INFORMIXOV OPTIMIZATOR POIZVEDBE SQL

Lilijana Mihelič

Povzetek

Informixov optimizator poizvedb SQL (Informix Cost-Based Optimizer) je del celotnega sistema, ki skrbi za delovanje podatkovne baze. Njegova naloga je določiti optimalen način poizvedbe. Članek opisuje delovanje Informixovega optimizatorja, ki je na voljo v Informixu, v izdaji 7 in v kasnejših izdajah.

V članku so podani povzetki tehnik, s katerimi povečamo učinkovitost poizvedb. Opisane so tri strategije, ki se uporabljajo pri izvajanju stika, značilnosti optimizatorja, algoritmi, ki jih uporablja optimizator, z namenom, da izvrši poizvedbo na najprimernejši način in nekaj tehnik pri pisanju poizvedb, s katerimi pomagamo optimizatorju, da poizvedbo izvrši hitreje.

Abstract

The Informix Cost-Based optimizer is the component of the database engine that decides how to perform a query. This article describes the optimizer used in Informix engines, version number 7 and higher.

The article reviews techniques for making queries more efficient. It covers the following topics: describes the three join strategies, the features of the cost-based optimizer, algorithms used by the optimizer to find the best method to process the query and an assortment of techniques to help optimizer choose the fastest way to accomplish a query.

1. Uvod

V današnjem času smo priče vse hitrejšega tempa na področju razvoja novih računalniških tehnologij in produktov. Kljub pojavitvi objektne tehnologije, porazdeljenih objektov, ki delujejo preko Interneta in podobnih novosti, se nas še veliko ukvarja z ne-najnovejšo programsko in strojno opremo, ki jo skušamo kar se da najbolje izkoristiti. V taki situaciji sem tudi sama, ko pišem aplikacije, ki delujejo na relacijskih podatkovnih bazah. Vsaka pridobitev pri hitrosti izvajanja poizvedb po bazi nam je zelo pomembna in dragocena. V ta namen sem preučila delovanje Informixovega optimizatorja poizvedb SQL.

Informixov optimizator nam pomaga pri pisanju čimbolj optimalnih poizvedb. Poizvedbe naj bi bile optimalne glede na ceno izvajanja. V nadaljevanju so opisani postopki in tehnike za pohitritev poizvedb in povečanje učinkovitosti, ob predpostavki, da ne moremo spreminjati strukturo obstoječe podatkovne baze.

Kadar izvajamo ukaz SQL nad tabelo, na katero je postavljeno več indeksov, sistem Informix sam odloči, s katerim indeksom dostopa do podatkov. Če za neko poizvedbo ni primernega indeksa, uporablja zaporedni dostop. V sistemu Informix moramo zapisati posebne ukaze, če želimo videti, kako se je dejansko izvedla neka poizvedba. Lahko se nam zgodi, da za neko poizvedbo mislimo, da se v njej dostopa do podatkov z indeksom, v resnici pa se podatki pregledujejo zaporedno. Take anomalije nam skuša odpravljati optimizator.

2. Strategije izvajanja operacije stik

Stik (join) dveh tabel vsebuje informacije iz obeh tabel, glede na relacijo, ki povezuje en ali več stolpcev iz vsake tabele.

Rezultat te operacije so lahko vsi, ali pa samo nekateri stolpci tabel.

Stik se lahko izvaja tudi nad večimi tabelami. V takem primeru rezultat operacije vsebuje informacije iz več tabel, relacije pa povezujejo en ali več stolpcev posameznih tabel.

Če v poizvedbi zamenjamo vrstni red naštetih tabel ali relacije povezav in pogojev, lahko dobimo isti rezultat, hitrosti izvajanja pa so lahko zelo različne.

Če želimo poizvedbo napisati tako, da se bo izvajala optimalno, lahko uporabimo Informixov optimizator, ki pregleda vse možnosti in nam vrne optimalno napisan ukaz.

Primer:

Če Informixov optimizator pri izvajanju stika dveh tabel ugotovi, da je bolje zamenjati vrstni red tabel, se

stik izvede tako, da se najprej indeksno ali zaporedno pregleda druga tabela in se ji, glede na filter, poišče pripadajoč zapis prve tabele.

