IMPLEMENTACIJA POIZVEDOVANJ V MREŽNIH DATOTEKAH

INFORMATICA 3/89

Keywords: grid file, file query, implementation

Viljan Mahnič

POVZETEK: V članku je opisana implementacija sistema za vzdrževanje mrežnih datotek, ki omogoča učinkovito izvajanje različnih poizvedovanj, značilnih za relacijske podatkovne baze. Uvodni razlagi osnovnih značilnosti mrežnih datotek sledi opis algoritmov za omejevanje iskalnega območja v podatkovnem prostoru in ovrednotenje selekcije nad izbrano relacijo. V relacijskih podatkovnih bazah se najpogosteje pojavljajo poizvedovanja, sestavljena iz selekcije, projekcije in stika. V članku sta opisani dve strategiji, ki omogočata realizacijo tovrstnih poizvedovanj. Na koncu sledi še opis programskega paketa, v okviru katerega je bila izvedena implementacija vseh navedenih algoritmov.

IMPLEMENTATION OF QUERIES IN GRID FILES: We describe an implementation of the grid file system which enables efficient evaluation of various relational queries. At first, the basic concepts of grid files are explained, and then the algorithms whose purpose is to restrict the search region in the data space and to compute the selection operation are described. General queries involving selection, projection, and join operations are often used to retrieve information from a relational database. Two strategies are proposed to evaluate such a query. Finally, a description of the software package in which all these algorithms were implemented is given.

1. UVOD

Mrežne datoteke (angl. grid files) predstavljajo podatkovno strukturo, ki omogoča učinkovito iskanje podatkov po večjem številu atributov. Za razliko od standardnih načinov fizične organizacije podatkov (n.pr. indeksno-sekvenčne datoteke, razpršene tabele) se v okviru mrežnih datotek vsi atributi, po katerih vršimo iskanje, obravnavajo enakovredno (simetrično), kar pomeni, da primarni ključ ni privilegiran v primerjavi z ostalimi atributi. Poleg tega pa se mrežne datoteke s sprotnim spreminjanjem svoje strukture (razcep oziroma zlivanje podatkovnih blokov in s tem povezano ažuriranje mrežnega direktorija) dinamično odzivajo na vstavljanje novih oziroma brisanje že obstoječih zapisov.

Osnovni koncepti mrežnih datotek so bili prvič opisani v članku [NHS84], ki pa je pustil odprta še nekatera vprašanja v zvezi z implementacijo skal in mrežnega direktorija kot tudi glede izbora algoritmov za razcep in zlivanje. Odgovore na ta vprašanja je poiskal Hinrichs v [Hin85a, Hin85b], ki je implementiral sistem za upravljanje mrežnih datotek in opisal možnosti njihove uporabe za hranjenje geometričnih podatkov. Schikuta [Sch86] pa je predlagal, da bi mrežne datoteke uporabili kot osnovo za implementacijo relacijskih podatkovnih baz.

Z namenom, da bi proučili možnosti uporabe mrežnih datotek na področju relacijskih podatkovnih baz, smo v Laboratoriju za programsko opremo in informatiko, ki deluje v okviru Fakultete za elektrotehniko in računalništvo, implementirali sistem za upravljanje mrežnih datotek in algoritme za izvajanje poizvedovanj na osnovi operacij relacijske algebre.

Pri implementaciji postopkov za vzdrževanje mrežnih datotek smo se v veliki meri oprli na ugotovitve iz [Hin85a, Hin85b], s to razliko, da smo namesto enostavne sheme vhodno-izhodnih vmesnikov vpeljali fleksibilnejšo, ki omogoča boljše izkoriščanje razpoložljivega prostora v glavnem pomnilniku, s tem pa zmanjšuje število dosegov na disk, potrebnih za realizacijo poizvedovanj.

Na novo pa smo implementirali postopke za omejevanje iskalnega območja in realizacijo poizvedovanj na osnovi operacij relacijske algebre.

Za boljše razumevanje članka podajamo v nadaljevanju najprej kratek opis temeljnih značilnosti mrežnih datotek. Nato sledi opis strategije za omejevanje iskalnega območja, opis algoritmov za izvajanje poizvedovanj v mrežnih datotekah in nazadnje še opis strukture programskega paketa.

2. OSNOVNE ZNAČILNOSTI MREŽNIH DATOTEK

Datoteko d smatramo kot zaporedje zapisov $z=(a_1,a_2,...,a_k)$, kjer so $a_1,a_2,...,a_k$ vrednosti posameznih atributov. Vsaka vrednost a_i pripada neki linearno urejeni domeni A_i.

Vsak zapis lahko ponazorimo kot točko v k-dimenzionalnem podatkovnem prostoru, tako da vrednosti atributov uporabimo kot koordinate v posameznih dimenzijah. Z uvedbo primerne pravokotne mreže lahko podatkovni prostor razdelimo na celice, ki imajo obliko hiperparalelepipeda, in poskrbimo, da so tiste točke (zapisi), ki se nahajajo znotraj iste celice, shranjene na disku v okviru istega bloka podatkov.

Za predstavitev mreže potrebujemo k enodimenzionalnih polj, ki jih imenujemo skale, in eno k-dimenzionalno polje, imenovano mrežni direktorij. Vsaka skala vsebuje več mej, pri čemer vsaka meja predstavlja (k-1)-dimenzionalno hiperravnino, ki deli podatkovni prostor na 2 dela. Mrežni direktorij skrbi za povezavo med mrežo v podatkovnem prostoru in dejansko organizacijo podatkov na disku. Vsaki celici v mreži ustreza en element mrežnega direktorija, ki vsebuje naslov tistega podatkovnega bloka, v katerem so shranjene vse točke (zapisi), ki se nahajajo v tej celici.

Da bi se izognili nizki zasedenosti podatkovnih blokov, lahko isti podatkovni blok hrani točke iz več različnih celic. Celice, ki jim pripada na disku isti podatkovni blok, tvorijo t.i. območje podatkovnega bloka, ki mora (zaradi algoritmov za razcep in zlivanje) imeti obliko hiperparalelepipeda.

