UNIVERSITATEA "POLITEHNICA" TIMIŞOARA
FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE

Şl.drd.ing ADI-CRISTINA MITEA

TEZĂ DE DOCTORAT

CONTRIBUŢII PRIVIND PROIECTAREA BAZELOR DE DATE RELAŢIONALE ŞI RELAŢIONAL-OBIECTUALE

Conducător ştiinţific:

Prof.dr.ing. IONEL JIAN

Contribuții privind proiectarea bazelor de date relaționale şi relațional-obiectuale

J. Piaget

"...invention...is nothing other than spontaneous reorganisation of earlier schemata which are accommodated by themselves to the new situations, through reciprocal assimilation…"

Prefață

Teza îşi propune aducerea de contribuții în domeniul proiectării bazelor de date ce folosesc modelul relațional sau relațional-obiectual al datelor. Deoarece baza de date este fundația pe care sunt construite multe dintre sistemele informatice, este esențial ca aceasta să fie robustă şi consistentă pentru a putea asigura o performanță corespunzătoare pentru sistemul din care face parte. Prin urmare, în cadrul tezei, procesului de proiectare este abordat fiind urmărită creşterea performanțelor bazelor de date ce folosesc aceste modele de date. În vederea îmbunătățirii calității şi performanței unei baze de date aflată în etapa operațională a ciclului său de viață, se propune o metodă de proiectare a reprezentării fizice a bazei de date. Metoda este înglobată într-o metodologie de proiectare fizică a bazelor de date, metodologie care foloseşte un formalism matematic ce permite luarea deciziilor de proiectare pe baza unor validări matematice care conduc la eliminarea erorilor umane de proiectare. Structura fizică a bazei de date este proiectată urmărind maximizarea performanțelor bazei de date, luând în calcul operațiile care se execută asupra bazei de date la nivelul tranzacțiilor care operează asupra ei.

Metodologia propune un cadru de proiectare care este suficient de general pentru a putea fi utilizat pentru o întreagă clasă de sisteme, dar şi suficient de specific pentru a putea rezolva aspectele caracteristice ale acestor sisteme.

Procesul de proiectare fizică a bazei de date poate fi automatizat folosind metodologia de proiectare propusă. Aceasta permite dezvoltarea unui utilitar de asistare a procesului de proiectare fizică a bazei de date. Un astfel de utilitar a fost dezvoltat şi în cadrul tezei, pentru a ilustra beneficiile aduse de metoda de proiectare fizică propusă.

Mulțumiri

Elaborarea tezei de doctorat a reprezentat o etapă de lucru intensă şi îndelungată la finele căreia aş fi reuşit să ajung cu siguranță mult mai greu fără sprijinul unor persoane cărora aş vrea să le mulțumesc şi pe această cale.

În primul rând aş dori să-i mulțumesc domnului profesor dr.ing. Ionel Jian, de a cărui îndrumare atentă şi competentă am beneficiat în toți aceşti ani. Vă mulțumesc pentru sprijinul acordat şi pentru tactul cu care mi-ați dirijat paşii pe drumul ales.

Doresc să mulțumesc colegilor mei din cadrul Catedrei de Calculatoare şi Automatizări a Facultății de Inginerie "Hermann Oberth" din Sibiu, pentru suportul tehnic asigurat de-a lungul realizării tezei, pentru sfaturile prețioase pe care le-am primit de la dânşii şi pentru îngăduința acordată în atribuirea sarcinilor de servici.

Mulțumirile mele persoanelor cu rol de decizie din cadul firmei S.C. "Boema Prest" S.R.L. din Sibiu, care m-au ajutat să înțeleg în urma discuțiilor purtate complexitatea problemelor cu care se confruntă în administrarea şi analizarea vânzărilor efectuate de firma pe care o conduc, şi care mi-au pus la dispoziție datele necesare.

Doresc, totodată, să mulțumesc firmei S.C."Industrial Software" S.R.L. din Sibiu, pentru sprijinul acordat în testarea metodologiei de proiectare propusă în cadrul tezei şi pentru suportul acordat la realizarea aplicației de asistare a procesului de proiectare fizică a bazelor de date.

La final, dar nu în ultimul rând, doresc să mulțumesc familiei mele, care m-a susținut moral în tot acest timp. Vă mulțumesc tuturor pentru înțelegerea şi sprijinul acordat în această lungă perioadă de pregătire şi realizare a tezei, în care v-am privat de prezența şi implicarea mea totală în viața de familie.

Adi-Cristina Mitea

Cuprins

I *Introducere*

1.1. Introducere

Producerea unor sisteme care satisfac cerinţele de performanţă şi de calitate ale utilizatorilor este un deziderat atât economic cât şi o provocare pentru ştiinţă şi tehnologie. Şi pentru că baza de date este o componentă importantă a multora dintre aceste sisteme, aceste cerinţe se răsfrâng şi asupra ei.

Performanţa în exploatare a unei baze de date este esenţială pentru sistemul informaţional care o utilizează. Gradul de satisfacţie inoculat utilizatorului sistemului este decisiv în acceptarea sau respingerea acestuia. Dacă prelucrarea datelor este greoaie şi dacă timpii de execuţie sunt prea mari, succesul general al aplicaţiei poate fi puternic periclitat. Pentru a avea o bază de date corect şi optim structurată este necesară parcurgerea unui proces de proiectare a acesteia care să elimine toate problemele.

Aplicarea metodelor formale de-a lungul procesului de dezvoltare a sistemelor software este considerată baza pentru dezvoltarea într-un mod riguros a acestora. În consecinţă, folosirea metodelor formale de-a lungul procesului de proiectare a bazei de date poate fi soluţia pentru un produs final superior, de o performanţă şi o calitate sporită.

Deoarece performanţa în exploatare a unei baze de date este puternic dependentă de structura fizică a acesteia, am ales tocmai acest domeniu ca domeniu de studiu în cadrul tezei. Prin urmare, mi-am concentrat atenţia asupra etapei de proiectare fizică a unei baze de date, dezvoltând o metodologie de proiectare fizică a unei baze de date bazată pe modelul relaţional

al datelor. Am ales acest model al datelor, pentru că el este cel mai utilizat model la ora actuală.

Metodologia propusă foloseşte o metodă formală de specificare. Modelul de dezvoltare a procesului de proiectare este unul bazat pe specificare, fiind inspirat de filosofia modelului „Cleanroom". Sunt definite verificări ale procesului de proiectare fizică a unei baze de date pentru a se asigura obținerea unei scheme fizice a bazei de date lipsită de erori.

Metodologie propusă poate fi aplicată la proiectarea oricărei baze de date ce foloseşte modelul relațional al datelor şi are ca punct de plecare schema logică a bazei de date. In final se va obține o schemă fizică a bazei de date, calitativ superioară, care va putea asigura o performanță sporită la nivelul bazei de date. Metoda de proiectare a reprezentării fizice a bazei de date a fost dezvoltată pornind de la rezultatele unor studii efectuate asupra comportamentului unor baze de date structurate fizic diferit şi implementate pe unele dintre cele mai performante sisteme de gestiune a bazelor de date de la ora actuală.

Prin urmare, obiectivul principal al prezentei teze este acela de a aduce contribuții în domeniul proiectării fizice a bazelor de date, urmărind creşterea performanței bazei de date aflată în etapa operațională a ciclului său de viață.

1.2. Structura tezei

Structura tezei este următoarea:

- *Capitolul I*

Este capitolul introductiv al tezei în care este prezentată o motivație a alegerii acestei teme şi a obiectivelor principale care au fost urmărite în cadrul ei. Modul în care au fost abordate şi realizate problemele propuse la nivelul fiecărui capitol al tezei este şi el, de asemenea, prezentat .

- *Capitolul II*

Descrie procesul de dezvoltare a unei baze de date şi evidențiază importanța utilizării unor modele formale de specificare în fazele de început ale dezvoltării, pentru a se asigura o calitate sporită a bazei de date şi o performanță superioară a acesteia în etapa ei operațională. Se realizează o prezentare a modelelor de dezvoltare software şi se arată că

prin folosirea specificațiilor formale se poate creşte calitatea produsului final al procesului de proiectare. La nivelul bazei de date sunt aplicabile cu succes modelele de dezvoltate bazate pe specificare, pentru că se porneşte de la cerințe cunoscute într-o măsură relativ mare, care pot fi prin urmare specificate. Folosirea metodelor de specificare formală asigură o calitate superioară pentru produsul final, care în cazul de față înseamnă o bază de date calitativ superioară. Creşterea performanței bazei de date se poate realiza, prin urmare, prin folosirea în cadrul procesului de proiectare a bazei de date a unei metode formale de specificare.

Un alt factor, care poate influența performanța bazei de date, este legat tot de calitatea procesului de proiectare a acesteia. Prin folosirea unei metodologii la nivelul procesului de proiectare acesta poate fi uşurat şi eficientizat. Metodologia de proiectare ajută la realizarea unei proiectări riguroase a bazei de date, care va asigura obținerea unei structuri optime pentru baza de date, astfel încât performanțele obținute în exploatarea acesteia să fie maxime.

Dintre etapele de proiectare a bazelor de date, cea de proiectare fizică este cea mai apropiată de structura fizică a viitoarei baze de date şi, ca atare, ea este cea care influențează cel mai direct performanța viitoarei baze de date. Teza propune utilizarea unei metodologii de proiectare şi a specificatiilor formale la nivelul etapei de proiectare fizică a bazei de date în scopul creşterii calității procesului de proiectare, rezultând astfel o bază de date cu o structură optimă care poate asigura creşterea performanțelor.

- *Capitolul III*

Prezintă rezultatele şi concluziile unor studii efectuate asupra performanțelor unor baze de date implementate folosind produsele Oracle 9*i*, SQL Server 2000 şi DB2, care folosesc diverse structuri de organizare a fişierelor de date şi diferite metode de acces la aceste structuri. Este analizată comportarea şi performanțele înregistrate de aceste baze de date în diverse situații operaționale.

Se arată că baza de date este fundamentul pe care se clădesc majoritatea sistemelor informatice. Pentru a avea un sistem informatic solid, care să răspundă cu succes tuturor cerințelor utilizatorilor săi, este esențial ca şi baza de date să fie una solidă şi robustă care să asigure performanța necesară. Alegerea unui sistem de gestiune a bazei de date performant nu este neapărat garantul unor performanțe deosebite ale bazei de date în exploatare. Un rol deosebit de important, vis-à-vis de performanțele în exploatare ale unei

baze de date, îl joacă structurile de date folosite pentru implementarea bazei de date şi metodele de accesare a acestor structuri.

Performanța unei baze de date este legată de timpii de răspuns pe care aceasta îi poate asigura pentru operațiile care se execută asupra ei. Creşterea performanțelor implică asigurarea unor timpi de răspuns cât mai mici cu putință. O modalitate de a realiza acest deziderat este printr-o corectă structurare a bazei de date. Tipurile de structuri de date alese pentru a implementa baza de date, precum şi metodele alese pentru a accesa aceste structuri sunt esențiale pentru realizarea obiectivului de performanță propus.

În cadrul capitolului se face într-o primă etapă o prezentare şi o analiză a structurilor de date şi a metodelor de acces folosite la nivelul a trei sisteme de gestiune a bazelor de date existente astăzi pe piață. Sistemele alese spre analiză sunt cele mai performante şi mai larg utilizate pe scară mondială pentru realizarea bazelor de date folosite în cadrul sistemelor informatice mari care prelucrează în mod complex volume mari de date.

Este realizată o analiză comparativă a celor trei sisteme pe segmentul de analiză propus.

În acelaşi capitol este realizat un studiu al influenței structurii fizice asupra performanței bazei de date. Este studiat comportamentul unor baze de date, implementate folosind cele trei sisteme, în diferite situații de organizare a structurii fizice a bazei de date şi a folosirii unor diferite metode de acces. Au fost identificate pattern-uri de organizare fizică a bazei de date care corespund anumitor situații operaționale care pot să apară la nivelul bazei de date.

A fost studiată influența numărului de structuri auxiliare de acces asupra performanței bazei de date, atât pentru fişiere mici cât şi pentru fişiere mari de date, atunci când numărul înregistrărilor prelucrate a variat între 0-25% din mărimea totală a fişierului de date. Au fost formulate concluzii.

A fost studiată comparativ structura de index de tip arbore B şi cea de index de tip bitmap. Au fost identificate situațiile în care folosirea fiecăreia dintre cele două structuri este mai avantajoasă. Au fost formulate concluzii.

Au fost studiate comparativ tabelele având o organizare de tip heap şi cele organizate indexat. Au fost identificate situațiile în care folosirea acestor structuri poate conduce la creşteri ale performanței. Au fost formulate concluzii.

Au fost studiate comparativ tabelele având o organizare de tip heap şi cele partiționate. Au fost identificate situațiile în care partiționarea este indicată şi au fost relevate

avantajele partiționării. Au fost analizate mai multe tipuri de partiționare stabilind situațiile în care folosirea lor prezintă un avantaj. Au fost formulate concluzii.

Toate aceste teste au avut ca scop identificarea criteriilor care stau la baza deciziilor de proiectare fizică a bazei de date şi ele au fost folosite la nivelul etapei de proiectare a reprezentării fizice a bazei de date din cadrul metodologiei de proiectare fizică a bazelor de date propusă în cadrul tezei.

- *Capitolul IV*

Prezintă metodologia de proiectare fizică a unei baze de date, bazată pe modelul relațional al datelor, propusă în cadrul tezei. Modelul de dezvoltare a procesului de proiectare a bazei de date este inspirat de modelul „Cleanroom" şi foloseşte o metodă formală de specificare. Paşii propuşi în cadrul metodologiei sunt validați pe baza unor argumente matematice, care permit astfel eliminarea erorilor din procesul de proiectare.

Aceste este capitolul principal al tezei. El propune o nouă variantă de metodologie de proiectare fizică a bazelor de date. Scopul introducerii acestei metodologii este acela de a îmbunătăți procesul de proiectare a bazei de date, astfel încât, produsul final obținut să fie calitativ superior şi să asigure o performanță sporită pentru baza de date atunci când ea se va afla în etapa operațională a ciclului său de viață.

Metodologia cuprinde mai mulți paşi care trebuie urmați de proiectantul bazei de date pentru a asigura obținerea pe baze riguroase şi fundamentate a unei baze de date optim structurată, consistentă şi coerentă.

Modelul de dezvoltare „Cleanroom" a fost ales pentru a sta la baza procesului de proiectare fizică a bazei de date. Erorile de proiectare sunt eliminate aproape complet din cadrul procesului de proiectare prin folosirea unei metode de specificare formală ce utilizează verificările şi validările statice.

Baza de date este proiectată într-un *„mediu curat"*, adică prin eliminarea într-o măsură cât mai mare a impurităților ce constau de fapt în erorile de proiectare. Eliminarea erorilor de proiectare este realizată prin folosirea unui formalism matematic în specificarea paşilor de proiectare şi prin utilizarea unor validări pe baze matematice a deciziilor de proiectare. În final, rezultă un proiect fizic al bazei de date liber de defecte. Etapa de testare a bazei de date poate fi aproape complet eliminată. Verificarea statică este eficientă în ceea ce priveşte costul, deoarece defectele pot fi detectate mult mai rapid în

cadrul procesului de dezvoltare a sistemului şi la un preţ de cost mai scăzut decât dacă s-ar utiliza tehnici de detecţie a erorilor.

Metoda de analiză a tranzacţiilor este originală. Ea se bazează pe calcularea unor punctaje ponderate pentru tranzacţiile care se execută asupra bazei de date, care ţin cont de frecvenţele de execuţie ale tranzacţiilor, de orele cu încărcare de vârf din sistem şi de restricţiile cu privire la timpii de răspuns impuşi tranzacţiilor. Aceste punctaje asociate tranzacţiilor permit identificarea tranzacţiilor critice din sistem, adică acele tranzacţii care aparţin categoriei celor 20% dintre tranzacţii care efectuează 80% dintre prelucrările asupra bazei de date. Este contorizat numărul operaţiilor de selecţie, inserare, modificare şi ştergere care au loc la nivelul fiecărei relaţii în parte, şi sunt calculate punctaje asociate relaţiilor care ajută la identificarea relaţiilor critice din sistem, adică a acelor relaţii asupra cărora trebuie să se realizeze o mai atentă proiectare în vederea obţinerii unei performanţe sporite la nivelul bazei de date. Au fost definiţi doi indici, *indicele de activitate* şi *indicele tipului de activitate*, care ajută la luarea deciziilor de proiectare pe baze matematice nu euristice. Au fost definite, de asemenea, două noi noţiuni care ajută la identificarea soluţiilor optime privind structurile auxiliare de acces. Acestea sunt *atribute verzi*, adică atributele unei relaţii care este indicat să facă parte din cheile de indexare ale acelei relaţii, şi *atribute roşii*, adică acele atribute ale unei relaţii care nu este indicat să aparţină cheilor de indexare ale relaţiei. Au fost definite, de asemenea, reguli pe baza cărora să fie făcută încadrarea atributelor relaţiilor în categoria atributelor verzi sau a atributelor roşii. Atributele verzi şi atributele roşii permit specificarea cheilor de indexare pentru fiecare relaţie a bazei de date.

Metodologia permite determinarea pe baze fundamentate matematic a relaţiilor ce trebuie partiţionate şi a cheilor optime de partiţionare a acestora. Pornind de la punctajele calculate pentru relaţiile din baza de date şi pentru tranzacţiile care operează asupra lor, precum şi de la celelalte decizii de proiectare adoptate, sunt alese tipurile optime ale structurilor de date atât pentru fişierele de date cât şi pentru structurile indexate.

Metodologia permite identificarea relaţiilor care pot avea probleme în satisfacerea cerinţelor de performanţă şi, ca atare, pot să necesite introducerea unei redundanţe controlate la nivelul bazei de date.

Este efectuată o estimare a mărimii totale a bazei de date rezultate. Dacă este necesar, sunt realizate ajustările necesare asupra bazei de date. La final, este obţinută structura de memorare a bazei de date şi, de asemenea, structurile adiţionale de acces care vor îmbunătăţi performanţa bazei de date.

O ultimă etapă a metodologiei de proiectare fizică a bazei de date permite proiectarea mecanismelor de securitate care țin de structura fizică a bazei de date. Sunt definite vederile utilizatorilor asupra bazei de date şi sunt proiectate regulile de acces ale utilizatorilor atât la nivelul tabelelor cât şi a vizualizărilor din baza de date.

La final, există toate datele necesare pentru trecerea la următoarea etapă a ciclului de viață a bazei de date, şi anume aceea de implementare.

Dezvoltarea etapelor şi a paşilor metodologiei propuse s-a realizat şi pe baza rezultatelor şi a concluziilor trase la nivelul studiilor efectuate în cadrul capitolului 3. Validările statice folosite în cadrul paşilor de proiectare fizică a bazei de date se bazează pe rezultatele obținute în cadrul testelor efectuate.

Este important pentru succesul viitoarelor aplicații, care vor rula pe baza de date, ca etapa de proiectare fizică a bazei de date să fie foarte bine realizată. O structură de memorare rău aleasă pentru fişierele de date şi pentru structurile auxiliare de acces poate deteriora foarte tare performanța bazei de date.

Această metodologie ajută proiectanții bazei de date să ia deciziile de proiectare a bazei de date pe baze fundamentate, corecte şi coerente.

- *Capitolul V*

Prezintă o aplicație de asistare a procesului de proiectare fizică a unei baze de date, dezvoltată cu scopul de a susține şi evidenția utilitatea şi plusurile introduse de metodologia de proiectare propusă în cadrul capitolului 4 al tezei.

Aplicația este implementată în Delphi 7 cu o bază de date SQL Server 2000 şi este dezvoltată pentru a urma metodologia propusă în cadrul tezei. Ea are rolul de a evidenția utilitatea unei astfel de metodologii şi a folosirii specificațiilor formale şi a validărilor statice în cadrul metodologiei. Prin folosirea unui formalism matematic în cadrul procesului de proiectare fizică a bazei de date devine posibil să se construiască utilitare de asistență a procesului de proiectare. Procesul de proiectare este astfel uşurat şi el devine şi mult mai riguros, deciziile de proiectare luându-se în urma unor verificări şi validări matematice. Erorile umane de proiectare pot fi astfel eliminate din cadrul procesului de proiectare. Produsul final este calitativ superior şi se realizează şi economie de timp şi de bani, pe de o parte pentru că procesul de proiectare fiind asistat se desfaşoară mai rapid, iar pe de altă parte deoarece etapa de testare a bazei de date, care are ca scop descoperirea şi eliminarea erorilor de proiectare, poate fi redusă foarte mult sau complet eliminată.

Tot în cadrul capitolului sunt prezentate şi rezultatele obţinute în urma aplicării metodologiei de proiectare, propuse în capitolul 4, la proiectarea unei baze de date specifică domeniului vânzărilor en-gros şi en-detail şi al analizei vânzărilor, bază de date care a fost folosită şi ca bază de date test în cadrul analizelor făcute în cadrul capitolului 3. Proiectarea bazei de date a fost realizată în paralel de două echipe de proiectanţi, una folosind metodologia de proiectare propusă precum şi utilitarul de proiectare dezvoltat pentru aplicarea acesteia, iar cealaltă folosind o metodologie de proiectare fizică a bazei de date similară celei propuse de Connolly în [Conno 01] la nivelul căreia analiza tranzacţiilor s-a făcut folosind metoda hărţilor de utilizare a tranzacţiilor. Prin aplicarea metodologiei de proiectare propusă în capitolul 4 costul procesului de proiectare s-a redus, iar deciziile de proiectare au putut fi luate prin eliminarea subiectivismului proiectantului bazei de date. Schema fizică obţinută pentru baza de date este una superioară.

- *Capitolul VI*

Sintetizează contribuţiile aduse în cadrul tezei în domeniul proiectării bazelor de date relaţionale şi relaţional-obiectuale. Proiectarea fizică a bazei de date este abordată astfel încât să se obţină o creştere a performanţei bazei de date în etapa operaţională a ciclului său de viaţă.

Procesul de dezvoltare a bazei de date

2.1. Modele ale procesului software

Ingineria software este acea ramură a ingineriei care se ocupă cu dezvoltarea sistemelor software, pentru domenii tehnice sau nu, pe baza unor principii inginereşti. Acestea propun rezolvarea problemelor într-o manieră strictă, urmărind o secvență de activități precum: analiză, specificare, proiectare, implementare, producție, asigurare a calității, întreținere, fiabilitate, etc., care sunt menite a asigura producerea unor produse calitativ superioare la un preț de cost acceptabil. Inginerul are la dispoziție un set de utilitare şi metodologii pentru facilitarea acestui proces structurat.

Ca orice alt proces ingineresc, şi ingineria software trebuie să asigure producerea unor produse de înaltă calitate pornind de la resurse limitate şi respectând un anumit proces tehnologic. Un produs software realizat inginereşte trebuie să asigure serviciile cerute de beneficiarii săi. El trebuie, de asemenea, să fie fiabil, uşor de întreținut, eficient şi să dispună de o interfață atractivă, uşor de utilizat şi adecvată cerințelor utilizatorilor.

Procesul tehnologic de realizare a produselor software poartă denumirea de *proces de dezvoltare software* sau *ciclu de viață al produsului software.* Acesta descrie nivelele de dezvoltare necesare, plecând de la etapa conceptuală a sistemului software până la implementarea şi întreținerea sa operațională. Fiecare organizație producătoare de software îşi are propriul proces de dezvoltare a software-ului, dar aceste particularizări de obicei urmează

unele modele abstracte ale ciclului de viață. Ian Sommerville în [Somme 96-1],[Somme 96-2] definește două mari clase de modele ale ciclului de viață pentru un produs software:

- modele bazate pe specificare (specification-based models)
- modele de dezvoltare evolutive (evolutionary development models)

2.1.1. Modele bazate pe specificare

Modelele bazate pe specificare folosesc metode de specificare formală sau in-formală a cerințelor utilizatorilor, pe care apoi le proiectează şi le implementează. Procesul de dezvoltare a software-ului este alcătuit din etape succesive care utilizează aceste specificații.

Marele dezavantaj al acestor modele este acela că, beneficiarii sistemului software nu sunt, de regulă, capabili să specifice de la început complet cerințele şi că acestea se modifică de-a lungul procesului de dezvoltare.

2.1.1.1. Modelul „waterfall"

Modelul primar al ciclului de viată propus de Royce în [Royce 70] în anul 1970 este denumit astăzi *modelul cascadă* (waterfall). Procesul software este văzut de acest model ca fiind o succesiune de etape sau activități. După ce fiecare etapă este definită şi parcursă, ea este „înghețată" şi procesul de dezvoltare continuă prin trecere la etapa următoare. Etapele modelului cascadă sunt:

- *analiza şi specificarea cerințelor* - serviciile asigurate de către sistem, constrângerile sub care trebuie să opereze, precum şi scopurile pe care trebuie să le atingă sunt stabilite prin consultarea tuturor categoriilor de utilizatori ai sistemului. Toate acestea trebuie apoi definite într-o manieră care să fie uşor de înțeles atât de utilizatorii cât şi de dezvoltatorii sistemului.
- *proiectarea sistemului şi a software-ului* – este stabilită arhitectura de ansamblu a sistemului, atât pe partea de hardware cât şi pe cea de software. Proiectarea software implică reprezentarea funcțiilor software ale sistemului, astfel încât ele să poată fi transformate într-unul sau mai multe programe executabile.

- *implementarea şi testarea unităţilor* – unităţile de program identificate şi proiectate în etapele anterioare sunt implementate într-un anumit limbaj de programare. Fiecare unitate în parte trebuie testată pentru a se verifica dacă îşi respectă specificaţiile.
- *integrarea şi testarea sistemului* – unităţile de program sunt integrate şi testate ca un sistem întreg, pentru a se verifica asigurarea cerinţelor iniţiale. După ce testarea este încheiată, produsul software este livrat beneficiarului.
- *operarea şi întreţinerea* – sistemul este instalat şi pus în exploatare. Intreţinerea lui poate necesita corectarea erorilor care nu au fost descoperite în fazele anterioare de testare, îmbunătăţirea lui la nivelul unora dintre unităţile de program sau încorporarea unor noi cerinţe care nu fuseseră iniţial descoperite.

Marele dezavantaj al acestui model este acela că, datorită „înghețării" fazelor, se poate ajunge la un produs final care să nu respecte cerinţele utilizatorilor, sau care să fie rău structurat deoarece problemele de proiectare au fost rezolvate prin trucuri de implementare. Critici ale modelul waterfall pot fi găsite în [Gladde 82] şi [McCra 82].

2.1.1.2. Modelul incremental

Modelele incrementale propuse în [Mills 87-1] sunt o extindere a modelului waterfall. Ele încearcă să asigure o stabilitate în dezvoltare, permiţând totuşi modificarea cerinţelor sistemului de-a lungul procesului de dezvoltare. Acest model propune partiţionarea funcţionalităţii sistemului într-o serie de paşi („*increments*") care sunt dezvoltaţi şi distribuiţi succesiv. După ce este dezvoltat un pas, specificarea sa este „înghețată" şi doar specificaţiile pentru paşii următori pot fi modificate. Utilizatorii primesc o versiune timpurie a sistemului şi pot experimenta asupra acesteia reuşind să-şi clarifice astfel cerinţele cu privire la pasul următor din dezvoltarea sistemului.

2.1.1.3. Modelul „Cleanroom"

Modelul „Cleanroom" [Mills 87-2], [Linger 94] se bazează pe modelul incremental, dar fiecare pas (increment) este specificat în mod formal şi este verificat utilizând tehnici de verificare statică. Filosofia modelului se bazează pe evitarea defectelor software mai degrabă decât pe detectarea şi repararea acestora. De-a lungul etapelor de dezvoltare a sistemului se

folosesc tehnici de verificare statică [Cobb 90], care asigură dezvoltarea lipsită de erori a software-ului. In locul testării unităților de program şi a modulelor, componentele software sunt formal specificate şi verificate matematic pe măsură ce sunt dezvoltate, astfel încât să rezulte un produs software lipsit de erori.

Această metodă a condus la realizarea de sisteme cu un număr scăzut de defecte la nivelul produsului final livrat beneficiarului [Selby 95].

Modelele bazate pe specificare sunt potrivite pentru dezvoltarea proiectelor mari. Subsistemele hardware şi software pot fi dezvoltate în paralel în cadrul aceleiaşi organizații sau a unor organizații diferite.

2.1.2. Modele de dezvoltare evolutive

Modelele de dezvoltare evolutive pleacă de la premisa că cerințele complete ale sistemului software nu pot fi cunoscute în fazele de început ale ciclului de viață. Prin urmare, etapele modelelor evolutive sunt:

- *formularea unei descrieri generale a sistemului* – descrierea nu trebuie să fie nici completă nici consistentă, ea trebuie să dea doar un cadru generic asupra a ceea ce va fi sistemul.
- *dezvoltarea sistemului cât mai rapid cu putință* – pornind de la descrierea cadrului general este construită o primă variantă a sistemului.
- *evaluarea acestui sistem împreună cu utilizatorii* – sistemul rezultat este evaluat împreună cu beneficiarii lui şi este modificat până când îndeplineşte cerințele acestora.

Şi în cazul modelelor evolutive există dezavantaje:

- sunt centrate pe beneficiar, astfel că nu se acordă suficientă prioritate cerințelor de organizare;
- modificările repetate ale sistemului software îi degradează structura, astfel încât rezultatul final este deseori dificil şi costisitor de modificat. Aceasta face ca

întreținerea software-ului să fie costisitoare şi adesea el trebuie complet rescris după un ciclu scurt de viață.

- procesul de dezvoltare nu are o vizibilitate mare şi este greu pentru managerul de proiect să stabilească cât de bine decurge dezvoltarea. In multe organizații în care managementul este principala problemă, acest model de dezvoltare a sistemului este refuzat.

2.1.2.1. Modelul programării explorative

In cadrul acestui model un sistem operațional este dezvoltat cât mai repede cu putință şi apoi el este modificat până când funcționează într-un mod corespunzător. Acest mod de lucru este cu precădere utilizat în inteligența artificială unde beneficiarii nu pot formula detaliat cerințele şi unde funcționarea adecvată a sistemului este principalul scop al proiectanților sistemului, nu realizarea unui sistem fără erori [Agres 86]]. Dezavantajul acestei metode este încapsularea deciziilor de proiectare. In primul rând, unele din deciziile fazelor timpurii se pot dovedi a fi eronate şi ele nu mai pot fi înlăturate niciodată din sistem. In al doilea rând, deoarece comportarea unui sistem interactiv este puternic dependentă de cunoştiințele înglobate în sistem, versiunile de început ale sistemului nu vor include toate cunoştiințele ce vor fi incluse în versiunea finală.

2.1.2.2. Modelul prototipului

Această abordare este similară cu programarea explorativă, deoarece prima ei fază implică dezvoltarea unui sistem operațional, prototipul, care va fi utilizat pentru ca utilizatorul să experimenteze pe el şi să-şi poată defini şi specifica cerințele. Obiectivul acestei dezvoltări constă în stabilirea completă şi consistentă a cerințelor sistemului [Ince 87]. Această fază este urmată de o reimplementare a produsului software, de regulă folosind un alt limbaj de programare decât cel folosit în prima fază, pentru a produce sistemul software dorit, calitativ superior.

Dezavantajele modelului de programare explorativă sunt preluate şi de modelul prototipului. Pe lângă acestea mai apare şi un alt dezavantaj major, şi anume că dezvoltarea sistemului se concentrează doar asupra caracteristicilor de suprafață ale proiectării şi nu asupra proiectării în profunzime a sistemului, ceea ce nu garantează că sistemul produs prezintă caracteristicile necesare de interacțiune.

Modelele evolutive sunt potrivite pentru dezvoltarea sistemelor interactive, care pun accent pe interfața cu utilizatorul, şi în cazul sistemelor de inteligență artificială ale căror cerințe nu pot fi anticipate.

2.1.3. Concluzii

Nu există un model ideal şi nu are sens să se încerce înglobarea tuturor modelelor într-unul singur. Fiecare dintre aceste modele se poate dovedi a fi soluția cea mai bună în anumite situații. Alegerea celui mai potrivit model de dezvoltare software trebuie să țină seama de caracteristicile organizației care dezvoltă sistemul, de tipul şi trăsăturile esențiale ale sistemului care se vrea dezvoltat, precum şi de capacitatea personalului care va face dezvoltarea.

Modelele bazate pe specificare sunt potrivite pentru sistemele mari la care cerințele şi arhitectura pot să fie specificate aproape complet în fazele timpurii ale procesului de dezvoltare a software-ului.

Modelele de dezvoltare evolutive sunt potrivite pentru sistemele la care cerințele nu pot fi specificate complet în fazele timpurii ale procesului de dezvoltare şi pentru sistemele cu precădere interactive, care pun accent pe interfața cu utilizatorul.

De cele mai multe ori însă, în special pentru sistemele mari, modelele hibride sunt cele mai eficiente. La nivelul acestor sisteme, componentele cunoscute sunt dezvoltate utilizând modele de specificare, iar componentele ale căror cerințe sunt dificil de precizat sunt dezvoltate utilizând modele evolutive [Sommer 87-2]. La nivelul procesului de dezvoltare a bazei de date modelele bazate pe specificații sunt cele mai adecvate şi cele mai des folosite. O bază de date îşi are propriul ciclu de viață care evidențiază etapele pe care baza de date le parcurge de-a lungul existenței ei. Cele mai des folosite modele de dezvoltare a unei baze de date sunt bazate pe modelul „waterfall„ sau pe derivate ale acestuia.

2.2. Specificații formale

O specificație software formală este o specificație exprimată într-un limbaj al cărui vocabular, sintaxă şi semantică sunt formal definite.

Sunt o serie de avantaje ale utilizării unei astfel de specificații:

- permite o mai corectă înțelegere a cerințelor sistemului şi a proiectării acestuia, ceea ce reduce erorile şi omisiunile şi oferă o bază corectă pentru o proiectare elegantă a software-ului;
- specificațiile formale sunt entități matematice şi pot fi analizate utilizând metode matematice. Poate fi dovedită astfel consistența şi caracterul complet al specificațiilor, precum şi faptul că implementarea este conformă cu specificațiile.
- specificațiile formale pot fi procesate automat. Se pot construi unelte software care să asiste la dezvoltarea acestora.

In figura 1 se poate observa cum costul procesului software este afectat de utilizarea specificațiilor formale [Sommer 87-2].

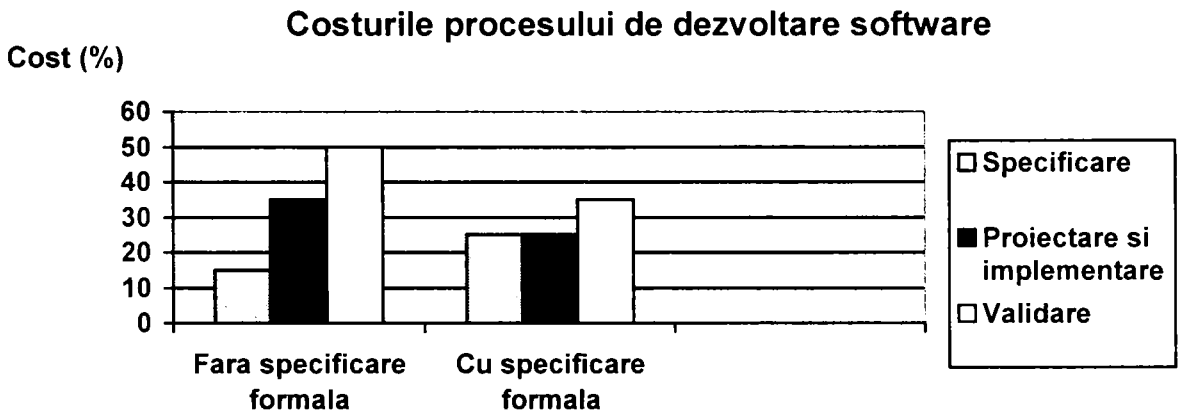


Figura 1. Influența specificării formale asupra costului procesului de dezvoltare software

Atunci când se utilizează un proces convențional, costurile de validare ating aproximativ 50% din costul de dezvoltare, şi costurile de proiectare şi implementare sunt de circa două ori mai mari decât costurile de specificare. Dacă se utilizează o metodă de specificare formală, costurile de specificare şi de proiectare şi implementare sunt comparabile pe ansamblu, dar costurile de validare scad considerabil.

Specificațiile formale se pot utiliza atât la nivelul etapei de specificare a cerințelor cât şi la nivelul etapei de proiectare a sistemelor software. Printre metodele formale se numără diagramele fluxului de date ale lui DeMarco [DeMar 78]], specificațiile algebrice structurate ale lui Cohen [Cohen 86]], specificațiile bazate pe model pentru care există limbaje de specificare precum limbajul Z al lui Spivey [Spive 92].

2.3. Modelarea datelor

Majoritatea sistemelor software mari utilizează baze de date cu informații. In unele cazuri, aceste baze de date există independent de sistemul software, în alte cazuri ele sunt create special pentru respectivul sistem software. Prin urmare, o parte a activității de modelare a sistemului se concentrează asupra definirii unui model logic al datelor care sunt manipulate de sistem.

Dicționarul Webster definește noțiunea de *model* ca o „descriere sau analogie utilizată pentru a vizualiza ceva ce nu poate fi observat în mod direct". Cu alte cuvinte, un *model* este o abstracție a unui obiect mult mai complex al lumii reale. Un *model de date* este o reprezentare relativ simplă, de regulă grafică, a unei structuri reale complexe a datelor. Funcția principală al unui model este aceea de a ne ajuta să înțelegem complexitatea unui mediu real. In mediul unei baze de date, un model al datelor reprezintă structura datelor şi caracteristicile acestora, relațiile existente între date, constrângerile pe care datele trebuie să le accepte, precum şi transformările care se produc asupra datelor.

La ora actuală cel mai utilizat model de date este modelul relațional al lui Codd [Codd 70]. El are un solid fundament matematic, provenit din teoria matematică a mulțimilor, şi deține supremația la nivel mondial. Ca alte modele de date se pot aminti modelul ierarhic şi modelul rețea, precum şi mai noul model obiectual. Există şi modele semantice ale datelor, precum modelul Entitate-Relație [Chen 76], modelul relațional extins [Codd 79], modelul SDM [Hammer 81], care sunt cu precădere utilizate în procesele de proiectare a bazei de date. Cel mai influent dintre aceste modele semantice este modelul Entitate-Relație al lui Chen [Chen 76], care este folosit în cadrul procesului de proiectare conceptuală a bazei de date.

2.4. Procesul de proiectare a bazei de date

Proiectarea bazei de date este parte componentă a unui process mai larg numit proiectarea sistemelor informaționale. La nivelul unui sistem informațional datele nu sunt doar colectate, memorate şi regăsite, ci sunt de asemenea transformate în informații valoroase. Utilizând aceste informații, persoanele de decizie din cadrul companiilor ce dețin aceste date, sunt capabile să ia acele decizii care să permită organizațiilor lor să prospere şi să se dezvolte.

Deoarece baza de date este o componentă fundamentală a sistemului informațional, dezvoltarea şi utilizarea sa trebuie privite din perspectiva cerințelor complexe ale organizației care o utilizează. Baza de date este practic fundamentul pe care se clădeşte viitorul sistem

informațional. Şi pentru că orice construcție robustă are nevoie de o fundație solidă, așa și sistemul informațional trebuie să se sprijine pe o bază de date adecvată. O bună proiectare a bazei de date este esențială pentru succesul sistemului informațional. Prin urmare, ciclul de viață al sistemului informațional este inevitabil legat de ciclul de viață al sistemului de baze de date care îl susține. Etapele ciclului de viață al unui sistem informațional se poate considera că sunt: planificarea, colectarea, analiza şi specificarea cerințelor, proiectarea sistemului, inclusiv a bazei de date, realizarea prototipului sau implementarea, testarea, conversia şi întreținerea operațională. Ca atare, proiectarea bazei de date este parte integrantă a procesului de proiectare a sistemului informațional. La rândul său ciclul de viață al bazei de date cuprinde, în esență, următoarele etape (figura 2) [Rob 95]:

- *planificarea bazei de date* – se planifică modul în care etapele ciclului de viață pot fi realizate cel mai eficient, avându-se în vedere munca care trebuie efectuată, resursele cu care se poate efectua şi banii cu care trebuie plătit totul;
- *definirea sistemului* – se specifică scopul şi limitele aplicației de tip bază de date, utilizatorii acesteia şi domeniul de aplicabilitate;
- *colectarea, analiza şi specificarea cerințelor* – sunt colectate, analizate şi specificate formal sau neformal cerințele tuturor categoriilor de utilizatori şi a domeniilor de aplicație;
- *proiectarea bazei de date* – se realizează un proiect al bazei de date ce va cuprinde reprezentarea datelor şi a relațiilor dintre ele, un model de date ce trebuie să accepte efectuarea oricărei tranzacții necesare asupra datelor şi să permită realizarea cerințelor de performanță ale sistemului;
- *realizarea prototipului* – opțional se poate construi un model de lucru al aplicației de tip bază de date, care să poată permite proiectanților şi utilizatorilor acesteia să vizualizeze şi să evalueze modul în care va funcționa şi va fi configurată interfața sistemul;
- *implementarea* – se realizează fizic proiectul bazei de date şi al aplicației care o gestionează, prin utilizarea limbajului DDL, DML, precum şi a unor limbaje de programare gazdă, de obicei de nivel înalt;
- *conversia şi încărcarea datelor* – sunt transferate în noua bază de date oricare date deja existente şi sunt convertite aplicațiile existente, astfel încât ele să poată funcționa în cadrul noului sistem;

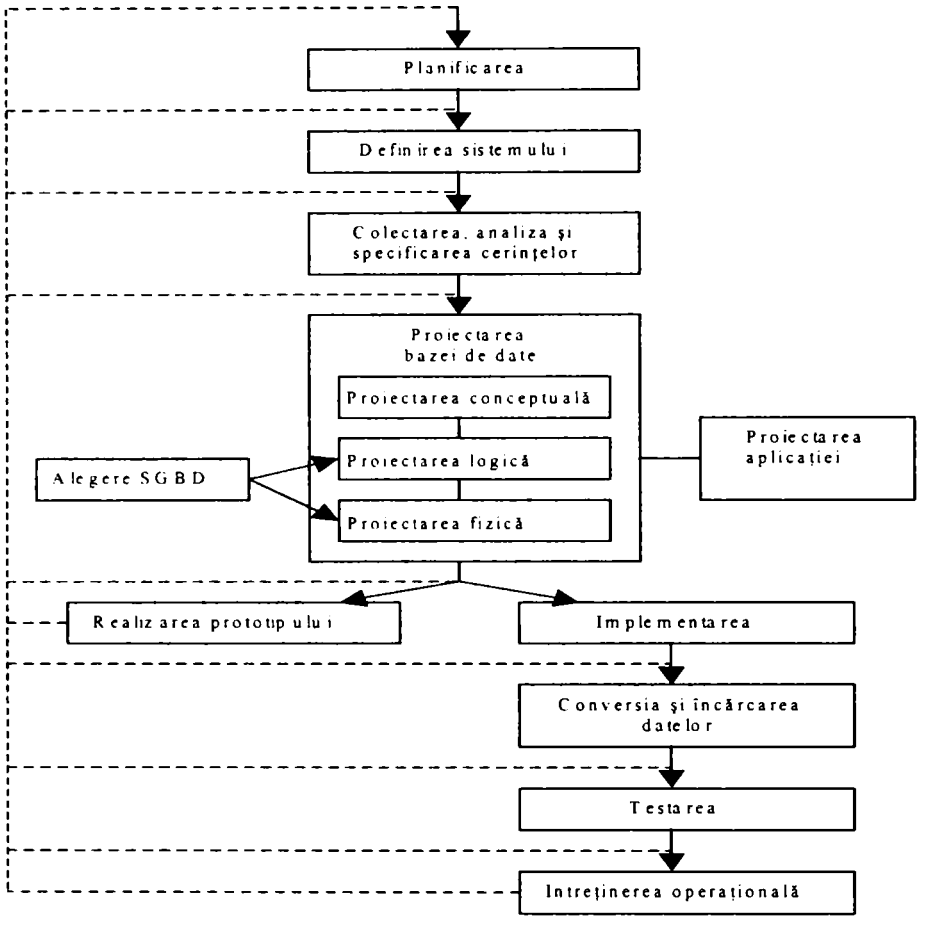


Figura 2. Ciclul de viaţă al unei baze de date

- *testarea* – asupra bazei de date şi a programelor de aplicaţii sunt executate operaţiile curente din exploatare, astfel încât să se scoată la iveală erorile din programele de aplicaţii şi din structura bazei de date;
- *întreţinerea operaţională* – după ce baza de date devine operaţională, ea este monitorizată în permanenţă pentru a se identifica degradarea performanţelor şi necesitatea reglării şi reorganizării acesteia, precum şi pentru a se identifica noile cerinţe care trebuie încorporate în aplicaţia de tip bază de date.

Aşa cum se observă şi în ciclul de viaţă al unei baze de date, procesul de proiectare al bazei de date este divizat în trei subetape majore, şi anume:

- proiectarea conceptuală a bazei de date;
- proiectarea logică a bazei de date;
- proiectarea fizică a bazei de date.

2.4.1. Proiectarea conceptuală a bazei de date

Proiectarea conceptuală a bazei de date este procesul prin care se construieşte un model de date conceptual pentru datele utilizate de acea companie sau organizaţie, model de date care este independent de orice detalii fizice de implementare a datelor, nu este influenţat de SGBD-ul care va fi ales pentru implementarea bazei de date, nu depinde de programele de aplicaţii ce vor fi dezvoltate, de limbajele de programare ce vor fi utilizate la dezvoltarea acestor programe de aplicaţii şi nici nu este dependent de platforma hardware pe care va rula sistemul. În [Rob 95], [Conno 01], [Popes 01], [Date 03], [Steph 00], precum şi în [Mitea 02-1], [Mitea 02-2], [Mitea 02-3] se pot găsi referinţe detaliate asupra procesului de proiectare conceptuală a bazei de date.

Pornindu-se de la cerinţele specificate ale bazei de date şi ale aplicaţiei, în această etapă se construieşte modelul de date conceptual local, pentru fiecare vedere a grupurilor de utilizatori identificaţi. Trebuie identificate entităţile de interes, relaţiile existente între aceste entităţi şi tipul fiecărei relaţii în parte, atributele necesare şi asocierea acestor atribute cu entităţile şi relaţiile existente. De asemenea, trebuie determinat domeniul fiecărui atribut, precum şi atributele care intră în componenţa cheilor candidat şi a cheilor primare.

In mod tradiţional proiectanţii de baze de date se bazează pe buna lor judecată atunci când dezvoltă proiectul unei baze de date. Din păcate, de multe ori acestă bună judecată se dovedeşte a fi bună numai în ochii lor, ceea ce conduce la proiecte ineficiente. Chiar şi în situaţia în care buna judecată există, ea a fost dobândită, din păcate, numai după multe încercări presărate cu foarte multe erori. Până la ora actuală conceptul de bună proiectare a unei baze de date nu a fost în mod adecvat definit, astfel încât să fie unanim recunoscut. De aceea, proiectarea bazei de date este atât artă cât şi ştiinţă. O modalitate de a apropia cât mai mult procesul de dezvoltare a unei baze de date de ştiinţă este aceea de a folosi metode de specificare formală în cadrul procesului de dezvoltare a bazei de date. Acestea permit o înţelegere completă şi coerentă a cerinţelor şi permit elaborarea unui proiect cu cât mai puţine erori şi omisiuni. Modelul conceptual al bazei de date este reprezentat formal prin intermediul diagramelor E-R [Chen 76].

Utilizatorii bazei de date au, de regulă, viziuni diferite asupra datelor. Necesităţile lor vis-à-vis de datele stocate în baza de date, de posibilitătile de prelucrare a acestora şi de transformare a lor în informaţii valoroase sunt diferite. Există mai multe scheme conceptuale locale ale datelor, fiecare dintre ele fiindu-i caracteristică unui grup de utilizatori. Aceste scheme conceptuale sunt exprimate formal în termenii modelului semantic E-R al datelor.

Modelele conceptuale locale ale datelor trebuie revizuite şi validate împreună cu utilizatorii lor.

2.4.2. Proiectarea logică a bazei de date

Proiectarea logică a bazei de date este procesul prin care modelul conceptual al datelor este transformat într-un model logic al datelor, în funcție de modelul de date ales pentru a fi folosit la nivelul bazei de date. In continuare, acest model de date este independent de orice detalii cu privire la modul de implementare fizică a bazei de date. In [Rob 95], [Conno 01], [Popes 01], [Date 03], [Steph 00], precum şi în [Mitea 02-2], [Mitea 02-3] există referințe detaliate în ceea ce priveşte procesul de proiectare logică a bazelor de date relaționale.

Modelul de date conceptual, creat în faza precedentă, este rafinat şi transpus într-un model de date logic. Acesta este influențat de modelul de date avut în vedere pentru baza de date, dar este în continuare independent de detaliile fizice de implementare ale acesteia.

Modelele de date conceptuale locale sunt transformate în modele de date logice, care sunt în permanență testate şi validate conform cerințelor utilizatorilor. Au fost definite reguli de transformare care trebuie aplicate modelului conceptual al datelor pentru a-l transforma în model logic în funcție de tipul modelului ales pentru date. In [Conno 01], [Popes 01] se pot găsi astfel de reguli pentru modelul relațional al datelor. Normalizarea este utilizată pentru a testa corectitudinea modelului de date logic aşa cum se poate observa în [Conno 01], [Date 03], [Steph 00], precum şi în [Mitea 02-3]. Ea garantează că relațiile derivate din modelul de date nu prezintă redundanță, care să conducă la anomalii de actualizare în faza operațională a bazei de date. Modelul de date logic este, de asemenea, analizat pentru a se garanta că acceptă tranzacțiile specificate de utilizatori. Tot la nivelul acestei etape sunt definite şi restricțiile de integritate asupra bazei de date. Modelele de date logice locale sunt integrate într-un model de date logic global, care la rândul lui este verificat, revizuit şi validat împreună cu grupurile de utilizatori. In conformitate cu acest model de date logic global este redesenată diagrama finală E-R.

Şi acest nivel al proiectării bazei de date poate fi abordat pe baze formale pentru a creşte calitatea produsului final. Se folosesc diagramele E-R care sunt rafinate pentru a se obține schema logică globală a bazei de date.

Modelul de date logic reprezintă sursa de informații pentru faza de proiectare fizică a bazei de date, el pune la dispoziția proiectantului bazei de date logice un instrument valoros

pentru efectuat negocieri, care sunt foarte importante pentru ca proiectarea bazei de date să fie un proces eficient. De asememea, modelul de date logic este deosebit de important pentru etapa de întreținere operațională din ciclului de viață al bazei de date, pentru că el permite efectuarea corectă a modificărilor impuse de întreținerea corectivă şi adaptivă a sistemului.

2.4.3. Proiectarea fizică a bazei de date

Proiectarea fizică a bazei de date este procesul prin care se decide asupra structurilor fizice de memorare ale fişierelor de date şi asupra metodelor de acces la aceste fişiere, pentru a obține un acces cât mai rapid şi eficient la date. In [Rob 95], [Conno 01], [Popes 01], [Steph 00], precum şi în [Mitea 02-2], [Mitea 02-3] sunt prezentate unele aspecte legate de procesul de proiectare fizică a bazelor de date.

Proiectarea fizică a bazei de date este cea de a treia etapă a procesului de proiectare a bazei de date, în care proiectantul stabileşte cum va fi implementată baza de date în conformitate cu SGBD-ul ales. Dacă în etapa anterioară de proiectare SGBD-ul nu era important, decât din punctul de vedere al modelului de date suportat, acum acesta este esențial în luarea deciziilor cu privire la proiectul fizic al bazei de date. In realizarea proiectării fizice, trebuie mai întâi ales sistemul de gestiune a bazei de date, şi în funcție de caracteristicile acestuia este croit proiectul fizic al bazei de date. Intre proiectarea fizică şi cea logică există o reacție inversă, deoarece în decursul proiectării fizice sunt luate decizii de îmbunătățire a performanțelor, care pot afecta structura modelului de date logic.

Proiectarea fizică a bazei de date descrie cum va fi implementat fizic proiectul logic al bazei de date, şi anume care vor fi relațiile şi restricțiile de integritate implementate în baza de date şi cum se va face această implementare, vor fi identificate structurile de stocare a datelor ce vor fi folosite şi metodele de acces la aceste structuri, astfel încât să se realizeze performanțe optime la nivelul sistemului, precum şi strategiile şi mecanismele care vor fi folosite pentru asigurarea securității datelor.

Pentru sistemele mari este ideal ca proiectarea conceptuală şi logică a bazei de date să fie separată de proiectarea fizică a bazei de date, deoarece tratează subiecte diferite şi sunt efectuate în momente de timp diferite. În mod normal, etapele proiectării conceptuale şi logice stabilesc *ce* trebuie făcut, pe când etapa proiectării fizice stabileşte *cum* o să se facă. Totodată ele necesită abilități de proiectare diferite, care, de regulă, se regăsesc la proiectanți diferiți.

Este important ca şi această etapă a proiectării bazei de date să fie parcursă inginerește, respectând reguli clare, care să lase cât mai puțin loc unor decizii personale ale proiectantului, bazate pe flerul şi experiența acestuia.

Introducerea unor metode formale de proiectare şi la nivelul acestei etape de proiectare a bazei de date poate fi soluția care să conducă la produse finale calitativ superioare, care să asigure performanțe sporite în exploatare pentru baza de date.

2.4.4. Concluzii

Procesul de proiectare a unei baze de date este divizat pe trei nivele. El este un proces ingineresc, şi ca atare trebuie să rezolve problemele într-o manieră stictă şi riguroasă, asigurând un raport calitate/preț acceptabil pentru produsul final. Reducerea costurilor şi creşterea calității unui proces de proiectare se poate realiza prin folosirea unor metode formale de specificare aşa cum se poate observa şi în figura 1. Aplicarea acestor metode permite eliminarea inconsistențelor şi ambiguității din cadrul procesului de proiectare, cerințele pot fi înțelese şi urmărite consecvent şi corect, astfel încât să se asigure un produs final de o calitate superioară. Aceste principii se pot aplica şi în cazul procesului de proiectare a bazelor de date. O bază de date cu o structură corect definită, având metode de acces riguros alese, oferă o performanță sporită în exploatare, ceea ce duce la creşterea gradului de satisfacție a utilizatorilor ei.

2.5 Metodologii de proiectare

Pentru a facilita procesul de proiectare, sunt create metodologii care ghidează proiectantul de-a lungul diferitelor etape ale procesului de proiectare.

O *metodologie de proiectare* conține diferiți pași, care trebuie urmați de proiectant, de-a lungul fiecărei etape de proiectare şi care-l ajută pe acesta să aleagă cele mai bune tehnici şi soluții pentru rezolvarea problemelor cu care se confruntă [Conno 01]. Şi la nivelul procesului de proiectare a bazelor de date au fost propuse metodologii de proiectare, care asistă proiectantul bazei de date de-a lungul procesului de proiectare. Utilizând o metodologie de proiectare, proiectantul poate planifica, gestiona controlul şi evalua proiectele asupra bazei de date mult mai uşor şi mai exact.

Pe parcursul întregii metodologii, utilizatorii bazei de date joacă un rol esențial în verificarea şi validarea modelului de date şi a documentației care susține acest model.

Proiectarea bazelor de date este un proces iterativ, care are un punct de plecare şi o succesiune aproape infinită de rafinări. Este posibil ca deciziile luate în cadrul unei etape să modifice deciziile luate în cadrul unei etape anterioare. Cu cât numărul de rafinări la care este supusă baza de date este mai mare, cu atât durata procesului de proiectare va fi mai mare. Este ideal ca etapele să fie parcurse cât mai complet, astfel încât necesitatea unor rafinări multiple să fie cât mai scăzută. Metodologia trebuie să acționeze ca un cadru care să ghideze în mod eficient proiectantul prin activitatea de proiectare a bazei de date.

2.5.1. Metodologii de proiectare a bazelor de date

Connolly propune în [Conno 01] o metodologie de proiectare a bazelor de date pentru baze de date relaționale care conține în esență următorii paşi:

Proiectarea conceptuală a bazei de date

Pasul 1. Construirea modelului de date conceptual local pentru fiecare vedere a utilizatorilor

1.1 Identificarea tipurilor de entități

1.2 Identificarea tipurilor de relații

1.3 Identificarea şi asocierea atributelor cu tipurile de entități sau relații

1.4 Determinarea domeniilor atributelor

1.5 Determinarea atributelor chei candidat şi chei primare

1.6 Specializarea/generalizarea tipurilor de entități

1.7 Trasarea diagramei E-R

1.8 Revizuirea modelului de date conceptual local, împreună cu utilizatorul.

Proiectarea logică a bazei de date pentru modelul relațional al datelor

Pasul 2. Construirea şi validarea modelului de date logic local pentru fiecare vedere a utilizatorului

2.1 Transpunerea modelului de date conceptual local în modelul de date logic local

2.2 Extragerea relațiilor din modelul de date logic local

2.3 Validarea modelului cu ajutorul normalizării

2.4 Validarea modelului conform tranzacțiilor utilizatorului

2.5 Trasarea diagramei E-R

2.6 Definirea constrângerilor de integritate

2.7 Revizuirea modelului de date conceptual local împreună cu utilizatorul

Pasul 3. Construirea şi validarea modelului de date logic global

3.1 Imbinarea modelelor de date logice locale în cadrul modelului global

3.2 Validarea modelului de date logic global

3.3 Verificarea în vederea dezvoltării viitoare

3.4 Trasarea diagramei E-R finale

3.5 Revizuirea modelului de date logic global împreună cu utilizatorii

Proiectarea fizică a bazei de date relaţionale

Pasul 4. Transformarea modelului de date logic global pentru sistemul SGBD ţintă

4.1 Proiectarea relaţiilor de bază pentru sistemul SGBD ţintă

4.2 Proiectarea constrângerilor întreprinderii pentru sistemul SGBD ţintă

Pasul 5. Proiectarea reprezentării fizice

5.1 Analiza tranzacţiilor

5.2 Alegerea organizării fişierelor

5.3 Alegerea indexurilor secundare

5.4 Introducerea unei redundanţe controlate

5.5 Estimarea necesarului de spaţiu pe disc

Pasul 6. Proiectarea mecanismelor de securitate

6.1 Proiectarea vederilor utilizatorilor

6.2 Proiectarea regulilor de acces

Pasul 7. Monitorizarea şi reglarea sistemului operaţional

Prima parte a metodologiei vizează proiectarea conceptuală a bazei de date şi se realizează un model conceptual al bazei de date, care este independent de un model particular al datelor, precum şi de un anume SGBD şi de orice alte consideraţii de ordin fizic. Formalismul folosit este cel al diagramelor Entitate-Relaţie.

Cea de a doua parte a metodologiei vizează proiectarea logică a bazei pentru modelul relațional al datelor. Schema conceptuală a bazei de date este transformată într-o schemă logică în conformitate cu caracteristicile modelului de date ales. Schema logică a bazei de date este în continuare independentă de un anumit SGBD şi de orice alte considerații de ordin fizic. Formalismul folosit este în continuare cel al diagramelor Entitate-Relație. Validarea modelului logic al bazei de date se face prin utilizarea tehnicii de normalizare.

Cea de a treia şi ultimă parte a metodologiei vizează proiectarea fizică a bazei de date relaționale. Proiectantul decide modalitatea prin care va transpune proiectul logic al bazei de date într-un proiect fizic al acesteia, care să poată fi implementat utilizând SGBD-ul țintă. Această etapă este puternic dependentă de un anumit SGBD şi de orice alte considerații de ordin fizic. Proiectantul trebuie să fie complet conştient de funcționalitatea SGBD-ului folosit pentru implementarea bazei de date şi să cunoască avantajele şi dezavantajele fiecărei alternative, pentru a putea selecta strategia de stocare adecvată, care să ia în considerare modul de utilizare al bazei de date.

În [Rob 95], [Date 03], [Steph 00] pot fi găsite, de asemenea, metodologii de proiectare a bazelor de date. Şi la nivelul acestor metodologii, etapa proiectării fizice a bazei de date este în esență similară cu cea prezentată anterior.

2.5.2. Concluzii

Pentru ca procesul de proiectare a unei baze de date să aibă un caracter riguros şi consistent s-au dezvoltat metodologii prin care se susține şi se facilitează acest proces. Metodologia asistă proiectantul în alegerea tehnicilor adecvate fiecărei etape a procesului de proiectare a bazei de date. Proiectantul este, de asemenea, ajutat să-şi planifice, administreze, controleze şi evalueze proiectul de dezvoltare a bazei de date. Metodologia de proiectare oferă o abordare structurată de analiză şi modelare a unui set de cerințe privind baza de date. Ea permite o corectă realizare a procesului de proiectare a bazei de date. Metodologiile de proiectare a bazelor de date existente sunt structurate pe cele trei mari etape ale procesului de proiectare şi conțin paşi succesivi care trebuie urmați de proiectanții de baze de date în cadrul procesului de proiectare. Cerințele de proiectare care trebuie atinse şi rezolvate în cursul procesului de proiectare a bazei de date sunt în general similare în cadrul acestor metodologii, chiar dacă uneori ordinea de abordare a acestora diferă. Calitatea procesului de proiectare a bazei de date, precum şi rezultatul final al procesului de proiectare, adică baza de date, sunt sensibil influențate de utilizarea unei metodologii de proiectare.

2.6. Concluzii finale

Nu există un model ideal al procesului de dezvoltare software, aplicabil tuturor sistemelor. La nivelul bazelor de date modelele bazate pe specificații sunt cele mai adecvate. Reducerea costurilor şi creşterera calității procesului de proiectare se poate realiza prin folosirea unor metode formale de specificare. Etapa de proiectare conceptuală a bazei de date foloseşte ca metodă formală de specificare diagramele E-R. Aceleaşi diagrame E-R sunt folosite ca metodă formală de specificare şi la nivelul etapei de proiectare logică a bazei de date. Introducerea unor metode formale de specificare şi în etapa de proiectare fizică a bazei de date poate fi soluția pentru produse caliativ superioare care să asigure performanțe sporite în exploatare. De asemenea, utilizarea unei metodologii de proiectare creşte calitatea procesului de proiectare şi îmbunătățeşte caracteristicile produsului final al procesului de proiectare.

Etapa proiectării fizice a bazei de date, fiind cea mai apropiată de structura finală a bazei de date, influențează substanțial performanța în exploatare a bazei de date. Parcurgerea acesteia urmărind principii inginereşti nu face decât să sporească calitatea bazei de date şi să asigure o performanță sporită în exploatare a acesteia.

Studiul influenței structurii fizice asupra performanței bazei de date

3.1. Prezentarea problemei

Majoritatea sistemelor de aplicații, dezvoltate astăzi în lumea informaticii, implică utilizarea unei baze de date. De regulă, baza de date constituie fundația pe care sunt dezvoltate aceste sisteme informatice. Şi pentru că un fundament solid este esențial pentru viitorul oricărei construcții, şi în acest domeniu, robustețea şi performanțele asigurate de baza de date sunt esențiale pentru succesul viitoarei aplicații. Alegerea bazei de date este o investiție care se face pe termen lung şi, de aceea, trebuie să i se asigure atenția necesară. Înțelegerea caracteristicilor produselor existente pe piață, precum şi a necesităților aplicației care trebuie dezvoltată este crucială pentru reuşita viitoarei aplicații. Există diverse caracteristici care trebuiesc urmărite atunci când se selectează sistemul de gestiune a bazei de date. În [Mitea 01] pot fi găsite mai multe despre criteriile de evaluare şi selecție a sistemului de gestiune a bazei de date.Transaction Pocessing Council (TPC) a dezvoltat chiar diferite tipuri de benchmark-uri folosite pentru a măsura performanța procesării tranzacțiilor şi performanța bazei de date exprimată în termenii numărului de tranzacții pe care sistemul le poate executa pe unitatea de timp, de exemplu tranzacții pe secundă (tps) sau tranzacții pe minut (tpm). Există mai multe tipuri de benchmark-uri, precum TPC-A şi TPC-B de tip Debit-Credit, TPC-C de tip OLTP, TPC-D de tip DSS, TPC-W de tip EEC, ce sunt folosite de producătorii de

software pentru baze de date pentru testarea şi evaluarea performanțelor produselor lor. Dar, un sistem de gestiune a bazei de date performant nu este neapărat garantul unor performanțe deosebite ale bazei de date în exploatare. Un rol deosebit de important, vis-à-vis de performanțele în exploatare ale unei baze de date, îl joacă structurile de date folosite şi metodele de accesare ale acestora.

Majoritatea sistemelor de gestiune mari a bazelor de date existente astăzi pe piață permit datelor să fie stocate pe mediile de memorare externe folosindu-se diferite structuri de date. De asemenea, accesul la aceste structuri de date se poate face prin mai multe metode de acces. Deoarece benchmark-urile clasice nu analizează aceste aspecte ale structurii fizice a bazei de date şi a influenței pe care o pot exercita ele asupra performanței bazei de date, mi-am propus să studiez tocmai aceste aspecte. Cum trebuie proiectată fizic baza de date pentru a obține performanță maximă la exploatarea ei?

Există o „formulă magică" pentru structura fizică a bazei de date? Care ar fi aceasta?

Acestea sunt întrebările la care am încercat să caut răspuns în cadrul testelor pe care le-am efectuat. Care este structura optimă a datelor şi ce metode de acces trebuie folosite pentru a se obține maximum de performanță la nivelul bazei de date.

Sistemele de gestiune a bazelor de date pe care le-am ales pentru a fi analizate au fost ORACLE 9*i*, SQL Server 2000 şi IBM DB2. Această alegere s-a bazat pe faptul că aceste SGBD-uri sunt lidere pe piața mondială a bazelor de date. Chiar dacă la noi în țară aceste sisteme nu se folosesc încă pe scară foarte largă, în viitor cu siguranță gradul lor de utilizare va fi mult mai mare, odată cu apariția necesității unor aplicații multiuser, cu sute sau chiar mii de utilizatori, care să gestioneze volume mari de date complexe.

Testele au fost efectuate pe o bază de date test specifică unei firme de comerț care efectuează vânzări en-gros şi en-detail. Structura bazei de date a fost astfel creată încât să răspundă cerințelor reale ale unei astfel de aplicații. Datele de test folosite respectă, de asemenea, cerințele şi restricțiile de integritate impuse de aplicația reală. Baza de date test conține relațiile prezentate în anexa 1.

