28. STRUCTURI DE DATE ŞI ACCES ÎN BAZE DE DATE

28.1 Stocarea datelor

O bază de date utilizează mai multe dispozitive de stocare a datelor. Aceste dispozitive, numite şi memorii, se deosebesc prin capacitatea lor de păstrare, viteza lor, modul de accesare a datelor (secvenţial sau direct) şi, în sfârşit, prin persistenţa lor. *Memoriile volatile* îşi pierd conţinutul când sistemul este întrerupt de la sursa de alimentare. *Memoriile nevolatile*, cum sunt discurile sau benzile magnetice îşi păstrează conţinutul chiar şi când sunt decuplate de la sursa de curent electric.

Dispozitive de stocare a datelor

În general, cu cât o memorie e mai rapidă cu atât ea este mai scumpă şi, prin urmare, cu atât capacitatea ei de stocare este mai redusă. Memoriile utilizate de un sistem de gestiune a bazelor de date (SGBD) constituie o ierarhie, figura 28.1, care porneşte de la cea mai mică, dar mai eficace, la memoria mai voluminoasă, dar mai lentă:

- memoria cache *este utilizată de procesor pentru stocarea datelor şi instrucţiunilor;*
- memoria principală *constituie spaţiul de lucru al maşinii; datele sau programele sunt încărcate în memoria principală, unde este posibilă tratarea lor de către procesor;*
- discurile magnetice *constituie principalul periferic de tip memorie; ele oferă o capacitate mare de stocare şi permit un acces relativ eficient la citire şi scriere;*
- benzile magnetice *sunt dispozitive ieftine, dar viteza joasă de lucru face ca să fie folosite pentru fişiere de salvare.*

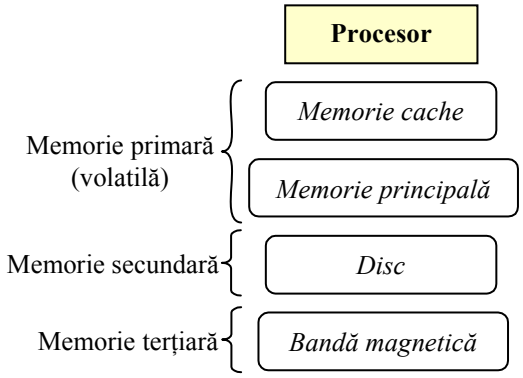


Figura 28.1 *Ierarhia de memorii*

Memoria primară, care constă din memoria cache şi memoria principală, asigură un acces foarte rapid la date. Actualmente, costul unui volum de memorie principală este de 100 ori mai mare decât acelaşi volum de memorie pe disc, iar banda magnetică este şi mai ieftină. Dispozitivele de stocare lentă, cum sunt discurile, joacă un rol important în bazele de date, deoarece volumul de date este, de obicei, foarte mare.

Există şi alt raţionament de păstrare a datelor în memoria secundară sau terţiară. În sistemele cu 32 de octeţi adresare, numai 2^{32} octeţi pot fi referiţi direct în memoria principală, or volumul de date este mult mai mare.

O bază de date este aproape întotdeauna păstrată pe disc şi din considerente de persistenţă. Cu toate acestea, datele recuperate sunt plasate în memoria principală pentru a fi prelucrate. Pornind de la această realitate, unde un mic fragment din baza de date poate sta în memoria centrală, SGBD-ul trebuie, deci, să efectueze, în permanenţă, transferuri de date între memoria principală şi memoria secundară. Costul acestor transferuri influenţează direct performanţa sistemului.

Funcţionarea discului magnetic

Un disc este o suprafaţă circulară, magnetizată, capabilă să înregistreze date. Suprafaţa magnetizată poate fi situată pe o singură parte (dacă discul are o singură faţă) sau pe două părţi ale discului (dacă discul are faţă dublă).

Discurile sunt divizate în *sectoare*, un sector constituind cea mai mică suprafaţă de adresare. Cu alte cuvinte, se pot citi sau scrie zone noi care încep pe un sector şi acoperă un număr întreg de sectoare. Mărimea unui sector este, de obicei, de 512 octeţi.

Dispozitivul. În mod obişnuit, bazele de date sunt stocate pe disc şi datele sunt transferate de pe disc în memoria principală în măsura necesităţilor. Pentru a limita costul dispozitivului şi mări capacitatea de stocare, mai multe discuri sunt montate pe o axă şi formează un pachet de discuri. În lucru, axa şi discurile sunt antrenate într-o mişcare de rotaţie cu o viteză mare, figura 28.2.

Cea mai mică unitate de date stocată pe un disc este un bit care poate avea valoarea 0 sau 1. Biţii sunt grupaţi câte 8 pentru a forma octeţi, iar o mulţime de octeţi formează, pe suprafaţa unui disc, o cunună circulară numită *pistă*. Mulţimea tuturor pistelor cu acelaşi diametru se numeşte *cilindru.*

Există o mulţime de capete de lectură/scriere situate la sfârşit pe un braţ. Capetele se mişcă în grup, astfel că acestea pot fi poziţionate asupra tuturor pistelor ce constituie un cilindru. Prin urmare, toate datele unui cilindru pot fi accesate fără a se produce vreo mişcare a braţului, care este o operaţie mai lentă. Noţiunea de cilindru corespunde, deci, tuturor datelor disponibile fără a avea nevoie de o deplasare a capetelor de lectură.

Capul de lectură nu este antrenat în mişcarea de rotaţie. El se deplasează pe un plan fixat care îi permite de a se apropia sau de a se îndepărta de axa de rotaţie a discului şi de a accesa o pistă.

Sunt tot atâtea capete de lectură câte discuri sunt (de două ori mai mult, dacă discurile sunt cu feţe duble) şi toate capetele sunt poziţionate respectiv pe planul lor de deplasare. În orice moment, pistele accesibile sunt cele de pe un cilindru, ceea ce constituie o constrângere de care trebuie să se ţină cont când se doreşte optimizarea amplasării datelor.

În fine, ultimul element al dispozitivului este *controlorul* care serveşte drept interfaţă între pachetul de discuri şi sistem. Controlorul primeşte, de la sistem, cererile de citire sau scriere şi le transformă în mişcări corespunzătoare ale braţului cu capete de lectură.

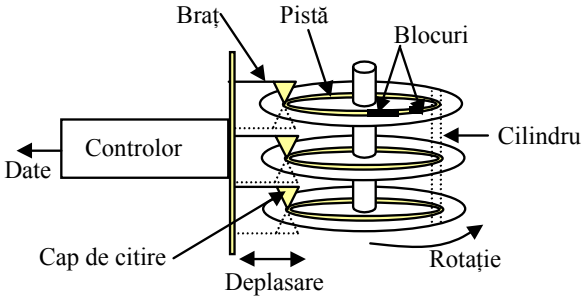


Figura 28.2 *Un disc magnetic*

Fiecare pistă este divizată în arce numite *sectoare*, mărimea cărora este o caracteristică a discului. Sectoarele sunt diviziuni fizice şi, de aceea, au dimensiuni fixe. Ele sunt numerotate astfel încât sectoarele consecutive au numere consecutive. Numărul minim de octeţi citiţi de capul de lectură este definit de mărimea sectorului, în general, de 512 octeţi.

Fiecare pistă este, deci, divizată în *blocuri* (sau *pagini*), care constituie unitatea de schimb dintre disc şi memoria principală. Blocul este o diviziune logică şi, prin urmare, dimensiunea lui este o mărime variabilă. Dimensiunea unui bloc poate fi setată când discul se iniţializează. Atunci se fixează unitatea de intrare/ieşire (adică blocul) mai mare (sau egală) decât a unui sector. Lungimea unui bloc este, de obicei, egală cu un multiplu al lungimii unui sector. Astfel, se obţin *blocuri* a căror lungime (tipică) este de 512 octeţi (un sector), 1024 de octeţi (2 sectoare) sau 4096 de octeţi (8 sectoare).

Toate lecturile sau toate scrierile pe disc se efectuează prin blocuri. Astfel, chiar dacă o lectură nu se referă decât la 4 octeţi, întreg blocul este transmis în memoria centrală. Această caracteristică este fundamentală pentru organizarea datelor pe disc. Unul din obiectivele SGBD–ului este de a face totul, încât, atunci când este necesară citirea unui bloc de 4096 de octeţi pentru a accesa un număr de 4 octeţi (4092 de octeţi constituind restul blocului), aceasta să se petreacă într-un timp scurt, adică blocul, deja, trebuie să se găsească încărcat în memoria centrală. Această motivaţie stă la baza mecanismului de *regrupare*, în special, la baza structurilor de date indexate şi dispersate.

Accesul la date. Un disc este o memorie cu acces direct. Spre deosebire de o bandă magnetică, de exemplu, este posibil de accesat o unitate de date situată în orice loc pe disc, fără a avea de parcurs secvenţial tot suportul. Accesul direct se bazează pe o adresă dată fiecărui bloc în momentul iniţializării discului de către sistemul de exploatare. În general, această adresă este compusă din trei elemente:

- numărul discului în pachet sau numărul suprafeţei, dacă discurile sunt cu suprafaţă dublă;
- numărul pistei;
- numărul blocului pe pistă.

Citirea unui bloc, dată fiind adresa lui, se petrece în trei etape:

- poziţionarea capului de lectură pe pista care conţine blocul;
- rotirea discului şi aşteptarea până în momentul când blocul va trece sub capul de lectură – capetele sunt fixate, discul se roteşte;

- transferul blocului.

Astfel, există trei factori care afectează direct viteza cu care datele sunt transferate între disc şi memoria principală:

- timpul de căutare;
- timpul de rotaţie sau aşteptare;
- timpul de transfer.

Timpul de căutare (sau *poziţionare*) este timpul necesar pentru a mişca braţul cu capetele de lectură/scriere din poziţia curentă, până în poziţia cilindrului adresat. *Timpul de rotaţie* sau al *stării latente*) (este timpul necesar discului pentru a se învârti până ce capul va fi situat deasupra sectorului de scriere sau citire. *Timpul de transfer* este timpul necesar pentru citirea sau scrierea datelor. Acest timp depinde de numărul de octeţi transferaţi. Timpul de transfer este neglijabil pentru un bloc, dar poate deveni important când trebuie citite mii de blocuri. Mecanismul scrierii este asemănător celui de citire, dar poate să consume puţin mai mult timp, în cazul când controlorul verifică dacă scrierea se efectuează corect.

Optimizarea accesului la date

Acum când se cunoaşte cum funcţionează un disc, este destul de evident că, pentru acelaşi volum de date, timpul de lectură poate varia considerabil, în funcţie de factori precum amplasarea datelor pe disc, ordinea instrucţiunilor de lectură/scriere sau prezenţa datelor într-o memorie *cache*.

Toate tehnicile care permit reducerea timpului consumat pentru accesul la disc sunt intensiv utilizate de SGBD-uri, a căror performanţă, în mare parte, este condiţionată de eficienţa acestor accesări. În această secţiune, vor fi descrise principalele tehnici de optimizare, realizate într-o arhitectură simplă, constituită dintr-un singur disc şi un singur procesor.

Regruparea datelor. Din cele relatate mai sus, reiese că timpul de executare a operaţiilor asupra datelor în baza de date este afectat semnificativ de faptul cum sunt stocate datele pe disc. Timpul de transfer al blocurilor, din sau spre disc, de obicei, domină timpul consumat de operaţiile bazei de date. Pentru a minimiza acest timp, este necesar de a alege o strategie de amplasare a datelor pe disc, ţinând cont atât de geometria discului, cât şi de mecanica discului.

De exemplu, fie SGBD-ul trebuie să citească 5 secvenţe de caractere a câte 1000 octeţi fiecare. Dacă un bloc are lungimea egală cu 4096 octeţi, două blocuri sunt suficiente pentru a păstra aceste secvenţe de caractere. În figura 28.3, sunt prezentate două tipuri de organizare a unui disc. În primul tip, fiecare secvenţă este plasată într-un bloc aparte şi blocurile sunt repartizate în mod aleatoriu pe pistele discului. În cel de-al doilea tip de organizare, secvenţele sunt stocate în două blocuri consecutive pe aceeaşi pistă a discului.

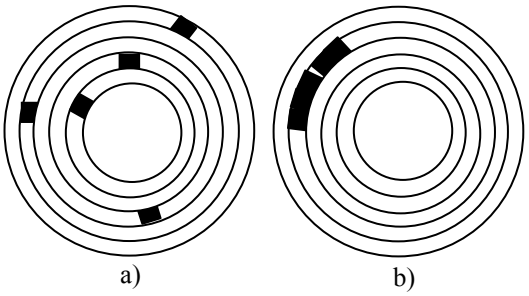


Figura 28.3 *a) - organizare rea; b) - organizare bună*

Performanţele obţinute reprezintă un raport de 1 la 5. Timpul minimal se obţine, apelând la gruparea datelor. Gruparea se bazează pe plasarea în acelaşi bloc a datelor care au şanse mari de a fi citite împreună. Criteriile folosite la determinarea grupurilor de date constituie bazele structurilor de date în memoria secundară.

În general, câştigul obţinut la transferarea a două date este cu atât mai impunător cu cât datele sunt mai „apropiate" pe disc. Această „apropiere" are mai multe valenţe.

Evident, apropierea maximal posibilă este obţinută când două unităţi de date se găsesc în acelaşi bloc – ele vor fi citite împreună. Două blocuri sunt foarte "apropiate", dacă sunt stocate consecutiv pe o pistă. Cu viteza discului, două blocuri vor fi citite sau scrise, dacă se găsesc pe aceeaşi pistă, sub acelaşi cap activ de citire. Toate datele de pe o pistă (pe discurile proiectate azi) se citesc şi se scriu într-o revoluţie a discului, fără a se deplasa capul de citire.

După ce o pistă este citită sau scrisă, alt cap al discului devine activ şi altă pistă a aceluiaşi cilindru este citită sau scrisă. Acest proces continuă până când toate pistele de pe cilindrul curent sunt citite sau scrise, şi apoi braţul în întregime se deplasează (înainte sau înapoi) spre cilindrul adiacent. Astfel arată noţiunea "apropierea" blocurilor care într-un sens exprimă noţiunile de bloc "următor" şi "precedent".

Prin urmare, în descreşterea ordinii, două unităţi de date pe disc trebuie să fie în acelaşi bloc, pe aceeaşi pistă, pe acelaşi cilindru sau pe cilindri adiacenţi.

Exploatarea noţiunii de "apropiere" şi aranjare a datelor astfel ca ele să fie scrise sau citite secvenţial este foarte importantă pentru reducerea timpului consumat la accesarea discului. Ea permite *lecturi secvenţiale* care, după cum reiese din exemplul anterior, sunt mult mai performante decât lecturile aleatorii, deoarece ele evită deplasarea capului de lectură şi minimizează timpul de căutare şi stare latentă.

SGBD-urile de azi optimizează distanţa dintre date în momentul stocării lor pe disc. De exemplu, o relaţie este stocată pe aceeaşi pistă sau, în caz că ea ocupă, pe mai multe piste, pe pistele aceluiaşi cilindru, pentru a putea realiza o parcurgere secvenţială eficientă.

Pentru ca SGBD-ul să poată efectua aceste optimizări, administratorul, când defineşte schema fizică a bazei de date, trebuie să rezerve un spaţiu important pe disc, pe care el (SGBD-ul) îl va gestiona. Dacă SGBD-ul se mulţumeşte de a cere sistemului de exploatare spaţiu pe disc atunci când are nevoie, stocarea fizică obţinută poate fi extrem de fragmentată.

Reordonarea accesului la date. Bazele de date sunt sisteme ce lucrează în regim de multiutilizator. Astfel, chiar dacă teoretic se admite că un fişier este stocat continuu pe aceeaşi pistă, citirea secvenţială a acestui fişier poate fi segmentată de interogările formulate de alţi utilizatori care accesează baza de date în mod concurenţial. Şi atunci, eficienţa organizării optimale a datelor pe disc, bineînţeles, scade. Apare, deci, problema ridicării eficienţei accesului la date în situaţia satisfacerii simultane a mai multor utilizatori, cererile cărora trebuie administrate concomitent.

De exemplu, dacă un utilizator U_1 cere lectura fişierului F_1, în timp ce utilizatorul U_2 cere lectura fişierului F_2, sistemul, probabil, va alterna lectura blocurilor din aceste fişiere. Chiar şi în cazul când ambele fişiere vor fi stocate secvenţial, deplasarea capului de citire va minimiza întru câtva avantajele acestei organizări.

SGBD-ul poate reduce acest dezavantaj prin conservarea temporară a operaţiilor de citire/scriere într-o zonă tampon (*cache*) şi reorganizarea (*secvenţială*) ordinii de acces.

Astfel, fie mai mulţi utilizatori formulează instrucţiuni de lectură şi scriere asupra fişierelor stocate în baza de date. Evident că multe din aceste cereri se pot suprapune când trec prin controlor. Pentru a evita accesul aleatoriu care survine în urma acestei suprapuneri, cererile de acces sunt stocate temporar într-un *tampon*. Sistemul le triază, atunci, pe piste, apoi pe blocuri în cadrul fiecărei piste şi apoi transmite lista ordonată controlorului de disc.

O metodă de sistematizare a acestei strategii este aşa-zisul „ascensor”. Adică se presupune că capul de lectură se deplasează de la marginea suprafeţei discului spre axa de rotaţie, apoi revine de la axă spre bord. Deplasarea se efectuează pistă după pistă şi, la fiecare pistă, sistemul transmite controlorului cererea de citire/scriere corespunzătoare pistei curente.

Acest algoritm reduce la maximum timpul de deplasare a capului, deoarece căutarea se realizează sistematic pe pistele adiacente. El este, în particular, eficient pentru sistemele cu multe cereri, fiecare antrenând mai multe blocuri de date. Dar, bineînţeles că pot fi şi unele efecte nedorite în cazul unor cereri mari consumatoare de date. Procesele care cer blocuri de pe pista 1, când capul, deja, trece pe pista 2, trebuie să aştepte un timp considerabil pentru a vedea cererea dată satisfăcută.

Utilizarea memoriei tampon. Utilizarea *memoriei tampon* sau a *buferului* pentru optimizarea accesului la date este pe larg practicată în toate SGBD-urile. O memorie tampon este o mulţime de blocuri în memoria principală, care sunt copii ale unor blocuri de pe disc. Când sistemul cere accesul la un bloc, mai întâi se inspectează memoria tampon. Dacă blocul se găseşte în bufer, este evitată o citire a dispozitivului secundar de stocare a datelor. Dacă nu, se efectuează lectura şi se stochează blocurile în memoria tampon.

Ideea este, deci, de a păstra în memoria principală o copie a unei părţi cât se poate de mare a bazei de date, chiar dacă o parte din blocurile plasate în memoria tampon nu este utilă la moment. O latură importantă a administrării unei baze de date o constituie specificarea unei părţi de memorie principală în calitate de memorie tampon disponibilă, în permanenţă, SGBD-ului. În plus, această memorie este utilă şi prin faptul că

se poate conserva o parte semnificativă a bazei, câştigând, astfel, în performanţă.

Dacă în memoria tampon rămâne loc neutilizat, se poate recurge la *lecturi în avans*. Tehnica lecturii în avans este utilizată frecvent de SGBD-uri pentru a efectua operaţia de joncţiune a două relaţii. Această tehnică se foloseşte pe larg şi în lucrul cu fişierele indexate.

28.2 Concepte generale de organizare a fişierelor

O bază de date este concepută ca o mulţime de fişiere stocate pe un suport persistent. Înainte de a vorbi despre conceptele fundamentale ale structurilor de fişiere în baze de date, trebuie menţionat, pentru a elimina confuzia, că aceste două concepte (*fişier* şi *fişier în baza de date*) sunt două noţiuni distincte. În primul rând, fişierele administrate de un SGBD sunt un pic mai structurate. Dar, de fapt, există trei particularităţi esenţiale care caracterizează fişierele din baze de date. Aceste particularităţi sunt:

- viziuni diferite ale aceloraşi date;
- independenţa date-prelucrare;
- redundanţa (gestionabilă) datelor.

De aceea, fără excepţie, SGBD-rile au propriile lor module de gestiune a fişierelor şi a memoriei cache.

Câmpuri şi înregistrări

Când se construiesc structuri de fişiere, datelor li se dă un aspect organizaţional şi de persistenţă în acelaşi timp, adică o aplicaţie creează, în memoria centrală, date şi le salvează într-un fişier şi cel puţin o altă aplicaţie poate citi şi rescrie aceste date în memorie, recuperându-le din fişier. Astfel, obiectivul este organizarea datelor într-o structură comprehensibilă de către fiinţele umane. Or, pentru un sistem de operare, fişierul este o succesiune de octeţi repartizaţi în unu sau mai multe blocuri.