Pri vstavljanju novega zapisa najprej s pomočjo skal določimo tisto celico v mreži, kamor se zapis preslika kot točka v podatkovnem prostoru, nato pa s pomočjo mrežnega direktorija dobimo naslov podatkovnega bloka, v katerega je treba zapis dodati. Če v podatkovnem bloku ni več prostora, se izvrši razcep podatkovnega bloka, ki lahko poteka na dva načina, odvisno od tega, kakšno je območje podatkovnega bloka.

Če območje podatkovnega bloka obsega eno samo celico, moramo v eno izmed skal dodati novo mejo, ki razdeli območje podatkovnega bloka na dva dela, in skladno s to mejo porazdeliti zapise med dva podatkovna bloka. Nova meja povzroči, da se mrežni direktorij poveča (vriniti je treba (k-1)-dimenzionalno rezino), kar zahteva prestrukturiranje mrežnega direktorija in ustrezno ažuriranje naslovov podatkovnih blokov, ki pripadajo posameznim celicam.

Če območje podatkovnega bloka obsega več celic, odpade potreba po dodajanju nove meje, saj se za razcep lahko uporabi ena izmed mej, ki že sekajo območje podatkovnega bloka. V tem primeru se mrežni direktorij ne poveča; poskrbeti moramo le za ažuriranje naslovov znotraj prvotnega območja podatkovnega bloka.

Struktura mrežnega direktorija in vsebina mrežne datoteke se dinamično prilagajata tudi pri brisanju zapisov. V primeru, ko zasedenost nekega podatkovnega bloka pade pod predpisano mejo, pride do zlitja s sosednim podatkovnim blokom, kar ima zopet za posledico ažuriranje naslovov v mrežnem direktoriju. Če meja, ki je ločevala območji obeh podatkovnih blokov, ni več potrebna, jo izločimo, pri tem pa se mrežni direktorij zmanjša (izločiti je treba (k-1)-dimenzionalno rezino).

Pri implementaciji mrežnih datotek povzroča največ težav implementacija mrežnega direktorija. Le-ta je v splošnem prevelik, da bi se lahko v celoti nahajal v glavnem pomnilniku, zato ga moramo hraniti na disku. To pa pomeni, da je prestrukturiranje mrežnega direktorija v primeru dodajanja novih mej (oziroma brisanja že obstoječih) precej zamudno. V naši implementaciji rešujemo ta problem podobno kot Hinrichs [Hin85a, Hin85b] z uvedbo dvonivojskega mrežnega direktorija. Zgornji nivo (v nadaljnjem besedilu glavni direktorij) skupaj s pripadajočimi skalami predstavlja skalirano verzijo prvotnega enonivojskega mrežnega direktorija in je dovolj majhen, da ga med obdelavo hranimo v

glavnem pomnilniku. Spodnji nivo, ki ustreza prvotnemu direktoriju, je razdeljen na strani in se nahaja na disku. Vsaka stran vsebuje del prvotnega direktorija (v nadaljnjem besedilu poddirektorij) in pripadajoče skale (v nadaljnjem besedilu subskale).

Ta organizacija omogoča, da se prestrukturiranje mrežnega direktorija omeji samo na tisto stran poddirektorija, v kateri je prišlo do spremembe, ostale strani pa niso prizadete. Obenem pa nam ta organizacija zagotavlja, da sta za dostop do iskanega zapisa (ob predpostavki, da poznamo vrednosti atributov, po katerih je zgrajena mreža), potrebna dva dosega na disk: branje ustrezne strani poddirektorija in branje ustreznega podatkovnega bloka.

Dvonivojska organizacija mrežnega direktorija zahteva dopolnitev prej opisanih postopkov za razcep oziroma zlivanje podatkovnih blokov, saj se lahko spremembe zaradi dodajanja oziroma odvzemanja mej odražajo na dveh nivojih: v glavnem direktoriju in v poddirektoriju. Pri razcepu podatkovnega bloka je treba najprej ažurirati subskale in poddirektorij na tekoči strani, to pa lahko privede do prekoračitve velikosti strani. V tem primeru moramo razcepiti stran ter ažurirati glavni direktorij in skale glavnega direktorija.

Podobno velja tudi pri zlivanju podatkovnih blokov: Eliminacija nepotrebne meje v poddirektoriju povzroči zmanjšanje zasedenosti tekoče strani, kar ima lahko za posledico potrebo po zlitju dveh strani, to pa se spet odraža v prestrukturiranju glavnega direktorija in njegovih skal.

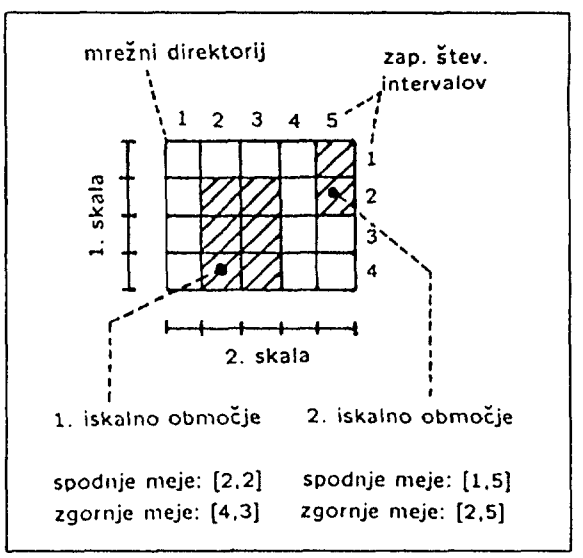
3. POSTOPEK OMEJEVANJA ISKALNEGA OBMOČJA

Učinkovita realizacija poizvedovanj zahteva, da se pri iskanju podatkov omejimo samo na tisti del podatkovnega prostora, v katerem se nahajajo zapisi, ki ustrezajo zahtevanim pogojem. Tipičen primer poizvedovanja, pri katerem pridejo lastnosti mrežnih datotek še posebej do veljave, je intervalsko poizvedovanje. Pri tem poizvedovanju iščemo tiste zapise z datoteke d, ki imajo vrednosti znotraj (za vsak atribut posebej) predpisanega intervala. V mrežnih datotekah dobi to poizvedovanje enostavno geometrično interpretacijo: vsi zapisi, ki usterzajo zahtevanim pogojem, ležijo v podatkovnem prostoru znotraj hiperparalelepipeda, katerega stranice so določene z vrednostmi intervalov v posameznih dimenzijah.