Sistem Informix pri izvajanju stika uporablja tri strategije:

- STIK Z VGNEZDENO ZANKO
- STIK Z UREJANJEM IN ZLIVANJEM
- STIK Z UPORABO RAZPRŠENE TABELE

2.1. Stik z vgnezdeno zanko

Stik z vgnezdeno zanko deluje tako, da se po nekem vrstnem redu pregleda prva tabela in se, glede na filter, poišče pripadajoč zapis iz druge tabele. Najprej se vzame prvi zapis iz prve tabele. Polja tega zapisa, ki se uporabljajo pri stiku, se imenujejo ključ zapisa. Potem se pregledujejo zapisi iz druge tabele in se primerjajo z vrednostjo ključa zapisa. Če je ključ zapisa druge tabele enoznačen, se pregledovanje konča, ko se najde prva ustrezna vrednost, se pravi, prvi zapis druge tabele, ki ima enako vrednost ključa, kot zapis prve tabele.

Če v poizvedbi nastopa še tretja tabela, se za vsak ustrezen zapis iz druge tabele išče pripadajoč zapis iz tretje tabele. Proces se nadaljuje, dokler ne pregledamo vseh navedenih tabel.

Vrstni red pregledovanja katerekoli tabele, ki nastopa v poizvedbi, je določen z indeksom, ki se uporablja. Če ustreznega indeksa ni, se iskanje izvaja zaporedno.

V splošnem velja, da se za vsak zapis iz prve tabele prebere vsaj en zapis iz druge tabele in iz vseh naslednjih tabel, razen, kadar ustreznega zapisa ni. Ker ni vsak prebrani zapis pravi, se v večini primerov prebere več zapisov, kot je potrebno.

2.2. Stik z urejanjem in zlivanjem

Tabele, ki nastopajo v operaciji stika, se uredi glede na vrednost filtra. Zapise različnih tabel se zliva, kadar so vrednosti stolpcev, glede na katere se dela stik, enake.

Primer:

Imamo dve tabeli z imeni B in C, na katerih niso postavljeni indeksi. Izvaja se naslednji ukaz SQL:

```
SELECT * FROM B,C
  WHERE C.x = B.w AND
    C.z = 17
```

Stik z urejanjem in zlivanjem naredi naslednje:

- izberejo se vsi zapisi iz tabele C, ki zadoščajo filtru, to je, za katere velja C.z = 17,
- ti zapisi se uredijo glede na vrednosti C.x,
- zapisi tabele B se uredijo glede na vrednosti B.w,
- urejeni zapisi tabele B in urejeni zapisi tabele C se zlijejo skupaj, kadar velja, da je C.x =B.w.

2.3. Stik z uporabo razpršene tabele

Uporaba stika z razpršeno tabelo je primerna, kadar je potrebno eno tabelo pregledati zaporedno ali kadar imajo tabele veliko število zapisov. Glede na rezultat zgostitvene funkcije pregledanih zapisov te tabele se zgradi razpršena tabela. Zapise iz drugih tabel se skuša povezati z vrednostmi v razpršeni tabeli.

Če ni potrebe, da bi se ena od tabel, ki nastopajo v operaciji stika, pregledala zaporedno, se razpršena tabela ustvari za tabelo, ki ima najmanj zapisov.

Primer:

Izvaja se poizvedba (prikazana na naslednji sliki) nad tabelami z imeni tabB in tabC. Za tabelo z imenom tabB, se ustvari razpršena tabela.

Za vsak prebrani zapis tabele tabB zgostimo njegovo vsebino z izračunom zgostitvene funkcije. Glede na vrednost zgostitvene funkcije se zapisi uvrstijo v določeno skupino.

Označba h(zapis) predstavlja vrednost zgostitvene funkcije za nek zapis. Ta vrednost se vpiše v razpršeno tabelo, v polje z imenom glava skupine. V neki skupini zapisov se nahajajo vsi zapisi tabele, ki imajo enako vrednost zgostitvene funkcije. Zapisi znotraj skupine niso urejeni.

Po izgradnji razpršene tabele se berejo zapisi tabele z imenom tabC. Tudi zanje se izračuna vrednost zgostitvene funkcije, da se ve, v kateri skupini razpršene tabele je možno najti ustrezen zapis.

Razpršena tabela se ustvari v skupnem pomnilniku sistema Informix, katerega velikost lahko določimo sami, ob prvotni postavitvi sistema. V kolikor tega prostora ni dovolj, se del razpršene tabele zapiše na disk, kar upočasni vse operacije, ki delajo z njo.

Velikost razpršene tabele za neko tabelo lahko izračunamo po obrazcu:

velikost razpršene tabele v zlogih =
(32 zlogov + dolžina zapisa tabele) * število zapisov v tabeli

2.4. Slikovni prikaz delovanja različnih načinov izvedbe operacije stik

Glej sliki 1A in 1B.

