Arhitecturi ale sistemelor paralele şi structuri de date pentru organizarea fizică a bazelor de date

Georgios KAPENEKAS
Technical Education Institute, Halkida, Greece

Sistemele paralele de baze de date au în compunere mai multe procesoare şi mai multe unităţi de discuri conectate prin intermediul unei reţele de interconectare rapidă. Funcţie de arhitectura hardware a maşinii de bază, precum şi funcţie de structurile de date utilizate în păstrarea şi regăsirea datelor se poate asigura întregului sistem o creştere a puterii de calcul prin creşterea posibilităţii de prelucrare în paralel a operaţiilor şi optimizarea interogărilor asupra bazei de date.

Cuvinte cheie: *sistem paralel, reţea, comunicaţie, înregistrare, index, bloc, fişier*

Arhitecturi ale sistemelor paralele

Sistemele paralele îmbunătăţesc viteza de prelucrare şi operaţiile de intrare/ieşire prin folosire mai multor unităţi centrale şi discuri în paralel. Pentru a asigura comunicaţia între componentele unui sistem (procesoare, memorie şi discuri) se foloseşte aşa numita *reţea de intercomunicare*. Tipurile posibile de reţele sunt:

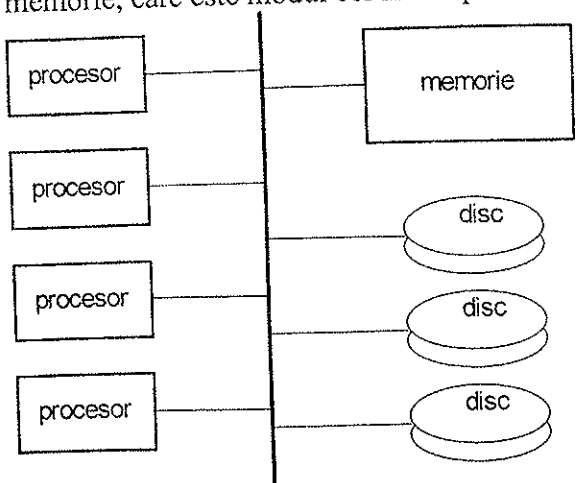
Magistrala. Toate componentele sistemului pot transmite sau recepţiona date doar printr-o singură magistrală de comunicaţie. Arhitectura de tip magistrală funcţionează bine pentru un număr redus de procesoare. În acest caz creşterea paralelismului nu este prea mare deoarece magistrala poate realiza comunicaţia doar pentru o singură componentă la un moment dat.

Plasa. Componentele constituie nodurile unei grile, în care fiecare este conectat cu toate componentele vecine din grilă. Într-o reţea bidimensională, fiecare nod este conectat cu alte patru noduri adiacente, iar pentru o reţea tridimensională, cu şase noduri adiacente. Pentru nodurile care nu se învecinează, comunicaţia se face cu mesaje transmise prin noduri intermediare care sunt conectate direct. Deoarece numărul de legături de comunicaţie creşte odată cu creşterea numărului de componente, îmbunătăţind capacitatea de comunicaţie, creşte şi paralelismul sistemului.

Hipercubul. Componentele sunt numerotate în binar, iar o componentă este conectată direct cu o alta pentru care reprezentarea binară a etichetei diferă doar printr-un bit. În acest caz, fiecare din cele *n* componente din sistem este conectată şi poate comunica direct cu *log(n)* componente, caz în care întârzierile datorate comunicaţiei între componente pentru o conexiune hipercub sunt mult reduse.

Având la bază reţeaua de interconectare, funcţie de modul de utilizare a componentelor se pot crea următoarele arhitecturi paralele pentru o bază de date.