Operacija relacijske algebre .selekcija (v nadaljnjem besedilu jo označujemo v skladu z notacijo iz [Ull80] kot σ_F) predstavlja posplošitev enostavnega intervalskega poizvedovanja. Kot rezultat omejevanja iskalnega območja dobimo pri tej operaciji seznam iskalnih območij, ki v splošnem lahko obsega večje število hiperparalelepipedov.

Formula F, ki ji morajo zadoščati iskani zapisi, je sestavljena iz:

- predikatov oblike *atr op konst*, kjer je *atr* ime atributa, *op* aritmetični operator (=,<>,<=,>=,< ali >) in *konst* konstanta, katere vrednost pripada isti domeni kot vrednost atributa *atr*;
- logičnih operatorjev *and*, *or* in *not*, s katerimi so posamezni predikati povezani med seboj.



Slika 1: Predstavitev seznama iskalnih območij v mrežni datoteki z dvema atributoma

V naši implementaciji operacije selekcija je formula F interno predstavljena v postfiksni obliki, za določitev iskalnega območja pa uporabljamo sklad seznamov iskalnih območij. Vsako iskalno območje v seznamu ima obliko hiperparalelepipeda in je predstavljeno z dvema poljema: poljem spodnjih mej iskalnega območja in poljem zgornjih mej iskalnega območja. Polje spodnjih mej vsebuje zaporedne

številke najnižjih intervalov, polje zgornjih mej pa zaporedne številke najvišjih intervalov v posameznih dimenzijah (glej sliko 1).

Omejevanje podatkovnega prostora, ki ga je treba preiskati, da bi našli zapise, ki ustrezajo formuli F, zahteva dve vrsti aktivnosti: obdelavo predikatov in obdelavo logičnih operatorjev.

Obdelava predikata

Pri obdelavi predikata določimo iskalno območje, ki ustreza tekočemu predikatu, in postavimo to iskalno območje na vrh sklada. Vsak predikat, razen tistega, pri katerem je operator *op* neenačaj, prispeva k omejitvi iskalnega območja v tisti dimenziji, ki ji pripada atribut *atr.* Iskalno območje za posamezen predikat določimo v skladu z naslednjim algoritmom:

inicializiraj iskalno območje predikata tako, da obsega celoten direktorij;
s pomočjo ustrezne skale poišči interval, ki pripada konstanti *konst*;
case operator *op* of
- >: spodnja meja iskalnega območja je enaka zaporedni številki intervala (ali za 1 večja, če konstanta sovpada z zgornjo mejo dobljenega intervala);
- >=: spodnja meja iskalnega območja je enaka zaporedni številki intervala;
- <,<=: zgornja meja iskalnega območja je enaka zaporedni številki intervala;
- =: spodnja in zgornja meja iskalnega območja sta enaki zaporedni številki intervala

end;

Isti algoritem se uporablja tako na nivoju glavnega direktorija kot na nivoju poddirektorija, s to razliko, da na nivoju poddirektorija pred iskanjem intervala preverimo, če vrednost konstante sploh pade v tisto območje, ki ga pokrivajo subskale.

Obdelava logičnega operatorja

Pri obdelavi logičnega operatorja se v primeru operatorja *and* izvrši presek seznamov iskalnih območij, v primeru operatorja *or* pa unija seznamov iskalnih območij, ki se trenutno nahajata na vrhu sklada.

Pri uniji iskalnih območij lahko dobimo seznam iskalnih območij, ki se med seboj prekrivajo. Ta problem rešujemo tako, da med postopkom pregledovanja mrežne datoteke vzdržujemo dva bitna vektorja: vektor pregledanih strani in vektor pregledanih podatkovnih blokov. S tem dosežemo, da se neka stran oziroma podatkovni blok, ki nastopa v dveh ali več različnih hiperparalelepipedih, pregleda samo enkrat.

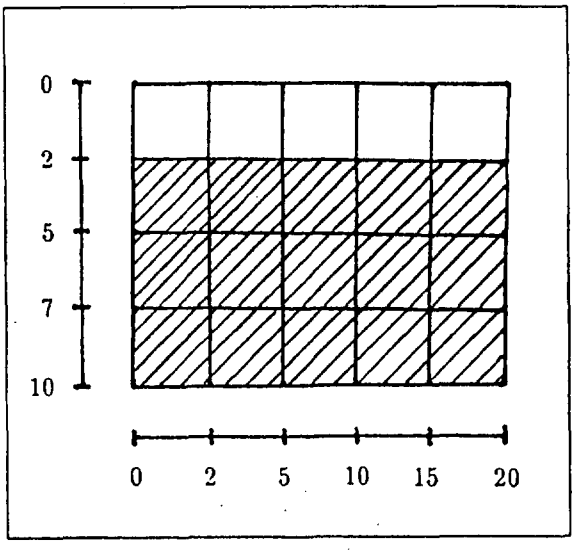
Naša implementacija zaenkrat ne podpira operatorja *not*, saj se je temu operatorju moč izogniti z ustreznim preoblikovanjem formule F. Lahko pa bi operator *not* implementirali tako, da bi poiskali negacijo seznama iskalnih območij, ki se trenutno nahaja na vrhu sklada. Negacija mora poleg komplementov prvotnih iskalnih območij vključevati tudi mejne intervale prejšnjih iskalnih območij.

Postopek omejevanja iskalnega območja v mrežni datoteki z dvema atributoma a_1 in a_2 je za formulo F

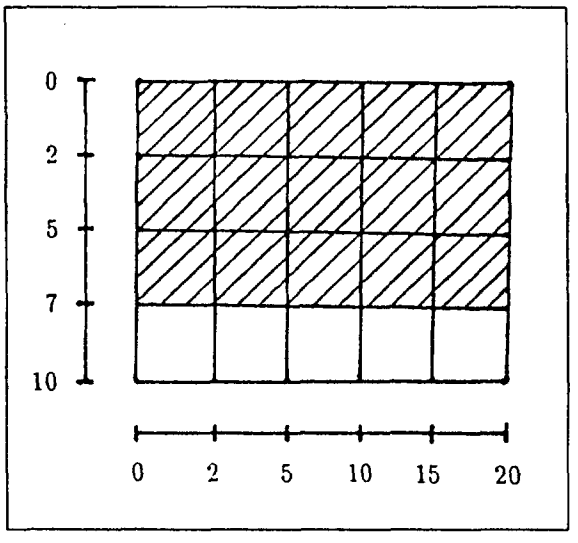
$(a_1>3)$ *and* $(a_1<6)$ *and* $((a_2=4)$ *or* $(a_2>14))$

prikazan na slikah 2a do 2g.