3. Vpliv delovanja optimizatorja

Delovanje Informixovega optimizatorja lahko vključimo v izvajanje ukazov SQL v celoti, samo delno, ali pa ga sploh ne vključimo. Vpliv delovanja optimizatorja na izvajanje ukazov SQL določimo s konfiguracijskim parametrom, ki se imenuje OPTCOMPIND.

Vrednost OPTCOMPIND pove, ali naj optimizator medsebojno primerja opisane strategije stika za posamezni dve tabeli.

OPTCOMPIND = 0,
pomeni, da naj optimizator poišče ustrezen indeks, s pomočjo katerega dostopa do podatkov, ko izvaja operacijo stik. Če indeksa ni, se izvede stik z uporabo razpršene tabele.

OPTCOMPIND = 1,
pomeni, da optimizator deluje na dva načina:

- če se izvaja transakcija nad podatkovno bazo, ki je transakcijsko zaščitena in se zapisi med izvajanjem operacije zaklenejo za ostale uporabnike, se optimizator obnaša kot v primeru, ko je OPTCOMPIND postavljen na vrednost 0.

sicer:

- optimizator izračuna najcenejšo pot dostopa do podatkov, glede na uporabo opisanih treh strategij stika.

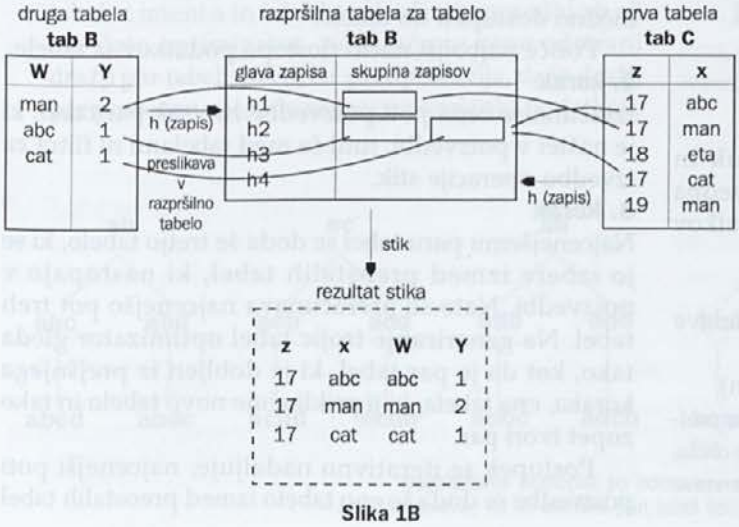
OPTCOMPIND = 2,
optimizator v vsakem primeru izračuna najcenejšo pot dostopa do podatkov, glede na uporabo opisanih treh strategij stika.

Primer:
Imamo dve tabeli z imeni A in B ter indeks na tabelo B, na polje z imenom x.

Izvajamo ukaz SQL:
SELECT * FROM A,B WHERE A.x = B.x;

Optimizator ugotovi, da lahko izvede stik na dva načina. Prouči vrstni red tabel A, B in vrstni red tabel B, A. Glede na vrednost parametra OPTCOMPIND naredi sledeče:

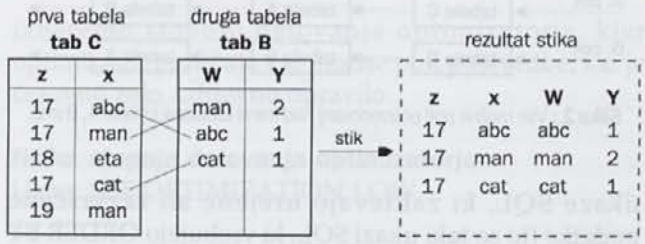
- če velja OPTCOMPIND = 0: Izvede stik z vgnezdeno zanko, z uporabo indeksa B.x v primeru, ko je vrstni red tabel A,B. Ko je vrstni red tabel B,A, uporabi stik z razpršeno tabelo.