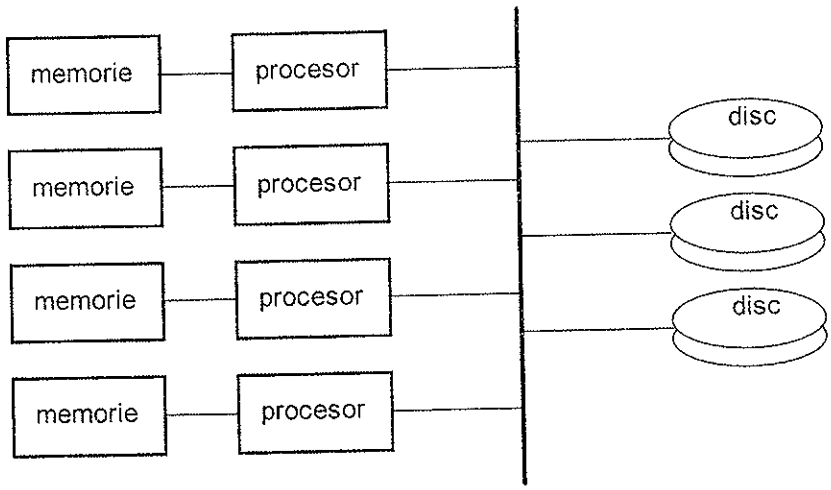
Arhitectură cu partajarea memoriei. Procesoarele şi discurile au acces la o memorie comună printr-o magistrală sau o reţea de intercomunicaţie. Principalul avantaj este eficienţa deosebită a comunicaţiei între procesoare; data din memoria comună poate fi accesată de oricare procesor fără a fi nevoie un transfer soft. Transmisia de mesaje între procesoare se face prin scriere în memorie, care este modul cel mai rapid.



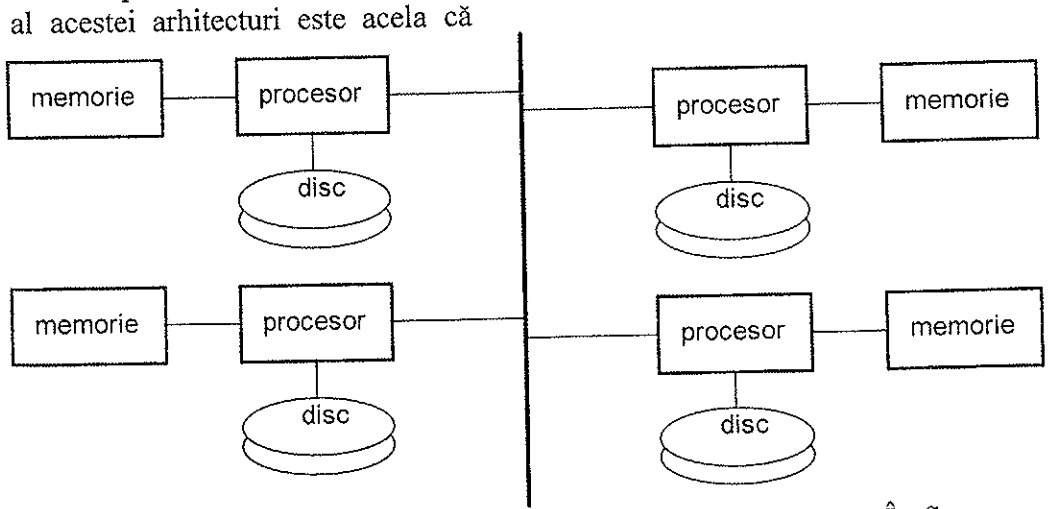
Dezavantajul principal este că există o limi-

tare puternică în cazul în care numărul de procesoare depăşeşte 32 sau 64, deoarece reţeaua de comunicaţie constituie o constrângere puternică, procesoarele consumând timp în aşteptarea validării accesului la magistrală.

Arhitectură cu discuri partajate. Procesoarele au acces direct la toate discurile existente în sistem, iar fiecare are propria memorie. Această arhitectură are două avantaje importante: prin existenţa unei memorii proprii se evită o strangulare importantă a sistemului şi în plus se asigură o bună toleranţă la defecte. Dacă un procesor sau o memorie se defectează, celelalte procesoare pot prelua sarcinile acestuia, deoarece baza de date se află stocată pe discuri şi poate fi accesată direct. Dezavantajele arhitecturii constau în reţeaua de intercomunicaţie cu discurile, care devine o strangulare a sistemului, iar comunicaţia între procesoare necesită folosirea reţelei de comunicaţie.