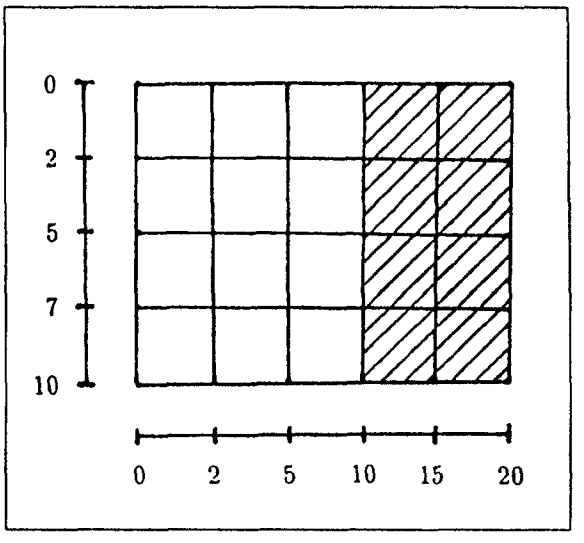
Na sliki 2g je poleg iskalnega območja vrisano tudi poizvedovalno območje, t.j. tisti del podatkovnega prostora, v katerem se dejansko nahajajo zapisi, ki zadoščajo formuli F. V splošnem je poizvedovalno območje podmnožica iskalnega, kar pomeni, da pri ovrednotenju poizvedovanja največkrat pregledamo nekoliko večji del podatkovnega prostora, kot bi bilo potrebno. Zato moramo imeti na voljo mehanizem za izločanje tistih zapisov, ki sicer ležijo v iskalnem območju, niso pa znotraj poizvedovalnega območja.



