puskuritila riittää

Koostefunktiot COUNT, SUM, MIN, MAX, AVG:

- relaation läpikäynti, tai
- koosteattribuuttiin liittyvän hakemiston käyttö

Esim. `select max(salary) from employee;`

Jos on tiheä salary-hakemisto, funktion arvo voidaan laskea hakemistosta (käymättä perustietueissa).

MIN ja MAX voidaan (ehkä) laskea myös harvasta hakemistosta, ja ainakin selvittämällä hakemiston avulla se ainoa perustiedoston jakso, jossa pienin tai suurin arvo on.

COUNT, AVG, SUM: hakemistossa on ehkä jokaista hakemistoavaimen arvoa vastaavien tietueiden määrä, joten voidaan selvittää ilman perustietueita

Jos MIN, MAX -kyselyjä paljon, saattaa olla kannattavaa ylläpitää hakemistotietueissa myös vastaavan alipuun minimi- ja maksimiarvoja (menee lisätilaa ...).

4.6 Muut operaatiot

Karteesinen tulo: koko $|r| \cdot |s|$ riviä

eli vähintään $m \cdot n$ jaksoa

- pyritään välttämään

Joukko-operaatiot (yhdiste, leikkaus, erotus):

- voidaan toteuttaa esim. lomitusalgoritilla, jos relaatiot järjestyksessä jonkin saman attribuutin mukaan (samat arvot)

Esim. yhdiste: tulokseen vain yksi rivi, jolle

$$r(i).A = s(j).B,$$

muuten kaikki läpikäydyt

- vastaavasti hajautus: haluttujen rivien valinta ja duplikaattien poisto voidaan tehdä hash-join- algoritmin toisessa vaiheessa

- leikkaus: tulokseen vain ne, joille $r(i).A = s(j).B$

- erotus: vertailu $r(i).A < s(j).B$ tulokseen
 $r(i).A = s(j).B$ ei tulokseen
 $r(i).A > s(j).B$ ei tulokseen

SQL:n ryhmittely (group by A) vaatii

- relaation järjestämistä attribuutin A mukaan, tai
- hakemistoa A:lle (ryvästävä hakemisto paras)

(Lineaarinen läpikäynti (file scan) on tietysti myös mahdollinen, mutta vaatii kaikkien laskurien ylläpitoa koko läpikäynnin ajan, kaikkien A-arvojen tallennusta ja tuloksen muodostamista vasta läpikäynnin jälkeen.)

Yleisemmässä tapauksessa

```
select r.A, s.D, max(E) from r,s
where r.B = s.C
group by r.A, s.D
```

joudutaan järjestämään $r:n$ ja $s:n$ liitoksen tulos attribuuttiparin $(r.A, s.D)$ mukaan.

4.7 Kustannuslaskentaan perustuva optimointi

Aliluvuissa 4.4 - 4.5 on jo johdattelua kustannuslaskentaoptimointiin (erityisesti luvun 4.5 liitokselle tehdyt esimerkkilaskelmat):

- vaihtoehtoiset kyselysuunnitelmat (koko kyselylle)
- vaihtoehtojen kustannusten arviointi
- edullisimman (tai riittävän edullisen) valinta

Vaihtoehtoja on yleensä paljon:

- erilaiset kyselypuut,
- yksittäisen operaation suoritusvaihtoehdot,
- hakemistoja käytettäessä joskus monia mahdollisia hakemistoja jne).
 - esim. (kurssi, ryhmä), (kurssi, opiskelija) antavat mahdollisuuden hakea kurssin tunnuksella

→ yleensä ei voida arvioida kaikkia

Arviointi voidaan suorittaa käänkösvaiheessa

- arvioinnin kustannus jakaantuu monille kyselyn toteutuskerroille
- tarkkuus vähenee, tarvitaan parametreja

Kyselyä ajonaikaisesti tulkittaessa optimointi voisi hidastaa liaksi vasteaikaa.

Periaatteessa kustannuslaskenta voidaan (pitäisi) ulottaa kaikkiin kustannustekijöihin:

- levyhaut (perustiedosto, hakemistot)
- välitulosten kustannukset (levyoperaatiot, tila)
- laskentakustannukset (keskusmuistissa)
- keskusmuistitila (puskurien määrä)
- tietoliikennekustannukset

Yleensä rajoitetaan levyhakuihin

- yksinkertaisuus (riittävyys)
- kustannustekijöiden välinen painottaminen vaikeaa
- pieni keskusmuistitietokanta: laskentakustannukset tärkein
- hajautettu tietokanta: tietoliikennekustannukset mukaan

Tyypilliset perustiedot (tietohakemistossa):

- tiedoston (taulun) koko: tietueina, jaksaina; tietueen pituus (keskiarvo, vaihtelu), jaksotuskerroin
- tiedostorakenteen tiedot:
 - perussaantipolku (järjestys, perushakemisto)
 - oheishakemistot
 - hakemistoista ainakin tasojen lukumäärä (melko pysyvä tieto), usein myös alimman tason jaksojen määrä
- attribuuttien tiedot:
 - eri arvojen lukumäärä
 - valitsevuus (keskiarvona)
 - ehkä arvojakaumat

Yksityiskohtaiset tiedot muuttuvat usein ...