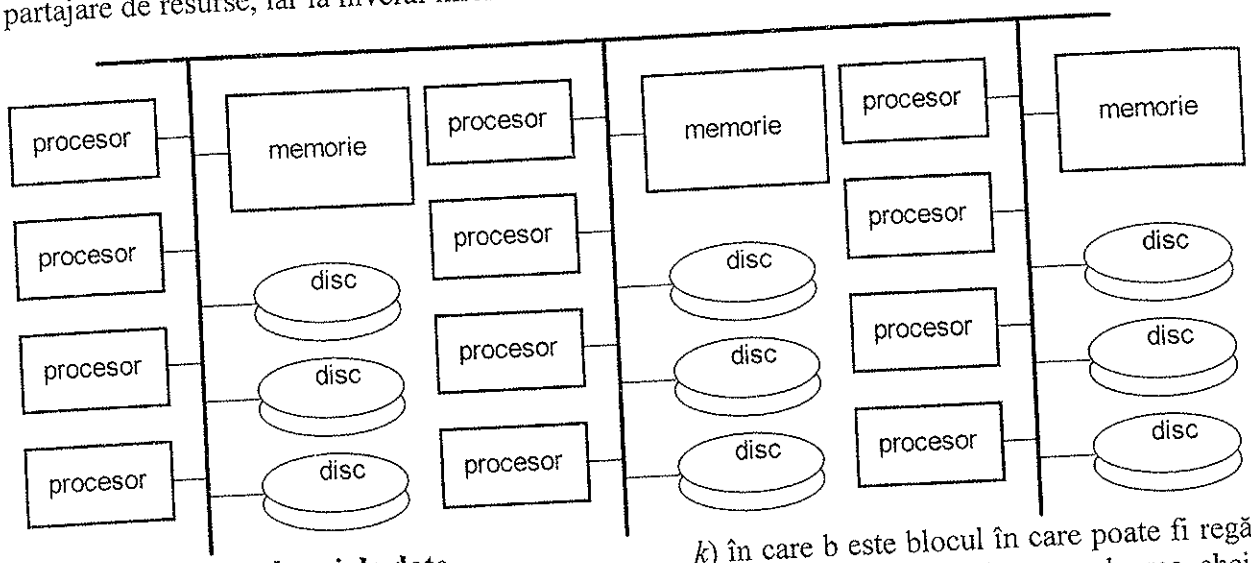


Arhitectură fără a partaja nimic. Fiecare nod constă dintr-un procesor, o memorie şi un disc, iar comunicaţia între procesoare se face prin reţeaua de intercomunicaţie. Un astfel de nod funcţionează ca un server pentru datele aflate pe discul local. Principalul avantaj al acestei arhitecturi este acela că referinţele la data locală aflată pe discul din nod nu mai trebuie transmise prin reţeaua de intercomunicaţie. Pe de altă parte, interogările care folosesc date de pe discurile care nu sunt locale necesită o comunicaţie între procesoare care este deosebit de costisitoare.



Arhitectură ierarhică. Arhitectura combină caracteristicele celor trei arhitecturi prezentate anterior. La nivelul cel mai ridicat, există noduri care sunt legate printr-o reţea de intercomunicaţie în care nu există discuri sau memorii în comun. În fiecare nod există un sistem cu un număr redus de procesoare care au o memorie comună sau un sistem cu un număr de procesoare care accesează în comun un număr de discuri. În acest mod, la

nivelul ierarhic superior arhitectura este fără partajare de resurse, iar la nivelul inferior fie există o arhitectură cu memorie comună, fie o arhitectură cu discuri comune.



Structura fizică a bazei de date

O bază de date este păstrată ca o colecție de înregistrări, fiecare compusă din unul sau mai multe câmpuri. Valorile câmpurilor sunt de tip elementar: întregi, reale, șiruri de caractere de lungime fixă. Mai pot fi regăsite și câmpuri de tip pointer sau șiruri de caractere de lungime variabilă.