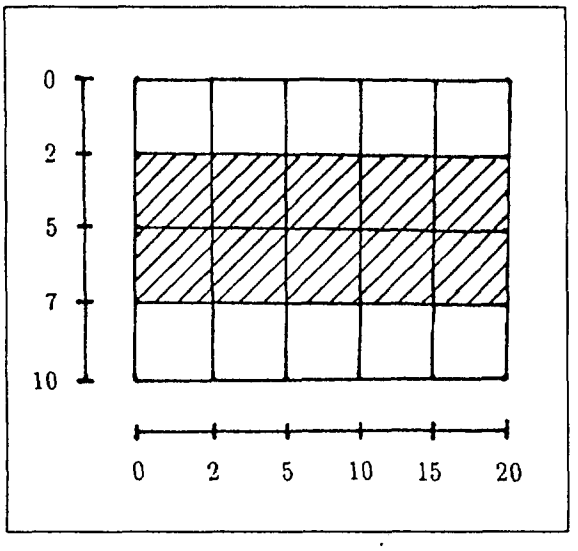
Slika 2a: Predikat $a_1>3$



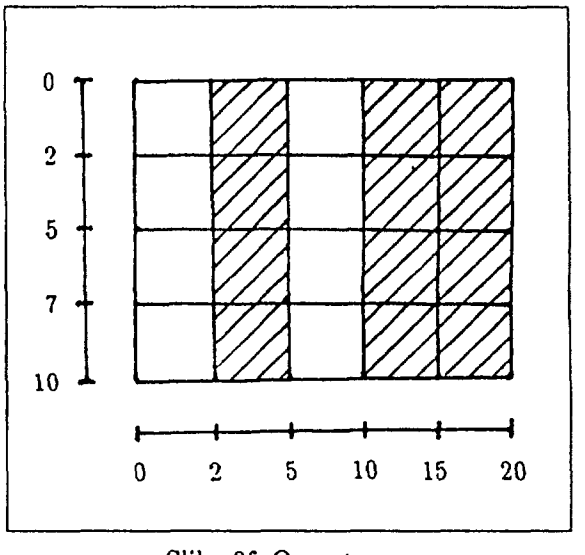
Slika 2b: Predikat $a_1<6$



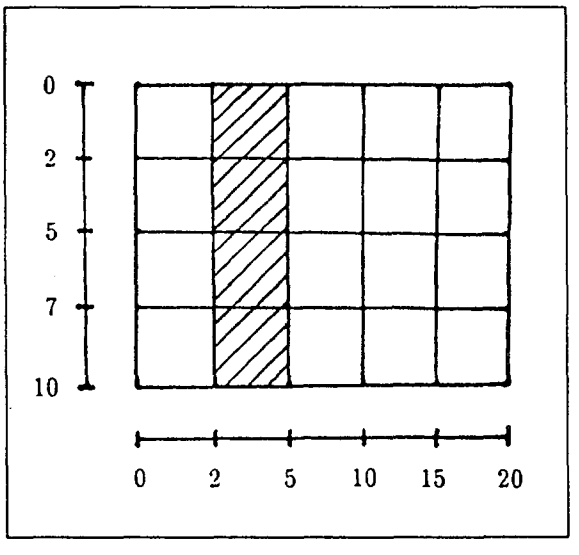
Slika 2e: Predikat $a_2>14$



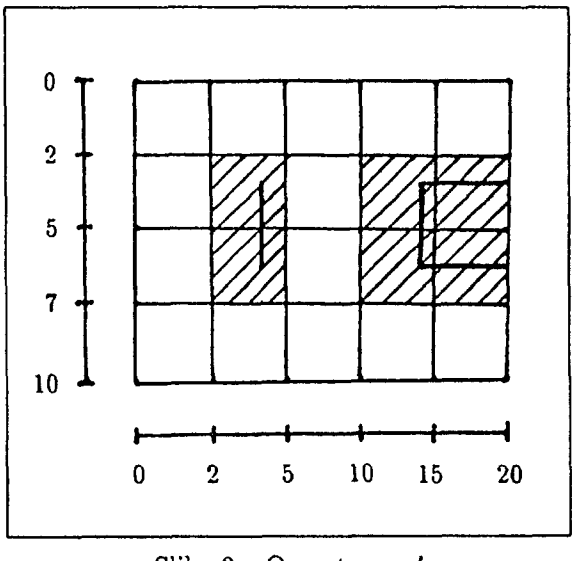
Slika 2c: Operator *and*



Slika 2f: Operator *or*



Slika 2d: Predikat $a_2=4$



Slika 2g: Operator *and*

4. ALGORITEM ZA IMPLEMENTACIJO SELEKCIJE

Implementacija selekcije zahteva, da se zgoraj opisani postopek omejevanja iskalnega območja uporabi na dveh nivojih: na nivoju glavnega direktorija in (za vsako stran posebej) na nivoju poddirektorija. Ko je obdelava formule F končana, dobimo vsakokrat na vrhu sklada ustrezen seznam iskalnih območij. Strani, ki se nahajajo v iskalnem območju glavnega direktorija, in podatkovne bloke, ki se nahajajo v iskalnem območju poddirektorija, lahko obdelujemo v kakršnemkoli vrstnem redu, saj v mrežnih datotekah - v nasprotju z običajnimi načini organizacije podatkov (glej n.pr. [BE77, SACLP79, Yao79]) - problem izbora najcenejše pristopne poti ne obstaja.

V našem programskem paketu je selekcija realizirana v skladu z naslednjim algoritmom:

```
na podlagi formule F določi seznam iskalnih
območij glavnega direktorija;
while niso obdelana vsa iskalna območja gl.dir. do
 while niso obdelane vse strani iz tekočega
  iskalnega območja do
   if stran je v celoti v poizvedovalnem območju
   then
    obdelaj vse podatkovne bloke, ki pripadajo
    tej strani {vsi zapisi v teh podatkovnih
    blokih zadoščajo formuli F}
   else
    begin
     na podlagi formule F določi seznam
     iskalnih območij poddirektorija;
     while niso obdelana vsa iskalna območja
     poddirektorija do
      while niso obdelani vsi podatk. bloki iz
      tekočega iskalnega območja do
       if podatk. blok je v celoti v poizvedoval-
       nem območju then
        vsi zapisi v podatkovnem bloku zado-
        ščajo formuli F
       else
        za vsak zapis posebej preveri, ali leži
        znotraj poizvedovalnega območja
   end;
```

Iz opisa algoritma je razvidno, da se testiranje pripadnosti poizvedovalnemu območju izvaja na treh nivojih: na nivoju strani, na nivoju podatkovnega bloka in na nivoju zapisa.

Če je stran v celoti v poizvedovalnem območju, so tudi vsi pripadajoči podatkovni bloki in v njih shranjeni zapisi v celoti v poizvedovalnem območju, t.j. zadoščajo formuli F. Če pa stran ni v celoti v poizvedovalnem območju, je treba omejiti iskalno območje tudi na nivoju poddirektorija in pregledati samo tiste podatkovne bloke, ki se nahajajo znotraj dobljenega seznama iskalnih območij. Vsak od teh podatkovnih blokov je lahko v celoti ali pa le delno v poizvedovalnem območju. V zadnjem primeru je treba za vsak zapis posebej preveriti, ali zadošča formuli F.

Preverjanje pripadnosti poizvedovalnemu območju izvajamo (podobno kot omejevanje iskalnega območja) s pomočjo sklada, le da so sedaj elementi sklada logične spremenljivke. Vsak predikat formule F povzroči, da se na vrh sklada postavi vrednost *true* ali *false*, medtem ko operatorja *and* in *or* izvršita konjunkcijo oziroma disjunkcijo vrednosti, ki se trenutno nahajata na vrhu sklada, in dobljen rezultat zopet postavita na sklad. Ko je obdelava formule F končana, nam vrednost na vrhu sklada pove, ali je tekoča stran (podatkovni blok, zapis) v celoti znotraj s to formulo določenega poizvedovalnega območja.

Pri ugotavljanju, ali je tekoča stran v celoti v poizvedovalnem območju, privzamemo, da je vrednost atributa *atr*, ki nastopa v predikatu formule F, enaka vrednosti spodnje oziroma zgornje meje območja strani v tisti dimenziji, ki ji pripada atribut *atr*. Če je operator *op* > ali >=, uporabimo vrednost spodnje meje, če pa je operator *op* < ali <=, uporabimo vrednost zgornje meje območja strani. Kadar je operator *op* neenačaj, morata biti vrednosti obeh mej območja strani (spodnje in zgornje) manjši ali večji od konstante *konst*. Če pa je operator *op* enačaj, vedno velja, da stran ni v celoti v poizvedovalnem območju.

Analogno postopamo pri obdelavi podatkovnih blokov, le da sedaj namesto vrednosti spodnje in zgornje meje območja strani upoštevamo vrednosti spodnje in zgornje meje območja podatkovnega bloka.

Pri ugotavljanju, ali je zapis znotraj poizvedovalnega območja, pa se *atr* nadomesti z vrednostjo tega atributa v tekočem zapisu.

5. ALGORITMI ZA IMPLEMENTACIJO SPLOŠNIH POIZVEDOVANJ

Splošna poizvedovanja v relacijskih podatkovnih bazah so najpogosteje sestavljena iz operacij selekcije, projekcije in stika [BE77].

Za učinkovito realizacijo stika je pomembno, da velikost relacij pred stikanjem čim bolj omejimo. Zato pri implementaciji splošnih poizvedovanj (če se le da) uporabljamo hevristično pravilo "Izvrši selekcijo takoj, ko je mogoče" [SC75, WY76, Ull80, KS86].

V običajni fizični organizaciji podatkov je možnost za izvršitev selekcije σ_F pred stikom odvisna od tega, ali obstajajo indeksi po atributih, ki nastopajo v formuli F. Vsak od teh indeksov omogoča omejitev iskalnega območja po enem samem atributu, kar pomeni, da moramo za vse zapise iz tako dobljenega iskalnega območja dodatno preverjati, ali ustrezajo preostalim predikatom iz formule F. Če imamo na voljo večje število indeksov, pregledujemo relacijo po tistem indeksu (pristopni poti), ki terja najmanj stroškov v smislu števila dosegov na disk in obsega procesiranja v glavnem pomnilniku [SACLP79]. Lahko pa s preseki razpoložljivih indeksov zreduciramo iskalno območje [AD75], vendar na račun dodatnega procesiranja, ki je potrebno za izvajanje presekov.

Sistematičen pregled različnih strategij za evaluacijo splošnih poizvedovanj z uporabo običajne fizične organizacije podatkov je podan v [Yao79].