Esimerkkejä tietohakemiston tiedoista (E&N, s. 623):

a) saraketietoja

| taulu | sarake | arvojen_lkm | alin_arvo | ylin_arvo |
|----------|-----------|-------------|-----------|-----------|
| project | plocation | 200 | 1 | 200 |
| project | pnumber | 2000 | 1 | 2000 |
| project | dnum | 50 | 1 | 50 |
| employee | ssn | 10000 | 1 | 10000 |
| employee | dno | 50 | 1 | 50 |
| employee | salary | 500 | 1 | 500 |

b) taulutietoja

| taulu | rivi_lkm | sivu_lkm |
|------------|----------|----------|
| project | 2000 | 100 |
| department | 50 | 5 |
| employee | 10000 | 2000 |

c) hakemistotietoja

| hakemisto | unique? | tasoja | lehtisivuja | avain_lkm |
|-----------|---------|--------|-------------|-----------|
| proj_ploc | no | 1 | 4 | 200 |
| emp_ssn | yes | 1 | 50 | 10000 |
| emp_sal | no | 1 | 50 | 500 |

(tasot ilman lehtitasoa)

E&N (18.4.3 - 18.4.4): yksityiskohtaisia arvioita valinnan ja liitoksen eri vaihtoehdoille

- edelleen: 18.4.6: esimerkki, joka jää suurelta osin 'harjoitustehtäväksi'

Liitosoperaatioiden järjestys?

Kyselyssä on usein ainakin kolme relaatiota, jotka yhdistetään liitoksella; esim. 'Staffordin projektit ja vastaavien osastojen johtajat' (s. 8). Yleisemmässä tapauksessa (vähintään 4 relaatiota) on useita mahdollisia liitosvaihtoehtoja eikä kaikkien tarkempi evaluointi ole järkevää.

Liitokset voidaan kuvata puurakenteena; yleensä rajoitetaan toispuoleisiin puihin, esim. left-deep-puuhun, jossa vasen haara sisältää tavallaan perusrelaation (alun jälkeen edellisen tason tuloksen) ja jokaisella tasolla tulee mukaan yksi uusi relaatio. Jälkimmäinen ehto antaa mahdollisuuden käyttää jokaisessa liitoksessa 'uuden' relaation (mahdollista) saantipolkua.

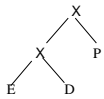
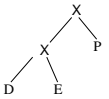
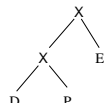
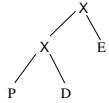
Välitulokset voidaan joko putkittaa seuraavaan operaatioon tai tallentaa levyille ('materialization').

Esimerkkitapauksessa

```
select pnumber, dnum, lname, address, bdate
from project, department, employee
where dnum = dnumber and mgrssn = ssn
and plocation = 'Stafford';
```

mahdollisia left-deep-järjestyksiä on neljä (liitos tässä yksinkertaisesti x):

1. (project x department) x employee
2. (department x project) x employee
3. (department x employee) x project
4. (employee x department) x project



Kunkin liitoksen kahdella relaatiolla voi olla useitakin hakemistoja, liitokselle 4 perusvaihtoehtoa jne. Käytännössä myös edelliset ja seuraavat operaatiot vaikuttavat.

4.8 Semanttinen kyselyoptimointi

Tietokantaan voi olla määriteltynä muitakin eheysrajoitteita kuin avain- ja viite-eheysrajoitteet.

```
Esim. create assertion salary_constraint
check (not exists(
select * from employee e, employee s
where e.salary > s.salary
and e.superssn = s.ssn));
```

“työntekijän palkka ei voi olla korkeampi kuin hänen esimiehensä palkka”

Rajoite tallennetaan kaaviotietoihin ja sen voimassaoloa valvotaan aina tietokannan tilan muuttuessa.

Jos nyt halutaan selvittää työntekijät, jotka ansaitsevat enemmän kuin esimiehensä, kyselyllä

```
select e.lname, m.lname
from employee e, employee m
where e.superssn = m.ssn
and e.salary > m.salary,
```

kyselyn vastaus selviää kaaviosta suorittamatta kyselyä ollenkaan.

- rajoitteiden ja kyselyjen yhdistäminen lienee vaikeaa
- rajoitteita voi olla paljon, jo sopivan rajoitteen löytäminen voi olla vaikeaa
- kehityksen suunta (?) - aktiivitetokannat jne

4.9 Kyselyn optimointi Oraclessa

(E&N, 18.5 & Oracle-manuaalit)

- rule-based (=heuristic) ja cost-based vaihtoehdot
- lisäksi käyttäjän vihjeet ("hints") kyselyssä
- kustannuslaskenta on tullut vallitsevaksi (v8 ...)