În gestionarea bazei de date se prelucrează colecții de înregistrări care au același număr de câmpuri și același tip de dată pentru câmpurile corespondente. Colecția de înregistrări care are un același format o numim fișier, ea fiind reprezentarea fizică pentru o relație.

Un factor care influențează costul operațiilor asupra bazei de date este modul de păstrare a informației, împărțită în blocuri. Transferul datei între memoria secundară de stocare și memoria principală de prelucrare se realizează doar în unități complete de astfel de blocuri. Se poate defini unitatea de cost pentru operații pe blocurile fizice ca fiind citirea sau scrierea unui singur bloc, numită și *acces bloc*.

Un pointer către o înregistrare reprezintă data suficientă pentru a localiza înregistrarea respectivă în memoria secundară. Datorită varietății structurilor pentru data stocată poate varia și natura pointerului. Se poate folosi adresa absolută din memoria virtuală sau adresa sistem de pe disc. Adesea se preferă folosirea ca pointer a unei perechi (b, k) în care b este blocul în care poate fi regăsită înregistrarea, iar k este valoarea cheie pentru înregistrare. În acest caz, pentru a regăsi înregistrarea în cadrul blocului trebuie ca: toate înregistrările din blocul b să aibă același format; să poată fi regăsit începutul fiecărei înregistrări din bloc, iar pentru fiecare înregistrare din bloc să fie posibilă delimitarea câmpurilor pentru a putea regăsi valorile câmpurilor.

Există situații frecvente în care interogarea unei baze de date selectează doar o mică parte din înregistrări. Pentru acest caz este costisitoare ca timp, parcurgerea întregii informații din baza de date, fiind necesare modalități de accesare rapidă a datei. Pentru a determina metoda optimă pentru o anumită aplicație se vor evalua următorii factori:

- *tipul de acces* care este cel mai eficient; tipul poate include regăsirea unei înregistrări cu o anumită valoare pentru un atribut sau regăsirea mai multor înregistrări pentru care valorile atributelor se încadrează într-un inteval;
- *timpul de acces*, care este timpul necesar pentru a regăsi un anumit articol sau o mulțime de articole;
- *timpul de inserare* este timpul necesar pentru a regăsi locul corect în care va fi adăugat un nou articol ca și timpul necesar actualizării structurii cu informația de regăsire a înregistrărilor;

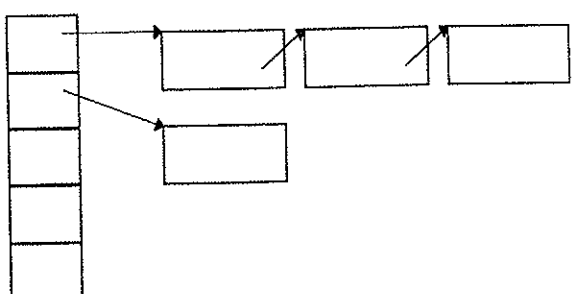
- *timpul de ştergere* este timpul necesar eliminării unui articol şi refacerea structurii cu informaţia de regăsire a înregistrărilor;
- *spaţiul necesar* care reprezintă spaţiul suplimentar structurilor cu informaţia de regăsire a înregistrărilor;