Prin urmare, se disting nivelul logic şi nivelul fizic de concepere a fişierului:

- structura logică este forma fişierului în care este văzută şi manipulată de aplicaţii; cum datele sunt organizate în aplicaţii (în general, organizate în acord cu obiectele pe care aplicaţia le manipulează); de exemplu, un fişier poate fi văzut ca o colecţie de entităţi ordonate de o cheie sau o structură ierarhică construită din entităţi principale şi entităţi subordonate;
- structura fizică este o viziune care reflectă reprezentarea şi organizarea datelor în mediul de stocare (sectoare, blocuri,…); este forma în care datele sunt stocate, organizate pe dispozitiv, ţinând cont de unitatea de bază pe care dispozitivul o poate manipula (o comandă de citire sau de scriere o manipulează în întregime).

La nivel fizic, fişierele sunt constituite din *înregistrări* (*records*) care reprezintă, din punct de vedere fizic, entităţile de lucru ale SGBD-ului. Conform modelului logic al SGBD-ului, aceste entităţi pot fi tupluri într-o relaţie sau obiecte. În cele ce urmează, este tratat primul caz, adică este considerat modelul relaţional de date.

Câmpuri cu lungime fixă şi lungime variabilă. Un tuplu al unei relaţii este constituit dintr-o listă de componente (atribute), fiecare având un tip. Acestui tuplu îi corespunde, la nivel fizic, o înregistrare, constituită din *câmpuri* (*field*). Fiecare tip de atribut determină lungimea câmpului necesar pentru stocarea unei instanţe a câmpului.

Lungimea unui tuplu este egală cu suma lungimilor câmpurilor care reprezintă atributele sale. În practică, lucrurile sunt ceva mai complicate. Câmpurile (şi, deci, înregistrările) pot fi de lungimi variabile. Dacă lungimea unei înregistrări de mărime variabilă creşte pe parcursul actualizării, trebuie să se găsească un spaţiu liber. De asemenea, apare şi problema reprezentării valorii *NULL*.

Tipurile de date pot fi divizate în două categorii: tipuri care pot fi reprezentate printr-un câmp de lungime fixă şi tipuri care au lungime variabilă. De exemplu, standardul SQL2 propune, printre altele, două tipuri de date pentru secvenţe de caractere: *CHAR* şi *VARCHAR*.

Tipul *CHAR* indică o secvenţă de lungime fixă. Astfel, *CHAR(5)* defineşte un câmp stocat pe *5* octeţi. Atunci apare întrebarea: cum se reprezintă valoarea *'Joc'*? Există două soluţii:

- ultimele două caractere se completează cu spaţii;
- ultimele două caractere se completează cu un caracter convenţional.

Convenţia adoptată influenţează compararea, fiindcă, într-un caz, se stochează *'Joc'* (cu 2 spaţii), iar în alt caz - *'Joc'* fără caractere de terminaţie. Dacă se utilizează tipul *CHAR,* este important să se studieze convenţia adoptată de SGBD-ul concret.

Tipul *VARCHAR(n)* permite stocarea secvenţelor de lungime variabilă. Există (cel puţin) două posibilităţi:

- câmpul este de lungimea n+1, primul octet conţine un întreg care indică lungimea exactă a secvenţei; dacă se stochează 'Joc' într-un VARCHAR(10), se obţine atunci '3Joc', unde primul octet păstrează un 3 în formă binară, urmat de trei octeţi cu caracterele 'J', 'o' şi 'c', iar următorii 7 octeţi rămân neutilizaţi;
- câmpul are lungimea l+1, unde l<n, aici nu sunt octeţii neutilizaţi, ceea ce permite economisirea spaţiului.

De notat că reprezentarea unui întreg de un octet limitează lungimea maximală a unui *VARCHAR* cu *255*. O variantă care poate depăşi această limită constă în înlocuirea octetului iniţial care indică lungimea cu un caracter de terminare a secvenţei (fie un *C*).

Antetul înregistrării. La fel cum se prefixează un câmp de lungime variabilă cu lungimea sa utilă, în antetul înregistrării se stochează unele date complementare. Aceste date pot fi:

- lungimea înregistrării, dacă lungimea este variabilă;
- un pointer spre schema relaţiei, pentru a şti care este tipul înregistrării;
- data ultimei actualizări etc.

Acest antet, de asemenea, se poate utiliza pentru indicarea valorilor *NULL*. Absenţa valorii pentru unul dintre atribute este o problemă delicată. Dacă nu se stochează nimic, există riscul să fie perturbată decuparea unui câmp, în timp ce, dacă se stochează o valoare convenţională, se pierde spaţiu. O soluţie posibilă este o mască de biţi, câte unul pentru fiecare câmp al înregistrării. Bitul ia valoarea *0*, dacă câmpul este *NULL* şi *1* dacă nu. Această mască poate fi stocată în *antetul înregistrării* şi, deci, nu este

necesar spaţiu pentru valoarea *NULL*. Totul rămâne în decodarea corectă a secvenţei de octeţi.

De exemplu, fie o schemă relaţională cu atributele *ID* de tipul *INTEGER*, *Nume_Prenume* de tipul *VARCHAR(50)* şi *An_nastere* de tipul *INTEGER* şi fie în această relaţie este înregistrarea *(202315, 'Odobescu, Alexandru', NULL),* adică anul naşterii este necunoscut.

Identificatorul *ID* este stocat pe *4* octeţi şi numele şi prenumele pe *8* octeţi, dintre care un octet este rezervat pentru lungimea câmpului. Antetul înregistrării conţine un pointer spre schema relaţiei, lungimea sa totală *(4+8)*, şi o mască de biţi *110* care indică faptul că al treilea câmp are valoarea *NULL*. Figura 28.4 reprezintă această înregistrare. De notat, că citind antetul, se poate calcula adresa înregistrării următoare.

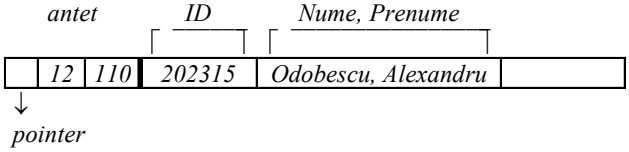


Figura 28.4 *O înregistrare cu antet*

Blocuri

Unitatea de transfer de date între fişierul memorat pe mediu şi memoria internă este *blocul*. Lungimea unui bloc este, de obicei, o putere a lui *2* cuprinsă între 2^9 şi 2^{12} octeţi. Un bloc în sistemul Oracle, de exemplu, ocupă 4096 sau 8092 octeţi. Fiecărui bloc i se asociază o adresă.

Structura blocului. Stocarea înregistrărilor într-un fişier trebuie să ţină cont de divizarea în blocuri a acestui fişier. În general, într-un bloc se pot plasa mai multe înregistrări, dar se evită ca o înregistrare să se împartă între două blocuri. Numărul maximal de înregistrări de lungimea *L* memorate într-un un bloc de lungimea *B* este $\lfloor B/L \rfloor$, unde notaţia $\lfloor x \rfloor$ desemnează cel mai mare întreg inferior lui *x*.

De exemplu, fie un fişier memorează o relaţie a cărei schemă nu conţine atribute de lungime variabilă, adică nu utilizează tipurile *VARCHAR* sau *BIT VARYING*. Înregistrările au, deci, o lungime egală cu suma lungimilor tuturor câmpurilor. Fie că această lungime este de aproximativ 84 octeţi, iar lungimea blocului este de 4096 octeţi. În afară de aceasta, fie că fiecare bloc conţine un antet de 100 octeţi pentru a stoca datele despre spaţiul liber disponibil în bloc, înlănţuirea cu alte blocuri etc. Atunci, se pot memora $\lfloor (4096-100)/84 \rfloor = 47$ înregistrări într-un bloc. De notat că în fiecare bloc rămân $3996-(47*84)=48$ octeţi neutilizaţi.

Transferul, în memorie, al înregistrării 563 a acestui fişier este simplu de efectuat: se determină în ce bloc se găseşte ea (fie $\lfloor 563/47 \rfloor + 1 = 12$), se încarcă al *12*-lea bloc în memoria centrală şi din acest bloc se extrage înregistrarea. Prima înregistrare a blocului este $11*47+1=518$, iar ultima înregistrare este $12*47=564$. Înregistrarea *563* este, deci, penultima în blocul cu numărul intern *46,* figura 28.5.

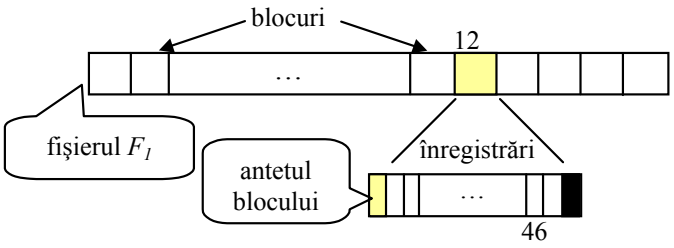


Figura 28.5 *Un fişier cu blocuri şi înregistrări*

Calculul de mai sus arată cum se poate localiza fizic o înregistrare: prin fişierul ei, apoi prin blocul ei, apoi prin poziţia ei în bloc. Dacă fişierul e nominalizat cu F_1, adresa înregistrării poate fi reprezentată prin *'F1.12.46'*.

Există multe alte metode de adresări. Dezavantajul utilizării adreselor fizice, de exemplu, este că nu se poate schimba locul unei înregistrări fără a genera adresări invalide ale pointerilor asupra acestei înregistrări (de exemplu, în indecşi).

Pentru a permite deplasarea înregistrărilor, se poate forma o *adresă logică,* ce ar identifica o înregistrare independent de locaţia ei. Un tabel de corespondenţă permite administrarea asocierii între adresa fizică şi adresa logică, figura 28.6. Acest mecanism face ca organizarea şi reorganizarea bazei de date să fie o procedură flexibilă. Acum e suficientă referirea unei înregistrări prin adresa sa logică, iar modificarea adresei fizice în tabel să se efectueze când are loc o deplasare. În schimb, această metodă necesită un cost suplimentar pentru că, sistematic, trebuie examinat tabelul de corespondenţă pentru a accesa datele.

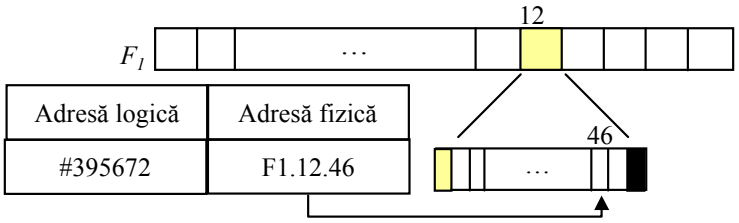


Figura 28.6 *O adresare indirectă*

O soluţie intermediară este îmbinarea adresărilor logică şi fizică. Pentru a localiza o înregistrare se indică adresa fizică a blocului, apoi în blocul propriu-zis se administrează un tabel care dă localizarea în bloc sau eventual în alt bloc.

Fie, din nou, înregistrarea *F1.12.46*. Aici *F1.12* indică blocul *12* al fişierului F_1. Se presupune că *46* este identificatorul logic al înregistrării administrate în interiorul blocului. Figura 28.7 prezintă această adresare în două niveluri: în blocul *F1.12*, înregistrarea *46* corespunde unei amplasări în acelaşi bloc, pe când înregistrarea *57* este plasată în alt bloc.

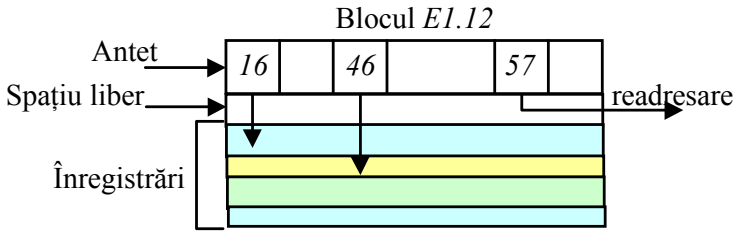


Figura 28.7 *O îmbinare a adresărilor logică şi fizică*

Trebuie menţionat că spaţiul liber din bloc este situat între antetul blocului şi înregistrări. El permite mărirea simultană a acestor două elemente în cazul unei inserări, de exemplu, fără efectuarea reorganizării interne a blocului. Acest mod de identificare oferă multe avantaje şi permite reorganizarea suplimentară a spaţiului intern al unui bloc.

Blocuri cu înregistrări de lungime variabilă. O relaţie care e definită pe atribute de tipul *VARCHAR* sau *BIT VARYING* este reprezentată prin înregistrări de lungime variabile. Când o înregistrare este inserată într-un fişier, lungimea se calculează nu după *tipul* de atribute, ci după numărul real de octeţi necesari reprezentării *valorilor* atributelor. Această mărime trebuie stocată la începutul înregistrării curente, pentru ca SGBD-ul să poată determina începutul înregistrării următoare.

Se poate întâmpla că înregistrarea este actualizată, adică este modificată valoarea unui atribut sau unui atribut iniţial i-a fost dată valoarea *NULL*. În acest caz, se poate întâmpla ca locul rezervat iniţial să fie insuficient pentru noile date şi, deci, înregistrarea urmează să fie memorată în alt loc al aceluiaşi fişier. Astfel, e necesară crearea unei legături între înregistrarea anterioară şi curentă, memorată în alt bloc.

La locul deplasării înregistrării întregi, unele SGBD-uri aplică tehnica de fragmentare a înregistrării şi de memorare în alt bloc a unui fragment, organizând, bineînţeles, o înlănţuire la nivel de înregistrare. Deplasarea (sau fragmentarea) înregistrărilor de lungime variabilă, evident, influenţează eficienţa de accesare a datelor. În afară de aceasta, înregistrările de lungime variabilă sunt ceva mai complicat de administrat decât cele de lungime fixă. În schimb, un fişier care conţine înregistrări de lungime variabilă utilizează, frecvent, mai puţin spaţiu decât i se atribuie.

Chei

Dacă fişierul stocat este organizat ca un grup de înregistrări secvenţiale, trebuie să fie forme de recuperare a unei înregistrări specifice, executând un număr minimal de accesări. Astfel, înregistrările trebuie să fie ordonate pentru acelaşi criteriu de căutare. Pentru aceasta, fiecărei înregistrări i se asociază o *cheie* bazată pe conţinutul său pentru a fi utilizată pentru identificarea univocă. În acest context, cheia este un instrument conceptual important pentru menţinerea consistenţei datelor şi pentru asigurarea procesului de restabilire. De exemplu, cheia în fişiere este mai bine de folosit împreună cu unele tehnici de căutare speciale (binară, pe blocuri etc…), în baza valorii ei, pentru a verifica înregistrările în mod secvenţial.

O *cheie primară* reprezintă unul sau mai multe atribute, care, univoc, identifică (una şi numai una) o înregistrare.

O *cheie secundară* nu identifică univoc o înregistrare şi poate fi utilizată pentru căutarea simultană a mai multor înregistrări cu aceeaşi valoare a cheii. Astfel, definiţia cheilor secundare trebuie să folosească unul sau mai multe câmpuri cu o semnificaţie pentru utilizator şi a căror date sunt, de obicei, folosite pentru căutarea datelor în fişier. Deci, este o cheie de ordonare pentru recuperarea unei mulţimi de date, organizate diferit de organizarea cheii primare.

Pe parcursul acestui capitol, se vor utiliza şi alte noţiuni legate de cheie. Urmează o explicare a acestor concepte:

O *cheie compusă* este o cheie formată din mai multe atribute sau câmpuri.

O *cheie externă* este o mulţime de atribute care constituie cheia primară a altui fişier. Mulţimea de valori a cheii externe constituie o submulţime a mulţimii de valori ale cheii primare.

O *cheie de acces* este cheia utilizată pentru identificarea (recuperarea) unei înregistrări. Aceasta reprezintă unul sau o mulţime de atribute pentru căutarea înregistrărilor în fişier.

Un *argument de căutare* este valoarea cheii de acces la înregistrare într-o operaţie de căutare.

O *cheie de ordonare* reprezintă o mulţime de atribute folosite la ordonarea înregistrărilor unui fişier.

O *cheie a unei înregistrări* este valoarea cheii primare pentru această înregistrare.

Accesul la datele stocate pe un suport periferic, spre deosebire de aplicaţiile care manipulează datele în memoria centrală, reprezintă una dintre particularităţile esenţiale ale SGBD-urilor. Aici apar probleme de îmbunătăţire a performanţei, deoarece timpul de citire a unei unităţi de date pe un disc este considerabil mai mare decât timpul de acces în memoria principală. Organizarea datelor pe un disc în fişiere, structurile de indexare şi algoritmii de căutare utilizaţi constituie aspecte foarte importante ale SGBD-urilor. Un sistem performant trebuie să utilizeze eficient tehnicile disponibile în scopul micşorării timpului de acces.

Organizarea fişierelor este o aranjare a unui tip de structură construită astfel încât să furnizeze un mediu eficient pentru stocarea şi manipularea datelor în memoria secundară, economisind spaţiul utilizat, şi mărind viteza de acces (minimizând timpul total de accesare şi transfer al blocurilor de date) la înregistrările din fişier. În organizarea fişierelor, se ţine cont de mai mulţi factori, cum sunt frecvenţa cu care se efectuează anumite operaţii, câmpurile implicate în operaţiile de căutare (cheile de căutare), dimensiunile înregistrărilor (fixe sau variabile).

Cele mai utilizate tipuri de organizare a fişierelor în bazele de date sunt: secvenţial, indexat secvenţial, indexat şi cu dispersie. Aceste tipuri de organizare a fişierelor sunt descrise în continuare.

28.3 Fişiere secvenţiale

Prin natura lor, fişierele secvenţiale sunt asociate dispozitivelor cu acces secvenţial, precum sunt, de exemplu, benzile magnetice. Ele se caracterizează prin prelucrarea în lot, mari volume de date, costuri minimale, spaţiu redus de păstrare, utilizare în benzile magnetice. Uneori, fişierele secvenţiale sunt stocate şi pe dispozitive cu acces direct, în

particular, când este necesară prelucrarea datelor, în mod secvenţial, cu o viteză mai înaltă.

Într-un mediu multiutilizator, cum este baza de date, trebuie ţinut cont că mai mulţi utilizatori pot partaja acelaşi dispozitiv de stocare şi accesul la o nouă înregistrare, de obicei, necesită o poziţionare nouă a capului de citire/scriere pe cilindrul care o conţine. În afară de aceasta, timpul de mişcare a capului de citire depinde de schema memoriei intermediare utilizate şi de faptul dacă sistemul realizează sau nu citirea anticipată.

Utilizarea acestui tip de organizare se recomandă pentru fişierele mici şi în cazul când nu sunt frecvent modificate. De exemplu, în fişierele pentru salvare, există puţine includeri de date şi, în general, ele sunt făcute în lot, periodic. Modificările şi suprimările sunt puţine sau absente. Consultările, evident, sporadice, pot fi făcute rapid, iar parcurgerea secvenţială este cea dezirabilă.

Fişiere secvenţiale ordonate

Fişierele secvenţiale ordonate reprezintă o structură de fişiere secvenţiale, unde înregistrările sunt clasificate de valorile câmpurilor care formează cheia de sortare (de obicei, cheia primară) şi stocate astfel încât ordinea logică coincide cu ordinea fizică. Fişierele secvenţiale sunt stocate pe disc în blocuri pe poziţii fizice continue pe pistele unuia şi aceluiaşi cilindru şi apoi pe pistele cilindrului adiacent. Deoarece înregistrările în fişierele secvenţiale sunt stocate în succesiune continuă, accesarea înregistrării *n* a fişierului presupune că cele (*n-1*) înregistrări (începând cu prima), de asemenea, sunt citite.

După cum fişierele sunt unice şi cheia de ordonare este unică (fiecare instanţă poate fi clasificată numai într-un singur mod). În cazul când fişierele secvenţiale nu posedă chei de organizare (ceea ce, în bazele de date, nu are loc), înregistrările sunt aranjate în serie, fiindcă, în general, fiecare înregistrare nouă se plasează la sfârşitul fişierului. În această situaţie, se adaugă un câmp suplimentar care conţine ordinea sau numerele de identificare conform cărora fişierul este ordonat şi acest câmp poate fi considerat cheie primară.

Organizarea secvenţială este mai perfectă decât organizarea serială a fişierelor (*heap file*), dar ele pierd din flexibilitate, deoarece nu se adaptează uşor la operaţiile de modificare. Fişierele secvenţiale sunt utile pentru clasificarea şi accesarea volumelor mari de date şi, din motive economice, în general, sunt stocate pe benzi magnetice.