Mrežne datoteke nudijo v primerjavi z običajno indeksno-sekvenčno organizacijo podatkov naslednje prednosti:

- Vedno je možno izvršiti selekcijo pred stikom. Omejitev iskalnega območja lahko izvršimo ne samo po enem, temveč po vseh atributih, ki nastopajo v formuli F (ob predpostavki, da je mreža generirana po vseh atributih relacije).

- Odpade problem pristopnih poti: V mrežnih datotekah se vsi atributi obravnavajo enakovredno, zato je vseeno, v kakšnem vrstnem redu pregledujemo strani in podatkovne bloke.

V okviru naše implementacije mrežnih datotek smo razvili dva algoritma, ki omogočata evaluacijo splošnih poizvedovanj naslednje oblike:

$$\pi(\sigma_{F_1}(R) \underset{i\theta j}{|\times|} \sigma_{F_2}(S))$$

Posamezni simboli imajo (v skladu z [Ull80]) naslednji pomen:

- R in S sta imeni relacij, ki nastopata v poizvedovanju.
- σ_{F_1} predstavlja selekcijo nad relacijo R, σ_{F_2} pa selekcijo nad relacijo S.
- π pomeni projekcijo.
- $|\times|$ predstavlja θ-stik, kjer je θ aritmetični primerjalni operator. Če je θ enačaj, imamo opravka s posebno vrsto stika, ki mu rečemo stik na osnovi enakosti (angl. equijoin).

Prvi algoritem temelji na bločno orientiranem gnezdenju iteracij, drugi pa na stikanju parov rezin, ki vsebujejo iste vrednosti skupnega atributa.

Stik z gnezdenjem

Pri tem algoritmu najprej uporabimo formuli F_1 in F_2 ter z njuno pomočjo omejimo iskalni območji glavnega direktorija v relacijah R oziroma S. Nato obdelujemo strani iz seznama iskalnih območij relacije R in za vsako stran določimo, katere podatkovne bloke moramo pregledati.

Če je stran v celoti vsebovana v poizvedovalnem območju, pregledamo vse pripadajoče podatkovne bloke, v nasprotnem primeru pa s pomočjo formule F_1 omejimo iskalno območje poddirektorija in pregledamo samo tiste podatkovne bloke, ki pripadajo seznamu iskalnih območij.

Za vsak podatkovni blok B_R relacije R skušamo čim bolj omejiti iskalno območje relacije S. Poleg formule F_2 pri tem uporabljamo še vrednosti spodnje in/ali zgornje meje območja podatkovnega bloka B_R v tisti dimenziji, ki ustreza atributu stika. Če je operator θ enačaj, upoštevamo obe meji. V nasprotnih primerih pa uporabimo samo spodnjo mejo (ko je θ > ali >=) oziroma zgornjo mejo (ko je θ < ali <=).

Omejevanje iskalnega območja relacije S zopet izvršimo na dveh nivojih: na nivoju glavnega direktorija in na nivoju poddirektorija. Za vsak podatkovni blok B_R pregledamo samo tiste strani in podatkovne bloke relacije S, ki pripadajo tako omejenemu iskalnemu območju.

Za ta algoritem je značilno, da vsako stran in vsak podatkovni blok prve relacije pregledamo samo enkrat, medtem ko strani in podatkovne bloke druge relacije obdelujemo večkrat. Zato lahko v primeru, ko je število pomnilniških vmesnikov, ki jih imamo na razpolago za hranjenje podatkovnih blokov oziroma strani,

majhno, pride do znatnega povečanja števila dosegov na disk.

Stik po rezinah

V praksi je stik na podlagi enakosti najbolj pogost primer operacije stikanja dveh relacij. Za realizacijo te vrste stika je bila v [JSBS83] uporabljena strategija stikanja z zlivanjem, ki zagotavlja, da se vsaka stran in vsak podatkovni blok obeh relacij obdela samo enkrat. Pač pa ta strategija zahteva, da obe relaciji pred zlivanjem sortiramo, saj v mrežnih datotekah ni moč vzdrževati urejenosti relacije po določenem atributu oziroma skupini atributov.

Algoritem za stik po rezinah, ki smo ga implementirali v okviru našega programskega paketa, ne zahteva predhodnega sortiranja obeh relacij, obenem pa zagotavlja (ob predpostavki, da imamo v glavnem pomnilniku dovolj prostora, da hranimo tekoči rezini obeh relacij), da vsak podatkovni blok in vsako stran obeh relacij preberemo z diska samo enkrat.

Zasnova algoritma temelji na dejstvu, da je moč vsako mrežno datoteko razdeliti na rezine. Na nivoju glavnega direktorija so rezine določene s tisto skalo glavnega direktorija, ki pripada atributu stika. Na enak način lahko vsako rezino glavnega direktorija "razrežemo" še naprej z uporabo subskal na nivoju poddirektorija. Vsaka rezina je omejena z dvema zaporednima mejama v skali. To pomeni, da vsaka rezina vsebuje samo tiste zapise, pri katerih je vrednost atributa stika znotraj intervala, določenega s spodnjo in zgornjo mejo rezine.

Da bi izračunali stik dveh relacij, obdelujemo rezine po parih: eno rezino relacije R in eno rezino relacije S. Na ta način se računanje stika prevede na zaporedje stikov parov rezin, ki so mnogo manjše od prvotnih relacij in jih lahko v večini primerov hranimo v glavnem pomnilniku. Obdelati je treba samo tiste pare rezin, ki vsebujejo zapise z istimi vrednostmi atributa stika.

Par rezin lahko staknemo na različne načine (z zlivanjem, z gnezdenjem iteracij itd.). V naši implementaciji smo se odločili za uporabo binarnega drevesa kazalcev na zapise prve rezine, ki ga zgradimo v glavnem pomnilniku in s tem pospešimo iskanje zapisov, pri katerih se vrednosti atributa stika ujemajo.

Čeprav lahko isti podatkovni blok nastopa v več zaporednih rezinah, ga beremo z diska samo enkrat. To dosežemo na ta način, da vsako rezino preberemo v glavni pomnilnik v dveh korakih:

- Najprej sprostimo prostor, ki ga zasedajo podatkovni bloki, ki ne pripadajo naslednji rezini. V pomnilniku tako ostanejo samo tisti podatkovni bloki prejšnje rezine, ki pripadajo tudi naslednji. Organizacija mrežnih datotek zagotavlja, da nobenega od sproščenih podatkovnih blokov v nadaljnjem postopku ne bomo več potrebovali.