Fişiere heap

Modul cel mai simplu de organizare a înregistrărilor este cel sub formă *heap*, în care înregistrările sunt împachetate în blocuri fără o ordine specială şi fără o organizare specială a blocurilor. Singura presupunere care se face este aceea că pointerii către toate blocurile din fişier sunt diponibili, iar aceştia se află în memoria principală. Dacă există prea multe blocuri, astfel încât pointerii să nu poată fi păstraţi în memoria principală, atunci pointerii vor fi păstraţi şi ei în memoria secundară şi sunt accesaţi când este necesar.
Pentru a analiza eficienţa unei astfel de organizări a informaţiei presupunem că există în total n înregistrări, iar într-un bloc se află R înregistrări. În acest caz numărul minim de blocuri necesare pentru a păstra fişierul este $[n/R]$. Timpul necesar pentru a executa o operaţie de tip inserare, ştergere sau căutare va fi măsurat în numărul de blocuri care trebuie schimbate între memoria secundară şi cea principală. În aceste condiţii, pentru a căuta o înregistrare în fişier heap trebuie citite în medie $n/2R$ blocuri până aceasta este regasită, iar dacă aceasta nu se află în fişier se parcurg n/R blocuri. Pentru a insera o nouă înregistrare trebuie regăsită ultima înregistrare din heap. Dacă acest ultim bloc nu mai conţine spaţiu pentru a păstra noua înregistrare trebuie scris într-un nou bloc. În ambele cazuri are loc doar o scriere a unui bloc în memoria secundară şi o citire a unui bloc.
Ştergerea unei înregistrări necesită regăsirea acesteia în fişier, urmată de o rescriere a blocului conţinând înregistrarea respectivă, deci o medie de $n/2R+1$ accesări de bloc dacă înregistrarea este regăsită şi n/R accesări dacă înregistrarea nu se află în fişier. Pentru înregistrările care sunt şterse există două situaţii:
- ştergerea logică, când există un bit setat corespunzător, va fi înregistrat păstrând în continuare informaţia din bloc;
- ştergerea fizică, când înregistrarea este eliminată fizic. În acest caz, este necesară rearanjarea informaţiei pe blocuri, pentru a elimina spaţiul ocupat de înregistrarea ştearsă. Dacă înregistrările sunt de lungime fixă procesul este mai simplu: este căutată ultima înregistrare din ultimul bloc, care va fi mutată în zona ştearsă. În cazul în care înregistrările au lungime variabilă se compactează spaţiul din cadrul blocului, iar dacă s-a creat suficient spaţiu în bloc pentru a putea muta o înregistrare din ultimul bloc, se realizează această schimbare. Altfel, spaţiul este lăsat nefolosit urmând ca la o nouă ştergere din bloc să existe suficient loc pentru a se putea muta înregistrări.

Fişiere grupate

Ideea de bază în organizarea fişierelor pe grupe este aceea a împărţirii înregistrărilor din fişier în grupe după valoarea unei chei. Pentru fiecare astfel de fişier există o funcţie de grupare, h, care are ca argument valoarea cheii şi produce un întreg ce corespunde grupului căruia îi aparţine înregistrarea.
Fiecare grupă este constituită dintr-un număr de blocuri, fiecare bloc păstrând un pointer către următorul bloc din grupă.
Există un vector de pointeri numit *directorul grupelor* care pentru fiecare intrare păstrează un pointer către primul bloc al grupei respective. Toate blocurile dintr-o grupă sunt legate într-o listă încheiată cu un pointer *null*.



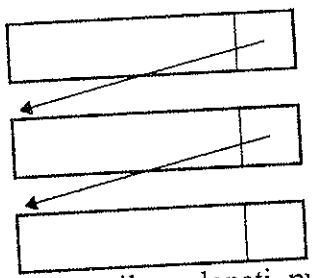
Funcţiile de grupare trebuie să producă un întreg cuprins între 1 şi numărul maxim de grupe posibile pentru fiecare valoare a cheii.

În plus, toate valorile rezultate trebuie să aibă aceeaşi probabilitate de apariţie.
Pentru a regăsi o înregistrare după o cheie cu valoarea v se calculează funcţia de grupare $h(v)$, pentru a găsi grupa din care face parte înregistrarea. Pornind cu pointerul din directorul grupei se parcurge lista asociată acestei grupe până când este regăsită înregistrarea sau se ajunge la sfârşitul listei. În acest ultim caz, nu se continuă căutarea şi în altă grupă, înregistrarea căutată considerându-se că nu se află în fişier.
Pentru a insera o înregistrare se calculează funcţia de grupare pentru a determina grupa în care va fi adăugată înregistrarea, după care trebuie parcursă lista din grupa respectivă pentru a regăsi sfârşitul listei, deoarece numai în acest loc există loc pentru inserare. Pentru a reduce timpul de căutare, se poate folosi o structură de date care să păstreze în directorul grupelor pointeri atât spre începutul, cât şi spre sfârşitul listei din grupă. Dacă s-a regăsit ultimul bloc din grupă şi mai este loc, poate fi adăugată noua înregistrare, altfel se alocă un nou bloc care este adăugat la listă. Ştergerea unei înregistrări presupune un proces similar de regăsire a înregistrării şi setarea bitului de înregistrare ştearsă din cadrul înregistrării. În cazul în care se face şi eliminarea fizică a înregistrării trebuie realizată o compactare a blocurilor cu o eventuală reducere a numărului de blocuri necesare pentru grupa ce conţine înregistrarea.
Principalul avantaj al fişierului grupat în B grupe este că el se comportă ca un fişier heap care este de B ori mai scurt, realizarea operaţiilor se execută într-un timp de B ori mai scurt. Numărul de grupe în care se va împărţi un fişier este limitat din următoarele consideraţii:
- o grupă trebuie să conţină cel puţin un bloc (sub această limită nu poate fi mărit numărul de blocuri);
- este preferabil ca directorul grupelor să fie păstrat în memoria principală (dacă acest director devine prea mare sunt necesare accesări suplimentare pentru a citi noi blocuri din director).