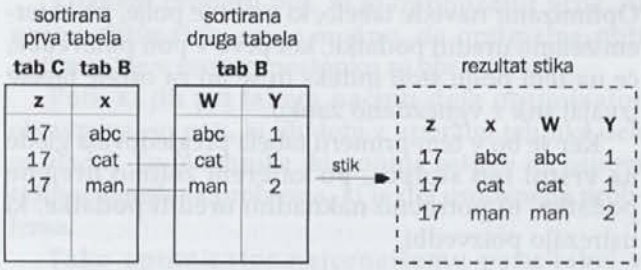
Slika 1B

Izvajanje ukaza
SELECT * FROM tabB, tabC
WHERE tabC.x = tabB.w AND
tabC.z = 17

STIK Z VGNEZDENO ZANKO



STIK S SORTIRANJEM IN ZLIVANJEM



Slika 1A

- če velja OPTCOMPIND = 2: Optimizator pri proučevanju vrstnega reda tabel B,A preučuje med možnostmi uporabe:

- dinamično ustvarjenega indeksa na tabeli A,
- stika z urejanjem in zlivanjem
- stika z uporabo razpršene tabele.

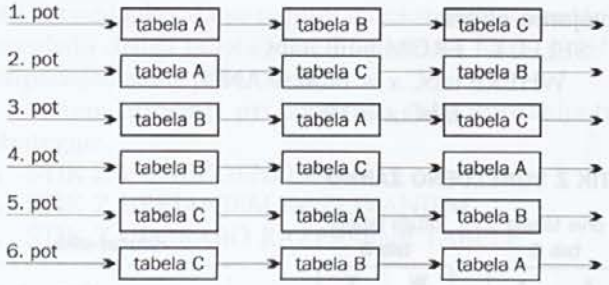
3.1. Pot poizvedbe

Pot poizvedbe je neko zaporedje vseh tabel, ki nastopajo v ukazu SQL. Optimalna pot poizvedbe je neka permutacija vseh tabel, ki so naštete v ukazu SQL.

Optimizator izračuna ceno izvedbe ukaza SQL, glede na različno pot poizvedbe.

Primer:
Če v ukazu SQL nastopajo tri tabele, z imeni A,B in C, optimizator pregleda, kako bi se izvedel ukaz SQL, če bi si tabele sledile v naslednjem redu:
Pot poizvedbe vpliva na količino prebranih zapisov. Zato je na začetku poti poizvedbe priporočljivo imeti krajše tabele, ali pa tabele, na katere so postavljeni zelo omejujoči filtri (naprimer, post_stev = 1001). Na tak način se porabi manj časa za branje zapisov iz ostalih tabel, ki ne ustrezajo kriterijem poizvedovanja.

Z optimalno potjo poizvedbe se lahko izognemo postopku urejanja podatkov za



Slika 2.: Vse možne poti poizvedovanj nad tremi tabelami z imeni A, B in C.

ukaze SQL, ki zahtevajo urejene ali razvrščene podatke (to so taki ukazi SQL, ki vsebujejo ORDER BY ali GROUP BY).

Primer:

Optimizator navede tabelo, ki vsebuje polje, po katerem želimo urediti podatke, kot prvo v poti poizvedbe, če na tem polju stoji indeks in se bo za ostale tabele izvajal stik z vgnezdeno zanko.

Ker se bo v tem primeru tabela pregledovala glede na vrstni red stolpca, po katerem želimo urejene podatke, ni potrebno naknadno urediti podatke, ki ustrezajo poizvedbi.

3.2. Cena poizvedbe in optimalna pot

cena poizvedbe za posamezno pot poizvedbe =
število dostopov do diska +
(W število procesiranih zapisov, ki ustrezajo poizvedbi)*

ali zapisano drugače:

cena poizvedbe za posamezno pot poizvedbe =
cena vhodno/izhodnih operacij +
(W* cena delovanja centralne procesne enote)

W = faktor uteži, ki je trenutno 0.3; predstavlja razmerje cene delovanja centralne procesne enote proti ceni vhodno/izhodnih operacij.

Faktor uteži je določen s strojno opremo in njegove vrednosti ne moremo spreminjati.

Optimalna pot poizvedbe = najcenejša pot poizvedbe.

3.3. Cena urejanja podatkov

Postopek urejanja obremeni hitri pomnilnik in potrebuje tudi trdi disk. V kolikor neka poizvedba sproži urejanje podatkov, se cena urejanja podatkov prišteje v ceno poizvedbe.

cena urejanja =
cena obdelave v hitrem pomnilniku + cena obdelave na trdem disku

cena obdelave v hitrem pomnilniku = c w n log2(n)

c = število stolpcev zapisa iz tabele, ki jih je potrebno urediti, predstavlja ceno branja podatkov iz tabele in prepis podatkov v obliko, kjer so podatki urejeni glede na vrednosti ključa, po katerem se ureja

w = dolžina ključa, po katerem se ureja, od te dolžine je odvisno, koliko stane primerjava in prepis podatkov v urejeno obliko. Numerična vrednost tega podatka je odvisna od strojne opreme.

n log2(n) = predstavlja potrebno število primerjav za urejanje tabele z n zapisi.

cena dela, ki ga opravi trdi disk = 2 n m

m = število nivojev zlivanja, ki jih uporabi urejevalni algoritem. Ta faktor zavisi od števila ključev, po katerih se ureja podatke.