Testele au fost făcute pe un sistem având un server multiprocesor Pentium III la 750 MHz, cu 512 Mo memorie internă şi un HDD de 40 Go, pe care rulează sistemul Microsoft Windows 2000.

In funcție de tipurile structurilor de date şi a metodelor de acces disponibile la nivelul sistemelor testate, testele s-au împărțit pe categorii. Fiecare categorie de teste efectuată a urmărit anumite aspecte ale datelor şi a încercat să găsească un răspuns la întrebarea:

„Cum se poate obține performanță maximă?".

Pentru fiecare tip de structură analizată s-a urmărit identificarea plusurilor şi a minusurilor, astfel încât aceste rezultate să poată fi folosite la luarea deciziilor de proiectare fizică a bazei de date.

In continuare sunt prezentate caracteristicile tehnice ale sistemelor analizate în ceea ce priveşte structurile fizice şi metodele de acces disponibile, precum şi testele efectuate şi rezultatele şi concluziile obținute la final.

3.2. Structuri de date identificate la sistemele analizate

Sistemele de gestiune a bazelor de date supuse analizei au fost Oracle 9*i*, produs al firmei Oracle Corporation, SQL Server 2000, produs al firmei Microsoft, şi DB2, produs al firmei IBM. Au fost identificate şi analizate stucturile de date şi metodele de acces la aceste structuri, pentru cele trei sisteme supuse studiului. Rezultatele analizei efectuate sunt următoarele:

3.2.1. Oracle

Oracle 9*i* este produsul firmei Oracle Corporation, firmă care la ora actuală este lider mondial pe piața bazelor de date [Oracl 02]. Prin urmare, el oferă multiple facilități şi în ceea ce priveşte structurile de date folosite pentru stocarea informațiilor în baza de date şi metodele de acces implementate pentru a utiliza aceste structuri.

In Oracle datele sunt stocate în fişiere de date care pot fi de unul din următoarele tipuri:

- ***tabele heap***

O *tabelă* este forma cea mai comună de organizare a datelor într-o bază de date. Tabelele obişnuite sunt, de regulă, stocate sub forma unor fişiere heap, exceptând cazul în

care tabela este sortată şi se memorează folosind o structură de fişier ordonat [Oracl 92] şi [Oracl 99]. Rândurile de date sunt stocate în blocurile de date ale bazei de date ca şi înregistrări de lungime variabilă. Coloanele unui rând sunt memorate în ordinea în care au fost definite la crearea tabelei, iar coloanele de sfârşit de rând care au valoare nulă nu sunt memorate în fişierul de date. Fiecare rând dintr-o tabelă poate avea, astfel, stocat un număr diferit de coloane în fişierul de date. Fiecare rând din tabelă are un header de rând care conține informații despre numărul de coloane memorate pentru acel rând, pointeri de legătură, şi starea blocării rândului, precum şi rândul efectiv de date. Pentru fiecare coloană a rândului este păstrată informație despre lungimea coloanei şi valoarea memorată în ea. Rândurile adiacente nu necesită spațiu suplimentar între ele.

In tabele pot fi memorate date de tipuri diferite, fie predefinite, fie definite de utilizator. La nivelul tipurilor predefinite există tipurile scalare clasice (CHAR, NCHAR, VARCHAR2, NUMBER, DATE, etc.) precum şi tipuri pentru stocarea obiectelor de mărimi mari (LONG, LONG RAW, LOB, BLOB, etc.), tipurile colecție precum VARRAY şi TABLE (nested table), precum şi tipul referință REF folosit în cazul bazelor de date relațional-obiectuale pentru a implementa identitatea obiectuală [Luers 95], [Schwi 00]. Cu ajutorul componentei Data Cartridges se pot folosi şi tipuri de date complexe, precum cele: text, audio, video, imagine, serii de timp, date spațiale [Oracl 02].

- ***tabele partiționate***

Tabelele partiționate permit construirea de aplicații scalabile. O astfel de tabelă are următoarele caracteristici:

- are una sau mai multe partiții, fiecare dintre ele stocând rândurile din tabelă care au o valoare specifică a *cheii de partiționare*;
- fiecare partiție dintr-o tabelă partiționată este un segment de date şi poate fi localizată într-un spațiu tabel diferit;
- partițiile sunt utile pentru tabelele de date de dimensiuni mari care pot fi interogate sau manipulate utilizând mai multe procese care operează concurent;

Tabelele partiționate permit, deci, unor baze de date de dimensiuni foarte mari să fie descompuse în piese mai mici care sunt mult mai uşor de gestionat. Partiționarea poate

îmbunătăți performanța sistemului pentru că operațiile care trebuie executate asupra datelor pot fi direcționate doar către acele partiții care conțin aceste date [Luers 95], [Honou 98]. Partițiile care nu conțin datele dorite de operație sunt eliminate din start din domeniul de căutare. Astfel, se reduce dramatic cantitatea de date accesată pe disc şi, deci, se scurtează timpii de procesare şi se îmbunătățește utilizarea resurselor sistemului, obținând o îmbunătățire a performanțelor de ansamblu. Partițiile pot îmbunătăți performanța sistemului şi în cazul unor operații de compunere între mai multe tabele partiționate după aceeaşi cheie de partiționare, pentru că operațiile de compunere vor fi aplicate la nivelul fiecărei partiții, rezultând astfel un timp total de efectuare a operației mai mic decât timpul necesar pentru efectuarea operației de compunere între două tabele obişnuite de aceleaşi dimensiuni. De asemenea, partițiile permit execuția paralelă a operațiilor DML la nivelul tabelei, rezultând astfel reducerea timpilor de răspuns în cazul operațiilor ce prelucrează masiv datele din tabele [Thomb 01].

Tabelele partiționate sunt implementate pe structuri de fişiere de tip hash.

Oracle 9*i* oferă mai multe metode de partiționare a tabelelor cu scopul de a satisface cerințele unor categorii cât mai largi de aplicații [Oracl 02].

Tipurile de partiții posibile sunt următoarele:

- *partiții pe intervale de valori* (range) – care utilizează intervalele de valori ale coloanelor ce alcătuiesc cheia de partiționare pentru a încadra rândurile tabelei la nivelul partițiilor. Acest tip de partiționare este deosebit de util în cazul bazelor de date cu informații istorice, precum şi pentru magaziile de date la care periodic un set de înregistrări vechi trebuie înlocuit cu unul nou.
- *partiții de tip hash* (hash) – care aplică o funcție de hash asupra coloanelor ce alcătuiesc cheia de partiționare pentru a încadra rândurile tabelei la nivelul partițiilor.
- *partiții pe liste de valori* (list) – care permit utilizatorilor să controlexe in mod explicit atribuirea rândurilor tabelei la nivelul partițiilor. Aceasta se realizează prin specificarea unei liste de valori discrete pe care le poate lua cheia de partiționare la nivelul fiecărei partiții din tabelă. In funcție de valoarea cheii de partiționare din rândul tabelei şi de apartenența ei la una din listele de valori se face distribuirea rândurilor tabelei la nivelul partițiilor.

- *partiții compuse range-hash* – este o mixare a celor două metode de partiționare, pe intervale de valori şi de tip hash, atunci când trebuie stabilită încadrarea rândurilor tabelei în cadrul partițiilor acesteia.
- *partiții compuse range-list* – este tot o mixare a celor două metode de partiționare, pe intervale de valori şi pe liste de valori, atunci când trebuie stabilită încadrarea rândurilor tabelei în cadrul partițiilor acesteia.

- ***tabele organizate indexat***

O tabelă organizată indexat păstrează toate datele din tabelă într-o structură de tip arbore B [Honou 98], [Schwi 00], [Oracl 02]. Spre deosebire de un arbore B folosit ca fişier index la care la nivelul nodurilor frunză există informație despre valoarea cheii de indexare şi adresa relativă a rândului din tabelă având acea valoare a cheii, o tabelă organizată indexat păstrează la nivelul nodurilor frunză în locul adresei relative a rândului din tabelă chiar acest rând. Astfel, este eliminată necesitatea de a avea un fişier de date şi suplimentar un fişier index asociat lui.

Accesul indexat la o tabelă obişnuită presupune că unul sau mai multe blocuri de date ale fişierului index sunt accesate pentru a regăsi adresele relative ale rândurilor căutate, urmate apoi de accesarea directă a fişierului de date la nivelul acestor înregistrări. In cazul unei tabele organizate indexat este suficientă doar accesarea blocurilor de date care păstrează structura de arbore B pentru a găsi nodurile frunză cu cheia căutată. Odată aceste noduri localizate nu mai este necesară nici o altă operație de acces deoarece întregul rând este disponibil la nivelul nodului frunză.

Tabelele organizate indexat sunt construite având ca şi *cheie de indexare* cheia primară a tabelei. O tabelă organizată indexat poate fi accesată utilizând fie cheia primară în întregimea ei, fie o porțiune de început a cheii primare.

Operațiile DML de adăugare şi ştergere efectuate asupra tabelei conduc la modificarea structurii arborelui B. De asemenea, operațiile DML de modificare pot conduce la modificarea structurii arborelui B, dacă este modificată şi cheia de indexare.

Principalul avantaj al tabelelor organizate indexat îl constituie economia de spațiu de memorare pe care o realizează, pentru că atât indexul cât şi tabela de date sunt memorate în acelaşi segment. Timpii de răspuns sunt şi ei mai mici atunci când interogările se bazează pe valorile cheii primare. In schimb, dacă interogările folosesc alte coloane decât cele ale cheii

primare, care stă la baza construcției tabelei organizate indexat, timpii de răspuns sunt mai mari.

Tabelele organizate indexat suportă aceleaşi funcționalități ca şi tabelele obişnuite. Este posibilă folosirea tipurilor de date din categoria LOB-urilor, sunt permise indexuri secundare, este posibilă partiționarea de tip range şi hash, oferă suport pentru obiecte, permit execuția paralelă a interogărilor, este posibilă crearea de indexuri bitmap, făcând astfel foarte util acest tip de structură de tabelă pentru mediul magaziilor de date, unde tabelele circumstanțiale pot fi create folosind o astfel de structură.

- ***tabele clusterate***

Tabelele obişnuite oferă utilizatorilor un control foarte limitat asupra modului cum datele sunt distribuite în tabelă. Tabelele clusterate, în schimb, permit un anumit grad de control asupra modului cum rândurile sunt memorate în tabele [Schwi 00], [Oracl 02]. Pe baza valorilor unei *chei de clusterare*, rândurile sunt stocate pe cât posibil în acelaşi bloc de date, pentru ca accesul la ele să se facă mult mai rapid, de preferință într-un singur acces la disc.

Un cluster poate fi folosit şi pentru a memora seturi de rânduri înrudite în acelaşi bloc de date. De exemplu, tabelele VANZARI şi MARFURI pot fi memorate într-un cluster şi astfel ele partajează acelaşi segment cluster. Un bloc de date din acest segment poate memora atât rânduri din tabela VANZARI cât şi rânduri din tabela MARFURI. O astfel de structură este eficientă atunci când informațiile din cele două tabele sunt consultate şi prelucrate întotdeauna împreună. In acest caz, printr-o singură accesare a blocului de date se obțin atât datele necesare aflate în tabela VANZARI cât şi cele aflate în tabela MARFURI. Dacă s-ar fi folosit structuri obişnuite de tabele, ar fi fost necesare două accese, unul la blocul de date care conține rândurile tabelei VANZARI şi unul la blocul de date care conține rândurile tabelei MARFURI.

Tabelele clusterate au următoarele caracteristici:

- este folosită o cheie de clusterare pe baza căreia sunt identificate rândurile care trebuie stocate împreună;
- cheia de clusterare poate conține una sau mai multe coloane din tabelă;
- tabelele clusterate au coloane care corespund cheii de clusterare;

- clusterarea este transparentă pentru aplicațiile care utilizează tabelele;
- actualizarea uneia dintre coloanele cheii de clusterare poate conduce la realocarea rândurilor în clustere;
- cheia de clusterare este independentă de cheia primară a tabelei. Tabelele dintr-un cluster pot avea o cheie primară, care poate fi una şi aceeaşi cu cheia de clusterare, sau poate fi diferită.
- accesul aleator la datele clusterate poate fi mai rapid, dar scanarea integrală a tabelelor este în general mai înceată dacă se foloseşte clusterarea decât dacă se folosesc tabelele obişnuite.

Există două tipuri de tabele clusterate în Oracle:

- *tabele indexat clusterate* - folosesc o structură index de tip arbore B pentru a localiza blocul de date care conține rândurile cu o anumită valoare a cheii de cluster. Structura unui cluster de tip index este similară cu a unui fişier index de tip arbore B. La nivelul clusterului există doar o singură intrare pentru fiecare valoare distinctă a cheii de clusterare. Dacă există mai multe rânduri care au aceeaşi valoare a cheii de clusterare, ea nu este repetată pentru fiecare rând în parte, ci este păstrată în index o singură dată făcând referire către primul rând cu acea valoare a cheii. Este posibil, astfel, să rezulte o reducere a spațiului de memorare necesar.
- *tabele hash clusterate* – folosesc o funcție hash pentru a calcula adresa la care va fi memorat rândul. Funcția hash utilizează cheia de clusterare şi poate fi definită de utilizator sau poate fi generată de sistem. Atunci când se inserează un rând într-o tabelă din cluster, coloanele cheii de hashing sunt folosite pentru a calcula valoarea de hash şi rândul este memorat în funcție de valoarea de hash calculată. Funcția hash este utilizată şi pentru localizarea rândurilor atunci când acestea trebuie citite din tabelă.

Dimensiunea clusterilor trebuie specificată atunci când se crează tabelele clusterate. La dimensionarea acestora trebuie avute în vedere dimensiunile medii ale rândurilor din tabelele care vor fi clusterate şi numărul de rânduri care vor fi clusterate pe aceeaşi valoare a cheii de clusterare. De exemplu, pentru două tabele Tabel1 şi Tabel2 ce vor fi clusterate, dimensiunea clusterului se obține conform formulei:

*dimensiune_cluster=NR1 * T1 + NR2 * T2*

unde

NR1 este numărul mediu de rânduri din tabela Tabel1 având valoarea K a cheii de clusterare

T1 este dimensiunea medie a unui rând din tabela Tabel1

NR2 este numărul mediu de rânduri din tabela Tabel2 având valoarea K a cheii de clusterare

T2 este dimensiunea medie a unui rând din tabela Tabel2

Pe lângă aceste structuri folosite pentru tabelele de date, Oracle 9*i* mai dispune şi de diverse tipuri de structuri indexate. Aceste structuri asigură un acces direct la rândurile din tabele. Indecşii sunt construiți pe baza unei chei de indexare, care poate fi alcătuită dintr-o singură coloană a tabelei sau prin concatenarea mai multor coloane ale acesteia. Indecşii pot, de asemenea, să fie indecşi cu valori unice, caz în care rândurile tabelei nu pot avea valori duplicat la nivelul coloanei sau a coloanelor ce alcătuiesc cheia de indexare, sau indecşi non-unici, caz în care mai multe rânduri din tabelă pot avea aceeaşi valoare pentru cheia de indexare. Prima situație corespunde unor indecşi denşi, iar cea de-a doua unor indecşi rari [Schwi 00].

Există mai multe tipuri de indecşi în Oracle 9*i*, şi anume:

- ***index clasic***

Indexul clasic este construit folosind o structură de tip arbore B [Honou 98], [Schwi 00], [Oracl 02]. Cheia de indexare poate fi simplă sau compusă, adică alcătuită dintr-una sau mai multe dintre coloanele tabelei. Cheia de indexare poate fi aceeaşi cu cheia primară a tabelei sau poate fi diferită de aceasta. De asemenea, cheia de indexare poate avea valori unice la nivelul rândurilor din tabel sau poate avea valori duplicat. Se pot construi oricât de multe fişiere index asociate unui fişier de date. Acestea sunt stocate într-un segment de date diferit de segmentul de date unde sunt stocate rândurile din tabelă.

- ***index cu cheie inversată***

Spre deosebire de un index clasic, un index cu cheie inversată inversează octeții fiecărei coloane care alcătuiesc cheia de indexare, păstrând ordinea coloanelor în cheie. Indexul este construit tot folosind o structură de arbore B, dar de data aceasta cheia de indexare este cea obținută prin inversarea valorilor cheii de indexare din rândurile tabelei [Schwi 00], [Oracl 02].

O astfel de structură poate fi foarte utilă în evitarea degradării performanțelor în mediile de prelucrare paralele. Dacă, de exemplu, mai mulți utilizatori efectuează în mod concurent operații de inserare masive succesive pe o tabelă, apar timpi de aşteptare considerabili datorită blocărilor instituite la nivelul blocurilor de date din index pe care toți utilizatorii trebuie să le actualizeze. Folosind un index cu cheie inversată se realizează o distribuire a actualizărilor, ce trebuie efectuate în index, de-a lungul mai multor blocuri de date, eliminându-se astfel timpii de aşteptare.

Indexul cu cheie inversată este util doar pentru acele interogări care conțin predicate bazate pe condiții de egalitate. Deoarece cheile de indexare cu valori adiacente nu sunt memorate în index unele lângă altele, căutările bazate pe intervale de valori nu pot fi realizate eficient folosind astfel de structuri.

- ***index bitmap***

Un index bitmap este tot o structură index de tip arbore B, dar care stochează la nivelul nodurilor frunză o hartă de biți pentru fiecare valoare distinctă a cheii de indexare în loc să păstreze o listă de ROWID-uri [Schwi 00], [Oracl 02]. Fiecare bit din harta de biți corespunde unui rând din tabelă. Dacă rândul din tabelă conține valoarea cheii de indexare, bitul din harta de biți ia valoarea „1" logic, iar dacă nu-l conține bitul din harta de biți ia valoarea „0" logic.

Indexul bitmap ocupă mult mai puțin spațiu decât indexul clasic, deci necesită şi mai puține operații de I/O pentru parcurgerea lui.

Indecşii bitmap sunt utili în special când cheia de indexare are cardinalitate scăzută, iar tabela este de dimensiuni mari. Ei mai sunt utili şi în situația în care interogările folosesc adesea în clauza WHERE combinații de condiții implicând operatorul OR, precum şi dacă coloanele care alcătuiesc cheia de indexare au o foarte scăzută rată de actualizare.

Oracle 9*i* suportă atât indecşi bitmap statici cât şi dinamici.

- ***index bazat pe o funcție***

Indexul bazat pe o funcție este tot o structură index de tip arbore B la care, însă, cheia de indexare este obținută prin aplicarea unei funcții asupra uneia sau a mai multor coloane din tabelă [Schwi 00], [Oracl 02]. In cazul unui astfel de index, în primul rând este calculată valoarea cheii de indexare prin aplicarea funcției asupra coloanei sau coloanelor din tabelă care formează cheia de indexare. Valoarea obținută prin calcul este cea care va fi memorată în index. Un index bazat pe o funcție poate fi creat atât ca un index clasic cât şi ca un index bitmap.

- ***index bitmap de tip join***

Un index bitmap de tip join este un index de tip bitmap pentru compunerea a două sau mai multe tabele [Oracl 02]. Structura lui este similară cu cea a unui index bitmap, cu deosebire că în acest caz cheia de indexare şi harta de biți se referă la tabele diferite. Un index bitmap de tip join poate fi utilizat pentru a evita compunerea efectivă a tabelelor, sau pentru a reduce volumul datelor care trebuie supuse operației de join. Interogările ce folosesc indecşi bitmap de tip join pot fi accelerate prin folosirea operațiilor la nivel de bit între hărțile de biți ale indexului.

Indexul bitmap de tip join este util în aplicațiile din domeniul magaziilor de date.

- ***index local pe partiție***

Indexul local pe partiție este o structură indexată construită pentru o tabelă partiționată, care la rândul ei este partiționată utilizând exact acceaşi strategie de partiționare ca şi tabela pentru care a fost construit [Honou 98], [Oracl 02].

Fiecare partiție a unui index de acest tip corespunde cu una şi numai una din partițiile tabelei pentru care a fost creat, deoarece cheia de partiționare este aceeaşi atât pentru index cât şi pentru tabelă.

Indexul local pe partiție este util pentru accelerarea interogărilor pe tabelele partiționate în situația când rândurile vizate de interogări se află în aceeaşi partiție.

Sunt folosite cu precădere în domeniul magaziilor de date.

- ***index global partiționat***

Un index global partiționat este tot o structură indexată creată pentru o tabelă partiționată sau nu, care este la rândul ei partiționată folosind o cheie de partiționare. Cheia de partiționare a indexului şi cheia de partiționare a tabelei trebuie să fie diferite în cazul în care tabela de bază este partiționată [Honou 98], [Oracl 02].

Un astfel de index se numeşte global pentru că conține informație despre întreaga tabelă pentru care a fost construit indexul. Indexul este partiționat pentru a mări gradul de concurență pe care îl suportă.

Indecşii globali partiționați sunt folosiți, cu precădere, în aplicațiile cu magazii de date şi sunt utili la accelerarea interogărilor care vizează toate rândurile tabelei de bază.

- ***index global nepartiționat***

Un index global nepartiționat este similar unui index global partiționat, doar că în acest caz indexul nu mai este partiționat. El este un index care va conține ca şi intrări în index toate valorile distincte ale cheii de indexare din toată tabela de bază. Intregul index este memorat în acelaşi segment de date [Honou 98], [Oracl 02].

Şi acest tip de index este folosit tot în domeniul magaziilor de date.

3.2.2. SQL Server 2000

SQL Server 2000 este un produs al firmei Microsoft [Micro 00]. El este mai puțin performant decât produsul Oracle 9*i*, dar s-a impus pe piață datorită politicii de promovare folosită de compania Microsoft şi pentru că beneficiază de marca numelui Microsoft care este foarte răspândită şi utilizată în domeniul tehnicii de calcul, cu toate că uneori nu este şi unanim apreciată la capitolul calitate.

Şi în SQL Server 2000 sunt implementate mai multe tipuri de structuri de date pentru memorarea datelor din fişierele de date şi din fişierele index asociate acestora, dar oferta este mai redusă decât în Oracle 9*i*.

In SQL Server 2000 datele sunt stocate în fişiere de date care pot fi de unul din următoarele tipuri:

- ***tabele heap***

O *tabelă* este principala structură de date folosită pentru stocarea datelor într-o bază de date SQL Server. Tabelele au o structură de tip heap [Micro 00]. Rândurile de date nu sunt stocate în nici o ordine particulară în cadrul tabelei. Blocurile de date ale unei tabele au mărimea de 8Ko. Fiecare bloc de date are un header de 96 de octeți conținând informație necesară sistemului, precum identificatorul tabelului care deține acea pagină, pointeri de legătură către blocul de date anterior şi blocul următor din fişier, etc. Restul octeților din bloc sunt ocupați de rândurile de date, care nu sunt memorate într-o ordine particulară în cadrul tabelei.

Pe lângă tipurile de date clasice, cele numerice şi alfanumerice, există şi tipuri de date noi care încearcă să satisfacă cerințele noilor tipuri de aplicații. Din categoria acestora din urmă fac parte tipurile de date TEXT, NTEXT, IMAGE.

- ***tabele membru***

SQL Server 2000 nu permite partiționarea tabelelor aşa cum se întâmplă în Oracle 9*i*. SQL Server 2000 permite construirea unei arhitecturi pentru baza de date de tip federație de baze de date, adică un grup de servere de baze de date administrate independent care cooperează în scopul de a partaja încărcarea de procesare de la nivelul sistemului [Micro 00] [Lejeu 00-2]. In cadrul federației, datele sunt divizate între diferitele servere din sistem. Când un utilizator se conectează la federația de baze de date, el este conectat de fapt la unul din serverele acesteia. Dacă utilizatorul solicită date care se găsesc la nivelul unui alt server din federație decât cel la care este conectat, regăsirea datelor va consuma semnificativ mai mult timp decât regăsirea datelor de pe serverul local.

Tabelele din serverele unei federații de baze de date se numesc *tabele membru*, dar ele sunt administrate independent de celelalte tabele existente la nivelul celorlalte servere din federație. Faptul că serverele federației de baze de date nu dețin un dicționar de date comun, conduce la degradări ale performanței şi scalabilității la nivelul aplicațiilor.

Tabelele membru ale unei federații pot fi ulilizate pentru a partiționa datele unei tabele. De exemplu, tabela MARFURI poate fi partiționată în mai multe tabele membru, fiecare memorată pe un alt server al federației de baze de date, în funcție de valoarea coloanei GRUPA. Dar, în acest caz sistemul nu este capabil să decidă singur care sunt tabelele membru care nu vor fi accesate ca urmare a unei interogări, pentru că nu pot conține datele cerute.

Interogarea este transmisă către toate serverele din federație, şi în urma interogării tabelelor membru ale acestora este obținut rezultatul final.

Pentru a îmbunătăți într-o oarecare măsură situația partiționării datelor este posibilă crearea unor *vizualizări partiționate*. O vizualizare partiționată (partitioned view) reuneşte pe orizontală partiții de date obținute din tabelele membre existente în unul sau mai multe din serverele federației. Aceste date apar ca şi când ar face parte dintr-o singură tabelă a bazei de date. Datele sunt partiționate între tabelele membre pe baza unor intervale de valori existente într-una din coloanele tabelelor, numită *coloană de partiționare*. Intervalele de valori pentru fiecare tabelă membru sunt definite cu ajutorul unei restricții de tip CHECK specificată asupra coloanei de partiționare. Vizualizarea partiționată este creată cu ajutorul clauzei UNION ALL pentru a combina toate selecțiile din toate tabelele membru în rezultatul final.

SQL Server 2000 admite două tipuri de vizualizări partiționate. Primul îl constituie *vizualizările partiționate locale*, la care toate tabelele participante la realizarea vizualizării, precum şi vizualizarea însăşi, există la nivelul aceleaşi instanțe SQL Server. Cel de-al doilea tip îl constituie *vizualizările partiționate distribuite*, la care cel puțin una din tabelele participante la realizarea vizualizării există la nivelul unui server diferit de serverul celorlalte.

- ***tabele organizate indexat***

Şi SQL Server 2000 permite stocarea datelor în tabele organizate indexat. Aceste tipuri de tabele se numesc însă, în SQL Server, *index clusterat*. Intr-o astfel de structură, rândurile de date sunt memorate într-o ordine logică dată de cheia de indexare. La nivelul nodurilor frunză din arborele B, construit pe baza valorilor cheii de indexare din rândurile tabelei, se memorează întregul rând de date al tabelei [Micro 00], [Micro 02-2].

Restricția de cheie primară crează automat un index clusterat dacă nu se specifică explicit crearea unui index non-clusterat.

Acest tip de structură pentru tabelele de date permite un acces rapid la acestea pentru interogările care implică potrivirea exactă şi/sau căutarea pe interval de valori asupra cheii primare, deoarece rândurile sunt memorate în nodurile frunză în ordinea valorilor cheii primare.

Firma Microsoft recomandă folosirea unui index clusterat bazat pe o cheie de indexare ale cărei valori sunt monoton crescătoare [Micro 02-2], [Micro 03-2]. Este indicat ca, cheia de indexare să fie alcătuită dintr-o singură coloană a tabelei şi această coloană să nu aibă o

lungime prea mare. Pentru mărirea performanțelor este indicat ca tabelele să fie organizate folosind o astfel de structură.

Asociat acestor tipuri de structuri, folosite pentru memorarea datelor, există şi structuri indexate, care au rolul de a asigura un acces direct la rândurile tabelelor. Indecşii sunt construiți pe baza unei chei de indexare, care poate fi alcătuită dintr-o singură coloană a tabelei sau prin concatenarea mai multor coloane ale tabelei. Indecşii pot, de asemenea, să fie indecşi cu valori unice, caz în care rândurile tabelei nu pot avea valori duplicat la nivelul coloanei sau a coloanelor ce alcătuiesc cheia de indexare, sau indecşi non-unici, caz în care mai multe rânduri din tabelă pot avea aceeaşi valoare pentru cheia de indexare. Ordonarea valorilor coloanelor care alcătuiesc cheia de indexare se poate face ascendent sau descendent [Micro 00], [Micro 02-1], [Micro 03-4].

În SQL Server 2000 există implementate următoarele tipuri de indecşi:

- ***index clusterat***

Acest tip de index a fost prezentat atunci când s-a tratat problema tabelelor organizate indexat. De fapt, un index clusterat este unul şi acelaşi lucru cu o tabelă organizată indexat. O tabelă poate avea un singur index clusterat [Micro 02-2], [Micro 03-2].

- ***index non-clusterat***

Pe lângă indexul clusterat, care a fost prezentat deja, fiind echivalentul unei tabele organizate indexat din Oracle, mai există indecşii clasici a căror stuctură este cea tipică de arbore B. Acest tip de index se numeşte *index non-clusterat* [Micro 03-3]. Dacă tabela pentru care este creat indexul non-clusterat nu este organizată indexat, adică nu are un index clusterat, la nivelul nodurilor frunză din stuctura de arbore B a indexului non-clusterat se păstrează adresa relativă a rândului din tabela de date, care are acea valoare a cheii de indexare. Dacă tabela pentru care este creat indexul non-clusterat este organizată indexat, adică are un index clusterat, la nivelul nodurilor frunză din stuctura de arbore B a indexului non-clusterat se păstrează cheia de indexare din indexul clusterat corespunzătoare rândului care are acea valoare a cheii de indexare a indexului non-clusterat.

Indexul non-clusterat este indicat atunci când cheia de indexare are selectivitate mare, când numărul rândurilor obținute în urma selecțiilor nu este prea mare, când coloana sau coloanele care alcătuiesc cheia de indexare au lărgime mare, când sunt efectuate operații de

compunere între tabele, când cheia de indexare corespunde cu condițiile de căutare din clauza WHERE sau cele de ordonare din clauza ORDER BY a blocurilor de interogare. Se pot crea până la 249 de indecşi non-clusterați pentru o tabelă [Micro 03-4], [Micro 03-6].

SQL Server 2000 nu permite construirea de indecşi bitmap sau de indecşi bitmap de tip join.

- ***index partiționat***

De asemenea, în SQL Server 2000 nu este posibil să se construiască indecşi globali partiționați. Se pot construi doar indecşi locali la nivelul tabelelor membru [Micro 00]. Totuşi, pentru vizualizările partiționate locale se poate construi un index pe coloana de partiționare. Acest index îmbunătățeşte performanța în situația în care condiția de căutare corespunde cu coloana de partiționare. Indexul asupra vizualizării partiționate poate fi de tip clusterat sau non-clusterat.

- ***index bazat pe o funcție***

Indexul bazat pe o funcție este tot o structură index de tip arbore B la care, însă, cheia de indexare este obținută prin aplicarea unei funcții asupra uneia sau a mai multor coloane din tabelă. In cazul unui astfel de index, în primul rând este calculată valoarea cheii de indexare prin aplicarea funcției asupra coloanei sau coloanelor din tabelă care formează cheia de indexare. Valoarea obținută prin calcul este cea care va fi utilizată la construcția indexului şi este cea care va fi memorată în index. Spre deosebire de varianta din Oracle la care valoarea calculată este doar folosită la construcția indexului, în SQL Server 2000 ea este memorată într-o nouă coloană a tabelei de bază, numită *coloană cu valori calculate* [Lejeu 02-2].

3.2.3. DB2

DB2 este produsul, din categoria baze de date, al firmei IBM [IBM 02]. Sub anumite aspecte acest produs este mai performant decât SQL Server 2000, dar nu depăşeşte performanțele lui Oracle 9*i*.

Şi DB2, ca şi celelalte două produse analizate, dispune de mai multe tipuri de structuri de date folosite pentru memorarea datelor bazei de date. Stocarea relațiilor în baza de date se poate face sub forma unor :

- ***tabele heap***

Tabelele clasice, cele construite pe baza unei structuri de tip heap, sunt prezente şi în DB2 [IBM 02], [Bloor 03-2]. Ele reprezintă cea mai des utilizată structură pentru memorarea datelor. Rândurile de date nu au nici o ordine logică prestabilită atunci când sunt stocate în fişierele de date. Ele sunt memorate în blocuri de date de mărime fixă, declarată la crearea tabelei. Fiecare bloc de date are un header, care conține informații necesare sistemului de gestiune a bazei de date, şi o zonă de date, care stochează rândurile de date. Blocurile de date pot fi ocupate cu informație utilă numai într-un anumit procent, stabilit la crearea tabelei prin intermediul parametrului numit factor de umplere.

Definirea datelor se poate face folosind tipuri clasice de date, precum cele numerice şi alfanumerice. Alături de acestea există şi tipuri noi de date, precum text, audio, video, imagine, date spațiale, introduse pentru a satisface cerințele noilor tipuri de aplicații. Acestea se folosesc prin intermediul unor componente distincte de la nivelul sistemului, numite DB2 Extenders [Bloor 03-2].

- ***tabele partiționate***

Partiționarea tabelelor permite unor structuri mari şi complexe de baze de date să fie descompuse în bucăți mai mici şi mai uşor de gestionat. Şi în DB2 este posibilă partiționarea tabelelor, dar oferta de partiționare este mult mai redusă decât în Oracle 9*i*.

DB2 suportă doar partiționarea de tip hash [Lejeu 02-1], [IBM 02]. Prin urmare anumite tipuri de interogări care folosesc cheia de partiționare, dar care presupun apartenența acesteia la intervale sau liste de valori, nu pot folosi avantajul partiționării datelor în vederea regăsirii mai rapide a datelor, doar prin consultatea unora dintre partiții. De asemenea, partiționarea de tip hash nu poate asigura avantaje în situația aplicațiilor care necesită înlocuirea periodică a unui set de date vechi cu un set de date noi. In această situație datele trebuie culese şi distribuite în toate partițiile tabelei, ceea ce nu conduce la îmbunătățirea timpilor de execuției [Bloor 03-1], [Bloor 02].

Şi în DB2 se pot defini mai multe tipuri de structuri indexate. Asociat stucturilor folosite pentru memorarea datelor, există structurile indexate, care au rolul de a asigura un acces direct la rândurile tabelelor. Indecşii sunt construiți pe baza unei chei de indexare, care poate fi alcătuită dintr-o singură coloană a tabelei sau prin concatenarea mai multor coloane ale tabelei. Indecşii pot, de asemenea, să fie indecşi cu valori unice, caz în care rândurile

tabelei nu pot avea valori duplicat la nivelul coloanei sau a coloanelor ce alcătuiesc cheia de indexare, sau indecşi non-unici, caz în care mai multe rânduri din tabelă pot avea aceeaşi valoare pentru cheia de indexare [IBM 02], [Bloor 03-2].

DB2 admite următoarele tipuri de indecşi:

- ***index clasic***

Indexul clasic este cel construit pe o structură clasică de arbore B [IBM 02]. Câmpul după care se ordonează la nivelul indexului rândurile tabelei de date se numeşte *cheie de indexare*. Cheia de indexare poate fi singulară, adică alcătuită dintr-o singură coloană a tabelei de date, sau compusă, adică alcătuită din două sau mai multe coloane ale tabelei de date. Ordonarea valorilor la nivelul cheii de indexare poate fi ascendentă sau descendentă. De asemenea, indexul poate fi cu valori unice sau non-unice.

In DB2 nu se pot crea indecşi cu cheie de indexare inversată.

- ***index bitmap dinamic***

Indexul bitmap dinamic este obținut prin convertirea într-o hartă de biți a structurii unui index clasic construit folosind arbori B [Lejeu 02-1], [IBM 02]. Indexul de tip arbore B există în baza de date. Convertirea structurii acestuia într-o hartă de biți se produce numai în memoria centrală pe durata execuției interogărilor. Crearea indecşilor bitmap dinamici se produce în momentul execuției interogărilor prin luarea ROWID-urilor din cadrul indecşilor clasici şi transformarea lor într-o hartă de biți. Hărțile de biți sunt utilizate pentru a mări viteza de execuție a unor interogări ce pot folosi operațiile booleane asupra hărților de biți sau combinarea hărților corespunzătoare mai multor intrări în indecşi bitmap asociați condițiilor din clauza WHERE a interogărilor.

Indecşii bitmap dinamici nu sunt totuşi la fel de performanți ca cei statici. Ei sunt utilizați cu precădere în domeniul depozitelor de date [Bloor 02].

DB2 nu suportă indexul bitmap static şi nici indexul bitmap de tip join.

- ***index bazat pe o funcție***

Indexul bazat pe o funcție este tot o structură index de tip arbore B la care, însă, cheia de indexare este obținută prin aplicarea unei funcții asupra uneia sau a mai multor coloane din

tabelă. In cazul unui astfel de index, în primul rând este calculată valoarea cheii de indexare prin aplicarea funcției asupra coloanei sau coloanelor din tabelă care formează cheia de indexare. Valoarea obținută prin calcul este cea care va fi utilizată la construcția indexului şi este cea care va fi memorată în index. Spre deosebire de varianta din Oracle la care valoarea calculată este doar folosită la construcția indexului, în DB2 ea este memorată într-o nouă coloană a tabelei de bază, numită *coloană cu valori generate*. Practic, în DB2 indexul bazat pe o funcție este, de fapt, un index obişnuit care se crează având ca şi cheie de indexare o coloană cu valori generate [Lejeu 02-1], [IBM 02].

- ***index local pe partiție***

Indexul local pe partiție este o structură indexată construită pentru o tabelă partiționată, care la rândul ei este partiționată utilizând exact aceeaşi strategie de partiționare ca şi tabela pentru care a fost construit [IBM 02], [Bloor 03-2].

Fiecare partiție a unui index de acest tip corespunde cu una şi numai una din partițiile tabelei pentru care a fost creat, deoarece cheia de partiționare este aceeaşi atât pentru index cât şi pentru tabelă.

Indexul local pe partiție este util pentru accelerarea interogărilor pe tabelele partiționate în situația când rândurile vizate de interogări se află în aceeaşi partiție.

Sunt folosite cu precădere în domeniul magaziilor de date.

3.2.4. Concluzii

Dintre cele trei sisteme de gestiune a bazelor de date analizate cel mai performant în ceea ce priveşte facilitățile oferite pentru stocarea şi accesarea datelor s-a dovedit a fi Oracle 9*i*, urmat de DB2 şi SQL Server 2000.

Paleta structurilor de date ce pot fi folosite la nivelul sistemului este mult mai largă în Oracle 9*i* comparativ cu celelalte două sisteme. Prin urmare acest produs poate face față cu succes unor cerințe mult mai variate apărute la nivelul aplicațiilor dezvoltate. Dintre celelalte două produse nu se poate desprinde cu uşurință unul în învingător, pentru că balanța se înclină când în favoarea unuia când a celuilalt. DB2 permite o mai bună partiționare a datelor şi crearea de indecşi bitmap dinamici utili în domeniul magaziilor de date, în timp ce SQL Server 2000 permite crearea de tabele organizate indexat, care limitează spațiul de memorare

ocupat de fişierul de date şi de indexul primar asociat acestuia şi micşorează timpii de răspuns la anumite categorii de interogări.

Tabelul 1 prezintă sinteza rezultatelor obținute în urma analizei făcute.

Tabel 1. Structuri de date implementate în SGBD-urile analizate.

Caracteristică	*Oracle 9i*	*DB2*	*SQL Server 2000*
Tabele heap	Da	Da	Da
Tabele organizate indexat	Da	Nu	Da
Tabele partiționate:			
- hash	Da	Da	Nu
- range	Da	Nu	Nu
- list	Da	Nu	Nu
- range-hash	Da	Nu	Nu
- range-list	Da	Nu	Nu
Tabele clusterate	Da	Nu	Nu
Index clasic	Da	Da	Da
Index bitmap static	Da	Nu	Nu
Index bitmap dinamic	Da	Da	Nu
Index bitmap de tip join	Da	Nu	Nu
Index pe cheie inversată	Da	Nu	Nu
Index bazat pe o funcție	Da	Parțial	Parțial
Index local pe partiție	Da	Da	Nu
Index global partiționat	Da	Nu	Nu
Index global non-partiționat	Da	Nu	Nu

După cum se poate observa, structura de tip heap este cea mai comună formă de organizare a fişierelor, ea fiind prezentă la nivelul fiecăruia dintre sistemele analizate. Structura de tip tabelă organizată indexat este comună numai sistemului de gestiune a bazelor de date Oracle 9*i* şi sistemului de gestiune a bazelor de date SQL Server 2000. Sistemul de gestiune a bazelor de date DB2 nu dispune de acest tip de structură, ceea ce constituie un minus pentru el. In schimb DB2 dispune de posibilitatea de a partiționa tabelele, chiar dacă această partiționare poate fi doar de tip hash. Şi în cazul tabelelor partiționate Oracle 9*i* se situează pe primul loc, deoarece oferta de partiționare a tabelelor este mult mai variată în acest caz. Sunt permise partiții de tip hash, range, list, range-hash şi range-list. Sistemul de gestiune a bazelor de date SQL Server 2000 nu dispune de posibilitatea de a defini partiții la nivelul tabelelor. De asemenea, dintre cele trei sisteme, Oracle 9*i* este singurul care permite construirea de tabele clusterate, ce permit stocarea înregistrărilor înrudite, din mai multe

tabele ale bazei de date, în acelaşi cluster de pe disc, micşorând astfel timpii de răspuns la interogările care vizează aceste înregistrări.

In ceea ce priveşte structurile indexate, toate cele trei sisteme permit construirea de indecşi clasici bazați pe arbori B. Indecşii de tip bitmap sunt suportați numai de Oracle 9*i* şi de DB2, cu toate că şi în această privință Oracle 9*i* este mai performant, pentru că suportă atât indecşi bitmap statici cât şi dinamici, pe când DB2 suportă doar indecşii bitmap dinamici. Indexul pe cheie inversată este suportat doar de Oracle 9*i*. Indexul bazat pe o funcție este suportat de toate cele trei sisteme, cu toate că implementările de la nivelul lui DB2 şi SQL Server 2000 sunt mai puțin evoluate decât cea din Oracle. Având în vedere că facilitățile de partiționare a tabelelor de date sunt prezente doar în Oracle 9*i* şi în DB2, aceste sisteme sunt şi singurele care oferă indecşi locali la nivel de partiție. Oracle 9*i* mai suportă pe lângă acest tip de index şi indecşii globali la nivelul tuturor partițiilor tabelei, care pot fi la rândul lor partiționați sau nu.

Urmărind informația din tabelul 1, se poate observa că Oracle 9*i* se desprinde în lider, oferind cele mai multe facilități, fiind urmat la oarecare distanță de celelalte două produse care sunt aproximativ echivalente ca performanțe.

3.3. Studiul influenței structurii fizice asupra performanței bazei de date

Baza de date test, cea prezentată în anexa 1, a fost supusă mai multor analize care au avut ca scop identificarea unor pattern-uri în comportamentul de performanță al acesteia pentru diferite situații de stimulare operațională. Au fost testate diferite structuri de organizare a fişierelor de date, precum şi comportamentul acestora în prezența unor structuri auxiliare de acces construite după condiții diverse şi complexe. S-a urmărit identificarea caracteristicilor şi valorilor ce țin de structura fizică a bazei de date şi care influențează performanța în exploatare a acesteia, rezultate care au fost folosite ulterior în cadrul capitolului 4 pentru dezvoltarea metodei de proiectare fizică a bazei de date. Testele au fost împărțite în mai multe categorii în funcție de comportamentul pe care l-au urmărit la nivelul bazei de date şi de stimulii la care aceasta a fost supusă. Rezultatele testelor efectuate au fost colectate tot în baza de date test, în tabela REZULTATE. Structura acestei tabele are următoarea semnificație:

- *NrOp* – identificator al tipului de operație efectuată asupra tabelelor bazei de date test
- *Operație* – indică operația efectuată asupra bazei de date test
- *TotalInreg* – numărul total de tupluri din tabela prelucrată de operația executată

- *InregPrel* –numărul de tupluri din tabelă afectat de operația executată
- *Timp* – timpul de execuție al operației exprimat în milisecunde
- *Valoare* – indică valorile folosite în clauza WHERE de către operația executată
- *Index* – indică dacă fişierul de date are sau nu index asociat (D/N)
- *TipIndex* – indică tipul de fişier index folosit (clasic/bitmap static/bitmap dinamic/etc.)
- *NrIndex* – indică numărul de fişiere index asociate fişierului de date

Fiecare operație SELECT, INSERT, UPDATE, DELETE a fost executată în mai multe variante şi în mod repetat asupra tabelelor bazei de date test, iar rezultatele obținute la fiecare execuție au fost colectate în tabela REZULTATE. Mediile valorilor timpilor de execuție obținuți pentru fiecare tip de operație executată în parte au fost colectate în tabela REZULTATE_FINALE, care are o structură similară cu REZULTATE, numai că în acest caz coloana *Timp* reprezintă media timpului de execuție al operației exprimat în milisecunde. Datele din tabela REZULTATE_FINALE au fost folosite pentru a trasa graficele prezentate în lucrare şi au fost analizate pentru a se putea formula concluziile.

Baza de date test a fost implementată în trei variante, folosindu-se cele trei sisteme de gestiune a bazelor de date analizate anterior. Prima variantă a fost o implementare în Oracle 9*i*, caz în care testele au fost rulate prin intermediul unor scripturi scrise în PL/SQL. A doua variantă a fost o implementare în SQL Server2000, caz în care testele au fost executate folosindu-se proceduri stocate scrise folosind Transact-SQL. Cea de a treia variantă a fost o implementare în DB2, dar în acest caz testele au constat în rularea directă a blocurilor SQL şi în analizarea timpilor de execuție obținuți comparativ cu timpii de execuție obținuți în celelalte două implementări.

S-au efectuat mai multe grupe de teste, fiecare dintre ele urmărind analizarea anumitor aspecte legate de influența structurii fizice asupra performanței bazei de date. Aceleaşi grupe de teste au fost rulate în toate cele trei implementări.

3.3.1. Influența indexului asupra timpului de răspuns la interogări. Concluzii

O primă categorie de teste a fost făcută pe tabele având o structură de tip heap, care au asociate structuri indexate de tip arbore B. Testele şi-au propus să studieze influența indexului asupra timpilor de răspuns la interogări formulate asupra tabelei de date. Condiția de căutare

din interogare este cea care stă la baza construirii indexului. Testele efectuate şi-au propus să analizeze:

Care este mărimea fişierului de date de la care prezența indexului îmbunătățește timpii de răspuns?

Cum depinde această îmbunătățire de tipul interogării formulate?

- **Prezentarea testelor şi a rezultatelor obținute. Concluzii.**

Pentru efectuarea testelor s-a folosit o tabelă organizată după o structură de tip heap. Ea are asociată un index clasic, construit pe o structură de tip arbore B. Indexul este construit pe baza unei chei de index care apare în condiția de căutare din clauza WHERE a interogării ce se execută. Pentru testare s-a folosit tabela FURNIZORI. Numărul de rânduri din această tabelă a variat între 424 şi 6.360, adică o mărime a fişierului de date cuprinsă între 100-800 Ko. Asupra acestuia au fost rulate următoarele tipuri de interogări SQL [Kochh 00]:

- ***Primul tip de interogări***

```
SELECT *
FROM FURNIZORI
WHERE COD_FURN='FLARO';
```

Acest tip de interogare este bazat pe o condiție ce implică codul furnizorului, care este unic la nivelul relației, deci este returnat un singur tuplu ca rezultat al interogării. Pentru acest tip de interogare s-a creat un fişier index având ca şi cheie de indexare codul furnizorului, şi anume

```
CREATE INDEX I_FURN_COD ON FURNIZORI(COD_FURN);
```

Pentru testele efectuate pe Oracle *9i* a fost rulat următorul script PL/SQL:

```
--FurnizoriCodFurn.sql
DECLARE
  v_NrOp NUMBER(3);
  v_Operatie VARCHAR(200);
```

```
 v_Start NUMBER(10,0);
 v_End NUMBER(10,0);
 v_Timp NUMBER(10,0);
 v_NrInreg NUMBER(12,0);
 v_Index CHAR(1);
 v_TipIndex VARCHAR(20);
 v_NrIndex NUMBER(2,0);
 v_Furnizori Furnizori%ROWTYPE;
 v_Cod_Furn Furnizori.Cod_Furn%TYPE;
 v_TotalInreg NUMBER(15,0);
 CURSOR c_Furnizori IS
  SELECT *
  FROM Furnizori
  WHERE Cod_Furn=v_Cod_Furn;
 CURSOR c_TotalInreg IS
  SELECT count(*)
  FROM Furnizori;
BEGIN
 v_NrOp:=&NrOp;
 OPEN c_TotalInreg;
 FETCH c_TotalInreg INTO v_TotalInreg;
 CLOSE c_TotalInreg;
 v_Cod_Furn:='&Cod_Furn';
 v_Index:= '&Index';
 v_TipIndex:= '&TipIndex';
 v_NrIndex:=&NrIndex;
 v_Operatie:='SELECT * FROM Furnizori WHERE Cod_Furn=';
 v_Start:=DBMS_UTILITY.GET_TIME;
 OPEN c_Furnizori;
 v_End:=DBMS_UTILITY.GET_TIME;
 v_Timp:=(v_End-v_Start)*10;
 LOOP
   FETCH c_Furnizori INTO v_Furnizori;
   EXIT WHEN c_Furnizori%NOTFOUND;
 END LOOP;
 v_NrInreg:=c_Furnizori%ROWCOUNT;
 CLOSE c_Furnizori;
 INSERT INTO Rezultate(NrOp, Operatie, TotalInreg, InregPrel,Timp, Valoare, Index, TipIndex,
        NrIndex) VALUES (v_NrOp, v_Operatie, v_TotalInreg, v_NrInreg, v_Timp, v_Cod_Furn,
        v_Index, v_TipIndex, v_NrIndex);
```

```
 COMMIT;
END;
```

La rularea scriptului se furnizează ca parametrii numărul operației, valoarea pentru COD_FURN care va fi folosită în clauza WHERE a blocului SELECT, dacă există sau nu index asociat fişierului de date, tipul indexului, precum şi numărul de fişiere index asociate fişierului de date. Timpul de execuție al operației SELECT, în cazul particular specificat, este inserat în tabela REZULTATE.

Acest tip de interogare a fost executat în mod repetat folosindu-se diferite valori pentru COD_FURN. Unele dintre valorile colectate în tabela REZULTATE pot fi observate în anexa 1.

Acelaşi tip de test a fost făcut şi pentru SQL Server2000. In acest caz s-a folosit procedura stocată FurnizoriCodFurn de mai jos:

```
/*procedura stocată FurnizoriCodFurn*/
CREATE PROCEDURE dbo.FurnizoriCodFurn
(       @nrOp NUMERIC(18,0), @vCodFurn CHAR(7), @index CHAR(1), @tipIndex VARCHAR(20), @nrIndex NUMERIC(2,0)
)
AS
SET NOCOUNT ON
BEGIN TRANSACTION
        DECLARE @op VARCHAR(200),@t1 NUMERIC(18,0),@t2 NUMERIC(18,0),@s1 INT,@s2 INT,@total NUMERIC(18,0),@inreg NUMERIC(18,0),@dif NUMERIC(18,6)
        SET @op = 'SELECT * FROM Furnizori WHERE Cod_Furn='
        SELECT @total=count(*) FROM Furnizori
        SELECT @inreg = count(*)  FROM Furnizori WHERE Cod_Furn=@vCodFurn
        SELECT @t1=datepart(ms,getdate()),@s1=datepart(s,getdate())
        SELECT *  FROM Furnizori WHERE Cod_Furn=@vCodFurn
        SELECT @t2=datepart(ms,getdate()),@s2=datepart(s,getdate())
        if @s2>@s1
                SET @dif=(@s2*1000+@t2)-(@s1*1000+@t1)
        else
                SET @dif=@t2-@t1
        INSERT Rezultate VALUES
                (@nr,@op,@total,@inreg,@dif,@vCodFurn,@index,@tipIndex,@nrIndex)
if @@error<>0
BEGIN
```

```
        ROLLBACK TRANSACTION
        RAISERROR ('Operatiunea nu s-a putut realiza din cauza unei erori!!',16,1) with NOWAIT
END
ELSE
BEGIN
        COMMIT TRANSACTION
        RETURN(0)
END
GO
```

La executarea procedurii se furnizează ca parametrii numărul operației, valoarea pentru COD_FURN care va fi folosită în clauza WHERE a blocului SELECT, dacă există sau nu index asociat fișierului de date, tipul indexului, precum și numărul de fișiere index asociate fișierului de date. Timpul de execuție al operației SELECT, pentru fiecare caz particular specificat, este inserat în tabela REZULTATE.

Și în acest caz, acest tip de interogare a fost executat în mod repetat folosindu-se diferite valori pentru COD_FURN. În anexa 1 pot fi observate câteva dintre rezultatele colectate în tabela REZULTATE.

- ***al doilea tip de interogări***

```
SELECT *
FROM FURNIZORI
WHERE LOC_FURN='SIBIU';
```

In acest caz testele efectuate pe Oracle 9*i* au folosit scriptul FurnizoriLocFurn.sql de mai jos:

```
--FurnizoriLocFurn.sql
DECLARE
 v_NrOp NUMBER(3);
 v_Operatie VARCHAR(200);
 v_Start NUMBER(10,0);
 v_End NUMBER(10,0);
 v_Timp NUMBER(10,0);
 v_NrInreg NUMBER(12,0);
 v_Index CHAR(1);
 v_TipIndex VARCHAR(20);
 v_NrIndex NUMBER(2,0);
```

```
v_Furnizori Furnizori%ROWTYPE;
v_Loc_Furn Furnizori.Loc_Furn%TYPE;
v_TotalInreg NUMBER(15,0);
CURSOR c_Furnizori IS
  SELECT *
  FROM Furnizori
  WHERE Loc_Furn=v_Loc_Furn;
CURSOR c_TotalInreg IS
  SELECT count(*)
  FROM Furnizori;
BEGIN
v_NrOp:=&NrOp;
OPEN c_TotalInreg;
FETCH c_TotalInreg INTO v_TotalInreg;
CLOSE c_TotalInreg;
v_Loc_Furn:='&Loc_Furn';
v_Index:= '&Index';
v_TipIndex:= '&TipIndex';
v_NrIndex:=&NrIndex;
v_Operatie:='SELECT * FROM Furnizori WHERE Loc_Furn=';
v_Start:=DBMS_UTILITY.GET_TIME;
OPEN c_Furnizori;
v_End:=DBMS_UTILITY.GET_TIME;
v_Timp:=(v_End-v_Start)*10;
LOOP
  FETCH c_Furnizori INTO v_Furnizori;
  EXIT WHEN c_Furnizori%NOTFOUND;
END LOOP;
v_NrInreg:=c_Furnizori%ROWCOUNT;
CLOSE c_Furnizori;
INSERT INTO Rezultate(NrOp, Operatie, TotalInreg, InregPrel,Timp, Valoare, Index, TipIndex,
      NrIndex) VALUES (v_NrOp, v_Operatie, v_TotalInreg, v_NrInreg, v_Timp, v_Loc_Furn,
      v_Index, v_TipIndex, v_NrIndex);
COMMIT;
END;
```

La rularea scriptului se furnizează ca parametrii numărul operației, valoarea pentru LOC_FURN care va fi folosită în clauza WHERE a blocului SELECT, dacă există sau nu index asociat fişierului de date, tipul indexului, precum şi numărul de fişiere index asociate

fișierului de date. Timpul de execuție al operației SELECT, în cazul particular specificat, este inserat în tabela REZULTATE.

Acest tip de interogare a fost executat în mod repetat folosindu-se diferite valori pentru LOC_FURN. Câteva dintre rezultatele colectate în tabela REZULTATE pot fi observate în anexa 1.

Pentru SQL Server2000 acest tip de test a folosit procedura stocată FurnizoriLocFurn de mai jos:

```
/*procedura stocată FurnizoriLocFurn*/
CREATE PROCEDURE dbo.FurnizoriLocFurn
(       @nrOp NUMERIC(18,0), @vLocFurn VARCHAR(20), @index CHAR(1), @tipIndex VARCHAR(20), @nrIndex NUMERIC(2,0)
)
AS
SET NOCOUNT ON
BEGIN TRANSACTION
        DECLARE @op VARCHAR(200),@t1 NUMERIC(18,0),@t2 NUMERIC(18,0),@s1 INT,@s2 INT,@total NUMERIC(18,0),@inreg NUMERIC(18,0),@dif NUMERIC(18,6)
        SET @op = 'SELECT * FROM Furnizori WHERE Loc_Furn='
        SELECT @total=count(*) FROM Furnizori
        SELECT @inreg = count(*) FROM Furnizori WHERE Loc_Furn=@vLocFurn
        SELECT @t1=datepart(ms,getdate()),@s1=datepart(s,getdate())
        SELECT * FROM Furnizori WHERE Loc_Furn=@vLocFurn
        SELECT @t2=datepart(ms,getdate()),@s2=datepart(s,getdate())
        if @s2>@s1
                SET @dif=(@s2*1000+@t2)-(@s1*1000+@t1)
        else
                SET @dif=@t2-@t1
        INSERT Rezultate VALUES
                (@nr,@op,@total,@inreg,@dif,@vLocFurn,@index,@tipIndex,@nrIndex)
if @@error<>0
BEGIN
        ROLLBACK TRANSACTION
        RAISERROR ('Operatiunea nu s-a putut realiza din cauza unei erori!!',16,1) with NOWAIT
END
ELSE
BEGIN
        COMMIT TRANSACTION
        RETURN(0)
```

```
END
GO
```

La executarea procedurii se furnizează ca parametrii numărul operației, valoarea pentru LOC_FURN care va fi folosită în clauza WHERE a blocului SELECT, dacă există sau nu index asociat fişierului de date, tipul indexului, precum şi numărul de fişiere index asociate fişierului de date. Timpul de execuție al operației SELECT, pentru fiecare caz particular specificat, este inserat în tabela REZULTATE.

Şi în acest caz, acest tip de interogare a fost executat în mod repetat folosindu-se diferite valori pentru LOC_FURN. O parte a rezultatelor colectate în tabela REZULTATE pot fi consultate în anexa 1.

Acest tip de interogare este bazat pe o condiție ce implică localitatea furnizorului care nu este unică la nivelul relației, deci vor fi returnate mai multe tupluri ca rezultat al interogării. Numărul de tupluri returnate ca răspuns al interogării depinde de valoarea precizată în interorage pentru LOC_FURN şi pentru majoritatea testelor efectuate s-a situat în jurul valorii de 3% din numărul total de tupluri al relației. Numărul de tupluri returnate ca răspuns al interogării este şi el colectat în tabela REZULTATE. Spre exemplu, pentru fişier FURNIZORI având 4.240 înregistrări, interogarea pentru LOC_FURN='SIBIU' returnează 120 de înregistrări. Numărul de înregistrări din fişierul de date a variat între 424 şi 6.360, ceea ce a condus la o mărime a fişierului cuprinsă între 100-800 Ko. Pentru acest tip de interogare s-a creat un fişier index având ca şi cheie de indexare localitatea furnizorului, şi anume:

```
CREATE INDEX I_FURN_LOC ON FURNIZORI(LOC_FURN);
```

Tot pentru acest tip de testare s-a folosit şi tabela MARFURI. Numărul de rânduri din această tabelă a variat într-o primă etapă între 1.263 şi 214.710, adică o mărime a fişierului de date cuprinsă între 0,1-16 Mo, şi în a doua etapă între 265.230 şi 4.234.680 rânduri, adică o mărime a fişierului cuprinsă între 20-310 Mo. Asupra acestei tabele a fost rulat următorul tip de interogare:

```
SELECT *
FROM MARFURI
WHERE DENP='ZAHAR';
```

Testele efectuate pe Oracle 9*i* au folosit scriptul MarfuriDenP.sql de mai jos:

```
--MarfuriDenP.sql
DECLARE
 v_NrOp NUMBER(3);
 v_Operatie VARCHAR(200);
 v_Start NUMBER(10,0);
 v_End NUMBER(10,0);
 v_Timp NUMBER(10,0);
 v_NrInreg NUMBER(12,0);
 v_Index CHAR(1);
 v_TipIndex VARCHAR(20);
 v_NrIndex NUMBER(2,0);
 v_Marfuri Marfuri%ROWTYPE;
 v_DenP Marfuri.DenP%TYPE;
 v_TotalInreg NUMBER(15,0);
 CURSOR c_Marfuri IS
  SELECT *
  FROM Marfuri
  WHERE DenP=v_DenP;
 CURSOR c_TotalInreg IS
  SELECT count(*)
  FROM Marfuri;
BEGIN
 v_NrOp:=&NrOp;
 OPEN c_TotalInreg;
 FETCH c_TotalInreg INTO v_TotalInreg;
 CLOSE c_TotalInreg;
 v_DenP:='&DenP';
 v_Index:= '&Index';
 v_TipIndex:= '&TipIndex';
 v_NrIndex:=&NrIndex;
 v_Operatie:='SELECT * FROM Marfuri WHERE DenP=';
 v_Start:=DBMS_UTILITY.GET_TIME;
 OPEN c_Marfuri;
 v_End:=DBMS_UTILITY.GET_TIME;
 v_Timp:=(v_End-v_Start)*10;
 LOOP
   F ETCH c_Marfuri INTO v_Marfuri;
   EXIT WHEN c_Marfuri%NOTFOUND;
```

```
 END LOOP;
 v_NrInreg:=c_Marfuri%ROWCOUNT;
 CLOSE c_Marfuri;
 INSERT INTO Rezultate(NrOp, Operatie, TotalInreg, InregPrel,Timp, Valoare, Index, TipIndex,
        NrIndex) VALUES (v_NrOp, v_Operatie, v_TotalInreg, v_NrInreg, v_Timp, v_DenP,
        v_Index, v_TipIndex, v_NrIndex);
 COMMIT;
END;
```

La rularea scriptului se furnizează ca parametrii numărul operației, valoarea pentru DENP care va fi folosită în clauza WHERE a blocului SELECT, dacă există sau nu index asociat fişierului de date, tipul indexului, precum şi numărul de fişiere index asociate fişierului de date. Timpul de execuție al operației SELECT, în cazul particular specificat, este inserat în tabela REZULTATE.

Acest tip de interogare a fost executat în mod repetat folosindu-se diferite valori pentru DENP. În anexa 1 pot fi consultate o parte dintre rezultatele obținute.

Testele pentru SQL Server2000 au folosit procedura stocată MarfuriDenP de mai jos:

```
/*procedura stocată MarfuriDenP*/
CREATE PROCEDURE dbo.MarfuriDenP
(       @nrOp NUMERIC(18,0), @vDenP VARCHAR(50), @index CHAR(1), @tipIndex
VARCHAR(20), @nrIndex NUMERIC(2,0)
        )
        AS
        SET NOCOUNT ON
        BEGIN TRANSACTION
                DECLARE @op VARCHAR(200),@t1 NUMERIC(18,0),@t2 NUMERIC(18,0),@s1
INT,@s2 INT,@total NUMERIC(18,0),@inreg NUMERIC(18,0),@dif NUMERIC(18,6)
                SET @op = 'SELECT * FROM Marfuri WHERE DenP='
                SELECT @total=count(*) FROM Marfuri
                SELECT @inreg = count(*)  FROM Marfuri WHERE DenP=@vDenP
                SELECT @t1=datepart(ms,getdate()),@s1=datepart(s,getdate())
                SELECT *  FROM Marfuri WHERE DenP=@vDenP
                SELECT @t2=datepart(ms,getdate()),@s2=datepart(s,getdate())
                if @s2>@s1
                        SET @dif=(@s2*1000+@t2)-(@s1*1000+@t1)
                else
                        SET @dif=@t2-@t1
                INSERT Rezultate VALUES
```

```
                (@nr,@op,@total,@inreg,@dif,@vDenP,@index,@tipIndex,@nrIndex)
if @@error<>0
BEGIN
        ROLLBACK TRANSACTION
        RAISERROR ('Operatiunea nu s-a putut realiza din cauza unei erori!!',16,1) with NOWAIT
END
ELSE
BEGIN
        COMMIT TRANSACTION
        RETURN(0)
END
GO
```

La executarea procedurii se furnizează ca parametrii numărul operației, valoarea pentru DENP care va fi folosită în clauza WHERE a blocului SELECT, dacă există sau nu index asociat fişierului de date, tipul indexului, precum şi numărul de fişiere index asociate fişierului de date. Timpul de execuție al operației SELECT, pentru fiecare caz particular specificat, este inserat în tabela REZULTATE.

Acest tip de interogare a fost executat de asemenea în mod repetat, folosindu-se diferite valori pentru DENP. Unele dintre rezultatele obținute pot fi consultate în anexa 1.

Interogarea este bazată pe o condiție ce implică denumirea produsului care nu este unică la nivelul relației, deci sunt returnate mai multe tupluri ca rezultat al interogării. Numărul de tupluri returnate ca răspuns al interogării depinde, de asemenea, de valoarea precizată în interorage pentru DENP şi se situează în general în jurul valorii de 1% din numărul total de tupluri al relației. Spre exemplu, pentru fişier MARFURI având 4.240 de înregistrări, interogarea pentru DENP='ZAHAR' returnează 40 de înregistrări. Pentru acest tip de interogare s-a creat un fişier index având ca şi cheie de indexare denumirea produsului, şi anume:

```
CREATE INDEX I_MARF_DEN ON MARFURI(DENP);
```

Cheia de indexare folosită în acest caz este de dimensiune mai mare decât cea anterioară, pentru că s-a urmărit să se analizeze şi influența mărimii cheii de indexare asupra timpilor de răspuns la interogări.

Testele s-au efectuat pe fişiere de date a căror mărime a crescut într-o primă etapă de la 10 Ko până la 20 Mo, iar în a doua etapă de la 20 Mo până la 310 Mo.