Fişierele sunt ordonate fizic şi, pe parcursul perioadei de păstrare a datelor, această ordine (logică şi fizică) trebuie menţinută.

Fişierul secvenţial stocat pe bandă magnetică poate fi deschis ca un fişier de ieşire sau de intrare. Dar, dacă fişierul este stocat pe disc, poate fi deschis pentru intrare, ieşire şi modificare. Dacă se deschide un fişier secvenţial, capul de citire întotdeauna este fixat pe prima înregistrare a fişierului şi de multe ori (pe bandă, de exemplu) nu are cum să se întoarcă la înregistrarea citită anterior. Pentru aceasta, fişierul trebuie să fie închis pentru a fi rebobinată banda magnetică şi apoi deschis.

De obicei, fişierul secvenţial este simplu de închis, dar, pentru banda magnetică, operaţia respectivă presupune rebobinarea ei până la începutul fişierului. Această sarcină este o funcţie fizică realizată în mod automat.

Operaţia de acces la un fişier secvenţial poate să se producă sub două forme: cu cheia de acces diferită de cheia de ordonare şi cu cheia de acces egală cu cheia de ordonare.

Manipularea înregistrărilor în ordinea stocării este eficientă, deoarece ele sunt stocate fizic în ordinea în care sunt solicitate – secvenţa fizică este egală cu cea logică. *Accesul secvenţial* (constă în obţinerea înregistrării care urmează după ultima accesată, în secvenţa (de obicei, ascendentă) definită de cheia de ordonare. În acest mod, dacă se folosesc tehnicile de buferizare şi *caracteristicile blocului*, în majoritatea accesărilor, înregistrarea dezirabilă deja va fi în memorie, deoarece înregistrarea succesivă va fi în acelaşi bloc cu precedenta. Astfel, parcurgerea secvenţială după cheia de ordonare este simplă, întrucât fişierul se parcurge de la început la sfârşit. Însă, parcurgerea secvenţială după altă cheie va necesita o ordonare prealabilă într-un fişier auxiliar.

Avantajul principal al acestei organizări este facilitatea de realizare a operaţiei de parcurgere secvenţială, în afară de simplitatea implementării celorlalte operaţii. Cel mai mare dezavantaj este timpul de executare a operaţiilor de includere şi excludere a înregistrărilor, pentru că multe înregistrări sunt deplasate pentru păstrarea ordinii fizice şi logice.

Accesul aleatoriu, interogare, este caracterizat de identificarea înregistrării prin specificarea argumentului de căutare. Secvenţa de acces nu este legată neapărat de ordinea fizică a fişierului, având ca rezultat înregistrări care nu sunt stocate în formă continuă. În general, consultarea aleatorie a unei singure înregistrări după cheia de ordonare se realizează prin folosirea tehnicilor de căutare mai eficiente, precum cea binară. În căutarea binară, numărul maximal de comparaţii, pentru a atinge înregistrarea căutată, este $(\log_2 n)+1$, unde *n* este numărul de înregistrări ale fişierului.

Utilizarea principală a fişierelor secvenţiale este legată de prelucrarea secvenţială a datelor. Avantajul posibilităţii accesării rapide a înregistrărilor în formă continuă devine un dezavantaj, dacă fişierul este utilizat pentru accesarea unei înregistrări care nu este una următoare sau dacă atributul de căutare nu este cheie de ordonare. Parcurgerea va fi lentă, creând impresia că fişierul nu este ordonat, necesitând o căutare exhaustivă şi citind, în medie, jumătate de fişier pentru a găsi înregistrarea specificată. Astfel, în general, procesarea unui fişier secvenţial se face conform modului de organizare a acestuia, adică, predomină procesarea secvenţială.

Prin urmare, dacă cheia de acces este diferită de cea de căutare, atunci fişierul este unul serial în care se realizează o căutare secvenţială, pornind de la prima înregistrare, până când se localizează cea cu valoarea cheii de acces egală cu argumentul de căutare sau se atinge extremitatea fişierului, adică înregistrarea căutată nu se găseşte în fişier. Când cheia de acces este egală cu cheia de ordonare există două alternative:

- pentru dispozitive cu acces secvenţial fişierele sunt citite secvenţial până când există înregistrări pentru examinare şi cheia de căutare este mai mică decât valoarea atributului cheie sau până când înregistrarea căutată este găsită;
- pentru dispozitive cu acces direct sunt utilizate tehnici mai eficiente, precum căutarea binară, prin blocuri sau interpolare.

Consultarea cu cheia de acces diferită de cheia de ordonare. Consultarea cu cheia de acces diferită de cheia de ordonare are loc ca în fişierele neordonate. Căutarea este făcută prin lectura exhaustivă până când

se localizează înregistrarea căutată sau se termină fişierul. Aici NMC = *(n+1)/2*, unde NMC este numărul mediu de comparaţii, iar *n* – numărul de înregistrări în fişier. Algoritmul poate fi următorul:

1. *Se merge la poziţia iniţială a fişierului.*
2. *Până când nu este atins sfârşitul fişierului:*
   a) *dacă înregistrarea curentă = înregistrarea dorită, terminare cu succes;*
   b) *avansare cu o înregistrare.*
3. *Terminare cu eşec.*

Consultarea cu cheia de acces egală cu cheia de ordonare. Consultarea cu cheia de acces egală cu cheia de ordonare (sau cu partea ei iniţială) a fişierului stocat pe un dispozitiv cu acces secvenţial se face cu o căutare secvenţială. Singurul avantaj este că, dacă înregistrarea curentă are valoarea cheii mai mare decât a celei căutate, ea nu există şi căutarea este întreruptă. Dacă, însă, dispozitivul permite accesul direct, se poate realiza o căutare mai eficientă, cum ar fi căutarea binară şi atunci NMC= $(\log_2 n)+1$ Algoritmul de căutare binară este următorul:

1. *Definirea P_i (poziţia primei înregistrări a fişierului) şi P_f (poziţia ultimei înregistrări a fişierului).*
2. *Până când ($P_i \leq P_f$) se face:*
   a) *calcularea P_m (poziţia medie) = (P_i + P_f)/2;*
   b) *se trece la poziţia P_m;*
   c) *dacă înregistrarea curentă P_m=înregistrarea căutată, terminare cu succes;*
   d) *dacă înregistrarea curentă P_m>înregistrarea curentă, P_i = P_m + 1;*
   e) *dacă înregistrarea curentă P_m<înregistrarea curentă, P_f = P_m - 1;*
3. *Terminare cu eşec.*

Inserarea înregistrărilor. Inserarea unei înregistrări în timp real are un cost înalt, deoarece trebuie efectuată reordonarea fişierului după cheia de ordonare. Pentru aceasta, se determină poziţia adecvată a înregistrării noi (conform cheii primare), se deplasează toate înregistrările care posedă cheia mai mare decât a celei incluse, se inserează înregistrarea nouă. Algoritmul de inserare a înregistrării

1. *Se merge la sfârşitul fişierului.*
2. *Până când argumentul de căutare nu este mai mare decât valoarea respectivă a înregistrării curente şi nu s-a ajuns la începutul fişierului:*
   a) *se deplasează înregistrarea cu o poziţie (poziţia nouă=poziţia anterioară+1);*
   b) *se trece la înregistrarea vecină (se micşorează cu o poziţie).*
3. *Se inserează înregistrarea nouă în fişier.*

O alternativă pentru inserarea unei înregistrări poate fi următorul algoritm: se creează un fişier auxiliar şi se copiază fişierul original, plasând înregistrarea nouă în poziţia corectă, se suprimă fişierul original şi se renumeşte fişierul auxiliar cu numele fişierului original.

Însă, pentru fişierele mari, stocate pe suport extern, aceste procese au un cost prohibitiv. De obicei, se utilizează un fişier auxiliar care conţine înregistrările noi, ordonate în acelaşi mod (după aceeaşi cheie) ca şi fişierul principal de date. Dar utilizarea fişierului auxiliar influenţează procesul de realizare a tuturor operaţiilor care trebuie efectuate asupra ambelor fişiere şi nu numai asupra unuia. Toate accesările înregistrărilor, care sunt realizate pentru toate operaţiile, trebuie să fie executate în două fişiere.

Suprimarea înregistrărilor. Suprimarea sau eliminarea înregistrării trebuie făcută la nivel fizic, cu reorganizarea fişierului în timpul executării operaţiei. Suprimarea presupune următoarele activităţi:

1. *Se merge la începutul fişierului.*
2. *Se localizează înregistrarea care trebuie exclusă.*
3. *Se trece, una după alta, fiecare înregistrare de după înregistrarea ce urmează a fi suprimată şi se micşorează poziţia ei cu 1.*

O alternativă, de asemenea, poate fi utilizarea unui fişier auxiliar. În acest caz, se trece la începutul fişierului, se copiază înregistrările de până la cea care trebuie suprimată, se ignoră înregistrarea care trebuie exclusă, se copiază înregistrările rămase în fişierul auxiliar, se substituie fişierul iniţial cu cel auxiliar.

Excluderea fizică, deci, necesită un cost prohibitiv pentru prelucrare în timp real, deoarece trebuie deplasate toate înregistrările care urmează după cea suprimată. Prelucrarea poate fi făcută în lot, dacă operaţiile pot fi realizate mai târziu, adică comenzile de excludere pot fi colectate într-un fişier de tranzacţii pentru realizarea lor ulterioară. De multe ori, această procedură nu poate fi acceptată. Mai frecventă este procedura care presupune includerea în înregistrare a unui câmp adiţional, în care se indică excluderea. Astfel, pentru a elimina o înregistrare se modifică doar valoarea acestui câmp (modificarea înregistrării), prin care se arată că ea a fost eliminată. Cu această procedură, este evitată necesitatea deplasării altor înregistrări pentru completarea spaţiilor eliberate în urma eliminării.

Modificarea înregistrărilor. Modificarea unei înregistrări presupune modificarea valorilor unor câmpuri ale înregistrării. Pentru aceasta, înregistrarea este localizată, citită, câmpurile ei sunt modificate şi apoi ea este scrisă. În general, sunt actualizate numai atributele care nu fac parte din cheie. Dar, dacă înregistrarea are lungimea variabilă şi modificarea măreşte lungimea ei, înregistrarea nu poate fi înscrisă în poziţia iniţială, deoarece lipseşte spaţiul necesar. Atunci, de obicei, ea este exclusă şi apoi inclusă, dar, deja, actualizată.

Dacă se modifică vreun atribut al cheii de ordonare, schimbarea valorii lui implică schimbarea poziţiei înregistrării. În acest caz, operaţia este, în general, implementată prin excluderea înregistrării vechi, urmată de includerea în poziţia respectivă a variantei modificate. Însă, în majoritatea aplicaţiilor, nu este permisă modificarea cheii. În alte aplicaţii poate fi folosit un fişier de tranzacţii. Dacă este utilizată banda magnetică, operaţia de rescriere poate fi executată doar mai târziu, întrucât nu poate fi rescrisă o înregistrare care este citită.

Lectura exhaustivă a înregistrărilor. Lectura exhaustivă a înregistrărilor este o operaţie eficientă, deoarece constă în citirea fiecărei înregistrări din fişier. Cea mai adecvată organizare pentru acest tip de operaţii este cea secvenţială.

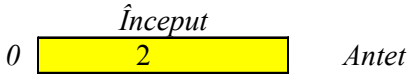
În general, dacă numărul de operaţii de actualizare a fişierului este foarte mare, se poate utiliza una din opţiuni: un fişier auxiliar adiţional cu actualizări care vor fi făcute mai târziu (procesul *batch*); inserarea la sfârşitul fişierului şi reorganizarea (ordonarea) ulterioară a lui.

Fişiere secvenţiale ordonate la nivel logic de pointeri

Tehnicile descrise anterior oferă avantaje în cazul prelucrării secvenţiale şi în cazul căutării înregistrărilor cu metode mai rapide de căutare. Însă dezavantajul lor este timpul cheltuit în operaţiile de actualizare (includere, excludere şi modificare a atributelor utilizate în ordonarea fişierului). Aceasta se datorează faptului că fişierele trebuie reorganizate în toate operaţiile de includere/excludere, deplasând înregistrările pentru a menţine ordinea fizică egală cu cea logică. Prin urmare, deoarece au un cost înalt, pentru includerea, modificarea şi excluderea înregistrărilor, trebuie propuse tehnici mai eficiente.

Fişierul secvenţial cu pointeri a fost elaborat pentru a mări eficienţa sistemelor cu multe inserări şi suprimări, cum sunt, de exemplu, bazele de date. Structura unui fişier cu pointeri este similară unei liste înlănţuite. În acest caz, nu este necesar a avea o ordine fizică a fişierului egală cu ordinea logică. Această tehnică, deşi are dezavantajul că este mai lentă în parcurgerea secvenţială a înregistrărilor, poate fi necesară la citirea şi recitirea unor sectoare care sunt plasate în diferite locuri ale fişierului (necontinuu).

Acest tip de fişiere posedă unele structuri adiţionale, figura 28.8. În primul rând, există un atribut care indică ordinea logică, adică adresa următoarei înregistrări din fişier, în secvenţa structurii logice conform cheii de clasificare. În al doilea rând, există atributul înregistrărilor suprimate, care indică faptul că înregistrarea este activă sau deja a fost eliminată. Altă structură este *înregistrarea zero* sau *antetul*. Înregistrarea zero posedă atributul *Început*, care arată adresa primei înregistrări în fişier, figura28.8, înregistrarea aflată pe adresa 2. Astfel, structura reprezintă o listă care indică înregistrările în uz şi ordinea lor în fişier, şi un câmp *Exclusă* pentru înregistrările şterse.



	Nume	…	*Exclusă*	*Pointer*
1	Donică	…		3
2	Amihălăchioaei	…		5
3	Guțu	…	*	4
4	Munteanu	…		6
5	Petrescu	…	*	1
6	Rotaru	…		*

Figura 28.8 *Un fişier secvenţial cu pointeri*

Există şi implementarea cu două liste, figura 28.9. Deşi, prin adăugarea unui pointer în înregistrare structura se complică, această formă de organizare permite utilizarea spaţiului înregistrărilor eliminate pentru inserarea înregistrărilor noi. Or, căutarea spaţiului disponibil, al unei înregistrări suprimate, într-un fişier cu multe înregistrări este lentă, deoarece necesită multe accesări. În afară de pointerul de păstrare a secvenţei logice a înregistrărilor este utilizat un alt pointer pentru păstrarea listei înregistrărilor excluse. Astfel, poate fi utilizat spaţiul eliberat, inserând, mult mai rapid, înregistrări noi. Nu se cunoaşte doar poziţia primei înregistrări şi nici prima înregistrare care a fost eliminată. Adică, este necesară adăugarea la fişier a unui antet pentru păstrarea acestor informaţii. În structura cu două liste, una indică înregistrările în uz şi ordinea lor în fişier, iar alta - înregistrările suprimate al căror spaţiu poate fi utilizat pentru inserare.

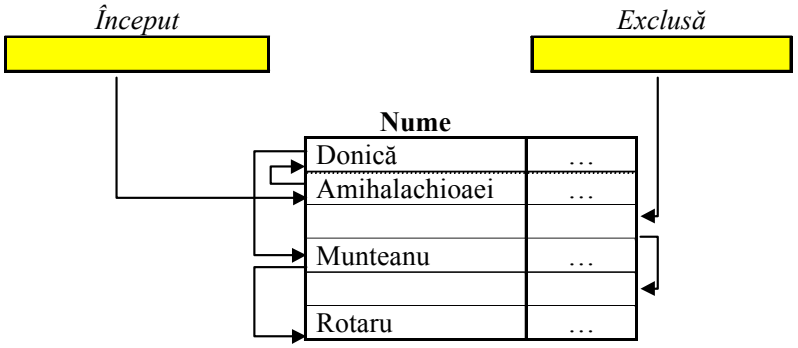


Figura 28.9 *Un fişier secvenţial cu două structuri de pointeri*

În structura cu doi pointeri, în procesul de consultare, se identifică prima înregistrare cu adresa indicată de atributul Început al antetului. Dat fiind faptul că fişierul nu este ordonat fizic, în această structură, nu este posibilă realizarea căutării binare. Apoi, se parcurge fişierul, folosind adresele indicate în atributul Pointer al secvenţei de înregistrări până ce câmpul cerut va fi localizat. Algoritmul de consultare:

1. *Se trece la poziţia primei înregistrări indicată în antet.*
2. *Până când înregistrarea curentă <= înregistrarea dezirabilă:*
   a) *dacă înregistrarea curentă = înregistrarea dezirabilă, terminare cu succes;*
   b) *dacă pointerul secvenţei ≥0, se trece la următoarea înregistrare indicată de pointer;*
   c) *altfel terminare cu eşec.*
3. *Terminare cu eşec.*

După cum se poate observa, sfârşitul căutării (sfârşitul fişierului) este atins când pointerul secvenţei de înregistrări nu indică spre altă înregistrare (nu posedă vreo valoare sau adresă).

Pentru a include o înregistrare, nu mai e nevoie de reorganizarea fişierului, deoarece ordinea fizică, în acest caz, poate fi diferită de ordinea logică (care este determinată de pointeri). Este suficientă inserarea înregistrării în primul loc liber. Primul loc liber poate fi o înregistrare suprimată şi pentru a şti că există o astfel de înregistrare, este destul de

verificat atributul *Exclusă* în antetul fişierului. În cazul în care nu există înregistrări excluse, un spaţiu nou este creat la sfârşitul fişierului. După selectarea locului de inserare se parcurge fişierul pentru identificarea înregistrărilor precedentă şi următoare ale înregistrării curente. Acestea trebuie să aibă valorile pointerilor actualizate, pentru ca înregistrarea nouă să ocupe poziţia logică intermediară.

Algoritmul este următorul:

1. *Localizarea unei înregistrări vide (suprimate) sau crearea unui spaţiu pentru înregistrarea nouă la sfârşitul fişierului.*
2. *Inserarea înregistrării în locul identificat sau creat.*
3. *Dacă înregistrarea a fost inserată în locul uneia eliminate, se exclude din lista înregistrărilor suprimate.*
4. *Identificarea înregistrărilor precedente şi următoare;*
5. *Dacă există înregistrarea precedentă se actualizează pointerul ei pentru ca să arate spre înregistrarea inserată.*
6. *În caz contrar, pointerul antetului este cel ce trebuie actualizat.*
7. *Dacă există înregistrarea următoare, pointerul înregistrării inserate se actualizează pentru a o indica.*

Pentru a exclude o înregistrare, se identifică înregistrarea care precede înregistrarea în cauză şi se actualizează pointerul ei. Deoarece înregistrarea precedentă trebuie să dea continuitate logică fişierului, pointerul ei se modifică pentru a indica înregistrarea ce urmează după înregistrarea exclusă. Astfel, valoarea pointerului înregistrării suprimate devine valoare a pointerului înregistrării anterioare.

Algoritmul constă din patru paşi:

1. *Localizarea înregistrării ce trebuie suprimată, utilizând lista secvenţei de înregistrări (pointerul).*
2. *Identificarea înregistrării precedente şi înregistrării următoare;*
3. *Actualizarea pointerului înregistrării precedente, atribuindu-i valoarea pointerului înregistrării ce trebuie suprimată;*
4. *Adăugarea înregistrării excluse în lista înregistrărilor excluse.*

Modificarea unei înregistrări poate fi realizată prin eliminarea înregistrării ce trebuie modificată şi inserarea înregistrării cu valorile deja modificate (metodă mai laborioasă, dar mai facilă şi practică).

Fişiere secvenţiale cu spaţiu de tranzacţii

Fişierele secvenţiale pot fi dotate cu un fişier auxiliar, numit *fişier de tranzacţii*. Deplasarea înregistrărilor este o operaţie costisitoare. Pentru a evita deplasarea înregistrărilor (rearanjarea) în fişierul principal în timpul executării operaţiilor, o altă formă de implementare a fişierului secvenţial presupune utilizarea a două structuri:

- fişierul principal (master) cu date.
- fişierul de modificări sau fişierul de tranzacţii (overflow) este un fişier auxiliar, în care sunt temporar înregistrate modificările curente ale datelor, care vor fi utilizate pentru actualizarea conţinutului fişierului principal. În acest mod, nu au loc deplasări de înregistrări în fişierul principal în timpul de executare a

operaţiilor. Fişierul de tranzacţii poate fi chiar unul virtual, adică, înregistrările pot fi plasate la sfârşitul fişierului principal.

Astfel, pentru a micşora costul înalt de păstrare a ordinii fizice, se utilizează două structuri, plus o listă de pointeri a secvenţei de înregistrări pentru menţinerea ordinii logice, adică se separă grupul de înregistrări care sunt fizic ordonate de operaţiile nou-realizate.