Kadar se vse vrednosti ključev, po katerih se ureja podatke, lahko nahajajo v hitrem pomnilniku, je vrednost **m** enaka ena in tako je cena dela, ki ga opravi trdi disk enaka 2 n.

Cena urejanja podatkov je odvisna od števila podatkov, ki jih urejamo.

Če ne moremo zmanjšati cene urejanja s tem, da urejamo manj podatkov, ceno lahko poskušamo zmanjšati tako, da urejamo po čim krajšem in enostavnejšem ključu, kar zmanjša faktorja c in w.

4. Optimizacija poizvedb

Proces optimizacije poizvedb je sestavljen iz treh korakov.

1. korak

To je postopek inicializacije, ki ga opravi optimizator, preden prične s postopkom izračunavanja cene poti.

Optimizator pregleda strukturo tabel, indeksov in filtra pri poizvedbi, proučuje nabor vrednosti zapisov, glede na filter in poskuša ugotoviti, kolikšna je propustnost filtra.

Propustnost filtra je število, ki ima lahko vrednost med 0 in 1. Enako je razmerju med številom zapisov v tabeli in delom zapisov, ki ustrezajo filtru.

Optimizator potem ugotavlja, če obstoječi indeksi na tabelah omogočajo dostop do podatkov na način, ki ga pričakuje postavljeni filter in če se obstoječi indeksi na tabelah lahko uporabijo v operacijah urejanja in razvrščanja podatkov.

Optimizator tudi proučuje vse načine, kako je možno dostopati do tabele.

Poišče najboljši način dostopa podatkov iz tabele.

2. korak

Izračuna se cena poti poizvedbe za vsak par tabel, ki je naštet v poizvedbi, tudi če med tabelami ni filtra za izvedbo operacije stik.

3. korak

Najcenejšemu paru tabel se doda še tretjo tabelo, ki se jo izbere izmed preostalih tabel, ki nastopajo v poizvedbi. Nato se preračunava najcenejšo pot treh tabel. Na generiranje trojic tabel optimizator gleda tako, kot da je par tabel, ki je dobljen iz prejšnjega koraka, ena tabela, ki ji priključuje novo tabelo in tako zopet tvori par.

Postopek se iterativno nadaljuje; najcenejši poti poizvedbe se doda še eno tabelo izmed preostalih tabel

in se zopet preračuna najcenejšo pot. Postopek se konča, ko so v pot poizvedb zajete vse tabele, ki nastopajo v poizvedbi.

Na koncu optimizator pogleda, če ukaz SQL vsebuje določilo ORDER BY ali GROUP BY, kar pomeni, da mora ukaz SQL vrniti urejene podatke. Določene poti so takšne, da se izognejo postopku urejanja. Za take poti, kjer je potrebno še naknadno urediti podatke, optimizator prišteje še ceno urejanja podatkov.

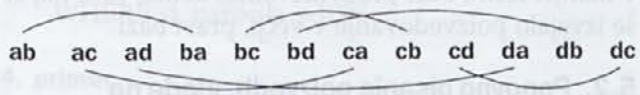
Optimizator po vsem tem izbere pot, ki ima najmanjšo ceno. Tako izbrana pot je optimalna pot poizvedbe.

4.1. Postopek izbire optimalne poti poizvedbe

Primer gradnje poti za neko poizvedbo:
Imamo štiri tabele z imeni a, b, c in d. Vsaka tabela ima tri stolpce, z imeni a, b in c. Izvajamo ukaz SQL:

```
SELECT *
 FROM a,b,c,d
  WHERE a.a = b.a AND
   b.b = c.b AND
   c.c = d.c
```

Možni pari tabel za opisan primer



s črto so povezani pari tabel, kjer je en par tabel odvečen

Za vsak posamezen par tabel bo optimizator:

- poiskal najprimernejši način za izvedbo stika dveh tabel
- izbral najprimernejši način dostopa do podatkov za vsako tabelo
- izračunal ceno operacije stik
- iz postopka bo odstranil odvečen pare. Naprimer, tabeli z imeni a in b lahko združi v zaporedju ab ali ba. Zato optimizator po izračunu cene odstrani dražji par tabel, razen, če predpostavlja, da je dražji par potreben zato, da se izogne urejanju podatkov.