Deoarece în tabela REZULTATE au fost colectați timpii de execuție obținuți pentru execuții repetate ale aceleiași operații de selecție pentru fiecare caz distinct de valoare din clauza WHERE, aceste date au fost sintetizate în tabela REZULTATE_FINALE. Inregistrările din tabela REZULTATE_FINALE au fost obținute prin rularea scripului:

```
--RezultateFinale.sql
BEGIN
  INSERT INTO Rezultate_Finale(NrOp,Operatie,TotalInreg,InregPrel,Timp,Index,TipIndex,NrIndex)
    SELECT NrOp, Operatie, TotalInreg, InregPrel, AVG(Timp), Index, TipIndex, NrIndex
    FROM Rezultate
    WHERE Index='D' AND TipIndex='Clasic' AND NrIndex=1
    GROUP BY NrOp,TotalInreg;
  INSERT INTO Rezultate_Finale(NrOp,Operatie,TotalInreg,InregPrel,Timp,Index,TipIndex,NrIndex)
    SELECT NrOp, Operatie, TotalInreg, InregPrel, AVG(Timp), Index, TipIndex, NrIndex
     FROM Rezultate
     WHERE Index='N'
     GROUP BY NrOp,TotalInreg;
  COMMIT;
END;
```

sau a procedurii stocate:

```
/*procedura stocată RezultateFinale*/
CREATE PROCEDURE dbo.RezultateFinale( )
AS
SET NOCOUNT ON
BEGIN TRANSACTION
  INSERT INTO Rezultate_Finale(NrOp,Operatie,TotalInreg,InregPrel,Timp,Index,TipIndex,NrIndex)
        SELECT NrOp, Operatie, TotalInreg, InregPrel, AVG(Timp), Index, TipIndex, NrIndex
        FROM Rezultate
        WHERE Index='D' AND TipIndex='Clasic' AND NrIndex=1
        GROUP BY NrOp,TotalInreg
  INSERT INTO Rezultate_Finale(NrOp,Operatie,TotalInreg,InregPrel,Timp,Index,TipIndex,NrIndex)
        SELECT NrOp, Operatie, TotalInreg, InregPrel, AVG(Timp), Index, TipIndex, NrIndex
        FROM Rezultate
        WHERE Index='N'
        GROUP BY NrOp,TotalInreg
if @@error<>0
BEGIN
```

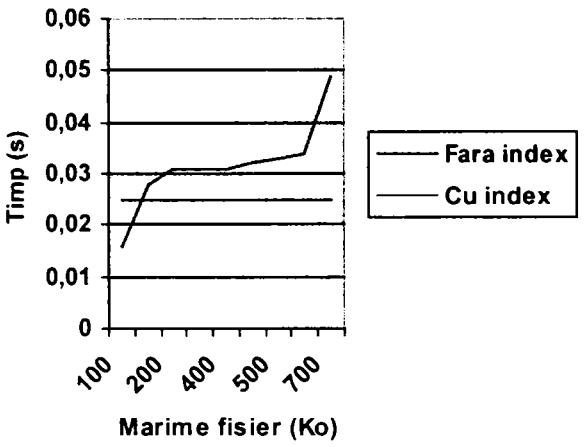
```
        ROLLBACK TRANSACTION
        RAISERROR ('Operatiunea nu s-a putut realiza din cauza unei erori!!',16,1) with NOWAIT
END
ELSE
BEGIN
        COMMIT TRANSACTION
        RETURN(0)
END
GO
```

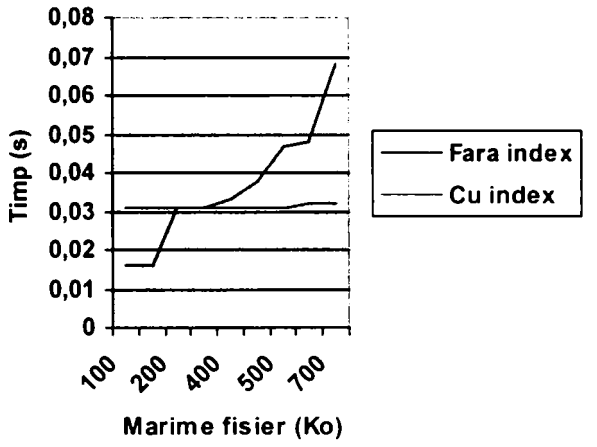
Informatiile obținute în tabela REZULTATE_FINALE pentru aceste teste au fost analizate şi interpretate şi sunt prezentate în tabelele 2-4 şi în graficele 1-5. Tabelul 2 prezintă rezultatele obținute la testele efectuate asupra tabelei FURNIZORI.

Tabel 2 Timpii de răspuns obținuți pentru tabela FURNIZORI

Număr Înregistrări	Mărime fişier (Ko)	COD_FURN (returnează 1 tuplu)			LOC_FURN (returnează 3% tupluri)		
		Timp acces fără index (s)	Timp acces cu index (s)	Imbunătățire (%)	Timp acces fără index (s)	Timp acces cu index (s)	Imbunătățire (%)
424	104	0,016	0,025	-55	0,016	0,031	-95
848	104	0,016	0,025	-55	0,016	0,031	-85
1272	160	0,028	0,025	11	0,031	0,031	0
1696	224	0,031	0,025	20	0,031	0,031	0
2968	344	0,031	0,025	20	0,031	0,031	0
3392	400	0,032	0,025	22	0,033	0,031	6
3816	464	0,031	0,025	20	0,038	0,031	18
4240	464	0,032	0,025	22	0,038	0,031	18
4664	520	0,033	0,025	25	0,047	0,031	34
5088	584	0,034	0,025	26	0,048	0,032	33
5512	640	0,046	0,025	45	0,068	0,032	53
5936	704	0,049	0,025	49	0,078	0,038	51
6360	760	0,052	0,025	52	0,086	0,039	54

Coloana *Imbunătățire* din tabelele următoare prezintă informație, în procente, despre reducerea timpului de acces la date în cazul folosirii unui fişier index auxiliar fişierului de date. Procentul reprezintă cu cât la sută s-a redus timpul de acces fără index în cazul introducerii indexului. Rezultatele din tabelul 2 sunt reprezentate grafic în graficul 1şi 2 pentru o mai uşoară interpretare a acestora. Graficul 1 prezintă rezultatele obținute pentru interogarea după cod furnizor, iar graficul 2 rezultatele pentru interogarea după localitate furnizor.





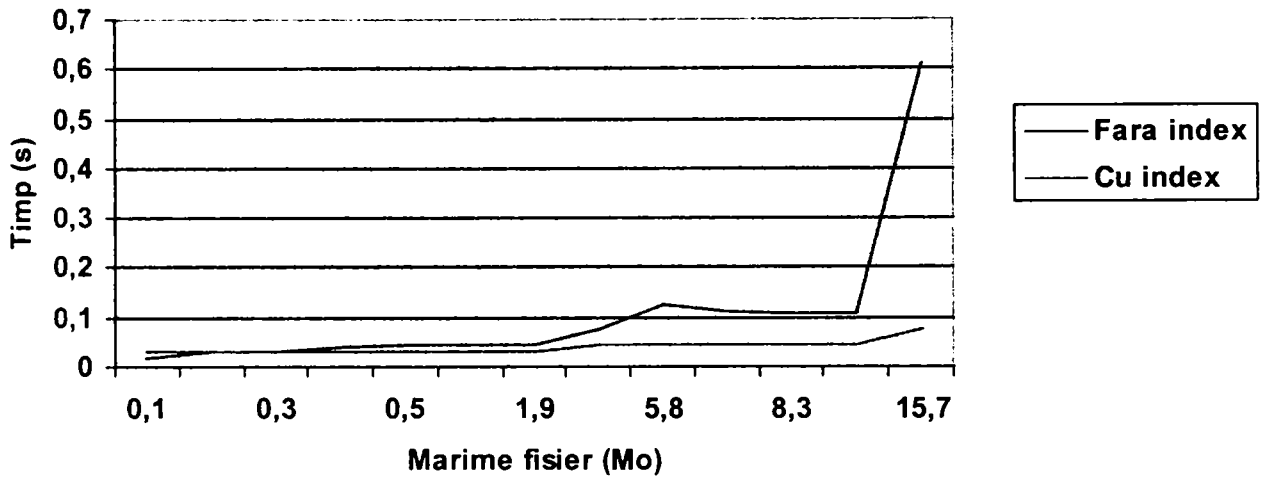
Tabelul 3 prezintă rezultatele obținute la testele efectuate asupra tabelei MARFURI ,pentru situația unui fișier de date a cărui mărime a variat între 0,1 Mo şi 16 Mo.

Tabel 3. Timpii de răspuns obținuți pentru tabela MARFURI

Număr înregistrări	Mărime fișier (Mo)	DENP (returnează 1% tupluri)		
		Timp acces fără index (s)	Timp acces cu index (s)	Imbunătățire (%)
1.263	0,1	0,016	0,031	-95
2.526	0,2	0,031	0,031	0
3.789	0,33	0,031	0,031	0
5.052	0,4	0,040	0,031	22
6.315	0,5	0,047	0,032	32
12.630	0,9	0,047	0,031	34
25.260	1,9	0,047	0,032	32
50.520	3,8	0,078	0,047	40
75.780	5,8	0,125	0,046	63
93.462	6,8	0,11	0,047	58
113.670	8,3	0,109	0,047	57
138.930	10	0,109	0,047	57
214.710	16	0,609	0,078	87

Rezultatele obținute în tabelul 3 sunt reprezentate grafic prin graficul 3.

Grafic 3. MARFURI - DENP (1% tupluri)



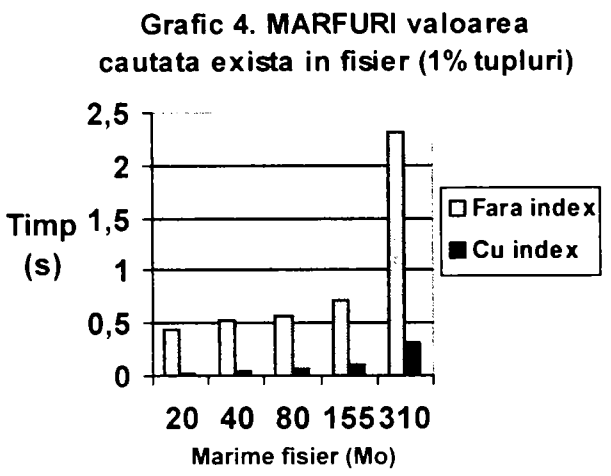
Tabelul 4 prezintă o serie de teste făcute tot asupra tabelei MARFURI, pentru situația unui fișier de date a cărui mărime a variat între 20 Mo şi 310 Mo..

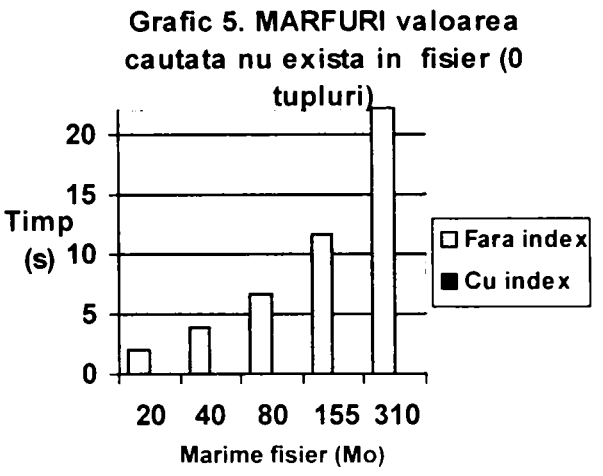
Tabel 4. Timpii de răspuns obținuți pentru tabela MARFURI

Număr înregistrări	Mărime fișier (Mo)	Valorile căutate există în fișierul de date (returnează 1% tupluri			Valorile căutate nu există în fișierul de date (returnează 0 tupluri)		
		Timp acces fără index (s)	Timp acces cu index (s)	Imbunătățire (%)	Timp acces fără index (s)	Timp acces cu index (s)	Imbunătățire (%)
265.230	20	0,437	0,031	93	2,076	0,016	99,23
530.460	40	0,532	0,047	91	3,890	0,016	99,59
1.060.920	80	0,562	0,063	89	6,587	0,016	99,76
2.121.840	155	0,720	0,109	85	11,672	0,016	99,86
4.234.680	310	2,320	0,320	86	22,109	0,016	99,93

Graficul 4 şi 5 prezintă datele obținute în tabelul 4.

Testele au fost efectuate atât pe produsul Oracle 9*i* cât şi produsele SQL Server 2000 şi DB2. În alegerea testelor s-a ținut cont de observațiile din [Honou 98], [Micro 03-7], [Bloor 03-2]. Comportamentul observat la acest tip de teste nu diferă de la un produs la altul. Deoarece Oracle *9i* este cel mai performant dintre aceste trei produse, rezultatele prezentate în tabelele şi graficele anterioare cuprind datele obținute la testele pe Oracle 9i .





Rezultatele obținute permit tragerea următoarelor concluzii:

- **pentru fișiere de date de mărime mică (0 -1 Mo)**

  - interogarea returnează un singur tuplu al relației

⇒ dacă fișierul de date are sub 150 Ko timpul de răspuns este mai mare dacă există fișier index asociat fișierului de date, cu cca. 50 %;

⇒ dacă fișierul de date are peste 150 Ko timpii de răspuns sunt mai buni dacă există fișier index asociat fișierului de date:

  - pentru mărimi ale fișierului de date cuprinse între 150 – 600 Ko îmbunătățirea timpilor de răspuns este de cca. 20 %;
  - pentru mărimi ale fișierului de date cuprinse între 600 – 900 Ko îmbunătățirea timpilor de răspuns este de cca. 50 %;

  - interogarea returnează mai multe tupluri ale relației (sub 5% din numărul de tupluri ale relației)

⇒ dacă fișierul de date are sub 150 Ko timpul de răspuns este mai mare dacă există fișier index asociat fișierului de date, cu cca. 90 %;

⇒ dacă fișierul de date are între 150 – 450 Ko timpii de răspuns sunt aproximativ identici în prezența sau în absența indexului asociat fișierului de date;

⇒ dacă fişierul de date are peste 450 Ko timpii de răspuns sunt mai buni dacă există fişier index asociat fişierului de date:

- pentru mărimi ale fişierului de date cuprinse între 450 – 600 Ko îmbunătățirea timpilor de răspuns este de cca. 30 - 35 %;
- pentru mărimi ale fişierului de date cuprinse între 600 – 900 Ko îmbunătățirea timpilor de răspuns este de cca. 50 %;

- **pentru fişiere de date de mărime medie (1 – 20 Mo)**

⇒ timpii de răspuns se îmbunătățesc considerabil dacă există fişier index asociat fişierului de date:

- pentru mărimi ale fişierului de date cuprinse între 1 – 2,5 Mo îmbunătățirea timpilor de răspuns este de cca. 35 %;
- pentru mărimi ale fişierului de date cuprinse între 2,5 – 4,5 Mo îmbunătățirea timpilor de răspuns este de cca. 50 %;
- pentru mărimi ale fişierului de date cuprinse între 4,5 – 20 Mo îmbunătățirea timpilor de răspuns este de peste 50 % mergând până la 90 %;

- **pentru fişiere de date de mărime mare (peste 20 Mo)**

⇒ timpii de răspuns se îmbunătățesc foarte mult dacă există fişier index asociat fişierului de date:

➢ dacă valoarea căutată există în fişierul de date

- pentru mărimi ale fişierului de date cuprinse între 20 – 80 Mo îmbunătățirea timpilor de răspuns este de cca. 90 - 95 %;
- pentru mărimi ale fişierului de date cuprinse între 80 – 300 Mo îmbunătățirea timpilor de răspuns este de cca. 85 - 90 %;

- dacă valoarea căutată nu există în fişier

  - timpii de răspuns scad de câteva sute de ori dacă fişierul de date are asociat un fişier index pe condiția de căutare;
  - pentru mărimi ale fişierului de date de cca. 20 Mo timpii de răspuns scad de cca. 100 de ori dacă există index asociat fişierului de date;
  - dublarea mărimii fişierului de date atrage după sine şi dublarea avantajului la nivelul timpilor de răspuns în prezența indexului.

Sinteza rezultatelor este următoarea:

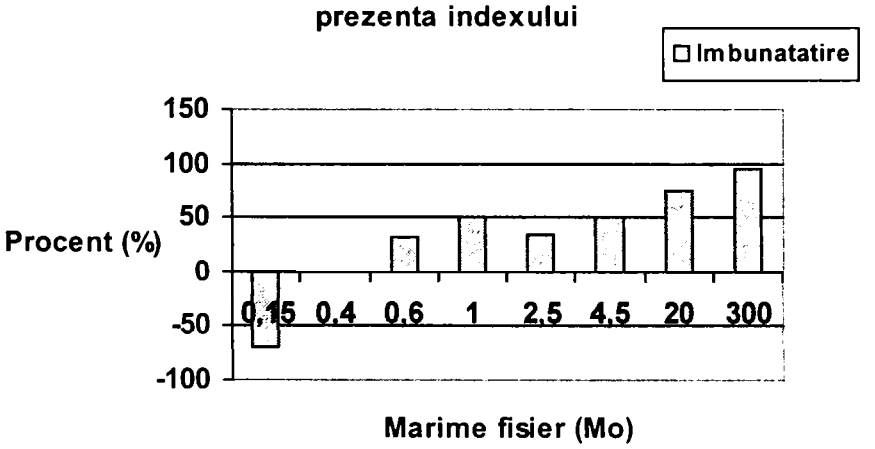
- dacă fişierul de date este < 450 Ko indexul nu îmbunătățeşte timpii de răspuns;

- dacă fişierul de date este > 450 Ko indexul îmbunătățeşte timpii de răspuns, îmbunătățirea este direct proporțională cu creşterea mărimii fişierului de date;

- dacă valoarea căutată nu există în fişierul de date existența indexului îmbunătăteşte foarte, foarte mult timpii de răspuns (de câteva sute de ori). Câştigul datorat existenței indexului este direct proporțional cu mărimea fişierului de date.

- îmbunătățirea timpilor de răspuns la interogări în prezența fişierului index asociat fişierului de date este prezentată în graficul 6.

Grafic 6. Imbunatatirea timpilor de raspuns in prezenta indexului



3.3.2. Influența numărului de fișiere index asupra operațiilor de actualizare

Pentru că asupra tabelelor cu date se execută nu numai operații de regăsire a datelor, mi-am propus să studiez:

Care este influența indecşilor asociați tabelei de date asupra performanței bazei de date. în condițiile în care asupra tabelei de date se efectuează operații de inserare, modificare sau ştergere?

Aceste teste le-am împărțit în două mari categorii:

- influența numărului de indecşi asociați unei tabele asupra performanței bazei de date în situația în care asupra tabelei se efectuează operații de actualizare, care prelucrează un număr mic de rânduri ale tabelei (sub 5% din mărimea tabelei);

- influența numărului de indecşi asociați unei tabele asupra performanței bazei de date în situația în care asupra tabelei se efectuează operații de actualizare, care prelucrează un număr mare de rânduri ale tabelei (în jur de 15-25% din mărimea tabelei).

3.3.2.1. Operațiile de actualizare afectează un număr mic de înregistrări din fişierul de date. Concluzii.

Următoarele teste şi-au propus să studieze influența numărului de fişiere index, asociate unui fişier de date, asupra timpilor de răspuns la operațiile de actualizare efectuate asupra fişierului de date. Fişierele index, fiind fişiere auxiliare fişierelor de date, trebuie să fie în permanență o oglindă vie a datelor prezente în fişierele de date, de aceea ele sunt actualizate de fiecare dată când la nivelul fişierului de date se efectuează operații de inserare, modificare sau ştergere care afectează aceste structuri [Honou 98], [Micro 03-1], [IBM 02]. Aceste actualizări suplimentare ale tuturor structurilor indexate asociate unei tabele cu date pot să atragă după sine o creştere a timpilor de execuție ai operațiilor de inserare, modificare sau ştergere efectuate asupra tabelelor unei baze de date.

Cu cât devin operațiile de actualizare ale tabelelor unei baze de date mai lente în prezența unor structuri indexate asociate acestor tabele?

Aceasta este principala întrebare la care am căutat răspuns prin aceste teste. Tabelele folosite în cadrul testelor au fost tot FURNIZORI şi MARFURI. Acestea au tot o structură de tip heap. Asociat fiecărei tabele s-au creat până la 6 structuri indexate clasice de tip arbore B.

- **Prezentarea testelor**

Operațiile de inserare, modificare, ştergere executate asupra fişierelor de date afectează un număr mic de înregistrări din fişier, raportat la mărimea acestuia (sub 5%).

Inserările s-au făcut pe baza unor blocuri SQL [32] de forma următoare:

```
INSERT INTO FURNIZORI
SELECT *
FROM FURNIZORI_NOI;
```

```
INSERT INTO MARFURI
SELECT *
FROM MARFURI_NOI;
```

În Oracle 9*i* pentru inserare s-au folosit scripturile FurnizoriInsert.sql şi MarfuriInsert.sql care la fiecare apelare inserează un bloc de rânduri în tabelele FURNIZORI sau MARFURI şi contorizează timpul de execuție al operației de inserare în tabela REZULTATE. Operațiile de inserare se execută atât pentru situația când fişierul de date nu are fişier index asociat, cât şi pentru situațiie în care există fişiere index asociate fişierului de date. Numărul fişierelor index variază de la 1 la 6. În anexa 1 pot fi consultate câteva dintre valorile obținute la aceste teste.

Scripturile FurnizoriInsert şi MarfuriInsert sunt prezentate mai jos:

```
--FurnizoriInsert.sql
DECLARE
 v_NrOp NUMBER(3);
 v_Operatie VARCHAR(200);
 v_Start NUMBER(10,0);
 v_End NUMBER(10,0);
 v_Timp NUMBER(10,0);
 v_NrInreg NUMBER(12,0);
 v_Index CHAR(1);
 v_TipIndex VARCHAR(20);
```

```
 v_NrIndex NUMBER(2,0);
 v_Valoare CHAR(50);
 v_TotalInreg NUMBER(15,0);
 CURSOR c_TotalInreg IS
  SELECT count(*)
  FROM Furnizori;
CURSOR c_NrInreg IS
  SELECT count(*)
  FROM Furnizori_Noi;
BEGIN
 v_NrOp:=&NrOp;
 v_Valoare:= ' ';
 OPEN c_TotalInreg;
 FETCH c_TotalInreg INTO v_TotalInreg;
 CLOSE c_TotalInreg;
 OPEN c_NrInreg;
 FETCH c_NrInreg INTO v_NrInreg;
 CLOSE c_NrInreg;
 v_Index:= '&Index';
 v_TipIndex:= '&TipIndex';
 v_NrIndex:=&NrIndex;
 v_Operatie:='INSERT INTO Furnizori SELECT * FROM Furnizori_Noi';
 v_Start:=DBMS_UTILITY.GET_TIME;
 INSERT INTO Furnizori SELECT * FROM Furnizori_Noi;
 v_End:=DBMS_UTILITY.GET_TIME;
 v_Timp:=(v_End-v_Start)*10;
 INSERT INTO Rezultate(NrOp, Operatie, TotalInreg, InregPrel,Timp, Valoare, Index, TipIndex,
        NrIndex) VALUES (v_NrOp, v_Operatie, v_TotalInreg, v_NrInreg, v_Timp, v_Valoare,
        v_Index, v_TipIndex, v_NrIndex);
 COMMIT;
END;
```

--MarfuriInsert.sql

```
DECLARE
 v_NrOp NUMBER(3);
 v_Operatie VARCHAR(200);
 v_Start NUMBER(10,0);
 v_End NUMBER(10,0);
 v_Timp NUMBER(10,0);
 v_NrInreg NUMBER(12,0);
```

```
 v_Index CHAR(1);
 v_TipIndex VARCHAR(20);
 v_NrIndex NUMBER(2,0);
 v_Valoare CHAR(50);
 v_TotalInreg NUMBER(15,0);
 CURSOR c_TotalInreg IS
   SELECT count(*)
   FROM Marfuri;
CURSOR c_NrInreg IS
   SELECT count(*)
   FROM Marfuri_Noi;
BEGIN
 v_NrOp:=&NrOp;
 v_Valoare:= ' ';
 OPEN c_TotalInreg;
 FETCH c_TotalInreg INTO v_TotalInreg;
 CLOSE c_TotalInreg;
 OPEN c_NrInreg;
 FETCH c_NrInreg INTO v_NrInreg;
 CLOSE c_NrInreg;
 v_Index:= '&Index';
 v_TipIndex:= '&TipIndex';
 v_NrIndex:=&NrIndex;
 v_Operatie:='INSERT INTO Marfuri SELECT * FROM Marfuri_Noi';
 v_Start:=DBMS_UTILITY.GET_TIME;
 INSERT INTO Marfuri SELECT * FROM Marfuri_Noi;
 v_End:=DBMS_UTILITY.GET_TIME;
 v_Timp:=(v_End-v_Start)*10;
 INSERT INTO Rezultate(NrOp, Operatie, TotalInreg, InregPrel,Timp, Valoare, Index, TipIndex,
        NrIndex) VALUES (v_NrOp, v_Operatie, v_TotalInreg, v_NrInreg, v_Timp, v_Valoare,
        v_Index, v_TipIndex, v_NrIndex);
 COMMIT;
END;
```

În cazul SQL Server2000 pentru aceste teste s-au folosit procedurile stocate FurnizoriInsert şi MarfuriInsert de mai jos:

```
/*procedura stocată FurnizoriInsert*/
CREATE PROCEDURE dbo.FurnizoriInsert
```

```
(       @nrOp NUMERIC(18,0), @index CHAR(1), @tipIndex VARCHAR(20), @nrIndex
NUMERIC(2,0)
)
AS
SET NOCOUNT ON
BEGIN TRANSACTION
        DECLARE @op VARCHAR(200),@t1 NUMERIC(18,0),@t2 NUMERIC(18,0),@s1
INT,@s2 INT,@total NUMERIC(18,0),@inreg NUMERIC(18,0),@dif NUMERIC(18,6)
        SET @op = 'INSERT INTO Furnizori SELECT * FROM Furnizori_Noi'
        SELECT @total=count(*) FROM Furnizori
        SELECT @inreg = count(*) FROM Furnizori_Noi
        SELECT @t1=datepart(ms,getdate()),@s1=datepart(s,getdate())
        INSERT INTO Furnizori SELECT * FROM Furnizori_Noi
        SELECT @t2=datepart(ms,getdate()),@s2=datepart(s,getdate())
        if @s2>@s1
                SET @dif=(@s2*1000+@t2)-(@s1*1000+@t1)
        else
                SET @dif=@t2-@t1
        INSERT Rezultate VALUES
                (@nr,@op,@total,@inreg,@dif,null,@index,@tipIndex,@nrIndex)
    if @@error<>0
    BEGIN
        ROLLBACK TRANSACTION
        RAISERROR ('Operatiunea nu s-a putut realiza din cauza unei erori!!',16,1) with NOWAIT
    END
    ELSE
    BEGIN
        COMMIT TRANSACTION
        RETURN(0)
    END
    GO

    /*procedura stocată MarfuriInsert*/
    CREATE PROCEDURE dbo.MarfuriInsert
    (       @nrOp NUMERIC(18,0), @index CHAR(1), @tipIndex VARCHAR(20), @nrIndex
NUMERIC(2,0)
    )
    AS
    SET NOCOUNT ON
    BEGIN TRANSACTION
```

```
DECLARE @op VARCHAR(200),@t1 NUMERIC(18,0),@t2 NUMERIC(18,0),@s1
INT,@s2 INT,@total NUMERIC(18,0),@inreg NUMERIC(18,0),@dif NUMERIC(18,6)
SET @op = 'INSERT INTO Marfuri SELECT * FROM Marfuri_Noi'
SELECT @total=count(*) FROM Marfuri
SELECT @inreg = count(*) FROM Marfuri_Noi
SELECT @t1=datepart(ms,getdate()),@s1=datepart(s,getdate())
INSERT INTO Marfuri SELECT * FROM Marfuri_Noi
SELECT @t2=datepart(ms,getdate()),@s2=datepart(s,getdate())
if @s2>@s1
        SET @dif=(@s2*1000+@t2)-(@s1*1000+@t1)
else
        SET @dif=@t2-@t1
INSERT Rezultate VALUES
        (@nr,@op,@total,@inreg,@dif,null,@index,@tipIndex,@nrIndex)
if @@error<>0
BEGIN
        ROLLBACK TRANSACTION
        RAISERROR ('Operatiunea nu s-a putut realiza din cauza unei erori!!',16,1) with NOWAIT
END
ELSE
BEGIN
        COMMIT TRANSACTION
        RETURN(0)
END
GO
```

Şi în acest caz se inserează blocuri de înregistrări la o apelare a procedurii. Timpii de execție ai operațiilor de inserare sunt contorizați şi salvați în tabela REZULTATE. Numărul fişierelor index asociate fişierelor de date variază de la 0 la 6. În anexa 1 pot fi consultate câteva dintre rezultatele obținute la această categorie de teste.

Numărul de tupluri inserate la o apelare a operației se situează în toate cazurile în jurul valorii de 3% din numărul total de tupluri ale relației în care se face inserarea. De exemplu, pentru fişierul FURNIZORI de mărime 160 Ko având 1.272 înregistrări s-au adăugat 38 de înregistrări.

Modificările s-au efectuat pe baza unor blocuri SQL [Kochh 00]] de forma următoare:

```
UPDATE FURNIZORI
SET TAR_FURN='ITALIA'
```

```
UPDATE MARFURI
SET PRET_ACHIZ=PRET_ACHIZ*1.1
```

WHERE TAR_FURN='ITL'; WHERE GRUPA='RAUCH';

În Oracle 9*i* modificările s-au efectuat cu ajutorul scripturilor FurnizoriModificare şi MarfuriModificare de mai jos:

```
--FurnizoriModificare.sql
DECLARE
 v_NrOp NUMBER(3);
 v_Operatie VARCHAR(200);
 v_Start NUMBER(10,0);
 v_End NUMBER(10,0);
 v_Timp NUMBER(10,0);
 v_NrInreg NUMBER(12,0);
 v_Vechi_Tar_Furn Furnizori.Tar_Furn%TYPE;
 v_Nou_Tar_Furn Furnizori.Tar_Furn%TYPE;
 v_Index CHAR(1);
 v_TipIndex VARCHAR(20);
 v_NrIndex NUMBER(2,0);
 v_TotalInreg NUMBER(15,0);
 CURSOR c_TotalInreg IS
  SELECT count(*)
   FROM Furnizori;
CURSOR c_NrInreg IS
   SELECT count(*)
   FROM Furnizori
   WHERE Tar_Furn=v_Nou_Tar_Furn;
BEGIN
 v_NrOp:=&NrOp;
 OPEN c_TotalInreg;
 FETCH c_TotalInreg INTO v_TotalInreg;
 CLOSE c_TotalInreg;
 v_Vechi_Tar_Furn:= '&Vechi_Tar_Furn';
 v_Nou_Tar_Furn:= '&Nou_Tar_Furn';
 v_Index:= '&Index';
 v_TipIndex:= '&TipIndex';
 v_NrIndex:=&NrIndex;
 v_Operatie:='UPDATE Furnizori SET Tar_Furn= WHERE Tar_Furn=';
 v_Start:=DBMS_UTILITY.GET_TIME;
 UPDATE Furnizori SET Tar_Furn=v_Nou_Tar_Furn WHERE Tar_Furn=v_Vechi_Tar_Furn;
```

```
 v_End:=DBMS_UTILITY.GET_TIME;
 v_Timp:=(v_End-v_Start)*10;
 OPEN c_NrInreg
 FETCH c_NrInreg INTO v_NrInreg;
 CLOSE c_NrInreg;
 INSERT INTO Rezultate(NrOp, Operatie, TotalInreg, InregPrel,Timp, Valoare, Index, TipIndex,
        NrIndex) VALUES (v_NrOp, v_Operatie, v_TotalInreg, v_NrInreg, v_Timp,
        v_Vechi_Tar_Furn, v_Index, v_TipIndex, v_NrIndex);
 COMMIT;
END;
```

La rularea scriptului se furnizează ca parametrii numărul operației, valoarea ce va fi modificată, valoarea cu care acesta va fi înlocuită, dacă există sau nu index asociat fişierului de date, tipul indexului şi numărul de fişiere index asociate fişierului de date. Pentru situațiile cu 2-6 fişiere index, scriptul a fost modificat astfel încât modificările făcute asupra fişierului de date să implice toate fişierele index. Rezultatele testelor sunt colectate în tabela REZULTATE şi o parte dintre ele pot fi consultate în anexa 1.

```
--MarfuriModificare.sql
DECLARE
 v_NrOp NUMBER(3);
 v_Operatie VARCHAR(200);
 v_Start NUMBER(10,0);
 v_End NUMBER(10,0);
 v_Timp NUMBER(10,0);
 v_NrInreg NUMBER(12,0);
 v_Grupa Marfuri.Grupa%TYPE;
 v_Index CHAR(1);
 v_TipIndex VARCHAR(20);
 v_NrIndex NUMBER(2,0);
 v_TotalInreg NUMBER(15,0);
 CURSOR c_TotalInreg IS
   SELECT count(*)
   FROM Marfuri;
CURSOR c_NrInreg IS
   SELECT count(*)
   FROM Marfuri
   WHERE Grupa=v_Grupa;
BEGIN
```

```
v_NrOp:=&NrOp;
OPEN c_TotalInreg;
FETCH c_TotalInreg INTO v_TotalInreg;
CLOSE c_TotalInreg;
v_Grupa:= '&Grupa';
v_Index:= '&Index';
v_TipIndex:= '&TipIndex';
v_NrIndex:=&NrIndex;
v_Operatie:='UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Grupa=';
v_Start:=DBMS_UTILITY.GET_TIME;
UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Grupa=v_Grupa;
v_End:=DBMS_UTILITY.GET_TIME;
v_Timp:=(v_End-v_Start)*10;
OPEN c_NrInreg
FETCH c_NrInreg INTO v_NrInreg;
CLOSE c_NrInreg;
INSERT INTO Rezultate(NrOp, Operatie, TotalInreg, InregPrel,Timp, Valoare, Index, TipIndex,
      NrIndex) VALUES (v_NrOp, v_Operatie, v_TotalInreg, v_NrInreg, v_Timp,
      v_Grupa, v_Index, v_TipIndex, v_NrIndex);
COMMIT;
END;
```

Şi în acest caz scriptul a fost modificat în situația a 2-6 fişiere index asociate fişierului de date, astfel încât toate aceste fişiere index să fie actualizate ca urmare a modificărilor făcute în fişierul de date. Rezultatele testelor sunt colectate în tabela REZULTATE şi o parte a acestora pot fi consultate în anexa 1.

În cazul SQL Server2000 pentru aceste teste s-au folosit procedurile stocate FurnizoriModificare şi MarfuriModificare de mai jos:

```
/*procedura stocată FurnizoriModificare*/
CREATE PROCEDURE dbo.FurnizoriModificare
(     @nrOp NUMERIC(18,0), @vVechi_Tar_Furn VARCHAR(15), @vNou_Tar_Furn
VARCHAR(15), @index CHAR(1), @tipIndex VARCHAR(20), @nrIndex NUMERIC(2,0)
)
AS
SET NOCOUNT ON
BEGIN TRANSACTION
      DECLARE @op VARCHAR(200),@t1 NUMERIC(18,0),@t2 NUMERIC(18,0),@s1
INT,@s2 INT,@total NUMERIC(18,0),@inreg NUMERIC(18,0),@dif NUMERIC(18,6)
```

```
        SET @op = 'UPDATE Furnizori SET Tar_Furn= WHERE Tar_Furn='
        SELECT @total=count(*) FROM Furnizori
        SELECT @t1=datepart(ms,getdate()),@s1=datepart(s,getdate())
        UPDATE Furnizori SET Tar_Furn=@vNou_Tar_Furn
                        WHERE Tar_Furn=@vVechi_Tar_Furn
        SELECT @t2=datepart(ms,getdate()),@s2=datepart(s,getdate())
        if @s2>@s1
                SET @dif=(@s2*1000+@t2)-(@s1*1000+@t1)
        else
                SET @dif=@t2-@t1
        SELECT @inreg = count(*)  FROM  Furnizori WHERE Tar_Furn=@vNou_Tar_Furn
        INSERT Rezultate VALUES
                (@nr,@op,@total,@inreg,@dif,@vVechi_Tar_Furn,@index,@tipIndex,@nrIndex)
if @@error<>0
BEGIN
        ROLLBACK TRANSACTION
        RAISERROR ('Operatiunea nu s-a putut realiza din cauza unei erori!!',16,1) with NOWAIT
END
ELSE
BEGIN
        COMMIT TRANSACTION
        RETURN(0)
        END
        GO
```

Procedura FurnizoriModificare a fost modificată în cazul a 2-6 fişiere index astfel încât toate fişierele index asociate fişierului de date să fie afectate de modificările făcute asupra acestuia. Rezultatele testelor sunt colectate în tabela REZULTATE. O parte a acestor rezultate pot fi consultate în anexa 1.

```
        /*procedura stocată MarfuriModificare*/
        CREATE PROCEDURE dbo.MarfuriModificare
        (       @nrOp NUMERIC(18,0), @vGrupa VARCHAR(8), @index CHAR(1), @tipIndex
VARCHAR(20), @nrIndex NUMERIC(2,0)
        )
        AS
        SET NOCOUNT ON
        BEGIN TRANSACTION
```

```
DECLARE @op VARCHAR(200),@t1 NUMERIC(18,0),@t2 NUMERIC(18,0),@s1
INT,@s2 INT,@total NUMERIC(18,0),@inreg NUMERIC(18,0),@dif NUMERIC(18,6)
SET @op = 'UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Grupa='
SELECT @total=count(*) FROM Marfuri
SELECT @t1=datepart(ms,getdate()),@s1=datepart(s,getdate())
UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Grupa=@vGrupa
SELECT @t2=datepart(ms,getdate()),@s2=datepart(s,getdate())
if @s2>@s1
        SET @dif=(@s2*1000+@t2)-(@s1*1000+@t1)
else
        SET @dif=@t2-@t1
SELECT @inreg = count(*) FROM Marfuri WHERE Grupa=@vGrupa
INSERT Rezultate VALUES
        (@nr,@op,@total,@inreg,@dif,@vGrupa,@index,@tipIndex,@nrIndex)
if @@error<>0
BEGIN
        ROLLBACK TRANSACTION
        RAISERROR ('Operatiunea nu s-a putut realiza din cauza unei erori!!',16,1) with NOWAIT
END
ELSE
BEGIN
        COMMIT TRANSACTION
        RETURN(0)
END
GO
```

Şi această procedură a fost modificată în situația în care fişierul de date avea asociate 2-6 fişiere index, astfel încât toate aceste fişiere index să fie afectate de modificările operate în fişierul de date. Rezultatele testelor au fost colectate în tabela REZULTATE şi câteva dintre ele pot fi observate în anexa 1.

Pentru această categorie de teste, operațiile de modificare au vizat, de asemenea, în jur de 3% din numărul total de tupluri ale relației. De exemplu, pentru fişierul MARFURI de 33 Mo având 395.736 înregistrări s-au modificat 11.870 înregistrări care aveau GRUPA='RAUCH'.

Ştergerile s-au efectuat pe baza unor blocuri SQL [Kochh 00] de forma următoare:

```
DELETE                          DELETE
FROM FURNIZORI                  FROM MARFURI
```

WHERE TAR_FURN='AUSTRIA'; WHERE GRUPA='TOBLERON';

În cazul testelor pe Oracle 9*i* au fost folosite scripturile FurnizoriStergere și MarfuriStergere de mai jos:

```
--FurnizoriStergere.sql
DECLARE
 v_NrOp NUMBER(3);
 v_Operatie VARCHAR(200);
 v_Start NUMBER(10,0);
 v_End NUMBER(10,0);
 v_Timp NUMBER(10,0);
 v_NrInreg NUMBER(12,0);
 v_Tar_Furn Furnizori.Tar_Furn%TYPE;
 v_Index CHAR(1);
 v_TipIndex VARCHAR(20);
 v_NrIndex NUMBER(2,0);
 v_TotalInreg NUMBER(15,0);
 CURSOR c_TotalInreg IS
  SELECT count(*)
  FROM Furnizori;
 CURSOR c_NrInreg IS
   SELECT count(*)
   FROM Furnizori
   WHERE Tar_Furn=v_Tar_Furn;
BEGIN
 v_NrOp:=&NrOp;
 OPEN c_TotalInreg;
 FETCH c_TotalInreg INTO v_TotalInreg;
 CLOSE c_TotalInreg;
 v_Tar_Furn:= '&Tar_Furn';
 v_Index:= '&Index';
 v_TipIndex:= '&TipIndex';
 v_NrIndex:=&NrIndex;
 v_Operatie:='DELETE FROM Furnizori WHERE Tar_Furn=';
 OPEN c_NrInreg;
 FETCH c_NrInreg INTO v_NrInreg;
 CLOSE c_NrInreg;
 v_Start:=DBMS_UTILITY.GET_TIME;
```

```
 DELETE FROM Furnizori WHERE Tar_Furn=v_Tar_Furn;
 v_End:=DBMS_UTILITY.GET_TIME;
 v_Timp:=(v_End-v_Start)*10;
 INSERT INTO Rezultate(NrOp, Operatie, TotalInreg, InregPrel,Timp, Valoare, Index, TipIndex,
        NrIndex) VALUES (v_NrOp, v_Operatie, v_TotalInreg, v_NrInreg, v_Timp,
        v_Tar_Furn, v_Index, v_TipIndex, v_NrIndex);
 COMMIT;
END;
```

La rularea scriptului se furnizează ca parametrii numărul operației, valoarea pentru țara furnizorului, dacă există sau nu index asociat fişierului de date, tipul indexului şi numărul de fişiere index asociate fişierului de date. Rezultatele testelor sunt colectate în tabela REZULTATE şi o parte dintre ele pot fi consultate în anexa 1.

```
--MarfuriStergere.sql
DECLARE
 v_NrOp NUMBER(3);
 v_Operatie VARCHAR(200);
 v_Start NUMBER(10,0);
 v_End NUMBER(10,0);
 v_Timp NUMBER(10,0);
 v_NrInreg NUMBER(12,0);
 v_Grupa Marfuri.Grupa%TYPE;
 v_Index CHAR(1);
 v_TipIndex VARCHAR(20);
 v_NrIndex NUMBER(2,0);
 v_TotalInreg NUMBER(15,0);
 CURSOR c_TotalInreg IS
   SELECT count(*)
   FROM Marfuri;
CURSOR c_NrInreg IS
   SELECT count(*)
   FROM Marfuri
   WHERE Grupa=v_Grupa;
BEGIN
 v_NrOp:=&NrOp;
 OPEN c_TotalInreg;
 FETCH c_TotalInreg INTO v_TotalInreg;
 CLOSE c_TotalInreg;
```

```
v_Grupa:= '&Grupa';
v_Index:= '&Index';
v_TipIndex:= '&TipIndex';
v_NrIndex:=&NrIndex;
v_Operatie:='DELETE FROM Marfuri WHERE Grupa=';
OPEN c_NrInreg;
FETCH c_NrInreg INTO v_NrInreg;
CLOSE c_NrInreg;
v_Start:=DBMS_UTILITY.GET_TIME;
DELETE FROM Marfuri WHERE Grupa=v_Grupa;
v_End:=DBMS_UTILITY.GET_TIME;
v_Timp:=(v_End-v_Start)*10;
INSERT INTO Rezultate(NrOp, Operatie, TotalInreg, InregPrel,Timp, Valoare, Index, TipIndex,
      NrIndex) VALUES (v_NrOp, v_Operatie, v_TotalInreg, v_NrInreg, v_Timp,
      v_Grupa, v_Index, v_TipIndex, v_NrIndex);
COMMIT;
END;
```

La rularea scriptului se furnizează ca parametrii numărul operației, grupa căreia aparține marfa care trebuie eliminată, dacă există sau nu index asociat fişierului de date, tipul indexului şi numărul de fişiere index asociate fişierului de date. Rezultatele testelor sunt colectate în tabela REZULTATE şi o parte dintre ele pot fi consultate în anexa 1.

În cazul testelor efectuate pe SQL Server2000 s-au folosit procedurile stocate FurnizoriStergere şi MarfuriStergere de mai jos:

```
/*procedura stocată FurnizoriStergere*/
CREATE PROCEDURE dbo.FurnizoriStergere
(     @nrOp NUMERIC(18,0), @vTar_Furn VARCHAR(15), @index CHAR(1), @tipIndex
VARCHAR(20), @nrIndex NUMERIC(2,0)
)
AS
SET NOCOUNT ON
BEGIN TRANSACTION
      DECLARE @op VARCHAR(200),@t1 NUMERIC(18,0),@t2 NUMERIC(18,0),@s1
INT,@s2 INT,@total NUMERIC(18,0),@inreg NUMERIC(18,0),@dif NUMERIC(18,6)
      SET @op = 'DELETE FROM Furnizori WHERE Tar_Furn='
      SELECT @total=count(*) FROM Furnizori
      SELECT @inreg = count(*)  FROM  Furnizori WHERE Tar_Furn=@vTar_Furn
      SELECT @t1=datepart(ms,getdate()),@s1=datepart(s,getdate())
```

```
        DELETE FROM Furnizori WHERE Tar_Furn=@vTar_Furn
        SELECT @t2=datepart(ms,getdate()),@s2=datepart(s,getdate())
        if @s2>@s1
                SET @dif=(@s2*1000+@t2)-(@s1*1000+@t1)
        else
                SET @dif=@t2-@t1
        INSERT Rezultate VALUES
                (@nr,@op,@total,@inreg,@dif,@vTar_Furn,@index,@tipIndex,@nrIndex)
if @@error<>0
BEGIN
        ROLLBACK TRANSACTION
        RAISERROR ('Operatiunea nu s-a putut realiza din cauza unei erori!!',16,1) with NOWAIT
END
ELSE
BEGIN
        COMMIT TRANSACTION
        RETURN(0)
END
GO
```

Procedura FurnizoriStergere este executată furnizându-se ca parametrii numărul operației, valoarea pentru țara furnizorului pe baza căreia se vor efectua ştergerile, dacă există sau nu index asociat fişierului de date, tipul indexului şi numărul de fişiere index asociate. Rezultatele testelor sunt colectate în tabela REZULTATE. O parte a acestor rezultate pot fi consultate în anexa 1.

```
        /*procedura stocată MarfuriStergere*/
        CREATE PROCEDURE dbo.MarfuriStergere
        (       @nrOp NUMERIC(18,0), @vGrupa VARCHAR(8), @index CHAR(1), @tipIndex
VARCHAR(20), @nrIndex NUMERIC(2,0)
        )
        AS
        SET NOCOUNT ON
        BEGIN TRANSACTION
                DECLARE @op VARCHAR(200),@t1 NUMERIC(18,0),@t2 NUMERIC(18,0),@s1
INT,@s2 INT,@total NUMERIC(18,0),@inreg NUMERIC(18,0),@dif NUMERIC(18,6)
                SET @op = 'DELETE FROM Marfuri WHERE Grupa='
                SELECT @total=count(*) FROM Marfuri
                SELECT @inreg = count(*)  FROM  Marfuri WHERE Grupa=@vGrupa
```

```
        SELECT @t1=datepart(ms,getdate()),@s1=datepart(s,getdate())
        DELETE FROM Marfuri WHERE Grupa=@vGrupa
        SELECT @t2=datepart(ms,getdate()),@s2=datepart(s,getdate())
        if @s2>@s1
                SET @dif=(@s2*1000+@t2)-(@s1*1000+@t1)
        else
                SET @dif=@t2-@t1
        INSERT Rezultate VALUES
                (@nr,@op,@total,@inreg,@dif,@vGrupa,@index,@tipIndex,@nrIndex)
if @@error<>0
BEGIN
        ROLLBACK TRANSACTION
        RAISERROR ('Operatiunea nu s-a putut realiza din cauza unei erori!!',16,1) with NOWAIT
END
ELSE
BEGIN
        COMMIT TRANSACTION
        RETURN(0)
END
GO
```

Procedura MarfuriStergere se execută furnizându-se ca parametrii numărul operației, valoarea pentru grupa mărfii pe baza căreia se vor efectua ştergerile, dacă există sau nu index asociat fişierului de date, tipul indexului şi numărul de fişiere index asociate. Rezultatele testelor sunt colectate în tabela REZULTATE. O parte a acestor rezultate pot fi consultate în anexa 1.

Şi în cazul operațiilor de ştergere numărul de tupluri care se şterg la o apelare a blocului SQL se situează în jurul valorii de 3% din numărul total de tupluri ale relației. De exemplu, pentru acelaşi fişier MARFURI de 33 Mo având 395.736 înregistrări s-au şters 11.080 înregistrări care aveau GRUPA='TOBLERON'.

Deoarece în tabela REZULTATE au fost colectați timpii de execuție obținuți pentru execuții repetate ale aceleiaşi operații de actualizare pentru fiecare caz distinct de valoare din clauza WHERE, aceste date au fost sintetizate în tabela REZULTATE_FINALE. Inregistrările din tabela REZULTATE_FINALE au fost obținute prin rularea scripului:

```
--RezultateFinale1.sql
BEGIN
```

```
INSERT INTO Rezultate_Finale(NrOp,Operatie,TotalInreg,InregPrel,Timp,Index,TipIndex,NrIndex)
  SELECT NrOp, Operatie, TotalInreg, AVG(InregPrel), AVG(Timp), Index, TipIndex, NrIndex
  FROM Rezultate
  WHERE Index='D' AND TipIndex='Clasic' AND NrIndex=1
  GROUP BY NrOp,TotalInreg;
INSERT INTO Rezultate_Finale(NrOp,Operatie,TotalInreg,InregPrel,Timp,Index,TipIndex,NrIndex)
  SELECT NrOp, Operatie, TotalInreg, AVG(InregPrel), AVG(Timp), Index, TipIndex, NrIndex
  FROM Rezultate
  WHERE Index='D' AND TipIndex='Clasic' AND NrIndex=2
  GROUP BY NrOp,TotalInreg;
INSERT INTO Rezultate_Finale(NrOp,Operatie,TotalInreg,InregPrel,Timp,Index,TipIndex,NrIndex)
  SELECT NrOp, Operatie, TotalInreg, AVG(InregPrel), AVG(Timp), Index, TipIndex, NrIndex
  FROM Rezultate
  WHERE Index='D' AND TipIndex='Clasic' AND NrIndex=3
  GROUP BY NrOp,TotalInreg;
INSERT INTO Rezultate_Finale(NrOp,Operatie,TotalInreg,InregPrel,Timp,Index,TipIndex,NrIndex)
  SELECT NrOp, Operatie, TotalInreg, AVG(InregPrel), AVG(Timp), Index, TipIndex, NrIndex
  FROM Rezultate
  WHERE Index='D' AND TipIndex='Clasic' AND NrIndex=4
  GROUP BY NrOp,TotalInreg;
INSERT INTO Rezultate_Finale(NrOp,Operatie,TotalInreg,InregPrel,Timp,Index,TipIndex,NrIndex)
  SELECT NrOp, Operatie, TotalInreg, AVG(InregPrel), AVG(Timp), Index, TipIndex, NrIndex
  FROM Rezultate
  WHERE Index='D' AND TipIndex='Clasic' AND NrIndex=5
  GROUP BY NrOp,TotalInreg;
INSERT INTO Rezultate_Finale(NrOp,Operatie,TotalInreg,InregPrel,Timp,Index,TipIndex,NrIndex)
  SELECT NrOp, Operatie, TotalInreg, AVG(InregPrel), AVG(Timp), Index, TipIndex, NrIndex
  FROM Rezultate
  WHERE Index='D' AND TipIndex='Clasic' AND NrIndex=6
  GROUP BY NrOp,TotalInreg;
INSERT INTO Rezultate_Finale(NrOp,Operatie,TotalInreg,InregPrel,Timp,Index,TipIndex,NrIndex)
  SELECT NrOp, Operatie, TotalInreg, AVG(InregPrel), AVG(Timp), Index, TipIndex, NrIndex
  FROM Rezultate
  WHERE Index='N'
  GROUP BY NrOp,TotalInreg;
COMMIT;
END;
```

sau a procedurii stocate:

```
/*procedura stocată RezultateFinale1*/
CREATE PROCEDURE dbo.RezultateFinale1( )
AS
SET NOCOUNT ON
BEGIN TRANSACTION
  INSERT INTO Rezultate_Finale(NrOp,Operatie,TotalInreg,InregPrel,Timp,Index,TipIndex,NrIndex)
SELECT NrOp, Operatie, TotalInreg, AVG(InregPrel), AVG(Timp), Index, TipIndex, NrIndex FROM Rezultate
WHERE Index='D' AND TipIndex='Clasic' AND NrIndex=1 GROUP BY NrOp,TotalInreg
  INSERT INTO Rezultate_Finale(NrOp,Operatie,TotalInreg,InregPrel,Timp,Index,TipIndex,NrIndex)
SELECT NrOp, Operatie, TotalInreg, AVG(InregPrel), AVG(Timp), Index, TipIndex, NrIndex FROM Rezultate
WHERE Index='D' AND TipIndex='Clasic' AND NrIndex=2 GROUP BY NrOp,TotalInreg
  INSERT INTO Rezultate_Finale(NrOp,Operatie,TotalInreg,InregPrel,Timp,Index,TipIndex,NrIndex)
SELECT NrOp, Operatie, TotalInreg, AVG(InregPrel), AVG(Timp), Index, TipIndex, NrIndex FROM Rezultate
WHERE Index='D' AND TipIndex='Clasic' AND NrIndex=3 GROUP BY NrOp,TotalInreg
  INSERT INTO Rezultate_Finale(NrOp,Operatie,TotalInreg,InregPrel,Timp,Index,TipIndex,NrIndex)
SELECT NrOp, Operatie, TotalInreg, AVG(InregPrel), AVG(Timp), Index, TipIndex, NrIndex FROM Rezultate
WHERE Index='D' AND TipIndex='Clasic' AND NrIndex=4 GROUP BY NrOp,TotalInreg
  INSERT INTO Rezultate_Finale(NrOp,Operatie,TotalInreg,InregPrel,Timp,Index,TipIndex,NrIndex)
SELECT NrOp, Operatie, TotalInreg, AVG(InregPrel), AVG(Timp), Index, TipIndex, NrIndex FROM Rezultate
WHERE Index='D' AND TipIndex='Clasic' AND NrIndex=5 GROUP BY NrOp,TotalInreg
  INSERT INTO Rezultate_Finale(NrOp,Operatie,TotalInreg,InregPrel,Timp,Index,TipIndex,NrIndex)
SELECT NrOp, Operatie, TotalInreg, AVG(InregPrel), AVG(Timp), Index, TipIndex, NrIndex FROM Rezultate
WHERE Index='D' AND TipIndex='Clasic' AND NrIndex=6 GROUP BY NrOp,TotalInreg
  INSERT INTO Rezultate_Finale(NrOp,Operatie,TotalInreg,InregPrel,Timp,Index,TipIndex,NrIndex)
SELECT NrOp, Operatie, TotalInreg, AVG(InregPrel), AVG(Timp), Index, TipIndex, NrIndex FROM Rezultate
WHERE Index='N' GROUP BY NrOp,TotalInreg
if @@error<>0
BEGIN
        ROLLBACK TRANSACTION
        RAISERROR ('Operatiunea nu s-a putut realiza din cauza unei erori!!',16,1) with NOWAIT
END
ELSE
BEGIN
        COMMIT TRANSACTION
        RETURN(0)
END
GO
```

Structurile indexate create asupra tabelelor de date au o structură clasică. De exemplu, asupra tabelei FURNIZORI au fost create următoarele structuri indexate:

```
CREATE INDEX I_FURN_COD ON FURNIZORI(COD_FURN);
CREATE INDEX I_FURN_DEN ON FURNIZORI(DEN_FURN);
CREATE INDEX I_FURN_LOC ON FURNIZORI(LOC_FURN);
CREATE INDEX I_FURN_TARA ON FURNIZORI(TAR_FURN);
CREATE INDEX I_FURN_ADR ON FURNIZORI(ADR_FURN);
CREATE INDEX I_FURN_NR ON FURNIZORI(NR_FISC);
```

Alegerea testelor a pornit şi de la observațiile întâlnite în [Honou 98], [Micro 03-1], [Micro 03-8], [IBM 02]. Toate cele trei produse testate prezintă un comportament similar la acest tip de test. Deoarece Oracle *9i* este cel mai performant dintre produsele analizate, rezultatele prezentate în tabelele şi graficele următoare cuprind datele obținute la testele pe Oracle 9i .

- **Prezentarea rezultatelor pentru fişiere mici sau medii de date. Concluzii.**

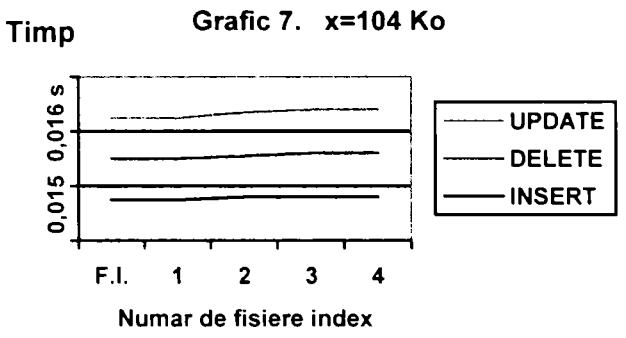
Testele au arătat că pentru fişiere de date de dimensiuni mici sau medii (0 – 50 Mo) timpii de inserare, modificare, ştergere se păstrează relativ aceeaşi indiferent de numărul fişierelor auxiliare de index care există pe lângă fişierul de date şi sunt aproximativ aceeaşi cu cei necesari dacă nu există fişiere index asociate fişierului de date. Tabelul 5 prezintă rezultatele obținute în urma testelor efectuate.

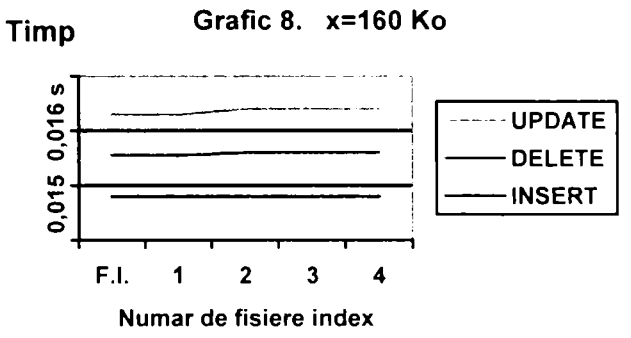
Tabel 5. Timpii de răspuns la efectuarea operațiilor INSERT, UPDATE, DELETE

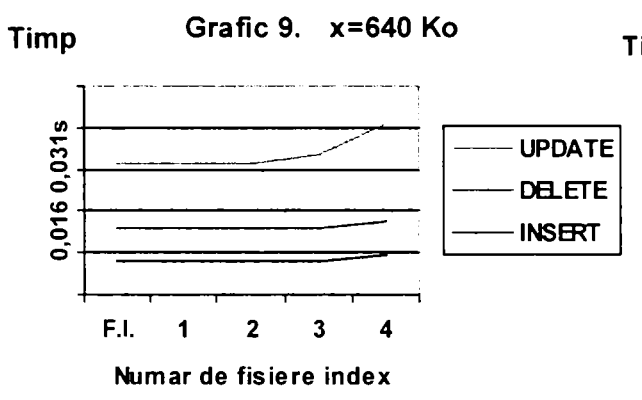
Numar înregistrări	Mărime fişier	Timp de răspuns (s)														
		INSERT (3% tupluri inserate)					UPDATE (3% tupluri modificate)					DELETE (3% tupluri şterse)				
		Fără index	1 index	2 indecşi	3 indecşi	4 indecşi	Fără index	1 index	2 indecşi	3 indecşi	4 indecşi	Fără index	1 index	2 indecşi	3 indecşi	4 indecşi
848	104 Ko	0,015	0,015	0,016	0,016	0,016	0,015	0,015	0,016	0,016	0,016	0,015	0,015	0,015	0,016	0,016
1.272	160 Ko	0,016	0,016	0,016	0,016	0,016	0,015	0,015	0,016	0,015	0,016	0,015	0,015	0,016	0,016	0,016
5.520	640 Ko	0,016	0,016	0,015	0,016	0,019	0,031	0,031	0,031	0,036	0,047	0,016	0,016	0,016	0,016	0,016
395.736	33 Mo	0,031	0,031	0,031	0,032	0,031	0,078	0,078	0,078	0,075	0,079	0,062	0,063	0,061	0,061	0,063

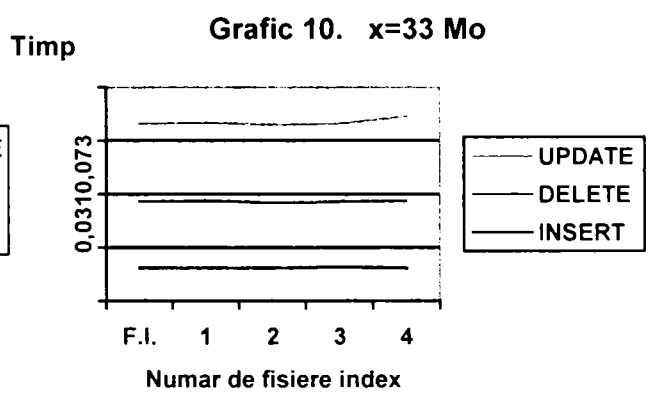
Graficele 7 – 10 prezintă rezultatele din tabelul 5 şi se poate observa că pentru fişiere de date de mărimi mici sau medii (până la 50 Mo) timpii de răspuns la efectuarea operațiilor INSERT, UPDATE, DELETE asupra acestor fişiere sunt aproximativ aceeaşi, fie că fişierul de date are sau nu fişiere index asociate, şi că aceşti timpi nu sunt influențați nici de numărul de fişiere index asociate fişierului de date.

Evoluția timpilor de răspuns la efectuarea operațiilor INSERT, UPDATE, DELETE asupra unui fişier de date de dimensiune x (3% tupluri afectate)









Concluziile care se pot desprinde din tabelul 5 şi din graficele 7-10 sunt următoarele:

- pentru fişiere de date de dimensiuni mici sau medii (sub 50 Mo) timpii de inserare, modificare şi ştergere nu sunt influențați vizibil de existența fişierelor index pe condițiile de căutare din clauza WHERE;
- numărul fişierelor index asociate fişierului de date nu influențează nici el vizibil timpii de efectuare ai operațiilor de actualizare, aceştia fiind aproximativ aceeaşi cu cei obținuți în absența fişierelor index.

- **Prezentarea rezultatelor pentru fişiere mari de date. Concluzii.**

Procesul de testare a fost continuat pentru fişiere de date de dimensiuni mai mari, peste 50 Mo. S-au efectuat aceleaşi tipuri de teste şi pentru fişiere de date a căror dimensiune a variat între 50 Mo şi 500 Mo. Numărul de tupluri prelucrate de operațiile de actualizare s-a situat în jurul valorii de 3% din numărul total de tupluri al relației. Testele alese au ținut cont şi de observațiile din [Thomb 01] şi [Micro 03-5]. S-a observat că, pentru astfel de fişiere,

timpii de execuție ai operațiilor de actualizare sunt influențați de existența fișierelor index, precum și de numărul acestora.

Rezultatele testelor efectuate sunt prezentate în continuare, defalcate pe fiecare tip de operație de actualizare.

- **INSERT**

Numărul de tupluri adăugate la o apelare a blocului SQL s-a situat în jurul valorii de 3% din numărul total de tupluri al relației în care s-a efectuat inserarea. De exemplu, pentru fișierul de date având 1.475.184 înregistrări s-au mai adăugat 44.250 înregistrări. Rezultatele obținute la testele efectuate pentru operațiile de inserare sunt sintetizate în tabelul 6 și sunt prezentate în graficul 11 pentru o mai ușoară interpretare. Creșterea procentuală reprezintă creșterea în procente a timpilor necesari pentru inserarea în fișierul de date a înregistrărilor dorite, în situația introducerii fișierelor index față de situația în care nu există fișier index asociat.

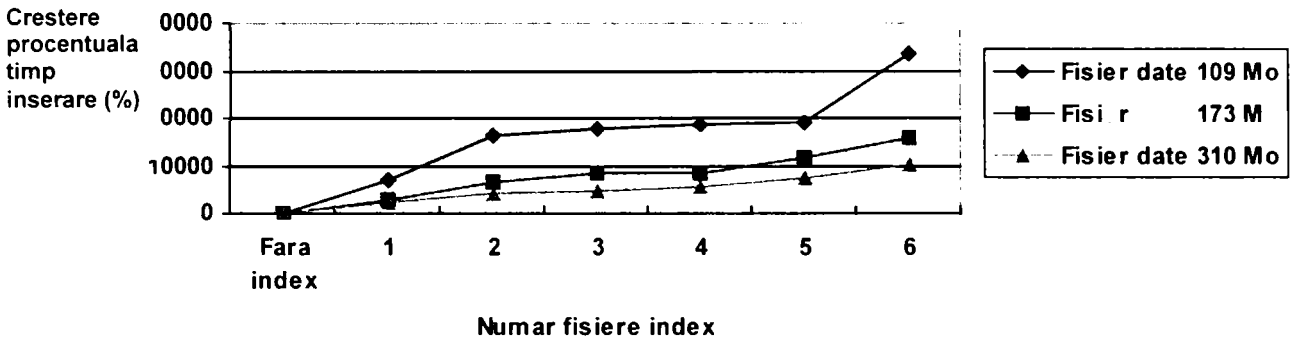
Tabel 6. Timpii de răspuns pentru operația INSERT

Număr înregistrări	Mărime fișier (Mo)	Timp inserare (s) (3% tupluri inserate)							Creștere procentuală (%)						
		Fără index	1 index	2 indecși	3 îndecși	4 indecși	5 indecși	6 indecși	Fără index	1 index	2 indecși	3 indecși	4 indecși	5 indecși	6 indecși
1.475.184	109	0,141	10,063	23,391	25,047	26,531	26,932	47,578	-	7.000	16.500	17.700	18.700	19.000	33.600
2.366.914	173	0,204	5,703	13,156	17,234	17,063	23,469	32,328	-	2.700	6.400	8.400	8.300	11.400	15.700
4.243.680	310	0,609	12,922	26,078	29,844	34,453	45,813	62,313	-	2.000	4.100	4.800	5.500	7.400	10.100

Concluziile care se pot desprinde din tabelul 6 și din graficul 11 sunt următoarele:

- se produce o creștere masivă a timpilor de inserare la apariția primului fișier index, majorarea de cel puțin 20 de ori timpilor de inserare, care poate crește până la 70 de ori;

Grafic 11. INSERT



- apariția celui de al-II-lea fişier index determină o dublare a timpilor de inserare față de situația cu un singur fişier index asociat fişierului de date;
- apariția celui de al-III-lea, al-IV-lea şi al-V-lea fişier index determină o creştere relativ scăzută a timpilor de inserare, undeva sub 10 % din timpul anterior pentru fiecare fişier index suplimentar introdus;
- apariția celui de al-VI-lea fişier index determină o creştere mai mare a timpilor de inserare, situată undeva în jurul valorii de 30-70 % față de situația cu 5 fişiere index asociate fişierului de date;
- cu cât fişierul de date este mai voluminos, deprecierea timpilor de inserare în cazul introducerii fişierelor index față de timpii de inserare în absența indexului este mai mică;
- mărimea cheii de indexare influențează direct proporțional timpii de inserare.