Fişiere secvenţiale cu toate operaţiile în fişierul de tranzacţii. Această variantă de fişiere secvenţiale presupune că toate modificările sunt făcute în fişierul de tranzacţii. Fişierul principal este actualizat numai în cazul reorganizării, folosind datele din fişierul de tranzacţii.

În fiecare înregistrare din fişierul de tranzacţii, trebuie să fie un câmp auxiliar, unde se indică tipul operaţiei ce a fost realizată (*I* – inserarea, *E* – eliminarea şi *M* – modificarea). Toate operaţiile (consultarea, inserarea, eliminarea, modificarea) trebuie să înceapă cu căutarea înregistrării în fişierul de tranzacţii şi examinarea etichetei acestui câmp. Dacă înregistrarea nu este găsită, atunci este căutată în fişierul principal. Trebuie menţionat că fişierul de tranzacţii este, de asemenea, un fişier secvenţial şi, deci, se supune aceloraşi reguli (înregistrări ordonate de cheia de ordonare).

Inserarea trebuie să fie precedată de o consultare. Dacă cheia este găsită în fişierul de tranzacţii cu eticheta *I* sau *M*, sau dacă cheia este întâlnită în fişierul principal, atunci trebuie să fie acţionat procesul de tratare a erorii (nu poate fi făcută o inserare cu o cheie deja existentă). Dacă cheia e găsită în fişierul de tranzacţii şi are eticheta *E*, se inserează o înregistrare nouă în fişierul de tranzacţie, care este stocată în ordinea cheii de ordonare. Astfel, pot fi necesare deplasări de înregistrări. Cu toate acestea, deoarece fişierul de tranzacţii este mic, executarea operaţiei de deplasare este uşoară. Eticheta *I* trebuie plasată în înregistrarea nouă, indicând că este una inserată.

Utilizarea fişierului de tranzacţii schimbă modul de consultare a înregistrărilor. Astfel, o consultare a unei singure înregistrări după cheia de ordonare, începe de la fişierul de tranzacţii, deoarece acesta conţine ultimele modificări. Trebuie, de asemenea, să fie examinată eticheta operaţiei (*I*, *E* sau *M*). Eticheta *E* indică faptul că cheia nu mai există, iar celelalte etichete indică faptul că consultarea va fi făcută asupra înregistrării în fişierul de tranzacţii. Însă, o consultare cu o cheie diferită de cea de organizare are un cost înalt, deoarece fişierul de tranzacţii trebuie analizat (cu examinarea etichetelor) în întregime şi apoi fişierul principal, la fel, în mod exhaustiv.

Operaţia de parcurgere secvenţială după cheia de ordonare este făcută în paralel în ambele fişiere şi urmează aceleaşi raţionamente ca ale operaţiei de reorganizare, cu intercalare. Dacă este necesară parcurgerea secvenţială a înregistrărilor în altă ordine, atunci operaţia trebuie făcută cu algoritmi de ordonare, ţinând cont de existenţa înregistrărilor duplicate în ambele fişiere.

Modificările câmpurilor care nu fac parte din cheia de ordonare trebuie făcute în fişierul de tranzacţii. Mai întâi, înregistrarea trebuie căutată (operaţia de consultare). Dacă ea nu este găsită în nici unul din fişiere, este acţionată o procedură de eroare. Dacă ea este găsită numai în fişierul principal, o înregistrare nouă va fi inserată în fişierul de tranzacţii, cu aceleaşi câmpuri ca ale înregistrării iniţiale, dar cu modificările necesare şi cu eticheta *M*. Dacă ea este găsită în fişierul de tranzacţii, se verifică

eticheta ei. Dacă este *E*, ea deja a fost exclusă şi operaţia este întreruptă. Dacă eticheta este *I*, se modifică direct această înregistrare cu eticheta *I* în fişierul de tranzacţii, fără a modifica eticheta. Modificările (dacă se permit) în câmpurile care fac parte din cheia de ordonare sunt realizate în două operaţii: una de eliminare a înregistrării originale şi una de inserare a înregistrării modificate.

Operaţia de suprimare se efectuează după următoarele reguli. Mai întâi, se caută înregistrarea (operaţia de consultare). Dacă nu sete găsită, există o greşeală. Dacă există numai în fişierul principal, o înregistrare este adăugată în fişierul de tranzacţii (în ordinea cheii) cu eticheta de excludere. Dacă, deja, există în fişierul de tranzacţii, eticheta ei va indica următoarele procese. Dacă eticheta este *E*, există o eroare (înregistrarea a fost, deja, eliminată). Dacă este *M*, se schimbă această etichetă cu *E,* iar dacă este *I*, această înregistrare trebuie eliminată fizic din fişierul de tranzacţii cu deplasări succesive de poziţii (această operaţie nu va putea fi desfăcută).

Periodic, operaţiile înregistrate în fişierul de tranzacţii trebuie actualizate în fişierul principal. De aceea, eticheta fiecărei înregistrări din fişierul de tranzacţii este analizată şi operaţiile respective sunt făcute în fişierul principal.

Deoarece această acţiune provoacă deplasări în fişierul principal, o alternativă este crearea unui fişier suplimentar, în care se copiază fişierul principal şi în acesta sunt aplicate modificările. În finalul operaţiei de reorganizare, fişierul de tranzacţii devine vid, fişierul principal este suprimat şi fişierul auxiliar este redenumit cu numele fişierului principal. Deoarece operaţia de reorganizare are un cost înalt, ea trebuie să fie făcută *off-line*, adică atunci când înregistrările nu sunt actualizate. Periodicitatea depinde de disponibilitatea de timp pentru realizarea acestei operaţii şi de numărul de înregistrări în fişierul de tranzacţii (se activează operaţia de reorganizare, dacă volumul de tranzacţii atinge o anumită limită, după care eficienţa generală de păstrare a datelor de SGBD scade).

Avantajul utilizării acestui tip de organizare apare în cazul în care sunt efectuate puţine modificări *on-line* şi când este posibilă întreruperea lor pentru realizarea operaţiei de reorganizare. Alt avantaj îl constituie parcurgerea secvenţială, în mod aproape optimal, deoarece înregistrările se găsesc, din punct de vedere fizic, continuu. Cu toate acestea, există tendinţa de a analiza fişierul principal şi cel tranzacţional împreună.

Un mare dezavantaj al acestui tip de organizare îl constituie faptul că consultările pot fi făcute, în mod eficient, numai după cheia de organizare. Dacă sunt necesare consultări sau parcurgeri pe mai multe chei, această organizare nu este recomandată.

Fişiere secvenţiale cu unele operaţii în fişierul principal. Pentru a reduce creşterea fişierului de tranzacţii, operaţiile care nu cer deplasări de înregistrări sunt realizate direct în fişierul principal. În acest mod, se diminuează viteza de creştere a fişierului de tranzacţii şi, prin urmare, micşorează perioada de aplicare a operaţiilor de reorganizare.

În afară de aceasta, pentru a înviora consultarea când se fac operaţii asupra fişierului de tranzacţii, el, de asemenea, ar trebui să fie menţinut ordonat, mărind, astfel, costul operaţiilor. De aceea, acest proces nu este acceptat şi inserările se realizează la sfârşitul fişierului.

Se poate observa că ordinea fizică şi ordinea logică sunt păstrate în fişierul principal. În fişierul de tranzacţii, înregistrările nu sunt ordonate, întrucât înregistrările sunt inserate la sfârşit. De aceea, pentru a menţine

aceeaşi ordine în care a fost ordonat, este necesară adăugarea unui pointer pentru fiecare înregistrare cu o adresă de deplasare (*offset*). Această adresă indică următoarea înregistrare care va fi folosită, dacă e necesar, pentru a întocmi lista, menţinând, astfel, secvenţa logică a datelor după cheia de ordonare.

O dată cu adăugarea unui câmp pentru fiecare înregistrare, ocupat de pointer, există dezavantajul ca procesul de parcurgere secvenţială se realizează mai lent, deoarece poate fi necesară citirea şi recitirea sectoarelor care se găsesc în diferite locuri ale fişierului (necontinuu), fapt ce consumă mai mult timp de căutare şi de stare latentă, în afară de supraîncărcarea cu spaţiul ocupat de pointeri.

Inserarea unei înregistrări este făcută în fişierul de tranzacţii sau, dacă este virtual, la sfârşitul fişierului principal în zona de tranzacţii (de parcă nu ar exista cheia de clasificare). Secvenţa logică este păstrată de lista definită de pointerul care trebuie actualizat la fiecare operaţie realizată. Mai târziu, fişierul este reorganizat.

În cazul operaţiei de *modificare*, dacă nici cheia primară şi nici lungimea înregistrării nu sunt schimbate, înregistrarea este actualizată în aceeaşi poziţie. În caz contrar, ea este eliminată şi cea nouă este inserată în fişierul de tranzacţii. Această operaţie trebuie să actualizeze lista secvenţei logice a datelor.

În cazul operaţiei de eliminare, fişierul principal poate fi reorganizat (operaţia necesitând mult timp). Atunci este necesară redefinirea listei secvenţei logice, care ar include toate înregistrările care nu au fost eliminate.

O altă metodă este adăugarea unui câmp auxiliar pentru păstrarea unui indicator al înregistrărilor excluse, folosit la restructurarea ulterioară a fişierului.

Reorganizarea fişierelor. Fişierul de tranzacţii, de obicei, este folosit doar pentru stocarea operaţiilor de actualizare a fişierului secvenţial şi în operaţia de reorganizare care este aplicată numai în cazul când fişierul principal se reorganizează, deoarece acumularea tranzacţiilor, poate diminua eficienţa executării lor. Aşadar, în general, când fişierul de tranzacţii atinge o limită determinată sau se doreşte elaborarea unui raport, sau efectuarea unei consultări are loc reorganizarea fişierului secvenţial. Or, în prelucrarea normală, este puţin obişnuită utilizarea fişierului de tranzacţii ca o prelungire a fişierului secvenţial.

Procesul de reorganizare a unui fişier secvenţial *S* presupune aplicarea unui algoritm, cum este cel de triere prin fuziune. Pentru aceasta, fişierul de tranzacţii *T* trebuie să fie ordonat după acelaşi criteriu (cheie) ca şi fişierul *S*. Din fişierele (de tranzacţii *T* şi principal *S*) ordonate după acelaşi criteriu se poate construi un fişier secvenţial nou *A*. Printr-o tratare secvenţială a fişierelor, cu o lectură exhaustivă a lui *S*, se copiază fişierul principal iniţial, înregistrare cu înregistrare, în fişierul principal actualizat *A* şi, pentru fiecare înregistrare copiată, se intercalează cu înregistrările fişierului de tranzacţii *T*, dacă există, făcând comparaţiile necesare şi producând copia actualizată a fişierului principal.

Algoritmul este următorul:

1. *Se citeşte o înregistrare a fişierului principal S (dacă nu este marcată ca fiind suprimată).*

2. *Se citeşte o înregistrare a fişierului de tranzacţii T (dacă nu este marcată ca fiind suprimată).*
3. *Se compară aceste două înregistrări. Cea cu valoare mai mică a cheii de ordonare este copiată în fişierul principal actualizat A.*
4. *În fişierul care conţine înregistrarea scrisă în fişierul principal actualizat A, pointerul înregistrării avansează (se localizează următoarea înregistrare).*
5. *Se citeşte o înregistrare nouă a acestui fişier (dacă nu este marcată ca fiind suprimată).*
6. *Se compară aceste două înregistrări, scriind-o pe cea mai mică şi repetând procesul până când se atinge sfârşitul unui fişier din cele două.*
7. *Se înscriu înregistrările nesuprimate ale fişierului, care încă nu a fost parcurs până la sfârşit, în fişierul principal actualizat A.*

În acest proces, se observă că:

- înregistrările suprimate nu sunt copiate (sunt eliminate efectiv);
- înregistrările fişierului principal S, care nu figurează în fişierul de tranzacţii T, sunt copiate în fişierul actualizat A;
- înregistrările fişierului de tranzacţii T sunt prelucrate în lot şi plasate în fişierul actualizat A cu păstrarea secvenţei ordonate după cheie.

În consecinţă, fişierul de tranzacţii devine vid, iar înregistrările fişierului principal actualizat *A* sunt ordonate astfel încât secvenţa logică este egală cu secvenţa fizică.

Căutarea secvenţială cu blocuri de înregistrări

Partea ce mai scumpă (lentă) a operaţiei de accesare a memoriei secundare este căutarea pentru obţinerea poziţiei corecte a unei înregistrări pe disc. Aşadar, se cere minimizarea numărului de accesări, pentru că transferarea datelor, odată iniţiată, este relativ rapidă, deşi este mult mai lentă decât transferarea datelor în memoria principală.

Evident, căutarea (*seek*) şi citirea unei înregistrări şi apoi căutarea şi citirea altei înregistrări au un cost mai mare decât căutarea şi citirea concomitentă a două înregistrări. Prin urmare, se poate îmbunătăţi căutarea secvenţială, dacă, în loc de înregistrări, se recuperează blocuri de înregistrări şi aceste blocuri se prelucrează în memoria principală. Astfel, gruparea înregistrărilor în blocuri este o tehnică utilizată pentru îmbunătăţirea căutării. În general, dimensiunea blocului este definită în funcţie de caracteristicile fizice ale dispozitivului de stocare şi ale datelor ce trebuie memorate.

Bineînţeles, fiecare accesare a unui bloc de înregistrări va lua ceva mai mult timp decât o accesare a unei înregistrări, dar beneficiul va fi considerabil datorită reducerii timpului de căutare şi timpului de rotaţie. Astfel, formarea blocurilor de înregistrări:

- măreşte eficienţa căutării prin diminuarea numărului de accesări;
- profită de diferenţa dintre costul de accesare în memoria principală şi costul de accesare a discului;
- economiseşte timpul, deoarece reduce timpii de căutare şi durata stării latente;
- nu modifică numărul de comparaţii în memoria principală, dar, probabil, măreşte cantitatea de date transferate între disc şi

memoria principală (este citit un bloc întreg, în timp ce înregistrarea căutată este una în bloc, iar restul nu sunt necesare).

Astfel, se poate concluziona că deosebirea dintre accesul la memoria principală şi disc este ceea ce ghidează proiectarea structurilor de fişiere şi, prin urmare, căutarea secvenţială este mai eficientă:

- în fişierele cu puţine înregistrări;
- în fişierele cu puţine căutări (de exemplu, păstrate pe benzi magnetice);
- în căutarea înregistrărilor după chei secundare, ale căror valori au mai multe duplicate.

28.4 Indecşi

Indecşii sunt structuri de acces care se utilizează pentru accelerarea accesului la înregistrări şi a răspunde anumitor criterii de căutare. Unele tipuri de indecşi, numite şi căi de acces secundare, nu afectează amplasarea fizică a înregistrărilor pe disc, fapt ce oferă căi alternative de acces pentru găsirea înregistrărilor, în mod eficient, în câmpurile indexate. Există şi alte tipuri de indecşi care se pot construi numai asupra fişierelor care au o anumită organizare.

Astfel, există tipuri de indecşi care se utilizează asupra fişierelor ordonate (indecşi cu un singur nivel) şi structuri sub formă de arbore (indecşi multinivel, B-arbori şi B^+-arbori). În afară de aceasta, se pot construi indecşi, folosind funcţii de dispersie şi alte structuri de date.

Utilizarea tehnicilor de indexare poate fi o alternativă a ordonării, dacă este necesară organizarea unui fişier, pentru a fi căutat cu ajutorul cheilor.

În general, indecşii ameliorează executarea accesului la date. În cazul fişierelor, permit localizarea rapidă a înregistrărilor, cu avantajul că fişierul de date nu trebuie să fie reorganizat, dacă sunt inserate noi înregistrări în acesta. Este de ajuns să fie reorganizaţi indecşii. Indecşii, de asemenea, permit, în afară de ameliorarea timpului de acces pentru căutarea după o cheie, susţinerea mai multor viziuni asupra înregistrărilor dintr-un fişier de date, graţie structurilor de indecşi secundari. Ba mai mult, cu ajutorul indecşilor, pot fi îmbinate mai multe viziuni particulare.

Indexul poate exista independent de organizarea fişierului de date, ceea ce permite crearea mai multor indecşi, dacă se doreşte optimizarea accesului la date al mai multor tipuri de interogări. În schimb, crearea fără discernământ a unui număr mare de indecşi poate fi costisitoare pentru SGBD-ul care trebuie să-i administreze. Pentru fiecare operaţie de actualizare a relaţiei, repercusiunile acestei actualizări se extind asupra tuturor indecşilor. O alegere judicioasă a indecşilor, într-un număr optim, este, deci, unul din factorii esenţiali ai performanţei unui sistem.

Concepte preliminare

Indexul este un fişier. Fişierul index este o structură auxiliară asociată fişierului de date, proiectat pentru a oferi forme mai eficiente de acces şi localizare a datelor specificate. Adică, fiind dat un argument de căutare,

scopul indexului este accelerarea procesului de identificare a adresei înregistrării necesare în fişierul de date.

Pot exista unul sau mai multe fişiere index, ale căror înregistrări leagă valorile cheii cu poziţiile lor în fişierul de date. Adică o înregistrare a fişierului index este constituită din două câmpuri, cheia înregistrării din fişierul de date şi adresa respectivă pe disc, <*cheie, adresă*>.

Astfel, un index este întotdeauna specificat de o cheie de acces, iar în calitate de cheie de acces poate fi luată orice submulţime de atribute ale schemei relaţionale, dacă această submulţime este definită în calitate de index. Există două tipuri principale de indecşi:

- indexul ordonat, care se bazează pe valorile ordonate ale cheilor. Înregistrările fişierului sunt stocate conform unui criteriu de ordonare;
- indexul hash sau index cu dispersie, care se bazează pe distribuirea uniformă a valorilor determinate de o funcţie, numită funcţie de dispersie (hash function).

Există diverse tehnici atât pentru indecşii ordonaţi, cât şi pentru indecşii cu dispersie. Nici una din ele nu e mai bună, dar fiecare tehnică este mai adecvată pentru anumite aplicaţii ale bazei de date. Factorii care determină selectarea tipului de index sunt:

- tipul de acces, care este admis în mod eficient; tipurile de acces pot căuta înregistrări după o valoare determinată a unui atribut dat sau înregistrări ale căror atribute iau valori dintr-un interval de valori;
- timpul de acces – timpul consumat pentru găsirea unei înregistrări de date, sau a unei mulţimi de înregistrări, utilizând tehnica respectivă;
- timpul de inserare – timpul consumat pentru includerea unui nou articol de date; acest factor cuprinde timpul cheltuit pentru găsirea locului corect, pentru includerea articolului şi, de asemenea, timpul cheltuit pentru actualizarea structurii indexului;
- timpul de suprimare – timpul consumat pentru eliminare, inclusiv timpul afectat găsirii înregistrării şi actualizării structurii indexului.
- spaţiul de memorie – spaţiul adiţional ocupat de fişierul index; dacă mărimea spaţiului adiţional este rezonabilă, în general, spaţiul sacrificat se recuperează prin obţinerea unei executări mai bune a operaţiilor.

Un fişier index este mai uşor de lucrat decât un fişier de date, deoarece înregistrările indexului sunt de lungime fixă (facilitează navigarea cu tehnici mai eficiente de căutare, cum este cea binară) şi este mult mai mic decât fişierul de date. Înregistrările de lungime fixă ale fişierului index impun şi o limită a lungimii cheii primare, fapt care, uneori, poate crea probleme în practică. Înregistrările indexului pot conţine şi alte câmpuri (lungimea înregistrării, de exemplu), în afara celor două.

De obicei, pentru unul şi acelaşi fişier de date pot exista mai multe fişiere index. Aplicând noţiunea de cheie de căutare, dacă sunt mai mulţi indecşi pentru un fişier de date, există mai multe chei de căutare în acest fişier.

Utilizarea indecşilor are următoarele avantaje:

- fişierul de date poate fi de tip serial (înregistrările sunt inserate la sfârşitul fişierului, în ordinea de intrare);

- adăugarea înregistrărilor este mult mai rapidă, dacă indexul poate fi păstrat în memoria principală;
- în index, se poate localiza rapid o cheie, utilizând căutarea binară;
- pornind de la cheie şi adresă, pentru recuperarea unei înregistrări este necesară o singură accesare a fişierului de date.