- Nato preberemo z diska še preostale podatkovne bloke, ki spadajo v tekočo rezino, a jih še ni v glavnem pomnilniku.

Vsak podatkovni blok se tako prebere z diska samo takrat, ko ga prvič potrebujemo, nato pa ostane v glavnem pomnilniku toliko časa, dokler ne obdelamo vseh rezin, v katerih se nahaja.

Podrobnejši opis obeh algoritmov je podan v [Mah89].

6. OPIS PROGRAMSKEGA PAKETA

Programski paket za vzdrževanje mrežnih datotek in izvajanje poizvedovanj na osnovi operacij relacijske algebre, ki smo ga razvili v okviru naših raziskav, je napisan v programskem jeziku Turbo Pascal [Bor87] in je prilagojen za uporabo na osebnih računalnikih tipa IBM PC XT/AT.

Vsaka mrežna datoteka je predstavljena z dvema Pascal-skima datotekama v skladu z naslednjimi deklaracijami:

```
const DolzinaBloka=512;
type Blok=array [1..DolzinaBloka] of char;
var PodMD,DirMD: file of Blok;
```

Datoteka PodMD hrani podatke, t.j. zapise, ki pripadajo neki relaciji. V vsakem bloku je lahko večje število zapisov, maksimalna dolžina zapisa pa ne sme presegati dolžine bloka.

Na datoteki DirMD je shranjen mrežni direktorij, ki je razdeljen na strani. Vsak zapis datoteke DirMD vsebuje eno stran poddirektorija. Poleg poddirektorija so na datoteki DirMD shranjeni tudi podatki o

glavnem direktoriju, podatki o strukturi zapisa (tip, začetek in dolžina vsakega atributa) ter vektorja zasedenih strani in podatkovnih blokov.

Ob kreiranju mrežne datoteke moramo definirati strukturo zapisa. Za vsak atribut navedemo njegovo ime, tip in dolžino. Če želimo po tem atributu graditi mrežo, pa moramo podati tudi minimalno in maksimalno vrednost, ki jo ta atribut lahko zavzame. Ti dve vrednosti se uporabita kot spodnja in zgornja meja v pripadajoči skali.

Naša implementacija podpira zaenkrat naslednje tipe atributov: *byte, integer, word, real* in *character (niz znakov)*. Dolžina atributov tipa *byte, integer, word* in *real* je določena skladno z interno predstavitvijo podatkov tega tipa v Turbo Pascal-u in znaša 1, 2, 2 oziroma 6 bytov. Dolžino atributov tipa *character* določa uporabnik, vendar le-ta ne sme preseči 255 znakov.

Programski paket je sestavljen iz 8 programskih enot (angl. units) z naslednjo vsebino:

- Programska enota DEKL vsebuje deklaracije konstant, tipov in spremenljivk, ki so skupne za celoten programski paket in jih uporabljajo vse ostale enote.

- Programska enota SKUPNE obsega vse skupne podprograme, ki jih lahko kličemo iz katerekoli druge enote. Med te podprograme spadajo:

  podprogrami za pretvorbo numeričnih podatkov (tipa *byte, integer, word* ali *real*) v znakovno obliko in obratno;

  podprogrami za vpisovanje podatkov v glavni direktorij, poddirektorij in pripadajoče skale;

  podprogrami za rekonstrukcijo podatkov, ki so zapisani v glavnem direktoriju, poddirektoriju in pripadajočih skalah (n.pr. število mej, vrednost in položaj posamezne meje, vrednost in položaj posameznega elementa v glavnem direktoriju oziroma poddirektoriju, število vseh elementov glavnega direktorija oziroma poddirektorija itd.;

  podprogrami za prenos podatkovnih blokov in strani z diska v glavni pomnilnik in obratno;

  podprogrami za določanje koordinat tiste celice v mreži, v katero se preslika zapis kot točka v podatkovnem prostoru;

  podprogrami za ugotavljanje območja strani in območja podatkovnega bloka

  itd.

- Programska enota VZDR vsebuje podprograme, ki so potrebni za vzdrževanje mrežnih datotek. To so podprogrami za kreiranje, odpiranje in zapiranje mrežnih datotek. Novo datoteko lahko kreira uporabnik, lahko pa jo dobimo kot rezultat nekega poizvedovanja. V tem primeru se struktura mrežne datoteke generira avtomatsko v skladu s seznamom atributov, navedenih v sklopu projekcije.

- Programska enota VST vsebuje podprogram za vstavljanje zapisov v mrežno datoteko, v okviru katerega so implementirani vsi potrebni algoritmi za dodajanje novih mej, razcep podatkovnih blokov in razcep strani. Programski paket omogoča tako interaktivno dodajanje posameznih zapisov kot tudi dodajanje zapisov z druge datoteke, ki mora biti tipa *text*, podatki na njej pa morajo biti pripravljeni v skladu s strukturo zapisa na mrežni datoteki.

- Programska enota BRIS vsebuje podprogram za brisanje zapisov iz mrežne datoteke, vključno s podprogrami za zlivanje podatkovnih blokov, izločanje nepotrebnih mej in zlivanje strani.

- Programska enota SKPOIZ obsega podprograme, ki so skupni za različna poizvedovanja. V glavnem gre za podprograme, ki omogočajo interpretacijo formule F pri selekciji, omejevanje iskalnega območja in ugotavljanje, ali je tekoča stran, podatkovni blok oziroma zapis v celoti v poizvedovalnem območju.

- Programska enota POIZ vsebuje podprograme za izvajanje različnih poizvedovanj. Na voljo so podprogrami za izvajanje selekcije in splošnih poizvedovanj, sestavljenih iz projekcije, selekcije in stika.

  Splošna poizvedovanja, ki vključujejo stik dveh datotek, so realizirana s tremi različnimi podprogrami: z bločno orientiranim gnezdenjem iteracij, s stikanjem po rezinah glavnega direktorija in s stikanjem po rezinah poddirektorija.