4.2. Stopnje delovanja optimizatorja

Na nivoju izvajanja aplikacij lahko določimo, na kakšen način naj optimizator deluje.

Visoka stopnja delovanja optimizatorja
Z ukazom: SET OPTIMIZATION HIGH

izberemo stopnjo delovanja optimizatorja, kjer optimizator proučuje vse možne poti poizvedbe, kar je časovno zelo zahtevno opravilo.

Nizka stopnja delovanja optimizatorja
Ukaz: SET OPTIMIZATION LOW

pove optimizatorju, naj proučuje le določene poti poizvedb in izbere eno izmed njih.

V takem primeru pot, ki jo optimizator vrne, ni nujno optimalna, saj je možno, da optimalne poti optimizator v svojem postopku ni obravnaval.

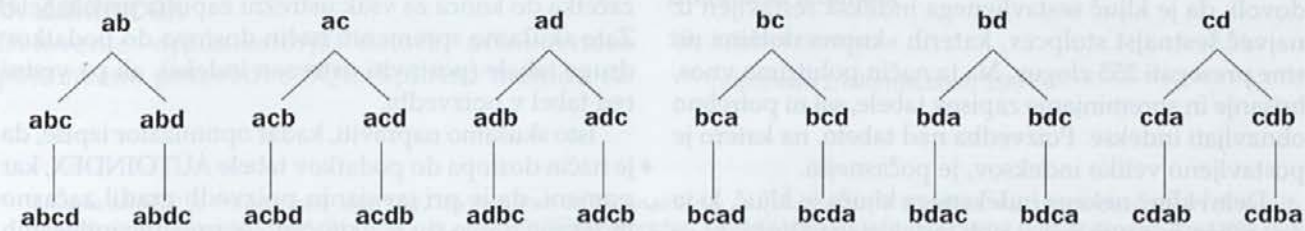
Poti, ki jih pri takem načinu dela optimizator obravnava, so poti, ki jih dobi z uporabo tehnike deli in vladaj. To je tehnika, ki ponuja rešitev na osnovi rešitve manjšega problema, ki je del osnovnega problema.

Tako optimizator najcenejšemu paru tabel v postopku tvorjenja poti poizvedb doda preostale tabelo, nato dodaja tabele najcenejši trojici. Na vsakem koraku postopek nadaljuje le za najcenejše zaporedje tabel, ki je hkrati že del končne dobljene poti.

Kadar izberemo nizko stopnjo delovanja optimizatorja, je čas optimizacije poizvedbe (to je čas, ko optimizator proučuje poizvedbo) krajši, vendar se lahko poizvedba izvaja dalj časa, ker optimizator ni našel optimalne poti.

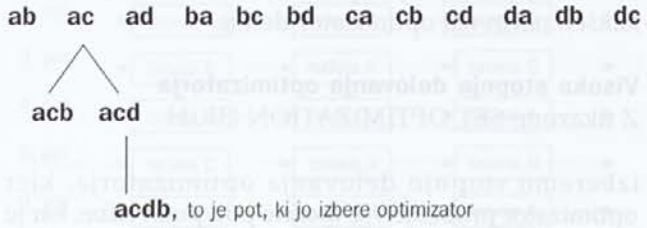
Primer gradnje poti poizvedbe pri izbiri nizke stopnje delovanja optimizatorja:

```
SELECT *
 FROM a,b,c,d
  WHERE a.a = b.a AND
   b.b = c.b AND
   c.c = d.c
```



Nadaljevanje postopka po odstranitvi odvečnih parov tabel
(predpostavimo, da so ustrezni pari tabel tisti, ki so tudi po abecedi prvi)

Možni pari tabel in nadaljevanje postopka ob predpostavki, da je najcenejši par tabel ac in najcenejša trojica tabel acd



Izbira nizke stopnje delovanja optimizatorja je primerna, kadar so za nas pomembni odzivni časi poizvedb, ali, kadar je v poizvedovanju naštetih več kot pet tabel.

Nizka stopnja delovanja optimizatorja je najverjetneje primernejša od visoke stopnje delovanja optimizatorja, kadar imajo poizvedbe obliko:

- ena tabela je povezana z večimi drugimi tabelami (taki primeri so podatkovne blagovnice) ali
- v poizvedovanju je naštetih več tabel in vsaka našteta tabela ima manj zapisov kot predhodno navedena, indeksi tabel pa so postavljeni na stolpce, preko katerih se povezujejo tabele.