Astfel, dacă înregistrările unui fişier sunt de lungime variabilă, este dificilă utilizarea metodelor de ordonare şi aplicarea căutării binare. Aici o alternativă poate fi construirea unui index. Structura indexului, în acest caz, poate fi foarte simplă: indexul este un fişier cu înregistrări de lungime fixă, în care fiecare are două câmpuri, un câmp pentru cheie şi altul pentru poziţia iniţială a înregistrării în fişierul de date. Fiecărui câmp cheie al fişierului de date îi corespunde un câmp cheie în index. Indexul este ordonat, în timp ce fişierul de date nu este. Fişierul de date poate fi organizat în ordinea intrării înregistrărilor.

Dacă indexul nu încape în memoria primară, accesarea şi menţinerea lui trebuie făcută în memoria secundară. În acest caz, utilizarea indecşilor simpli este problematică, deoarece:

- căutarea binară poate necesita multe accesări ale discului;
- deplasarea înregistrărilor, după operaţiile de inserare şi eliminare, devine inevitabilă pentru menţinerea indexului.

În acest context, dacă viteza de acces este caracteristica dezirabilă principală, indexul poate fi organizat, utilizând tehnici cu funcţii de dispersie. Dacă nu, B-arborele poate fi o structură acceptabilă atunci când se cer accesări după cheie sau accesări secvenţiale eficiente.

Indecşi ordonaţi

Indexul ordonat păstrează ordinea cheilor de căutare şi oricărei chei îi asociază adresele înregistrărilor care o conţin. Indexul ordonat poate fi primar sau secundar.

Indecşi primari. Indexul ordonat este un *index primar,* dacă fişierul de date şi indexul au acelaşi criteriu de ordonare, adică, dacă fişierul este ordonat de un câmp cheie, indexul care se defineşte pe acest câmp este un index primar. Deseori, termenul index primar este utilizat pentru desemnarea uni index al cheii primare (a unei relaţii din baza de date), dar această formă nu este un standard şi poate fi evitată.

Dacă toate fişierele sunt ordonate secvenţial după o singură cheie de căutare, aceste fişiere cu un index primar pe cheia de căutare sunt numite *fişiere indexat secvenţiale*. Fişierele indexat secvenţiale reprezintă una din cele mai vechi scheme de index utilizate în bazele de date. Ele sunt proiectate pentru aplicaţiile care necesită atât prelucrarea secvenţială a fişierelor, cât şi accesul aleatoriu la înregistrări individuale.

Un fişier index primar poate fi:

- un index dens, dacă pentru orice valoare a cheii de căutare din fişierul de date există o înregistrare în fişierul index (sau intrare index) respectivă. Evident, înregistrarea index conţine valoarea cheii de căutare şi un pointer spre prima înregistrare de date cu această valoare a cheii de căutare.
- un index rar, dacă înregistrările fişierului de date sunt grupate după un criteriu şi pentru fiecare grup (organizat în bloc) există o înregistrare index asociată. Astfel, fiecare înregistrare index are o valoare a câmpului cheie egală cu valoarea cheii primei

înregistrări de date a blocului respectiv şi un pointer spre acest bloc.

Pentru inserarea datelor în poziţia corectă în fişierul de date este necesară, în afară de deplasarea înregistrărilor pentru a deschide spaţiu pentru cea nouă, modificarea unor înregistrări ancoră (prima înregistrare a blocului). Ca şi în indecşii denşi, fiecare înregistrare a indexului rar conţine o valoare a cheii de căutare şi un pointer spre prima înregistrare de date cu această valoare a cheii.

Pentru un fişier index rar, localizarea unei înregistrări după un argument de căutare se face în două etape: în prima este consultat indexul şi determinat blocul în care trebuie să se găsească înregistrarea şi în a doua - blocul selectat este căutat şi localizată în el înregistrarea dorită. Adică, pentru localizarea unei înregistrări, se obţine intrarea indexului cu cea mai mare valoare a cheii, mai mică sau egală cu valoarea cheii căutate. Adresa din acea înregistrare permite accesul la blocul care poate conţine înregistrarea căutată. Apoi, în blocul respectiv, se face o căutare secvenţială sau prin altă metodă.

Deoarece în indexul dens există o înregistrare care indică direct înregistrarea căutată în fişierul de date, se poate crede că recuperarea unei înregistrări cu un index dens este mai eficientă decât a uneia cu un index rar. Însă, indecşii denşi sunt prea mari şi ocupă mai mult spaţiu, fapt ce complică sarcina de actualizare (inserare şi suprimare) a lui. În afară de aceasta, indecşii rari explorează bine caracteristicile fizice ale dispozitivelor secundare. Astfel, fiecare bloc definit de indexul rar poate fi încărcat în memoria principală dintr-o singură accesare şi manipulat în memoria principală, consumând un timp nesemnificativ. Prin urmare, când trebuie să facă o alegere, proiectantul trebuie să găsească un compromis între timpul de acces şi spaţiul suplimentar, pentru a lua o decizie mai bună, care este influenţată direct de aplicaţia în cauză.

Indecşi secundari. Indexul asociat cheii de ordonare a fişierului de date este numit, în afară de index primar, şi index *cluster*. Alţi indecşi, ale căror chei de căutare specifică o ordine diferită de ordinea secvenţială, se numesc *indecşi secundari*. În ambele cazuri, intrările indexului sunt ordonate de valoarea de acces, având ca obiectiv creşterea eficienţei căutării.

Astfel, dacă un fişier nu este ordonat, orice index care se defineşte asupra lui este un index secundar. De asemenea, un index secundar este indexul definit pe un câmp diferit de câmpul de ordonare al fişierului.

Indexul secundar poate fi dens sau rar, având aceleaşi specificaţii ca şi indexul primar. Indexul secundar, ca şi indexul primar, este un fişier ordonat cu două câmpuri: câmpul de indexare (orice câmp al fişierului de date diferit de cel de ordonare) şi un pointer spre un bloc sau spre o înregistrare. Dacă indexul secundar este definit pe un *câmp cheie* (*cheie secundară*), există câte o înregistrare index pentru fiecare cheie a fişierului, adică indexul secundar este un index dens.

Cu introducerea indecşilor secundari, a dispărut necesitatea de pointeri adiţionali în înregistrări şi necesitatea de parcurgere secvenţială în ordinea cheii secundare. Însă utilizarea acestei structuri îl impune pe administratorul bazei de date să analizeze mulţimea potenţială de interogări pentru a selecta câmpurile asupra cărora se vor defini indecşii secundari.

Indecşi multinivel. Pentru fişierele cu multe înregistrări, indexul este, totuşi, destul de mare şi nu poate fi încărcat în memoria principală. De aceea sunt utilizaţi indecşi cu multe niveluri, adică indecşi cu niveluri rare.

Un index multinivel este format pentru un fişier index, care se numeşte primul nivel sau nivel-bază al indexului multinivel. Primul nivel este un fişier ordonat cu o valoare distinctă a câmpului de indexare în fiecare intrare. De aceea, asupra primului nivel se poate crea un index primar. Acest index constituie al doilea nivel al indexului multinivel. Deoarece al doilea nivel este un index primar, în el există câte o intrare în fiecare bloc al primului nivel. Procesul poate continua şi asupra acestui nivel, dacă este necesar. Atunci al treilea nivel va fi un index primar pentru nivelul doi.

Această schemă multinivel se poate utiliza asupra oricărui tip de index, fie primar, fie secundar, cu condiţia ca, întotdeauna, primul nivel să ia valori distincte în câmpul de indexare şi intrările să fie de lungime fixă. Astfel, dacă fiecare bloc se presupune că are *r* intrări, adică fiecare bloc reduce numărul de intrări de la nivelul anterior de *r* ori, atunci un index multinivel cu *i* intrări, la primul nivel, va avea aproximativ *n* niveluri, unde $n = \lceil \log_r i \rceil$.

Indecşi cu dispersie

Se pot crea structuri de acces similare indecşilor, dar bazate pe dispersii. Astfel, intrările indexului (*K,Pr*) se pot organiza ca un fişier dispersat, care îşi va schimba dimensiunea, folosind dispersia dinamică, extensibilă sau liniară. Algoritmul aplică funcţia de dispersie asupra cheii *K*. Îndată ce este prezentată o intrare (o cheie), pointerul *Pr* se utilizează pentru localizarea înregistrării în fişierul de date.

Funcţia de dispersie asociată poate fi utilizată atât pentru organizarea fişierelor, cât şi pentru crearea structurii indexului.

28.5 Fişiere indexat secvenţiale

Fişierele indexat secvenţiale sunt un tip de organizare a fişierelor în care înregistrările sunt identificate de un index, numit cheie de acces şi fiecare cheie reprezintă valoarea cu care sunt identificate alte date ale înregistrării în fişier. Această cheie de acces trebuie să fie unică pentru fiecare înregistrare, adică nu sunt admise două înregistrări cu aceeaşi valoare a cheii. În afară de aceasta, toate fişierele (inclusiv indexul) sunt ordonate secvenţial după cheia de acces, care, de fapt, este cheia primară a fişierului cu date.

Structuri ale fişierelor indexat secvenţiale

Întrucât volumul datelor dintr-un fişier secvenţial poate fi foarte mare şi numărul de accesări (cu o executare joasă) înalt, se utilizează alte tehnici pentru manipularea eficientă a înregistrărilor. Astfel, se foloseşte o structură de acces, asociată la fişier, care oferă o mai mare eficienţă în localizarea înregistrării prezentate de argumentul de căutare, decât metodele aplicate în fişierele secvenţiale.

Un fişier indexat secvenţial este constituit dintr-un fişier secvenţial şi un index (structura de acces). Structura lui constituie trei spaţii de lucru: spaţiul pentru date (fişier principal), spaţiul rezervat înregistrărilor excedentare (fişier de tranzacţii) şi spaţiul pentru index (fişier index). Fişierul index poate fi organizat în niveluri, de la indexul principal (rădăcina), până la nivelul de acces la înregistrare.

Astfel, pentru accelerarea căutării înregistrărilor, fişierul indexat secvenţial utilizează fişierul de date şi fişierul index. Fişierul principal (de date) este de tip secvenţial, cu înregistrări grupate în blocuri şi ordonate de o cheie de ordonare. Fişierul index este format din perechi <*cheie, adresă*>. Adresa indică înregistrări sau blocuri în fişierul principal sau indică blocuri de înregistrări în propriul fişier index (formând un arbore). Adică, pot fi mai multe niveluri de indecşi, ei fiind stocaţi în acelaşi fişier fizic sau în fişiere diferite. Când se prelucrează blocurile, perechea <*cheie, adresă*> (fie bloc de date sau bloc de index) care arată spre următorul nivel conţine cheia ce corespunde înregistrării cu valoarea cea mai mare (variantă mai comună) în bloc.

Avantajul utilizării structurilor în blocuri constă în faptul că blocurile sunt citite şi încărcate în memoria centrală în întregime. Dimensiunea blocului trebuie determinată**,** ţinându-se cont de caracteristicile dispozitivelor fizice de lectură şi scriere şi limitele sistemului utilizat (responsabil de operaţiile de lectură şi scriere pe dispozitivele fizice).

O strategie bună este rezervarea în fiecare bloc al fişierului a unui spaţiu liber pentru înregistrările noi ce vor fi incluse. Astfel, la prima încărcare a fişierului de date şi la fiecare reorganizare a fişierului (rearanjare), este ocupată numai o parte din fiecare bloc. Pentru a evita umplerea rapidă a blocurilor, spaţiul liber este rezervat, ţinându-se cont de statisticile de includeri de înregistrări, iar reorganizarea este o procedură ce trebuie făcută periodic.

O alternativă mai potrivită este utilizarea unui singur spaţiu pentru înregistrările excedentare, deoarece nici o dată nu se poate prezice când vor fi completate unele blocuri. Şi dacă această problemă apare, ceva trebuie întreprins pentru a include o înregistrare nouă, care nu poate fi plasată imediat în poziţia adecvată în fişier (această operaţie necesită reorganizarea fişierului care se executată în momente speciale).

Pentru acumularea înregistrărilor excedentare, una din cele mai bune opţiuni este crearea unui fişier de tranzacţii (*overflow*). Acest fişier are aceeaşi structură ca şi fişierul de date, cu înregistrări suplimentate cu un câmp care permite înlănţuirea înregistrărilor. Înregistrările excedentare sunt incluse în fişierul de tranzacţii (prima poziţie vidă) şi formează o listă înlănţuită (crescător după cheia de ordonare).

Fişierul de tranzacţii poate fi utilizat în diverse moduri. O variantă este folosirea unei liste de înregistrări excedentare pentru fiecare bloc al fişierului de date, figura 28.10. De aceea, în fişierul principal există câte un pointer asociat fiecărui bloc. El indică o înregistrare în fişierul de tranzacţii, care este prima în lista de înregistrări excedentare ale blocului. Această listă poate fi formată din cheile mai mari decât cea mai mare din bloc (şi mai mici decât cea mai mică a blocului următor), sau, din chei mai mici decât cea mai mică din bloc (şi mai mari decât cea mai mare a blocului anterior). Aici trebuie ţinut cont că cea mai mare cheie a blocului nu poate fi exclusă fizic, deoarece este utilizată în nivelul intern al indecşilor.

Această variantă necesită reorganizarea înregistrărilor între blocul din fişierul de date şi lista de înregistrări excedentare, păstrând, după inserări şi suprimări, ordinea cheii în bloc şi în listă (deplasând înregistrările respective din bloc spre listă sau din listă spre bloc). Avantajul acestei structuri este o mai bună utilizare a spaţiului în blocuri, deoarece spaţiul din fişier ocupat de înregistrările excluse (logic) poate fi ocupat de alte înregistrări numai după reorganizare.

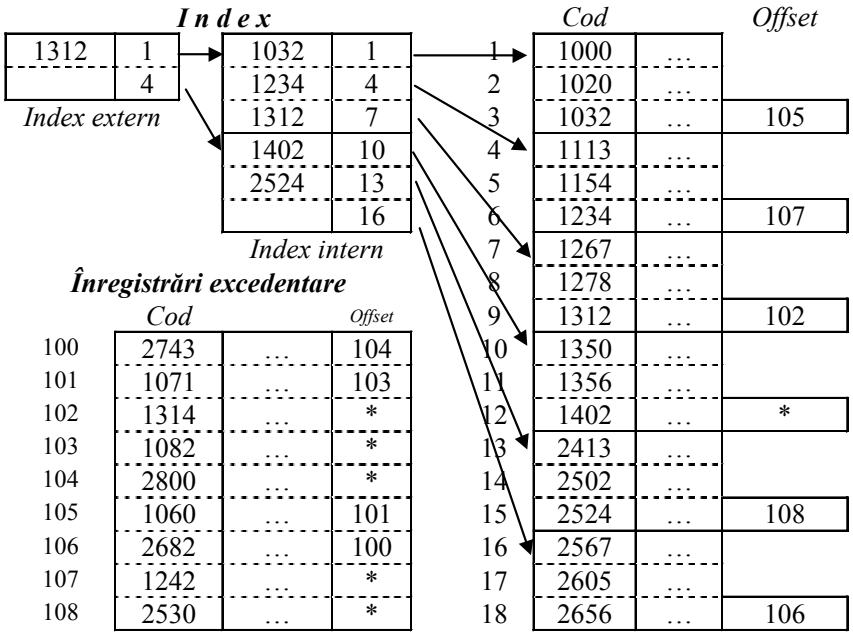


Figura 28.10 *Înregistrări excedentare asociate cu blocuri*

Există şi alternativa când listele cu înregistrări excedentare se asociază cu înregistrările din fişierul principal şi nu cu blocurile, figura 28.11. Astfel, fiecare înregistrare are câte un pointer nul sau care indică spre lista respectivă de înregistrări excedentare. Această listă este constituită din înregistrări cu chei mai mici decât a înregistrării asociate şi mai mari decât a înregistrării anterioare. În această structură, eliminările de înregistrări în blocuri vor fi făcute numai la nivel logic (cu includerea unei etichete).

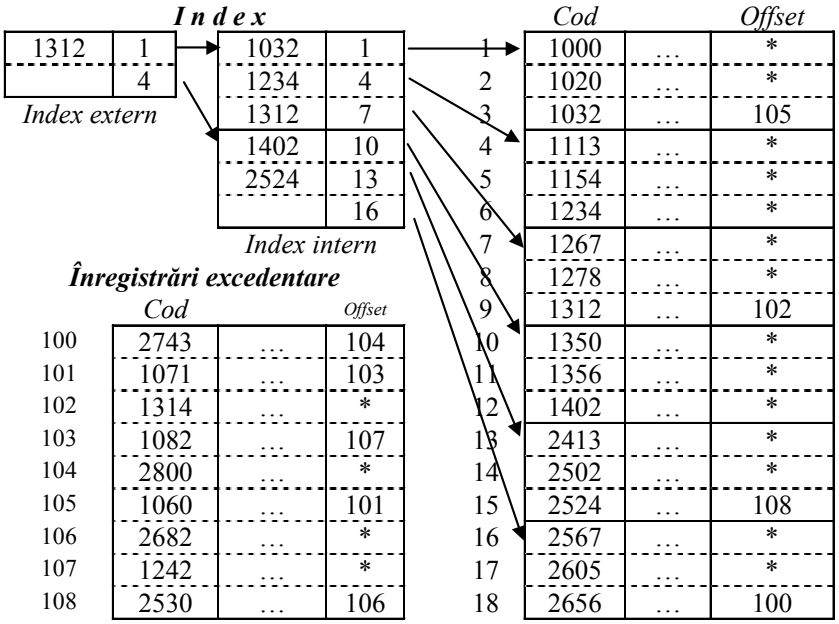


Figura 28.11 *Înregistrări excedentare asociate cu înregistrări*

Avantajele organizării indexat secvenţiale sunt cele ale organizării directe şi organizării secvenţiale. Adică, un avantaj al acestei organizări este posibilitatea de a îmbina consultarea rapidă a unei înregistrări (prin index) cu parcurgerea facilă secvenţială a tuturor înregistrărilor (după cheia de ordonare).

Cu toate acestea, structura nu este adecvată pentru consultarea după mai multe chei (în afară de cea de ordonare) şi pentru sistemele în care operaţiile de actualizare sunt făcute 24 ore pe zi, în toate zilele. În sistemele, unde sunt multe inserări (cu multe liste de înregistrări excedentare), are loc degradarea generală a executării operaţiilor din cauza deplasărilor de înregistrări între blocuri şi liste în momentul inserărilor sau eliminărilor (prima alternativă), sau din cauza consultărilor exhaustive în listele cu înregistrări excedentare (alternativa a doua).

Astfel, indexul trebuie actualizat încontinuu, pe măsură ce înregistrările sunt adăugate sau eliminate. În afară de aceasta, în raport cu organizarea directă, sunt necesare etape suplimentare pentru localizarea înregistrărilor - indexul trebuie să fie utilizat pentru orice accesare.

Fişierele indexat secvenţiale şi dispozitivele de stocare

Metoda indexat secvenţială presupune că înregistrările sunt stocate pe dispozitive cu acces direct, iar pentru operaţiile de acces direct există indecşi care permit căutarea adresei unei înregistrări individuale. În cazul în care se solicită accesul aleatoriu, folosind diverse chei de acces, trebuie construit un index pentru fiecare din ele. În general, indecşii au o structură ierarhică (pe niveluri), pentru o localizare rapidă a pistei care conţine înregistrarea căutată.

Pentru a profita de caracteristicile dispozitivelor secundare de stocare a datelor şi a face mai eficient procesul de căutare, este indicat să se construiască indecşi structuraţi, în mai multe niveluri, în acord cu nivelurile de organizare fizică a acestor dispozitive. Numărul de niveluri este

proporţional cu numărul de intrări (numărul de indecşi). Aceşti indecşi pot fi implementaţi, de exemplu, în formă de B-arbori, datorită flexibilităţii mari a acestei structuri în ceea ce priveşte operaţiile de inserare şi eliminare în fişierul index şi eficienţei de căutare pe care o oferă.

Astfel, fişierul index se construieşte structurat în acord cu nivelurile fizice ale modului de stocare pe discurile magnetice. Nivelurile fizice sunt discul, cilindrul, pista şi sectorul. Indexul de cilindru conţine pentru fiecare cilindru ocupat de fişier, o intrare ce indică cea mai mare valoare stocată a cheii de ordonare. Cu fiecare cilindru este legat un index de pistă, care indică, pentru fiecare pistă a cilindrului, cea mai mare valoare a cheii care o conţine. Indexul de pistă al unui cilindru este stocat pe cilindru. Indexul de nivel mai înalt, asupra indexului de cilindru, este numit index principal şi conţine, pentru fiecare disc (sau grup de cilindri) ocupat de fişier, valoarea cea mai mare a cheii stocate.