Ca o concluzie finală se poate spune că fişierele de date care sunt des actualizate prin inserarea unor noi înregistrări nu este avantajos să aibă fişiere index asociate. Timpii de inserare cresc considerabil în cazul în care există fişiere index asociate fişierului de date, pentru că şi aceste fişiere trebuie actualizate concomitent cu fişierul de date. Apariția primului şi al celui de-al doilea fişier index determină o depreciere mai mare a timpilor de inserare decât cea datorată apariției celui de-al treilea, al patrulea şi al cincilea fişier index. Poate fi mai avantajos să se elimine fişierele index pe durata efectuării operațiilor de inserare şi apoi să se recreeze aceste fişiere, dacă timpii de creare a lor sunt inferiori celor de inserare în prezența fişierelor index.

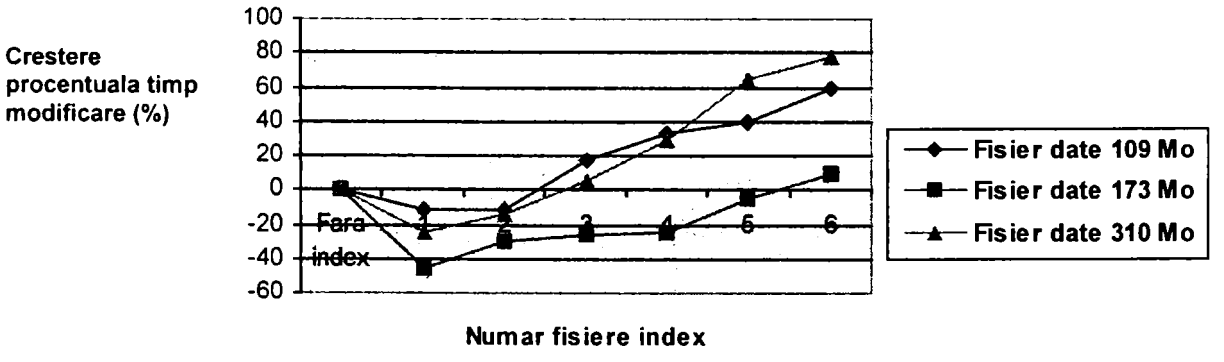
- **UPDATE**

Şi în cazul operațiilor de modificare, numărul tuplurilor prelucrate la o apelare a blocului SQL se situează în jurul valorii de 3% din numărul total de tupluri ale relației. De exemplu, pentru situația fişierului de date de 1.475.184 înregistrări au fost modificate 41.305 înregistrări. Rezultatele obținute la testele efectuate pentru operațiile de modificare sunt sintetizate în tabelul 7 şi sunt prezentate în graficul 12 pentru o mai uşoară interpretare. Creşterea procentuală reprezintă creşterea în procente a timpilor necesari pentru modificarea în fişierul de date a înregistrărilor dorite în situația introducerii fişierelor index față de situația în care nu există fişier index asociat.

Tabel 7. Timpii de răspuns pentru operația UPDATE

Număr înregistrări	Mărime fişier (Mo)	Timp modificare (s) (3% tupluri modificate)							Creştere procentuală (%)						
		Fără index	1 index	2 indecşi	3 indecşi	4 indecşi	5 indecşi	6 indecşi	Fără index	1	2	3	4	5	6
1.475.184	109	13,734	12,156	12.172	16,110	18,312	19,156	21,938	-	-11	-11	17	33	40	60
2.366.914	173	45,531	25.188	31,782	33,610	34,469	42,953	49,89 0	-	-45	-30	-26	-25	-5	10
4.243.680	310	61,843	46,609	53,255	64,813	79,610	101,218	110,172	-	-25	-14	5	29	64	78

Grafic 12. UPDATE



Concluziile care se pot desprinde din tabelul 7 şi din graficul 12 sunt următoarele:

- fişierele index pot îmbunătăți timpii de efectuare ai operațiilor UPDATE dacă regăsirea înregistrărilor care trebuiesc modificate se face folosind o structură indexată;

- existența indexului pe condiția de căutare a clauzei WHERE din blocul UPDATE aduce o îmbunătățire cu 20-50 % a timpilor de execuție ai operației de modificare;
- timpii de execuție ai operației de modificare se păstrează mai buni în cazul existenței indexului pe condiția de căutare chiar şi în cazul existenței a 2 fişiere index asociate fişierului de date, care trebuie ambele actualizate datorită operației de modificare;
- în cazul existenței a 3 fişiere index asociate fişierului de date, care trebuiesc de asemenea să fie modificate, timpii de modificare sunt aproximativ aceeaşi cu cei necesari efectuării operației de modificare în absența fişierelor index;
- pentru situațiile când există 4, 5 sau 6 fişiere index asociate fişierului de date, care trebuie, de asemenea, să fie modificate ca urmare a operației de modificare efectuate asupra fişierului de date, se înregistrează o creştere a timpilor de execuție ai operației de modificare cu cca. 10-20 % pentru fiecare nou fişier index creat;
- pentru situațiile în care înregistrările care se doresc a fi modificate nu există, de fapt, în fişierul de date, existența indexului reduce foarte, foarte mult timpii de răspuns la interogarea SQL formulată. Aceştia se reduc cu aproape 100% în cazul în care există index pe condiția de căutare din clauza WHERE.

Ca o concluzie finală se poate spune că fişierele de date care sunt des actualizate prin modificarea unor înregistrări este avantajos să aibă fişiere index asociate, dacă aceste fişiere index sunt construite pe baza unor chei de indexare care apar ca şi condiții în clauza WHERE a blocului UPDATE. Timpii de modificare scad sesizabil în cazul în care există fişiere index asociate fişierului de date, pentru că înregistrările care trebuie să fie modificate sunt localizate mai rapid prin intermediul indexului. Timpii de modificare se depreciază doar dacă numărul fişierelor index asociate fişierului de date este mai mare decât 3 şi dacă toate aceste fişiere index trebuie să fie actualizate odată cu efectuarea modificării datelor.

- **DELETE**

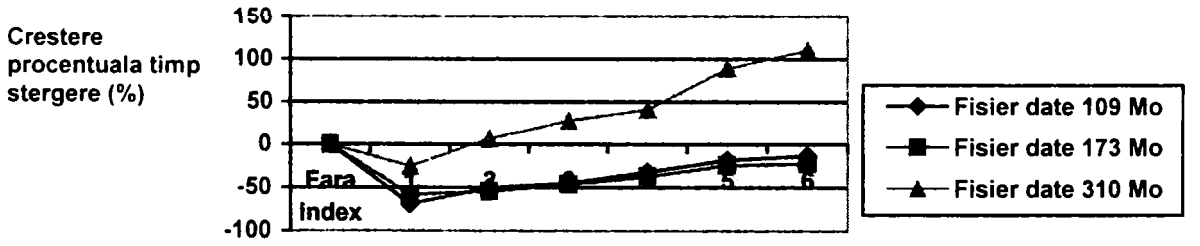
Operațiile de ştergere implică un număr de tupluri care se situează în jurul valorii de 3% din numărul total de tupluri al relației. De exemplu, pentru cazul fişierului de date având 4.243.680 înregistrări s-au eliminat prin operația de ştergere 127.310 înregistrări. Rezultatele obținute la testele efectuate pentru operațiile de ştergere sunt sintetizate în tabelul 8 şi sunt

prezentate în graficul 13 pentru o mai uşoară interpretare. Creşterea procentuală reprezintă creşterea în procente a timpilor necesari pentru ştergerea din fişierul de date a înregistrărilor dorite în situația introducerii fişierelor index față de situația în care nu există fişier index asociat.

Tabel 8. timpii de răspuns pentru operația DELETE

Număr înregistrări	Mărime fişier (Mo)	Timp ştergere (s) (3% tupluri şterse)							Creştere procentuală (%)						
		Fără index	1 index	2 indecşi	3 indecşi	4 indecşi	5 indecşi	6 indecşi	Fără index	1	2	3	4	5	6
1 475 184	109	23,890	7,469	12,156	12,938	15,953	19,672	20,719	-	-69	-50	-45	-32	-18	-13
2 366 914	173	59,625	24,860	27,141	31,922	37,141	45,125	45,953	-	-58	-54	-46	-37	-24	-22
4 243 680	310	47,531	35,485	50,907	60,765	67,078	89,937	99,500	-	-25	7	28	41	89	110

Grafic 13. DELETE



Concluziile care se pot desprinde din tabelul 8 şi din graficul 13 sunt următoarele:

- fişierele index pot îmbunătăți considerabil timpii de efectuare ai operațiilor DELETE dacă regăsirea înregistrărilor care trebuiesc şterse se face folosind o structură indexată;
- existența indexului pe condiția de căutare a clauzei WHERE din blocul DELETE aduce o îmbunătățire cu 25-70 % a timpilor de execuție ai operației de ştergere;
- dacă fişierul de date este de până la 300 Mo, timpii de execuție ai operației de ştergere se păstrează mai buni în cazul existenței indexului pe condiția de căutare chiar şi în cazul existenței a 2-6 fişiere index asociate fişierului de date, care trebuie toate actualizate datorită operației de ştergere;
- dacă fişierul de date are o mărime de peste 300 Mo, timpii de execuție ai operației de ştergere cresc în cazul existenței a mai mult de un fişier index asociat fişierului de date;

- pentru cazul a 2 fişiere index asociate fişierului de date creşterea este totuşi mică, fiind în jur de 10 %;
- pentru cazul a 3-4 fişiere index asociate fişierului de date creşterea este în jur 30-40 %;
- pentru cazul a 5-6 fişiere index asociate fişierului de date creşterea este mare, fiind peste 90 %;

- pentru situațiile când înregistrările care se doreşte a fi şterse nu există, de fapt, în fişierul de date, existența indexului reduce foarte, foarte mult timpii de răspuns la interogarea SQL formulată. Aceştia se reduc cu aproape 100% în cazul în care există index pe condiția de căutare din clauza WHERE.

Ca o concluzie finală se poate spune că fişierele de date care sunt des actualizate prin ştergerea unor înregistrări este avantajos să aibă fişiere index asociate, dacă aceste fişiere index sunt construite pe baza unor chei de indexare care apar ca şi condiții în clauza WHERE a blocului DELETE. Timpii de ştergere scad sesizabil în cazul în care există fişiere index asociate fişierului de date, pentru că înregistrările care trebuie să fie şterse sunt localizate mai rapid prin intermediul indexului. Aceşti timpi de ştergere sunt mai mici chiar şi în cazul în care numărul fişierelor de index asociate fişierului de date este mai mare (5-6), dacă mărimea fişierului de date este sub 300 Mo. Timpii de ştergere se depreciază doar dacă mărimea fişierului de date este de peste 300 Mo şi numărul fişierelor index asociate fişierului de date este mai mare decât 3.

3.3.2.2. Operațiile de actualizare afectează un număr mare de înregistrări din fişierul de date. Concluzii.

Următoarele teste şi-au propus să studieze influența numărului de structuri indexate, asociate unei tabele, asupra timpilor de execuție a operațiilor de actualizare efectuate asupra tabelei de date, atunci când numărul de rânduri prelucrate de operațiile de actualizare este mare, adică se situează în jurul valorii de 15-25% din numărul total de rânduri al tabelei. Tabelele testate conțin un număr relativ mare de rânduri, în jur de 4.000.000. In aceste condiții, mărimea fişierelor de date se situează undeva în jurul valorii de 300 Mo. Această analiză şi-a propus să urmărească comportamentul fişierelor de date mari, specifice unor aplicații de tip *„magazie de date"*, precum cele din domeniul analizei vânzărilor. Fişierelor de

date li se asociază fişiere index capabile să îmbunătățească timpii de răspuns la operațiile de interogare asupra bazei de date [Thomb 01], [Micro 03-5].

Ce se întâmplă, însă, în situația efectuării unor operații de inserare, modificare sau ştergere asupra fişierelor de date mari, dacă numărul înregistrărilor actualizate este mare?

Fişierele index, fiind fişiere auxiliare fişierelor de date, trebuie să fie în permanență o oglindă vie a datelor prezente în fişierele de date, de aceea ele sunt actualizate de fiecare dată când la nivelul fişierului de date se efectuează operații de inserare, modificare sau ştergere. Aceste actualizări suplimentare ale tuturor fişierelor index asociate unui fişier de date pot să atragă după sine o creştere mult prea mare a timpilor de execuție ai operațiilor de inserare, modificare sau ştergere efectuate asupra fişierului de date, mai ales când numărul înregistrărilor din fişierul de date care sunt supuse operației de actualizare este considerabil. Pentru că la nivelul unui data warehouse, în general, operațiile de inserare, modificare şi ştergere implică un volum mare de înregistrări prelucrate, am considerat în cadrul testelor efectuate că numărul acestora se situează undeva între 15-25 % din numărul total de înregistrări din fişier.

Cât de lente devin operațiile de actualizare ale unor astfel de fişiere de date dacă există fişiere index asociate lor?

Este indicat să se păstreze fişierele index pe durata operațiilor de actualizare sau este mai indicat să se elimine aceste fişiere înaintea operațiilor de actualizare urmând ca după efectuarea lor să se recreeze indecşii?

Acestea sunt principalele întrebări la care am căutat răspuns prin aceste teste. Unele dintre testele efectuate au pornit de la caracteristici ale sistemelor testate care au fost identificate în [Lumpk 03], Micro 03-3], [Micro 03-6], [Micro 03-8]. Tabelele folosite în cadrul testelor au fost tot FURNIZORI şi MARFURI. Acestea au tot o structură de tip heap. Asociat fiecărei tabele s-au creat până la 6 structuri indexate clasice, având o structură bazată pe arbori B.

- **Prezentarea testelor**

Operațiile de inserare, modificare, ştergere executate asupra tabelelor de date prelucrează un număr mare de rânduri din tabelă, raportat la mărimea acesteia (între 15-25% din mărimea tabelei). De exemplu, pentru fişierul MARFURI de 310 Mo având 4.234.680 înregistrări s-a prelucrat la o apelare a blocului SQL de actualizare un număr de înregistrări situat între 635.320 înregistrări şi 1.058.634 înregistrări. Numărul efectiv de înregistrări supuse operațiilor de actualizare depinde de valorile specificate în clauza WHERE.

Inserările s-au făcut pe baza unor blocuri SQL [Kochh 00] de forma următoare:

- condiție simplă în clauza WHERE

```
INSERT INTO FURNIZORI
SELECT *
FROM FURNIZORI_NOI
WHERE TAR_FURN='ITL';
```

```
INSERT INTO MARFURI
SELECT *
FROM MARFURI_NOI
WHERE GRUPA='STICKS';
```

- condiție compusă în clauza WHERE

```
INSERT INTO FURNIZORI
SELECT *
FROM FURNIZORI_NOI
WHERE TAR_FURN='ITL'
   OR TAR_FURN='AUT';
```

```
INSERT INTO MARFURI
SELECT *
FROM MARFURI_NOI
WHERE GRUPA='STICKS'
 OR GRUPA='NESTLE';
```

Şi în acest caz testele în implementarea Oracle 9*i* s-au efectuat folosindu-se scripturi PL/SQL care au permis colectarea rezultatelor în tabela REZULTATE. Spre exemplu, pentru inserare in tabela MARFURI folosind condiție compusă în clauza WHERE s-a folosit scriptul MarfuriInsertCompus.sql de mai jos:

```
--MarfuriInsertCompus.sql
DECLARE
 v_NrOp NUMBER(3);
 v_Operatie VARCHAR(200);
 v_Start NUMBER(10,0);
 v_End NUMBER(10,0);
```

```
v_Timp NUMBER(10,0);
v_NrInreg NUMBER(12,0);
v_Index CHAR(1);
v_TipIndex VARCHAR(20);
v_NrIndex NUMBER(2,0);
v_Gupa1 VARCHAR(8);
v_Gupa2 VARCHAR(8);
v_TotalInreg NUMBER(15,0);
CURSOR c_TotalInreg IS
  SELECT count(*)
  FROM Marfuri;
CURSOR c_NrInreg IS
  SELECT count(*)
  FROM Marfuri_Noi
  WHERE Grupa=v_Grupa1 OR Grupa=v_Grupa2;
BEGIN
v_NrOp:=&NrOp;
v_Grupa1:= '&Grupa1';
v_Grupa2:= '&Grupa2';
OPEN c_TotalInreg;
FETCH c_TotalInreg INTO v_TotalInreg;
CLOSE c_TotalInreg;
OPEN c_NrInreg;
FETCH c_NrInreg INTO v_NrInreg;
CLOSE c_NrInreg;
v_Index:= '&Index';
v_TipIndex:= '&TipIndex';
v_NrIndex:=&NrIndex;
v_Operatie:='INSERT INTO Marfuri SELECT * FROM Marfuri_Noi WHERE Grupa= OR Grupa=';
v_Start:=DBMS_UTILITY.GET_TIME;
INSERT INTO Marfuri SELECT * FROM Marfuri_Noi WHERE Grupa=v_Grupa1 OR
Grupa=v_Grupa2;
v_End:=DBMS_UTILITY.GET_TIME;
v_Timp:=(v_End-v_Start)*10;
v_Grupa1:=v_Grupa1||' OR '||v_Grupa2
INSERT INTO Rezultate(NrOp, Operatie, TotalInreg, InregPrel,Timp, Valoare, Index, TipIndex,
      NrIndex) VALUES (v_NrOp, v_Operatie, v_TotalInreg, v_NrInreg, v_Timp, v_Grupa1,
      v_Index, v_TipIndex, v_NrIndex);
COMMIT;
END;
```

Celelalte scripturi folosite pentru inserare pot fi găsite în anexa 1. Tot în anexa 1 pot fi consultate câteva dintre rezultatele obținute în urma testelor efectuate. Testele au vizat atât situațiile în care tabelele FURNIZORI şi MARFURI nu aveau index asociat cât şi situațiile în care numărul indecşilor varia între 1 şi 6.

Pentru implementarea în SQL Server2000 s-au folosit, de asemenea, proceduri stocate scrise în Transact-SQL. De exemplu, procedura pentru cazul de inserare specificat anterior este:

```
/*procedura stocată MarfuriInsertCompus*/
CREATE PROCEDURE dbo.MarfuriInsertCompus
(       @nrOp NUMERIC(18,0), @vGrupa1 VARCHAR(8), @vGrupa2 VARCHAR(8), @index CHAR(1), @tipIndex VARCHAR(20), @nrIndex NUMERIC(2,0)
)
AS
SET NOCOUNT ON
BEGIN TRANSACTION
        DECLARE @op VARCHAR(200),@t1 NUMERIC(18,0),@t2 NUMERIC(18,0),@s1 INT,@s2 INT,@total NUMERIC(18,0),@inreg NUMERIC(18,0),@dif NUMERIC(18,6)
        SET @op = 'INSERT INTO Marfuri SELECT * FROM Marfuri_Noi WHERE Grupa= OR Grupa='
        SELECT @total=count(*) FROM Marfuri
        SELECT @inreg = count(*) FROM Marfuri_Noi WHERE Grupa=@vGrupa1 OR Grupa=@vGrupa2
        SELECT @t1=datepart(ms,getdate()),@s1=datepart(s,getdate())
        INSERT INTO Marfuri SELECT * FROM Marfuri_Noi WHERE Grupa=@vGrupa1 OR Grupa=@vGrupa2
        SELECT @t2=datepart(ms,getdate()),@s2=datepart(s,getdate())
        if @s2>@s1
                SET @dif=(@s2*1000+@t2)-(@s1*1000+@t1)
        else
                SET @dif=@t2-@t1
        SET @vGrupa1=@vGrupa1+' OR '+@vGrupa2
        INSERT Rezultate VALUES
                (@nr,@op,@total,@inreg,@dif,@vGrupa1,@index,@tipIndex,@nrIndex)
if @@error<>0
BEGIN
        ROLLBACK TRANSACTION
        RAISERROR ('Operatiunea nu s-a putut realiza din cauza unei erori!!',16,1) with NOWAIT
END
ELSE
```

```
BEGIN
        COMMIT TRANSACTION
        RETURN(0)
END
GO
```

Celelalte proceduri stocate pentru inserare se găsesc în anexa 1. Tot în anexa 1 se găsesc şi o parte din rezultatele obținute în urma testelor efectuate. Şi în acest caz s-au efectuat teste atât pentru situația fără index cât şi cu număr variabil de indecşi asociați (1-6).

Modificările s-au efectuat pe baza unor blocuri SQL [Kochh 00]] asemănătoare celor de mai jos:

```
UPDATE FURNIZORI
SET TAR_FURN='ITALIA'
WHERE TAR_FURN='ITL';

UPDATE FURNIZORI
SET TAR_FURN='ITALIA'
WHERE LOC_FURN='ROMA';
```

```
UPDATE MARFURI
SET PRET_ACHIZ=PRET_ACHIZ*1.1
WHERE GRUPA='RAUCH';

UPDATE MARFURI
SET PRET_ACHIZ=PRET_ACHIZ*1.1
WHERE PRET_ACHIZ=59850;
```

Şi în acest caz pentru implementarea în Oracle 9*i* s-au folosit scripturi PL/SQL care colectează rezultatele în tabela REZULTATE. Primele două blocuri de modificare au fost executate rulând scripturile FurnizoriModificare.sql şi MarfuriModificare.sql, iar următoarele două tipuri de modificări au fost executate cu ajutorul scripturilor FurnizoriModificare1.sql şi MarfuriModificare1.sql de mai jos:

```
--FurnizoriModificare1.sql
DECLARE
 v_NrOp NUMBER(3);
 v_Operatie VARCHAR(200);
 v_Start NUMBER(10,0);
 v_End NUMBER(10,0);
 v_Timp NUMBER(10,0);
 v_NrInreg NUMBER(12,0);
 v_Tar_Furn Furnizori.Tar_Furn%TYPE;
 v_Loc_Furn Furnizori.Loc_Furn%TYPE;
 v_Index CHAR(1);
 v_TipIndex VARCHAR(20);
```

```
 v_NrIndex NUMBER(2,0);
 v_TotalInreg NUMBER(15,0);
 CURSOR c_TotalInreg IS
  SELECT count(*)
  FROM Furnizori;
CURSOR c_NrInreg IS
  SELECT count(*)
  FROM Furnizori
  WHERE Loc_Furn=v_Loc_Furn;
BEGIN
 v_NrOp:=&NrOp;
 OPEN c_TotalInreg;
 FETCH c_TotalInreg INTO v_TotalInreg;
 CLOSE c_TotalInreg;
 v_Tar_Furn:= '&Tar_Furn';
 v_Loc_Furn:= '&Loc_Furn';
 v_Index:= '&Index';
 v_TipIndex:= '&TipIndex';
 v_NrIndex:=&NrIndex;
 OPEN c_NrInreg
 FETCH c_NrInreg INTO v_NrInreg;
 CLOSE c_NrInreg;
 v_Operatie:='UPDATE Furnizori SET Tar_Furn= WHERE Loc_Furn=';
 v_Start:=DBMS_UTILITY.GET_TIME;
 UPDATE Furnizori SET Tar_Furn=v_Tar_Furn WHERE Loc_Furn=v_Loc_Furn;
 v_End:=DBMS_UTILITY.GET_TIME;
 v_Timp:=(v_End-v_Start)*10;
 INSERT INTO Rezultate(NrOp, Operatie, TotalInreg, InregPrel,Timp, Valoare, Index, TipIndex,
       NrIndex) VALUES (v_NrOp, v_Operatie, v_TotalInreg, v_NrInreg, v_Timp,
       v_Loc_Furn, v_Index, v_TipIndex, v_NrIndex);
 COMMIT;
END;
```

La rularea scriptului se furnizează ca parametrii numărul operației, localitatea furnizorului pentru care vor fi operate modificările, noua valoare pentru țara furnizorului, dacă există sau nu index asociat fişierului de date, tipul indexului şi numărul de fişiere index asociate fişierului de date. Pentru situațiile cu 2-6 fişiere index, scriptul a fost modificat astfel încât modificările făcute asupra fişierului de date să implice toate fişierele index. Rezultatele testelor sunt colectate în tabela REZULTATE şi o parte dintre ele pot fi consultate în anexa 1.

```
--MarfuriModificare1.sql
DECLARE
 v_NrOp NUMBER(3);
 v_Operatie VARCHAR(200);
 v_Start NUMBER(10,0);
 v_End NUMBER(10,0);
 v_Timp NUMBER(10,0);
 v_NrInreg NUMBER(12,0);
 v_Pret_Achiz Marfuri.Pret_achiz%TYPE;
 v_Index CHAR(1);
 v_TipIndex VARCHAR(20);
 v_NrIndex NUMBER(2,0);
 v_TotalInreg NUMBER(15,0);
 CURSOR c_TotalInreg IS
   SELECT count(*)
   FROM Marfuri;
CURSOR c_NrInreg IS
   SELECT count(*)
   FROM Marfuri
   WHERE Pret_Achiz=v_Pret_Achiz;
BEGIN
v_NrOp:=&NrOp;
OPEN c_TotalInreg;
FETCH c_TotalInreg INTO v_TotalInreg;
CLOSE c_TotalInreg;
v_Pret_Achiz:=&Pret_Achiz;
v_Index:= '&Index';
v_TipIndex:= '&TipIndex';
v_NrIndex:=&NrIndex;
OPEN c_NrInreg
FETCH c_NrInreg INTO v_NrInreg;
CLOSE c_NrInreg;
v_Operatie:='UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Pret_Achiz=';
v_Start:=DBMS_UTILITY.GET_TIME;
UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Pret_Achiz=v_Pret_Achiz;
v_End:=DBMS_UTILITY.GET_TIME;
v_Timp:=(v_End-v_Start)*10;
INSERT INTO Rezultate(NrOp, Operatie, TotalInreg, InregPrel,Timp, Valoare, Index, TipIndex,
      NrIndex) VALUES (v_NrOp, v_Operatie, v_TotalInreg, v_NrInreg, v_Timp,
      v_Pret_Achiz, v_Index, v_TipIndex, v_NrIndex);
```

```
COMMIT;
END;
```

Şi în acest caz scriptul a fost modificat în situația a 2-6 fişiere index asociate fişierului de date, astfel încât toate aceste fişiere index să fie actualizate ca urmare a modificărilor făcute în fişierul de date. Rezultatele testelor sunt colectate în tabela REZULTATE şi o parte a acestora pot fi consultate în anexa 1.

În cazul SQL Server2000 pentru aceste teste s-au folosit proceduri stocate. Primele două operații de actualizare au fost executate cu ajutorul procedurilor FurnizoriModificare şi MarfuriModificare, iar următoarele două prin executarea procedurilor FurnizoriModificare1 şi MarfuriModificare1 de mai jos:

```
/*procedura stocată FurnizoriModificare1*/
CREATE PROCEDURE dbo.FurnizoriModificare1
(       @nrOp NUMERIC(18,0), @vTar_Furn VARCHAR(15), @vLoc_Furn VARCHAR(20),
@index CHAR(1), @tipIndex VARCHAR(20), @nrIndex NUMERIC(2,0)
)
AS
SET NOCOUNT ON
BEGIN TRANSACTION
        DECLARE @op VARCHAR(200),@t1 NUMERIC(18,0),@t2 NUMERIC(18,0),@s1
INT,@s2 INT,@total NUMERIC(18,0),@inreg NUMERIC(18,0),@dif NUMERIC(18,6)
        SET @op = 'UPDATE Furnizori SET Tar_Furn= WHERE Loc_Furn='
        SELECT @total=count(*) FROM Furnizori
        SELECT @inreg = count(*) FROM Furnizori WHERE Loc_Furn=@vLoc_Furn
        SELECT @t1=datepart(ms,getdate()),@s1=datepart(s,getdate())
        UPDATE Furnizori SET Tar_Furn=@vTar_Furn
                WHERE Loc_Furn=@vLoc_Furn
        SELECT @t2=datepart(ms,getdate()),@s2=datepart(s,getdate())
        if @s2>@s1
                SET @dif=(@s2*1000+@t2)-(@s1*1000+@t1)
        else
                SET @dif=@t2-@t1
        INSERT Rezultate VALUES
                (@nr,@op,@total,@inreg,@dif,@vLoc_Furn,@index,@tipIndex,@nrIndex)
if @@error<>0
BEGIN
        ROLLBACK TRANSACTION
```

```
        RAISERROR ('Operatiunea nu s-a putut realiza din cauza unei erori!!',16,1) with NOWAIT
    END
    ELSE
    BEGIN
        COMMIT TRANSACTION
        RETURN(0)
    END
    GO
```

Procedura FurnizoriModificare1 a fost modificată în cazul a 2-6 fişiere index astfel încât toate fişierele index asociate fişierului de date să fie afectate de modificările făcute asupra acestuia. Rezultatele testelor sunt colectate în tabela REZULTATE. O parte a acestor rezultate pot fi consultate în anexa 1.

```
    /*procedura stocată MarfuriModificare1*/
    CREATE PROCEDURE dbo.MarfuriModificare1
    (   @nrOp NUMERIC(18,0), @vPret_Achiz NUMBER(14,2), @index CHAR(1), @tipIndex
VARCHAR(20), @nrIndex NUMERIC(2,0)
    )
    AS
    SET NOCOUNT ON
    BEGIN TRANSACTION
        DECLARE @op VARCHAR(200),@t1 NUMERIC(18,0),@t2 NUMERIC(18,0),@s1
INT,@s2 INT,@total NUMERIC(18,0),@inreg NUMERIC(18,0),@dif NUMERIC(18,6)
        SET @op = 'UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Pret_Achiz='
        SELECT @total=count(*) FROM Marfuri
        SELECT @inreg = count(*) FROM Marfuri WHERE Pret_Achiz=@vPret_Achiz
        SELECT @t1=datepart(ms,getdate()),@s1=datepart(s,getdate())
        UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Pret_Achiz=@vPret_Achiz
        SELECT @t2=datepart(ms,getdate()),@s2=datepart(s,getdate())
        if @s2>@s1
            SET @dif=(@s2*1000+@t2)-(@s1*1000+@t1)
        else
            SET @dif=@t2-@t1
        INSERT Rezultate VALUES
            (@nr,@op,@total,@inreg,@dif,@vPret_Achiz,@index,@tipIndex,@nrIndex)
    if @@error<>0
    BEGIN
        ROLLBACK TRANSACTION
```

```
        RAISERROR ('Operatiunea nu s-a putut realiza din cauza unei erori!!',16,1) with NOWAIT
END
ELSE
BEGIN
        COMMIT TRANSACTION
        RETURN(0)
END
GO
```

Şi această procedură a fost modificată în situația în care fişierul de date avea asociate 2-6 fişiere index, astfel încât toate aceste fişiere index să fie afectate de modificările operate în fişierul de date. Rezultatele testelor au fost colectate în tabela REZULTATE şi câteva dintre ele pot fi observate în anexa 1.

Ştergerile s-au efectuat pe baza unor blocuri SQL [Kochh 00] de forma următoare:

```
DELETE                          DELETE
FROM FURNIZORI                  FROM MARFURI
WHERE TAR_FURN='AUSTRIA';       WHERE GRUPA='TOBLERON';

DELETE                          DELETE
FROM FURNIZORI                  FROM MARFURI
WHERE LOC_FURN LIKE 'C%';       WHERE DENP LIKE 'CAFEA%';
```

Implementarea Oracle 9*i* a folosit pentru executarea primelor două operații de ştergere scripturile FurnizoriStergere.sql şi MarfuriStergere.sql. Celelalte două operații de ştergere au fost executate prin intermediul scripturilor PL/SQL FurnizoriStergere1.sql şi MarfuriStergere1.sql de mai jos:

```
--FurnizoriStergere1.sql
DECLARE
 v_NrOp NUMBER(3);
 v_Operatie VARCHAR(200);
 v_Start NUMBER(10,0);
 v_End NUMBER(10,0);
 v_Timp NUMBER(10,0);
 v_NrInreg NUMBER(12,0);
 v_Loc_Furn Furnizori.Loc_Furn%TYPE;
```

```
 v_Index CHAR(1);
 v_TipIndex VARCHAR(20);
 v_NrIndex NUMBER(2,0);
 v_TotalInreg NUMBER(15,0);
 CURSOR c_TotalInreg IS
   SELECT count(*)
   FROM Furnizori;
CURSOR c_NrInreg IS
   SELECT count(*)
   FROM Furnizori
   WHERE Loc_Furn LIKE v_Loc_Furn;
BEGIN
 v_NrOp:=&NrOp;
 OPEN c_TotalInreg;
 FETCH c_TotalInreg INTO v_TotalInreg;
 CLOSE c_TotalInreg;
 v_Loc_Furn:= '&Loc_Furn';
 v_Index:= '&Index';
 v_TipIndex:= '&TipIndex';
 v_NrIndex:=&NrIndex;
 v_Operatie:='DELETE FROM Furnizori WHERE Loc_Furn LIKE';
 OPEN c_NrInreg;
 FETCH c_NrInreg INTO v_NrInreg;
 CLOSE c_NrInreg;
 v_Start:=DBMS_UTILITY.GET_TIME;
 DELETE FROM Furnizori WHERE Loc_Furn LIKE v_Loc_Furn;
 v_End:=DBMS_UTILITY.GET_TIME;
 v_Timp:=(v_End-v_Start)*10;
 INSERT INTO Rezultate(NrOp, Operatie, TotalInreg, InregPrel,Timp, Valoare, Index, TipIndex,
        NrIndex) VALUES (v_NrOp, v_Operatie, v_TotalInreg, v_NrInreg, v_Timp,
        v_Loc_Furn, v_Index, v_TipIndex, v_NrIndex);
 COMMIT;
END;
```

La rularea scriptului se furnizează ca parametrii numărul operației, localitatea pentru furnizorii ce vor fi eliminați, dacă există sau nu index asociat fişierului de date, tipul indexului şi numărul de fişiere index asociate fişierului de date (1-6). Rezultatele testelor sunt colectate în tabela REZULTATE şi o parte dintre ele pot fi consultate în anexa 1.

```
--MarfuriStergere1.sql
DECLARE
 v_NrOp NUMBER(3);
 v_Operatie VARCHAR(200);
 v_Start NUMBER(10,0);
 v_End NUMBER(10,0);
 v_Timp NUMBER(10,0);
 v_NrInreg NUMBER(12,0);
 v_DenP Marfuri.DenP%TYPE;
 v_Index CHAR(1);
 v_TipIndex VARCHAR(20);
 v_NrIndex NUMBER(2,0);
 v_TotalInreg NUMBER(15,0);
 CURSOR c_TotalInreg IS
   SELECT count(*)
   FROM Marfuri;
CURSOR c_NrInreg IS
   SELECT count(*)
   FROM Marfuri
   WHERE DenP LIKE v_DenP;
BEGIN
 v_NrOp:=&NrOp;
 OPEN c_TotalInreg;
 FETCH c_TotalInreg INTO v_TotalInreg;
 CLOSE c_TotalInreg;
 v_DenP:= '&DenP';
 v_Index:= '&Index';
 v_TipIndex:= '&TipIndex';
 v_NrIndex:=&NrIndex;
 v_Operatie:='DELETE FROM Marfuri WHERE DenP LIKE ';
 OPEN c_NrInreg;
 FETCH c_NrInreg INTO v_NrInreg;
 CLOSE c_NrInreg;
 v_Start:=DBMS_UTILITY.GET_TIME;
 DELETE FROM Marfuri WHERE DenP LIKE v_DenP;
 v_End:=DBMS_UTILITY.GET_TIME;
 v_Timp:=(v_End-v_Start)*10;
 INSERT INTO Rezultate(NrOp, Operatie, TotalInreg, InregPrel,Timp, Valoare, Index, TipIndex,
        NrIndex) VALUES (v_NrOp, v_Operatie, v_TotalInreg, v_NrInreg, v_Timp,
        v_DenP, v_Index, v_TipIndex, v_NrIndex);
```

```
COMMIT;
END;
```

La rularea scriptului se furnizează ca parametrii numărul operației, denumirea mărfii care trebuie eliminată, dacă există sau nu index asociat fişierului de date, tipul indexului şi numărul de fişiere index asociate fişierului de date (1-6). Rezultatele testelor sunt colectate în tabela REZULTATE şi o parte dintre ele pot fi consultate în anexa 1.

În cazul testelor efectuate pe SQL Server2000 s-au folosit proceduri stocate. Primele două operații de ştergere au fost executate prin intermediul procedurilor FurnizoriStergere şi MarfuriStergere, iar următoarele două prin executarea procedurilor FurnizoriStergere1 şi MarfuriStergere1 de mai jos:

```
/*procedura stocată FurnizoriStergere1*/
CREATE PROCEDURE dbo.FurnizoriStergere1
(       @nrOp NUMERIC(18,0), @vLoc_Furn VARCHAR(15), @index CHAR(1), @tipIndex VARCHAR(20), @nrIndex NUMERIC(2,0)
)
AS
SET NOCOUNT ON
BEGIN TRANSACTION
        DECLARE @op VARCHAR(200),@t1 NUMERIC(18,0),@t2 NUMERIC(18,0),@s1 INT,@s2 INT,@total NUMERIC(18,0),@inreg NUMERIC(18,0),@dif NUMERIC(18,6)
        SET @op = 'DELETE FROM Furnizori WHERE Loc_Furn LIKE '
        SELECT @total=count(*) FROM Furnizori
        SELECT @inreg = count(*)  FROM  Furnizori WHERE Loc_Furn LIKE @vLoc_Furn
        SELECT @t1=datepart(ms,getdate()),@s1=datepart(s,getdate())
        DELETE FROM Furnizori WHERE Loc_Furn=@vLoc_Furn
        SELECT @t2=datepart(ms,getdate()),@s2=datepart(s,getdate())
        if @s2>@s1
                SET @dif=(@s2*1000+@t2)-(@s1*1000+@t1)
        else
                SET @dif=@t2-@t1
        INSERT Rezultate VALUES
                (@nr,@op,@total,@inreg,@dif,@vLoc_Furn,@index,@tipIndex,@nrIndex)
if @@error<>0
BEGIN
        ROLLBACK TRANSACTION
        RAISERROR ('Operatiunea nu s-a putut realiza din cauza unei erori!!',16,1) with NOWAIT
```

```
END
ELSE
BEGIN
        COMMIT TRANSACTION
        RETURN(0)
END
GO
```

Procedura FurnizoriStergere1 este executată furnizându-se ca parametrii numărul operației, valoarea pentru localitatea furnizorului pe baza căreia se vor efectua ştergerile, dacă există sau nu index asociat fişierului de date, tipul indexului şi numărul de fişiere index asociate. Rezultatele testelor sunt colectate în tabela REZULTATE. O parte a acestor rezultate pot fi consultate în anexa 1.

```
/*procedura stocată MarfuriStergere1*/
CREATE PROCEDURE dbo.MarfuriStergere1
(       @nrOp NUMERIC(18,0), @vDenP VARCHAR(8), @index CHAR(1), @tipIndex VARCHAR(20), @nrIndex NUMERIC(2,0)
)
AS
SET NOCOUNT ON
BEGIN TRANSACTION
        DECLARE @op VARCHAR(200),@t1 NUMERIC(18,0),@t2 NUMERIC(18,0),@s1 INT,@s2 INT,@total NUMERIC(18,0),@inreg NUMERIC(18,0),@dif NUMERIC(18,6)
        SET @op = 'DELETE FROM Marfuri WHERE DenP LIKE '
        SELECT @total=count(*) FROM Marfuri
        SELECT @inreg = count(*)  FROM  Marfuri WHERE DenP LIKE @vDenP
        SELECT @t1=datepart(ms,getdate()),@s1=datepart(s,getdate())
        DELETE FROM Marfuri WHERE DenP LIKE @vDenP
        SELECT @t2=datepart(ms,getdate()),@s2=datepart(s,getdate())
        if @s2>@s1
                SET @dif=(@s2*1000+@t2)-(@s1*1000+@t1)
        else
                SET @dif=@t2-@t1
        INSERT Rezultate VALUES
                (@nr,@op,@total,@inreg,@dif,@vDenP,@index,@tipIndex,@nrIndex)
if @@error<>0
BEGIN
        ROLLBACK TRANSACTION
```

```
        RAISERROR ('Operatiunea nu s-a putut realiza din cauza unei erori!!',16,1) with NOWAIT
END
ELSE
BEGIN
        COMMIT TRANSACTION
        RETURN(0)
END
GO
```

Procedura MarfuriStergere1 se execută furnizându-se ca parametrii numărul operației, valoarea pentru denumirea mărfii pe baza căreia se vor efectua ştergerile, dacă există sau nu index asociat fişierului de date, tipul indexului şi numărul de fişiere index asociate. Rezultatele testelor sunt colectate în tabela REZULTATE. O parte a acestor rezultate pot fi consultate în anexa 1.

Deoarece în tabela REZULTATE au fost colectați timpii de execuție obținuți pentru execuții repetate ale aceleiaşi operații de actualizare pentru fiecare caz distinct de valoare din clauza WHERE, aceste date au fost sintetizate în tabela REZULTATE_FINALE. Inregistrările din tabela REZULTATE_FINALE au fost obținute prin rularea scripului RezultateFinale1.sql sau prin executarea procedurii stocate RezultateFinale1.

Structurile indexate create asupra tabelelor de date au o structură clasică. De exemplu, asupra tabelei MARFURI au fost create următoarele structuri indexate:

```
CREATE INDEX I_MARF_COD ON MARFURI(CODP);
CREATE INDEX I_MARF_DEN ON MARFURI(DENP);
CREATE INDEX I_MARF_GRUPA ON MARFURI(GRUPA);
CREATE INDEX I_MARF_FURN ON MARFURI(COD_FURN);
CREATE INDEX I_MARF_UM ON MARFURI(UM);
CREATE INDEX I_MARF_PRET ON MARFURI(PRET_ACHIZ);
```

Comportamentul observat la acest tip de teste este similar pentru cele două produse testate, Oracle 9*i* şi SQL Server2000. Deoarece Oracle *9i* este cel mai performant dintre cele două produse, rezultatele prezentate în tabelele şi graficele următoare cuprind datele obținute la testele pe Oracle 9i .

- **Prezentarea rezultatelor. Concluzii.**

Procesul de testare s-a efectuat asupra unor fişiere de date de dimensiuni mari, având în jur de 4.000.000 de înregistrări, adică dimensiuni de peste 300 Mo. Operațiile de actualizare efectuate la nivelul acestor fişiere de date prelucrează în jur de 15-25 % din înregistrările fişierului. O parte a rezultatelor obținute la aceste teste pot fi consultate în anexa 1. S-a observat că, pentru astfel de fişiere, timpii de execuție ai operațiilor de actualizare sunt influențați de existența fişierelor index, precum şi de numărul acestora.

Rezultatele testelor efectuate sunt prezentate în continuare, defalcate pe fiecare tip de operație de actualizare.

- **INSERT**

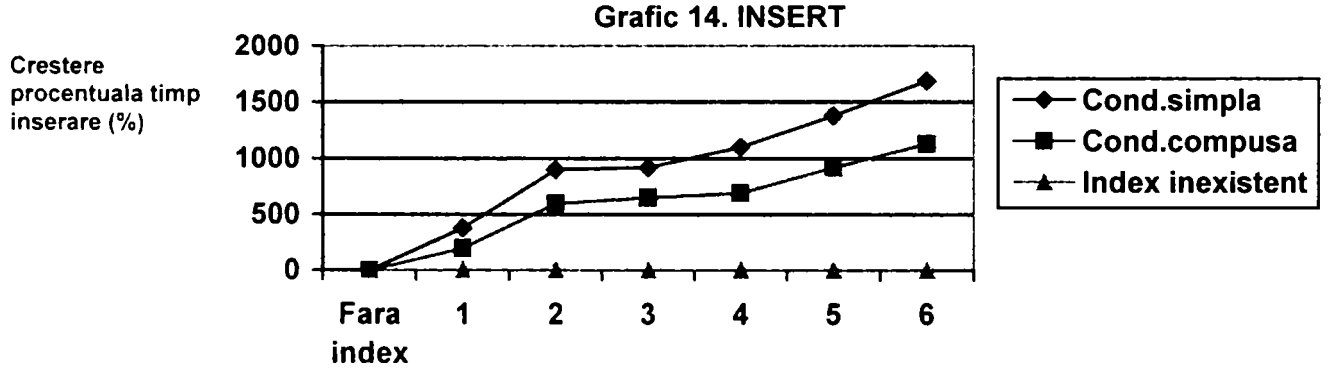
Operațiile de inserare presupun inserarea în fişierul de date a unor blocuri de înregistrări preluate dintr-un alt fişier de date, care are asociat un fişier index construit pe baza unei chei de indexare care apare ca şi condiție de căutare în clauza WHERE a blocului de inserare. Fişierului de date în care se inserează înregistrările i se asociază un număr variabil de fişiere index. Rezultatele obținute la testele efectuate pentru operațiile de inserare sunt sintetizate în tabelul 9 şi sunt prezentate în graficele 14, 15 şi 16 pentru o mai uşoară interpretare. Creşterea procentuală reprezintă creşterea în procente a timpilor necesari pentru inserarea în fişierul de date a înregistrărilor dorite în situația introducerii fişierelor index față de situația în care nu există fişier index asociat. Graficele 15 şi 16 prezintă comparativ timpii de inserare în situația în care există fişiere index asociate fişierului de date în momentul efectuării operației de inserare față de situația în care aceste fişiere index au fost eliminate şi sunt recreate ulterior.

Tabel 9. Timpii de răspuns pentru operația INSERT

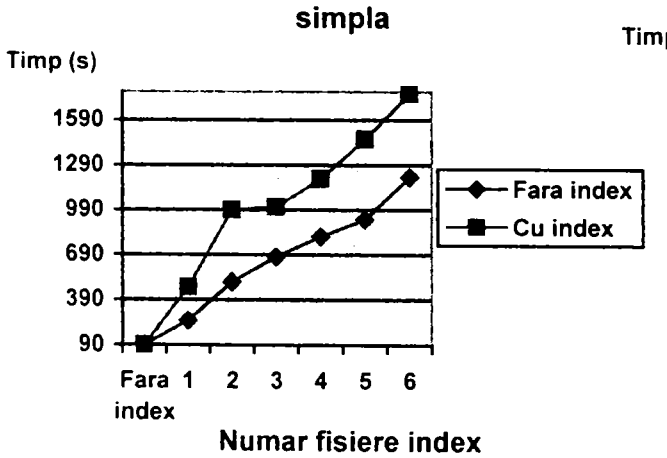
Condiție în clauza WHERE	Timp inserare (hh:mm:ss) (15-25% tupluri inserate)							Creştere procentuală (%)						
	Fără index	1 index	2 indecşi	3 indecşi	4 indecşi	5 indecşi	6 indecşi	Fără index	1	2	3	4	5	6
1. Simplă	00:01:33	00:07:56	00:16:31	00:16:50	00:19:59	00:24:24	00:29:29	-	380	900	920	1.100	1.380	1.690
2. Compusă	00:03:05	00:09:10	00:21:49	00:23:06	00:24:21	00:31:19	00:37:51	-	200	600	650	690	920	1.130

Timp DROP INDEX+ CREATE INDEX	-	00:02:41	00:06:58	00:09:45	00:12:00	00:13:54	00:18:39	Cât % din timpul de inserare cu index reprezintă timpul de inserare fără index					
Timp INSERT+ DROP INDEX+ CREATE INDEX	1	00:04:14	00:08:31	00:11:18	00:13:33	00:15:27	00:20:12	1.	53	51	67	67	63	68
	2.	00:05:46	00:10:03	00:12:50	00:15:05	00:16:59	00:21:44	2.	62	46	55	62	59	57

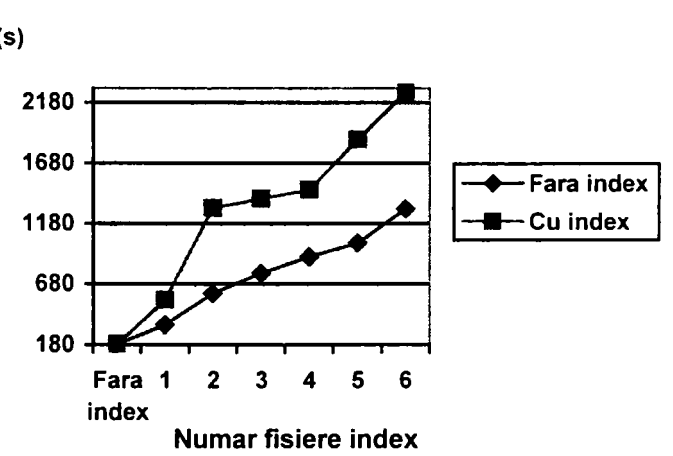
Grafic 14. INSERT



Grafic 15. INSERT Conditie simpla



Grafic 16. INSERT Conditie compusa



Concluziile care se pot desprinde din tabelul 9 şi din graficul 14 sunt următoarele:

- apariția primului fişier index determină creşterea timpilor de execuție ai operației de inserare. Aceştia cel puțin se dublează, dar se pot înregistra creşteri de până la 4-5 ori;
- apariția celui de al-II-lea fişier index determină o dublare a timpilor de inserare față de situația cu un singur fişier index asociat fişierului de date;

- apariția celui de al-III-lea şi al-IV-lea fişier index determină o creştere relativ scăzută a timpilor de inserare, undeva între 0-10 % din timpul anterior pentru fiecare fişier index suplimentar introdus;
- apariția celui de al-V-lea şi celui de al-VI-lea fişier index determină o creştere mai mare a timpilor de inserare, situată undeva în jurul valorii de 25 % din timpul anterior pentru fiecare fişier index suplimentar introdus;
- dacă condiția de căutare din clauza WHERE este compusă, timpii de execuție ai operației de inserare sunt mai mari comparativ cu cei pentru condiție simplă;
- creşterile timpilor de execuție ai operației INSERT în prezența fişierelor index sunt mai mici dacă condiția de căutare din clauza WHERE este compusă, în raport cu creşterile înregistrate dacă condiția de căutare este simplă, dar sunt totuşi semnificativ mai mari decât timpul necesar efectuării operației de inserare în absența fişierelor index;
- în cazul operației INSERT este mai avantajos să se elimine indecşii pe durata operației de inserare urmând a fi recreați după terminarea operației. Se obține astfel o reducere de aproximativ 30-40% a timpilor necesari efectuării operației de inserare, care persistă indiferent de tipul condiției din clauza WHERE, simplă sau compusă, şi de numărul de fişiere index asociate fişierului de date.

Ca o concluzie finală se poate spune că fişierele de date care sunt des actualizate prin inserarea unor noi înregistrări nu este avantajos să aibă fişiere index asociate. Timpii de inserare cresc considerabil în cazul în care există fişiere index asociate fişierului de date, pentru că şi aceste fişiere trebuie actualizate concomitent cu fişierul de date. Apariția primului şi al celui de-al doilea fişier index determină o depreciere mai mare a timpilor de inserare decât cea datorată apariției celui de-al treilea, al patrulea, al cincilea şi al şaselea fişier index. In general este mai avantajos să se elimine, dacă este posibil, fişierele index pe durata efectuării operațiilor de inserare şi apoi să se recreeze aceste fişiere. Se obține în această manieră o reducere de cca. 30-40% a timpilor necesari efectuării operației de inserare față de situația în care aceste fişiere index există şi trebuie şi ele supuse prelucrărilor cauzate de operațiile de inserare.

- **UPDATE**

Operațiile de modificare s-au aplicat asupra unor fişiere de date având în jur de 4.000.000 înregistrări şi s-au modificat la o apelare a blocului SQL un număr de înregistrări ce a variat între 635.320 înregistrări şi 1.058.634 înregistrări, adică în jur de 15-25% din mărimea fişierului de date. La nivelul operațiilor de modificare au fost analizate următoarele trei situații care pot să apară în exploatarea tabelei de date:

- nu există index pe condiția de căutare din clauza WHERE a blocului UPDATE, dar operația de actualizare implică modificări la nivelul coloanelor tabelei care apar în cheile de indexare ale unor indecşi asociați tabelei de date, prin urmare aceşti indecşi trebuiesc refăcuți de către sistem ca urmare a executării operației UPDATE;
- există index pe condiția de căutare din clauza WHERE a blocului UPDATE, dar operația de actualizare nu implică modificări la nivelul acestui index, ci numai la nivelul coloanelor tabelei care apar în cheile de indexare ale altor indecşi asociați tabelei de date, prin urmare numai aceşti indecşi trebuiesc refăcuți de către sistem ca urmare a executării operației UPDATE;
- există index pe condiția de căutare din clauza WHERE a blocului UPDATE şi operația de actualizare implică atât modificări la nivelul coloanelor tabelei care apar în cheia de indexare a acestui index cât şi în cheile de indexare ale altor indecşi asociați tabelei de date, prin urmare toți aceşti indecşi trebuiesc refăcuți de către sistem ca urmare a executării operației UPDATE.

Rezultatele obținute la testele efectuate pentru operațiile de modificare sunt sintetizate în tabelul 10 şi sunt prezentate în graficul 17, 18, 19 şi 20 pentru o mai uşoară interpretare. Creşterea procentuală reprezintă creşterea în procente a timpilor necesari pentru modificarea în fişierul de date a înregistrărilor dorite în situația introducerii fişierelor index față de situația în care nu există fişier index asociat. Graficele 18, 19 şi 20 prezintă comparativ timpii de modificare în situația în care există fişiere index asociate fişierului de date în momentul efectuării operației de modificare față de situația în care aceste fişiere index au fost eliminate şi sunt recreate ulterior pentru toate cele trei cazuri analizate.

Tabel 10. Timpii de răspuns pentru operația UPDATE

Condiție	Timp inserare (hh:mm:ss) (15-25% tupluri modificate)	Creştere procentuală (%)

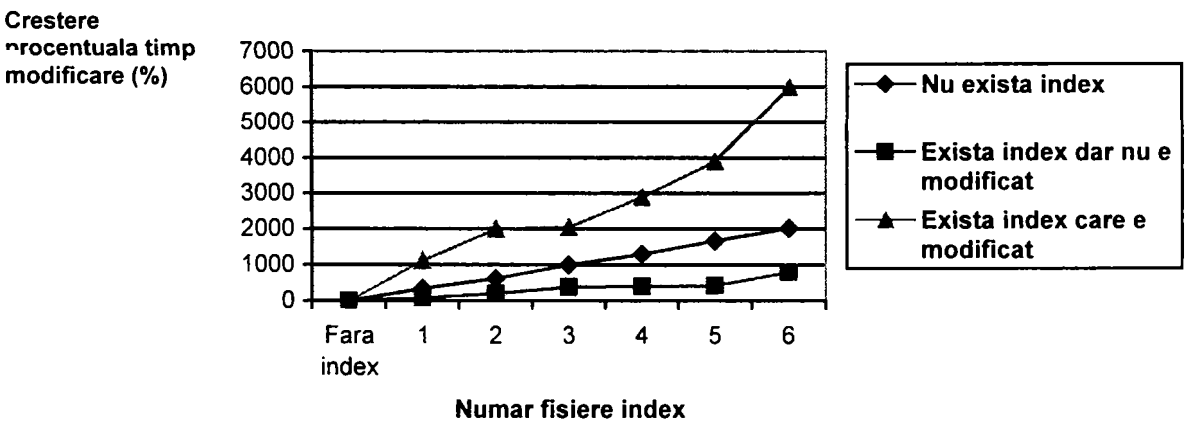
	Fără index	1 index	2 indecşi	3 indecşi	4 indecşi	5 indecşi	6 indecşi	Fără index	1	2	3	4	5	6
Condiție1	00:02:21	00:10:28	00:17:02	00:25:41	00:32:54	00:41:41	00:49:51	-	345	625	1.000	1.300	1.680	2.020
Condiție2	00:02:21	00:04:09	00:07:13	00:11:32	00:11:58	00:12:30	00:21:09	-	75	210	390	410	430	800
Condiție3	00:02:21	00:28:32	00:49:15	00:50:32	01:10:33	01:32:54	02:21:50	-	1.120	2.000	2.050	2.900	3.900	6.000
Timp DROP INDEX+ CREATE INDEX	-	00:02:41	00:06:58	00:09:45	00:12:00	00:13:54	00:18:39	Cât % din timpul de actualizare cu index reprezintă timpul de actualizare fără index						
Timp UPDATE+ DROP INDEX+ CREATE INDEX	-	00:05:02	00:09:19	00:12:06	00:14:21	00:16:15	00:21:00	1.	48	54	47	43	39	42
								2.	121	129	105	120	130	100
								3.	18	19	24	20	18	15

Condiție 1 - Nu există index pe condiția de căutare din WHERE, dar UPDATE modifică ceilalți indecşi existenți

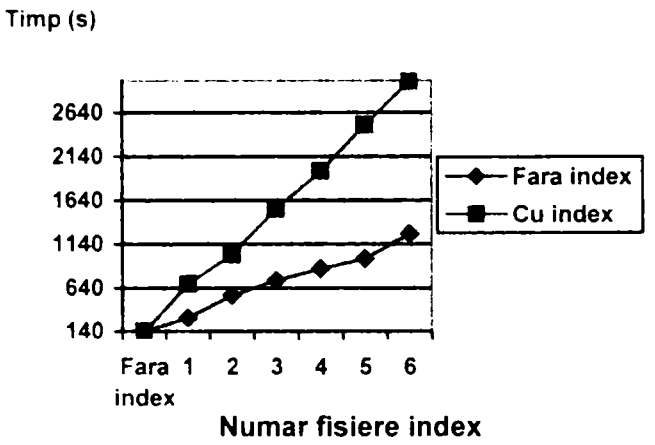
Condiție 2 - Există index pe condiția de căutare din WHERE, dar acesta nu este modificat de UPDATE

Condiție 3 - Există index pe condiția de căutare din WHERE şi acesta este modificat de UPDATE

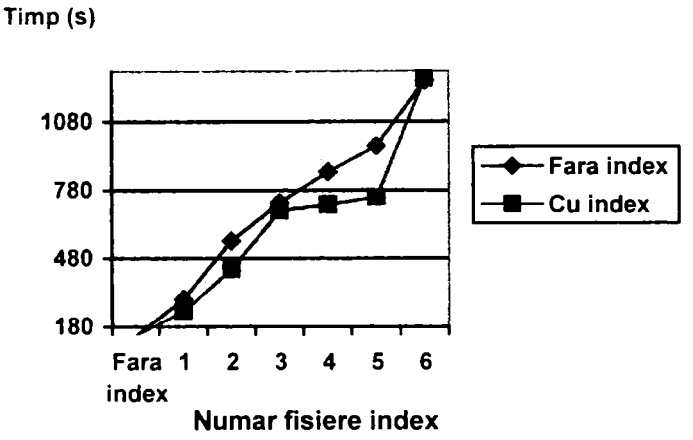
Grafic 17. UPDATE



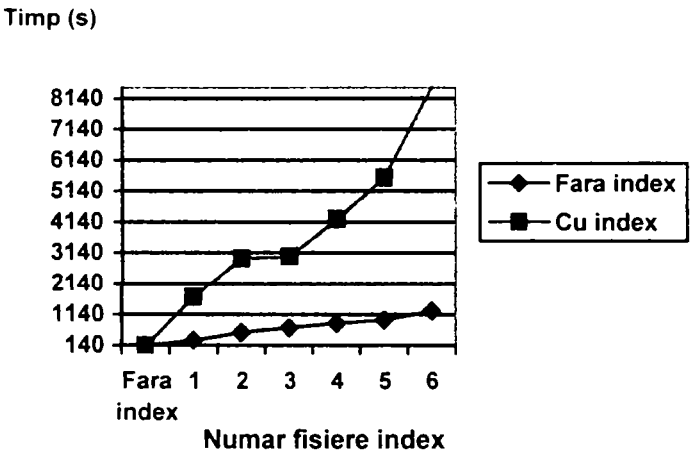
Grafic 18. UPDATE Conditie1



Grafic 19. UPDATE Conditie 2



Grafic 20. UPDATE Conditie 3



Concluziile care se pot desprinde din tabelul 10 şi din graficele 17-20 sunt următoarele:

- în cazul unor modificări masive ale înregistrărilor din fişierul de date timpii de execuție ai operației UPDATE cresc dacă fişierul de date are asociate fişiere index;
- cele mai mici creşteri se înregistrează în situația în care există index pe condiția de căutare a clauzei WHERE din blocul UPDATE;

    - dacă există un singur fişier index timpul de modificare aproape se dublează;
    - apariția celui de al-II-lea şi al-III-lea fişier index determină, de asemenea, o dublare a timpilor de execuție față de situațiile cu un număr mai mic cu o unitate de fişiere index;

- apariția celui de al-IV-lea şi al-V-lea fişier index determină o creştere mult mai mică a timpului, undeva în jurul a 5 % din timpul anterior de execuție;
- apariția celui de al-V-lea fişier index determină iarăşi o dublare a timpului de execuție al operației;

- creşteri ceva mai mari se înregistrează în situația în care nu există index pe condiția de căutare a clauzei WHERE din blocul UPDATE, dar există alte fişiere index care trebuie actualizate datorită operației de modificare efectuate;

  - dacă există un singur fişier index timpul de modificare se triplează;
  - apariția celui de al-II-lea şi al-III-lea fişier index determină aproximativ o dublare a timpilor de execuție față de situațiile cu un număr mai mic cu o unitate de fişiere index;
  - apariția celui de al-IV-lea, al-V-lea şi al-VI-lea fişier index determină o creştere mult mai mică a timpului, undeva în jurul a 25 % din timpul anterior de execuție;

- cele mai mari creşteri se înregistrează în situația în care există index pe condiția de căutare a clauzei WHERE din blocul UPDATE şi acest index trebuie actualizat ca urmare a operației UPDATE împreună cu alte posibile fişiere index care există pentru fişierul de date;

  - fiecare fişier index existent, care trebuie actualizat ca urmare a operației UPDATE, introducere o creştere de aproximativ 10 ori a timpului de efectuare a operației;

- în situația în care nu există index pe condiția de căutare din clauza WHERE, dar există alți indecşi care sunt modificați de operația UPDATE, este indicat să se elimine indecşii inaintea efectuării operației de modificare după care să se recreeze aceşti indecşi. Se obține astfel o reducere de aproximativ 45-60% a timpilor necesari efectuării operației de modificare, care persistă indiferent de numărul de fişiere index asociate fişierului de date.

- în situația în care există index pe condiția de căutare din clauza WHERE, dar acesta nu este modificat de operația UPDATE ci doar alți indecşi existenți, este indicat să se păstreze indecşii pe durata operației de modificare pentru că timpii de execuție sunt mai mici decât în situația în care indecşii ar fi eliminați inaintea efectuării operației UPDATE şi ar fi recreați ulterior. Păstrarea fişierelor index aduce un câştig de cca. 20-30% la nivelul timpilor de execuție.
- în situația în care există index pe condiția de căutare din clauza WHERE, şi acest index este modificat de operația de actualizare împreună cu alți indecşi existenți, este indicat să se elimine indecşii inaintea efectuării operației de modificare după care să se recreeze aceşti indecşi. Se obține astfel o reducere de aproximativ 80% a timpilor necesari efectuării operației de modificare, care persistă indiferent de numărul de fişiere index asociate fişierului de date.

Ca o concluzie finală se poate spune că fişierele de date care sunt des actualizate prin modificarea unui număr mare de înregistrări din fişier nu este avantajos să aibă fişiere index asociate care trebuie să fie actualizate ca urmare a operației UPDATE efectuate.Totuşi, dacă aceste fişiere index sunt construite pe baza unor chei de indexare care apar ca şi condiții în clauza WHERE a blocului UPDATE, poate fi acceptabilă existența a 1-2 fişiere index.

În situația în care există index pe condiția de căutare din clauza WHERE, dar acesta nu este modificat de operația UPDATE ci doar alți indecşi existenți, este indicat să se păstreze indecşii pe durata operației de modificare pentru că timpii de execuție sunt mai mici decât în situația în care indecşii ar fi eliminați inaintea efectuării operației UPDATE şi ar fi recreați ulterior. In celelalte situații este mai avantajos să se elimine indecşii pe durata efectuării operațiilor de modificare şi apoi să se recreeze aceste fişiere.

- **DELETE**

Operațiile de ştergere s-au efectuat asupra unor fişiere de date de mărime mare, având în jur de 4.000.000 de înregistrări. Numărul înregistrărilor supuse operațiilor de ştergere a fost situat între 635.320 înregistrări şi 1.058.634 înregistrări, adică în jur de 15-25% din mărimea fişierului de date. Analiza operațiilor de ştergere s-a efectuate pentru următoarele situații:

- nu există index pe condiția din clauza WHERE a blocului DELETE;

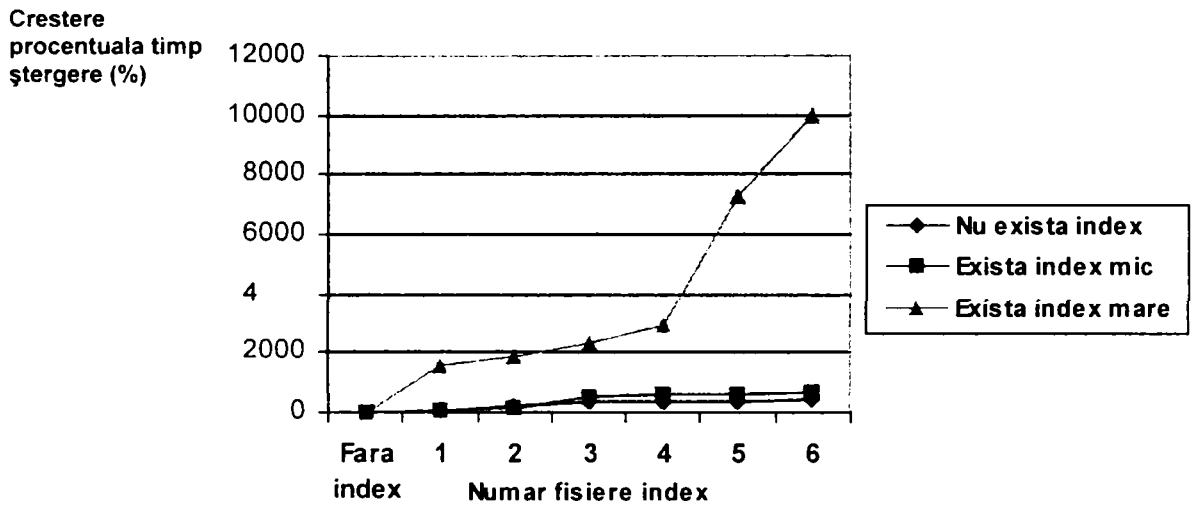
- există index pe condiția de selecție din clauza WHERE a blocului DELETE şi acest index are o cheie de indexare de mărime mică;
- există index pe condiția de selecție din clauza WHERE a blocului DELETE şi acest index are o cheie de indexare de mărime mare.