Heuristinen optimointi:

15 saantipolkutyyppeä on järjestetty (heuristisesti) edullisuusjärjestykseen:

1° ainoan tulosrivin tunniste ROWID tunnetaan annetaan kyselyssä, upotetun SQL-lauseen kursori

2°-4° yksi tulosrivi avaimen arvolla, rypäästä jne

5°-7° rypään hyväksikäyttöä

8° moniosaiseen avaimen perustuva haku: hakemiston avulla saadaan rivitunnisteet

9° yhtäläisyyskysely monen hakemiston avulla (rivitunnisteiden leikkaus)

10°-11° arvovälilyksely tai erisuuruuskysely monen hakemiston avulla

12° lomituliitos

13° indeksoidun sarakkeen max tai min (hakemistosta)

14° order by -kysely indeksoidulle sarakkeelle

15° taulun lineaarinen läpikäynti

Pääsääntö on, että valitaan mahdollisista suoritustavoista se, joka on luettelossa mahdollisimman alussa. Tämä ei johda aina tehokkaimpaan vaihtoehtoon (ei oteta huomioon attribuuttien arvoja jne).

Käyttäjän antamat vihjeet:

Kohde voi olla:

- käytetäänkö heuristiikkaa vai kustannuslaskentaa
- kustannuslaskennan tavoite
koko tulos: ALL_ROWS
alku nopeasti: FIRST_ROWS
- saantipolku
- liitosten järjestys, saantipolut

Käyttö: kyselyn tekijällä voi olla tietoa, jota optimoijalla ei ole.

Esim.

- tiedetään, että taulussa on hyvin vähän miehiä:

```
SELECT /*+ INDEX (patients sex_index) */
  name, height, weight
FROM patients
WHERE sex = 'M';
```

Jos miehiä olisi perusoletuksen mukaisesti noin puolet, hakemiston käyttö tuskin kannattaisi.

Vastaavasti `SELECT /*+ FULL (patients) */ ...` pakottaisi lineaariseen läpikäyntiin.

Liitosten järjestys: käyttäjällä voi olla tietoa liitososallistuvuudesta t. tuloksen koosta:

```
SELECT /*+ ORDERED */ ...
FROM A, B, C WHERE ...
```

- liitokset järjestyksessä (A |><| B) |><| C

Laskentaa kevyempi muoto on tietojen arviointi otantaan perustuen; ANALYZE-lauseessa ilmoitetaan otannan parametrit.

```
ANALYZE TABLE emp ESTIMATE STATISTICS
SAMPLE 10 PERCENT
```

Esim. taulusta tilastoitavat tiedot:

- rivien määrä
- taulun jaksojen määrä
- taulun (käyttämättömät) varatilat
- keskimääräinen rivinpituus
- ketjutettujen rivien määrä
- vastaavasti hakemistolle:
- hakemiston syvyys
- lehtitason jaksojen määrä
- erillisten hakemistoavainten lukumäärä
- lehtijaksojen keskimäärä avainta kohti
- perustiedoston jaksojen keskimäärä avainta kohti
- ryvästystieto (rivien järjestys suhteessa avainarvojen järjestykseen)
- attribuuttikohtaisesti:
- eri arvojen lukumäärä
- maksimi- ja minimiarvo
- jakaumatiedot (histogramma)
tavoitteena tasainen osiinjako ('height-balanced')

Kyselysuunnitelmasta voi saada tietoja:
`explain plan for select . . .`

Kustannuslaskentaoptimointi:

1° optimoija muodostaa mahdolliset kyselysuunnitelmat, taustana käytettävissä olevat saantipolut ja kyselyissä mahdollisesti annetut vihjeet

2° optimoija laskee kunkin suunnitelman kustannukset, taustana tietohakemistossa olevat tilastotiedot

3° optimoija valitsee tehokkaimman ('halvimman') vaihtoehdon

- oletustavoite on läpimenoajan eikä vasteajan minimointi (ALL_ROWS)

- kriteereinä siirräntäaika, cpu-aika, muistin käyttö

Tilastotiedot:

- lasketaan SQL-lauseella ANALYZE, esim.

```
ANALYZE TABLE emp COMPUTE STATISTICS
FOR COLUMNS addr, street;
```

- monenlaisia kohteesta riippuen, kohde voi olla taulu, hakemisto, attribuutti (tai näiden joukko, jopa ALL TABLES)

Muuta kyselyjen toteuttamisesta:

Tietoisuus kyselyn toteutustavasta voi ohjata käyttäjää kirjoittamaan tehokkaita kyselyjä (vaikka logiikka olisi sama).

- Vakiolauseke evaluoidaan vain käännösaikana; lauseke voi estää hakemiston käytön.

```
salary > 24000/12
salary > 2000
salary*12 > 24000 (ei evaluoida)
```

- disjunctiivinen ehto → kysely1 UNION ALL kysely2 kumpikin erikseen, ei duplikaattien poistoa (riippuu hakemistojen olemassaolosta)

- sisäkysely pyritään muuttamaan liitokseksi (joka on helpompi optimoida)

```
esim. select * from accounts
      where custno in
          (select custno from customers)
```

= (millä ehdolla ?)

```
select accounts.* from accounts, customers
where accounts.custno = customers.custno;
```