Indexul principal şi de cilindri, când este posibil, se păstrează în memoria principală, astfel ca, în procesul de căutare, cilindrul, unde se găseşte înregistrarea căutată, să fie identificat, fără accesarea discului. După obţinerea cilindrului, acesta este accesat pentru lectura indexului lui de piste, care permite localizarea pistei, unde se găseşte înregistrarea. Pista selectată este, atunci, citită şi în ea este localizată (de o metodă de căutare) înregistrarea solicitată. În acest proces, numai un cilindru este accesat. E important de menţionat că indexul de pistă nu precizează (nu are nevoie) conţinutul pentru fiecare înregistrare sau adresă a pistei, întrucât el este determinat de secvenţa în care apare la intrarea în index. Acelaşi lucru se întâmplă şi cu indexul cilindrilor.

Chiar dacă nu se manipulează direct structura *disc/cilindri/piste* este importantă structurarea indexului în niveluri, în primul rând, pentru că nivelurile externe (care nu sunt direct legate cu fişierul de date) antrenează puţine intrări, ceea ce permite să fie stocate în memoria principală, astfel devenind eficientă căutarea în indexul primar (numai un bloc), fapt ce, la rândul său, eficientizează căutarea fişierului de date (numai un bloc).

Operaţii cu fişiere indexat secvenţiale

Accesul aleatoriu la înregistrări este realizat, folosind indexul. Argumentul de căutare defineşte calea în index, care duce spre adresa înregistrării solicitate. Adresa obţinută de index poate fi adresa înregistrării sau adresa blocului care o conţine. În ultimul caz, pentru localizarea înregistrării, este necesară efectuarea unei căutări în blocul respectiv care, la rândul său, poate cere accesarea spaţiului rezervat înregistrărilor excedentare.

Fişierele indexat secvenţiale trebuie, de exemplu, să fie reorganizate când înregistrările excluse logic nu sunt refolosite de algoritmul de includere a înregistrărilor. În acest caz, spaţiul eliberat de înregistrările excluse trebuie să fie ocupat după reorganizarea fişierului, care presupune şi o reorganizare completă a fişierului index, deoarece poziţiile fizice ale unor înregistrări vor fi modificate.

Deschiderea şi închiderea fişierelor. Deschiderea unui fişier indexat secvenţial implică identificarea fişierului de date, cheilor de acces pentru care există indecşi şi care sunt fişierele index, fără a cunoaşte modul cum sunt structurate acestea, câte niveluri are fişierul index asociat. De multe ori, indecşii pot fi creaţi numai când se utilizează fişierul (adică indexul nu e

stocat pe disc). Prin urmare, întotdeauna când un fişier este deschis, trebuie să se creeze în memorie structurile respective de acces.

Închiderea fişierului este, de obicei, o sarcină simplă şi depinde de faptul cum se lucrează cu el. Dacă indecşii sunt manipulaţi numai în memoria principală, fişierul de date se poate închide fără nici o operaţie adiţională. Cu toate că manipularea indecşilor are loc în memoria primară, deseori (cazul când această structură este modificată), este necesară salvarea fişierului index pe disc, or generarea fişierelor index poate fi o sarcină foarte retardă. În acest caz, este important să se cunoască care indecşi sunt temporari şi care indecşi sunt permanenţi. De asemenea, trebuie prelucrat spaţiul de înregistrări excedentare, pentru că se recomandă ca fişierul închis să aibă spaţiul de tranzacţii liber. Prin urmare, mai întâi se face reorganizarea fişierului, apoi se închide.

Consultarea înregistrărilor. Căutarea unei înregistrări după cheia de ordonare este făcută mai întâi în fişierul index (pornind de la nivelul extern). Se caută cheia imediat mai mare sau egală cu cea solicitată. Cu adresa obţinută se caută în nivelul următor al indexului, citind un bloc de înregistrări din index. În acest bloc, se caută cea mai mare cheie. Se repetă căutarea până nu mai există niveluri de indecşi şi adresa obţinută corespunde unui bloc de înregistrări în spaţiul de date al fişierului principal. Apoi se citeşte acest bloc, căutând cheia cerută. Dacă este găsită, trebuie să se analizeze eticheta de eliminare (dacă eticheta există, atunci cheia nu mai există). Dacă cheia nu se întâlneşte în acest bloc, se verifică dacă se află în lista de înregistrări excedentare, fie în bloc sau înregistrare. Dacă este, se ia adresa indicată (corespunzătoare unei poziţii în fişierul de tranzacţii) şi se parcurge lista în căutarea cheii cerute. În general, în fişierul de tranzacţii nu se lucrează cu etichete de eliminare, deoarece aici eliminările sunt, de obicei, fizice.

Consultarea unei înregistrări după cheia care nu e de ordonare necesită o căutare exhaustivă a tuturor înregistrărilor atât în blocuri, cât şi în listele de tranzacţii.

Inserarea înregistrărilor. Pentru inserarea unei înregistrări, iniţial se caută în fişierul index şi se determină în ce bloc de date înregistrarea trebuie să fie inclusă. Îndată după determinarea poziţiei, înregistrarea este inclusă în spaţiul de înregistrări excedentare şi se actualizează legăturile, dacă spaţiul de tranzacţii există pentru fiecare înregistrare.

Dacă spaţiile de înregistrări excedentare sunt rezervate blocurilor, înregistrarea este inserată în blocul selectat în spaţiul rezervat pentru date. În cazul când acesta este plin sau nu există spaţiu pentru inserarea înregistrării respective, astfel ca înregistrările acestui fişier să păstreze ordinea secvenţială, este inserată în spaţiul de tranzacţii al blocului, dacă cheia sa este mai mare decât cheia ultimei înregistrări a blocului principal şi se actualizează legăturile.

În caz contrar, ultima înregistrare a blocului principal este trecută în spaţiul de tranzacţii şi înregistrarea nouă este inserată în poziţia sa corectă din blocul principal (pot fi deplasări înăuntru blocului). Poate exista un câmp similar, în acest spaţiu, care ar indica următoarea adresă în spaţiul de date sau de tranzacţie.

În unele implementări inserările pot fi realizate în blocuri, în spaţiile eliberate de înregistrările care au fost excluse sau în spaţiile rezervate pentru aceasta la sfârşit. În acest caz, spaţiile pentru înregistrările excedentare pot să nu fie folosite. Aici, sistemul localizează în index blocul

unde va fi stocată înregistrarea. După această operaţie, se găseşte, în fişierul de date, blocul dezirabil şi se include înregistrarea în poziţia corectă, adică în poziţie ordonată.

Înregistrarea este scrisă pe prima adresă liberă din fişier şi sunt actualizate poziţiile relative ale acestei adrese (cheia înregistrării, adresa înregistrării), apoi e scrisă în index (indecşi). Înregistrarea nouă poate fi stocată pe orice adresă vidă din spaţiul alocat pentru fişier. În continuare, trebuie inclusă o intrare în fişierul index primar (ce se referă la cheia primară) şi în fiecare din celelalte fişiere index care există. Inserările în fişierele index atrag după sine restructurarea lor pentru ca înregistrarea să fie scrisă în poziţia adecvată în fişierul index.

Excluderea înregistrărilor. Ca şi în alte organizări analizate anterior, în fişierele indexat secvenţiale se pot realiza atât excluderi fizice, cât şi logice. Excluderea fizică presupune eliberarea spaţiului în fişierul de date (poate dura mult timp), precum şi eliminarea înregistrării din fişierul de index.

O soluţie mai bună, în majoritatea cazurilor, este excluderea logică. În acest caz, se poate opta pentru mai multe variante. Este posibilă includerea unui câmp în fişierul de date care ar indica dacă înregistrarea a fost exclusă. Astfel, nu e necesară modificarea indexului, însă devine obligatorie verificarea acestui câmp la fiecare accesare a înregistrării.

Altă opţiune este utilizarea în index a unui indicator de excludere asociat cheii primare. În acest mod, la accesarea fişierului index deja se va putea verifica dacă înregistrarea a fost exclusă. În afară de aceasta, înregistrarea ar putea fi detectată de algoritmul de inserare, care ar localiza uşor spaţiile eliberate pentru inserări.

Dacă fişierul posedă mai multe structuri index (asupra diferitelor chei de acces), sarcina de excludere a înregistrărilor se poate complica. În orice caz, pentru orice structură index, înregistrarea exclusă nu trebuie luată în seamă.

Modificarea înregistrărilor. Pentru modificarea înregistrărilor, mai întâi, se localizează înregistrarea ca într-o consultare (prin structura index), folosind un argument de căutare în calitate de cheie de acces. În continuare, se modifică câmpurile respective şi se înscrie înregistrarea.

Pot fi întâlnite două situaţii. Dacă nu se modifică câmpul cheii de ordonare şi lungimea înregistrării este mai mică, înregistrarea poate fi scrisă în locul în care se găseşte. Dacă modificarea datelor înregistrării cere modificarea poziţiei ei în fişier, adică are loc modificarea unui câmp al cheii de ordonare sau a lungimii ei, atunci înregistrarea se exclude din fişierul iniţial şi se inserează, în calitate de înregistrare nouă, în fişierul de tranzacţii.

Astfel, sistemul de operare localizează în index adresa blocului unde poate fi stocată cheia căutată. Dacă blocul este găsit în fişierul de date, înregistrarea este căutată secvenţial. Modificarea unei înregistrări este făcută cu o nouă înscriere a înregistrării actualizate, în cazul când nu se modifică dimensiunea ei, pe aceeaşi adresă. Iar în cazul când lungimea este modificată, înregistrarea veche este exclusă, iar cea nouă este inserată pe altă adresă, unde înregistrarea nouă poate fi scrisă.

Modificarea unei înregistrări poate provoca modificarea indecşilor, deoarece aceasta poate modifica valoarea câmpurilor pentru care există indecşi. Acest fapt implică eliminarea intrărilor anterioare în fişierul de index şi în includerea unei intrări noi, ce va implica o reorganizare a indexului. De

obicei, nu se admite modificarea cheii specificate pe câmpurile cheii primare.

Lectura exhaustivă a înregistrărilor. Lectura exhaustivă este realizată fără utilizarea indexului. Se citeşte fiecare înregistrare din fişierul principal şi continuă această lectură, folosind lista de pointeri ce leagă fişierul principal de cel de tranzacţii, în mod analogic, ca în fişierul secvenţial.

Însă, din cauza accesării spaţiului de înregistrări excedentare, această operaţie este mai puţin eficientă, în comparaţie cu operaţia respectivă în fişierul secvenţial. Ordinea fizică şi logică a înregistrărilor nu este aceeaşi. Deci, pentru a realiza eficient lectura exhaustivă a fişierului, este necesară definirea unui index asupra acestui fişier care ar determina ordinea logică de parcurgere a înregistrărilor. Dacă există cel puţin un index exhaustiv asupra fişierului, această operaţie poate fi efectuată prin accesul serial la fişierul index, urmată de accesări directe ale fişierului de date.

Reorganizarea fişierelor. Reorganizarea presupune o lectură exhaustivă şi transferarea tuturor înregistrărilor într-un fişier nou de date. De fapt, toate înregistrările sunt transferate în fişierul principal şi spaţiul de tranzacţii este eliberat. În continuare, se generează indexul nou.

Reorganizarea unui fişier indexat secvenţial devine întotdeauna necesară, dacă: spaţiul de tranzacţii devine foarte mare, diminuând, astfel, eficienţa executării accesărilor de fişiere. În afară de aceasta, are loc eliminarea înregistrărilor excluse logic (dacă aceste sectoare nu au fost folosite de algoritmul de inserare a înregistrărilor). Dacă trebuie, se restructurează fişierele index pentru îmbunătăţirea accesului la înregistrări.

Astfel, reorganizarea fişierelor constă în lectura tuturor înregistrărilor şi scrierea lor în alt fişier de date. Totodată, se eliberează spaţiile ocupate de înregistrările excluse şi se generează o structură nouă de indecşi.

Operaţia de reorganizare trebuie făcută periodic când sistemul nu este utilizat (cel puţin când nu sunt făcute inserări, modificări şi suprimări), deoarece reorganizarea necesită o analiză exhaustivă a înregistrărilor în două fişiere (principal şi de tranzacţii) şi crearea unui fişier auxiliar cu aceeaşi structură, ca a fişierului principal.

28.6 Fişiere indexate

În fişierele indexate, înregistrările sunt identificate de un index, numit cheie de acces, sau cheie a înregistrării, sau cheie principală. Fiecare cheie reprezintă valoarea cu care celelalte date din înregistrare sunt identificate în fişier. Această cheie de acces trebuie să fie unică pentru fiecare înregistrare, adică, nu există două înregistrări cu aceeaşi valoare a cheii. Înregistrările sunt accesate întotdeauna de unu sau mai mulţi indecşi, care nu au nici o legătură cu ordinea de stocare fizică a înregistrărilor.

Păstrarea şi respectarea ordinii fizice a înregistrărilor (pentru un acces serial eficient) în fişierele indexat secvenţiale provoacă mai multe probleme, în principal, probleme care ţin de inserarea înregistrărilor noi. Pe măsură ce necesitatea de accesări seriale scade, în raport cu numărul de accesări aleatorii, păstrarea secvenţei fizice a fişierului afectează negativ eficienţa accesării bazei de date, adică organizarea fişierelor devine contraproductivă.

Prin urmare, devine mai convenabilă utilizarea unui fişier indexat, în care înregistrările sunt accesate întotdeauna prin unu sau mai mulţi indecşi,

fără a ţine cont de ordinea fizică a înregistrărilor. Indecşii pot fi structuraţi ca în organizarea indexat secvenţială, ceea ce face mai eficientă căutarea înregistrării.

Structuri şi operaţii în fişierele indexate

Fişierele indexate, ca şi fişierele indexat secvenţiale, utilizează structuri auxiliare (indecşi) pentru accelerarea accesului la înregistrările cu date. Spre deosebire de precedentele organizări, în fişierele indexate există posibilitatea de utilizare a mai multor indecşi (respectiv, mai multor chei) şi lipseşte restricţia că înregistrările în fişierul principal trebuie să fie ordonate de o cheie. Ultima caracteristică exclude utilizarea spaţiilor de tranzacţii.

Astfel, fişierul cu organizare indexată este constituit din două componente:

- indexul, care conţine cheile de acces şi adresele fiecărei înregistrări din fişierul de date; toate accesările de înregistrări sunt făcute prin această componentă, care este păstrată întotdeauna ordonat, după cheia de acces;
- fişierul de date, care stochează celelalte câmpuri ale tuturor înregistrărilor fişierului, independente de orice ordonare sau secvenţă.

Ca şi fişierele indexat secvenţiale, fişierele indexate susţin două metode de acces: secvenţial şi aleatoriu.

Modul secvenţial de acces reprezintă o lectură secvenţială a indexului unui fişier indexat. Pentru fiecare intrare în index, este obţinută adresa înregistrării din index în fişierul de date şi cu această adresă se citeşte înregistrarea. În recuperarea secvenţială a unui fişier, datele sunt citite întotdeauna ordonate după cheie.

Modul aleatoriu de acces se caracterizează prin inexistenţa unei secvenţe logice pentru recuperarea înregistrărilor, adică, după citirea ultimei înregistrări, poate fi citită a cincea sau prima, sau oricare alta - această secvenţă este determinată de programul aplicaţie. În procesul de lectură, sistemul localizează valoarea cheii dezirabile în index, identificând adresa înregistrării corespunzătoare în fişierul de date, şi asigură accesul, cu această adresă, pentru recuperarea înregistrării.

Consultarea fişierelor indexate poate fi realizată, folosind mai multe chei. Dacă există un index pentru o cheie, căutarea este făcută, pornind de la indexul în chestiune (prelucrarea ca în organizarea indexat secvenţială) până se obţine adresa unei înregistrări în fişierul de date. Nu este nevoie de multe liste (nu mai sunt accesări). Însă, dacă excluderile înregistrărilor sunt logice, este necesară examinarea existenţei etichetelor de excludere. Dacă nu există vreun index pentru cheia cerută, se realizează căutarea exhaustivă a cheii în fişierul de date (analiza tuturor înregistrărilor până ce se întâlneşte cea căutată).

Inserările sunt făcute întotdeauna în fişierul de date, în orice spaţiu liber, deoarece înregistrările în acest fişier nu trebuie să respecte o anumită ordine. Pentru a putea reexploata spaţiile înregistrărilor suprimate, se construieşte o listă de spaţii libere. Însă, fiecare inserare implică şi actualizarea imediată (*on-line*) a indecşilor, fapt ce poate influenţa negativ executarea acestei operaţii.

Modificările sunt făcute direct în fişierul de date şi ele, de asemenea, provoacă actualizarea indecşilor (numai în care se indică câmpurile modificate).

Excluderea înregistrărilor poate fi fizică, cu actualizarea *on-line* a indexului, sau logică, cu utilizarea etichetelor. Ultima variantă implică, ulterior, ştergeri fizice cu scopul reutilizării spaţiului eliberat.

Parcurgerea secvenţială este puţin ineficientă, dar nu influenţează executarea sistemului în general. Ordinea înregistrărilor depinde de cheia aleasă şi este dictată de informaţiile din index (nivelul cel mai intern conţine ordinea înregistrărilor). Problema este că fiecare pereche <*cheie, adresă*>, în ultimul nivel direcţionează accesul la fişierul de date (deoarece înregistrările sunt dispersate fără ordine şi nu sunt utilizate blocuri de înregistrări continue). Pentru parcurgerea secvenţială, este de ajuns analizarea directă a ultimului nivel al indexului (nu este necesară navigarea, pornind de la nivelurile externe). Parcurgerea secvenţială după o cheie care nu este index necesită un fişier auxiliar şi tehnici de ordonare.

Un mare avantaj al fişierelor indexate este posibilitatea de consultare după mai multe chei, ceea ce deosebeşte această organizare de celelalte. Consultările aleatorii (pentru o singură înregistrare), de asemenea, sunt făcute rapid. Acum operaţia de parcurgere secvenţială, cu toate că nu poate fi făcută în mod optimal, nu este foarte lentă ca operaţia respectivă în organizarea indexat secvenţială, întrucât nu trebuie să se parcurgă liste înlănţuite.

Un dezavantaj este necesitatea de actualizare *on-line* a indecşilor, ceea ce poate influenţa executarea operaţiilor, dar, pe de altă parte, permite aplicarea lor în sistemele cu utilizare neîntreruptă, cum sunt bazele de date. Însă, o dată cu implementarea indecşilor eficienţi şi moderni (precum se va vedea în continuare), se poate soluţiona această problemă. Alt dezavantaj este că organizarea indexată cere mult spaţiu pentru stocarea tuturor indecşilor. Cu toate acestea, fişierele indexate sunt larg utilizate în majoritatea SGBD-urilor.

Fişiere inversate

Nu toate căutările în bazele de date se fac după cheia primară. Dacă un atribut sau un grup de atribut apare des în cereri, atunci, pentru acest atribut sau grup de atribute, se construieşte un index secundar, ce permite accesul rapid la înregistrările corespunzătoare valorilor date.

Un fişier cu un index secundar, corespunzător unui atribut sau grup de atribute *X*, se spune că este *fişier inversat* în raport cu *X*. În indexul secundar, înregistrările sunt indicate fie prin pointeri la ele, fie prin valorile cheilor primare corespunzătoare lor.

Referirea la înregistrări prin pointeri (listă inversată) are avantajul accesului mai rapid la date, dar produce constrângeri din cauza fixării înregistrărilor în locul unde au fost introduse, la nivel de bloc sau la nivel de înregistrări. Referirea prin cheia principală asociată are dezavantajul unui acces lent, dar nu mai fixează înregistrările.

De exemplu, o listă inversată după câmpul *Departament* al unui fişier de funcţionari ai unei întreprinderi va conţine toate valorile acestui câmp, şi fiecăreia din aceste valori îi va fi asociată o listă cu adrese sau pointeri ale înregistrărilor funcţionarilor care conţin valoarea specificată în câmpul

Departament. Astfel, lungimea fiecărei liste va fi numărul de funcţionari care lucrează în departamentul respectiv.

De fiecare dată când o înregistrare este inclusă, exclusă sau modificată, fişierul inversat, de asemenea, trebuie să fie modificat. O alternativă, precum s-a menţionat, este stocarea nu a adresei, ci a cheii primare a fiecărei înregistrări. În acest mod, se evită actualizarea indexului când are loc reorganizarea fizică a fişierului (ordinea înregistrărilor). Însă, această opţiune frânează executarea consultărilor, o dată ce nu se cunoaşte adresa, este necesară localizarea ei în alt mod (sau de alt index, sau secvenţial).