Dacă fişierul are n înregistrări din care R pot încăpea într-un bloc, iar fişierul este grupat în B grupe a căror director poate fi păstrat în memoria principală sunt necesare, în medie: $n/2BR$ accesări de blocuri în cazul unei căutări cu succes, a unei ştergeri sau a modificării unei înregistrări existente şi n/BR accesări de blocuri pentru o căutare fără succes.

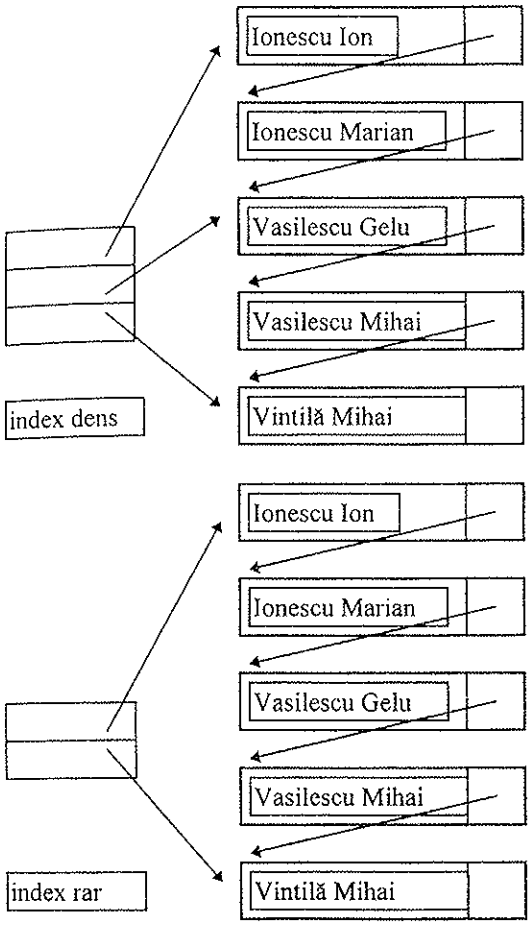
Fişiere indexate

Pentru a realiza un acces aleatoriu rapid la un fişier se poate folosi o structură de indexare. Fiecare structură va fi asociată cu o cheie de căutare. Înregistrările în fişierul indexat sunt păstrate sortate. Un fişier poate avea mai mulţi indexuri după diferite chei de căutare. Dacă fişierul ce conţine înregistrările este ordonat secvenţial, indexul a cărui cheie de căutare determină ordinea secvenţială a fişierului se numeşte index primar, iar indexurii a căror cheie de căutare determină o altă ordine decât cea secvenţială a fişierului se numesc indexuri secundari.
Index primar. Reprezintă unul din cele mai vechi moduri de ordonare din sistemele bază de date. El este proiectat pentru aplicaţii care necesită atât prelucrarea secvenţială a fişierului cât şi un acces aleator la înregistrări.