V takih primerih je najverjetneje optimalna pot poizvedbe prav pot, ki jo optimizator najde tudi pri nizki stopnji delovanja.

V splošnem pa ni pravila, kdaj s katero izbiro pridemo do boljših rezultatov.

5. Pohitritev poizvedb

V splošnem velja, da je možno poizvedbe pohitriti:

- če nam uspe zmanjšati število zapisov, po katerih poizvedujemo,
- če se je možno izogniti postopku urejanja podatkov, ali če je možno izvesti urejanje po enostavnejšem ključu,
- če je možno dostopati do podatkov preko indeksa namesto zaporedno, ali, če so podatki primerno porazdeljeni po večih fragmentih in
- če se z uporabo sestavljenega indeksa ali iskanja preko delnega ključa lahko izognemo postavitvi večih indeksov na tabeli.

Sestavljen indeks je indeks, katerega ključ je sestavljen iz dveh ali več polj zapisa v tabeli. Sistem Informix dovoli, da je ključ sestavljenega indeksa sestavljen iz največ šestnajst stolpcev, katerih skupna dolžina ne sme presegati 255 zlogov. Na ta način pohitrimo vnos, brisanje in spreminjanje zapisov tabele, saj ni potrebno obnavljati indekse. Poizvedba nad tabelo, na katero je postavljeno veliko indeksov, je počasnejša.

Delni ključ nekega indeksnega ključa je ključ, ki je sestavljen iz prvih nekaj polj indeksnega ključa, ki so navedena v enakem vrstnem redu kot v indeksu. Preko indeksa se dostopa do podatkov tudi, kadar poizvedujemo glede na vrednost njegovega delnega ključa. Nepotrebno je postavljati indeks na polja, ki so delni ključ nekega že obstoječega indeksa.

5.1. Postavitev testnega okolja, v katerem proučujemo poizvedbe

Izberemo poizvedbo, ki se nam zdi prepočasna in jo skušamo izvajati v takem okolju, kjer ponavljanje poizvedbe traja vedno enako časa. Edino tako bomo lahko proučili, ali bodo spremembe, ki jih bomo naredili kaj prispevale k hitrosti izvajanja poizvedbe.

Okolje, ki ni primerno za proučevanje poizvedb, je naprimer okolje, kjer so odzivni časi odvisni od delovanja virov, katerih obremenitev se kar naprej spreminja. Prav tako ni primerno proučevati poizvedbe znotraj nekega kompleksnega programa. V takem primeru skušamo poizvedbo izluščiti in jo izvajati interaktivno ali pa v okviru enostavnejšega programa.

Kadar imamo opravka s kompleksnimi poizvedbami, ki trajajo več ur, si za testiranje pripravimo bazo, v kateri so krajše tabele in na taki bazi izvajamo poizvedbe. Optimizator se v primerih majhnega števila zapisov lahko odloči za drugačno izvajanje, kot v primeru, kadar je teh zapisov veliko. V splošnem pa velja, da je čas izvajanja poizvedbe linearno odvisno od števila zapisov.

Tako lahko na osnovi časa izvajanja poizvedovanj v manjši testni bazi predvidevamo, koliko časa naj bi se izvajalo poizvedovanje v večji, pravi bazi.

5.2. Ponovno pisanje poizvedb, glede na izvajanje, ki ga izbere optimizator

Ko pregledamo, kako se je neka poizvedba izvedla, lahko ugotovimo, kako bi čas izvajanja poizvedbe pohitrili.

Nekaj primerov:

1. primer

Optimizator izpiše, da je pisal vmesne datoteke, s pomočjo katerih je kasneje izvajal urejanje podatkov. Število dostopov do diska je zato vsaj dvakrat večje, kot če se temu uspemo izogniti.

2. primer

Optimizator izpiše, da je do določene tabele, ki v poizvedbi nastopa kot druga, dostopal zaporedno, kar pomeni, da je to tabelo zaporedno pregledoval od začetka do konca za vsak ustrezni zapis iz prve tabele. Zato skušamo spremeniti način dostopa do podatkov druge tabele (postaviti ustrezen indeks), ali pa vrstni red tabel v poizvedbi.

Isto skušamo napraviti, kadar optimizator izpiše, da je način dostopa do podatkov tabele AUTOINDEX, kar pomeni, da je pri izvajanju poizvedb gradil začasno indeksno drevo (tu je vključeno še urejanje indeksnih ključev), zato, da bi se izognil ponavljajočemu se zaporednemu dostopu do zapisov te tabele.