O problemă ce diminuează calităţile necesare pentru viabilitatea fişierelor inversate apare când valorile câmpului pe care e definit indexul secundar nu posedă repetări de valori. Altă problemă ce trebuie să fie considerată este că listele, în general, nu au lungime fixă (numărul de înregistrări pentru fiecare valoare se poate schimba cu utilizarea sistemului).

Indecşi Bitmap

Un index *bitmap* este o variantă a indexului secundar care asociază fiecărei valori a atributului indexat o secvenţă de biţi, în loc de o listă de adrese sau chei. Fiecare bit identifică o înregistrare concretă a fişierului. Dacă înregistrarea în câmpul atributului indexat are valoarea specificată, atunci bitul respectiv conţine valoarea 1. Un bit cu valoarea 0 semnifică faptul că înregistrarea nu conţine valoarea specificată în câmpul indexat.

Lungimea secvenţei de biţi este numărul maximal de înregistrări în fişier. Dacă fişierul creşte mult sau are loc reorganizarea fizică a înregistrărilor, indexul *bitmap* trebuie să fie refăcut.

Dacă se caută înregistrările cu valoarea *v* a atributului indexat, este suficient să fie examinat *bitmap*-ul asociat valorii *v*, căutaţi toţi biţii cu 1 şi să fie accesate înregistrările respective. Un index *bitmap* este foarte eficient, dacă numărul de valori posibile ale atributului indexat este mic.

Unul din avantajele indecşilor *bitmap* este că nu ocupă mult spaţiu. Un index *bitmap* este foarte mic în dimensiuni, în raport cu un B-arbore construit pe acelaşi atribut. Este foarte util în aplicaţii de tipul înmagazinărilor de date care conţin volume mari şi clasifică informaţiile conform unor atribute definite pe domenii mici de valori. Unele interogări pot fi executate foarte eficient, uneori fără a apela la fişierul de date.

O dată ce o înregistrare este inclusă, exclusă sau modificată (în câmpul respectiv), *bitmap*-ul, de asemenea, trebuie modificat. Acest tip de index poate fi folosit eficient şi pentru câmpurile atributelor multivaloare.

Arborii binari

Arborii, în general, formează o clasă de indecşi destul de eficienţi şi mult utilizaţi, astăzi, în bazele de date. Fiecare nod trebuie să posede o cheie şi pointeri (vizi sau îndreptaţi spre nodurile fii).

Un arbore important este arborele binar. În el, fiecare nod poate avea cel mult doi fii. În calitate de index arborele binar poate fi organizat precum urmează. Dat fiind faptul că fiecare nod posedă o cheie, nodul fiu din stânga trebuie să conţină o cheie mai mică decât a nodului părinte, în timp ce

nodul fiu din dreapta trebuie să conţină o cheie mai mare (şi aşa succesiv până la frunze).

Dacă se caută o înregistrare, localizarea ei se face, analizând nodurile fii. Astfel, pentru găsirea unei chei, se parcurge arborele de la rădăcină, mergând spre stânga, dacă cheia căutată este mai mică decât cheia nodului curent, sau spre dreapta dacă cheia căutată este mai mare (până se întâlneşte cheia căutată sau până când nu mai sunt noduri fii).

Inserarea cheii în arbore este făcută după o anumită frunză (în calitate de nod fiu al unui nod frunză), în locul respectiv, aplicând aceleaşi proceduri ca în procesul de căutare (explicate anterior). Excluderea, însă, a unei chei presupune substituirea ei cu alta (mai mică sau mai mare decât următoarea, care se găseşte în nodul frunză respectiv). Modificarea valorii cheii indexului este realizată în arbore de o excludere a cheii vechi şi o includere a cheii noi.

Sunt două soluţii de stocare a adreselor înregistrărilor:

- adresa se plasează în fiecare nod al arborelui, formând perechea <cheie, adresă>;
- adresa se păstrează numai în nodurile frunze, adică unele chei sunt repetate şi în calitate de noduri frunze.

Soluţia când fiecărei chei îi este asociată o adresă permite accesarea înregistrării în momentul în care s-a găsit cheia ei în arbore şi, prin urmare, nu este necesară parcurgerea arborelui până la nodurile frunze. A doua variantă, când adresele sunt asociate numai nodurilor frunze, facilitează consultarea secvenţială a înregistrărilor (după cheia indexului), deoarece nodurile frunze pot fi înlănţuite, formând o listă.

Arborele binar este puţin folosit în calitate de index în SGBD-uri, deoarece după o serie de operaţii de actualizare poate deveni nebalansat. Chiar dacă cheia unui nod rădăcină are o valoare intermediară, arborele poate avea unele ramuri mai lungi sau mai adânci decât altele, fapt ce poate genera un dezechilibru în executarea operaţiilor. Evident că acest lucru poate fi depăşit, dacă se va recurge permanent la reorganizarea arborelui. Dar această operaţie este laborioasă şi sunt antrenate în ea toate nodurile arborelui. Cu atât mai mult că baza de date este un sistem care este interogat încontinuu şi nu există răgaz pentru a face astfel de restructurări.

B-arbori

B-arborele este un tip de index în formă de arbore, care întotdeauna este balansat. Căutarea înregistrărilor este similară pentru orice cheie, indiferent de localizarea ei în arbore (fie în nodul rădăcină, sau în nodul intermediar sau frunză).

Fiecărui *B-arbore* i se asociază un număr numit ordinul arborelui, care determină numărul maximal şi numărul minimal de noduri frunze pe care le poate conţine. În afară de aceasta, un *B-arbore* (de ordinul *m*) trebuie să respecte următoarele reguli:

- fiecare nod posedă m sau mai puţini subarbori;
- fiecare nod, cu excepţia rădăcinii, posedă m/2 (ia o valoare întreagă) sau mai mulţi subarbori;
- nodul rădăcină posedă, cel puţin, doi subarbori nevizi, cu excepţia când este un nod frunză;

- orice nod frunză se găseşte la aceeaşi distanţă de nodul rădăcină şi toţi subarborii lor sunt vizi;
- într-un nod cu k subarbori se stochează k-1 înregistrări;
- orice nod de derivare (care nu este frunză) posedă exclusiv subarbori nevizi.

Consultarea după cheie este făcută ca în orice arbore: se parcurge subarborele stâng, dacă cheia căutată este mai mică decât cheia analizată (menţionăm că un nod poate avea mai mult de o cheie) sau se parcurge subarborele drept, dacă cheia este mai mare. Acest proces continuă până se întâlneşte cheia căutată sau se întâlneşte subarborele vid).

Pentru a insera o cheie într-un B-arbore, se verifică dacă argumentul de căutare nu coincide cu o cheie deja existentă. Adică, mai întâi, se face consultarea arborelui.

Includerea unei chei este făcută într-un nod frunză, dacă acesta are spaţiu suficient (nodul se găseşte în limitele date de ordinul arborelui). Dacă nu este spaţiu suficient pentru inserarea cheii, trebuie să se producă divizarea nodului (*splitt*) şi formarea a două noduri noi. Cheia valorii intermediare trebuie să urce, adică trebuie să fie inclusă în nodul părinte, cu corecţiile corespunzătoare în pointerii precedent şi următor, din această cheie, în nodul părinte (acum pointerii devin indicatori spre nodurile create). Nodul nou din stânga va conţine cheile mai mici decât cea intermediară şi nodul din dreapta, cheile mai mari.

Dacă în nodul părinte nu este spaţiu pentru includerea cheii intermediare, care "urcă", nodul părinte trebuie divizat (aceeaşi procedură). Acest proces poate provoca divizări succesive, până la crearea unei noi rădăcini (din inexistenţa unui nod părinte, cheia care urcă formează un nou nod rădăcină).

Excluderea unei chei este un proces mai complicat. Mai întâi se localizează cheia respectivă. Acţiunile de mai departe depind de faptul dacă cheia se găseşte într-un nod frunză sau nu.

Dacă cheia se găseşte într-un nod frunză, ea este exclusă. Rămâne să se verifice dacă nodul frunză se găseşte în interiorul limitelor admise. Dacă numărul de chei este mai mic de limita permisă de ordinul arborelui, nodul afectat trebuie să "solicite" o cheie de la nodul frate din stânga sau nodul frate din dreapta (dacă din unul din noduri poate fi extrasă o cheie, iar nodul respectiv să se încadreze în limitele admise). Cheia solicitată de la un nod frate trebuie să fie una de vârf (cea mai mare în nodul frate din stânga sau cea mai mică în nodul frate din dreapta). Atunci, această cheie se ridică la nodul părinte, substituind-o pe cea care se găsea între pointerii celor doi fraţi afectaţi. Cheia substituită este coborâtă în nodul de unde a fost exclusă cheia specificată (cheia coborâtă este poziţionată în ordinea respectivă).

Dacă, însă, nici un nod frate nu poate "elibera" o cheie pentru nodul din care s-a exclus cheia specificată, se produce concatenarea (*unsplitt*) între nodul afectat şi un nod frate. Atunci, cheia nodului părinte care se găseşte între identificatorii fraţilor coboară pentru a fi cheie intermediară între cheile nodurilor fraţi.

Dacă nodul părinte, după pierderea unei chei, conţine mai puţine chei decât limita admisă, el trebuie, de asemenea, unit la unul din fraţii săi, aplicând aceeaşi procedură de concatenare. Acest proces poate provoca o concatenare în cascadă, până este eliminat nodul rădăcină (rădăcina nouă fiind nodul obţinut din concatenarea nodurilor fiu).

În cazul când cheia ce trebuie exclusă se găseşte într-un nod de derivare, aceasta va trebui să fie substituită de cheia imediat mai mare sau mai mică. Or, pentru aceasta e destul de parcurs unul din subarbori până la nodul frunză mai apropiat. Dacă nodul frunză, după pierderea cheii (pentru substituirea celei excluse), conţine mai puţine chei decât limita admisă, procedura de corecţie este aceeaşi ca şi în cazul excluderii cheii din nodul frunză.

Trebuie menţionat că datele înregistrărilor (câmpurile) sunt păstrate într-un fişier separat. Aşadar, fiecare nod are, în afară de pointeri spre nodurile fii, un pointer spre înregistrarea respectivă de date.

O alternativă este păstrarea în noduri şi a datelor înregistrărilor. Avantajul acestei structuri este că nu mai e nevoie de încă o accesare pentru găsirea înregistrărilor. Pe de altă parte creşterea dimensiunii nodurilor face ca operaţiile de divizare şi concatenare să fie mai lente.

B^+-arbori

Indecşii de tipul *B+-arbore* sunt asemănători celor de tipul B-arbore, cu deosebirea că toate cheile din nodurile derivate sunt, de asemenea, replicate şi în nodurile frunze. Aceasta dă posibilitatea ca nodurile frunze să fie înlănţuite, formând o listă (fiecare nod are câte un pointer pentru nodurile precedent şi următor). Această listă facilitează operaţia de parcurgere secvenţială a înregistrărilor (după cheia indexului).

Câştigul în urma aplicării B+-arborelui este cu atât mai spectaculos, cu cât numărul înregistrărilor creşte. Se poate observa o creştere foarte rapidă (exponenţială) a numărului de înregistrări indexate în funcţie de numărul nivelurilor indexului şi invers, o creştere foarte mică (logaritmică) a numărului de niveluri în funcţie de numărul înregistrărilor. Întrucât costul unei căutări după cheie este proporţional cu numărul nivelurilor şi nu cu numărul înregistrărilor, indexarea permite micşorarea considerabilă a timpului de căutare.

B-arborii oferă o mai mare flexibilitate de găsire a adresei în fişierul de date, unde este stocată înregistrarea şi asigură o mai mare eficienţă, în principal, în operaţiile de inserare şi eliminare a înregistrărilor. În plus, B+-arborii nu exclud posibilitatea de parcurgere secvenţială a datelor. Pentru aceasta, este suficient să se parcurgă mulţimea de noduri frunze, folosind lanţul de pointeri ce le leagă. Deci, accesul la fişierele cu un index B+-arbore poate fi făcut în mod secvenţial şi aleatoriu.

Totodată, se observă o simplificare în structura generală a fişierului, deoarece fişierele indexate sunt scutite de utilizarea spaţiului de tranzacţii. Dacă fişierele organizate secvenţial şi indexat secvenţial trebuiau reorganizate periodic pentru a aduce ordinea logică la cea fizică, în fişierele indexate, ordinea fişierului de date este neesenţială. Aceasta se datorează faptului că înregistrările sunt accesate prin index, fără a avea de-a face cu configuraţia fizică a acestora.

Cu toate acestea, o mare problemă întâlnită în fişierele cu organizare de tip indexat este necesitatea de actualizare a tuturor indecşilor când, de exemplu, o nouă inserare este făcută.

28.7 Fişiere cu dispersie

Într-un fişier cu dispersie există o relaţie predefinită între valoarea cheii utilizate pentru identificarea unei înregistrări şi localizarea înregistrării în fişier. Înregistrările nu sunt, neapărat, ordonate fizic în acord cu valoarea cheilor. Înregistrările sunt stocate pe dispozitivele externe de stocare de acces direct şi recuperate prin utilizarea acestei legături.

Acesta este tipul de organizare care conţine numai un spaţiu de date pentru stocarea înregistrărilor. Înregistrările sunt identificate de o cheie principală, a cărei adresă fizică, pentru stocare, este dată de valoarea cheii sau de o valoare calculată, pornind de la valoarea cheii. Într-un fişier cu organizare dispersată, de obicei, cheia este de tip numeric, pentru a calcula adresa şi localiza înregistrarea în spaţiul de date.

Caracteristicile fişierelor cu dispersie

Fişierele cu dispersie prezintă o organizare a datelor destul de des utilizată în tehnicile de proiectare fizică a bazelor de date, mai ales în modelele orientate pe obiecte. Ele necesită timp, relativ, redus de acces al înregistrărilor individuale, însă lectura secvenţială a unui fişier cu dispersie este o sarcină anevoioasă. Există două forme de organizare a fişierelor cu dispersie:

- cu adresare directă;
- cu adresare indirectă.

Caracteristica principală a fişierelor cu dispersie este că adresa fiecărei înregistrări e determinată în funcţie de valoarea cheii primare a înregistrării. Astfel, este utilizată o *funcţie de dispersie* (*hash function*), care, pentru fiecare valoare a cheii *C*, returnează adresa *h(C)* într-un spaţiu de stocare, unde înregistrarea respectivă trebuie plasată.

Aceasta este cea mai rapidă metodă de acces la date, numai dacă căutarea se petrece cu condiţia de egalitate asupra câmpului de dispersie.

Funcţia de dispersie trebuie să ţină cont de spaţiul alocat fişierului, pentru a realiza o distribuire uniformă a înregistrărilor în spaţiul rezervat. Există funcţii care păstrează ordinea (cheii) şi care nu păstrează ordinea (cele mai populare). Între ultimele există funcţiile deterministe (o singură adresă poate fi generată de o cheie) şi funcţii nedeterministe (când mai mult de o cheie poate genera aceeaşi adresă, fapt care se numeşte coliziune).

Pentru includerea în fişier a unei înregistrări, este suficient să se calculeze adresa ei, aplicând valoarea cheii în funcţia de dispersie aleasă. Dacă apare o coliziune, adică, dacă există o altă înregistrare pe aceeaşi poziţie, trebuie să fie utilizate unele proceduri de tratare a coliziunilor.

Pentru excluderea unei înregistrări, este de ajuns să se găsească înregistrarea prin aceeaşi metodă şi să se excludă datele ei sau să se marcheze că e exclusă. Procesul de modificare se petrece în mod analog. Pentru modificări în cheia primară se apelează la operaţia de excludere (a cheii vechi) urmată de o operaţie de includere (a cheii noi).

Există eventualitatea ca funcţia de dispersie să producă, pentru două sau mai multe valori diferite ale cheii, aceeaşi adresă. În acest caz, prima înregistrare inclusă rămâne în domeniul de date, iar pentru celelalte se

apelează la o procedură de prelucrare a coliziunilor pentru a fi incluse în spaţiul de coliziuni. Există mai multe proceduri de tratare a coliziunilor:

- utilizarea unui fişier de tranzacţii (overflow), unde sunt plasate înregistrările excedentare (care nu pot fi incluse în fişierul principal, deoarece, deja, există altă înregistrare pe aceeaşi adresă), formând o listă înlănţuită;
- alocarea înregistrării în poziţia următoare liberă (fapt ce poate mări probabilitatea apariţiei coliziunilor);
- înregistrările care se ciocnesc sunt alocate în locurile libere ale aceluiaşi fişier, formând liste înlănţuite;
- utilizarea unei a doua funcţii de dispersie, pentru poziţionarea înregistrării pe altă adresă (de asemenea, măreşte probabilitatea coliziunilor care, la rândul lor, necesită alt mod de tratament al coliziunilor);
- utilizarea blocurilor – adresa generată de funcţia de dispersie aparţine unui bloc unde înregistrările pot fi plasate în formă serială.

Consultarea înregistrărilor (după cheia primară) este mai eficientă în acest tip de organizare, dar trebuie să se ţină cont de procedura aleasă pentru tratarea coliziunilor.

Dezavantajele acestui tip de organizare sunt: imposibilitatea de consultare după mai multe chei (trebuie să fie utilizate împreună alte mecanisme) şi necesitatea de rearanjare (reorganizare), când fişierul creşte prea mult (multe includeri). În acest caz, trebuie aleasă o altă funcţie de dispersie şi toate înregistrările repoziţionate de funcţia nouă.

Pentru a diminua probabilitatea de apariţie a coliziunilor, se poate aloca un spaţiu mai mare pentru fişier (un spaţiu suplimentar de 20%-30%). O bună funcţie de dispersie, de asemenea, poate diminua coliziunile (dar nu le poate evita). Acest tip de organizare este optimal pentru aplicaţiile care au nevoie de consultări aleatorii rapide (ale unor înregistrări specifice) şi când fişierul nu creşte mult.

Dacă fişierul creşte mult, pot fi utilizate alte tehnici. Tehnicile cunoscute ca *dispersia dinamică* şi *dispersia extensibilă* utilizează blocuri pentru alocarea înregistrărilor. Dacă blocul a crescut prea mult (până aproape de limita sa), el trebuie să fie divizat şi înregistrările lui redistribuite în blocurile nou-produse.

Cu dispersia dinamică, realocarea înregistrărilor în blocurile noi este dictată de o funcţie nouă. Acum, pentru găsirea unei înregistrări, este necesară examinarea istoriei realocării (retrecând toate funcţiile deja utilizate).

În cazul dispersiei extensibile, realocarea este determinată de creşterea cheii. Într-adevăr, cheia produsă de funcţia de dispersie este de lungime fixă, dar codificată de o secvenţă de biţi de lungime suficient de mare. Numărul de biţi ce este analizat pentru identificarea blocului înregistrării se modifică (se majorează când blocurile sunt divizate şi se diminuează în cazul fuzionării blocurilor).

Aceste două tehnici implică reorganizarea unei părţi a fişierului, de aceea nu influenţează mult funcţionarea generală a sistemului.

Dat fiind faptul că înregistrările nu sunt ordonate, ci distribuite în formă aleatorie în fişier, structura fişierului cu dispersie nu funcţionează bine pentru consultări secvenţiale (parcurgerea secvenţială a înregistrărilor în ordinea cheii primare).

Dacă există o aşa necesitate, poate fi utilizată tehnica *dispersia indexată*. Aceasta este o combinare între organizările de tip indexat şi dispersat. În dispersia indexată, valoarea produsă de funcţia de dispersie nu este adresa directă a înregistrării de date, ci o adresă în fişierul index (cu perechea care reprezintă cheia şi adresa înregistrării).

Fişierul index funcţionează ca cel mai intern nivel al organizărilor de tip indexat, unde cheile apar ordonate. Consultarea secvenţială poate fi făcută asupra acestui fişier index, în timp ce, pentru cele de consultare aleatorie apare numai o accesare în plus. O altă alternativă poate fi înlănţuirea înregistrărilor cu utilizarea pointerilor.

Astfel, pentru fişierele cu dispersie există două moduri de acces:

- secvenţial, care reprezintă o lectură secvenţială a datelor, conform ordinii dictate de cheie.
- aleatoriu, care localizează o înregistrare după adresa calculată cu ajutorul unei funcţii de dispersie, pornind de la valoarea cheii de acces.

Adresarea directă

Adresa directă, în forma sa mai simplă, este o adresă hardware, fie relativă, fie absolută, care permite sistemului să acceseze înregistrarea corespunzătoare. Adresa hardware a unei înregistrări este formată din numărul cilindrului, numărul pistei şi din poziţia înregistrării pe pistă.

O abordare mai simplă a acestui tip de organizare constă în tratarea valorii cheii ca un număr tradus de sistem într-o adresă hardware relativă.