Rezultatele obținute la testele efectuate pentru operațiile de ştergere sunt sintetizate în tabelul 11 şi sunt prezentate în graficele 21-24 pentru o mai uşoară interpretare. Creşterea procentuală reprezintă creşterea în procente a timpilor necesari pentru ştergerea din fişierul de date a înregistrărilor dorite în situația introducerii fişierelor index față de situația în care nu există fişier index asociat. Graficele 22, 23 şi 24 prezintă comparativ timpii de ştergere în situația în care există fişiere index asociate fişierului de date în momentul efectuării operației de ştergere față de situația în care aceste fişiere index au fost eliminate şi sunt recreate ulterior pentru toate cele trei cazuri analizate.

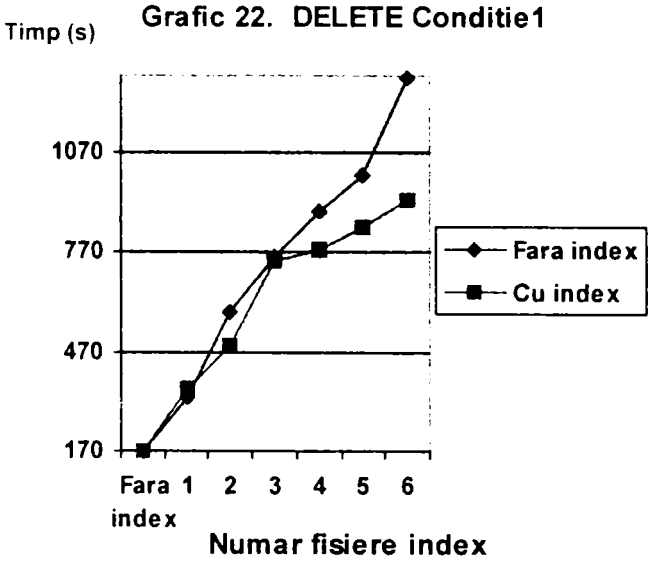
Tabel 11. Timpii de răspuns pentru operația DELETE

Condiție	Timp inserare (hh:mm:ss) (15-25% tupluri şterse)							Creştere procentuală (%)						
	Fără index	1 index	2 indecşi	3 indecşi	4 indecşi	5 indecşi	6 indecşi	Fără index	1	2	3	4	5	6
1. Nu există index	00:02:51	00:05:57	00:08:06	00:12:22	00:13:02	00:14:09	00:15:26	-	110	190	340	360	400	440
2. Există index mic	00:02:51	00:04:09	00:07:13	00:17:45	00:20:22	00:20:30	00:22:14	-	50	160	530	620	620	680
3. Există index mare	00:02:51	00:47:25	00:57:03	01:09:55	01:25:25	03:30:48	04:48:12	-	1570	1900	2360	2900	7300	10000
Timp DROP INDEX+ CREATE INDEX	-	00:02:41	00:06:58	00:09:45	00:12:00	00:13:54	00:18:39	Cât % din timpul de ştergere cu index reprezintă timpul de ştergere fără index						
Timp DELETE+ DROP INDEX+ CREATE INDEX	-	00:05:32	00:09:49	00:12:36	00:14:51	00:16:45	00:21:30	1.	93	121	102	114	118	140
								2.	133	136	71	73	81	97
								3.	12	17	18	17	8	8

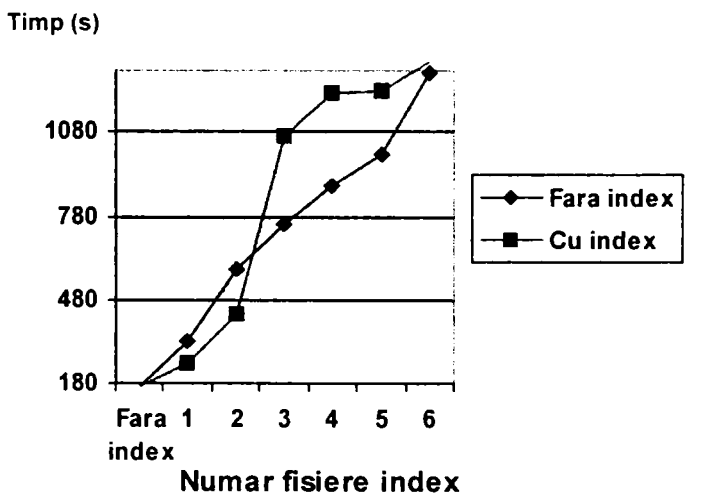
Grafic 21. DELETE



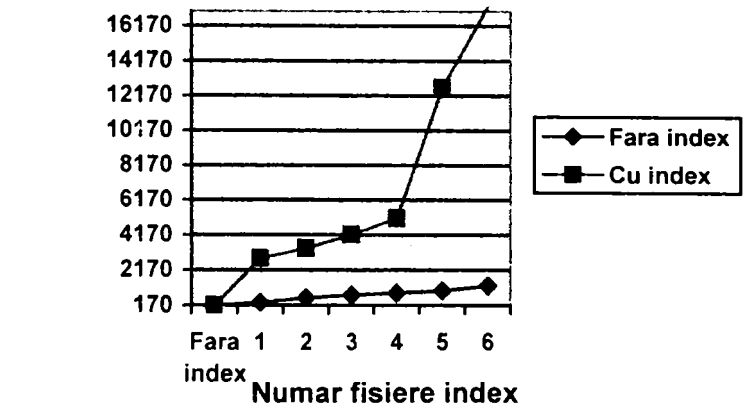
Grafic 22. DELETE Conditie1



Grafic 23. DELETE Conditie 2



Grafic 24. DELETE Conditie 3



Concluziile care se pot desprinde din tabelul 11 şi din graficele 21-24 sunt următoarele:

- în cazul unor ştergeri masive ale înregistrărilor din fişierul de date, timpii de execuție ai operației DELETE cresc dacă fişierul de date are asociate fişiere index;
- în situația existenței fişierelor index asociate fişierului de date, timpii de execuție ai operației DELETE sunt influențați într-o foarte mare măsură de dimensiunea cheilor de indexare:
    - pentru chei de indexare de dimensiuni mici creşterile timpilor de execuție sunt mici:
        - pentru un singur fişier index timpul creşte aproximativ cu jumătate față de valoarea timpului de execuție a operației în absența fişierelor index;
        - pentru 2 fişiere index timpul creşte aproximativ cu o dată şi jumătate față de valoarea timpului de execuție a operației în absența fişierelor index;
        - pentru 3-6 fişiere index timpul creşte de aproximativ 5-6 ori față de valoarea timpului de execuție a operației în absența fişierelor index:
    - pentru chei de indexare de dimensiuni mari creşterile timpilor de execuție sunt foarte mari:
        - existența numai a unui singur fişier index, având o cheie de indexare de dimensiuni mari, face ca timpul de execuție al operației să se mărească de 15 ori;
        - cu cât numărul de fişiere index se măreşte timpul de execuție al operației creşte, putându-se înregistra timpi cu până la 100 de ori mai mari;
- în cazul a 1, 2 fişiere index asociate fişierului de date, dacă există index pe condiția de selecție din clauza WHERE a blocului DELETE, se obțin timpi de execuție ceva mai buni față de situația în care acest index nu există;

- în cazul a 3-6 fişiere index asociate fişierului de date, nu se mai obține nici un avantaj vis-à-vis de viteza de execuție a operației DELETE în situația când există index pe condiția de selecție din clauza WHERE a blocului DELETE;
- în situația în care nu există index pe condiția de căutare din clauza WHERE, dar există alți indecşi care sunt prelucrați de operația DELETE, este indicat să se păstreze indecşii pe durata operației de ştergere pentru că timpii de execuție sunt mai mici decât în situația în care indecşii ar fi eliminați inaintea efectuării operației DELETE şi ar fi recreați ulterior. Păstrarea fişierelor index aduce un câştig de cca. 10-20% la nivelul timpilor de execuție.
- în situația în care există index pe condiția de căutare din clauza WHERE, având o cheia de indexare de mărime mică şi mai există şi alți indecşi asociați fişierului de date, se pot păstra aceşti indecşi pe durata operației de ştergere.

    - dacă există 1-2 indecşi se obține chiar un câştig la nivelul timpului de execuție de cca. 30%;
    - pentru 3 sau mai multe fişiere index timpii de execuție se depreciază în situația existenței fişierelor index, dar depreciarea este relativ mică, în jur de 20%.

    - în situația în care există index pe condiția de căutare din clauza WHERE, având o cheia de indexare de mărime mare şi mai există şi alți indecşi asociați fişierului de date, este indicat să se elimine indecşii inaintea efectuării operației de ştergere după care să se recreeze aceşti indecşi. Se obține astfel o reducere de aproximativ 80-90% a timpilor necesari efectuării operației de ştergere, care persistă indiferent de numărul de fişiere index asociate fişierului de date.

Ca o concluzie finală se poate spune că fişierele de date care sunt des actualizate prin ştergeri masive de înregistrări nu este avantajos să aibă fişiere index asociate. Timpii de ştergere cresc, în general, sesizabil în cazul în care există fişiere index asociate fişierului de date. Totuşi, dacă aceste fişiere index sunt construite pe baza unor chei de indexare de mărime mică care apar ca şi condiții în clauza WHERE a blocului DELETE, poate fi acceptabilă existența a 1-2 fişiere index.

In situația în care nu există index pe condiția din clauza WHERE sau acest index are o cheiei de indexare de mărime mică este oportun să se păstreze indecşii pe durata efectuării operației DELETE. In schimb, dacă cheia de indexare are mărime mare este indicat să se elimine fişierele index pe durata efectuării operațiilor de ştergere şi apoi să se recreeze aceste fişiere.

3.3.3. Influența structurilor indexate asupra necesarului de spațiu de memorare. Concluzii.

In luarea deciziei, cu privire la structurile indexate care vor fi create pe baza de date, trebuie luate în considerare şi aspectele legate de spațiul de memorare necesar stocării acestor structuri. Uneori este posibil să existe restricții în ceea ce priveşte capacitatea memoriei secundare aflată la dispoziția bazei de date, şi în această situație trebuie puse în balanță avantajele în timpi de răspuns mai mici la operațiile efectuate asupra bazei de date şi dezavantajele ocupării stațiului de memorare de către fişierele index.

Au fost analizate mărimile fişierelor index create pentru un fişier de date a cărui mărime a variat de la 100 Mo la 350 Mo. Mărimea cheii de indexare a fost şi ea variabilă şi, de asemenea, numărul de valori distincte ale cheii de indexare. Rezultatele analizei sunt prezentate în tabelul 12.

Tabel 12. Necesarul de spațiu de memorare pentru structurile indexate

Denumire index	Cheie de indexare	Mărime tabelă de bază (Mo)	Mărime index (Mo)	Câte % din mărimea tabelei este indexul	Câte % din mărimea rândului de date este cheia de indexare	Numărul de înregistrări	Numărul de valori distincte ale cheii de indexare
I_MARF_COD	CODP	109	30	27	9	1.475.184	1.475.184
I_MARF_COD	CODP	173	45	27	9	2.366.914	2.366.914
I_MARF_COD	CODP	310	83	27	9	4.243.680	4.243.680
I_MARF_COMP1	GRUPA+COD_FURN+DENP	109	66	60	60	1.475.184	778
I_MARF_COMP1	GRUPA+COD_FURN+DENP	173	101	60	60	2.366.914	778
I_MARF_COMP1	GRUPA+COD_FURN+DENP	310	185	60	60	4.243.680	778
I_MARF_DEN	DENP	109	55	51	46	1.475.184	803
I_MARF_DEN	DENP	173	90	52	46	2.366.914	803
I_MARF_DEN	DENP	310	157	51	46	4.243.680	803
I_MARF_COMP2	GRUPA+COD_FURN	109	35	32	14	1.475.184	79
I_MARF_COMP2	GRUPA+COD_FURN	173	54	32	14	2.366.914	79

I_MARF_COMP2	GRUPA+COD_FURN	310	99	32	14	4.243.680	79
I_MARF_PRET	PRET_ACHIZ	109	26	24	4	1.475.184	760
I_MARF_PRET	PRET_ACHIZ	173	41	24	4	2.366.914	760
I_MARF_PRET	PRET_ACHIZ	310	74	24	4	4.243.680	760
I_MARF_GRUPA	GRUPA	109	31	29	7	1.475.184	81
I_MARF_GRUPA	GRUPA	173	49	29	7	2.366.914	81
I_MARF_GRUPA	GRUPA	310	87	29	7	4.243.680	81
I_MARF_UM	UM	109	25	23	3	1.475.184	7
I_MARF_UM	UM	173	39	23	3	2.366.914	7
I_MARF_UM	UM	310	70	23	3	4.243.680	7

Se pot trage următoarele conluzii observând datele din tabelul 12:

- indexul creşte ca mărime cu acelaşi procent cu care creşte şi tabela de bază pentru care a fost creat indexul;
- raportul dintre mărimea indexului şi mărimea tabelei de bază este influențat doar de mărimea cheii de indexare şi de numărul de valori distincte ale cheii de indexare;
- influența mărimii cheii de indexare asupra mărimii indexului este mai mare decât cea a numărului de valori distincte ale cheii de indexare;
- un index pentru o cheie de indexare de mărime mică, sub 10 % din mărimea rândului de date, ocupă minim 20 - 25 % din mărimea tabelei pentru care a fost creat indexul;
- un index pentru o cheie de indexare de mărime mare, în jur de 50 % din mărimea rândului de date, ocupă puțin peste 50 % din mărimea tabelei pentru care a fost creat indexul;

3.3.4. Index bitmap versus index clasic de tip arbore B

Un alt set de teste a fost elaborat, pentru a studia comparativ performanțele obținute pe baza de date, în cazul folosirii unor structuri indexate de tip diferit [Lumpk 03]. Au fost luate în calcul structura clasică de index, cea la care indexul este construit pe baza unor arbori B care la nivelul nodurilor frunză păstrează identificatorii rândurilor tabelei care au acea valoare a cheii de indexare, şi structura mai nou apărută, cea de index bitmap, care foloseşte în locul identificatorilor de rând hărțile de biți.

Testele au fost efectuate în acest caz pe produsul Oracle9*i*, deoarece la nivelul acestuia există implementate pentru indecşi mai multe tipuri de structuri bitmap. Oracle9*i* dispune de indecşi bitmap dinamici, statici şi de tip join [Oracl 02]. Spre deosebire de acest produs, celelalte două produse analizate sunt mult mai sărace la acest capitol. DB2 dispune doar de

indecşi bitmap dinamici [IBM 02], în timp ce SQL Server 2000 nu dispune de nici un fel de structură de tip bitmap pentru indecşi [Micro 00]. Toate cele trei produse au implementate structuri indexate de tip arbore B.

Indecşii de tip bitmap sunt deosebit de utili în aplicațiile din domeniul depozitelor de date.

3.3.4.1. Câştigul în *spațiu de memorare*. Concluzii.

Spațiul ocupat de fişierele index de tip bitmap este semnificativ mai mic decât spațiul ocupat de fişierele index construite pe baza unor arbori B. Din acest punct de vedere este mult mai avantajos să se utilizeze fişiere index bitmap, de fiecare dacă când acest lucru este posibil. Acest lucru este însă posibil numai în Oracle 9*i* pentru că el este singurul la care indexul bitmap este salvat pe suport fizic. In cazul produsului DB2 indexul este construit doar în memoria internă a calculatorului, înaintea executării operațiilor DML care ar putea beneficia de pe urma acestei structuri.

In tabelul 13 sunt prezentate comparativ mărimile fişierelor index bitmap şi de tip arbore B, construite pentru fişierele de date MARFURI şi MISCMARFURI.

Tabel 13. Mărimea fişierelor index asociate tabelelor MARFURI şi MISCMARFURI

Număr de înregistrări	Mărime fişier date (Mo)	Mărime fişier index (Mo)		Cât % din fişierul de date reprezintă indexul?		Cât % din indexul Arbore B este indexul Bitmap
		Arbore B	Bitmap	Arbore B	Bitmap	
1.475.184	109	25	0,5	23	0,458	2
2.366.914	173	39	0,7	23	0,404	2
4.243.680	310	70	1,4	23	0,451	2
219.196.176	16.020	3.926	221	24	1,4	6
219.196.176	16.020	4.526	305	28	1,9	7
219.196.176	16.020	5.079	1.148	32	7,2	22

Primele trei poziții din tabel reprezintă mărimile structurilor indexate care s-au construit asupra tabelei MARFURI a cărui mărime a variat după cum se poate observa.

Indexul bitmap s-a creat folosind comanda:

```
CREATE BITMAP INDEX I_MARFURI_UM ON MARFURI(UM);
```

iar indexul de tip arbore B s-a creat folosind comanda:

```
CREATE INDEX I_MARFURI_UM ON MARFURI(UM);
```

Numărul de valori distincte ale cheii de indexare este foarte mic în acest caz. Există numai 7 valori distincte ale acestei chei la nivelul fișierului de date.

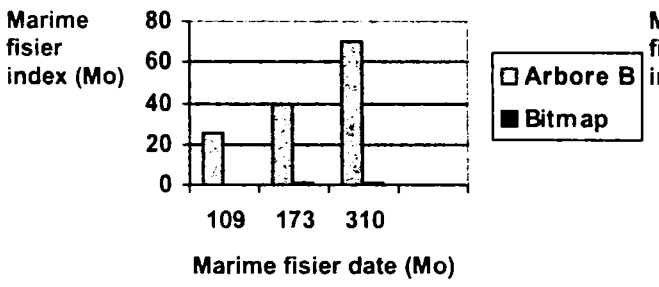
Ultimile trei poziții din tabelul 13 prezintă mărimile structurilor indexate, care s-au construit asupra tabelei MISCMARFURI.

Cheile de indexare folosite au fost următoarele:

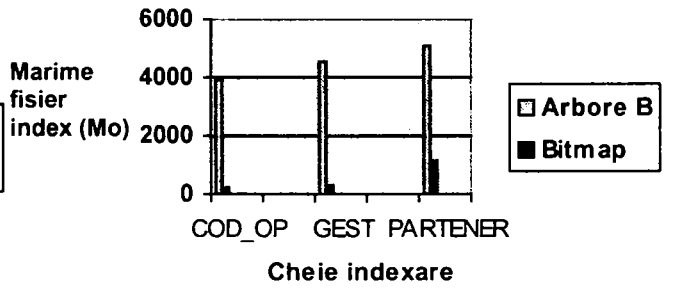
- COD_OPER având 8 valori distincte;
- GEST având 25 valori distincte;
- PARTENER având 12.000 valori distincte;

Graficele 25 şi 26 prezintă datele din tabelul 13.

Grafic 25. Comparatie index Bitmap - index Arbore B (spatiu de memorare). Fisier date MARFURI.



Grafic 26. Comparatie index Bitmap - index Arbore B (spatiu de memorare). Fisier date MISCMARFURI.



Analizând datele din tabelul 13 se pot trage următoarele concluzii:

- spațiul de memorare ocupat de fişierele index bitmap este mult mai mic decât cel ocupat de fişierele index de tip arbore B;
    - fişierul index bitmap ocupă sub 5% din mărimea fişierului index de tip arbore B, pentru o cheie de indexare de mărime mică şi selectivitate mare;
    - fişierul index bitmap ocupă sub 10% din mărimea fişierului index de tip arbore B, pentru o cheie de indexare de mărime mică şi selectivitate medie;

- fișierul index bitmap ocupă în jur de 20% din mărimea fișierului index de tip arbore B, pentru o cheie de indexare de mărime medie și selectivitate medie;

- raportul dintre mărimea fișierului index bitmap și mărimea fișierului index de tip arbore B nu este influențat de mărimea fișierului de date asupra căruia se crează aceste structuri auxiliare de acces, deci câștigul în spațiu de memorare se păstrează constant;
- raportul dintre mărimea fișierului index bitmap și mărimea fișierului index de tip arbore B este influențat doar de selectivitatea cheii de indexare pe care se construiește fișierul index.
- cu cât numărul valorilor distincte ale cheii de indexare este mai mare, cu atât câștigul în spațiu de memorare este mai mic.

3.3.4.2. Câștigul în *timp de creare*. Concluzii.

Timpul de creare al fișierelor index de tip bitmap este și el semnificativ mai mic decât timpul necesar pentru crearea fișierelor index construite pe baza unor arbori B. Și din acest punct de vedere este mult mai avantajos să se utilizeze fișiere index bitmap, de fiecare dacă când acest lucru este posibil. Din nou această afirmație este valabilă numai în cazul lui Oracle 9*i*.

In tabelul 14 sunt prezentați comparativ timpii de creare ai fișierelor index bitmap și ai celor de tip arbore B. Structurile indexate sunt construite, de asemenea, pentru tabelele MARFURI și MISCMARFURI.

Tabel 14. Timpii de creare ai fișierelor index asociate tabelelor MARFURI și MISCMARFURI

Număr de înregistrări	Mărime fișier date (Mo)	Timpul de creare a fișierelor index (hh:mm:ss)		Cât % din timp Arbore B reprezintă timp Bitmap
		Arbore B	Bitmap	
1.475.184	109	00:00:51	00:00:20	40
2.366.914	173	00:00:58	00:00:21	36
4.243.680	310	00:01:49	00:00:24	23
219.196.176	16.020	01:23:37	00:09:08	11
219.196.176	16.020	01:25:19	00:09:23	11
219.196.176	16.020	01:28:02	00:13:30	15

Primele trei poziții din tabel reprezintă timpii de creare ai structurilor indexate care s-au construit asupra tabelei MARFURI. Atât indexul bitmap cât şi cel clasic s-au construit folosind cheia de indexare UM. Numărul de valori distincte ale cheii de indexare este foarte mic în acest caz. Există numai 7 valori distincte ale acestei chei la nivelul tabelei MARFURI.

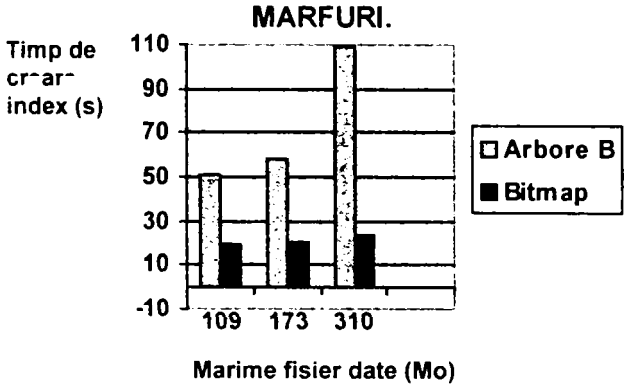
Ultimile trei poziții din tabelul 14 reprezintă timpii de creare ai structurilor indexate care s-au construit asupra tabelei MISCMARFURI.

Cheile de indexare folosite au fost următoarele:

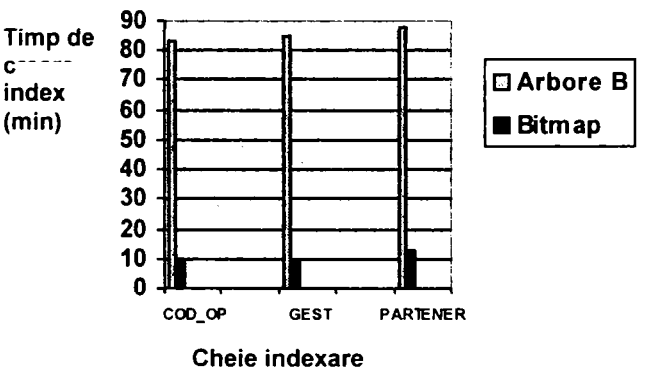
- COD_OPER având 8 valori distincte;
- GEST având 25 valori distincte;
- PARTENER având 12.000 valori distincte;

Graficele 27 şi 28 prezintă datele din tabelul 14.

Grafic 27. Comparatie index Bitmap - index Arbore B (timp de creare index). Fisier date MARFURI.



Grafic 28. Comparatie index Bitmap - index Arbore B (timp de creare index). Fisier date MISCMARFURI.



Analizând datele din tabelul 14 se pot trage următoarele concluzii:

- timpul de creare a fişierelor index bitmap este mult mai mic decât cel necesar pentru crearea fişierele index de tip arbore B. Timpul de creare a fişierul index bitmap este cel mult 10-15 % din timpul de creare a fişierului index de tip arbore B.

3.3.4.3. Câştigul în *timp de răspuns* pentru operațiile de interogare. Concluzii.

Pentru tabelele MARFURI şi MISCMARFURI, având în jur de 4.500.000 de rânduri s-au construit atât indecşi clasici cât şi indecşi bitmap pe cheile de indexare UM, COD_OPER, GEST, PARTENER. Asupra tabelelor au fost executate interogări de forma celor următoare, atât în situația în care nu existau indecşi asociați fişierului de date, cât şi în situațiile în care existau aceşti indecşi, fie clasici fie de tip bitmap.

1.
```
SELECT COUNT(*)
FROM MARFURI
WHERE UM='BUC';
```
2.
```
SELECT DISTINCT UM
FROM MARFURI
WHERE UM <>'BUC';
```
3.
```
SELECT GEST, COUNT(*)
FROM  MISCMARFURI
GROUP BY GEST;
```
4.
```
SELECT DISTINCT COD_OPER
FROM  MISCMARFURI;
```
5.
```
SELECT COUNT(*)
FROM MISCMARFURI
WHERE COD_OPER='E'
      AND PARTENER='ANA S.R.L.';
```
6.
```
SELECT COUNT(*)
FROM MISCMARFURI
WHERE COD_OPER IN ('I', 'E','R');
```
7.
```
SELECT *
FROM MARFURI
WHERE UM='BUC' OR UM='PAC'
```
8.
```
SELECT DENP, UM
FROM MARFURI
WHERE UM ='BUC';
```

Pentru implementarea în Oracle 9*i* operațiile de interogare de mai sus s-au executat prin intermediul unor scripturi PL/SQL care au colectat valorile timpilor de execuție în tabela REZULTATE. De exemplu, operația numărul 5 a fost executată folosind scriptul MiscMarfuriSelect3.sql, iar operația numărul 7 executând scriptul MarfuriSelect3.sql, ambele prezentate mai jos:

```
--MiscMarfuriSelect3.sql
DECLARE
 v_NrOp NUMBER(3);
 v_Operatie VARCHAR(200);
```

```
v_Start NUMBER(10,0);
v_End NUMBER(10,0);
v_Timp NUMBER(10,0);
v_Valoare VARCHAR(50);
v_NrInreg NUMBER(12,0);
v_Index CHAR(1);
v_TipIndex VARCHAR(20);
v_Cod_Oper MiscMarfuri.Cod_Oper%TYPE;
v_Partener MiscMarfuri.Partener%TYPE;
v_TotalInreg NUMBER(15,0);
CURSOR c_MiscMarfuri IS
  SELECT count(*)
  FROM MiscMarfuri
  WHERE Cod_Oper=v_Cod_Oper AND Partener=v_Partener;
CURSOR c_TotalInreg IS
  SELECT count(*)
  FROM MiscMarfuri;
BEGIN
v_NrOp:=&NrOp;
OPEN c_TotalInreg;
FETCH c_TotalInreg INTO v_TotalInreg;
CLOSE c_TotalInreg;
v_Cod_Oper:='&Cod_Oper';
v_Partener:='&Partener';
v_Index:= '&Index';
v_TipIndex:= '&TipIndex';
v_Operatie:='SELECT count(*) FROM MiscMarfuri WHERE Cod_Oper= AND Partener=';
v_Start:=DBMS_UTILITY.GET_TIME;
OPEN c_MiscMarfuri;
v_End:=DBMS_UTILITY.GET_TIME;
v_Timp:=(v_End-v_Start)*10;
FETCH c_MiscMarfuri INTO v_NrInreg;
CLOSE c_MiscMarfuri;
v_Valoare:=v_Cod_Oper||' AND '||v_Partener
INSERT INTO Rezultate(NrOp, Operatie, TotalInreg, InregPrel,Timp, Valoare, Index, TipIndex,
      NrIndex) VALUES (v_NrOp, v_Operatie, v_TotalInreg, v_NrInreg, v_Timp, v_Valoare,
      v_Index, v_TipIndex, null);
COMMIT;
END;
```

La rularea scriptului se furnizează ca parametrii numărul operației, valorile pentru COD_OPER şi PARTENER care vor fi folosite în clauza WHERE a blocului SELECT, dacă există sau nu index asociat fişierului de date, precum şi tipul indexului. Timpul de execuție al operației SELECT, în cazul particular specificat, este inserat în tabela REZULTATE.

Acest tip de interogare a fost executat în mod repetat folosindu-se diferite valori pentru COD_OPER şi PARTENER. Unele dintre valorile colectate în tabela REZULTATE pot fi observate în anexa 1.

```
--MarfuriSelect3.sql
DECLARE
 v_NrOp NUMBER(3);
 v_Operatie VARCHAR(200);
 v_Start NUMBER(10,0);
 v_End NUMBER(10,0);
 v_Timp NUMBER(10,0);
 v_Valoare VARCHAR(50);
 v_NrInreg NUMBER(12,0);
 v_Index CHAR(1);
 v_TipIndex VARCHAR(20);
 v_Marfuri Marfuri%ROWTYPE;
 v_Um1 Marfuri.Um%TYPE;
 v_Um2 Marfuri.Um%TYPE;
 v_TotalInreg NUMBER(15,0);
 CURSOR c_Marfuri IS
  SELECT *
  FROM Marfuri
  WHERE Um=v_Um1 OR Um=v_Um2;
 CURSOR c_TotalInreg IS
  SELECT count(*)
  FROM Marfuri;
BEGIN
 v_NrOp:=&NrOp;
 OPEN c_TotalInreg;
 FETCH c_TotalInreg INTO v_TotalInreg;
 CLOSE c_TotalInreg;
 v_Um1:='&Um1';
 v_Um2:='&Um2';
 v_Index:= '&Index';
 v_TipIndex:= '&TipIndex';
```

```
v_Operatie:='SELECT * FROM Marfuri WHERE Um= OR Um=';
v_Start:=DBMS_UTILITY.GET_TIME;
OPEN c_Marfuri;
v_End:=DBMS_UTILITY.GET_TIME;
v_Timp:=(v_End-v_Start)*10;
LOOP
  FETCH c_Marfuri INTO v_Marfuri;
  EXIT WHEN c_Marfuri%NOTFOUND;
END LOOP;
v_NrInreg:=c_Marfuri%ROWCOUNT;
CLOSE c_Marfuri;
v_Valoare:=v_Um1||' OR '||v_Um2
INSERT INTO Rezultate(NrOp, Operatie, TotalInreg, InregPrel,Timp, Valoare, Index, TipIndex,
      NrIndex) VALUES (v_NrOp, v_Operatie, v_TotalInreg, v_NrInreg, v_Timp, v_Valoare,
      v_Index, v_TipIndex, null);
COMMIT;
END;
```

La rularea scriptului se furnizează ca parametrii numărul operației, valorile pentru unitatea de măsură ce vor fi folosite în clauza WHERE a blocului SELECT, dacă există sau nu index asociat fișierului de date, precum şi tipul indexului. Timpul de execuție al operației SELECT, pentru fiecare caz particular specificat, este inserat în tabela REZULTATE. Tot în anexa 1 pot fi observate câteva dintre rezultatele colectate în tabela REZULTATE.

Cu ajutorul scriptului RezultateFinale2.sql datele testelor anterioare au fost sintetizate şi colectate în tabela REZULTATE_FINALE.

```
--RezultateFinale2.sql
BEGIN
  INSERT INTO Rezultate_Finale(NrOp,Operatie,TotalInreg,InregPrel,Timp,Index,TipIndex)
    SELECT NrOp, Operatie, TotalInreg, AVG(InregPrel), AVG(Timp), Index, TipIndex
    FROM Rezultate
    WHERE Index='D' AND TipIndex='Clasic'
    GROUP BY NrOp;
  INSERT INTO Rezultate_Finale(NrOp,Operatie,TotalInreg,InregPrel,Timp,Index,TipIndex)
    SELECT NrOp, Operatie, TotalInreg, AVG(InregPrel), AVG(Timp), Index, TipIndex
    FROM Rezultate
    WHERE Index='D' AND TipIndex='Bitmap static'
    GROUP BY NrOp;
```

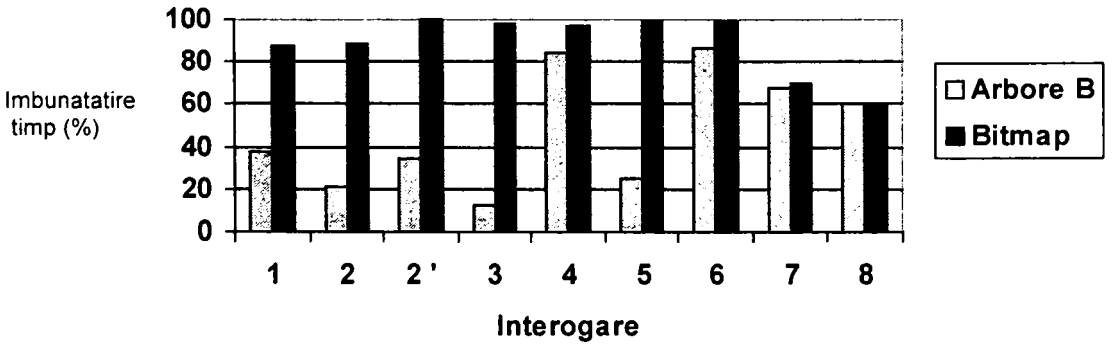
```
    INSERT INTO Rezultate_Finale(NrOp,Operatie,TotalInreg,InregPrel,Timp,Index,TipIndex)
      SELECT NrOp, Operatie, TotalInreg, AVG(InregPrel), AVG(Timp), Index, TipIndex
       FROM Rezultate
       WHERE Index='N'
       GROUP BY NrOp;
   COMMIT;
END;
```

Rezultatele testelor efectuate au fost analizate şi sunt prezentate în tabelul 15 şi în graficul 29.

Tabel 15. Timpii de răspuns la interogări index Bitmap - index Arbore B.

Interogare	Timp de răspuns (s)			Imbunătățire (%)	
	Fără index	Cu index arbore B	Cu index bitmap	Index arbore B	Index bitmap
1	1,25	0,079	0,015	37	88
2 (.BUC` există în fişierul de date)	0,437	0,344	0,047	21	89
2 (.BUC` nu există în fişierul de date)	10,344	6,829	0,062	34	99,5
3	22,022	19,528	0,521	12	97,5
4	22,522	3,475	0,991	84	96,5
5	22,552	17,045	0,340	25	98,5
6	21,521	2,894	0,161	86	99
7	6,802	2,133	2,031	68	70
8	0,511	0,214	0,227	60	60

Grafic 29. Comparatie index Bitmap - index Arbore B (timp de raspuns interogare).



In situațiile în care blocul de interogare trebuie să selecteze din tabela de bază şi alte informații decât cele aflate în cheia de indexare, dacă indexul este de tip bitmap el nu este practic util pentru regăsirea informațiilor din tabela de bază. Asupra acesteia se efectuează un acces de tip FULL ACCES. In cazul în care indexul este clasic, se face FULL ACCES pe tabela de bază doar dacă numărul rândurilor returnate de interogare este mare, pentru condiții de selecție de tipul WHERE UM NOT IN (....), WHERE UM <>'...', etc.

Şi în cazul produsului DB2 se obține o îmbunătățire a timpilor de răspuns la interogări dacă indexul bitmap este folosit. Imbunătățirea este cu atât mai mare cu cât numărul valorilor distincte ale cheii de indexare este mai mic.

Concluziile ce se pot trage în ceea ce priveşte avantajele în timp de execuție la nivelul interogărilor formulate asupra unor tabele la nivelul cărora există indecşi bitmap sunt următoarele:

- dacă răspunsul la interogarea formulată poate fi extras exclusiv din indexul bitmap, precum în interogările 1-6, timpii de răspuns se îmbunătățesc considerabil, şi sunt mult mai mici decât timpii de răspuns dacă s-ar folosi index clasic:
  - îmbunătățirea timpului de răspuns se situează între 90-100 % în cazul existenței indexului bitmap pe condiția de căutare, față de situația în care nu există nici un fel de index asociat tabelei;
  - îmbunătățirea timpului de răspuns, în cazul existenței indexului bitmap față de situația unui index clasic pe aceeaşi cheie de indexare, se situează între 50-70 %;

- dacă răspunsul la interogarea formulată nu poate fi extras exclusiv din indexul bitmap, precum în interogările 7-8, timpii de răspuns se îmbunătățesc față de situația în care nu există index, dar îmbunătățirea este mult mai mică. Ea este comparabilă ca valoare cu cea obținută pentru index clasic.

3.3.4.4. Câştigul în *timp de răspuns* pentru operațiile de actualizare. Concluzii.

Avându-se în vedere concluziile trase la punctele 3.3.2.1. şi 3.3.2.2. s-au efectuat teste şi pentru a se vedea dacă existența indexului bitmap poate îmbunătății timpii de execuție ai

operațiilor de actualizare, în cazul în care numărul rândurilor din tabela de bază ce se prelucrează este mic sau foarte mare.

Asupra tabelei MARFURI având 4.243.680 înregistrări s-a creat un index având ca şi cheie de indexare pe UM. Numărul valorilor distincte ale cheii de indexare este de 7, fiind foarte mic în comparație cu numărul înregistrărilor din fişier. Acest index a fost creat atât ca index clasic cât şi ca index bitmap. Datele din tabelă au fost modificate astfel încât modificările făcute să determine reorganizarea indexului. Blocul SQL folosit pentru modificare a fost de forma:

```
UPDATE MARFURI
SET UM='BUC'
WHERE UM='buc';
```

Pentru implementarea în Oracle 9*i* s-a folosit scriptul MarfuriModifTipIndex.sql care preia ca parametrii numărul operație, valoarea unității de măsură pentru care se vor efectua modificările, noua valoare a unității de măsură, dacă există sau nu index asociat fişierului de date, precum şi tipul indexului. Valorile timpilor de execuție precum şi numărul de înregistrări prelucrate la fiecare apelare a scriptului sunt stocate în tabela REZULTATE. Unele dintre valorile colectate în tabela REZULTATE pot fi observate în anexa 1. Cu ajutorul scriptului RezultateFinale2.sql rezultatele obținute această categorie de teste sunt centralizate în tabela REZULTATE_FINALE.

```
--MarfuriModifTipIndex.sql
DECLARE
 v_NrOp NUMBER(3);
 v_Operatie VARCHAR(200);
 v_Start NUMBER(10,0);
 v_End NUMBER(10,0);
 v_Timp NUMBER(10,0);
 v_NrInreg NUMBER(12,0);
 v_Vechi_Um Marfuri.Um%TYPE;
 v_Nou_Um Marfuri.Um%TYPE;
 v_Index CHAR(1);
 v_TipIndex VARCHAR(20);
 v_NrIndex NUMBER(2,0);
 v_TotalInreg NUMBER(15,0);
```

```
 CURSOR c_TotalInreg IS
  SELECT count(*)
  FROM Marfuri;
CURSOR c_NrInreg IS
  SELECT count(*)
  FROM Marfuri
  WHERE Um=v_Nou_Um;
BEGIN
 v_NrOp:=&NrOp;
 OPEN c_TotalInreg;
 FETCH c_TotalInreg INTO v_TotalInreg;
 CLOSE c_TotalInreg;
 v_Vechi_Um:= '&Vechi_Um';
 v_Nou_Um:= '&Nou_Um';
 v_Index:= '&Index';
 v_TipIndex:= '&TipIndex';
 v_NrIndex:=&NrIndex;
 v_Operatie:='UPDATE Marfuri SET Um= WHERE Um=';
 v_Start:=DBMS_UTILITY.GET_TIME;
 UPDATE Marfuri SET Um=v_Nou_Um WHERE  Um=v_Vechi_Um;
 v_End:=DBMS_UTILITY.GET_TIME;
 v_Timp:=(v_End-v_Start)*10;
 OPEN c_NrInreg
 FETCH c_NrInreg INTO v_NrInreg;
 CLOSE c_NrInreg;
 INSERT INTO Rezultate(NrOp, Operatie, TotalInreg, InregPrel,Timp, Valoare, Index, TipIndex,
        NrIndex) VALUES (v_NrOp, v_Operatie, v_TotalInreg, v_NrInreg, v_Timp,
        v_Vechi_Um, v_Index, v_TipIndex, v_NrIndex);
 COMMIT;
END;
```

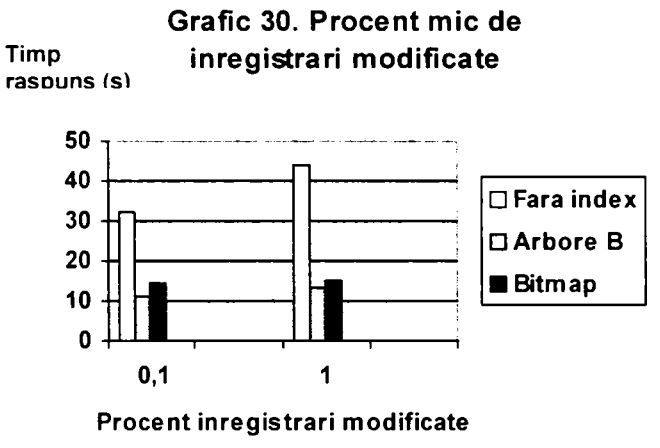
Rezultatele testelor sunt prezentate în tabela 16 şi în graficele 30 şi 31. Imbunătățire reprezintă procentul în care se reduce timpul de răspuns în cazul folosirii unei structuri indexate, fie de tip clasic fie de tip bitmap, față de situația în care nu este folosit nici un index.

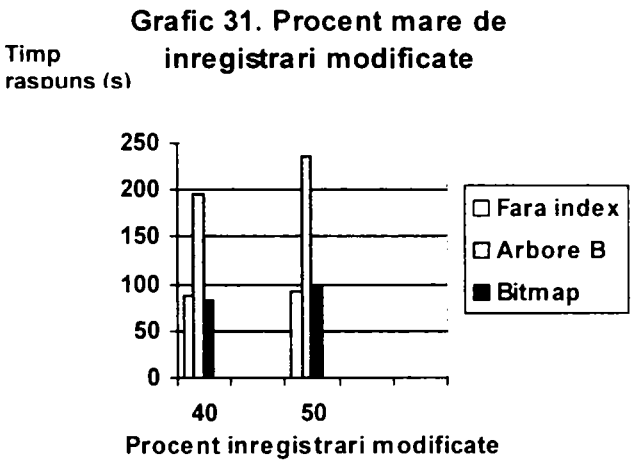
Tabel 16. Timpii de răspuns la operația UPDATE index Bitmap– index Arbore B.

Tip index	Timp de răspuns (s)				Imbunătățire (%)			
	Număr de înregistrări prelucrat (% din fişierul de date):							
	0,1	1	40	55	0,1	1	40	55

Fără index	32,312	43,844	87,016	91,422	-	-	-	-
Index arbore B	10,972	13,172	195,375	235,251	66	70	-124	-157
Index Bitmap	14,587	15,031	83,409	98,141	55	66	4	-7

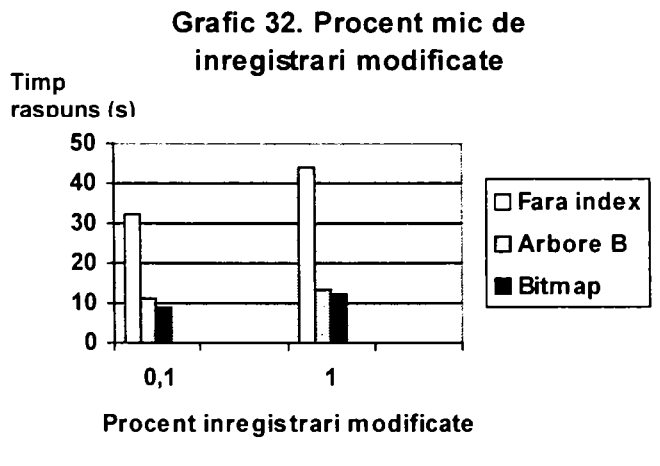
Comparație index Bitmap – index clasic Arbore B (timp răspuns UPDATE) Oracle 9*i*

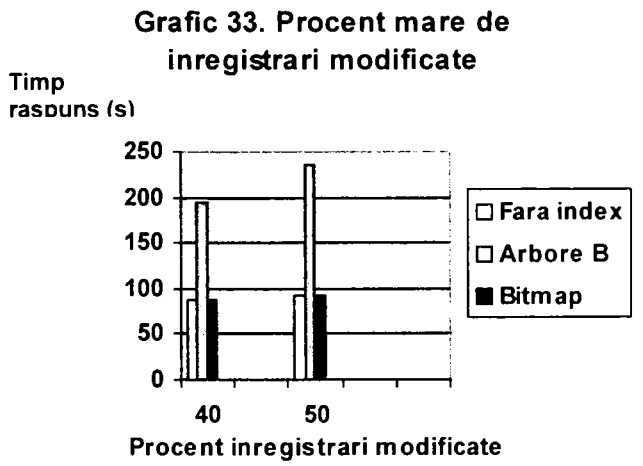




In cazul produsului DB2 indexul bitmap nu mai trebuie recreat odată cu efectuarea operațiilor de actualizare, deoarece el este un index dinamic, deci nu se mai consumă timpi suplimentari pentru recrearea acestuia. Şi în cazul unui număr de înregistrări modificate mic şi în cazul unui număr de înregistrări modificate mare este mai avantajos să se utilizeze un index bitmap decât un index clasic de tip arbore B aşa cum se poate observa şi în graficele 32 şi 33.

Comparație index Bitmap – index clasic Arbore B (timp răspuns UPDATE) DB2





Concluziile ce se pot trage pornind de la testele efectuate pentru Oracle *9i* sunt următoarele:

- operațiile de modificare care prelucrează un număr mic de înregistrări din fişier, raportat la mărimea acestuia, se execută mai rapid în cazul în care fişierul de date are asociată o

structură indexată având cheie de indexare compatibilă cu condiția de selecție din clauza WHERE a operației UPDATE;

  - îmbunătățirea timpilor de răspuns este ceva mai mare, circa 10 %, în situația în care indexul este de tip arbore B, decât în cazul în care el este de tip bitmap;
  - îmbunătățirea timpilor de răspuns în situația folosirii unui index clasic este de aproximativ 60-70 %;
  - îmbunătățirea timpilor de răspuns în situația folosirii unui index bitmap este de aproximativ 50-60 %;

- operațiile de modificare care prelucrează un număr mare de înregistrări din fişier, raportat la mărimea acestuia, se execută mai lent în cazul în care fişierul de date are asociată o structură indexată având cheie de indexare compatibilă cu condiția de selecție din clauza WHERE a operației UPDATE;

  - degradarea timpilor de răspuns este mult mare în situația în care indexul este de tip arbore B, decât în cazul în care el este de tip bitmap;
  - degradarea timpilor de răspuns în situația folosirii unui index clasic este de aproximativ 100-150 %;
  - degradarea timpilor de răspuns în situația folosirii unui index bitmap este de aproximativ 5-10 %;

- dacă procentul înregistrărilor prelucrate de o operație UPDATE variază în timp foarte mult, putând fi când foarte mic când foarte mare, este mai indicată o structură de index bitmap pentru că aceasta este mai stabilă vis-à-vis de îmbunătățirea timpilor de răspuns;

Concluziile ce se pot trage pornind de la testele efectuate pentru DB2 sunt următoarele:

- operațiile de modificare care prelucrează un număr mic de înregistrări din fişier, raportat la mărimea acestuia, se execută mai rapid în cazul în care fişierul de date are asociată o structură indexată având cheie de indexare compatibilă cu condiția de selecție din clauza WHERE a operației UPDATE;

  - îmbunătățirea timpilor de răspuns este ceva mai mare, sub 10 %, în situația în care indexul este de tip bitmap, decât în cazul în care el este de tip arbore B;
  - îmbunătățirea timpilor de răspuns în situația folosirii unui index clasic este de aproximativ 60-70 %;
  - îmbunătățirea timpilor de răspuns în situația folosirii unui index bitmap este de aproximativ 65-75 %;

- operațiile de modificare care prelucrează un număr mare de înregistrări din fişier, raportat la mărimea acestuia, se execută cu aceeaşi viteză cu care s-ar executa şi în absența indexului bitmap, dar cu viteză mult mai mare, aproximativ dublă, decât în cazul în care fişierul de date are asociată o structură indexată clasică de tip arbore B având cheie de indexare compatibilă cu condiția de selecție din clauza WHERE a operației UPDATE;
- indiferent de numărul înregistrărilor prelucrate de operațiile de actualizare este mai avantajos să se utilizeze un index bitmap decât un index clasic de tip arbore B.

Ca o concluzie finală, se poate spune că, ori de câte ori se pune problema alegerii între un index clasic şi un index bitmap, este preferabil să se opteze pentru structura bitmap a indexului, deoarece aceasta aduce avantaje atât la nivelul spațiului ocupat de disc şi a timpului necesar pentru crearea lui, cât şi la nivelul timpilor de execuție ai operațiilor de interogare cât şi de modificare şi ştergere. Deci, dacă cardinalitatea cheii de indexare este mică şi dacă operațiile DML se execută selectiv asupra înregistărilor din fişier, în funcție de criterii ce țin de cheia de indexare, este indicat să se folosească o structură de index bitmap.

3.3.5. Comparație între tabelele heap şi tabelele organizate indexat

Oracle 9*i* şi SQL Server 2000 permit memorarea datelor şi sub forma unor tabele organizate indexat. In Oracle 9*i* acestea se numesc *tabele organizate indexat* sau *IOT*, de la varianta englezească *indexed organized tables* [Oracl 02]. In SQL Server 2000 tabela organizată indexat este de fapt un index, denumirea folosită fiind aceea de *index c lusterat* [Micro 02-2].

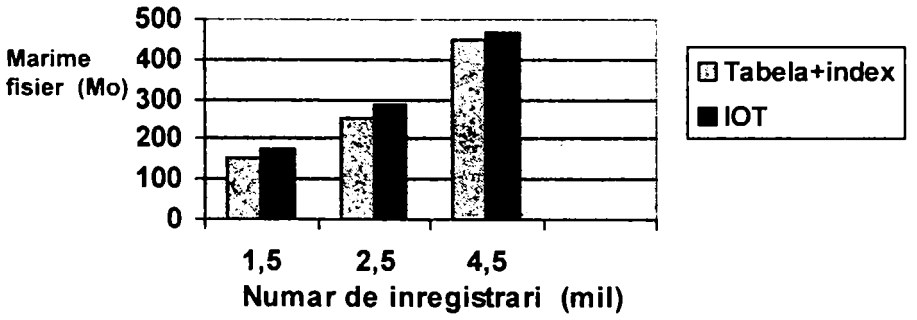
3.3.5.1. Câştigul în *spațiu de memorare*. Concluzii.

Testele efectuate pe Oracle 9*i* au arătat că la nivelul spațiului de memorare nu se obține câştig. Dimensiunea spațiului ocupat de tabela organizată indexat este mai mare decât dimensiunea spațiului ocupat de tabela organizată ca fişier heap care are asociat un index pe cheia primară. In tabelul 17 sunt trecute măsurătorile efectuate ale spațiului de memorare ocupat de tabela MARFURI, care este organizată fie ca tabelă indexată fie ca tabelă heap cu index asociat pe cheia primară. Graficul 34 prezintă evoluția ca mărime a spațiului de memorare necesar pentru organizarea tabelei sub formă de tabelă clasică cu index asociat sau sub formă de tabelă organizată indexat.

Tabel 17 Dimensiuni comparative tabele organizate indexat şi tabele heap cu index asociat.

Număr înregistrări	Heap (Mo)	Index pe PK (Mo)	Dimensiune totală (Mo)	IOT (Mo)
1 525 176	114	37	151	172
2 567 853	191	63	254	285
4 543 667	315	136	451	467

Grafic 34. Comparatie tabela cu index asociat - tabela organizata indexat (spatiu de memorare). Fisier date MARFURI.



Nici în SQL Server 2000 nu se obține câştig în ceea ce priveşte spațiul de memorare dacă se utilizează un index clusterat față de o structură heap pentru tabela de bază.

Pornind de la rezultatele testelor efectuate se pot trage următoarele concluzii:

- nu se obține câştig la nivelul spațiului de memorare dacă se utilizează tabele organizate indexat comparativ cu situația în care se utlizează o tabelă având structură heap şi un index asociat construit pe cheia primară a relației care stă la baza tabelei;

- în acest caz există chiar o creştere a necesarului de spațiu de memorare care se situează în jurul valorii de 10% în cazul folosirii stucturii de tip IOT față de situația folosirii structurii heap cu index asociat.

3.3.5.2. Câştigul în *timp de răspuns* pentru operațiile de interogare. Concluzii.

Tabela MARFURI a fost construită atât având o structură de tip heap cât şi o structură de tip tabelă organizată indexat. In situația de tabelă heap, i s-a asociat acesteia şi un index clasic de tip arbore B având ca şi cheie de indexare cheia primară CODP a tabelei. Asupra tabelelor au fost executate interogări de forma celor următoare:

1.
```
SELECT *
FROM MARFURI
WHERE CODP='ZAHA----00';
```

2.
```
SELECT *
FROM MARFURI
WHERE CODP IN ('ZAHA----00',
        'RAD4----', 'ZAHA----01');
```

3.
```
SELECT *
FROM  MARFURI
WHERE GRUPA='BOMBOANE ';
```

4.
```
SELECT DISTINCT COD_FURN
FROM  MARFURI
WHERE CODP='ZAHA----00' AND
                CONT='111111 ';
```

Pentru situația implementării în Oracle 9*i* interogările de mai sus au fost executate prin intermediul unor scripturi PL/SQL, care au permis transmiterea ca parametrii a valorilor folosite în clauza WHERE a interogării. Rezultatele timpilor de execuție pentru fiecare caz de apel în parte au fost colectate în tabela REZULTATE. Mai jos sunt prezentate scripturile pentru operația 1 şi 4.

--MarfuriIOTSelect1.sql
```
DECLARE
 v_NrOp NUMBER(3);
 v_Operatie VARCHAR(200);
 v_Start NUMBER(10,0);
 v_End NUMBER(10,0);
 v_Timp NUMBER(10,0);
 v_Valoare VARCHAR(50);
```

```
 v_NrInreg NUMBER(12,0);
 v_Index CHAR(1);
 v_TipIndex VARCHAR(20);
 v_NrIndex NUMBER(2,0);
 v_Marfuri Marfuri%ROWTYPE;
 v_CodP Marfuri.CodP%TYPE;
 v_TotalInreg NUMBER(15,0);
 CURSOR c_Marfuri IS
   SELECT *
   FROM Marfuri
   WHERE CodP=v_CodP;
 CURSOR c_TotalInreg IS
   SELECT count(*)
   FROM Marfuri;
BEGIN
 v_NrOp:=&NrOp;
 OPEN c_TotalInreg;
 FETCH c_TotalInreg INTO v_TotalInreg;
 CLOSE c_TotalInreg;
 v_CodP:='&CodP';
 v_Index:= '&Index';
 v_TipIndex:= '&TipIndex';
 v_NrIndex:= '&NrIndex';
 v_Operatie:='SELECT * FROM Marfuri WHERE CodP=';
 v_Start:=DBMS_UTILITY.GET_TIME;
 OPEN c_Marfuri;
 v_End:=DBMS_UTILITY.GET_TIME;
 v_Timp:=(v_End-v_Start)*10;
 LOOP
    FETCH c_Marfuri INTO v_Marfuri;
    EXIT WHEN c_Marfuri%NOTFOUND;
 END LOOP;
 v_NrInreg:=c_Marfuri%ROWCOUNT;
 CLOSE c_Marfuri;
 v_Valoare:=v_CodP
 INSERT INTO Rezultate(NrOp, Operatie, TotalInreg, InregPrel,Timp, Valoare, Index, TipIndex,
        NrIndex) VALUES (v_NrOp, v_Operatie, v_TotalInreg, v_NrInreg, v_Timp, v_Valoare,
        v_Index, v_TipIndex, null);
 COMMIT;
END;
```

```
--MarfuriIOTSelect4.sql
DECLARE
 v_NrOp NUMBER(3);
 v_Operatie VARCHAR(200);
 v_Start NUMBER(10,0);
 v_End NUMBER(10,0);
 v_Timp NUMBER(10,0);
 v_Valoare VARCHAR(50);
 v_NrInreg NUMBER(12,0);
 v_Index CHAR(1);
 v_TipIndex VARCHAR(20);
 v_NrIndex NUMBER(2,0);
 v_Marfuri Marfuri.Cod_Oper%TYPE;
 v_CodP Marfuri.CodP%TYPE;
 v_Cont Marfuri.Cont%TYPE;
 v_TotalInreg NUMBER(15,0);
 CURSOR c_Marfuri IS
  SELECT DISTINCT Cod_Furn
  FROM Marfuri
  WHERE CodP=v_CodP AND Cont=v_Cont;
 CURSOR c_TotalInreg IS
  SELECT count(*)
   FROM Marfuri;
BEGIN
 v_NrOp:=&NrOp;
 OPEN c_TotalInreg;
 FETCH c_TotalInreg INTO v_TotalInreg;
 CLOSE c_TotalInreg;
 v_CodP:='&CodP';
 v_Cont:='&Cont';
 v_Index:= '&Index';
 v_TipIndex:= '&TipIndex';
 v_NrIndex:= '&NrIndex';
 v_Operatie:='SELECT DISTINCT Cod_Furn FROM Marfuri WHERE CodP= AND Cont=';
 v_Start:=DBMS_UTILITY.GET_TIME;
 OPEN c_Marfuri;
 v_End:=DBMS_UTILITY.GET_TIME;
 v_Timp:=(v_End-v_Start)*10;
 LOOP
    FETCH c_Marfuri INTO v_Marfuri;
```

```
    EXIT WHEN c_Marfuri%NOTFOUND;
  END LOOP;
  v_NrInreg:=c_Marfuri%ROWCOUNT;
  CLOSE c_Marfuri;
  v_Valoare:=v_CodP||' AND '||v_Cont
  INSERT INTO Rezultate(NrOp, Operatie, TotalInreg, InregPrel,Timp, Valoare, Index, TipIndex,
        NrIndex) VALUES (v_NrOp, v_Operatie, v_TotalInreg, v_NrInreg, v_Timp, v_Valoare,
        v_Index, v_TipIndex, null);
  COMMIT;
END;
```

Scripturile PL/SQL pentru operația 2 şi 3 se găsesc în anexa 1. Pentru cea de a patra interogare s-a considerat o cheie primară compusă din atributele CODP şi CONT pentru a se studia comportamentul şi în cazul unor chei de indexare compuse.

Pentru implementarea în SQL Server2000 s-au folosit proceduri stocate, care de asemenea au permis transmiterea unor valori ca parametrii de intrare şi colectarea rezultatelor testelor în tabela REZULTATE. Spre exemplu, pentru cele două operații de interogare de mai sus s-au folosit procedurile stocate MarfuriIOTSelect1 şi MarfuriIOTSelect4 de mai jos. Celelalte două proceduri se găsesc în anexa 1.

```
/*procedura stocată MarfuriIOTSelect1*/
CREATE PROCEDURE dbo.MarfuriIOTSelect1
(       @nrOp NUMERIC(18,0), @vCodP VARCHAR(10), @index CHAR(1), @tipIndex
VARCHAR(20), @nrIndex NUMERIC(2,0)
)
AS
SET NOCOUNT ON
BEGIN TRANSACTION
        DECLARE @op VARCHAR(200),@t1 NUMERIC(18,0),@t2 NUMERIC(18,0),@s1
INT,@s2 INT,@total NUMERIC(18,0),@inreg NUMERIC(18,0),@dif NUMERIC(18,6)
        SET @op = 'SELECT * FROM Marfuri WHERE CodP='
        SELECT @total=count(*) FROM Marfuri
        SELECT @inreg = count(*)  FROM Marfuri WHERE CodP=@vCodP
        SELECT @t1=datepart(ms,getdate()),@s1=datepart(s,getdate())
        SELECT *  FROM Marfuri WHERE CodP=@vCodP
        SELECT @t2=datepart(ms,getdate()),@s2=datepart(s,getdate())
        if @s2>@s1
                SET @dif=(@s2*1000+@t2)-(@s1*1000+@t1)
```

```
            else
                    SET @dif=@t2-@t1
            INSERT Rezultate VALUES
                    (@nr,@op,@total,@inreg,@dif,@vCodP,@index,@tipIndex,@nrIndex)
    if @@error<>0
    BEGIN
            ROLLBACK TRANSACTION
            RAISERROR ('Operatiunea nu s-a putut realiza din cauza unei erori!!',16,1) with NOWAIT
    END
    ELSE
    BEGIN
            COMMIT TRANSACTION
            RETURN(0)
    END
    GO

    /*procedura stocată MarfuriIOTSelect4*/
    CREATE PROCEDURE dbo.MarfuriIOTSelect4
    (       @nrOp NUMERIC(18,0), @vCodP VARCHAR(10), @vCont CHAR(11), @index CHAR(1),
@tipIndex VARCHAR(20), @nrIndex NUMERIC(2,0)
    )
    AS
        SET NOCOUNT ON
        BEGIN TRANSACTION
                DECLARE @op VARCHAR(200),@t1 NUMERIC(18,0),@t2 NUMERIC(18,0),@s1
INT,@s2 INT,@total NUMERIC(18,0),@inreg NUMERIC(18,0),@dif NUMERIC(18,6), @valoare
VARCHAR(50)
                SET @op = 'SELECT DISTINCT Cod_Furn FROM Marfuri WHERE CodP= AND Cont= '
                SELECT @total=count(*) FROM Marfuri
                CREATE TABLE #t (Cod_Furn VARCHAR(7))
                INSERT #t SELECT DISTINCT Cod_Furn FROM Marfuri WHERE CodP=@vCodP AND
Cont=@vCont
                SELECT @inreg = count(*) FROM #t
                SELECT @t1=datepart(ms,getdate()),@s1=datepart(s,getdate())
                SELECT DISTINCT Cod_Furn FROM Marfuri WHERE CodP=@vCodP AND
Cont=@vCont
                SELECT @t2=datepart(ms,getdate()),@s2=datepart(s,getdate())
                if @s2>@s1
                        SET @dif=(@s2*1000+@t2)-(@s1*1000+@t1)
                else
```

```
                SET @dif=@t2-@t1
            SET @valoare = @vCodP +' AND '+@vCont
            INSERT Rezultate VALUES
                (@nr,@op,@total,@inreg,@dif,@valoare,@index,@tipIndex,@nrIndex)
    if @@error<>0
    BEGIN
        ROLLBACK TRANSACTION
        RAISERROR ('Operatiunea nu s-a putut realiza din cauza unei erori!!',16,1) with NOWAIT
    END
    ELSE
    BEGIN
        COMMIT TRANSACTION
        RETURN(0)
    END
GO
```

Rezultatele obținute în urma testelor au fost centralizate în tabela REZULTATE_FINALE prin rularea scriptului RezultateFinale3.sql sau a procedurii stocate RezultateFinale3 de mai jos:

```
--RezultateFinale3.sql
BEGIN
  INSERT INTO Rezultate_Finale(NrOp,Operatie,TotalInreg,InregPrel,Timp,Index,TipIndex)
    SELECT NrOp, Operatie, TotalInreg, AVG(InregPrel), AVG(Timp), Index, TipIndex
    FROM Rezultate
    WHERE Index='D' AND TipIndex='Primar'
    GROUP BY NrOp;
  INSERT INTO Rezultate_Finale(NrOp,Operatie,TotalInreg,InregPrel,Timp,Index,TipIndex)
    SELECT NrOp, Operatie, TotalInreg, AVG(InregPrel), AVG(Timp), Index, TipIndex
    FROM Rezultate
    WHERE Index='D' AND TipIndex='Secundar'
    GROUP BY NrOp;
INSERT INTO Rezultate_Finale(NrOp,Operatie,TotalInreg,InregPrel,Timp,Index,TipIndex)
    SELECT NrOp, Operatie, TotalInreg, AVG(InregPrel), AVG(Timp), Index, TipIndex
    FROM Rezultate
    WHERE Index='D' AND TipIndex='IOTSecundar'
    GROUP BY NrOp;
  INSERT INTO Rezultate_Finale(NrOp,Operatie,TotalInreg,InregPrel,Timp,Index,TipIndex)
    SELECT NrOp, Operatie, TotalInreg, AVG(InregPrel), AVG(Timp), Index, TipIndex
```

```
    FROM Rezultate
    WHERE Index='N' AND TipIndex='IOT'
    GROUP BY NrOp;
  INSERT INTO Rezultate_Finale(NrOp,Operatie,TotalInreg,InregPrel,Timp,Index,TipIndex)
   SELECT NrOp, Operatie, TotalInreg, AVG(InregPrel), AVG(Timp), Index, TipIndex
    FROM Rezultate
    WHERE Index='N' AND TipIndex='Heap'
    GROUP BY NrOp;
  COMMIT;
END;
```

/*procedura stocată RezultateFinale3*/

```
CREATE PROCEDURE dbo.RezultateFinale3( )
AS
SET NOCOUNT ON
BEGIN TRANSACTION
  INSERT INTO Rezultate_Finale(NrOp,Operatie,TotalInreg,InregPrel,Timp,Index,TipIndex)
    SELECT NrOp, Operatie, TotalInreg, AVG(InregPrel), AVG(Timp), Index, TipIndex
    FROM Rezultate
    WHERE Index='D' AND TipIndex='Primar'
    GROUP BY NrOp
  INSERT INTO Rezultate_Finale(NrOp,Operatie,TotalInreg,InregPrel,Timp,Index,TipIndex)
    SELECT NrOp, Operatie, TotalInreg, AVG(InregPrel), AVG(Timp), Index, TipIndex
    FROM Rezultate
    WHERE Index='D' AND TipIndex='Secundar'
    GROUP BY NrOp
  INSERT INTO Rezultate_Finale(NrOp,Operatie,TotalInreg,InregPrel,Timp,Index,TipIndex)
    SELECT NrOp, Operatie, TotalInreg, AVG(InregPrel), AVG(Timp), Index, TipIndex
    FROM Rezultate
    WHERE Index='D' AND TipIndex='IOTSecundar'
    GROUP BY NrOp
  INSERT INTO Rezultate_Finale(NrOp,Operatie,TotalInreg,InregPrel,Timp,Index,TipIndex)
    SELECT NrOp, Operatie, TotalInreg, AVG(InregPrel), AVG(Timp), Index, TipIndex
    FROM Rezultate
    WHERE Index='N' AND TipIndex='IOT'
    GROUP BY NrOp
  INSERT INTO Rezultate_Finale(NrOp,Operatie,TotalInreg,InregPrel,Timp,Index,TipIndex)
    SELECT NrOp, Operatie, TotalInreg, AVG(InregPrel), AVG(Timp), Index, TipIndex
    FROM Rezultate
    WHERE Index='N' AND TipIndex='Heap'
```

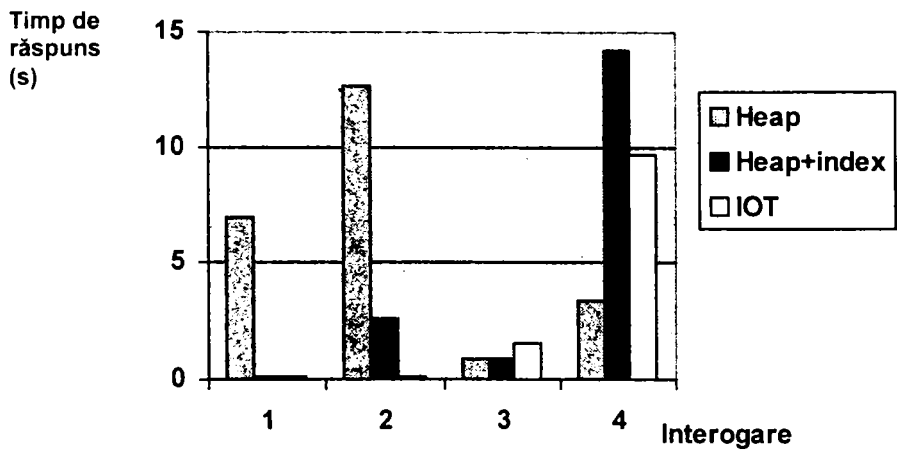
```
    GROUP BY NrOp
if @@error<>0
BEGIN
        ROLLBACK TRANSACTION
        RAISERROR ('Operatiunea nu s-a putut realiza din cauza unei erori!!',16,1) with NOWAIT
END
ELSE
BEGIN
        COMMIT TRANSACTION
        RETURN(0)
END
GO
```

O parte a rezultatelor finale obținute in urma acestor tipuri de teste pot fi consultate în tabela REZULTATE_FINALE din anexa 1. Interogările au fost formulate asupra unor tabele având un număr de înregistrări egal cu 1.525.176, 2.567.853, 4.543.667. Comportamentul s-a dovedit a fi acelaşi indiferent de numărul înregistrărilor din tabelă. Rezultatele testelor efectuate pentru cazul a 1.525.176 înregistrări sunt prezentate în tabela 18 şi graficul 35.