Pentru indexurile ordonaţi putem să avem următoarele situaţii:
- index dens: - caz în care o înregistrare index apare pentru fiecare valoare a cheii de căutare din fişier. Înregistrarea index conţine valoarea cheii de căutare şi un pointer către prima înregistrare cu această valoare a cheii de căutare.
- index rar: o înregistrare index este creată pentru o parte din valori. Fiecare înregistrare index are, de asemenea, o valoare cheie de căutare şi un pointer către prima înregistrare care are această valoare a cheii de căutare. Pentru a localiza o înregistrare, în acest caz, se caută intrarea index cu cea mai mare valoare a cheii de căutare mai mică sau egală

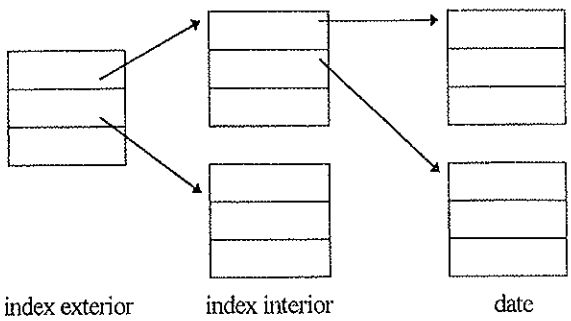
cu valoarea căutată. Pornind de la înregistrarea indicată de intrarea index se parcurg pointerii din fişier până se găseşte înregistrarea dorită.



După cum se vede, în general, localizarea înregistrării se face mai rapid dacă se foloseşte un index dens. Totuşi, indexurile rari au avantajul că necesită un spaţiu suplimentar mai mic, iar la inserarea şi ştergerea de înregistrări din baza de date sunt necesare prelucrări mai reduse. Decizia privind care tip de index va fi folosit depinde de aplicaţie şi va fi un compromis între timpul de acces şi spaţiul necesar. Un bun compromis este şi folosirea unui index rar cu o intrare index pentru fiecare bloc care poate fi citit odată. Motivul este că odată blocul adus în memoria principală, timpul de căutare în cadrul blocului este mult mai scăzut. Indexul rar va localiza practic blocul în care trebuie căutat, căutarea în cadrul blocului urmând să aibă loc în memoria principală a calculatorului.

Indexuri pe mai multe niveluri. Chiar dacă se folosesc indexuri rare, există situaţii în care tabela index este prea mare pentru o prelucrare eficientă. Dacă numărul de înregistrări din fişier este prea mare, tabela index însăşi nu poate să fie păstrată în memoria principală, caz în care indexurii sunt păstraţi ca un fişier secvenţial pe disc, pentru a parcurge întregul fişier cu indexuri fiind necesare citiri de blocuri disc. În cadrul unui bloc se poate folosi căutarea binară pentru a localiza o intrare, dar căutarea în tot fişierul index are un cost ridicat. Pentru un index care ocupă *b* blocuri, căutarea binară necesită [*log2(b)*] citiri de blocuri. De notat că în cazul în care lungimea indexului depăşeşte un bloc, căutarea binară nu poate fi realizată, fiind necesară căutarea secvenţială care va face *b* citiri de bloc.

Pentru a rezolva problema, indexul va fi tratat ca orice fişier secvenţial şi se va construi un index rar pentru indexul primar. Pentru a regăsi o înregistrare se face o căutare în indexul exterior, după care urmărind pointerul găsit se continuă cautarea în indexul interior care va regăsi blocul de date ce conţine înregistrarea dorită. Procesul de creare a unui nou nivel de indexuri se poate continua dacă numărul de înregistrări este extrem de mare. Fiecare index de căutare poate corespunde la o unitate fizică de stocare; pot fi indexuri la pistă, cilindri, disc.