De exemplu, fie un fişier cu organizare directă păstrează înregistrări cu valorile cheii primare între 0 şi 9999. Dacă valoarea cheii este tratată ca număr relativ al înregistrării, atunci trebuie rezervat un spaţiu pentru 1000 înregistrări, care corespunde unei valori posibile a cheii între 0 şi 9999 inclusiv.

Adresarea directă semnifică faptul că sistemului îi este furnizată adresa fizică a înregistrării căutate, pentru a-i permite recuperarea directă a înregistrării. Adresa furnizată poate fi atât fizică a hardware-ului (numărul cilindrului, numărul pistei, numărul înregistrării), cât şi un număr de adresă relativă (poziţia într-un fişier). A doua opţiune este mai utilizată, şi din faptul că adresa relativă poate fi automat tradusă într-o adresă fizică.

Tratarea unei valori a cheii ca număr de adresă relativă are un dezavantaj. De exemplu, dacă un fişier cu organizare directă are înregistrări cu valorile cheii între 0 şi 9999, trebuie rezervate 10000 înregistrări, fiecare corespunzând unei valori posibile a cheii. Această rezervă trebuie făcută, chiar dacă în fişier există numai o înregistrare (cheia 9999, de exemplu).

Spaţiul neutilizat, în această organizare, poate fi foarte mare. Pentru aceasta administratorul bazei de date trebuie să analizeze necesitatea reală de adrese directe în ce priveşte spaţiul real ocupat de fişier şi spaţiul irosit. Dacă spaţiul liber, procentual, nu este mare, poate fi compensat de avantajele utilizării adresării directe.

Când numărul de chei posibile este, relativ, aproape de numărul real de chei, această formă de adresare este foarte eficientă. Dar, aproape întotdeauna, numărul real de chei este mult mai mic decât numărul de spaţii de înregistrări care trebuie păstrate. Când aceasta se întâmplă, pentru accesarea unui fişier cu dispersie, trebuie să fie utilizate consultarea dicţionarului sau adresarea indirectă. Pentru ca sistemul să poată converti

această valoare într-o adresă hardware, trebuie cunoscute următoarele mărimi: valoarea cheii, numărul de cilindri pe dispozitivul extern de stocare, numărul de înregistrări pe pistă, numărul de piste pe cilindru, adresa hardware absolută a fişierului.

Pentru a găsi adresa relativă şi absolută a unei înregistrări sunt trei etape trecute de sistem:

- valoarea cheii se împarte la numărul de înregistrări pe pistă, unde câtul este numărul pistei relative a înregistrării, iar restul este poziţia relativă a înregistrării pe această pistă;
- numărul pistei relative a înregistrării este împărţit la numărul de piste pe cilindru şi se obţine numărul relativ al cilindrului şi pista relativă a cilindrului (restul);
- adresa hardware absolută a înregistrării se găseşte, dacă se adaugă adresa hardware relativă a înregistrării la adresa hardware absolută a fişierului.

Adresarea indirectă

Adresarea indirectă presupune că se aplică o funcţie de transformare a valorii cheii într-o adresă directă, care poate fi utilizată de sistemul de gestiune pentru localizarea înregistrării dorite. Aceasta, spre deosebire de adresarea directă, unde nu se face nici o transformare a valorii cheii înainte de utilizarea ei de sistem.

În fişierele cu dispersie, cu adresare indirectă, înregistrările se grupează în clase. Apartenenţa unei înregistrări la una din clase este determinată, în funcţie de valoarea pe care o are cheia înregistrării.

Spaţiul necesar pentru stocarea claselor de înregistrări este divizat în blocuri de stocare. Un bloc de stocare poate conţine una sau mai multe înregistrări logice, capacitatea lor fiind determinată de proiectant. Funcţia de dispersie calculează, de fapt, adresa primului bloc al clasei, unde este stocată înregistrarea. În implementarea fişierelor cu dispersie se construieşte o listă numită *directoriu* cu pointeri la blocul de începere corespunzător fiecărei clase, figura 28.12.

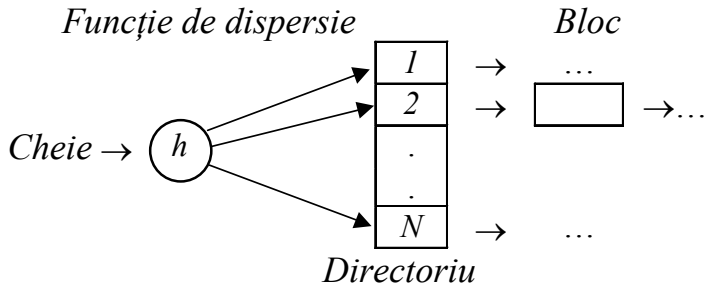


Figura 28.12 *Un fişier cu dispersie*

Funcţia de dispersie, de obicei, poate mapa diferite valori ale cheii pe aceeaşi adresă a spaţiului de stocare, provocând fenomenul numit *coliziune*. Evident că înregistrările care au aceeaşi adresă produsă de funcţia de dispersie aparţin aceleiaşi clase. Stocarea înregistrărilor ce aparţin unei clase în blocuri depinde de metodele de soluţionare a coliziunilor.

Principala problemă ce ţine de adresarea indirectă este alegerea funcţiei *h*, care transformă valoarea *C* a cheii unei înregistrări în adresa *A,* care îi corespunde în fişier. O altă problemă este găsirea unei metode adecvate de soluţionare a coliziunilor şi de plasare a înregistrărilor excedentare în blocuri. Astfel, algoritmul de lucru cu fişierele cu dispersie presupune, printre altele, două activităţi: maparea (utilizând funcţia de dispersie) valorii cheii într-o adresă absolută sau relativă şi citirea înregistrărilor excedentare.

Dacă proiectantul alege un spaţiu mic de stocare a primului bloc din fiecare clasă de înregistrări, apare un număr relativ înalt de coliziuni, fapt ce produce lecturi suplimentare pentru localizarea înregistrării dezirabile în spaţiul de înregistrări excedentare. O capacitate de stocare mai mare provoacă mai puţine coliziuni, dar mult spaţiu rămâne nevalorificat. În practică, capacitatea spaţiului de stocare depinde de caracteristicile dispozitivului secundar de stocare a fişierului cu date. Raportul dintre numărul de înregistrări existente în fişier şi numărul total de locuri disponibile pentru înregistrări în fişier se numeşte *grad de încărcare* a fişierului. Pe măsură ce gradul de încărcare creşte, probabilitatea unei înregistrări, ce trebuie adăugată, de a deveni una excedentară (care provoacă o coliziune), de asemenea, creşte.

Eficienţa unui fişier cu dispersie, cu utilizarea adresării indirecte, este influenţată de mai mulţi factori:

- dimensiunea spaţiului de stocare;
- gradul de încărcare;
- valorile cheilor;
- funcţia de dispersie;
- metoda de soluţionare a coliziunilor;
- metoda de citire a înregistrărilor excedentare.

Calcularea adresei

Există diverse tehnici pentru transformarea cheii într-o adresă.

Pot fi considerate două tipuri de funcţii de dispersie, primul fiind constituit din *funcţii deterministe* care pentru două valori diferite ale cheii produc adrese diferite. Acest tip de funcţii prezintă avantaje evidente. Dar, în practică, pentru un număr mare de înregistrări, este imposibil de găsit o funcţie deterministă simplă. Acele funcţii care pot fi folosite sunt aşa de complexe, că elimină toate avantajele accesului direct. În afară de necesitatea de adaptare la fiecare inserare suferită de fişier, de obicei, produc adrese necontinue, ceea ce duce la utilizarea unui spaţiu larg, similar adresării directe. Din acest motiv, funcţiile deterministe nu prezintă un mare interes practic.

Al doilea tip formează funcţiile nedeterministe, care produc pentru fiecare valoare a cheii o singură valoare a adresei, iar, uneori, pentru valori distincte ale cheii, aceeaşi adresă, ceea ce constituie o coliziune. Funcţiile nedeterministe pot fi proiectate şi cu scopul de a atinge unele obiective secundare, obiective care nu sunt incompatibile:

- păstrarea ordinii înregistrărilor după valoarea cheii de acces;
- mărirea gradului de unicitate a adreselor produse, adică, distribuirea înregistrărilor în fişier, cu o mare uniformitate posibilă.

O funcţie de dispersie nedeterministă care nu ţine cont de ordinea cheii se mai numeşete *funcţie de randomizare* şi are ca obiectiv distribuirea uniformă a înregistrărilor în fişier.

Funcţia de dispersie "restul de la împărţire". Această funcţie de dispersie este utilizată frecvent. În calitate de valoare a funcţiei de dispersie se foloseşte restul de la împărţirea unei chei la un număr întreg *N*: $h(C) = C \bmod N + 1$. Aici la rezultatul împărţirii se adaugă *1* pentru a obţine o valoare a funcţiei între *1* şi *N*.

Evident că eficienţa distribuirii cheilor depinde mult de valoarea lui *N*. Astfel, contraindicat ca *N* să fie puterea bazei unui sistem de numeraţie, deoarece, în acest caz, valorile funcţiei de dispersie vor fi reprezentate de ultima cifră a cheii. Cele spuse sunt adevărate şi pentru cheile alfanumerice. Dacă $N=2^8$, de exemplu, valoarea funcţiei de dispersie va fi ultimul caracter al cheii. Pentru a depăşi concentrarea cheilor în jurul unor adrese, se recomandă ca *N* să fie un număr prim.

Deseori, mulţimea de chei reprezintă nişte secvenţe de progresii de tipul XXA,XXB,… sau FT01, FT02,… Funcţia de dispersie *restul de la împărţire* transformă astfel de chei în adrese succesive. O asemenea situaţie poate fi considerată destul de satisfăcătoare.

Metoda înmulţirii. Fie cheia *C* este reprezentată în forma unui număr binar şi fie $N=2^n$. Se înmulţeşte o fracţie α cu *C* şi se ia partea fracţionară care este notată prin $\{C\alpha\}$, iar în calitate de valoare a funcţiei de dispersie se folosesc numai primii *n* octeţi. Cu alte cuvinte, $h(C)=\lfloor N\times\{C\alpha\}\rfloor$, unde $\lfloor x\rfloor$ este cel mai mare număr întreg mai mic decât *x*.

Se recomandă a se folosi în calitate de valoare α un număr iraţional, aproape de lungimea unui cuvânt. De exemplu, dacă α este reprezentată de $\alpha = (\sqrt{5}-1)/2$, atunci segmentul [0,1] se împarte foarte bine la $\{\alpha\},\{2\alpha\},\ldots$. Cu alte cuvinte, oricât de mărunt s-ar diviza segmentul, lungimea segmentelor obţinute după divizare nu se vor exprima prin mai mult de trei valori α^C, α^{C+1}, α^{C+2}. În cazul divizării următoare segmentul de lungime α^C se împarte în segmente cu lungimile α^{C+1}, α^{C+2}. Această proprietate, în cazul aplicării metodei înmulţirii, permite a obţine rezultate bune.

Cerinţa $N=2^n$ pentru calcularea funcţiei de dispersie *h(C)* nu este obligatorie. Trebuie menţionat că, dacă $\alpha=1/N$, atunci această metodă este echivalentă cu metoda *restul de la împărţire*.

Foarte aproape de această metodă de dispersie este metoda *"mijlocul" pătratului*, în care, în calitate de valoare a funcţiei de dispersie *h(C)*, se folosesc *n* octeţi din mijlocul numărului C^2. Şi aceasta este o metodă de dispersare, dar, după mulţi parametri, cedează metodei înmulţirii.

Metoda transformarea sistemelor de numeraţie. Metoda de calculare a valorii funcţiei de dispersie presupune transformarea valorii cheii *C*, reprezentată în sistemul de numeraţie în baza *p*, $C = a_0 + a_1 p + a_2 p^2 + \ldots$, într-o valoare reprezentată în sistemul de numeraţie în baza *q* cu constrângerea *s* asupra ordinii rezultatului: $h(C) = a_0 + a_1 q + \ldots + a_{s-1} q^{s-1}$. Aici *p* şi *q* sunt numere pozitive, astfel, că $p<q$.

Pentru calcularea valorii funcţiei de dispersie *h(C)*, sunt necesare *s* operaţii de înmulţire sau împărţire. Prin urmare, complexitatea (numărul de operaţii) acestei metode este mai înaltă decât a metodei *înmulţirea*.

Metoda împărţirea polinoamelor. Fie cheia *C*, exprimată în sistemul binar de numeraţie, se scrie $C = 2^{m-1} b_{m-1} + ... + 2b_1 + b_0$ şi fie $N=2^n$. Cheia binară *C* se reprezintă în forma unui polinom $C(t) = b_m t^m + ... + b_1 t + b_0$, păstrând aceiaşi coeficienţi. Acum se determină restul de la împărţirea acestui polinom la polinomul constant de forma $P(t) = t^n + p_{n-1} t^{n-1} + ... + p_1 t + p_0$. Restul obţinut se consideră din nou în sistemul binar de numeraţie şi se foloseşte în calitate de valoare a funcţiei de dispersie *h(C)*. Pentru calcularea restului de la împărţirea polinoamelor, se foloseşte aritmetica polinomială după modulul 2. Dacă în calitate de *P(t)* este ales un polinom ireductibil simplu, atunci, pentru două chei foarte aproape, dar nu egale, $C_1 \neq C_2$, întotdeauna se va realiza condiţia $h(C_1) \neq h(C_2)$. Această funcţie de dispersie posedă proprietăţi mai puternice de dispersare a mulţimii de chei decât precedentele.

Trebuie menţionat că, în fişierele cu dispersie, transformările cheii în adresă sunt similare funcţiilor de dispersie utilizate pentru căutarea în tabelele stocate în memoria principală. Deosebirea constă în caracteristicile accesului la memoria secundară, care diferă de cele ale accesului în memoria internă accesibilă direct. Pentru funcţiile de dispersie în memoria principală timpul de calculare a adresei poate fi critic, însă, pentru memoria secundară, acest timp poate fi neglijat.

Tratarea coliziunilor

Tratarea coliziunilor este unul din aspectele mai importante în adresarea indirectă şi este consecinţa utilizării funcţiilor nedeterministe pentru transformarea valorilor cheii în adrese ale fişierului. Se spune că există o coliziune când două valori diferite ale cheii de acces sunt transformate în aceeaşi adresă.

Deoarece pe o adresă se poate stoca o singură înregistrare, este necesară stabilirea unei proceduri pentru a alege o altă adresă, unde va fi plasată înregistrarea care a provocat coliziunea. Această procedură se aplică şi în operaţia de acces, dacă, pe adresa produsă în baza argumentului de căutare, se găseşte o înregistrare cu o cheie diferită de cea a argumentului.

Soluţiile mai frecvent utilizate sunt *adresarea deschisă* şi *înlănţuirea*.

Tratarea cu adresare deschisă. Această metodă de soluţionare a coliziunilor presupune că se foloseşte o anumită regulă de parcurgere a înregistrărilor. O dată cu apariţia unei coliziuni la operaţia de inserare, este făcută o căutare în fişier pentru localizarea unei adrese libere, unde va fi stocată înregistrarea. Procesul de inserare se supune unei secvenţe predefinite, care este urmată şi în timpul accesului la înregistrare. Regula de căutare trebuie să fie astfel, încât, chiar în lipsa pointerilor, aceeaşi secvenţă să poată fi parcursă. Procedura poate arăta astfel:

1. *i=0*
2. *A=h_i(C):*
   - *3.1. Dacă adresa A este liberă, stop (dacă aceasta e procedura de inserare, algoritmul se termină cu succes, însă, dacă aceasta e procedura de căutare, înseamnă că aşa cheie nu există);*
   - *3.2. Dacă cheia de pe adresa A este egală cu C, stop (dacă este procedura de inserare, inserarea a doua nu se mai face, iar dacă e procedura de căutare, algoritmul se termină cu succes);*

4. În caz contrar – coliziune; i:=i+1 şi se trece la pasul 2.

În sensul larg, $\{h_i\}_{i=0}^{\infty}$ reprezintă o secvenţă de funcţii de dispersie. Alegerea lor este o problemă destul de complicată. Considerând că h_0 este una din funcţiile examinate anterior, mai departe se examinează celelalte funcţii din secvenţă.

Căutarea poziţiei libere poate fi realizată cu una din următoarele metode de adresare deschisă:

- cercetarea liniară;
- cercetarea pătratică;
- rerandomizarea.

Cercetarea liniară. Aceasta este cea mai simplă schemă de adresare deschisă, în care $h_i(C) = h_0(C) + ki$, unde *k* este o constantă. Regula este simplă, însă există pericolul unei concentrări a înregistrărilor la începutul procesului, unde foarte mult sunt mixate adresele. Pentru a evita aceasta, *k* şi *N* trebuie să fie reciproc numere prime, iar *k* un număr nu prea mic. Dar în cazul când numărul cheilor este mic în raport cu numărul *N*, pericolul creşterii căii de căutare nu este mare, iar faptul că sinonimele se vor stoca pe adrese adiacente este un avantaj. În figura 28.13 numărul de adrese *N=10,* funcţia de dispersie iniţială este $h_0(C)=Cmod10+1$, iar $h_i(C)=h_0(C)+i$.

C	$h_0(C)$	$h_i(C)$
25	6	
43	4	
56	7	
35	6	8
54	5	
13	4	9
80	1	
104	5	10

Adresă	Înregistr.
1	80
2	
3	
4	43
5	54
6	25
7	56
8	35
9	13
10	104

Figura 28.13 *Cercetarea liniară*

Cercetarea pătratică. Aceasta este schema de adresare deschisă, în care $h_i(C) = h_0(C) + ki + di^2$, unde *k* şi *d* sunt constante. Datorită neliniarităţii acestei adresări se poate evita creşterea numărului de probe pentru mai mult de două sinonime. Însă, în cazul unui fişier aproape plin, nu se poate evita a doua concentrare de chei. Aceasta e condiţionată de faptul că, în cazul unui număr mare de chei, sinonimele unui grup intră în coliziune cu alte chei.

De acest dezavantaj practic este absolvită metoda următoare, în care secvenţele de probe pentru astfel de chei se dovedesc a fi diferite.

Rerandomizarea. Rerandomizarea este schema de adresare deschisă în care, $h_i(C) = g(h_{i-1}(C))$, unde *g(C)* este o funcţie asemănătoare cu h_0, dar nu echivalentă ei. Procesul este repetat până se găseşte o adresă liberă. Avantajul acestei metode, în comparaţie cu cele anterioare, este de a

evita concentrarea excesivă a înregistrărilor în unele regiuni ale fişierului, mărind, deci, eficienţa operaţiilor şi accesului. În figura 28.14, este reprezentat rezultatul aplicării funcţiilor de dispersie $h_0(C)=Cmod10+1$ şi $h_i(C)=(h_{i-1}(C)+1)mod10+1$.

C	$h_0(C)$	$h_1(C)$	$h_2(C)$	$h_3(C)$
25	6			
43	4			
56	7			
35	6	8		
54	5			
13	4	6	8	10
80	1			
104	5	7	9	

Adresă	Înregistr.
1	80
2	
3	
4	43
5	54
6	25
7	56
8	35
9	104
10	13

Figura 28.14 *Rerandomizarea*

Tratarea prin înlănţuire. Tratarea prin înlănţuire a coliziunilor presupune că toate sau o parte de chei sinonime sunt adunate şi înlănţuite cu ajutorul pointerilor, formând o listă. Fiecare listă se accesează prin adresa produsă de funcţia de dispersie, figura 28.15. Listele pot fi păstreze atât împreună cu directoriul, cât şi într-un spaţiu de memorie separat.

C	h(C)
25	6
43	4
56	7
35	6
54	5
13	4
80	1
104	5

Adresă	Înregistr.	
1	80	
2		
3		
4	43	•
5	54	•
6	25	•
7	56	
8		
9		
10		

13	
104	
35	

Figura 28.15 *Tratarea prin înlănţuire*

În cazul când listele de adrese sunt stocate aparte, consultarea lor este simplă. După calcularea valorii funcţiei de dispersie $A=h(C)$, problema se reduce la căutarea secvenţială a listei cu adresa A (eventual stocată în memoria secundară). Memoria secundară, în acest caz, poate fi divizată în blocuri, astfel încât cheile sinonime să se păstreze într-un bloc. Deoarece directoriul va conţine numai pointeri, numărul lor poate fi destul de mare pentru a micşora probabilitatea apariţiei coliziunilor. Această metodă este adecvată unui număr mare de înregistrări de lungime variabilă.

Nu apar probleme la includerea unor chei noi. Dacă, însă, numărul de sinonime devine prea mare, în loc de liste liniare, pot fi utilizate structuri arborescente.