Pri stikanju po rezinah glavnega direktorija se rezine formirajo samo na osnovi skal glavnega direktorija, zato je ta metoda hitrejša, vendar daje večje rezine in je primerna samo za manjše datoteke. Pri stikanju po rezinah poddirektorija pa pri oblikovanju rezin upoštevamo tudi subskale ter s tem zmanjšamo velikost posameznih rezin. Zato je ta metoda primerna za večje datoteke.

Programska enota INFO vsebuje podprograme za izpis vsebine mrežne datoteke in mrežnega direktorija. Izpisujemo lahko celotno datoteko, posamezne podatkovne bloke in posamezne zapise: Od podatkov, ki so shranjeni v okviru mrežnega direktorija, lahko izpišemo strukturo datoteke (tip, začetek in dolžino posameznih atributov), glavni direktorij in posamezne strani poddirektorija.

V to enoto spadajo tudi podprogrami za izračun raznih statističnih podatkov (n.pr. število zasednih strani in podatkovnih blokov, povprečna velikost območja strani oziroma podatkovnega bloka, povprečno število elementov v poddirektoriju itd.).

7. SKLEP

V članku so opisane osnovne značilnosti mrežnih datotek in algoritmi za implementacijo poizvedovanj na osnovi operacij relacijske algebre. Bistvena prednost mrežnih datotek je, da omogočajo učinkovito omejevanje podatkovnega prostora, ki ga je treba preiskati, da bi našli podatke, ki ustrezajo posameznim poizvedovanjem. Postopek omejevanja iskalnega območja predstavlja izhodišče za realizacijo selekcije.

Splošna poizvedovanja, ki zahtevajo stikanje dveh datotek, lahko realiziramo na osnovi različnih strategij. V članku sta opisana dva algoritma: prvi temelji na bločno orientiranem gnezdenju iteracij, drugi pa na stikanju parov rezin. S pomočjo drugega algoritma se postopek stikanja dveh obsežnih relacij pretvori v zaporedje stikov mnogo manjših relacij. S tem se izognemo problemu sortiranja relacij, ki je potrebno pri stiku z zlivanjem. Poleg tega pa dosežemo, da se vsaka stran in vsak podatkovni blok obeh relacij prebere z diska samo enkrat ob predpostavki, da je v glavnem pomnilniku dovolj prostora za en par rezin.

Vse opisane algoritme smo implementirali v okviru programskega paketa, ki omogoča vzdrževanje mrežnih datotek in izvajanje ustreznih poizvedovanj. Način predstavitve mrežnih datotek in zgradba programskega paketa sta opisana v zadnjem razdelku.

8. SODELAVCI

Implementacijo algoritma za brisanje zapisov je v okviru svojega diplomskega dela opravila študentka Fakultete za elektrotehniko in računalništvo Nataša Žabkar. Študent Božo Urh pa v svoji diplomski nalogi proučuje vpliv različnih načinov oblikovanja mreže na velikost mrežnega direktorija in performanse posameznih poizvedovanj. Obema se za njun prispevek k obravnavani problematiki najtopleje zahvaljujem.

LITERATURA:

[AD75] M.M. Astrahan, D.D. Chamberlin: IMPLEMENTATION OF A STRUCTURED ENGLISH QUERY LANGUAGE, Communications of the ACM, Vol. 18, No. 10, October 1975

[BE77] M.W. Blasgen, K.P. Eswaran: STORAGE AND ACCESS IN RELATIONAL DATA BASES, IBM Systems Journal, No. 4, 1977

[Bor87] TURBO PASCAL OWNER'S HANDBOOK, Version 4.0, Borland International, 4585 Scotts Valley Drive, Scotts Valley, CA 95066, ISBN 0-87524-170-0, 1987

[Hin85a] K.H. Hinrichs: THE GRID FILE SYSTEM: IMPLEMENTATION AND CASE STUDIES OF APPLICATIONS, Diss. ETH Nr. 7734, Zurich, 1985

[Hin85b] K.H. Hinrichs: IMPLEMENTATION OF THE GRID FILE: DESIGN CONCEPTS AND EXPERIENCE, BIT 25, 1985

[KS86] H.F. Korth, A. Silberschatz: DATA-

BASE SYSTEMS CONCEPTS, McGraw-Hill, 1986

[JSBS83] S. M. Joshi, S. Sanyal, S. Banerjee, S. Srikumar: USING GRID FILES FOR A RELATIONAL DATABASE MANAGEMENT SYSTEM, Technical Report No. 11, Speech and Digital Systems Group, Tata Institute of Fundamental Research, Bombay, 1983

[Mah89] V. Mahnič: COMPUTING JOINS OF RELATIONS USING GRID FILES AS A PHYSICAL STORAGE STRUCTURE, Zbornik del 11. mednarodnega simpozija "Kompjuter na sveučilištu", Cavtat, junij 1989

[NHS84] J. Nievergelt, H. Hinterberger, K.C. Sevcik: GRID FILE: AN ADAPTABLE, SYMMETRIC MULTIKEY FILE STRUCTURE, ACM Transactions on Database Systems, Vol. 9, No. 1, March 1984

[SACLP79] P.G. Selinger, M.M. Astrahan, D.D. Chamberlin, R.A. Lorie, T.G. Price: ACCESS PATH SELECTION IN A RELATIONAL DATABASE MANAGEMENT SYSTEM, Proceedings of the ACM SIGMOD International Conference on the Management of Data, 1979

[SC75] J.M. Smith, P.Y. Chang: OPTIMIZING THE PERFORMANCE OF A RELATIONAL ALGEBRA DATABASE INTERFACE, Communications of the ACM, Vol. 18, No. 10, October 1975

[Sch86] E. Schikuta: THE GRIDFILE: A DATA STRUCTURE FOR RELATIONAL DATABASE SYSTEMS, Zbornik del 8. mednarodnega simpozija "Kompjuter na sveučilištu", Cavtat, maj 1986

[Ull80] J.D. Ullman: PRINCIPLES OF DATABASE SYSTEMS, Computer Science Press, Potomac, Maryland, 1980

[WY76] E. Wong, K. Youssefi: DECOMPOSITION - A STRATEGY FOR QUERY PROCESSING, ACM Transactions on Database Systems, Vol. 1, No. 3, September 1976

[Yao79] S.B. Yao: OPTIMIZATION OF QUERY EVALUATION ALGORITHMS, ACM Transactions on Database Systems, Vol 4., No. 2, June 1979