3. primer

Imamo tabelo in indekse:

```
table narocila
(
 st_narocila serial not null ,
 st_stranke integer ,
 ...
);
unique index  o_num_ix  on narocila (st_narocila);
index         o_s_num_ix on narocila (st_stranke);
```

Poizvedba, ki povzroči zaporedni dostop do podatkov, kljub postavitvi indeksov:

```
SELECT * FROM narocila
 WHERE (st_stranke > 1000 AND st_narocila < 1005)
 OR st_narocila =1008
```

To poizvedbo lahko pohitrimo na tak način, da jo pretvorimo v ukaz z uporabo združitve, kar pomeni, da se izvede poizvedba za posamezen del izraza, potem pa se dobljeni podatki združijo. Pri poizvedbi nad delom izraza se uporabljajo indeksi.

```
SELECT * FROM narocila
 WHERE (st_stranke > 1000 AND st_narocila < 1005)
UNION
SELECT * FROM narocila
 WHERE st_narocila =1008
```

4. primer

Poizvedba:

```
SELECT * FROM stranka
 WHERE koda[4,5] > "50"
```

pomeni, da želimo izpis tistih zapisov, kjer sta četrti in peti znak polja koda večja od 50. V takih primerih se ne dostopa do podatkov preko indeksa, tudi če indeks obstaja. Zato skušamo izraz zapisati z relacijskimi operatorji ali z uporabo izraza BETWEEN, na celotnem polju, kar omogoča indeksni dostop do podatkov.

Če tudi tako poizvedovanje ni hitrejše, je boljše poizvedovati na način:

```
SELECT * FROM stranka
  IF koda[,5] > "50" THEN ...
```

6. Zaključek

Delovanje optimizatorja določa učinkovitost posamezne poizvedbe SQL. Splošna metoda za pohitritev poizvedb je torej poznavanje delovanja optimizatorja, ki se z uporabo ukaza SET EXPLAIN ON zapiše v datoteko. Nadalje je potrebno vedeti, katere operacije v izvajanju poizvedb so časovno potratne in kako se jim izognemo ali pa vsaj zmanjšamo njihov obseg.

Kadar nismo zadovoljni s hitrostjo izvajanja poizvedb neke aplikacije, uporaba optimizatorja ni prva stvar, na katero je potrebno pomisliti. Gledano na sistem s širšega vidika, je najprej potrebno skrbno proučiti nastali problem, razumeti delovanje aplikacije, ovrednotiti delovanje aplikacije z merili, poiskati ozko grlo delovanja, proučiti, koliko programov izvajamo, koliko računalnikov imamo na razpolago, koliko uporabnikov izvaja poizvedbe, itd. Vse to ima namreč lahko večji vpliv na učinkovitost izvajanja poizvedb, kot pa je prispevek optimizatorja. Kadar pa na take stvari nimamo vpliva, ali jih ne moremo spreminjati, nam optimizator lahko bistveno pomaga, zato je pomembno poznati njegovo delovanje.

Literatura

1. Advanced Programming in the UNIX Environment
   W. Richard Stevens, Addison - Wesley Publishing Company 1992
2. INFORMIX - 4GL User Guide Database Tool
   Informix Software, Inc., 1996
3. INFORMIX - Guide to SQL Tutorial
   Informix Software, Inc., 1996
4. INFORMIX - NET and INFORMIX - STAR Configuration Guide
   Informix Software, Inc., 1996
5. INFORMIX - OnLine Dynamic Server Databases
   Informix Software, Inc., 1996
6. INFORMIX - OnLine Dynamic Server System Administration
   Informix Software, Inc., 1996
7. IRIS InSight Library
   Silicon Graphics OnLine Help, 1997
8. IRIS User's Man Pages
   Silicon Graphics, Inc., 1994
9. Writing Solid Code
   Stephen A. MAguire, Microsoft Press, 1993

Literatura z Interneta

10. Informix products
    http://www1.informix.com, 1997

◆

Lilijana Mihelič je magistrirala leta 1997 na Fakulteti za računalništvo in informatiko v Ljubljani. Na svojih dosedanjih delovnih mestih se je ukvarjala s tehnično podporo računalniškega omrežja, z administracijo Informix-ovega podatkovnega strežnika in z razvojem informacijskih sistemov. Trenutno je zaposlena v podjetju Telekom Slovenije, v Sektorju za informatiko, kjer sodeluje pri razvoju informacijskega sistema krajevnega omrežja, podprtega z grafiko.

◆