Indiferent de tipul indexului folosit, acesta trebuie actualizat dacă se realizează o inserare sau o ştergere de înregistrare. Modul de actualizare în cazul indexurilor pe un singur nivel este:

- La ştergere: pentru a şterge o înregistrare, se verifică dacă aceasta este singura înregistrare cu o valoare particulară a cheii de căutare. Dacă da, se şterge înregistrarea şi

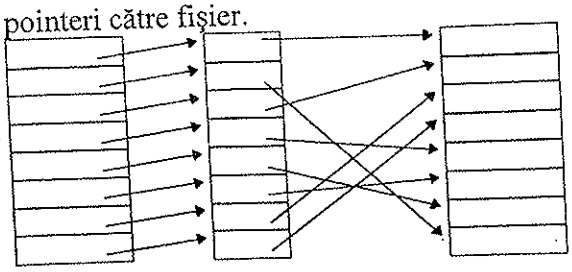
intrarea din tabelul index. Pentru index dens, ştergerea unei valori este identică cu ştergerea unei înregistrări din fişier. Pentru un index rar, ştergerea presupune înlocuirea intrării din index cu următoarea valoare a cheii de căutare din fişier. Dacă următoarea valoare are deja o intrare index, intrarea este ştearsă şi nu înlocuită

- La inserare: pentru a insera o nouă inregistrare, se caută dacă valoarea cheii apare în inregistrare. Dacă indexul este dens şi valoarea nu apare, aceasta este inserată ca o nouă intrare. Dacă indexul este rar şi păstrează o intrare pentru fiecare bloc, nu sunt necesare modificări dacă noua înregistrare poate fi adăugată la un bloc existent, altfel la adăugarea unui nou bloc, o nouă intrare se inserează în index.

Pentru indexuri pe mai multe niveluri pentru inserare şi ştergere se începe cu nivelul cel scăzut, cel interior, continuând, dacă este necesar cu nivelurile exterioare.

Index secundar. Un index secundar, pentru o cheie candidat, arată ca un index primar dens cu excepţia faptului că înregistrările care indică valori succesive nu sunt păstrate în index secvenţial. În general, indexurii secundari pot avea o structură diferită faţă de cei primari.

Pentru o cheie de căutare a indexului secundar care nu este cheie candidat se observă că dacă, pentru o valoare a cheii de căutare există mai multe înregistrări, în fişier nu există nici o garanţie că acestea vor fi grupate, ele putând fi oriunde în cadrul fişierului. În acest caz, indexul secundar trebuie să conţină pointer către toate înregistrările. Trebuie folosit un nivel suplimentar de indirectare pentru a putea implementa indexul secundar, deoarece pointerii din tabela index nu indică direct către fişier, fiecare intrare indică spre un grup de pointeri către fişier.



O parcurgere secvenţială în ordinea indexului primar este eficientă, deoarece înregistrările în fişier sunt păstrate fizic în aceeaşi ordine ca şi ordinea din index. Cum nu putem păstra fizic un fişier ordonat şi după cheia de căutare a indexului primar şi după cea a indexului secundar, rezultă că, în principiu, parcurgerea fişierului secvenţial după o cheie secundară de ordonare presupune citirea unui nou bloc pentru fiecare nouă înregistrare, rezultând o căutare greoaie.

Dacă un index secundar ar păstra doar o parte din valorile cheii de cautare, nu se pot regăsi, în general, valorile intermediare decât printr-o căutare în tot fişierul. Din acest motiv indexurii secundari vor fi doar indexuri denşi, cu o intrare index pentru fiecare valoare a cheii de cautare.

Concluzii

Arhitectura unui sistem baze de date este puternic influenţată de sistemul de calcul pe care este prelucrată informaţia. Aspecte ale arhitecturii calculatorului, cum ar fi conectarea în reţea, paralelismul sau distribuţia, sunt direct reflectate în arhitectura sistemului de gestine a bazei de date. Comparativ cu arhitecturile convenţionale, prelucrarea paralelă permite ca operaţiile cu bazele de date să fie mai rapide, cu un răspuns mai bun pentru tranzacţii, precum şi un număr de tranzacţii mai mare pe unitatea de timp; interogările pot fi procesate astfel încât să exploateze la maxim paralelismul oferit de maşină.

Bibliografie

- Foster Ian System programming in parallel logic languages - Addison Wesley 1995
- Fox GeoffreySolving problems on concurent processors - Prentice Hall 1988
- Culler David E. Modern parallel computer architecture - ACM Press 1996
- Gamma E, Helm R, Johnson R Design Patterns: Elements of reusable obiect-oriented software - Addison Wesley 1995