Tabel 18 Timpii de răspuns la interogări tabele Heap - IOT.

Interogare	Heap	Heap+Index	Heap+Index+Index secundar	IOT	IOT+Index secundar
1	6,99	0,09	-	0,05	-
2	12.698	2,653	-	0,06	-
3	0,871	0,871	0.31	1,503	0,461
4	3,425	14,204	-	9,704	-

Grafic 35. Comparatie tabela cu index asociat - tabela organizata indexat (timp de raspuns). SELECT



Analizând datele obținute în urma testelor efectuate se pot trage următoarele concluzii:

- în situația în care căutările se efectuează în funcție de valori ale cheii primare, o structură de tip IOT este mai eficientă decât o structură de tip heap care are asociată un index pe cheia primară, dar ambele situații sunt mai performante decât situația în care stuctura ar fi doar de tip heap;
- dacă operațiile de interogare se efectuează după criterii diferite de cheia primară, structura de tip IOT este mai neperformantă decât structura de tip heap care are asociat un index pe cheia primară:

    - pentru a îmbunătăți performanța structurii IOT este indicat să se construiască un index secundar care va fi folosit de criteriul de căutare;
    - în situația în care se foloseşte un index secundar este totuşi mai avantajoasă o structură de tip heap cu index asociat decât una de tip IOT;

- dacă cheia de indexare este compusă din mai multe atribute şi în cadrul operațiilor de interogare se folosesc criterii de căutare dependente de atributele de pe pozițiile 2, 3.... ale cheii primare performanțele se înrăutățesc foarte mult;

    - înrăutățirea performanțelor este ceva mai mică în cazul structurii IOT decât în cazul unei structuri heap cu index asociat;

- tabelele organizate indexat sunt mai performante în cazul interogărilor care implică condiții de căutare bazate pe egalitate de valori sau pentru căutările bazate pe intervale de valori.

3.3.5.3. Câştigul în *timp de răspuns* pentru operațiile de actualizare. Concluzii.

Au fost analizați comparativ şi timpii de execuție ai operațiilor UPDATE şi DELETE asupra tabelei de date MARFURI care este organizată fie ca tabelă heap cu index asociat fie ca tabelă organizată indexat. Operațiile care au fost aplicate tabelei sunt asemănătoare cu următoarele:

1.

```
UPDATE MARFURI
SET CODP='ZAHAR---00'
WHERE CODP='ZAHA----00';
```

2.

```
UPDATE MARFURI
SET GRUPA='ZAHAR'
WHERE CODP IN ('ZAHA----00',
```

```
'ZAHA----01', 'ZAHA----02');
```

3.

```
UPDATE MARFURI
SET GRUPA='ZAHAR'
WHERE GRUPA='zahar ';
```

4.

```
DELETE FROM MARFURI
WHERE CODP='ZAHA----00';
```

5.

```
DELETE FROM MARFURI
WHERE GRUPA='ZAHAR';
```

Şi în acest caz pentru implementare în Oracle 9*i* s-au folosit scripturi PL/SQL pentru execuția operațiilor de actualizare, iar pentru implementarea în SQL Server2000 proceduri stocate. Pentru operațiile 1 şi 2 scripturile şi procedurile se găsesc în anexa 1. Pentru operația 5 s-a folosit scriptul MarfuriStergere în implementare Oracle 9*i* sau procedura MarfuriStergere pentru implemntarea în SQL Server2000. Pentru operațiile 3 şi 4 s-au folosit următoarele scripturi şi proceduri:

```
--MarfuriIOTModificare3.sql
DECLARE
 v_NrOp NUMBER(3);
 v_Operatie VARCHAR(200);
 v_Start NUMBER(10,0);
 v_End NUMBER(10,0);
 v_Timp NUMBER(10,0);
 v_NrInreg NUMBER(12,0);
 v_Vechi_Grupa Marfuri.Grupa%TYPE;
 v_Nou_Grupa Marfuri.Grupa%TYPE;
 v_Index CHAR(1);
 v_TipIndex VARCHAR(20);
 v_NrIndex NUMBER(2,0);
 v_TotalInreg NUMBER(15,0);
 CURSOR c_TotalInreg IS
   SELECT count(*)
   FROM Marfuri;
CURSOR c_NrInreg IS
   SELECT count(*)
   FROM Marfuri
   WHERE Grupa=v_Vechi_Grupa;
BEGIN
```

```
v_NrOp:=&NrOp;
OPEN c_TotalInreg;
FETCH c_TotalInreg INTO v_TotalInreg;
CLOSE c_TotalInreg;
v_Vechi_Grupa:=&GrupaVeche;
v_Nou_Grupa:=&GrupaNoua;
v_Index:= '&Index';
v_TipIndex:= '&TipIndex';
v_NrIndex:=&NrIndex;
OPEN c_NrInreg
FETCH c_NrInreg INTO v_NrInreg;
CLOSE c_NrInreg;
v_Operatie:='UPDATE Marfuri SET Grupa= WHERE Grupa=';
v_Start:=DBMS_UTILITY.GET_TIME;
UPDATE Marfuri SET Grupa=v_Nou_Grupa WHERE Grupa=v_Vechi_Grupa;
v_End:=DBMS_UTILITY.GET_TIME;
v_Timp:=(v_End-v_Start)*10;
INSERT INTO Rezultate(NrOp, Operatie, TotalInreg, InregPrel,Timp, Valoare, Index, TipIndex,
      NrIndex) VALUES (v_NrOp, v_Operatie, v_TotalInreg, v_NrInreg, v_Timp,
      v_Vechi_Grupa, v_Index, v_TipIndex, v_NrIndex);
COMMIT;
END;
```

--MarfuriIOTStergere.sql

```
DECLARE
 v_NrOp NUMBER(3);
 v_Operatie VARCHAR(200);
 v_Start NUMBER(10,0);
 v_End NUMBER(10,0);
 v_Timp NUMBER(10,0);
 v_NrInreg NUMBER(12,0);
 v_CodP Marfuri.CodP%TYPE;
 v_Index CHAR(1);
 v_TipIndex VARCHAR(20);
 v_NrIndex NUMBER(2,0);
 v_TotalInreg NUMBER(15,0);
 CURSOR c_TotalInreg IS
   SELECT count(*)
   FROM Marfuri;
CURSOR c_NrInreg IS
```

```
  SELECT count(*)
  FROM Marfuri
  WHERE CodP=v_CodP;
BEGIN
 v_NrOp:=&NrOp;
 OPEN c_TotalInreg;
 FETCH c_TotalInreg INTO v_TotalInreg;
 CLOSE c_TotalInreg;
 v_CodP:= '&CodP';
 v_Index:= '&Index';
 v_TipIndex:= '&TipIndex';
 v_NrIndex:=&NrIndex;
 v_Operatie:='DELETE FROM Marfuri WHERE CodP= ';
 OPEN c_NrInreg;
 FETCH c_NrInreg INTO v_NrInreg;
 CLOSE c_NrInreg;
 v_Start:=DBMS_UTILITY.GET_TIME;
 DELETE FROM Marfuri WHERE CodP=v_CodP;
 v_End:=DBMS_UTILITY.GET_TIME;
 v_Timp:=(v_End-v_Start)*10;
 INSERT INTO Rezultate(NrOp, Operatie, TotalInreg, InregPrel,Timp, Valoare, Index, TipIndex,
        NrIndex) VALUES (v_NrOp, v_Operatie, v_TotalInreg, v_NrInreg, v_Timp,
        v_CodP, v_Index, v_TipIndex, v_NrIndex);
  COMMIT;
END;
```

/*procedura stocată MarfuriIOTModificare3*/

```
CREATE PROCEDURE dbo.MarfuriIOTModificare3
(       @nrOp NUMERIC(18,0), @vVecheGrupa VARCHAR(8), @vNouaGrupa VARCHAR(8),
@index CHAR(1), @tipIndex VARCHAR(20), @nrIndex NUMERIC(2,0)
)
AS
SET NOCOUNT ON
BEGIN TRANSACTION
        DECLARE @op VARCHAR(200),@t1 NUMERIC(18,0),@t2 NUMERIC(18,0),@s1
INT,@s2 INT,@total NUMERIC(18,0),@inreg NUMERIC(18,0),@dif NUMERIC(18,6)
        SET @op = 'UPDATE Marfuri SET Grupa= WHERE Grupa='
        SELECT @total=count(*) FROM Marfuri
        SELECT @t1=datepart(ms,getdate()),@s1=datepart(s,getdate())
        UPDATE Marfuri SET Grupa=@vNouaGrupa WHERE Grupa=@vVecheGrupa
```

```
            SELECT @t2=datepart(ms,getdate()),@s2=datepart(s,getdate())
            if @s2>@s1
                  SET @dif=(@s2*1000+@t2)-(@s1*1000+@t1)
            else
                  SET @dif=@t2-@t1
            SELECT @inreg = count(*)  FROM  Marfuri WHERE Grupa=@vNouaGrupa
            INSERT Rezultate VALUES
                  (@nr,@op,@total,@inreg,@dif,@vVecheGrupa,@index,@tipIndex,@nrIndex)
      if @@error<>0
      BEGIN
            ROLLBACK TRANSACTION
            RAISERROR ('Operatiunea nu s-a putut realiza din cauza unei erori!!',16,1) with NOWAIT
      END
      ELSE
      BEGIN
            COMMIT TRANSACTION
            RETURN(0)
      END
GO

      /*procedura stocată MarfuriIOTStergere*/
      CREATE PROCEDURE dbo.MarfuriIOTStergere
      (     @nrOp NUMERIC(18,0), @vCodP VARCHAR(10), @index CHAR(1), @tipIndex
VARCHAR(20), @nrIndex NUMERIC(2,0)
      )
      AS
      SET NOCOUNT ON
      BEGIN TRANSACTION
            DECLARE @op VARCHAR(200),@t1 NUMERIC(18,0),@t2 NUMERIC(18,0),@s1
INT,@s2 INT,@total NUMERIC(18,0),@inreg NUMERIC(18,0),@dif NUMERIC(18,6)
            SET @op = 'DELETE FROM Marfuri WHERE CodP= '
            SELECT @total=count(*) FROM Marfuri
            SELECT @inreg = count(*)  FROM  Marfuri WHERE CodP=@vCodP
            SELECT @t1=datepart(ms,getdate()),@s1=datepart(s,getdate())
            DELETE FROM Marfuri WHERE CodP=@vCodP
            SELECT @t2=datepart(ms,getdate()),@s2=datepart(s,getdate())
            if @s2>@s1
                  SET @dif=(@s2*1000+@t2)-(@s1*1000+@t1)
            else
                  SET @dif=@t2-@t1
```

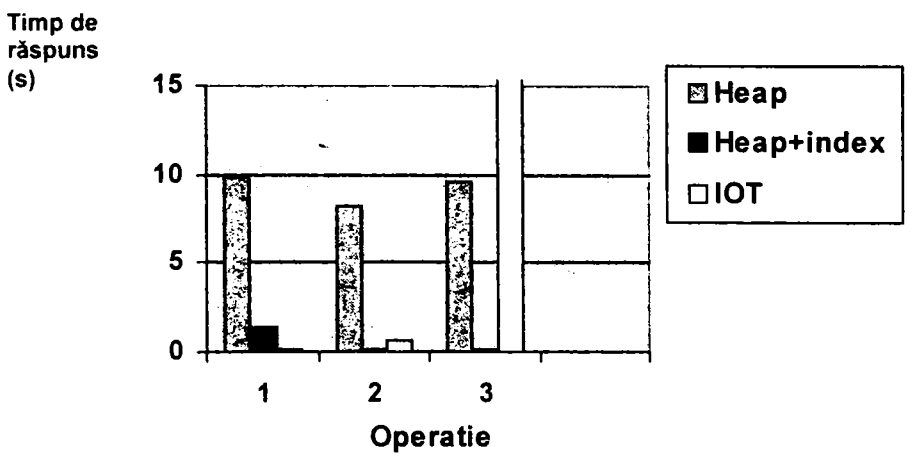
```
        INSERT Rezultate VALUES
                (@nr,@op,@total,@inreg,@dif,@vCodP,@index,@tipIndex,@nrIndex)
if @@error<>0
BEGIN
        ROLLBACK TRANSACTION
        RAISERROR ('Operatiunea nu s-a putut realiza din cauza unei erori!!',16,1) with NOWAIT
END
ELSE
BEGIN
        COMMIT TRANSACTION
        RETURN(0)
END
GO
```

Pentru operațiile de la punctul 3 şi 5 există index secundar construit pe cheia de indexare GRUPA atât pentru situația tabelei heap cât şi a celei organizate indexat. Rezultatele testelor pentru operațiile de tip UPDATE se găsesc în tabelul 19, iar cele pentru operațiile de tip DELETE în tabelul 20. Evoluția timpilor de execuție se poate observa în graficele 36 şi 37.

Tabel 19. Timpii de răspuns ai operațiilor UPDATE tabele Heap - IOT.

Operație	Număr înregistrări 1.525.176			Număr de înregistrări 2.567.853			Număr de înregistrări 4.543.667		
	Heap	Heap+Index	IOT	Heap	Heap+Index	IOT	Heap	Heap*Index	IOT
1	9,874	1,402	0,14	13,91	0,962	0,751	22,893	1,673	1,492
2	8,222	0,15	0,571	11,677	0,16	0,391	19,158	1,152	0,521
3	9,575	0.08	21,671	14,805	39,157	55,831	19,038	68,769	82,989

Grafic 36. Comparatie tabela cu index asociat - tabela organizata indexat (timp de raspuns). UPDATE

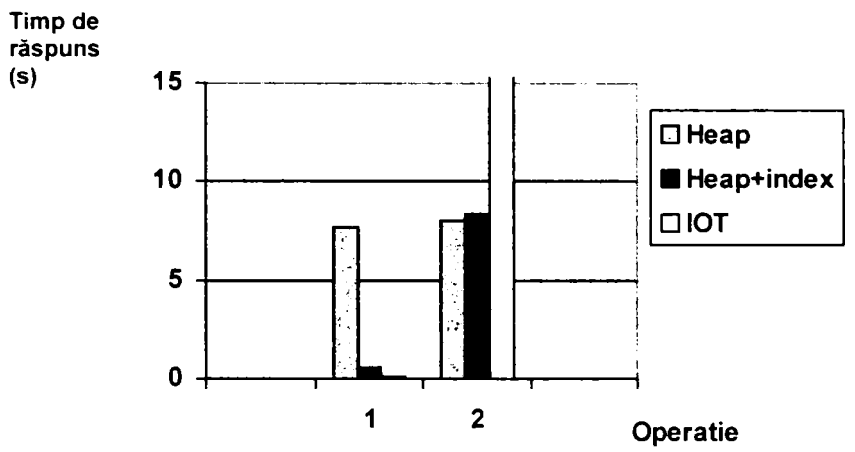


Tabel 20. Timpii de răspuns ai operațiilor DELETE tabele Heap - IOT.

Operație	Număr înregistrări 1.525.176	Număr de înregistrări 2.567.853	Număr de înregistrări 4.543.667

	Heap	Heap+Index	IOT	Heap	Heap+Index	IOT	Heap	Heap*Index	IOT
1	7.681	0.521	0.11	11.726	0.39	0.421	19.278	1.271	0.681
2	8.05	8.332	16.283	11.727	11.586	29.903	72.584	37.834	47.799

Grafic 37. Comparatie tabela cu index asociat - tabela organizata indexat (timp de raspuns). DELETE



Concluziile care se pot trage în urma testelor efectuate sunt următoarele:

- în cazul operațiilor UPDATE o structură de tabelă organizată indexat este mai performantă doar în situația în care condiția de căutare din clauza WHERE este aceeaşi cu cheia primară care stă la baza construirii tabelei organizate indexat;
  - câştigul în timp de execuție se reduce odată cu creşterea numărului de înregistrări din fişierul de date;
- pentru toate celelalte situații este mai avantajoasă o structură de tip heap pentru tabelă dacă ea are asociat un index pe condiția de căutare din clauza WHERE a operației UPDATE;
- pentru situațiile în care condiția de căutare din clauza WHERE a operației UPDATE este diferită de cea care stă la baza construirii tabelei organizate indexat se obține o degradare foarte mare a performanței față de situația cu tabelă heap şi index asociat;
- în cazul operațiilor DELETE o structură de tabelă organizată indexat este mai performantă, de asemenea, doar în situația în care condiția de căutare din clauza WHERE este aceeaşi cu cheia primară care stă la baza construirii tabelei organizate indexat;

- câştigul în timp de execuție se reduce odată cu creşterea numărului de înregistrări din fişierul de date;

- pentru toate celelalte situații este mai avantajoasă o structură de tip heap pentru tabelă dacă ea are asociat un index pe condiția de căutare din clauza WHERE a operației DELETE;
- tabelele organizate indexat sunt mai performante în cazul operațiilor de actualizare care implică condiții de căutare bazate pe egalitate de valori sau pentru căutările bazate pe intervale de valori relative la valorile cheii primare.

Ca o concluzie generală se poate spune că în situația în care căutările se efectuează în funcție de valori ale cheii primare, o structură de tip IOT este mai eficientă decât o structură de tip heap care are asociată un index pe cheia primară, dar ambele situații sunt mai performante decât situația în care stuctura ar fi doar de tip heap. In schimb, dacă operațiile de manipulare se efectuează după criterii diferite de cheia primară, structura de tip IOT este mai neperformantă decât structura de tip heap care are asociat un index pe cheia primară. Performanțele structurii IOT se pot îmbunătăți dacă se construieşte un index secundar pe condiția de căutare din clauza WHERE, dar ele rămân în continuare inferioare celor obținute cu o structră de tip heap care are indecşi asociați.

3.3.6. Comparație între tabele heap şi tabele partiționate

Tabelele partiționate permit descompunerea unor structuri mari de date în structuri mult mai mici şi mai uşor de gestionat. Partiționarea poate ajuta considerabil la creşterea performanței în cazul bazelor de date foarte mari. Operațiile de manipulare pot fi aplicate doar asupra acelor partiții care conțin datele necesare. Partițiile care nu pot conține datele cerute sunt eliminate din start din domeniul de căutare. De asemenea, partiționarea tabelei poate îmbunătăți considerabil timpii de execuție ai operațiilor de join multiplu între tabele, dacă tabelele sunt partiționate după aceeaşi cheie de partiționare. Un alt avantaj al partiționării tabelelor este acela că permite execuția paralelă a operațiilor DML, reducând timpul de răspuns pentru operațiile care prelucrează masiv date în baze de date mari, precum cele din domeniul sistemelor de suport al deciziilor sau al magaziilor de date.

Dintre sistemele analizate doar Oracle 9*i* şi DB2 permit partiționarea tabelelor. În Oracle 9*i* există mai multe variante de partiționare [Oracl 02], [Lejeu 02-1], şi anume: list,

range, hash, range-hash, range-list, pe când în DB2 este posibilă doar partiționarea de tip hash [35, 49]. Oracle 9*i* permite, de asemenea, crearea de indecşi la nivelul partițiilor. Aceştia pot fi indecşi locali la nivelul fiecărei partiții în parte, sau indecşi globali la nivelul întregii tabele, care la rândul lor pot fi partiționați sau nu [Lejeu 02-1]. În DB2 se pot construi doar indecşi locali [Lejeu 02-1]. SQL Server 2000 nu permite partiționarea tabelelor, ci doar o formă surogat de partiționare care constă de fapt în crearea unor vizualizări globale, care să reunească într-o tabelă virtuală prin intermediul clauzei UNION ALL date preluate din mai multe tabele independente numite tabele membru [Lejeu 02-2]. Tabelele membru sunt administrate separat independent una de alta, astfel încât în cazul lui SQL Server 2000 nu poate fi vorba de îmbunătățirea performanțelor prin folosirea partiționării.

3.3.6.1. Câştigul în *flexibilitate*

Partiționarea este foarte utilă pentru că oferă o mai mare flexibilitate pentru structura tabelei. Fiecare partiție poate avea proprii săi parametri de stocare. Acest lucru este foarte important pentru că se poate controla astfel mărimea spațiului de memorare rezervată fiecărei partiții în parte. Astfel, o partiție poate avea câteva înregistrări în timp ce o alta poate avea milioane de înregistrări. În Oracle 9*i* partiționarea putând fi şi de tip range sau list este mai uşor să se previzioneze numărul de înregistrări ale fiecărei partiții şi să se dimensioneze corect spațiul de memorare necesar acestora. In DB2 fiind posibilă doar partiționarea de tip hash este mai dificil de făcut acest lucru.

3.3.6.2. Câştigul în *disponibilitate*

Partiționarea este, de asemenea, importantă pentru că măreşte disponibilitatea datelor. Dacă o partiție a unei tabele este inaccesibilă dintr-un anumit motiv, utilizatorii pot accesa în continuare celelalte partiții ale tabelei. Se pot efectua operații de inserare, modificare, ştergere şi selecție în aceste partiții. Se poate, de asemenea, bloca accesul la o partiție în timp ce restul partițiilor sunt disponibile. Fiecare partiție a unei tabele poate fi stocată în zone diferite pe disc. Acest lucru este benefic în cazul în care una dintre aceste zone este inaccesibilă.

3.3.6.3. Câştigul în *timp de răspuns* pentru operațiile de interogare. Concluzii.

Testele au fost efectuate pe tabela VANZARI a cărei dimensiune a variat de la 100 Mo până la 3 Go. S-a observat că doar de la mărimi mai mari de 1 Go ale tabelei este sesizabil câştigul în timp de răspuns la operațiile de interogare. Pentru mărimi mai mici ale tabelei, partiționarea este utilă doar pentru că oferă o mai mare disponibilitate a datelor.

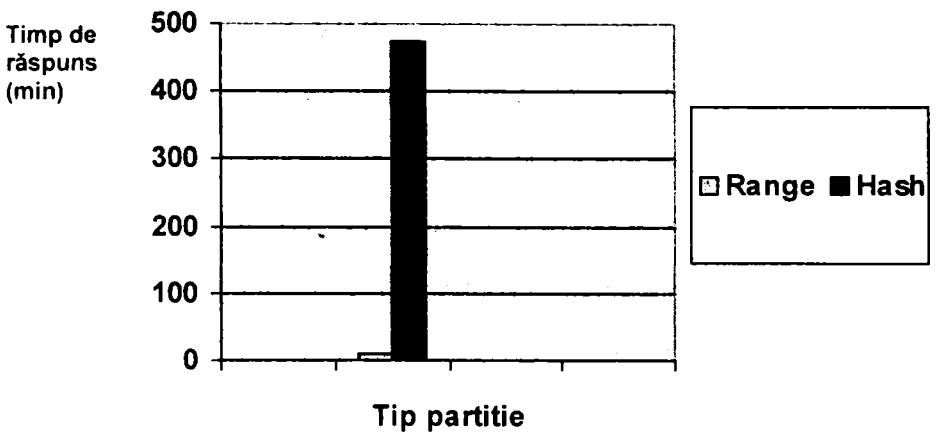
De asemenea, performanțele cresc dacă tabela partiționată poate fi accesată de mai multe drivere care citesc şi/sau scriu în paralel.

În continuare vor fi prezentate comparativ datele obținute pe tabela VANZARI care a fost partiționată folosind o partiție de tip range şi o partiție de tip hash. Tabela VANZARI conținea datele vânzărilor efectuate pe o perioadă de trei ani. Partiția de tip range a fost creată pe intervale de timp de o lună, folosind coloana TIMP, în timp ce partiția de tip hash a fost creată pe coloana COD_CLIENT. S-a realizat o încărcare incrementală de date în tabelă. Datele lunii celei mai vechi au fost înlocuite cu unele noi. În tabelul 21 sunt prezentați comparativ timpii necesari efectuării acestei operații.

Tabel 21. Comparație partiționare range - partiționare hash. INSERT

Tip partiție	Timp adăugare date noi hh:mm:ss	Timp ştergere date vechi hh:mm:ss	Total hh:mm:ss
Range	00:09:19	00:00:01	00:09:20
Hash	07:05:03	00:47:51	07:52:54

Grafic 38. Comparatie partitie Range - Hash. INSERT



După cum se observă şi în graficul 38 partiția de tip range este mult mai performantă pentru acest tip de operație.

Asupra celor două tabele s-au efectuat teste şi pentru a evalua timpii de răspuns la interogări. Au fost testate câteva interogări de tip *„star transformation"* foarte utilizate în domeniul magaziilor de date.

1.

```
SELECT M.DENP, SUM(V.CANT)
FROM VANZARI V, MARFURI M, CLIENTI C, TIMPI T
WHERE V.CODP=M.CODP AND V.COD_CLIENT=C.COD_CLIENT
        AND V.CODT=T.CODT AND T.LUNA='MAY'
        AND C.GRUPA='ALIMENTE' AND M.GRUPA='ZAHAR'
GROUP BY M.DENP;
```

2.

```
SELECT M.DENP, SUM(V.CANT)
FROM VANZARI V, MARFURI M, CLIENTI C
WHERE V.CODP=M.CODP AND V.COD_CLIENT=C.COD_CLIENT
        AND C.GRUPA='ALIMENTE' AND M.GRUPA='ZAHAR'
GROUP BY M.DENP;
```

3.

```
SELECT COUNT(*) FROM  VANZARI;
```

4.

```
SELECT COUNT(*)
FROM VANZARI
WHERE COD_CLIENT='ANA' AND CODP='ZAHA----00';
```

5.

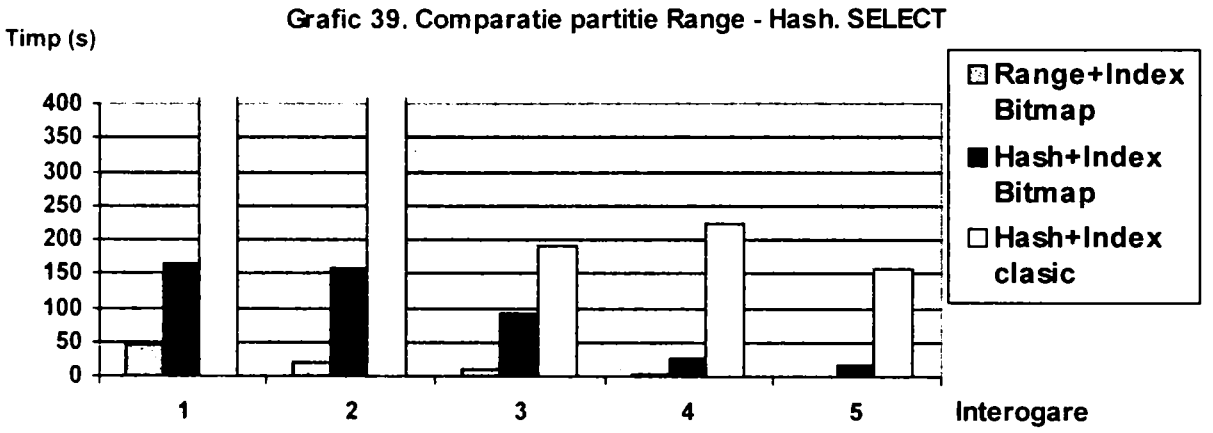
```
SELECT COUNT(*)
FROM VANZARI
WHERE COD_CLIENT='ANA' AND CODP='ZAHA----00'
        AND CODT=TO_DATE('20-MAY-2003', 'DD-MON-YYYY');
```

Atât la nivelul tabelei circumstanțiale VANZARI, cât şi la nivelul tabelelor dimensionale MARFURI, CLIENTI şi TIMPI s-au construit indecşi pentru a mări performanța sistemului. Indecşii au fost de tip bitmap static, bitmap dinamic sau clasic. Rezultatele sunt prezentate în tabelul 22 şi în graficul 39.

Tabel 22. Timpii de răspuns la interogări pentru tabele partiționate.

Interogare	Range+Index bitmap (s)	Hash+Index bitmap (s)	Hash+Index clasic (s)
1.	47	165	494
2.	21	156	532

3	9	93	190
4	2	26	223
5	1	18	159



Pormind de la rezultatele testelor efectuate se pot formula următoarele concluzii:

- partiționarea tabelelor asigură o flexililitate sporită la nivelul gestionării structurii fizice a acestora;
- disponibilitatea datelor stocate în tabelele partiționate este superioară;
- dacă dimensiunea tabelei este mai mare de 1 Go se îmbunătățesc sesizabil timpii de răspuns la interogări dacă se foloseşte partiționarea;
- partiționarea de tip range şi list este mai performantă decât cea de tip hash. De câte ori este posibil este indicat să fie folosite aceste tipuri de partiționare;
- folosirea de indecşi asociați tabelelor partiționate îmbunătățeşte performanțele acestora;
    - indecşii de tip bitmap asigură creşteri mai mari ale performanței;

Ca o concluzie generală se poate spune că tabelele partiționate sunt o obțiune deosebit de atractivă pentru situația unor volume mari de date. Ele sunt cu precădere potrivite pentru aplicațiile de tip magazie de date sau pentru sistemele de suport al deciziilor. Ele permit descompunerea unor structuri mari de date în partiții mult mai mici şi mai uşor de gestionat. Flexibilitatea la nivelul structurii datelor creşte şi, de asemenea, disponibilitatea acestora. Partiționarea asigură şi o creştere a performanței bazei de date prin asigurarea unor timpi de răspuns mai mici la operațiile DML. Acestea pot fi aplicate doar asupra acelor partiții care conțin datele necesare şi/sau pot fi executate în paralel. Timpii de execuție ai operațiilor de

join multiplu între tabele sunt şi ei îmbunătățiți, dacă tabelele sunt partiționate după aceeaşi cheie de partiționare.

3.4. Impactul orientării-obiect asupra structurii fizice a bazei de date

Paradigma orientării-obiect a introdus la nivelul limbajelor de programare concepte deosebit de puternice şi de atractive pentru dezvoltatorii de aplicații. Apariția tipurilor de date abstracte a deschis noi orizonturi în ceea ce priveşte capacitatea de a modela date complexe ale lumii reale. Identitatea obiectuală, încapsularea, moştenirea, polimorfismul sunt concepte ce au îmbogățit tipurile de date abstracte şi au permis dezvolarea de noi tipuri de aplicații. Toate acestea, precum şi capacitatea de a reutiliza codul scris, au dus la larga răspândire a limbajelor de programare orientate-obiect.

Nici producătorii de software pentru gestiunea bazelor de date nu au stat departe de aceste concepte ale orientării-obiect. Dacă la început s-a crezut că orientarea-obiect va revoluționa complet şi domeniul bazelor de date, s-a constatat ulterior că nu era aşa de uşor să se asigure gestionarea unor obiecte persistente într-o bază de date şi să se asigure identitatea obiectuală a acestora, precum şi toate celelalte concepte ale orientării-obiect pe lângă cele specifice bazelor de date. O analiză completă a caracteristicilor obiectuale la nivelul bazelor de date poate fi găsită în [Mitea 00]. Un sistem de gestiune a bazelor de date pur orientat-obiect necesita un model cu totul diferit de cel al sistemelor de gestiune a bazelor de date relaționale. Deoarece modelul relațional al datelor este un model puternic, având un solid fundament matematic, care a fost dezvoltat corespunzător de-a lungul anilor, el s-a impus în fața mai noului său rival.

Astăzi, există şi implementări de sisteme de gestiune a bazelor de date pur orientate-obiect, dar ele nu dețin o prea mare cotă de piață. Modelul relațional al datelor este tot cel care domină piața bazelor de date. Pentru a încorpora totuşi conceptele atractive ale orientării-obiect şi la nivelul bazelor de date, producătorii de sisteme de gestiune a bazelor de date au lansat pe piață conceptul de sistem de gestiune a bazelor de date relațional-obiectuale. La nivelul acestor sisteme modelul de date care se foloseşte este tot cel relațional, dar el a fost extins pentru a încorpora concepte orientate-obiect [Mitea 99-1], [Mitea 99-2].

La nivelul acestor sisteme, baza de date este tot o colecție de relații materializate în tabele, doar că este permisă definirea de tipuri de date abstracte pe lângă tipurile de date de bază ale sistemului, tipuri care permit încapsularea, moştenirea, polimorfismul. De fapt, la nivelul sistemelor de gestiune a bazelor de date relațional-obiectuale, orientarea-obiect oferă o

modalitate conceptuală de a modela datele lumii reale prin intermediul unor obiecte care sunt stocate în baza de date sub forma unor tabele.

Sistemele analizate anterior intră în categoria sistemelor de gestiune a bazelor de date care au încercat să integreze unele concepte orientate-obiect rămânând în continuare relaționale. Aceste trăsături orientate-obiect nu afectează, însă, structurile fizice de memorare şi metodele de acces la ele, în sensul că nu au fost create structuri şi metode noi care să fie folosite exclusiv de modelările obiectuale ale datelor. Fie că datele sunt modelate relațional sau relațional-obiectual, ele sunt stocate tot în tabele pentru care sunt posibile aceleaşi tipuri de structuri de date şi aceleaşi metode de acces.

Din acest punct de vedere, observațiile făcute anterior cu privire la influența pe care o pot exercita diferitele tipuri de structuri fizice de memorare şi a metodelor de acces la ele asupra performanței bazei de date în etapa operațională sunt valabile şi în cazul folosirii unor abordări relațional-obiectuale.

Totuşi, în cazul modelării obiectuale a datelor, flexibilitatea în alegerea tipurilor de structuri fizice de memorare a datelor şi a metodelor de acces la acestea nu este la fel de mare la nivelul SGBD-urilor ca în cazul modelării relaționale a datelor.

Spre exemplu, pentru situația unei modelări obiectuale a bazei de date din anexa 2 realizată în Oracle 9*i* sub forma de mai jos:

```
CREATE TYPE Furnizori_TipOb AS OBJECT (
 CodFurn   varchar2(7),
 DenFurn   varchar2(30),
 Adr_Ob        Adresa_TipOb,
 NrFisc     number(8),
 Cont       char(11),
 SoldCrAn  number(14,2),
 RulDbAn  number(14,2),
 RulCrAn  number(14,2),
 RulDbLun  number(14,2),
 RulCrLun  number(14,2)
 );

CREATE TYPE Marfuri_TipOb AS OBJECT (
 CodP   varchar2(10),
 DenP     varchar2(50),
 Pret     number(20,2),
 Grupa     vrchar2(8),
 CotaTva   number(5,2),
 Furn_Ref   REF Furnizori_TipOb SCOPE IS Furnizori_TabOb

 MEMBER FUNCTION PretCuTVA RETURN NUMBER DETERMINISTIC
 );

CREATE OR REPLACE TYPE BODY Marfuri_TipOb IS
 MEMBER FUNCTION PretCuTVA RETURN NUMBER IS
```

```
 BEGIN
  RETURN (Pret+Pret*CotaTVA);
 END;
END;

CREATE TYPE ProdusFactura_TipOb AS OBJECT (
  NrProdus    number(2),
  Marfa_Ref   REF Marfuri_TipOb SCOPE IS Marfuri_TabOb,
  Cantitate   number(15,3),
  Discount    number(2)
  );

CREATE TYPE ListaTelefoane_TipVar AS VARRAY(10) OF number(13);

CREATE TYPE Adresa_TipOb AS OBJECT (
  Strada       varchar2(30),
  Oras         varchar2(20),
  Tara         varchar2(15),
  CodPostal  number(6)
   )

CREATE TYPE Clienti_TipOb AS OBJECT (
  CodClient        varchar2(7),
  DenClient        varchar2(30),
  Adr_Ob          Adresa_TipOb,
  ListaTelefoane_Var   ListaTelefoane_TipVar,
  NrFisc       number(8),
  PlatitTVA  char(1),
  Cont         char(11),
  SoldCrAn  number(14,2),
  RulDbAn  number(14,2),
  RulCrAn  number(14,2),
  RulDbLun  number(14,2),
  RulCrLun  number(14,2)
) NOT FINAL;

CREATE TYPE ListaProduseFactura_TipNestTab AS TABLE OF ProdusFactura_TipOb;

CREATE TYPE Facturi_TipOb AS OBJECT (
  NrFact             number(8),
  Client_Ref         REF Clienti_TipOb SCOPE IS Clienti_TabOb,
  DataEmi            date,
  ListaProduseFactura_NestTab    ListaProduseFactura_TipNestTab,
  Expediat_Ob       Adresa_TipOb
  );

CREATE TYPE FacturiExterne_TipOb UNDER Facturi_TipOb (
  DataCurs date,
   Moneda VARCHAR2(20)
);

CREATE TYPE FacturiInterne_TipOb UNDER Facturi_TipOb (
   CotaTVA number(5,2),
   Agent VARCHAR2(20)
);

CREATE TYPE Incasari_TipOb AS OBJECT (
  Factura          Facturi_TipOb,
  DataIncas        date,
  Valoare          number(20,2),
```

```
 Document      char(3),
 );

CREATE TABLE Clienti_TabOb OF Clienti_TipOb;

CREATE INDEX I_DenClient ON Clienti_TaOb(DenClient);

CREATE TABLE Furnizori_TabOb OF Furnizori_TipOb (CodFurn PRIMARY KEY)
  OBJECT IDENTIFIER IS PRIMARY KEY;

CREATE INDEX I_FurnizoriOras ON Furnizori_TabOb(Adr_Ob.Oras);

CREATE TABLE Marfuri_TabOb OF Marfuri_TipOb (CodP PRIMARY KEY)
  OBJECT IDENTIFIER IS PRIMARY KEY;

CREATE INDEX I_MarfuriFurnRef ON Furnizori_TabOb(Furn_Ref);

CREATE INDEX  I_Marfuri_PretCuTVA ON Marfuri_TabOb x (x.PretCuTVA( ));

CREATE TABLE Facturi_TabOb OF Facturi_TipOb (
  PRIMARY KEY (NrFact),
  FOREIGN KEY (Client_Ref) REFERENCES Clienti_TabOb)
  OBJECT IDENTIFIER IS PRIMARY KEY
  NESTED TABLE ListaProduseFactura_NestTab STORE AS Produse_NestTab (
   (PRIMARY KEY(NESTED_TABLE_ID, NrProdus))
   ORGANIZATION INDEX COMPRESS)
  RETURN AS LOCATOR;

CREATE TABLE Incasari_TabOb OF Incasari_TipOb;
CREATE BITMAP INDEX I_IncasariTypeid ON Incasari_TabOb (SYS_TYPEID(Factura));
```

Situația unui tabele Facturi_TabOb partiționate

```
CREATE TYPE ListaProduseFactura_TipVar AS VARRAY(10000) OF ProdusFactura_TipOb;

CREATE TYPE Facturi_TipOb AS OBJECT (
 NrFact            number(8),
 Client_Ref        REF Clienti_TipOb SCOPE IS Clienti_TabOb,
 DataEmi           date,
 ImagineFactura    blob
 ListaProduseFactura_Var    ListaProduseFactura_TipVar,
 Expediat_Ob       Adresa_TipOb
 );


CREATE TABLE Facturi_TabOb OF Facturi_TipOb
   LOB (ImagineFactura) STORE AS (NOCACHE LOGGING)
   PARTITION BY RANGE (Expediat_Ob.CodPostal)
    (PARTITION FacturiZona1_part
      VALUES LESS THAN (499999)
      LOB (ImagineFactura) STORE AS (
          STORAGE (INITIAL 10 MINEXTENTS 10 MAXEXTENTS 100))
      VARRAY ListaProduseFactura_Var STORE AS LOB (
          STORAGE (INITIAL 10 MINEXTENTS 10 MAXEXTENTS 100)),
   PARTITION FacturiZona2_part
      VALUES LESS THAN (799999)
      LOB (ImagineFactura) STORE AS (
          STORAGE (INITIAL 10 MINEXTENTS 10 MAXEXTENTS 100))
      VARRAY ListaProduseFactura_Var STORE AS LOB (
```

```
        STORAGE (INITIAL 10 MINEXTENTS 10 MAXEXTENTS 100)),
  PARTITION FacturiZona3_part
    VALUES LESS THAN (999999)
    LOB (ImagineFactura) STORE AS (
        STORAGE (INITIAL 10 MINEXTENTS 10 MAXEXTENTS 100))
    VARRAY ListaProduseFactura_Var STORE AS LOB (
        STORAGE (INITIAL 10 MINEXTENTS 10 MAXEXTENTS 100)));
```

Pot fi făcute următoarele observații:

- au fost definite mai multe tipuri de obiecte, aceste tipuri de obiecte sunt doar nişte şabloane cărora serverul Oracle nu le acordă nici un fel de spațiu fizic de stocare;
- un tip de obiecte poate fi definit folosind exclusiv tipuri de date de bază, cum este cazul tipului obiect Adresa_TipOb;
- tipurile de obiecte pot să conțină alte tipuri de obiecte în cadrul lor, precum în cazul tipului obiect Furnizori_TipOb sau a tipului obiect Clienti_TipOb;
- tipurile de obiecte pot să facă referiri către alte tipuri de obiecte, precum în cazul tipurilor Marfuri_TipOb, ProdusFactura_TipOb, Facturi_TipOb;
- pentru a acorda spațiu fizic de memorare pentru aceste obiecte trebuie create obiecte de tip tabelă cu ajutorul comenzii CREATE TABLE. O tabelă obiect este o tabelă în care fiecare rând reprezintă o instanțiere a acelui tip de obiect.
- fiecare rând obiect dintr-o tabelă obiect are asociat un identificator obiectual logic (OID) care îl identifică în mod unic;
- acesta poate fi generat automat de sistem, caz în care este de lungime 16 octeți. Coloana OID a unei tabele obiect este o coloană ascunsă şi sistemul crează automat un index pe această coloană. Pentru exemplul dat tabelele Clienti_TabOb şi Incasari_TabOb folosesc un astfel de identificator obiectual.
- într-un mediu în care există identificatori unici locali, chei primare, care se poate presupune că sunt unici şi la nivel global, pot fi folosiți aceşti identificatori ca şi OID. Folosirea valorii cheii primare ca şi identificator obiectual economiseşte spațiul de memorare al celor 16 octeți pe care sistemul îi generează automat ca OID. În cazul tabelelor Furnizori_TabOb, Marfuri_TabOb, Facturi_TabOb din exemplul dat se folosesc OID-uri bazate pe cheia primară.
- tabelele obiect pot conține obiecte încapsulate, cazul tipului de obiect Adresa_TipOb din cadrul tabelelor Furnizori_TabOb, Clienti_TabOb, Facturi_TabOb;

- tabelele obiect pot indica rândurile altei tabele obiect prin intermediul operatorului REF, precum în cazul tabelelor Marfuri_TabOb, Facturi_TabOb;
- operatorul REF îmbunătățeşte performanțele compunerii în detrimentul unei sintaxe mai complicate a operațiilor DML, deoarece dereferențierea unui pointer către un obiect este în esență o reunire foarte rapidă, aproape fără operații suplimentare de I/E;
- se poate construi index pentru coloana REF care contine clauza SCOPE utilizând comanda CREATE INDEX. Indexul este utilizat pentru a evalua eficient interogările care dereferențiază operatorul REF. Aceste interogări sunt transformate implicit în operații de join. Pentru anumite tipuri de interogări, Oracle utilizează indexul construit asupra lui REF pentru a evalua operația de join în mod corect. Lipsa clauzei SCOPE face imposibilă definirea indexului pe coloana REF.
- modelarea relațiilor 1-M ce implică un număr fix de elemente în partea M se face cu ajutorul tablourilor VARRAY. Acestea sunt indicate atunci când numărul M de elemente este fix şi cunoscut, când se doreşte parcurgerea elementelor tabloului totdeauna în ordine sau când se doreşte regăsirea şi manipularea întregii colecții de elemente ca o singură valoare. Un astfel de tip este ListaTelefoare_TipVar.
- dacă dimensiunea unui VARRAY nu depăseşte 4Ko el este stocat in-line, adică în acelaşi segment de date cu tabela din care face parte. În acest caz, supraîncărcarea asociată cu interogarea şi manipularea acestora este scăzută. Dacă dimensiunea lui este mai mare de 4Ko este stocat out-of-line sub forma unui BLOB. Pentru un astfel de tip de obiect nu se poate crea index.
- tablourile VARRAY nu violează prima formă normală, deoarece serverul Oracle consideră aceste tablouri ca unități atomice;
- tablourile VARRAY pot elimina în parte supraîncărcarea asociată cu operațiile de compunere dintre tabelele implicate în relații 1-M;
- modelarea relațiilor 1-M ce implică un număr variabil de elemente în partea M se face cu ajutorul tabelelor imbricate. Acestea sunt indicate atunci când numărul M de elemente este variabil în timp, când se doreşte interogarea unor elemente aparte ale colecției şi pentru operații masive de inserare, modificare, ştergere. Pentru exemplul dat ListaProduseFactura_TipNestTab este un astfel de tip.
- tabelele imbricate sunt stocate în spații de stocare separate față de cele ale tabelei principale. Tabela imbricată are asociată o coloană ascunsă numită

NESTED_TABLE_ID cu o valoare generată de sistem care permite sistemului să lege rândurile copil ale tabelei imbricate de rândul părinte corespunzător.

- tabelele imbricate pot să folosească fişiere de tip heap pentru stocare. În acest caz este indicat să se creeze un index pe coloana NESTED_TABLE_ID a tabelei imbricate, care va ajuta la executarea mai rapidă a operațiilor de join dintre tabela părinte şi tabela imbricată.
- dacă o tabelă imbricată are o cheie primară, ea poate fi organizată ca o tabelă organizată indexat (IOT). Dacă coloana NESTED_TABLE_ID este un prefix al cheii primare pentru un rând părinte dat, Oracle clusterează fizic rândurile sale copil. Astfel, când un rând părinte este accesat, toate rândurile sale copil pot fi regăsite în mod eficient. Când doar rândurile părinte sunt accesate, eficiența este menținută deoarece rândurile copil nu sunt intermixate cu rândurile părinte.
- pentru seturi mari de rânduri copil asociate unui rând părinte este indicat să se folosească clauza RETURN AS LOCATOR, care permite ca rândurile copil să fie transportate către rândul părinte numai la cerere. Tabela ListaProduseFactura_NestTab foloseşte o astfel de modalitate de transport.
- pentru a uşura gestionarea unor volume mari de date este permisă partiționarea acestora.
- Oracle permite partiționarea tabelelor care conțin obiecte, referințe REF către obiecte, VARRAY-uri on-line sau tabele imbricate. VARRAY-rile out-of-line sunt partiționate similar coloanelor de tip LOB. Tabelele imbricate sunt premise în tabele părinte care pot fi partiționate, ceea ce însă nu conduce şi la partiționarea spațiului de memorare care păstrează tabela imbricată. În exemplul dat este prezentată şi o situație de partiționare a tabelei obiect Facturi_TabOb după codul poştal la care este expediată factura.
- pentru îmbunătățirea performanțelor se pot construi indecşi asupra unei tabele obiect.
- cheia de indexare poate fi alcătuită din atributele scalare ale tipului obiect, precum în cazul indexului I_DenClient.
- cheia de indexare poate fi alcătuită din atributele scalare ale unei coloane obiect din cadrul tipului obiect, precum în cazul indexului I_FurnizoriOras.
- cheia de indexare poate fi alcătuită şi din atributul REF, dar numai dacă acesta este însoțit de clauza SCOPE, precum în cazul indexului I_MarfuriFurnRef.

- cheia de indexare poate fi şi o metodă a unui tip obiect. În acest caz indexul stochează valorile precalculate ale metodei pentru fiecare instanță obiectuală a tipului de obiect care este indexat. Pe viitor ele vor fi referite fără a mai fi evaluate din nou prin intermediul metodei. Acest tip de index este posibil numai dacă funcția returnează întotdeauna aceeaşi valoare pentru fiecare dintre instanțele tipului obiect indexat. În cazul exemplului prezentat indexul I_MarfuriPretCuTVA se încadrează în această categorie.
- utilizând funcția SYS_TYPEID se poate construi un index asupra coloanei ascunse TYPEID pe care o are orice coloană substituibilă. Coloana TYPEID conține valori care identifică tipul obiectual specific al fiecărei instanțe obiectuale din acea coloană substituibilă. Informația din acest index este folosită pentru a evalua mai rapid interogările care utilizează predicatul IS OF pentru a realiza o filtrare după tipul obiectului.
- deoarece, de regulă, numărul de tipuri obiectuale dintr-o ierarhie de obiecte nu este prea mare este indicat ca acest index să fie construit ca index bitmap. Este şi cazul indexului I_IncasariTypeid din exemplul dat.

M *Metodologie de proiectare fizică a bazelor de date*

4.1. Considerații generale

După cum s-a arătat în capitolul 2 procesul de proiectare al bazei de date este structurat pe trei nivele: proiectarea conceptuală, proiectarea logică şi proiectarea fizică a bazei de date [Conno 01]. Deşi proiectarea fizică este ultima etapă din cadrul procesului de proiectare a bazei de date, ea nu trebuie pierdută din vedere şi tratată superficial, mai ales dacă este vorba despre proiectarea unei baze de date de mărime mare, care trebuie să suporte prelucrări multiple şi variate.

Dacă structura fizică a bazei de date nu este foarte bine gândită şi realizată, se vor obține rezultate foarte slabe în etapa operațională a bazei de date atunci când aceasta trebuie consultată, actualizată şi întreținută.

Pentru ca procesul de proiectare să se desfăşoare în bune condiții au fost concepute metodologii de proiectare, care ghidează proiectantul de-a lungul procesului de proiectare prin stabilirea unui cadru de proiectare care trebuie urmat de acesta. Şi la nivelul procesului de proiectare a unei baze de date au fost concepute metodologii de proiectrare, care ajută proiectantul în realizarea proiectului bazei de date [Medin 97], [Conno 01], [Mitea 04-1]. Lucrarea de față îşi propune să introducă o nouă abordare pentru o metodologie de proiectare fizică a bazelor de date, care are ca scop eliminarea erorilor de proiectare prin folosirea unor proceduri de validare matematică a deciziilor de proiectare. Procesul de proiectare fizică a bazei de date poate fi, astfel, cu uşurință asistat de un utilitar de proiectare, care trebuie să

urmeze paşii propuşi în cadrul metodologiei şi să ia deciziile de proiectare după aplicarea validărilor matematice.

4.2. Aplicarea modelului „Cleanroom" şi a specificării formale la proiectarea fizică a unei baze de date

Pornind de la ideea modelului „cleanroom" de dezvoltare a sistemelor software, propun o metodologie de proiectare fizică a bazei de date ce foloseşte aceleaşi principii. La rândul lui, modelul „cleanroom" propus de Mills [Mills 87-2] s-a inspirat din procesul de fabricaţie al semiconductorilor, unde apariţia defectelor este evitată prin manufacturarea semiconductorilor într-o atmosferă extrem de curată.

Prin urmare, metodologia îşi propune să evite apariţia defectelor la nivelul procesului de proiectare fizică a bazei de date. Ea se bazează pe tehnici de verificare statică de-a lungul procesului de proiectare pentru a asigura un produs final lipsit de defecte. Etapele procesului de proiectare fizică a bazei de date sunt specificate formal şi sunt verificate matematic pe măsură ce sunt dezvoltate. Astfel, erorile sunt eliminate din procesul de proiectare înainte ca baza de date să fie implementată. Fagan arată în [Fagan 86] că în general 60% dintre erorile dintr-un program pot fi detectate utilizând inspecţiile sistematice ale programului. Mills, de asemenea, în [Mills 90] sugerează că, mai mult, utilizând metode formale de validare statică a programelor bazate pe verificări matematice s e p ot d etecta m ai m ult de 90% dintre erorile dintr-un program. Prin urmare, produsul rezultat este aproape liber de defecte. Pornind de la aceste observaţii am considerat că este oportun să încerc aplicarea aceleiaşi abordări şi la nivelul proiectării bazei de date. Ideea principală este aceea de a realiza proiectarea bazei de date într-un *„mediu curat"*, adică prin eliminarea într-o măsură cât mai mare a impurităţilor ce constau de fapt în erorile de proiectare. Eliminarea erorilor de proiectare este realizată prin folosirea unui formalism matematic în specificarea paşilor de proiectare şi prin utilizarea unor validări pe baze matematice a deciziilor de proiectare. In final, rezultă un proiect fizic al bazei de date liber de defecte. Etapa de testare a bazei de date, ce urmează după cea de implementare a acesteia, poate fi aproape complet eliminată.

Verificarea statică este eficientă în ceea ce priveşte costul. Defectele pot fi detectate mult mai rapid în cadrul procesului de dezvoltare a sistemului şi la un preţ de cost mai scăzut decât dacă s-ar utiliza tehnici de detecţie a erorilor. In plus, procesul de verificare statică poate fi de asemenea preocupat de asigurarea altor indicatori de calitate [Koopm 03] care să permită conformarea la standarde.

4.3. Prezentarea metodologiei propuse

Metodologia pe care am propus-o este dedicată etapei de proiectare fizică a unei baze de date ce foloseşte modelul relaţional al datelor. Ea ajută la elaborarea etapei de proiectare fizică a bazei de date cu mult mai mare acurateţe, astfel încât în final să se obţină o structură fizică liberă de defecte pentru baza de date.

Metodologia cuprinde mai mulţi paşi care trebuie urmaţi de proiectantul bazei de date şi se aplică după ce a fost selectat sistemul de gestiune al bazei de date care va fi folosit pentru implementarea bazei de date. In cadrul paşilor urmaţi conform metodologiei propuse, trebuie să se ţină cont şi de deciziile luate în cadrul etapelor anterioare de proiectare, şi anume la nivelul etapei de proiectare conceptuală şi logică a bazei de date. Proiectul logic al bazei de date va fi transformat prin intermediul acestei metodologii într-un proiect fizic al bazei de date.

Paşii metodologiei propuse pentru proiectarea fizică a unei baze de date sunt următorii:

Proiectarea fizică a bazei de date relaţionale

Etapa 1. Transformarea modelului de date logic global pentru sistemul SGBD ales

Pasul 1. Relaţiile de bază sunt proiectate în concordanţă cu SGBD-ul ales

Pasul 2. Restricţiile de integriate sunt proiectate în concordanţă cu SGBD-ul ales

Etapa 2. Proiectarea reprezentării fizice

Pasul 3. Analizarea tranzacţiilor

Pasul 4. Identificarea tranzacţiilor critice

Pasul 5. Identificarea relaţiilor critice

Pasul 6. Estimarea dimensiunii relaţiilor

Pasul 7. Identificarea relaţiilor ce pot fi partiţionate

Pasul 8. Identificarea structurilor auxiliare de acces

Pasul 9. Alegerea organizării fişierelor de date

Pasul 10. Alegerea organizării structurilor indexate

Pasul 11. Analiza introducerii unei redundanţe controlate

Pasul 12. Estimarea spațiului de memorare necesar bazei de date

Etapa 3. Proiectarea mecanismelor de securitate

Pasul 13. Proiectarea vizualizărilor

Pasul 14. Proiectarea regulilor de acces

În capitolul 2 s-a arătat că folosirea unor metode formale de specificare duce la reducerea costurilor de proiectare şi la obținerea unor produse calitativ superioare. Ca atare, au fost folosite specificațiile formale la nivelul paşilor metodologiei. În continuare vor fi prezentate pe larg toate cele trei etape ale metodologiei pentru procesul de proiectare fizică a bazei de date. Etapele sunt specificate formal şi acolo unde este posibil au fost introduse metode de validare statică, bazate pe verificări matematice, pentru a se elimina erorile de proiectare.

4.3.1. Transformarea modelului de date logic global pentru sistemul SGBD ales

Prima etapă a metodologiei propuse pentru proiectarea fizică a unei baze de date urmăreşte transformarea modelului logic global al bazei de date în conformitate cu caracteristicile sistemului de gestiune a bazelor de date ales pentru implementarea acesteia. Etapa de proiectare logică a bazei de date a stabilit un model logic global pentru baza de date. Acest model logic trebuie transformat într-un model fizic al bazei de date în conformitate cu trăsăturile sistemului de gestiune a bazelor de date care a fost ales pentru implementarea bazei de date. Aceste transformări implică parcurgea următorilor paşi:

Pasul 1. ***Relațiile de bază sunt proiectate în concordanță cu SGBD-ul ales.***

Relațiile de bază, identificate în procesul de proiectare anterior parcurs, trebuie să fie proiectate în conformitate cu caracteristicile sistemul de gestiune a bazei de date ales.

Etapa este specificată formal astfel:

Funcția: ProiectareRelație

Descriere: Este proiectată relația ținând cont de SGBD țintă

Intrări: Relație_N

Sursa: Schema logică globală

Iesiri: Tabela_N

Destinația: Schema fizică

Pre-condiții: Identificare atribute relație, cheie primară, chei alternative şi chei externe ale relației din schema logică globală

Post-condiții:

(dom Atribut=Domeniu_N) and ((Val_Atribut=NULL) or (Val_Atribut={v1, v2, ..., vn} or (Val_Atribut=Funcție_Calcul)) and (Cheie_Primară=(Atribut1, Atribut2,..)) and (Cheie_Alternativă=(Atribut1, Atribut2,...)) and ((Cheie_Externă=(Atribut1, Atribut2,...) referă Relația_M) and (DeletePK in (NO ACTION, SET NULL, CASCADE)) and (UpdatePK in (NO ACTION, SET NULL, CASCADE)))

Funcția ProiectareRelație se aplică fiecărei relații de bază identificate în etapele anterioare de proiectare. Rezultatele aplicării funcției ProiectareRelație asupra relațiilor de bază sunt colectate în tabelul 1.

Tabel 1. Relații de bază.

Nume Relație	Atribute	Domeniu	Valoarea NULL	Valori Implicite	Atribut Derivat	Cheie Primară	Chei Alternative	Chei Externe	Restricția Cheii Externe	DELETE pe PK	UPDATE pe PK

Unde

- ***Nume Relație*** – este numele relației şi o identifică pe aceasta în mod unic.
- ***Atribute*** – conține numele fiecărui atribut identificat pentru acea relație.
- ***Domeniu*** – este domeniul pe care va fi definit atributul.
- ***Valoare NULL*** – specifică dacă atributul poate avea sau nu valoarea NULL.
- ***Valori Implicite*** – conține valorile implicite pentru acel atribut, dacă ele există.

- ***Atribut Derivat*** – dacă atributul este unul derivat trebuie specificat cum va fi el calculat. Este precizat algoritmul de calcul al atributului derivat.

- ***Cheie Primară*** – conține cheia primară a relației.

- ***Chei Alternative*** – conține cheile alternative pentru acea relație, dacă astfel de chei există.
- ***Chei Externe*** – conține cheile externe (străine) pentru acea relație, dacă există astfel de chei.

- ***Restricția Cheii Externe*** – specifică relația cu care se va stabili integritatea referențială şi cheia primară a acesteia.

- ***DELETE pe PK*** – specifică acțiunea care se va executa asupra cheii externe dacă se încearcă ştergerea cheii primare PK de care aceasta este legată. Variantele posibile de acțiuni sunt: NO ACTION, SET NULL, CASCADE.

- ***UPDATE pe PK*** – specifică acțiunea care se va executa asupra cheii externe dacă se încearcă modificarea cheii primare PK de care aceasta este legată.Variantele posibile de acțiuni sunt: NO ACTION, SET NULL, CASCADE.

Domeniile pe care vor fi definite atributele relațiilor de bază trebuie şi ele să fie determinate. Această funcție de definire a domeniilor atributelor are următoarea specificație formală:

Funcția: DefinireDomeniu

Descriere: Este definit domeniul pe care un atribut poate lua valori

Intrări: Atribut_N

Sursa: Schema logică globală

Iesiri: Domeniu_N

Destinația: Schema fizică

Pre-condiții: Cerințele cu privire la valorile atributului şi domeniile de bază ale SGBD-ului

Post-condiții:

(Nume_Domeniu=Nume) and (Tip_Data=Tip) and (Lungime=Lung) or (Set_Valori={v1, v2, ..., vn}) or (Interval_Valori=[val1..val2])

Funcția DefinireDomeniu trebuie aplicată fiecărui atribut al unei relații de bază. Rezultatul aplicării acestei funcții este colectat în tabelul 2.

Tabel 2. Domenii.

Nume Domeniu	Tip de Date	Lungime	Set Posibil de Valori	Interval Posibil de Valori

Unde

- ***Nume Domeniu*** – este numele dat acelui domeniu de valori, el poate fi numele unui domeniu de bază recunoscut de SGBD-ul țintă sau poate fi un domeniu nou creat

- ***Tip de Date*** – este tipul de date ales pentru acel domeniu.

- ***Lungime*** – este lungimea maximă a datelor din acel domeniu de date

- ***Setul Posibil de Valori*** – este setul valorilor posibile pentru acel domeniu, dacă există un număr limitat de astfel de valori.

- ***Intervalul Posibil de Valori*** – este intervalul de valori acceptat pentru valorile acelui domeniu, dacă există un astfel de interval de valori.

Pasul 2. ***Restricțiile de integritate sunt proiectate în concordanță cu SGBD-ul ales.***

Restricțiile de integritate, aplicabile asupra bazei de date, identificate anterior în procesul de proiectare a bazei de date, sunt proiectate în concordanță cu sistemul de gestiune a bazei de date ales pentru implementare. Pentru fiecare restricție de integritate în parte trebuie hotărât nivelul la care se va face implementarea acesteia. In funcție de caracteristicile SGBD-ului ales, implementarea restricțiilor de integritate se poate face la nivelul limbajului de definire a datelor sau la nivelul programelor de aplicații.

Acest pas este specificat formal astfel:

Funcția: ProiectareRestricție

Descriere: Este proiectată restricția de integritate tinând cont de caracteristicile SGBD-ului țintă

Intrări: Restricția_N

Sursa: Schema logică globală

Iesiri: Restricția_N

Destinația: Schema fizică

Pre-condiții: Identificare Relație_N asupra căreia se aplică restricția

Post-condiții:

(Nume_Restricție=Nume) and (Tip_Restricție in (PK, FK, CK)) and (Nivel_Implementare in (CREATE TABLE, CREATE TRIGGER, APLICATIE)) and (Implemnetare_Restricție=Implementare)

Funcția ProiectareRestricție trebuie aplicată fiecărei constrângeri de integritate a entității sau referențială identificată în etapa de proiectare anterioară a bazei de date, precum şi tuturor constrângerilor de integritate semantică identificate în etapa de specificare a cerințelor. Rezultatele sunt colectate în tabelul 3, care permite o mai uşoară urmărire şi validare a acestora.

Tabel 3. Restricții de integritate.

Nume Restricție de Integritate	Relația	Definiția Restricției de Integritate	Nivelul de Implementare	Implementare

Unde

- ***Nume Restricție de Integritate*** - este numele ales pentru a identifica în mod unic restricția de integritate.

- ***Relația*** – indică relația la nivelul căreia se aplică constrângerea de integritate.

- ***Definiția Restricției de Integritate*** – este definiția restricției de integritate, prin care se precizează şi tipul restricției de integritate: constrângere la nivelul cheii primare, constrângere referențială sau constrângere semantică .

- ***Nivel de Implementare*** – specifică care este nivelul de implementare al restricției ținând cont de caracteristicile SGBD-ului ales pentru implementare. De exemplu, acesta ar putea fi: CREATE TABLE, CREATE TRIGGER, APPLICATION, etc.

- ***Implementare*** – conține implementarea restricției de integritate ce ține seama de tipul restricției şi de facilitățile oferite de SGBD pentru implementarea acelui tip de restricție.

4.3.2. Proiectarea reprezentării fizice

Cea de a doua etapă a proiectării fizice a bazei de date presupune proiectarea reprezentării fizice a acesteia. Pornind de la operațiile care trebuie să se execute asupra bazei de date pentru a putea răspunde cerințelor utilizatorilor acesteia, trebuie făcută o analiză completă şi consistentă a tuturor acestor tranzacții, asftel încât să se poată alege cea mai bună structură pentru fişierele de date şi pentru structurile auxiliare de acces.

A fost propusă o nouă metodă de a realiza proiectarea reprezentării fizice a bazei de date [Mitea 04-2]. Ea permite validarea statică pe baze matematice a deciziilor de proiectare, astfel încât erorile de proiectare să fie eliminate din cadrul procesului de proiectare a bazei de date. Elaborarea paşilor etapei de proiectare a reprezentării fizice a bazei de date a ținut cont şi de rezultatele testelor efectuate în cadrul capitolului 3.

Proiectarea reprezentării fizice a bazei de date se realizează în mai mulți paşi.

Pasul 3. ***Analizarea tranzacțiilor.***

Toate tranzacțiile identificate în etapa de specificare a cerințelor trebuie supuse unei analize *cantitative* şi *calitative*. Aceste analize trebuie să releve care sunt tranzacțiile cel mai des utilizate, ce relații implică fiecare dintre aceste tranzacții şi care sunt operațiile care se execută la nivelul fiecărei tranzacții în parte. Pasul implică efectuarea următoarelor analize:

- estimarea frecvențelor de execuție și a orelor sau intervalelor de timp de încărcare maximă pentru fiecare tranzacție în parte și a restricțiilor cu privire la timpii de execuție maximi ai tranzacției.
- specificarea și analizarea operațiilor care se execută la nivelul fiecărei tranzacții.

De regulă, în literatura de specialitate metoda propusă pentru analiza tranzacțiilor este aceea a *hărților de utilizare a tranzacțiilor*. În [Conno 01] și [Rusin 95] se pot găsi referințe legate de modul de construire a acestor hărți și de realizare a analizei cu ajutorul lor. Teza propune o metodă nouă de realizare a analizei tranzacțiilor, bazată pe calculul unor punctaje asociate tranzacțiilor și relațiilor din baza de date. Aceste punctaje se calculează ponderat în funcție de frecvențele de execuție ale tranzacțiilor, de orele de încărare de vârf și de restricțiile cu privire la timpii de execuție ai tranzacțiilor. Funcția de estimare a frecvenței de execuție a tranzacțiilor, cea de estimare a orelor cu încărcare de vârf și a timpului de execuție maxim admis pentru tranzacție poate fi specificată formal astfel:

Funcția: FrecvențăTranzacție

Descriere: Este definită frecvența de execuție a tranzacției, orele cu încărcare de vârf și restricția timpului de execuție a acesteia

Intrări: Tranzacția_N

Sursa: Lista cerințelor funcționale

Iesiri: FrecventăTranzacție_N

Destinația: Analiza tranzacțiilor

Pre-condiții: Identificare cerințe Tranzacție_N

Post-condiții:

(Nume_Tranzacție=Nume) and (ID_Tranzacție=ID) and (Frecvență_Tranzacție=Frecv) and (OreVârf_Tranzacție=Ore) and (TimpExecuție_Tranzacție=Timp)

Funcția FrecvențăTranzacție trebuie să fie aplicată fiecărei tranzacții identificate în etapa de specificare a cerințelor sistemului. Frecvențele de execuție ale tranzacțiilor, chiar dacă sunt raportate la unități de timp diferite specifice fiecărei tranzacții în parte, trebuie aduse la un numitor comun, adică trebuie raportate la aceeași unitate comună de timp. Este de

preferat ca raportarea să se facă față de cea mai mică unitate de timp folosită. Informațiile rezultate în urma acestei operații pot fi colectate într-un tabel de forma celui următor:

Tabel 4. Tranzacții.

Identificator Tranzacție	Nume Tranzacție	Frecvența Tranzacției	Orele cu Incărcare de Vârf	Restricție Timp de Execuție

Unde

- ***Identificator Tranzacție*** – este un identificator ales pentru a identifica în mod unic fiecare tranzacție.

- ***Nume Tranzacție*** – este numele dat acelei tranzacții.

- ***Frecvența Tranzacției*** – este frecvența estimată a acelei tranzacții. Frecvența va fi exprimată în număr de execuții ale tranzacției într-un interval specificat şi fix de timp. De exemplu, 2 ori/min, 20 ori/oră, 100 ori/zi, 2 ori/săpt., 10 ori/lună, etc.

- ***Orele cu Incărcare de Vârf*** – sunt orele sau intervalele orare când acea tranzacție este cel mai des procesată.

- ***Restricție Timp de Execuție*** – este constrângerea impusă timpului de execuție al tranzacției. De exemplu, 2 s, 2 min, etc.

Pasul 4. ***Identificarea tranzacțiilor critice.***

Este stipulat în literatura de specialitate că aproape orice aplicație se conformează regulei 80/20 [Wiede 83]. Aceasta spune că 80% din prelucrările efectuate asupra unei baze de date se regăsesc la nivelul a 20% dintre tranzacțiile care operează la nivelul acesteia. Tranzacțiile care se încadrează în categoria celor 20% trebuie identificate deoarece ele sunt punctele critice ale sistemului. In continuare aceste tranzacții trebuie corect şi atent analizate pentru a determina structura fizică optimă a bazei de date.

Informațiile din tabelul 4 ajută la identificarea tranzacțiilor care sunt executate doar de foarte puține ori la intervale relativ mari de timp şi care, astfel, nu vor influența în mod

evident performanța în exploatare a bazei de date. Aceste tranzacții pot fi eliminate din viitoarea analiză, având în vedere că ele nu sunt puncte critice la nivelul sistemului.

Pentru a putea decide pe baze corecte, care sunt tranzacțiile critice care vor fi analizate mai amănunțit, propun umătoarea metodă ce se bazează pe verificări şi validări matematice ale procesului de selecție a tranzacțiilor critice.

Funcția de selecție pe baze cantitative a tranzacțiilor care se încadrează în categoria celor 20% este definită formal astfel:

Funcția: SelecțieCantitativă1Tranzacție

Descriere: Sunt selectate pe baze cantitative tranzacțiile critice din sistem

Intrări: Punctajul calculat P_{T_N} pentru Tranzacție_N, punctajul de prag al tranzacției P_{PRAG}

Sursa: CalculPunctajTranzacție

Iesiri: ID_Tranzacție_N

Destinația: Lista tranzacțiilor critice L_T

Pre-condiții: Este calculat punctajul P_{T_N} pentru Tranzacție_N

Post-condiții:

if ($P_{T\_N} \geq P_{PRAG}$)

ID_Tranzacție_N in L_T

endif

Selecția tranzacțiilor critice se face în funcție de punctajul calculat al acestora. Este ales un punctaj de prag, P_{PRAG}, în funcție de care tranzacțiile sunt considerate sau nu critice. Punctajul de prag se alege astfel încât să se respecte regula 80/20. Punctajul calculat pentru o tranzacție, P_{T_N}, este funcție de caracteristicile cantitative ale tranzacției. Funcția de calcul a punctajului unei tranzacții este definită formal astfel:

Funcția: CalculPunctajTranzacție

Descriere: Este calculat punctajul P_{T_N} aferent tranzacției

Intrări: Punctajul calculat Pf_{T_N} al frecvenței de execuție pentru Tranzacție_N, ponderea pf_{T_N} a acestuia în punctajul total, punctajul calculat Pc_{T_N} al gradului de execuție concurentă pentru Tranzacție_N, ponderea pc_{T_N} a acestuia în punctajul total, punctajul

calculat Pt_{T_N} al restricției de timp de execuție pentru Tranzacție_N, ponderea pt_{T_N} a acestuia în punctajul total

Sursa: CalculPunctajFrecvențăTranzacție, CalculPunctajConcurențăTranzacție, CalculPunctajTimpTranzacție

Iesiri: Punctaj P_{T_N} pentruTranzacție_N

Destinația: SelecțieCantitativă1Tranzacție

Pre-condiții: Este calculat punctajul frecvenței de execuție Pf_{T_N} pentru Tranzacție_N, punctajul gradului de execuție concurentă Pc_{T_N} pentru Tranzacție_N şi punctajul restricției de timp de execuție Pt_{T_N} pentru Tranzacție_N

Post-condiții:

$P_{T\_N} = pf_{T\_N} * Pf_{T\_N} + pc_{T\_N} * Pc_{T\_N} + pt_{T\_N} * Pt_{T\_N}$

După cum se observă se poate asocia o pondere fiecărei dintre caracteristicile cantitative ale tranzacției. Valorile acestor ponderi sunt determinate de gradul în care caracteristica asociată ponderii influențează comportarea tranzacției. Punctajul tranzacției este determinat în funcție de aceste ponderi şi de punctajele calculate pentru fiecare caracteristică cantitativă în parte, şi anume punctajul frecvenței de execuție Pf_{T_N}, punctajul gradului de execuție concurentă Pc_{T_N} şi punctajul restricției de timp de execuție Pt_{T_N} .

Funcția: CalculPunctajFrecvențăTranzacție

Descriere: Este calculat punctajul Pf_{T_N} al frecvenței de execuție aferent tranzacției

Intrări: Frecvența de execuție F_{T_N} pentru Tranzacție_N, intervalul de acordare a punctelor ΔP, valoarea maximă a frecvențelor de execuție F_{T_MAX}, valoarea minimă a frecvențelor de execuție F_{T_MIN}

Sursa: FrecvențăTranzacție

Iesiri: Punctaj frecvență de execuție Pf_{T_N} pentruTranzacție_N

Destinația: CalculPunctajTranzacție

Pre-condiții: Este aplicată funcția FrecvențăTranzacție tuturor tranzacțiilor

Post-condiții:

$$Pf_{T\_N} = F_{T\_N} * \frac{\Delta P}{F_{T\_MAX} - F_{T\_MIN}} \quad , F_{T\_MIN} \neq 0$$

Calculul punctajului frecvenței de execuție a tranzacției se face pe baza frecvenței de execuție a tranzacției, ținând cont de valorile maxime şi minime ale frecvențelor de execuție ale tuturor tranzacțiilor executate în sistem. Punctajul frecvenței de execuție a tranzacției se acordă într-un interval de punctaj ΔP care este ales de proiectantul bazei de date.

Funcția: CalculPunctajConcurențăTranzacție

Descriere: Este calculat punctajul Pc_{T_N} al gradului de execuție concurentă a tranzacției

Intrări: Vectorul punctajelor pentru gradul de execuție concurentă a tranzacțiilor V2

Sursa: GenerareVector_V2

Iesiri: Punctaj grad de execuție concurentă Pc_{T_N} pentruTranzacție_N

Destinația: CalculPunctajTranzacție

Pre-condiții: Este generat vectorul V2

Post-condiții:

Pc_{T_N} = MAX(V2(j)) ; j corespunde intervalului orar cu încărcare de vârf în care se încadrează Tranzacție_N

Calculul punctajului aferent gradului de execuție concurentă a tranzacției este calculat prin intermediul vectorilor V1 şi V2, care sunt generați în conformitate cu următoarele specificații formale.

Funcția: GenerareVector_V2

Descriere: Este generat vectorul V2 al punctajelor pentru gradul de execuție concurentă a tranzacțiilor

Intrări: Vectorul numărului de tranzacții care se execută concurent V1, intervalul de acordare a punctelor ΔP, valoarea maximă $V1_{MAX}$ din vectorul V1, valoarea minimă $V1_{MIN}$ din vectorul V1

Sursa: GenerareVector_V1

Iesiri: Vector V2

Destinația: CalculPunctajConcurențăTranzacție

Pre-condiții: Este generat vectorul V1

Post-condiții:

$$V2(i) = V1(i) * \frac{\Delta P}{V1_{MAX} - V1_{MIN}} \quad ; V1_{MIN} \neq 0$$

i ia valori de la 1 la n

Vectorul V2 este obținut pornind de la vectorul V1 prin transformarea valorilor din vectorul V1 în punctaje. Punctajele sunt acordate tot în limitele unui interval de punctaj ΔP specificat de proiectantul bazei de date.

Funcția: GenerareVector_V1

Descriere: Este generat vectorul V1 al numărului de tranzacții care se execută concurent

Intrări: Orele cu încărcare de vârf pentru Tranzacție_N

Sursa: FrecvențăTranzacție

Iesiri: Vector V1

Destinația: GenerareVector_V2

Pre-condiții: Este aplicată funcția FrecvențăTranzacție tuturor tranzacțiilor

Post-condiții:

$$V1 = (NR_1, NR_2, ..., NR_i, ..., NR_n) \quad ; \text{unde } n = \frac{24\ h}{\Delta t}$$

Δt este intervalul de timp etalon ales pentru analiza concurenței

i corespunde intervalului orar Δt_i

$$NR_i = \sum_{j=1}^{M} (F_{T\_j} * Ph)$$

M este numărul total de tranzacții identificate în etapa de specificare a cerințelor

Ph = 0 dacă Tranzacție_N nu se execută în intervalul orar Δt_i

1 dacă Tranzacție_N se execută în intervalul orar Δt_i

Pentru generarea vectorului V1 este necesară alegerea unui interval de timp etalon Δt, care va fi folosit pentru analiza activității concurente care se desfăşoară la nivelul sistemului. Cele 24 de ore ale unei zile sunt împărțite în segmente de mărimea etalonului de timp şi la nivelul fiecărui segment de timp este analizată activitatea concurentă care se desfăşoară în

sistem. Vectorul V1 conține valori care specifică încărcarea maximă care se poate produce la nivelul sistemului pentru fiecare interval orar ales pentru analiză.

Funcția: CalculPunctajTimpTranzacție

Descriere: Este calculat punctajul Pt_{T_N} al restricției de timp de execuție aferent tranzacției

Intrări: Restricția de timp de execuție T_{T_N} pentru Tranzacție_N, intervalul de acordare a punctelor ΔP, valoarea maximă a restricției de timp de execuție T_{T_MAX}, valoarea minimă a restricției de timp de execuție T_{T_MIN}

Sursa: FrecvențăTranzacție

Iesiri: Punctaj restricție de timp de execuție Pt_{T_N} pentruTranzacție_N

Destinația: CalculPunctajTranzacție

Pre-condiții: Este aplicată funcția FrecvențăTranzacție tuturor tranzacțiilor

Post-condiții:

$$Pt_{T\_N} = (T_{T\_MAX} - T_{T\_MIN} - T_{T\_N}) * \frac{\Delta P}{T_{T\_MAX} - T_{T\_MIN}} \quad \text{pentru } 0<T_{T\_N} < T_{T\_MAX}$$

0 pentru $T_{T\_N} = 0$

ΔP pentru $T_{T\_N} = T_{T\_MAX}$

Punctajul restricției de timp de execuție aferent tranzacției se calculează pornind de la constrângerea de timp de răspuns impusă tranzacției, ținând cont de valorile maxime şi minime ale acestor restricții la nivelul tuturor tranzacțiilor din sistem. Acest punctaj se acordă, de asemenea, în limitele unui interval de punctaj ΔP ales de proiectantul bazei de date.

Tranzacțiile rămase în urma aplicării funcției SelecțieCantitativăTranzacție alcătuiesc lista L_T a tranzacțiilor critice. Pentru aceste tranzacții analiza este continuată, specificându-se şi analizându-se operațiile care se execută la nivelul fiecărei tranzacții în parte. Tabelul 5 colectează datele pentru această analiză. Fiecare tranzacție din lista L_T este analizată şi sunt marcate cu x operațiile (selecție, inserare, modificare, ştergere) pe care tranzacția trebuie să le efectueze asupra relațiilor din baza de date pentru a satisface cerințele utilizatorilor. Funcția OperațiiTranzacție prin care se specifică operațiile executate de tranzacție asupra relațiilor bazei de date are următoarea specificație formală:

Funcția: OperațiiTranzacție

Descriere: Sunt definite operațiile executate de tranzacție asupra relațiilor din baza de date

Intrări: Cerințele funcționale ale tranzacției

Sursa: Etapa de specificare a cerințelor

Iesiri: Operații_Tranzacție_N

Destinația: NumărOperațiiTranzacție

Pre-condiții: Sunt specificate cerințele funcționale pentru Tranzacție_N

Post-condiții:

```
for i=1 to M                          ; unde M este numărul total de relații de bază
    if Tranzacție_N prelucrează Relație_i
        { if (Tranzacție_N execută Operație_Căutare pe Relație_i)
            Căutare=x
          endif
          if (Tranzacție_N execută Operație_Inserare pe Relație_i)
            Inserare=x
          endif
          if (Tranzacție_N execută Operație_Modificare pe Relație_i)
            Modificare=x
          endif
          if (Tranzacție_N execută Operație_Ștergere pe Relație_i)
            Ștergere=x
          endif
          if (Tranzacție_N execută Operație_Join pe Relație_i)
            Join=x
            Contor_Join_Relație_i_Relație_j= Contor_Join_Relație_i_Relație_j +1
          endif
        }
```

După stabilirea operațiilor executate de tranzacție asupra relațiilor bazei de date se calculează numărul total de operații executate asupra relațiilor din baza de date de fiecare tranzacție în parte, notat cu Nop. Funcția de calcul a lui Nop este specificată formal astfel:

Funcția: NumărOperațiiTranzacție

Descriere: Este calculat numărul total de operații executate de tranzacție asupra relațiilor din baza de date

Intrări: Operațiile executate de tranzacție asupra relațiilor bazei de date

Sursa: OperațiiTranzacție

Iesiri: Numărul total de operații Nop pentru Tranzacție_N

Destinația: CalculPunctajActivitateConcurentăTranzacție

Pre-condiții: Este aplicată funcția OperațiiTranzacție pentru Tranzacție_N

Post-condiții:

```
Nop=0
for i=1 to M                          ; unde M este numărul total de relații de bază
        { if (Căutare_Relație_i=x)
                Nop=Nop+1
          endif
          if (Inserare_Relație_i=x)
                Nop=Nop+1
          endif
          if (Modificare_Relație_i=x)
                Nop=Nop+1
          endif
          if (Ştergere_Relație_i=x)
                Nop=Nop+1
          endif
        }
```

Se calculează un punctaj al activității concurente desfăşurate de fiecare tranzacție în parte care este dependent de frecvența estimată de execuție a tranzacției şi de numărul de operații executate de acea tranzacție asupra relațiilor din baza de date. Acest punctaj de activitate concurentă, notat cu Pac_{T_N}, este calculat cu ajutorul funcției CalculPunctajActivitateConcurentăTranzacție definită formal astfel:

Funcția: CalculPunctajActivitateConcurentăTranzacție

Descriere: Este calculat punctajul Pac_{T_N} al activității concurente executate de tranzacției

Intrări: Frecvența de execuție F_{T_N} pentru Tranzacție_N, intervalul de acordare a punctelor ΔP, numărul total de operații Nop_{T_N} executate de Tranzacție_N, valoarea maximă a produsului dintre frecvențele de execuție ale tranzacțiilor şi numărul total de operații executate de ele $F_{T\_N}* Nop_{T\_N} \mid_{MAX}$ şi valoarea minimă a produsului dintre frecvențele de execuție ale tranzacțiilor şi numărul total de operații executate de ele $F_{T\_N}* Nop_{T\_N} \mid_{MIN}$

Sursa: FrecvențăTranzacție, NumărOperațiiTranzacție

Iesiri: Punctaj activitate concurentă Pac_{T_N} pentruTranzacție_N

Destinația: SelecțieTranzacție

Pre-condiții: Sunt aplicate funcțiile FrecvențăTranzacție şi NumărOperațiiTranzacție tuturor tranzacțiilor

Post-condiții:

$$Pac_{T\_N} = F_{T\_N} * Nop_{T\_N} * \frac{\Delta P}{F_{T\_N}* Nop_{T\_N} \mid_{MAX} - F_{T\_N}* Nop_{T\_N} \mid_{MIN}} \; ; \quad F_{T\_N}* Nop_{T\_N} \mid_{MIN} \neq 0$$

Punctajul activității concurente desfăşurată la nivelul bazei de date de fiecare tranzacție în parte este calculat tot prin raportarea sa la intervalul de punctaj ΔP stabilit de proiectantul bazei de date. Selecția finală pe criterii cantitative a tranzacțiilor critice se face în funcție de punctajul calculat al acestora. Este ales un punctaj de prag, Pac_{PRAG}, în funcție de care tranzacțiile sunt considerate sau nu critice. Valoarea lui Pac_{PRAG} este aleasă astfel încât să se respecte regula 80/20.

Funcția: SelecțieCantitativă2Tranzacție

Descriere: Sunt selectate pe baze cantitative tranzacțiile critice din sistem

Intrări: Punctajul calculat Pac_{T_N} pentru Tranzacție_N, punctajul de prag al tranzacțiilor Pac_{PRAG}

Sursa: CalculPunctajActivitateConcurentăTranzacție

Iesiri: ID_Tranzacție_N

Destinația: Lista tranzacțiilor critice L'_T

Pre-condiții: Este calculat punctajul Pac_{T_N} pentru Tranzacție_N

Post-condiții:

if $(Pac_{T\_N} \geq Pac_{PRAG})$

ID_Tranzacție_N in L'_T

endif

Tranzacțiile din lista L'_T sunt acele tranzacții care implică cel mai mare volum de prelucrări asupra datelor din baza de date şi care sunt procesate cel mai frecvent. Ele sunt tranzacțiile ce alcătuiesc categoria „celor 20% dintre tranzacții care efectuează 80% dintre prelucrările bazei de date". Ele sunt tranzacțiile critice la nivelul sistemului, care determină cum va trebui să arate structura optimă a fişierelor de date şi ce metode auxiliare de acces se vor utiliza. Pentru aceste tranzacții analiza trebuie continuată pe considerente calitative, de astă dată, pentru a decide cum va fi structurată baza de date. Analiza calitativă a tranzacțiilor presupune luarea în considerare a tipurilor de operații care se execută la nivelul acestor tranzacții pentru a putea hotărî pe baze consistente şi corecte care va fi structura ideală pentru fiecare relație în parte. In tabelul 5 sunt colectate informațiile calitative despre tranzacții, care folosesc după aceea la luarea deciziilor de proiectare fizică a bazei de date.

Tabel 5. Analiza operațiilor executate de tranzacții

Relație / Tranzacție	Relație 1				Relație 2				Relație 3				...Relație n				Nop	Punctaj Pac Tranzacție
	C	I	M	S	C	I	M	S	C	I	M	S	C	I	M	S		
Tranz_ID1																		
Tranz_ID2																		
Tranz_ID3																		
...																		
Tranz_IDm																		
Pentru tranzacțiile din lista L'_T																		
Op./Rel.																		
Total C/(I+M+S)																		
Total I/(M+S)																		
Total (C+I+M+S)																		
Punctaj P_R Relație																		
Punctaj Concurență Pc_R																		
Pentru relațiile din lista L_R																		
$I_{activity}$																		
$I_{type\ activity}$																		

Unde

- ***Relație***$_i$ – sunt relațiile de bază identificate în etapele anterioare ale procesului de proiectare.

- ***C*** – operație de selecție (căutare)

- ***I*** – operație de inserare

- ***M*** – operație de modificare

- ***S*** – operație de ştergere

- ***Nop*** – este numărul total de operații, indiferent de tip, executat de acea tranzacție.

- ***Punctaj Pac Tranzacție*** – este punctajul calculat al activității concurente a tranzacției.

- ***Tranz_ID$_i$*** – este identificatorul unic al tranzacției. Identificatorii tranzacțiilor sunt cei din lista L_T. Liniile din tabel având un background mai închis la culoare corespund tranzacțiilor prezente în lista $L`_T$.

- ***Op./Rel.*** – conține numărul total de operații, pentru fiecare tip de operație în parte, executat asupra fiecărei relații de bază. Sunt totalizate operațiile pentru fiecare tranzacție din lista L'_T.

- ***Total C/(I+M+S)*** – conține numărul total de operații de selecție şi numărul total de operații de inserare+modificare+ştergere executate asupra fiecărei relații în parte. Sunt totalizate operațiile pentru fiecare tranzacție din lista L'_T.

- ***Total I/(M+S)*** – conține numărul total de operații de inserare şi numărul total de operații de modificare+ştergere executate asupra fiecărei relații în parte. Sunt totalizate operațiile pentru fiecare tranzacție din lista L'_T.

- ***Total (C+I+M+S)*** – este numărul total de operații, indiferent de tip, executate asupra fiecărei relații în parte. Sunt totalizate operațiile pentru fiecare tranzacție din lista L'_T.

- ***Punctaj P_R Relație*** – este punctajul calculat al relației. Este calculat punctajul pentru fiecare relație ținând cont de tranzacțiile din lista L'_T.

- ***Punctaj Concurență*** Pc_R – este punctajul calculat pentru execuții concurente asupra relației. Este calculat punctajul pentru fiecare relație ținând cont de tranzacțiile din lista L'_T.

- $I_{activitate}$ – este valoarea indicelui de activitate pentru fiecare relație în parte. Este calculat indicele de activitate pentru fiecare relație din lista L_R.

- $I_{tip\_activitate}$ – este valoarea indicelui tipului de activitate pentru fiecare relație în parte. Este calculat indicele tipului de activitate pentru fiecare relație din lista L_R.

Analiza calitativă a tranzacțiilor presupune execuția următorilor pași prin care se calculează numărul de operații executate asupra fiecărei relații în parte. Specificația formală a acestora este:

Funcția: NumărOperațiiCăutareRelație

Descriere: Este calculat numărul total de operații de selecție executate asupra relației de bază de către toate tranzacțiile din lista L'_T

Intrări: Operațiile executate de tranzacție asupra relațiilor bazei de date, frecvența de execuție a tranzacțiilor din lista L'_T

Sursa: OperațiiTranzacție, FrecvențăTranzacție

Iesiri: Numărul total de operații de selecție NC_{R_N} efectuate asupra Relație_N

Destinația: CalculPunctajRelație

Pre-condiții: Sunt aplicate funcțiile OperațiiTranzacție și FrecvențăTranzacție pentru fiecare Tranzacție_N

Post-condiții:

$NC_{R\_N}=0$

for i=1 to M ; unde M este numărul total de tranzacții din L'_T

{Tranz_ID = $L'_T(i)$

if (Tranz_ID _Căutare_Relație$_N$=x)

$NC_{R\_N} = NC_{R\_N} + F_{T\_N}$

endif

}

Funcția calculează numărul operațiilor de selecție efectuate asupra relației de către tranzacțiile critice.

Funcția: NumărOperațiiInserareRelație

Descriere: Este calculat numărul total de operații de inserare executate asupra relației de bază de către toate tranzacțiile din lista L'_T

Intrări: Operațiile executate de tranzacție asupra relațiilor bazei de date, frecvența de execuție a tranzacțiilor din lista L'_T

Sursa: OperațiiTranzacție, FrecvențăTranzacție

Iesiri: Numărul total de operații de inserare NI_{R_N} efectuate asupra Relație_N

Destinația: CalculPunctajRelație

Pre-condiții: Sunt aplicate funcțiile OperațiiTranzacție şi FrecvențăTranzacție pentru fiecare Tranzacție_N

Post-condiții:

$NI_{R\_N}=0$

for i=1 to M ; unde M este numărul total de tranzacții din L'_T

{Tranz_ID = L'_T(i)

if (Tranz_ID _Inserare_Relație$_N$=x)

$NI_{R\_N} = NI_{R\_N} + F_{T\_N}$

endif

}

Funcția calculează numărul operațiilor de adăugare efectuate asupra relației de către tranzacțiile critice.

Funcția: NumărOperațiiModificareRelație

Descriere: Este calculat numărul total de operații de modificare executate asupra relației de bază de către toate tranzacțiile din lista L'_T

Intrări: Operațiile executate de tranzacție asupra relațiilor bazei de date, frecvența de execuție a tranzacțiilor din lista L'_T

Sursa: OperațiiTranzacție, FrecvențăTranzacție

Iesiri: Numărul total de operații de modificare NM_{R_N} efectuate asupra Relație_N

Destinația: CalculPunctajRelație

Pre-condiții: Sunt aplicate funcțiile OperațiiTranzacție şi FrecvențăTranzacție pentru fiecare Tranzacție_N

Post-condiții:

NM_{R_N}=0

for i=1 to M ; unde M este numărul total de tranzacții din L'_T

{Tranz_ID = L'_T(i)

if (Tranz_ID _Modificare_Relație$_N$=x)

$NM_{R\_N} = NM_{R\_N} + F_{T\_N}$

endif

}

Funcția calculează numărul operațiilor de modificare efectuate asupra relației de către tranzacțiile critice.

Funcția: NumărOperațiiŞtergereRelație

Descriere: Este calculat numărul total de operații de ştergere executate asupra relației de bază de către toate tranzacțiile din lista L'_T

Intrări: Operațiile executate de tranzacție asupra relațiilor bazei de date, frecvența de execuție a tranzacțiilor din lista L'_T

Sursa: OperațiiTranzacție, FrecvențăTranzacție

Iesiri: Numărul total de operații de ştregere NS_{R_N} efectuate asupra Relație_N

Destinația: CalculPunctajRelație

Pre-condiții: Sunt aplicate funcțiile OperațiiTranzacție şi FrecvențăTranzacție pentru fiecare Tranzacție_N

Post-condiții:

NS_{R_N}=0

for i=1 to M ; unde M este numărul total de tranzacții din L'_T

{Tranz_ID = L'_T(i)

if (Tranz_ID _Ştergere_Relație$_N$=x)

$NS_{R\_N} = NS_{R\_N} + F_{T\_N}$

endif

}

Funcția calculează numărul operațiilor de ştergere efectuate asupra relației de către tranzacțiile critice.

Pasul 5. ***Identificarea relațiilor critice.***

Relațiile asupra cărora tranzacțiile executate în sistem efectuează frecvent un număr mai mare de prelucrări sunt relațiile la nivelul cărora analizele trebuie să fie mai atent şi mai riguros efectuate, deoarece ele reprezintă puncte critice la nivelul bazei de date. Performanța în exploatare a bazei de date depinde de structura aleasă pentru aceste relații. Alegerea structurii optime se va face în urma unei analize cantitative şi calitative a activității desfăşurate la nivelul relațiilor. Pentru fiecare relație în parte se va calcula punctajul relației. Funcția de calcul este următoarea:

Funcția: CalculPunctajRelație

Descriere: Este calculat punctajul P_{R_N} al relației

Intrări: Numărul total de operații executate asupra relației, intervalul de acordare a punctelor ΔP, valoarea maximă a numărului total de operații executate asupra unei relații şi valoarea minimă a numărului total de operații executate asupra unei relații

Sursa: NumărOperațiiCăutareRelație, NumărOperațiiInserareRelație, NumărOperațiiModificareRelație, NumărOperațiiŞtergereRelație

Iesiri: Punctaj relație P_{R_N} pentru Relație_N

Destinația: SelecțieCalitativăRelație

Pre-condiții: Sunt aplicate funcțiile NumărOperațiiCăutareRelație, NumărOperațiiInserareRelație, NumărOperațiiModificareRelație, NumărOperațiiŞtergereRelație

Post-condiții:

$$P_{R\_N} = (NC_{R\_N}+NI_{R\_N}+NM_{R\_N}+NS_{R\_N}) * \frac{\Delta P}{(NC_{R\_N}+NI_{R\_N}+NM_{R\_N}+NS_{R\_N})\,|_{MAX} - (NC_{R\_N}+NI_{R\_N}+NM_{R\_N}+NS_{R\_N})\,|_{MIN}}$$

$$\text{unde } (NC_{R\_N}+NI_{R\_N}+NM_{R\_N}+NS_{R\_N})\,|_{MIN} \neq 0$$

Punctajul relației este calculat ținând cont de numărul de operații de selecție, adăugare, modificare şi ştergere care se execută asupra relației şi de valorile maxime şi minime ale numărului de operații executate la nivelul tuturor relațiilor din sistem. Şi acest punctaj se acordă în limitele unui interval de punctaj ΔP specificat de proiectantul bazei de date.

Punctajul calculat al relației ajută la determinarea relațiilor la nivelul cărora prelucrările sunt mai frecvente. Prin urmare, aceste relații trebuie mai atent analizate atunci când se va stabili structura lor de organizare fizică. Selecția de criterii cantitative a relațiilor critice se face în funcție de punctajul calculat al acestora. Este ales un punctaj de prag, P_{R_PRAG}, în funcție de care relațiile sunt considerate sau nu critice.

Funcția: SelecțieCantitativăRelație

Descriere: Sunt selectate pe baze cantitative relațiile critice din sistem

Intrări: Punctajul calculat P_{R_N} pentru Relație_N, punctajul de prag al relației P_{R_PRAG}

Sursa: CalculPunctajRelație

Iesiri: Relație_N

Destinația: Lista relațiilor critice L_R

Pre-condiții: Este calculat punctajul P_{R_N} pentru Relație_N

Post-condiții:

if ($P_{R\_N} \geq P_{R\_PRAG}$)

Relație_N in L_R

endif

Relațiile prezente în lista L_R sunt relații critice ale bazei de date, asupra cărora trebuie continuată analiza calitativă.

În [Mitea 04-2] poate fi consultată o formă mai concisă a acestei noi metode de realizare a analizei tranzacțiilor.

Pasul 6. ***Estimarea dimensiunii relațiilor.***

Analizele calitative efectuate la nivelul relațiilor trebuie să releve tipurile optime de structuri de date şi structuri auxiliare de acces pentru acestea. Studiile făcută în capitolul 3 au arătat că dimensiunea relației influențează atât tipul de structură de date ce poate fi folosită pentru implementarea fizică a relației, cât şi necesitatea şi oportunitatea unor structuri

auxiliare de acces. Prin urmare, o primă estimare a spațiului ocupat de fiecare relație în parte este foarte utilă pentru că ea poate elimina din analiza viitoare acele relații care nu este indicat să aibă structuri auxiliare de acces sau să fie partiționate. Funcția prin care se estimează inițial dimensiunea relațiilor are următoarea specificație formală:

Funcția: EstimarePrimarăDimensiuneRelație

Descriere: Este estimată dimensiunea relației

Intrări: Numărul estimat al tuplurilor Nt_{R_N} relației, mărimea medie Mt_{R_N} a unui tuplu

Sursa: EstimareNumărTupluriRelație, MărimeTupluRelație

Iesiri: Dimensiunea estimată D_{R_N} pentru Relație_N

Destinația: EliminareRelațieMică

Pre-condiții: Este estimat numărul de tupluri Nt_{R_N} al relației şi mărimea medie Mt_{R_N} a unui tuplu

Post-condiții:

$D_{R\_N} = Nt_{R\_N} * Mt_{R\_N}$

Estimarea primară a dimensiunii relațiilor se face ținând cont de numărul estimat de tupluri al relației şi de mărimea medie a unui tuplu.

Această funcție foloseşte următoarele două funcții prin care se estimează numărul de tupluri ale relației şi mărimea medie a unui tuplu.

Funcția: EstimareNumărTupluriRelație

Descriere: Este estimat numărul de tupluri al relației

Intrări: Frecvențele de execuție ale tranzacțiilor care efectuează operații de inserare şi ştergere asupra relației, lista tranzacțiilor LI_T care efectuează operații de inserare şi lista LS_T care efectuează operații de ştergere asupra relației

Sursa: FrecventăTranzacție, ListăTranzacțiiInserareRelație,

ListăTranzacțiiŞtergereRelație

Iesiri: Numărul estimat Nt_{R_N} de tupluri pentru Relație_N

Destinația: EstimarePrimarăDimensiuneRelație

Pre-condiții: Este stabilită frecvența de execuție a tuturor tranzacțiilor și sunt determinate listele tranzacțiilor care efectuează operații de inserare sau de ștregere asupra relației

Post-condiții:

$$Nt_{R\_N} = \sum_{j=1}^{M} (F_{T\_J} * Rh) - \sum_{j=1}^{N} (F_{T\_j} * Rh)$$

unde M este numărul total de tranzacții din lista LI_T

N este numărul total de tranzacții din lista LS_T

Rh este valoarea corespunzătoare raportului între durata de viată a relației și unitatea de timp la care se raportează frecvența tranzacției

---

Numărul de tupluri al unei relații este influențat de operațiile de inserare și de ștergere care se execută asupra acelei relații de-a lungul duratei sale de viață. Pentru o estimare corectă a numărului de tupluri ale relației trebuie identificate tranzacțiile care efectuează operații de inserare și de ștergere asupra relației și trebuie ținut cont și de frecvențele de execuție ale acestor tranzacții.

Funcțiile cu ajutorul cărora se identifică tranzacțiile care efectuează operații de inserare sau ștergere asupra relației sunt ListăTranzacțiiInserareRelație și ListăTranzacțiiȘtergereRelație și sunt specificate formal mai jos:

Funcția: ListăTranzacțiiInserareRelație

Descriere: Este determinată lista LI_T a tranzacțiilor care efectuează operații de inserare asupra relației

Intrări: Operațiile executate de tranzacții asupra Relație_N

Sursa: OperațiiTranzacție

Iesiri: Lista LI_T a tranzacțiilor care efectuează operații de inserare asupra Relație_N

Destinația: EstimareNumărTupluriRelație

Pre-condiții: Este aplicată funcția OperațiiTranzacție pentru fiecare Tranzacție_N

Post-condiții:

$LI_T=\{\}$

for i=1 to M ; unde M este numărul total de tranzacții identificare în etapa de specificare a cerințelor

if (Tranz_ID$_i$ _Inserare_Relație$_N$=x)

LI_T= LI_T + Tranz_ID$_i$

endif

Funcția ajută la identificarea tuturor tranzacțiilor care efectuează operații de inserare asupra relației.

Funcția: ListăTranzacțiiȘtergereRelație

Descriere: Este determinată lista LS_T a tranzacțiilor care efectuează operații de ştergere asupra relației

Intrări: Operațiile executate de tranzacții asupra Relație_N

Sursa: OperațiiTranzacție

Iesiri: Lista LS_T a tranzacțiilor care efectuează operații de ştergere asupra Relație_N

Destinația: EstimareNumărTupluriRelație

Pre-condiții: Este aplicată funcția OperațiiTranzacție pentru fiecare Tranzacție_N

Post-condiții:

LS_T={}

for i=1 to M ; unde M este numărul total de tranzacții identificare în etapa de specificare a cerințelor

if (Tranz_ID$_i$ _Ştergere_Relație$_N$=x)

LS_T= LS_T + Tranz_ID$_i$

endif

Funcția ajută la identificarea tuturor tranzacțiilor care efectuează operații de ştergere asupra relației.

Mărimea medie a unui tuplu al relației se calculează prin intermediul funcției MărimeTupluRelație, care este specificată formal în continuare:

Funcția: MărimeTupluRelație

Descriere: Este calculată mărimea medie a unui tuplu al relației

Intrări: Tipul de dată şi lungimea atributelor pentru Relație_N

Sursa: ProiectareRelație

Ieşiri: Mărimea medie Mt_{R_N} a unui tuplu pentru Relaţie_N

Destinaţia: EstimarePrimarăDimensiuneRelaţie

Pre-condiţii: Sunt stabilite tipurile de dată şi lungimile atributelor relaţiei

Post-condiţii:

$$Mt_{R\_N} = \sum_{j=1}^{M} M_{A\_j}$$

unde M este numărul total al atributelor pentru Relaţie_N

M_{A_j} este mărimea medie a atributului A_j

Mărimea medie a unui tuplu se obţine ca o sumă de mărimi medii ale tuturor atributelor relaţiei.

Având dimensiunile estimate ale fiecărei relaţii se poate analiza în continuare oportunitatea partiţionării acestora sau a construirii de structuri auxiliare de acces.

Pasul 7. ***Identificarea relaţiilor ce pot fi partiţionate.***

Necesitatea de a partiţiona unele relaţii de dimensiuni mari izvorăşte din gradul de utilizare concurentă a acelor relaţii. Pentru ca performanţele în exploatare ale bazei de date să nu fie degradate de blocările instituite la nivelul obiectelor din baza de date şi de timpii de aşteptare pe care aceste blocări îi introduc la nivelul sistemului, este necesar să se analizeze oportunitatea partiţionării unora dintre relaţiile bazei de date.

Studiile efectuate în capitolul 3 au arătat că pentru relaţiile cu o dimensiune mare este adecvat să se încerce o partiţionare. Dacă relaţia are o dimensiune situată în jurul valorii de 1 Go sau mai mare, se obţin timpi de răspuns la interogări considerabil mai mici în cazul folosirii partiţionării. Dacă dimensiunile sunt mai mici, îmbunătăţirea performanţelor se referă doar la creşterea flexibilităţii şi a disponibilităţii. Tipul partiţiei este influenţat de tipul predicatului folosit în condiţiile de selecţie [Bella 00-1], [Bella 00-2], [Datta 90].

Pentru început trebuie identificate relaţiile care au dimensiune mare şi care sunt pretabile unor operaţii de partiţionare. Funcţia RelaţieDimensiuneMare este cea care selectează relaţiile a căror dimensiune este superioară unei dimensiuni de prag Dim_{PRAG}. Studiile efectuate în capitolul 3 au arătat că dimensiunea de prag este indicat să fie superioară valorii de 1 Go.

Funcția: RelațieDimensiuneMare

Descriere: Sunt determinate relațiile de dimensiune mare, a căror dimensiune depășește o valoare minimă admisă Dim_{PRAG}

Intrări: Dimensiunea estimată D_{R_N} pentru Relație_N, lista relațiilor critice L_R, dimensiunea de prag Dim_{PRAG} a relațiilor

Sursa: EstimarePrimarăDimensiuneRelație, SelecțieCantitativăRelație

Iesiri: Relație_N

Destinația: Lista relațiilor de dimensiune mare LM_R

Pre-condiții: Este estimată dimensiunea D_{R_N} pentru Relație_N

Post-condiții:

if (Relație_N in L_R) and ($D_{R\_N} \geq Dim_{PRAG}$)

Relație_N in LM_R

endif

Relațiile din lista LM_R sunt cele pretabile partiționării, şi ca atare trebuie determinată oportunitatea unei astfel de operații, precum şi cea mai bună soluție de partiționare. Aceasta se bazează pe analiza gradului de utilizare concurentă a relației şi a predicatelor folosite la nivelul operațiilor care prelucrează relația.

Analiza gradului de utilizare concurentă a unei relații din baza de date porneşte cu selectarea tranzacțiilor care au punctajul gradului de execuție concurentă Pc_T mai mare decât o valoare de prag Pc_{T_PRAG}. Calculul punctajului de execuție concurentă a unei tranzacții este realizat de funcția CalculPunctajConcurențăTranzacție. Funcția prin intremediul căreia sunt selectate tranzacțiile având Pc_T mai mare decât Pc_{T_PRAG} este SelecțieTranzacțiiConcurente şi este specificată formal astfel:

Funcția: SelecțieTranzacțiiConcurente

Descriere: Sunt selectate tranzacțiile care se execută concurent cu o influență mai mare asupra bazei de date

Intrări: Punctajul calculat de execuție concurentă Pc_{T_N} pentru Tranzacție_N, punctajul de prag al execuției concurente a unei tranzacții Pc_{T_PRAG}

Sursa: CalculPunctajConcurențăTranzacție

Iesiri: ID_Tranzacție_N

Destinația: Lista tranzacțiilor concurente Lc_T

Pre-condiții: Este calculat punctajul de execuție concurentă Pc_{T_N} pentru Tranzacție_N

Post-condiții:

if ($Pc_{T\_N} \geq Pc_{T\_PRAG}$)

Tranzacție_N in Lc_T

endif

Valoarea pentru punctajul de prag Pc_{T_PRAG} este aleasă astfel încât să se respecte regula 80/20. Pornind de la elementele listei Lc_T se vor determina relațiile de bază din lista LM_R asupra cărora aceste tranzacții se execută. Pentru fiecare relație de bază din lista LM_R se va calcula un punctaj al activității concurente Pc_R la nivelul relației. Relațiile care au cele mai mari punctaje sunt cele critice care pot fi partiționate pentru a mări performanța bazei de date. Calculul punctajului Pc_R se face prin intermediul funcției CalculPunctajConcurențăRelație, specificată formal atfel:

Funcția: CalculPunctajConcurențăRelație

Descriere: Este calculat punctajul Pc_{R_N} al activității concurente desfăşurate asupra relației

Intrări: Activitatea concurentă C_{R_N} asupra Relație_N, intervalul de acordare a punctelor ΔP, valoarea maximă a activității concurente asupra unei relații C_{R_MAX}, valoarea minimă a activității concurente asupra unei relații C_{R_MIN}

Sursa: ActivitateConcurentăRelație

Iesiri: Punctajul activității concurente Pc_{R_N} asupra Relație_N

Destinația: SelecțieTranzacțiiConcurente

Pre-condiții: Este aplicată funcția ActivitateConcurentăRelație tuturor relațiilor

Post-condiții:

$$Pc_{R\_N} = C_{R\_N} * \frac{\Delta P}{C_{R\_MAX} - C_{R\_MIN}} \quad ; \quad C_{R\_MIN} \neq 0, \; R\_N \text{ aparține } LM_R$$

Calculul punctajului concurenței la nivelul unei relații se face în funcție de activitatea concurentă înregistrată la nivelul acelei relații. Acest punctaj este calculat raportându-se la un interval de punctaj ΔP stabilit de proiectantul bazei de date. Activitatea concurentă este

determinată de numărul de tranzacții din lista tranzacțiilor concurente Lc_T care execută prelucrări asupra relației.

Funcția: ActivitateConcurentăRelație

Descriere: Este calculată activitatea concurentă C_{R_N} asupra relației

Intrări: Lista tranzacțiilor concurente Lc_T şi operațiile executate asupra relației de către toate tranzacțiile

Sursa: SelecțieTranzacțiiConcurente, OperațiiTranzacție

Iesiri: Activitatea concurentă C_{R_N} asupra Relație_N

Destinația: CalculPunctajConcurențăRelație

Pre-condiții: Este obținută lista tranzacțiilor concurente Lc_T şi sunt specificate operațiile executate de tranzacții asupra Relație_N

Post-condiții:

$C_{R\_N}=0$

for i=1 to M ; unde M este numărul total de tranzacții din Lc_T

{Tranz_ID = Lc_T(i)

if (Tranz_ID_Căutare_Relație$_N$=x)

$C_{R\_N} = C_{R\_N} +1$

endif

if (Tranz_ID_Inserare_Relație$_N$=x)

$C_{R\_N} = C_{R\_N} +1$

endif

if (Tranz_ID_Modificare_Relație$_N$=x)

$C_{R\_N} = C_{R\_N} +1$

endif

if (Tranz_ID_Ştergere_Relație$_N$=x)

$C_{R\_N} = C_{R\_N} +1$

endif

}

Cunoscând punctajul Pc_{R_N} al activității concurente desfăşurate asupra fiecăreia dintre relațiile din lista LM_R, ce conține relațiile pretabile pentru partiționare, sunt alese relațiile care vor fi partiționate. Selecția se face ținând cont de o valoare de prag Pc_{R_PRAG} a punctajului

activității concurente asupra relației. Funcția cu ajutorul căreia sunt selectate aceste relații este definită formal astfel:

Funcția: SelecțieRelațiiPartiționate

Descriere: Sunt selectate relațiile ce este indicat să fie partiționate

Intrări: Punctajul calculat Pc_{R_N} al activității concurente desfăşurate asupra Relație_N, punctajul de prag al activității concurente desfăşurate asupra unei relații Pc_{R_PRAG}

Sursa: CalculPunctajConcurențăRelație

Iesiri: Relație_N

Destinația: Lista relațiilor partiționate Lp_R

Pre-condiții: Este calculat punctajul Pc_{R_N} pentru Relație_N

Post-condiții:

if ($Pc_{R\_N} \geq Pc_{R\_PRAG}$)

Relație_N in Lp_R

endif

Cunoscând relațiile care trebuie partiționate, cele din lista Lp_R , trebuie determinate cheile de partiționare optime pentru aceste relații. Cheile de partiționare sunt determinate de predicatele de selecție definite pe acea relație.

Fiind dată o relație *R(A1, A2, ..., An)*, unde *Ai* este un atribut al relației definit pe domeniul *Di*, un *predicat simplu pj* definit pe relația *R* are forma: *pj:Ai θ Valoare*, unde θ aparține{=, >, <, ≤, ≥, ≠} şi *Valoare* aparține *Di*.

Un *predicat de selecție* pentru relația *R* poate fi de forma *pj*, caz în care este un predicat simplu de selecție, sau (*pj AND pk AND ...*), caz în care este un predicat compus de selecție, format din conjuncția mai multor predicate simple. Partiționarea unei relații este determinată de predicatele de selecție ale relației.

Determinarea cheilor de partiționare ale relațiilor din lista Lp_R este specificată formal mai jos:

Funcția: IdentificareCheiPartiționare

Descriere: Sunt identificate cheile de partiționare ale relației

Intrări: Lista tranzacțiilor critice L'_T, lista relațiilor partiționate Lp_R, operațiile executate de tranzacții asupra relațiilor

Sursa: SelecțieCantitativă2Tranzacție, SelecțieRelațiiPartiționate, OperațiiTranzacție

Iesiri: Lista seturilor de atribute ale cheilor de partiționare $L_{Chei\_partiționare\_R\_N}$ pentru Relație_N

Destinația: CheiPartiționarePotențiale

Pre-condiții: Este construită lista tranzacțiilor critice L'_T, lista relațiilor partiționate Lp_R, și sunt determinate operațiile executate de tranzacții asupra relațiilor

Post-condiții:

```
if (Relație_N in Lp_R)
        for i=1 to M            ; M este egal cu numărul de tranzacții din L'_T
                {
                Tranzacție=ID_Tranzacție(i)
                if (Căutare_Tranzacție=x)
                        if (Atribute_WHERE există în L_Chei_partiționare_R_N)
                                Contor_Atribute=Contor_Atribute+F_T_N(i)
                        else
                                Atribute_WHERE in L_Chei_partiționare_R_N
                        endif
                        if (Atribute_GROUP_BY există în L_Chei_partiționare_R_N)
                                Contor_Atribute=Contor_Atribute+ F_T_N(i)
                        else
                                Atribute_GROUP_BY in L_Chei_partiționare_R_N
                        endif
                endif
                if (Join_Tranzacție=x)
                        if (Atribute_JOIN există în L_Chei_partiționare_R_N)
                                Contor_Atribute=Contor_Atribute+ F_T_N(i)
                        else
                                Atribute_JOIN in L_Chei_partiționare_R_N
                        endif
                endif
                if (Modificare_Tranzacție=x)
                        if (Atribute_WHERE există în L_Chei_partiționare_R_N)
                                Contor_Atribute=Contor_Atribute+ F_T_N(i)
                        else
```

Atribute_WHERE in $L_{Chei\_partiționare\_R\_N}$

endif

endif

if (Ştergere_Tranzacție=x)

if (Atribute_WHERE există în $L_{Chei\_partiționare\_R\_N}$)

Contor_Atribute=Contor_Atribute+ $F_{T\_N(i)}$

else

Atribute_WHERE in $L_{Chei\_partiționare\_R\_N}$

endif

endif

}

endif

Fiecare cheie posibilă de partiționare are ataşat un contor care indică de câte ori a fost folosit un predicat de selecție, de forma cheii de partiționare, pentru consultarea sau prelucrarea relației bazei de date. Cheile de partiționare care au asociată cea mai mare valoare a contorului sunt cele care vor fi folosite pentru partiționarea relațiilor. Funcția prin care sunt selectate aceste chei de partiționare este următoarea:

Funcția: SelecțieCheiePartiționare

Descriere: Este selectată, din lista cheilor de partiționare posibile, acea cheie de partiționare care va fi folosită pentru partiționarea relației

Intrări: Lista cheilor de partiționare potențiale $L_{Chei\_partiționare\_R\_N}$ pentru Relație_N,

Sursa: IdentificareCheiPartiționare

Iesiri: Cheia de partiționare $Cheie\_Partiționare_{R\_N}$ pentru Relație_N

Destinația: TipFişierDate

Pre-condiții: Este construită lista cheilor de partiționare potențiale $L_{Chei\_partiționare\_R\_N}$ ale relației

Post-condiții:

$Contor\_Cheie\_Partiționare_{R\_N}$ = Contor_Atribute(0)

$Cheie\_Partiționare_{R\_N}$ = Atribute_Chei_Partiționare(0)

for i=1 to M ; M este egal cu numărul de elemente ale listei $L_{Chei\_partiționare\_R\_N}$

if (Contor_Atribute(i) > $Contor\_Cheie\_Partiționare_{R\_N}$)

Cheie_Partiționare$_{R_N}$ = Atribute_Chei_Partiționare(i)

endif

Odată identificate cheile de partiționare pentru relațiile de dimensiuni mari la nivelul cărora numărul prelucrărilor concurente este considerabil, astfel încât să justifice partiționarea relație, se poate continua analiza în vederea stabilirii relațiilor care este indicat să aibă asociate structuri auxiliare de acces. Cheile de indexare cele mai indicate pentru construirea structurilor indexate sunt şi ele, de asemenea, determinate.

Pasul 8. ***Identificarea structurilor auxiliare de acces.***

Studiile efectuate în capitolul 3 au arătat că pentru relații de dimensiune mică, sub 450 Ko, nu se obțin îmbunătățiri ale performanței dacă se construiesc structuri auxiliare de acces. Din contră, pentru relații de dimensiuni sub 150 Ko performanțele se înrăutățesc considerabil.

Având o primă estimare a dimensiunilor relațiilor de bază pot fi eliminate din lista relațiilor critice, L_R, acele relații a căror dimensiune este foarte mică, sub Dim_{PRAG}, şi a căror performanță în exploatare nu poate fi îmbunătățită prin introducerea unor structuri auxiliare de acces. Funcția cu ajutorul căreia sunt eliminate aceste relații are următoarea specificație formală:

Funcția: EliminareRelațieMică

Descriere: Sunt eliminate relațiile din L_R a căror dimensiune nu depăşeşte o valoare minimă admisă Dim_{PRAG}

Intrări: Lista relațiilor critice L_R şi lista relațiilor de dimensiune mică Lm_R

Sursa: SelecțieCantitativăRelație, RelațieDimensiuneMică

Iesiri: Lista relațiilor critice L'_R

Destinația: CalculIndiceActivitate

Pre-condiții: Este obținută lista relațiilor critice L_R şi lista relațiilor de dimensiune mică Lm_R

Post-condiții:

$L'_R = L_R - Lm_R$

Lista relațiilor de dimensiune mică Lm_R este obținută cu ajutorul funcției RelațieDimensiuneMică a cărei specificație formală este:

Funcția: RelațieDimensiuneMică

Descriere: Sunt determinate relațiile a căror dimensiune nu depăşeşte o valoare minimă admisă Dim_{PRAG}

Intrări: Dimensiunea estimată D_{R_N} pentru Relație_N, dimensiunea de prag Dim_{PRAG}

Sursa: EstimarePrimarăDimensiuneRelație

Iesiri: Relație_N

Destinația: Lista relațiilor de dimensiune mică Lm_R

Pre-condiții: Este estimată dimensiunea D_{R_N} pentru Relație_N

Post-condiții:

if ($D_{R\_N} \leq Dim_{PRAG}$)

Relație_N in Lm_R

endif

Studiile efectuate în capitolul 3 au arătat că valoarea pentru Dim_{PRAG} trebuie să fie inferioară valorii de 450 Ko. Relațiile din lista relațiilor critice L'_R sunt relațiile pentru care este indicat să se analizeze oportunitatea construirii unor structuri auxiliare de acces pentru a îmbunătăți performanța în exploatare a bazei de date. Analiza calitativă a tranzacțiilor este cea care stabileşte necesitatea ca o relație să aibă asociate structuri auxiliare de accces pentru a micşora timpii de răspuns la interogările formulate asupra bazei de date şi implicit a creşte performanța bazei de date. Tipul operațiilor efectuate de tranzacțiile critice asupra relațiilor critice determină eficiența sau ineficiența unor structuri indexate. Pentru cuantificarea oportunității creării unor structuri indexate asociate fişierelor de date am definit doi indicatori: *indicele de activitate* şi *indicele tipului de activitate*. Aceştia trebuie evaluați pentru fiecare relație critică a listei L'_R şi în funcție de valoarea lor se va stabili necesitatea creării de structuri indexate asociate relației. Indicele de activitate $I_{activitate}$ este un indicator pe care l-am definit pentru a putea analiza tipul de activitate care se desfăşoară cu precădere la nivelul unei relații de bază. Interpretând valoarea acestui indice se poate şti dacă relațiile sunt cu precădere consultate sau actualizate.

Indicele de activitate este definit astfel:

$$I_{activitate} = \frac{N_C}{N_I + N_M + N_S}$$

Unde

N_C –este numărul operațiilor de căutare

N_I – este numărul operațiilor de inserare

N_M – este numărul operațiilor de modificare

N_S – este numărul operațiilor de ştergere

Dacă $I_{activitate} > 1$ relația este mai adesea consulatată decât actualizată prin inserare, modificare sau ştergere, deci ea este o bună candidată pentru o structură de acces auxiliară.

Dacă $I_{activitate} \leq 1$ relația este mai adesea actualizată prin inserare, modificare sau ştergere decât consulatată, deci ea nu este întotdeauna o bună candidată pentru o structură de acces auxiliară.

Formal această analiză se prezintă astfel:

Funcția: CalculIndiceActivitate

Descriere: Este calculat indicele de activitate $I_{activitate\_R\_N}$ al relației, pentru relațiile din lista relațiilor critice L'_R

Intrări: Numărul operațiilor de căutare NC_{R_N}, numărul operațiilor de inserare NI_{R_N}, numărul operațiilor de modificare NM_{R_N} şi numărul operațiilor de ştergere NS_{R_N} efectuate asupra Relație_N, lista relațiilor critice L'_R

Sursa: NumărOperațiiCăutareRelație, NumărOperațiiInserareRelație, NumărOperațiiModificareRelație, NumărOperațiiŞtergereRelație, EliminareRelațieMică

Iesiri: Indicele de activitate $I_{activitate\_R\_N}$

Destinația: Selecție1RelațieIndexabilă

Pre-condiții: Este calculat numărul operațiilor de căutare NC_{R_N}, numărul operațiilor de inserare NI_{R_N}, numărul operațiilor de modificare NM_{R_N} şi numărul operațiilor de ştergere NS_{R_N} efectuate asupra Relație_N şi este construită lista relațiilor critice L_R

Post-condiții:

if (Relație_N in L'_R)

$$I_{activitate\_R\_N} = \frac{NC_{R\_N}}{NI_{R\_N} + NM_{R\_N} + NS_{R\_N}}$$

endif

Relațiile ce au indicele de activitate supraunitar sunt relații pretabile pentru structuri indexate. Aceste relații vor fi identificate şi selectate în lista L_{RI}. Funcția de selecție este definită formal astfel:

Funcția: SelecțielRelațieIndexabilă

Descriere: Sunt selectate relațiile bune candidate pentru construirea unor structuri indexate

Intrări: Indicele de activitate $I_{activitate\_R\_N}$ pentru Relație_N

Sursa: CalculIndiceActivitate

Iesiri: Lista relațiilor bune candidate pentru structuri auxiliare de acces L_{RI}

Destinația: IdentificareAtributeVerzi, IdentificareAtributeRoşii, Selecție2RelațieIndexabilă

Pre-condiții: Este calculat indicele de activitate $I_{activitate\_R\_N}$ al relațiilor critice din lista L'_R

Post-condiții:

if ($I_{activitate\_R\_N} > 1$)

Relație_N in L_{RI}

endif

Pentru relațiile care au $I_{activitate}$ subunitar este posibil, totuşi, ca existența unei structuri indexate să imbunătățească performanța bazei de date. Testele efectuate în capitolul 3 au arătat că, dacă operațiile de modificare şi/sau ştergere nu implică mai mult de 10-15% din tuplurile din relație, este oportun să existe index pe condiția de căutare a clauzei WHERE din blocul UPDATE sau DELETE. In această situație timpii de prelucrare sunt mai mici decât cei obținuți în absența indexului. Prin urmare, am definit un alt indicator cu ajutorul căruia se poate cuantifica oportunitatea unor structuri indexate pentru relațiile la care operațiile de

selecție nu sunt prioritare. Pe acest indice l-am numit *indicele tipului de activitate* $I_{tip\_activitate}$ ce se desfăşoară la nivelul relației. Acest indice este definit astfel:

$$I_{tip\_activitate} = \frac{N_I}{N_M + N_S}$$

Unde

N_I – este numărul operațiilor de inserare

N_M – este numărul operațiilor de modificare

N_S – este numărul operațiilor de ştergere

Dacă $I_{tip\_activitate} \geq 1$ relația nu este o bună candidată pentru o structură auxiliară de acces, deoarece numărul operațiilor de inserare excede numărului operațiilor de modificare şi ştergere.

Dacă $I_{tip\_activitate} < 1$ relația este o bună candidată pentru o structură auxiliară de acces, deoarece numărul operațiilor de inserare este mai mic decât cel al operațiilor de modificare şi ştergere.

Sunt identificate relațiile din lista relațiilor critice L'_R care au indicele de activitate subunitar, după care este calculat indicele tipului de activitate pentru aceste relații şi sunt selectate în lista L_{RI} a relațiilor bune candidate pentru structuri auxiliare de acces şi acele relații care au indicele tipului de activitate subunitar. Formal această nouă analiză se prezintă astfel:

Funcția: Selecție2RelațieIndexabilă

Descriere: Sunt selectate relațiile din lista relațiilor critice L'_R care au $I_{activitate\_R\_N}$ subunitar

Intrări: Lista relațiilor critice L'_R şi lista relațiilor bune candidate pentru structuri indexate L_{RI}

Sursa: EliminareRelațieMică, Selecție1RelațieIndexabilă

Iesiri: Lista relațiilor ce nu sunt bune candidate pentru structuri auxiliare de acces L'_{RI}

Destinația: CalculIndiceTipActivitate

Pre-condiții: Este construită lista relațiilor critice L'_R şi lista relațiilor critice L_{RI}

Post-condiții:

$L'_{RI} = L'_R - L_{RI}$

Relațiile pentru care indicele de activitate este subunitar sunt analizate în continuare pentru a se stabili dacă pot fi totuşi bune candidate pentru structuri auxiliare de acces. În acest scop se calculează pentru ele indicele tipului de activitate.

Funcția: CalculIndiceTipActivitate

Descriere: Este calculat indicele tipului de activitate $I_{tip\_activitate\_R\_N}$ al relației pentru relațiile din lista relațiilor critice L'_{RI}

Intrări: Numărul operațiilor de inserare NI_{R_N}, numărul operațiilor de modificare NM_{R_N} şi numărul operațiilor de ştergere NS_{R_N} efectuate asupra Relație_N, lista relațiilor L'_{RI}, procentul tuplurilor prelucrate

Sursa: NumărOperațiiInserareRelație, NumărOperațiiModificareRelație, NumărOperațiiŞtergereRelație, Selecție2RelațieIndexabilă

Iesiri: Indicele tipului de activitate $I_{tip\_activitate\_R\_N}$

Destinația: Selecție3RelațieIndexabilă

Pre-condiții: Este calculat numărul operațiilor de inserare NI_{R_N}, numărul operațiilor de modificare NM_{R_N} şi numărul operațiilor de ştergere NS_{R_N} efectuate asupra Relație_N

Post-condiții:

if (Relație_N in L'_{RI}) and (ProcentTupluriPrelucrate <15)

$$I_{tip\_activitate\_R\_N} = \frac{NI_{R\_N}}{NM_{R\_N} + NS_{R\_N}}$$

endif

Relațiile ce au indicele tipului de activitate subunitar sunt relații pretabile pentru structuri indexate. Aceste relații vor fi identificate şi selectate în lista L_{RI}. Funcția de selecție este definită formal astfel:

Funcția: Selecție3RelațieIndexabilă

Descriere: Sunt selectate relațiile bune candidate pentru construirea unor structuri indexate

Intrări: Indicele de activitate $I_{tip\_activitate\_R\_N}$ pentru Relație_N

Sursa: CalculIndiceTipActivitate

Iesiri: Lista relațiilor bune candidate pentru structuri auxiliare de acces L_{RI}

Destinația: Identificare AtributeVerzi, Identificare AtributeRoşii

Pre-condiții: Este calculat indicele tipului de activitate $I_{tip\_activitate\_R\_N}$ al relațiilor critice din lista L'_{RI}

Post-condiții:

if ($I_{tip\_activitate\_R\_N} < 1$)

Relație_N in L_{RI}

endif

După ce au fost identificate relațiile care este indicat să aibă structuri auxiliare de acces la date, se va pune problema identificării atributelor ce vor alcătui cheile de indexare. Determinarea cheilor de indexare este rezultatul unei analize calitative a operațiilor efectuate de tranzacțiile critice asupra fiecărei relații critice din lista L_{RI} în parte. Structurile auxiliare de acces sunt benefice în special în cazul operațiilor de căutare executate asupra bazei de date, pentru că ele pot micşora considerabil timpii de răspuns la interogări şi deci pot îmbunătăți performanța sistemului. De asemenea, aceste structuri pot fi eficiente şi în cazul unor operații de modificare şi ştergere executate în anumite condiții restrictive asupra bazei de date. Este important ca aceste structuri indexate să fie construite pe baza unor chei de indexare care să ofere o acoperire maximă şi eficientă a interogărilor formulate asupra bazei de date. Am introdus două noi noțiuni, acelea de *atribute verzi* şi *atribute roşii*. Atributele verzi sunt atributele relației care este indicat să facă parte din cheile de indexare, iar atributele roşii sunt atributele care nu este indicat să intre în componența unor chei de indexare. Rezultatele obținute în urma analizării relațiilor critice în vederea determinării atributelor verzi şi a atributelor roşii sunt colectate în tabelul 6.

Tabel 6. Atribute roşii şi atribute verzi.

Nume Relație	Cheie Primară	Identificator Tranzacție	Atribute		Cheie de Indexare
			Verzi	Roşii	

Unde

- ***Nume Relație*** – este numele relației de bază obținut din Tabelul 5.
- ***Cheie Primară*** – este cheia primară a relației.
- ***Identificator Tranzacție*** – este identificatorul tranzacției pentru care sunt identificate atributele verzi şi atributele roşii.
- ***Atribute Verzi*** – sunt atributele indicate a fi utilizate la alcătuirea cheilor de indexare.
- ***Atribute Roşii*** – sunt atributele care nu este indicat să fie folosite la alcătuirea cheilor de indexare.
- ***Cheie de Indexare*** – cuprinde toate posibilele chei de indexare identificate pentru acea relație.

Atributele verzi corespund atributelor utilizate în criteriile de căutare, în criteriile de join şi în criteriile de selecție a înregistrărilor care vor fi supuse operațiilor de modificare sau ştergere, pentru acele operații care nu prelucrează mai mult de 10-15% din înregistrările relației.

Atributele roşii corespund atributelor modificate în cadrul operațiilor de modificare şi, de asemenea, sunt implicate de operațiile de inserare şi ştergere.

Am definit următoarele reguli de încadrare a atributelor relațiilor în categoria atributelor verzi sau a atributelor roşii:

INSERT → Atribute Roşii = Toate atributele (în special în cazul inserărilor masive de înregistrări)

UPDATE → Atribute Roşii = Atributele actualizate
Atribute Verzi = Atributele utilizate în condițiile WHERE

DELETE → Atribute Roşii = Toate atributele (în special în cazul ştergerilor masive

de înregistrări)

Atribute Verzi = Atributele utilizate în condițiile WHERE

SELECT → Atribute Verzi = Atributele utilizate în condițiile WHERE
Atribute Verzi = Atributele utilizate în condițiile de JOIN
Atribute Verzi = Atributele utilizate în condițiile GROUP BY
Atribute Verzi = Atributele utilizate în condițiile ORDER BY
Atribute Verzi = Atributele utilizate în clauza DISTINCT

Pentru fiecare relație va fi obținută o listă cu atribute verzi şi o listă cu atribute roşii. Aceste liste sunt utilizate pentru a genera posibilele chei de indexare.

Mecanismul prin care sunt identificate atributele verzi şi atributele roşii este specificat formal astfel:

Funcția: IdentificareAtributeVerzi

Descriere: Sunt identificate seturile de atribute verzi ale relației

Intrări: Lista tranzacțiilor critice L'_T, lista relațiilor critice indexabile L_{RI}, operațiile executate de tranzacții asupra relațiilor, procentul tuplurilor prelucrate de operațiile de modificare sau ştergere

Sursa: SelecțieCantitativă2Tranzacție, Selecție3RelațieIndexabilă, OperațiiTranzacție

Iesiri: Lista seturilor de atribute verzi $L_{Verzi\_R\_N}$ pentru Relație_N

Destinația: CheiIndexarePotențiale

Pre-condiții: Este construită lista tranzacțiilor critice L'_T, lista relațiilor critice indexabile L_{RI}, şi sunt determinate operațiile executate de tranzacții asupra relațiilor

Post-condiții:

```
if (Relație_N in L_RI)
        for i=1 to M            ; M este egal cu numărul de tranzacții din L'_T
                {
                Tranzacție=ID_Tranzacție(i)
                if (Căutare_Tranzacție=x)
                        if (Atribute_WHERE există în L_Verzi_R_N)
                                Contor_Atribute=Contor_Atribute+ F_T_N(i)
                        else
                                Atribute_WHERE in L_Verzi_R_N
```

endif

if (Atribute_JOIN există în $L_{Verzi\_R\_N}$)

Contor_Atribute=Contor_Atribute+ $F_{T\_N(i)}$

else

Atribute_JOIN in $L_{Verzi\_R\_N}$

endif

if (Atribute_GROUP_BY există în $L_{Verzi\_R\_N}$)

Contor_Atribute=Contor_Atribute+ $F_{T\_N(i)}$

else

Atribute_GROUP_BY in $L_{Verzi\_R\_N}$

endif

if (Atribute_ORDER_BY există în $L_{Verzi\_R\_N}$)

Contor_Atribute=Contor_Atribute+ $F_{T\_N(i)}$

else

Atribute_ORDER_BY in $L_{Verzi\_R\_N}$

endif

if (Atribut_DISTINCT există în $L_{Verzi\_R\_N}$)

Contor_Atribute=Contor_Atribute+ $F_{T\_N(i)}$

else

Atribut_DISTINCT in $L_{Verzi\_R\_N}$

endif

endif

if (Modificare_Tranzacție=x) and (ProcentTupluriPrelucrate<15)

if (Atribute_WHERE există în $L_{Verzi\_R\_N}$)

Contor_Atribute=Contor_Atribute+ $F_{T\_N(i)}$

else

Atribute_WHERE in $L_{Verzi\_R\_N}$

endif

endif

if (Ştergere_Tranzacție=x) and (ProcentTupluriPrelucrate<15)

if (Atribute_WHERE există în $L_{Verzi\_R\_N}$)

Contor_Atribute=Contor_Atribute+ $F_{T\_N(i)}$

else

Atribute_WHERE in $L_{Verzi\_R\_N}$

```
                    endif
                endif
                }
        endif
```

Funcția folosește pentru determinarea tuturor atributelor verzi, cele pretabile pentru cheile de indexare, și contorizează numărul de utilizări ale fiecărui atribut verde în parte. Pe lângă atributele verzi trebuie determinate și atributele roșii, cele care nu este indicat să aparțină cheilor de indexare. Funcția folosită pentru determinarea atributelor roșii este specificată formal mai jos:

Funcția: IdentificareAtributeRoșii

Descriere: Sunt identificate seturile de atribute roșii ale relației

Intrări: Lista tranzacțiilor critice L'_T, lista relațiilor critice indexabileL_{RI}, operațiile executate de tranzacții asupra relațiilor

Sursa: SelecțieCantitativă2Tranzacție, Selecție3RelațieIndexabilă, OperațiiTranzacție

Iesiri: Lista seturilor de atribute roșii $L_{Roşii\_R\_N}$ pentru Relație_N

Destinația: CheiIndexarePotențiale

Pre-condiții: Este construită lista tranzacțiilor critice L'_T, lista relațiilor critice indexabile L_{RI}, și sunt determinate operațiile executate de tranzacții asupra relațiilor

Post-condiții:

```
if (Relație_N in L_RI)
        for i=1 to M            ; M este egal cu numărul de tranzacții din L'_T
                {
                Tranzacție=ID_Tranzacție(i)
                if (Inserare_Tranzacție=x)
                        if (Atribute_Relație_N există în L_Roşii_R_N)
                                Contor_Atribute=Contor_Atribute+ F_T_N(i)
                        else
                                Atribute_Relație_N in L_Roşii_R_N
                        endif
                endif
                if (Modificare_Tranzacție=x)
```

```
                    if (Atribute_SET există în L_Roşii_R_N)
                            Contor_Atribute=Contor_Atribute+ F_T_N(i)
                    else
                            Atribute_SET in L_Roşii_R_N
                    endif
            endif
            if (Ştergere_Tranzacţie=x)
                    if (Atribute_Relaţie_N există în L_Roşii_R_N)
                            Contor_Atribute=Contor_Atribute+ F_T_N(i)
                    else
                            Atribute_Relaţie_N in L_Roşii_R_N
                    endif
            endif
            }
    endif
```

Funcţia determină atributele roşii ale relaţiilor, cele care nu este indicat să apară în cheile de indexare pentru că duc la degradarea performanţelor sistemului. După ce seturile de atribute roşii şi verzi au fost determinate, pot fi stabilite cheile de indexare potenţiale. Acestea se obţin prin excluderea din lista atributelor verzi a acelor seturi de atribute care apar şi în lista atributelor roşii şi au o valoare a contorului asociat mai mică decât valoarea pe care a au asociată, atunci când se găsesc în lista atributelor roşii.

Formal funcţia este specificată astfel:

Funcţia: CheiIndexarePotenţiale

Descriere: Sunt identificate seturile de atribute verzi ale relaţiei care pot constitui cheile de indexare pentru relaţie

Intrări: Lista atributelor verzi $L_{Verzi\_R\_N}$ şi lista atributelor roşii $L_{Roşii\_R\_N}$ ale relaţiei

Sursa: IdentificareAtributeVerzi, IdentificareAtributeRoşii

Iesiri: Lista cheilor de indexare potenţiale $L_{Chei\_index\_R\_N}$ pentru Relaţie_N

Destinaţia: SelecţieCheiIndex

Pre-condiţii: Este construită lista atributelor verzi $L_{Verzi\_R\_N}$ şi lista atributelor roşii $L_{Roşii\_R\_N}$ ale relaţiei

Post-condiții:

for i=1 to M ; M este egal cu numărul de elemente din lista $L_{Verzi\_R\_N}$

if (Contor_Atribute(i)$_{Atribut\_N\_Verde}$ < Contor_Atribute$_{Atribut_N_Roşu}$)

$L_{Chei\_index\_R\_N} = L_{Verzi\_R\_N}$ – Atribut_N_Verde

endif

Din lista cheilor de indexare potențiale vor fi alese acele chei de indexare care vor fi folosite pentru crearea de structuri auxiliare de acces ale relațiilor bazei de date. Se poate specifica o valoare de prag, A_{PRAG}, a contorului asociat unei chei de indexare potențiale care să fie folosit pentru a selecta cheile de indexare care vor fi folosite pentru crearea de structuri auxiliare de acces. Studiile efectuate în capitolul 3 au arătat cum numărul structurilor auxiliare de acces influențează performanțele bazei de date. Ținând cont de aceste rezultate vor fi alese valorile de prag A_{PRAG}. Funcția pe baza căreia se face selecția cheilor de indexare este SelecțieCheiIndex şi este specificată formal astfel:

Funcția: SelecțieCheiIndex

Descriere: Sunt selectate, din lista cheilor de indexare posibile, acele chei de indexare care vor fi folosite pentru construcția de structuri auxiliare de acces pentru relație

Intrări: Lista cheilor de indexare potențiale $L_{Chei\_index\_R\_N}$ pentru Relație_N, valoarea de prag A_{PRAG} a contorului setului de atribute care alcătuiesc cheia de indexare

Sursa: CheiIndexarePotențiale

Iesiri: Lista cheilor de indexare $L_{Chei\_index\_R\_N}$ pentru Relație_N

Destinația: Schema fizică

Pre-condiții: Este construită lista cheilor de indexare potențiale $L_{Chei\_index\_R\_N}$ ale relației

Post-condiții:

for i=1 to M ; M este egal cu numărul de elemente ale listei $L_{Chei\_index\_R\_N}$

{

Cheie_Index = Atribute_Chei_Index(i)

if (Contor_Cheie_Index < A_{PRAG})

$L_{Chei\_index\_R\_N} = L_{Chei\_index\_R\_N}$ – Cheie_Index

endif

}

Având cheile de indexare ale relațiilor critice din lista L_{RI} se poate trece la următorii paşi ai metodologiei.

Pasul 9. ***Alegerea organizării fişierelor de date.***

Pentru fiecare relație de bază identificată, care este prezentă în Tabelul 1, va fi necesar să se aleagă o structură de date folosită pentru organizarea la nivel fizic a datelor relației. Aceste structuri trebuie alese cu mare atenție, ținând cont de toate analizele anterioare, pentru a obține o organizare fizică optimă a bazei de date, care va oferi o foarte bună performanță în exploatare a bazei de date.

Relațiile prezente în lista Lm_R sunt relații de dimensiune mică. Din studiile efectuate în capitolul 3 rezultă că structura optimă pentru aceste relații este cea de fişier secvențial (heap).

Tot studiile efectuate în capitolul 3 mi-au permis să identific cele mai bune decizii de proiectare vis-à-vis de alegerea structurii fişierelor de date, şi anume.

Relațiile prezente în lista Lp_R sunt relații ce trebuie partiționate. Tipul de partiționare este dictat de valorile posibile ale cheii de partiționare [Datta 90]. Dacă se folosesc selecțiile pe intervale de valori este indicat ca partiția să fie de tip *range*. Dacă selectivitatea atributelor cheii de partiționare este mică, ceea ce implică un număr mic de valori distincte ale cheii de partiționare, este indicat să se utilizeze partiționarea de tip *list*. Selectivitatea cheii de partiționare se obține după o formulă similară selectivității cheii de indexare, doar că în acest caz cheia de indexare este înlocuită cu cheia de partiționare a relației.

$$\text{Selectivitate}_{\text{Cheie\_partiționare}} = \frac{\text{Număr valori distincte}_{\text{Cheie\_partiționare}}}{\text{Număr total tupluri relație}}$$

Pentru toate celelalte situații se va folosi partiționarea de tip *hash*.

Dacă lista cheilor posibile de indexare $L_{Chei\_index\_R\_N}$ ale unei relații conține doar o cheie de indexare care este identică cu cheia primară a relației, se va alege pentru acea relație o structură de tip tabelă organizată indexat. De asemenea, dacă numărul cheilor de indexare posibile nu depăşeşte valoarea 4 şi una dintre aceste chei este identică cu cheia primară a relației, se poate opta tot pentru structura de tabelă organizată indexat.

Sunt indicate, de asemenea, relațiile care este posibil să fie clusterate pentru că sunt folosite cel mai adesea împreună în cadrul interogărilor care operează pe baza de date.

Pasul alegerii organizării fișierelor de date este specificat formal mai jos:

Funcția: TipFișierDate

Descriere: Este ales tipul fișierului de date care va stoca datele relației

Intrări: Lista relațiilor de dimensiune mică Lm_R, lista relațiilor partiționate Lp_R și cheia de partiționare a fiecărei relații din lista Lp_R, lista relațiilor bune candidate pentru structuri auxiliare de acces L_{RI} și lista cheilor de indexare $L_{Chei\_index\_R\_N}$ pentru fiecare Relație_N care aparține lui L_{RI}, cheia primară pentru Relație_N

Sursa: RelațieDimensiuneMică, SelecțieRelațiiPartiționate, SelecțieCheiePartiționare Selecție1RelațieIndexabilă, Selecție3RelațieIndexabilă, SelecțieCheiIndex, ProiectareRelație

Iesiri: Tipul fișierului de date care va stoca datele pentru Relație_N

Destinația: Schema fizică

Pre-condiții: Este determinată lista relațiilor de dimensiune mică Lm_R, lista relațiilor partiționate Lp_R și cheia de partiționare a fiecărei relații din lista Lp_R, lista relațiilor bune candidate pentru structuri auxiliare de acces L_{RI} și lista cheilor de indexare $L_{Chei\_index\_R\_N}$ pentru fiecare Relație_N care aparține lui L_{RI}, precum și cheia primară pentru Relație_N

Post-condiții:

if (Relație_N in Lm_R)
    TipFișier = Secvențial
endif
if (Relație_N in Lp_R)
    if ($Selectivitate_{Cheie\_partiționare\_R\_N} < 5\%$)
        TipFișier = Partiționat_deTip_List
    else
        if (Cheie_Partiționare$_{R_N}$ folosește intervale de valori)
            TipFișier = Partiționat_deTip_Range
        else
            TipFișier = Partiționat_deTip_Hash
        endif
    endif
endif
if (Relație_N in L_{RI})

```
        if (Cheie_Primară R_N in L_Chei_index_R_N ) and (Nr_Elem(L_Chei_index_R_N ) < 4)
                TipFişier = Tabelă_Organizată_Indexat
        else
                TipFişier = Secvențial
        endif
else
        TipFişier = Secvențial
endif
if (Contor_Join_Relație R_N _Relație R_M = MAX )
        TipFişier = Cluster
endif
```

Funcția TipFişierDate se va aplica fiecărei relații identificate la nivelul schemei logice globale a bazei de date şi prezente în Tabel 1. Ca urmare a aplicării funcției va rezulta organizarea fizică a bazei de date. Va fi ales tipul optim de structură fizică pentru fiecare relație a bazei de date. Rezultatele proiectării fizice a bazei de date pot fi colectate în următorul tabel (Tabel 7):

Tabel 7. Structurile fişierelor de date pentru relațiile de bază.

Nume Relație	Tip Fişier de Date	Partiționat	Cheie de Partiționare	Tip Partiție

Unde

- ***Nume Relație*** – este numele relației, pentru relațiile identificate în Tabelul 1.
- ***Tip Fişier de Date*** – indică tipul fişierului de date ales pentru acea relație (de exemplu, HEAP, HASH, INDEXED ORGANIZED, CLUSTERED, etc.).
- ***Partiționat*** – specifică dacă acea relație va fi partiționată sau nu (DA, NU).
- ***Cheie de Partiționare*** – specifică cheia pe baza căreia este partiționată relația.

- ***Tip Partiție*** – specifică tipul partiției (de exemplu, RANGE, LIST, HASH, etc.)

Structurile fişierelor de date vor fi alese dintre structurile pe care SGBD-ul ales pentru implementarea bazei de date le suportă.

Pasul 10. ***Alegerea organizării structurilor indexate.***

Plecând de la analiza făcută în pasul 8 al metodologiei, va fi construită o listă cu toate structurile de indexare posibile, acelea care ajută la îmbunătățirea performanțelor bazei de date. Tabelul 6 conține toate relațiile bazei de date care pot fi indexate în vederea creşterii performanței bazei de date, şi pentru fiecare variantă de indexare în parte şi cheia de indexare optimă. Pornind de la aceste date şi de la structurile fişierelor de date stabilite în pasul 9 al metodologiei se va decide care vor fi structurile auxiliare de acces ce se vor implementa la nivelul bazei de date şi care va fi tipul de index folosit pentru fiecare structură în parte. Studiile efectuate în capitolul 3 au ajutat la identificarea unora dintre deciziile optime de proiectare la nivelul alegerii organizării structurilor indexate. De asemenea, în [Micro 03-6], [O'Neil97], [Valdu 87], [Mitea 04-3] pot fi consultate criterii de alegere a structurilor indexate în vederea creşterii performanței bazei de date.

Formal acest pas al metodologiei este specificat astfel:

Funcția: TipIndex

Descriere: Este ales tipul indexului asociat relației

Intrări: Lista relațiilor pretabile la indexare L_{RI}, lista cheilor de indexare ale relației $L_{Cheie\_index\_R\_N}$, tipul fişierului de date pentru Relație_N, cheia de partiționare pentru relațiile partiționate, selectivitatea cheii de indexare, cheia primară pentru Relație_N, punctajul activității concurente de la nivelul relației Pc_{R_N}

Sursa: Selecție3RelațieIndexabilă, SelecțieCheiIndex, TipFişierDate, SelecțieCheiePartiționare, SelectivitateCheieIndex, ProiectareRelație, CalculPunctajConcurențăRelație

Iesiri: Tipul indexului care va fi asociat pentru Relație_N

Destinația: Schema fizică

Pre-condiții: Este determinată lista relațiilor bune candidate pentru structuri auxiliare de acces L_{RI} şi lista cheilor de indexare $L_{Chei\_index\_R\_N}$ pentru fiecare Relație_N care aparține lui L_{RI}, este stabilit tipul fişierului de date care va stoca datele pentru Relație_N, este aleasă

cheia de partiționare pentru fiecare relație partiționată, este calculată selectivitatea cheii de indexare, este specificată cheia primară a relației

Post-condiții:

if ($TipFişier_{R\_N}$ = Partiționat)

if ($Cheie\_Index_{R\_N}$ = $Cheie\_Partiționare_{R\_N}$)

TipIndex = (IndexLocalPartiție, ArboreB)

else

TipIndex = (IndexGlobal, ArboreB)

endif

if ($Selectivitate_{Cheie\_index\_R\_N}$ < 1%)

TipIndex[2] = Bitmap

endif

else

if ($TipFişier_{R\_N}$ = Tabelă_Organizată_Indexat)

and ($Cheie\_Index_{R\_N}$ <> $Cheie\_Primară_{R\_N}$)

if ($Selectivitate_{Cheie\_index\_R\_N}$ < 1%)

TipIndex = (IndexSecundar, Bitmap)

else

TipIndex = (IndexSecundar, ArboreB)

endif

else

if ($TipFişier_{R\_N}$ = Secvențial)

if ($Cheie\_Index_{R\_N}$ = $Cheie\_Primară_{R\_N}$)

if ($Selectivitate_{Cheie\_index\_R\_N}$ < 1%)

TipIndex = (IndexPrimar, Bitmap)

else

TipIndex = (IndexPrimar, ArboreB)

endif

else

if ($Selectivitate_{Cheie\_index\_R\_N}$ < 1%)

TipIndex = (IndexSecundar, Bitmap)

else

if (Pc_{R_N} > Pc_{PRAG})

TipIndex = (IndexCheieInversată, ArboreB)

```
                                    else
                                        TipIndex = (IndexSecundar, ArboreB)
                                    endif
                                endif
                            endif
                        endif
                    endif
                endif
    if (Contor_Join_Relație R_N _Relație R_M = MAX ) and (Relație_M aparține LM R)
        TipIndex = (IndexJoin, Bitmap)
    endif
```

Selectivitatea cheii de indexare este dată de raportul dintre numărul de valori distincte ale cheii de indexare şi numărul total de tupluri al relației şi se calculează conform funcției:

Funcția: SelectivitateCheieIndex

Descriere: Este calculată selectivitatea cheii de indexare a relației

Intrări: Numărul estimat de tupluri pentru Relație_N, numărul estimat de valori distincte ale cheii de indexare

Sursa: EstimareNumărTupluriRelație, EstimareValoriDistincteCheieIndex

Iesiri: Selectivitatea cheii de indexare $Selectivitate_{Cheie\_index\_R\_N}$

Destinația: TipIndex

Pre-condiții: Este estimat numărul de tupluri pentru Relație_N şi este estimat numărul de valori distincte pentru cheia de indexare

Post-condiții:

$$Selectivitate_{Cheie\_index\_R\_N} = \frac{Nv_{Cheie\_index\_R\_N}}{Nt_{R\_N}}$$

Estimarea numărului de valori distincte ale cheii de indexare a relației se realizează coform următoarei specificații:

Funcția: EstimareValoriDistincteCheieIndex

Descriere: Este estimat numărul de valori distincte ale cheii de indexare a relației

Intrări: Relație_N şi cheia de indexare a relației

Sursa: ProiectareRelație, SelecțieCheiIndex

Iesiri: Numărul estimat de valori distincte ale cheii de indexare $Nv_{Cheie\_index\_R\_N}$

Destinația: SelectivitateCheieIndex

Pre-condiții: Sunt determinate cheile de indexare pentru Relație_N

Post-condiții:

$Lvalori_{Cheie\_index\_R\_N}=\{\}$

$Nv_{Cheie\_index\_R\_N}=0$

do while not eof()

if ($Valoare_{Cheie\_index\_R\_N}$ in $Lvalori_{Cheie\_index\_R\_N}$)

skip

else

$Lvalori_{Cheie\_index\_R\_N} = Lvalori_{Cheie\_index\_R\_N} + Valoare_{Cheie\_index\_R\_N}$

$Nv_{Cheie\_index\_R\_N} = Nv_{Cheie\_index\_R\_N} + 1$

skip

endif

enddo

Rezultatele aplicării funcției TipIndex pot fi colectate într-un tabel de forma tabelului 8. Acesta permite o mai uşoară vizualizare a deciziilor finale de proiectare.

Tabel 8. Structuri de indexare folosite.

Nume Relație	Nume Index	Cheie de Indexare	Lungime Cheie de Indexare	Selectivitate Cheie de Indexare	Tip Index	Implementat

Unde

- ***Nume Relație*** – este numele relațiilor de bază care necesită structuri auxiliare de acces.

- ***Nume Index*** – este numele ales pentru structura indexată.

- ***Cheie de Indexare*** – este cheia de indexare utilizată pentru a construi acea structură indexată.

- ***Lungime Cheie de Indexare*** – este lungimea cheii de indexare. Dacă cheia de indexare are lungime foarte mare nu se va obține o structură indexată eficientă.

- ***Selectivitate Cheie de Indexare*** – este selectivitatea cheii de index dată de numărul de valori distincte ale cheii de indexare. Dacă este un număr mic, de valori distincte pentru cheia de index, va fi mult mai potrivit să se aleagă un index de tip bitmap pentru acea structură de index. Dacă este un număr mare este mult mai potrivit să se utilizeze o structură de index bazată pe arbori B.

- ***Tip Index*** – este tipul ales pentru structura indexată. Poate fi un index normal bazat pe o structură de arbore B, poate fi un index bitmap, poate fi un index construit pe baza inversării cheii de indexare, dacă gradul de utilizare concurentă a acelei relații este mare, poate fi un index partiționat local sau global, dacă relația este partiționată, poate fi un index de tip join, dacă este o operație de join între acea relație şi o altă relație din baza de date, etc.

- ***Implementat*** – specifică dacă indexul va fi implementat sau nu în baza de date.

Tipul structurii de index este unul dintre tipurile suportate de SGBD-ul ales pentru a implementa baza de date. Numărul de structuri indexate, ce vor fi implementate pentru fiecare relație de bază în parte, trebuie analizat cu foarte mare atenție deoarece ele trebuie să realizeze o îmbunătățire a performanțelor bazei de date, nu o deteriorare a lor.

Pasul 11. ***Analiza introducerii unei redundanțe controlate.***

Acest pas are ca scop determinarea oportunității introducerii redundanței la nivelul bazei de date, pentru a îmbunătăți performanțele acesteia. Normalizarea este un procedeu

folosit pentru a decide care sunt atributele care trebuie să aparţină aceleiaşi relaţii. Conform teoriei relaţionale atributele se grupează împreună în cadrul unei relaţii, deoarece între ele există o dependenţă funcţională. Există mai multe forme normale în care poate fi încadrată o relaţie. Pentru ca o relaţie să fie corect definită ea trebuie să se încadreze cel puţin în a treia formă normală. Rezultatul normalizării îl constituie un proiect logic al bazei de date, care are o redundanţă minimă a informaţiei şi o structură coerentă şi consistentă [Mitea 98-1], [Mitea 98-2]. Totuşi, există în literatură [Rob 95], [Conno 01], [Steph 00] argumentări conform cărora sunt situaţii când un proiect al bazei de date normalizat nu prezintă şi o eficienţă maximă de prelucrare. Prin urmare, pot exista situaţii în care s-ar putea dovedi necesar să se accepte introducerea unei redundanţe controlate în favoarea unor performanţe crescute. În [Roger 89] şi [Flemi 89] sunt prezentate cazuri de denormalizare a bazelor de date în vederea creşterii performanţei. Aplicarea denormalizării trebuie avută în vedere numai atunci când se estimează că sistemul nu va fi capabil să îndeplinească cu succes cerinţele de performanţă care i-au fost impuse. Personal sunt de părere că procesul de *denormalizare* trebuie aplicat cât mai rar cu puţinţă, ţinându-se cont şi de următoarele aspecte:

- denormalizarea poate accelera regăsirea datelor, dar încetineşte reactualizarea lor;
- denormalizarea poate compromite consistenţa bazei de date;
- denormalizarea duce la o complexitate sporită a implementării;
- denormalizarea sacrifică flexibilitatea.

Dacă performanţele nu sunt satisfăcătoare şi relaţia are o rată de reactualizare scăzută şi o rată a interogărilor foarte înaltă, denormalizarea poate aduce uneori beneficii. Relaţiile, care se consideră că pot avea probleme în satisfacerea cerinţelor de performanţă, vor fi analizate pentru a se observa dacă denormalizându-le se poate obţine o îmbunătăţire a acestora. Identificarea relaţiilor care pot avea probleme în satisfacerea cerinţelor de performanţă se va face cu ajutorul funcţiei RelaţiiCriticePerformanţă care este specificată formal astfel

---

Funcţia: RelaţiiCriticePerformanţă

Descriere: Sunt determinate relaţiile ce este posibil să nu satisfacă cerinţele de performanţă

Intrări: Lista tranzacțiilor critice în ceea ce privește timpii de execuție Lt_T, operațiile executate de tranzacții asupra relațiilor

Sursa: SelecțieTranzacțiiCriticeTimpExecuție, OperațiiTranzacție

Iesiri: Lista relațiilor critice din punctul de vedere al performanțelor Lcp_R

Destinația: SelecțieRelațiiDenormalizabile

Pre-condiții: Este generată lista tranzacțiilor critice în ceea ce privește timpii de execuție Lt_T şi este aplicată funcția OperațiiTranzacții asupra tuturor tranzacțiilor

Post-condiții:

for i=1 to M ; unde M este numărul total de tranzacții din lista Lt_T

for j=1 to N ; unde N este numărul relațiilor de bază

if (Tranzacție_N_i prelucrează Relație_N_j)

Relație_N_j in Lcp_R

endif

Ea foloseşte funcția SelecțieTranzacțiiCriticeTimpExecuție pentru a determina tranzacțiile care pot avea probleme în a-şi respecta restricțiile de timp de execuție.

Funcția: SelecțieTranzacțiiCriticeTimpExecuție

Descriere: Sunt selectate tranzacțiile critice în ceea ce privește timpii de execuție

Intrări: Punctajul calculat Pt_{T_N} al restricției de timp de execuție pentru Tranzacție_N, punctajul de prag Pt_{PRAG} al restricției de timp de execuție

Sursa: CalculPunctajTimpTranzacție

Iesiri: Lista tranzacțiilor critice în ceea ce privește timpii de execuție Lt_T

Destinația: SelecțieRelațiiDenormalizabile

Pre-condiții: Este calculat punctajul restricției de timp de execuție aferent tranzacției

Post-condiții:

if ($Pt_{T\_N} \geq Pt_{PRAG}$)

ID_Tranzacție_N in Lt_T

endif

Relațiile ce pot fi denormalizate sunt numai acele relații la nivelul cărora numărul operațiilor de actualizare este mai mic decât al operațiilor de interogare. Indicele de activitate

este cel care ne ajută să determinăm aceste relații. Funcția SelecțieRelațiiDenormalizabile este folosită pentru identificarea relațiilor ce pot fi denormalizate. Ea este specificată formal astfel:

Funcția: SelecțieRelațiiDenormalizabile

Descriere: Sunt selectate relațiile ce este posibil să fie denormalizate

Intrări: Indicele de activitate $I_{activitate\_R\_N}$ al relațiilor pretabile la denormalizare, cele din lista Lcp_R

Sursa: CalculIndiceActivitate, RelațiiCriticePerformanță

Iesiri: Lista relațiilor denormalizabile L_{RD}

Destinația: Schema fizică

Pre-condiții: Este calculat indicele de activitate $I_{activitate\_R\_N}$ al relațiilor pretabile la denormalizare, cele din lista Lcp_R

Post-condiții:

if ($I_{activitate\_R\_N} > 1$)

Relație_N in L_{RD}

endif

Relațiile existente în lista L_{RD} sunt relații la nivelul cărora denormalizarea poate fi justificată. Ea va consta în adăugarea unor noi atribute în cadrul relațiilor, atribute care sunt de fapt redundante la nivelul bazei de date. Aceste atribute pot fi fie atribute derivate fie atribute dublate. Atributele derivate sunt de fapt atribute care pot fi calculate pornind de la valorile atributelor deja existente. Pentru a decide dacă un atribut derivat va fi stocat în baza de date sau va fi calculat de fiecare dată când este necesar, trebuie avute în vedere următoarele aspecte:

- costurile suplimentare implicate de stocarea datelor derivate în noi câmpuri ale tabelei și pentru pastrarea acestora în concordanță cu datele operaționale din care provin;
- costurile necesare calculării datelor derivate, de fiecare dată când acest lucru este necesar.

În cazul atributelor dublate ele pot să provină din combinarea relațiilor de tip 1:1 într-o singură relație, din dublarea atributelor care nu sunt chei în relații de tip 1:M sau M:N cu

scopul de a reduce operațiile de compunere, din folosirea tabelelor de referință, din dublarea atributelor chei străine în relații de tip 1:M tot cu scopul de a reduce operațiile de compunere, din introducerea de grupuri repetitive sau din crearea de tabele de extragere. Din nefericire pentru proiectanți, nu există în literatură definite reguli fixe pentru stabilirea situațiilor în care este indicată denormalizarea relațiilor. În [Roger 89] şi [Flemi 89] se pot găsi pledoarii pro şi contra situațiilor când denormalizarea este indicată.

Pasul 12. ***Estimarea spațiului de memorare necesar bazei de date.***

Este, de asemenea, important să se realizeze o estimare a spațiului de memorare necesar pentru stocarea întregii baze de date. Aceasta ne va da o imagine de perspectivă asupra bazei de date şi ne va ajuta să înțelegem dacă pot apărea probleme în etapa operațională a bazei de date. Se va realiza o estimare a spațiului de memorare necesar pentru stocarea fiecărui fişier de date, precum şi a structurilor auxiliare de acces, astfel încât să se obțină o estimare a dimensiunii întregii baze de date. Această estimare ne oferă posibilitatea să observăm dacă este posibil să fie probleme cu spațiul de memorare necesar bazei de date. Dacă astfel de probleme se întrevăd, este necesar să se ia decizii de soluționare a lor. Aceste decizii ar putea să implice modificarea lungimii unor atribute ale relațiilor, modificarea structurii unor relații, modificarea structurilor indexate, sau poate conduce la necesitatea unor modificări vis-à-vis de cerințele hardware ale sistemului. Toate informațiile rezultate în urma acestei etape de estimare a necesarului de spațiu pe disc sunt colectate în Tabelul 9.

Tabel 9. Estimarea spațiului de memorare a bazei de date.

Nume Relație	Tip Fişier de Date	Dimensiune Fişier de Date	Nume Index	Tip Index	Dimensiune Index	Dimensiune Totală
Relație 1						
Total/Rel.1	---		---	---		
...Relație n						
Total/Rel.n	---		---	---		
TOTAL	---		---	---		

Unde

- ***Nume Relație*** – este numele relației de bază. Va fi trecută în tabel fiecare relație de bază care va fi implementată în baza de date.

- ***Tip Fişier de Date*** – este tipul ales pentru fişierul de date care va stoca acea relaţie. Acest tip este cel stabilit la nivelul Tabelului 7.

- ***Dimensiune Fişier de Date*** – este dimensiunea estimată a spaţiului de memorare necesar pentru acel fişier de date.

- ***Nume Index*** – este numele structurii indexate care va fi implementată pentru acea relaţie de bază. Pentru fiecare relaţie de bază sunt evidenţiate în tabel toate structurile auxiliare de acces care s-a decis că vor fi implementate în baza de date pentru acea relaţie de bază.

- ***Tip Index*** – este tipul structurii indexate ales la nivelul analizei făcute în pasul 10 şi evidenţiat în Tabelul 8.

- ***Dimensiune Index*** – este dimensiunea estimată a spaţiului de memorare necesar pentru acea structură auxiliară de acces.

- ***Total/Rel.$_i$*** – rândul din tabelă conţine trei valori, şi anume: dimensiunea fişierului de date care implementează Relaţia$_i$, suma dimensiunilor tuturor structurilor auxiliare de acces asociate cu Relaţia$_i$, precum şi suma celor două valori identificate anterior, adică valoarea pentru ***Dimensiune Totală***.

- ***TOTAL*** – rândul conţine, de asemenea, trei valori, şi anume: suma tuturor dimensiunilor fişierelor de date, suma tuturor dimensiunilor structurilor auxiliare de acces, precum şi suma acestor două valori.

La nivelul pasului 6 din cadrul metodologiei propuse s-a realizat o primă estimare a dimensiunii relaţiilor bazei de date. Aceste estimări sunt folosite la nivelul acestei etape pentru a calcula dimensiunile fişierelor de date. Funcţia de calcul a dimensiunii fişierelor de date este specificată formal astfel:

Funcţia: DimensiuneFişierDate

Descriere: Este estimată dimensiunea fişierului de date care păstrează datele relaţiei

Intrări: Numărul estimat al tuplurilor Nt_{R_N} ale relației, mărimea medie Mt_{R_N} a unui tuplu, tipul fişierului de date

Sursa: EstimareNumărTupluriRelație, MărimeTupluRelație, TipFişierDate

Iesiri: Dimensiunea estimată $D_{FişierDate\_N}$ pentru FişierDate_N

Destinația: Schema fizică

Pre-condiții: Este estimat numărul de tupluri Nt_{R_N} al relației şi mărimea medie Mt_{R_N} a unui tuplu şi este stabilit tipul fişierului de date

Post-condiții:

$D_{FişierDate\_N} = Nt_{R\_N} * Mt_{R\_N} + D_{Sistem}$

unde D_{Sistem} este determinat de TipFişierDate şi de SGBD-ul folosit pentru implementarea bazei de date

---

In mod similar şi pentru structurile indexate asociate fişierului de date va fi estimată dimensiunea. Aceasta depinde de numărul de tupluri ale relației, de lungimea cheii de indexare şi de tipul de index ales pentru acea structură. Estimarea necesarului de spațiu pe disc pentru structurile indexate se poate face conform funcției DimensiuneIndex ce este specificată formal mai jos:

Funcția: DimensiuneIndex

Descriere: Este estimată dimensiunea indexului asociat fişierului de date care păstrează datele relației

Intrări: Numărul estimat al tuplurilor Nt_{R_N} ale relației, lungimea medie a cheii de indexare $Lung_{Cheie\_index\_R\_N}$, tipul indexului

Sursa: EstimareNumărTupluriRelație, SelecțieCheiIndex, TipIndex

Iesiri: Dimensiunea estimată $D_{Index\_R\_N}$ pentru indexul construit asupra Relație_N

Destinația: Schema fizică

Pre-condiții: Este estimat numărul de tupluri Nt_{R_N} al relației şi lungimea medie a cheii de indexare $Lung_{Cheie\_index\_R\_N}$ pentru care este construit indexul şi este stabilit tipul de index

Post-condiții:

$D_{Index\_R\_N} = Nt_{R\_N} * Lung_{Cheie\_index\_R\_N} + D_{Sistem}$

unde D_{Sistem} este determinat de TipIndex şi de SGBD-ul folosit pentru implementarea bazei de date

---

Pornind de la dimensiunile estimative calculate pentru fişierele de date şi fişierele index asociate acestora se totalizează necesarul de spaţiu de memorare. In funcţie de valoarea obţinută şi de disponibilităţile existente la nivelul sistemului implementat se vor face sau nu ajustări care să conducă la o reducere a necesarului de spaţiu de memorare.

4.3.3. Proiectarea mecanismelor de securitate

O bază de date reprezintă o sursă comună de informaţii pentru mai mulţi utilizatori. De regulă, nu toţi utilizatorii au aceleaşi necesităţi faţă de baza de date. Prin urmare, drepturile lor de a accesa informaţia din baza de date vor fi diferite. Această etapă a procesului de proiectare fizică a bazei de date este preocupată tocmai de stabilirea mecanismelor de asigurare a securităţii datelor pentru diferitele categorii de utilizatori, mecanisme care ţin de structura fizică a bazei de date [Elbra 92], [Herbe 90].

Pasul 13. ***Proiectarea vizualizărilor.***

O primă modalitate de restricţionare a accesului la informaţiile memorate fizic într-o tabelă este aceea de a defini vizualizări asupra acelei tabele. Folosind modelele de date logice locale, identificate anterior în procesul de proiectare a bazei de date, sunt definite vederile utilizator. Etapa este specificată formal astfel:

Funcţia: ProiectareVizualizare

Descriere: Este proiectată vizualizarea ţinând cont de SGBD ţintă

Intrări: Cerinţe de securitate

Sursa: Schema logică locală

Iesiri: Vizualizare_N

Destinaţia: Schema fizică

Pre-condiţii: Identificare cerinţe de securitate la nivelul etapei de proiectare logică a bazei de date

Post-condiţii:

(Nume_Vizualizare=Nume) and ((Atribut1=Nume_Atribut1), (Atribut2=Nume_Atribut2), ...) and (CondiţieCreare=Bloc_SELECT)

In funcţie de cerinţele de securitate specificate în etapa de proiectare conceptuală şi logică a bazei de date sunt definite toate vederile utilizator necesare. Asupra unei relaţii de bază pot fi definite mai multe astfel de vizualizări, care să restricţioneze în moduri diferite accesul utilizatorilor la datele stocate fizic în tabelele bazei de date. Toate informaţiile despre vizualizări pot fi colectate în tabelul 10 pentru o mai uşoară urmărire.

Tabel 10. Vizualizări.

Nume Vizualizare	Atribute	Relaţie de Bază	Condiţie Creare

Unde

- ***Nume Vizualizare*** – este numele specificat pentru vizualizare. Numele trebuie să fie unic şi diferit de numele relaţiilor de bază.

- ***Atribute*** – este numele specificat pentru fiecare atribut al vizualizării, dacă acesta s-a ales să difere de cel al tabelei de bază care stă la baza vizualizării.

- ***Relaţie de Bază*** – este relaţia de bază care stă la baza construirii vizualizării.

- ***Condiţie Creare*** – este blocul SELECT care stă la baza construirii vizualizării.

Pasul 14. ***Proiectarea regulilor de acces.***

Pasul anterior nu este suficient pentru asigurarea securităţii datelor. El trebuie să fie însoţit de cel de specificare a drepturilor de acces la obiectele bazei de date, fie ele tabele sau vizualizări. Anumitor categorii de utilizatori li se pot acorda drepturi de acces doar la nivelul vizualizărilor, securizând astfel, faţă de aceştia, acele date care există în tabele, dar care nu sunt accesibile vizualizărilor. De asemenea, securitatea datelor poate fi întărită şi prin acordarea unor privilegii de acces limitate.

Funcția: PrivilegiiAccesUtilizator

Descriere: Sunt atribuite privilegiile de acces pentru Utilizator_N asupra tabelelor şi vizualizărilor bazei de date

Intrări: Utilizatorii definiți ai bazei de date, tabelele şi vizualizările existente în baza de date

Sursa: DefinireUtilizatori, ProiectareRelație, ProiectareVizualizare

Iesiri: Lista privilegiilor de acces pentru Utilizator_N

Destinația: Schema fizică

Pre-condiții: Sunt definiți utilizatorii bazei de date şi sunt proiectate relațiile şi vizualizările din baza de date

Post-condiții:

for i=1 to M ; unde M este egal cu numărul de utilizatori definiți

{

for j=1 to N ; unde N este egal cu numărul de tabele din baza de date

Privilegiu_Utilizator_N_i = Privilegii asupra Tabelă_N_j

for j=1 to N ; unde N este egal cu numărul de vizualizări din baza de date

Privilegiu_Utilizator_N_i = Privilegii asupra Vizualizare_N_j

}

Privilegiile de acces ale utilizatorilor asupra obiectelor bazei de date pot fi colectate în tabelul 11 pentru o mai uşoară urmărire la nivelul etapei de implementare.

Tabel 11. Privilegii de acces.

Nume Utilizator	Parolă	Prililegii Acces	Obiect al Bazei de Date

Unde

- ***Nume Utilizator*** – este numele specificat pentru utilizatorul bazei de date.
- ***Parolă*** – este parola de acces a utilizatorului.

- ***Privilegii de Acces*** – sunt privilegiile de acces oferite utilizatorului asupra obiectelor din baza de date.

- ***Obiect al Bazei de Date*** – este indicat numele obiectului din baza de date asupra căruia sunt acordate acele privilegii de acces.

Funcția pe baza căreia se face definirea utilizatorilor bazei de date este următoarea:

Funcția: DefinireUtilizator

Descriere: Este definit un utilizator al bazei de date, ținând cont de caracteristicile SGBD țintă

Intrări: Utilizatorii identificați în etapa proiectării conceptuale a bazei de date

Sursa: Schema conceptuală

Iesiri: Utilizator_N

Destinația: Schema fizică

Pre-condiții: Identificarea utilizatorilor bazei de date la nivelul etapei de proiectare conceptuală a bazei de date

Post-condiții:

(Nume_Utilizator_N = Nume) and (Parolă_Utilizator_N = Parolă)

În acest moment procesul de proiectare al bazei de date se poate considera încheiat. Baza de date poate trece în următoarea etapă din ciclul său de viață, şi anume în etapa de implementare. Folosind deciziile de proiectare adoptate pe parcursul etapei de proiectare fizică a bazei de date, se trece la crearea bazei de date în SGBD-ul ales pentru implementare.

4.4. Concluzii

Scopul acestei metodologii de proiectare fizică a bazei de date este acela de a ajuta proiectanții unei baze de date, de-a lungul diferitelor etape pe care aceştia trebuie să le parcurgă atunci când au de realizat proiectul unei baze de date. Metodologia conține mai mulți paşi, care ajută proiectantul să aleagă soluția cea mai bună de la nivelul fiecărei etape. Cu ajutorul acestei metodologii proiectanții de baze de date pot planifica, gestiona, controla şi evalua proiectul bazei de date mult mai uşor şi mai exact.

Această metodologie de proiectare fizică a bazei de date este o unealtă utilă pentru proiectanții de baze de date relaționale şi relațional-obiectuale. Ea conține mai mulți paşi, care ghidează proiectantul de-a lungul etapei de proiectare fizică a bazei de date.

Aplicarea metodologiei porneşte de la schema logică a bazei de date, după ce s-a ales sistemul de gestiune a bazelor de date care va fi utilizat pentru implementarea bazei de date. După ce relațiile de bază şi restricțiile de integritate sunt specificate folosind SGBD-ul ales, se realizează o analiză foarte atentă a tranzacțiilor care operează asupra bazei de date, cu scopul de a identifica operațiile care sunt cel mai des executate la nivelul bazei de date şi relațiile care sunt implicate în aceste operații. Este propusă o metodă nouă de analiză a tranzacțiilor, care înlocuieşte metoda hărților de utilizare a tranzacțiilor. Aceasta oferă posibilitatea unei reprezentări formale mai facile a activităților care au lor la nivelul fiecărei tranzacții şi a fiecărei relații din baza de date. Metoda propusă permite o mai uşoară urmărire şi interpretare a informației acumulate. Pentru a sprijini proiectantul în luarea celor mai bune decizii cu privire la procesul de proiectare fizică a bazei de date, au fost introduse câteva metrici care permit contorizarea activităților care au loc la nivelul relațiilor şi al tranzacțiilor din baza de date. Determinarea tranzacțiilor şi a relațiilor critice se face pe baza unor punctaje calculate atât pentru tranzacții cât şi pentru relații. Punctajele se calculează ținând cont de frecvențele de execuție ale tranzacțiilor, de orele cu încărcare de vârf şi de restricțiile cu privire la timpii de execuție ai tranzacțiilor. Este contorizat numărul de operații de selecție, inserare, modificare şi ştergere care au loc la nivelul fiecărei tranzacții în parte, pentru a determina care sunt tranzacțiile care efectuează cele mai multe prelucrări asupra bazei de date. Tinând cont şi de frecvențele de execuție ale acestor tranzacții se poate stabili cu exactitate care sunt tranzacțiile din categoria celor 20% care execută 80% din prelucrările bazei de date. De asemenea, este contorizat numărul operațiilor de selecție, inserare, modificare şi ştergere care au loc la nivelul fiecărei relații în parte, precum şi indicii de activitate şi tip activitate care folosesc la identificarea relațiilor critice din punctul de vedere al performanțelor ce se pot obține la prelucrarea lor. In funcție de valorile obținute de aceşti indici se pot lua pe baze fundamentate, nu euristice, deciziile cu privire la structura optimă a fişierelor de date şi a celor mai eficiente metode de acces care se pot folosi. Pornind de la aceste relații şi de la tranzacțiile care se execută asupra lor, este analizată şi identificată necesitatea pentru structuri de acces auxiliare. Au fost definite două noi noțiuni, care ajută la identificarea soluțiilor optime. Acestea sunt *atributele verzi*, adică atributele unei relații care este indicat să facă parte din cheile de indexare ale acelei relații, şi *atributele roşii*, care sunt acele atribute ale unei relații care nu este indicat să aparțină cheilor de indexare ale relației. Au fost definite, de

asemenea, reguli pe baza cărora să fie făcută încadrarea atributelor relațiilor fie în categoria atributelor verzi fie în cea a atributelor roşii. Prin urmare, se obține pentru fiecare relație setul cheilor de indexare posibile din care vor fi selectate cele ce se vor implementa în baza de date. Metodologia permite determinarea pe baze fundamentate matematic a relațiilor ce trebuie partiționate şi a cheilor optime de partiționare a acestora. Pornind de la punctajele calculate pentru relațiile din baza de date şi pentru tranzacțiile care operează asupra lor, precum şi de la celelalte decizii de proiectare adoptate, sunt alese tipurile optime ale structurilor de date atât pentru fişierele de date cât şi pentru structurile indexate.

Metodologia permite identificarea relațiilor care pot avea probleme în satisfacerea cerințelor de performanță şi, ca atare, pot să necesite introducerea unei redundanțe controlate la nivelul bazei de date.

Este efectuată o estimare a mărimii totale a bazei de date rezultate. Dacă este necesar, sunt realizate ajustările necesare asupra bazei de date. La final, este obținută structura de memorare a bazei de date şi, de asemenea, structurile adiționale de acces care vor îmbunătăți performanța bazei de date.

O ultimă etapă a metodologiei de proiectare fizică a bazei de date permite proiectarea mecanismelor de securitate care țin de structura fizică a bazei de date. Sunt definite vederile utilizatorilor asupra bazei de date şi sunt proiectate regulile de acces ale utilizatorilor atât la nivelul tabelelor cât şi a vizualizărilor din baza de date.

La final, există toate datele necesare pentru trecerea la următoarea etapă a ciclului de viață a bazei de date, şi anume aceea de implementare.

Este important pentru succesul viitoarelor aplicații, care vor rula pe baza de date, ca etapa de proiectare fizică a bazei de date să fie foarte bine realizată. O structură de memorare rău aleasă pentru fişierele de date şi pentru structurile auxiliare de acces poate deteriora foarte tare performanța bazei de date.

Această metodologie ajută proiectanții bazei de date să ia deciziile de proiectare a bazei de date pe baze fundamentate, corecte şi coerente.

A Aplicație pentru asistarea procesului de proiectare fizică a bazei de date axată pe metodologia propusă

5.1. Prezentarea aplicației de asistare a procesului de proiectare fizică a bazei de date

Producerea unor sisteme care satisfac cerințele de performanță şi de calitate ale utilizatorilor lor este un deziderat atât economic cât şi o provocare pentru ştiință şi tehnologie. Baza de date este de cele mai multe ori o componentă importantă a acestor sisteme, prin urmare aceste cerințe se răsfrâng şi asupra ei.

Performanța în exploatare a unei baze de date este vitală pentru sistemul informațional care o utilizează. Gradul de satisfacție asigurat utilizatorilor sistemului este decisiv în acceptarea sau respingerea acestuia. Dacă prelucrarea datelor este greoaie şi dacă timpii de răspuns sunt prea mari, succesul general al aplicației poate fi puternic periclitat. Pentru a avea o bază de date corect şi optim structurată este necesară parcurgerea unui proces de proiectare a acesteia care să elimine toate aceste problemele.

Teza propune o metodologie de proiectare fizică a bazelor de date care urmăreşte eliminarea erorilor de proiectare din cadrul acestui proces prin folosirea unui formalism matematic care permite validarea deciziilor de proiectare pe baze matematice. În acest mod

este posibil să se realizeze proiectarea fizică a bazei de date într-un *„mediu curat"*, prin eliminarea erorilor umane de proiectare. Introducerea acestui formalism matematic prezintă şi avantajul că permite dezvoltarea unor utilitare de asistență a procesului de proiectare. Acestea uşurează procesul de proiectare şi scurtează considerabil timpii şi costurile aferente etapei de proiectare. Metodologia de proiectare propusă, folosind specificațiile formale şi verificările şi validările statice prin care se elimină erorile de proiectare, prezintă şi avantajul că permite scurtarea sau chiar eliminarea etapei de testare, în vederea detectării erorilor de la nivelul structurii fizice a bazei de date, din ciclul de viață al bazei de date. Se realizează astfel o nouă reducere a timpilor şi costurilor necesare dezvoltării bazei de date.

În scopul de a susține beneficiile acestei metodologii de proiectare fizică a bazelor de date a fost dezvoltat un utilitar de proiectare care urmăreşte metodologia propusă.

Pornind de la schema logică a a bazei de date şi de la informațiile privind frecvența de execuție a tranzacțiilor, orele de încărcare de vârf din sistem şi restricțiile cu privire la timpii de răspuns ai tranzacțiilor, se realizează asistat proiectarea fizică a bazei de date. La final va rezulta schema fizică a bazei de date. Pentru fiecare relație existentă la nivelul schemei conceptuale şi logice a bazei de date se obține structura fizică optimă a tabelei care va implementa acea relație. La final, vor rezulta structurile fizice de memorare optime pentru tabelele bazei de date, precum şi structurile auxiliare de acces necesare pentru a creşte performanța bazei de date.

Aplicația a fost scrisă în Delphi 7, iar baza de date în care sunt colectate datele necesare procesului de proiectare fizică este implementată în SQL Server 2000. În anexa 2 se pot găsi tabelele ce alcătuiesc baza de date de gen *repository*, precum şi programele sursă ale aplicației.

Parcurgerea procesului de proiectare fizică a bazei de date se realizează în mai multe etape, şi ca atare, şi aplicația a fost dezvoltată în mod similar.

Prima etapă este cea în care se preiu datele necesare procesului de proiectare fizică a bazei de date de la nivelul schemei conceptuale şi a schemei logice a bazei de date. Sunt preluate date despre relații (PreluareRelatii), şi anume: numele relației, atributele acesteia, cheia primară, durata de viață estimată a relației, şi despre tranzacțiile care operează asupra bazei de date (Preluare Tranzactii), şi anume: numele tranzacției, descrierea acesteia, frecvența de execuție, orele cu încărcare de vârf ale tranzacției, restricția cu privire la timpul de răspuns al tranzacției, precum şi ponderile asociate acestor caracteristici ale tranzacției care relevă importanța acordată acestora. Datele despre relațiile şi tranzacțiile bazei de date pot fi actualizate prin inserare, modificare sau ştergere atâta timp cât procesul de proiectare a

reprezentării fizice a bazei de date nu a început. Dacă acest proces este început, actualizarea datelor despre relațiile sau tranzacțiile bazei de date implică reluarea procesului de proiectare a reprezentării fizice a bazei de date. Datele despre relațiile şi tranzacțiile din baza de date pot fi, de asemenea, în permanență consultate, fie individual fie în grup. După ce aceste date inițiale, necesare procesului de proiectare fizică a bazei de date, au fost furnizate se poate trece la următoarea etapă a procesului de proiectare.

În figura 1 şi 2 sunt prezentate capturile de ecran ale formelor care permit preluarea datelor relațiilor şi a tranzacțiilor. Codul sursă se găseşte în anexa 2.

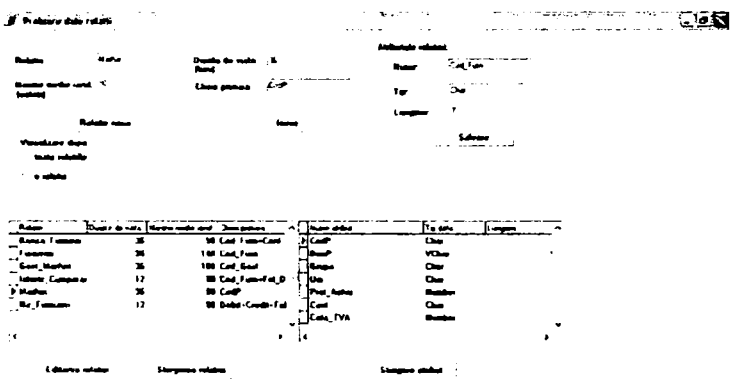


Figura 1. Forma pentru preluarea datelor inițiale ale relațiilor

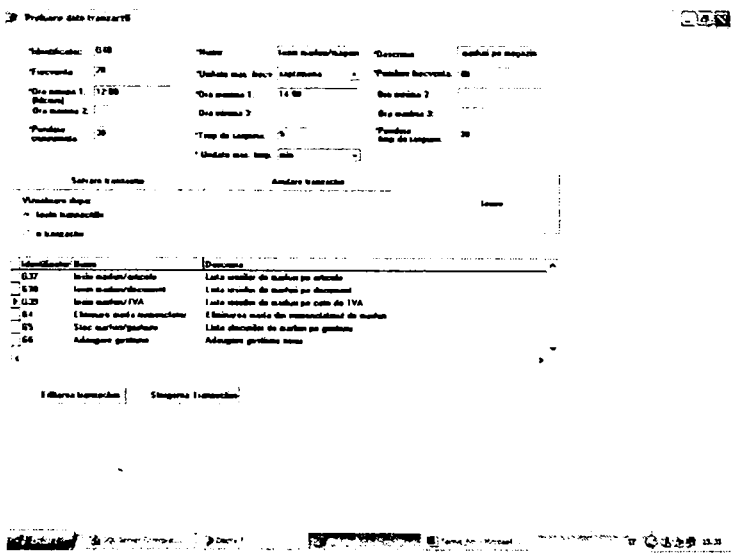


Figura 2. Forma pentru preluarea datelor inițiale ale tranzacțiilor

Cea de a doua etapă a procesului de proiectare fizică a bazei de date presupune proiectarea reprezentării fizice a bazei de date şi începe prin analizarea tranzacțiilor care operează asupra bazei de date. Pentru început sunt identificate tranzacțiile critice din sistem

(Analiza1), adică acele tranzacții care efectuează cel mai mare număr de prelucrări la nivelul sistemului şi care, astfel, influențează într-o măsură mai mare măsură performanța în exploatare a bazei de date. Alegerea tranzacțiilor critice se face pe baza punctajului calculat pentru fiecare tranzacție în parte, punctaj care ține cont de frecvența de execuție a tranzacției, de orele de încărcare de vârf din sistem, de restricția de timp de răspuns impusă tranzacției şi de operațiile care se execută la nivelul tranzacției. Modul de calcul al punctajului tranzacțiilor şi modalitatea de selecție a tranzacțiilor critice este definit în metodologia de proiectare propusă în capitolul 4 la nivelul pasului 4 – *Identificarea tranzacțiilor critice*. Figura 3 prezintă captura de ecran a formei folosită pentru identificarea tranzacțiilor critice. Codul sursă se găseşte în anexa 2.

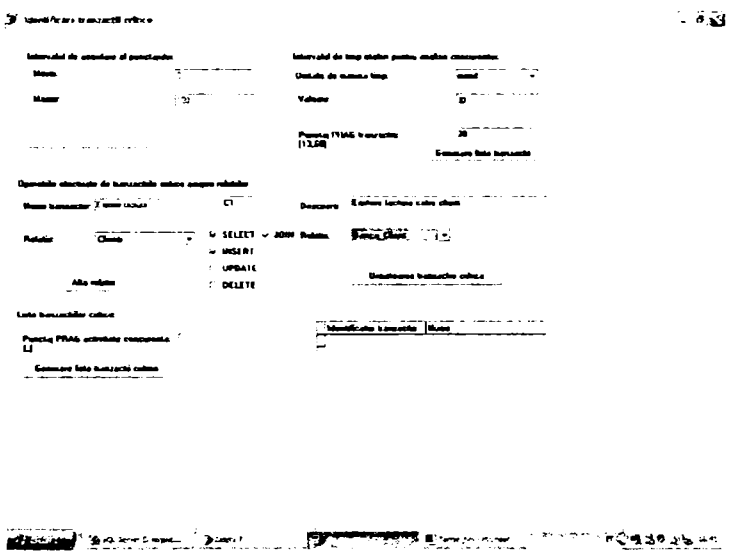


Figura 3. Forma pentru identificarea tranzacțiilor critice

Tranzacțiile critice ajută la determinarea relațiilor critice (IdRelCrit), adică a acelor relații asupra cărora numărul de prelucrări este mai mare şi, prin urmare, este necesar să se acorde o atenție deosebită acestor relații atunci când este proiectată structura lor fizică.

Selecția relațiilor critice se face tot pe baza unui punctaj calculat pentru fiecare relație în parte, punctaj care ține cont de numărul operațiilor executate asupra relației de către tranzacțiile critice din sistem şi de frecvențele de execuție ale acestor tranzacții. Modul de calcul al punctajului relațiilor şi cel de selecție a relațiilor critice este dat de pasul 5 – *Identificarea relațiilor critice* din cadrul metodologiei propuse în capitolul anterior. Figura 4 prezintă captura de ecran a formei folosită pentru identificarea relațiilor critice. Codul sursă aferent acestei etape se găseşte în anexa 2.

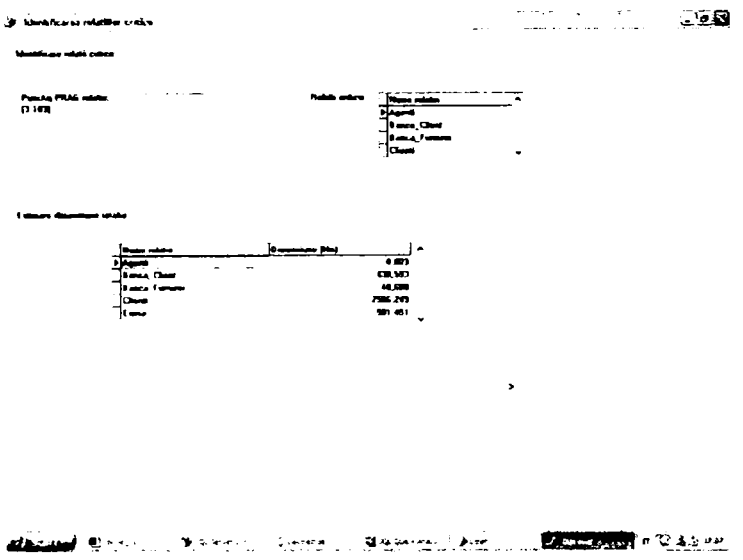


Figura 4. Forma pentru identificarea relațiilor critice

Estimarea dimensiunii relațiilor este importantă (IdRelCrit), pentru că ea poate stabili cât de mari vor fi aceste relații în timp, şi în funcție de dimensiunile rezulate se pot lua decizii cu privire la structura fizică cea mai indicată pentru acea relație. Dimensiunile estimate ale relațiilor sunt calculate automat pornind de la durata de viață estimată a relației, de la frecvența de execuție a tranzacțiilor care efectuează prelucrări asupra relației şi de la tipul operațiilor efectuate. Această estimare urmăreşte pasul 6 – *Estimarea dimensiunii relațiilor* din cadrul metodologiei prezentată în capitolul 4.

Pornind de la dimensiunile estimate ale relațiilor se pot selecta relațiile de dimensiune mare a căror performanță poate fi îmbunătățită prin folosirea partiționării (IdRelPartit).

Relațiile mari la nivelul cărora activitatea concurentă este considerabilă sunt bune candidate pentru partiționare. Selecția relațiilor partiționabile se face în funcție de punctajul de concurență calculat al relației, punctaj care ține cont de tranzacțiile care se execută concurent asupra relației şi de operațiile executate de aceste tranzacții. Pentru fiecare relație partiționabilă este identificată cheia optimă de partiționare. Modul de calcul al punctajului de concurență al relației, precum şi mecanismul de selecție a cheilor de partiționare sunt conforme cu pasul 7 – *Identificarea relațiilor ce pot fi partiționate* din cadrul metodologiei propuse în capitolul anterior. În figura 5 sunt prezentate capturi de ecran ale formei ce foloseşte la identificarea relațiilor partiționabile ale bazei de date şi a cheilor lor de partiționare. În anexa 2 se găseşte codul sursă aferent acestei etape.

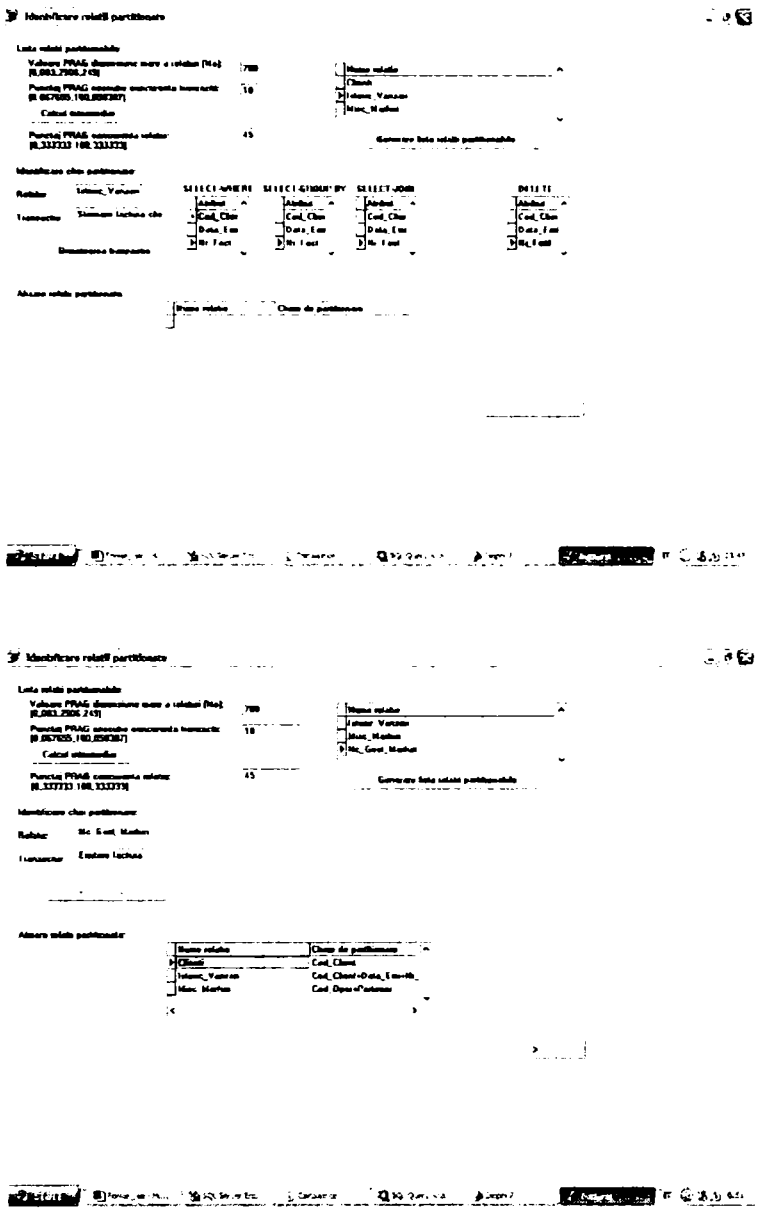


Figura 5. Forma pentru identificarea relațiilor partiționabile

În continuare sunt determinate structurile auxiliare de acces necesare la nivelul bazei de date pentru a creşte performanța acesteia (IdStructAuxAcces). Sunt calculați indicii de activitate şi indicii tipului de activitate, definiți în cadrul metodologiei, pentru a putea detecta relațiile care se pretează indexării. Pentru aceste relații, pe baza atributelor verzi şi a atributelor roşii, alte două noi noțiuni introduse de metodologia propusă, sunt determinate cheile posibile de indexare. Cheile de indexare au asociate contoare care ajută la identificarea cheilor cel mai des utilizate. Acestea sunt cheile care vor fi folosite pentru crearea structurilor de indexare asociate tabelelor. Determinarea structurilor auxiliare de acces şi a cheilor de indexare posibile pentru acestea se realizează respectând pasul 8 *–Identificarea structurilor auxiliare de acces* din metodologia de proeictare fizică a bazelor de date propusă în capitolul

4 al tezei. Figura 6 prezintă captura de ecran a formei la nivelul căreia sunt identificate structurile auxiliare de acces. Codul sursă aferent se găseşte în anexa 2.

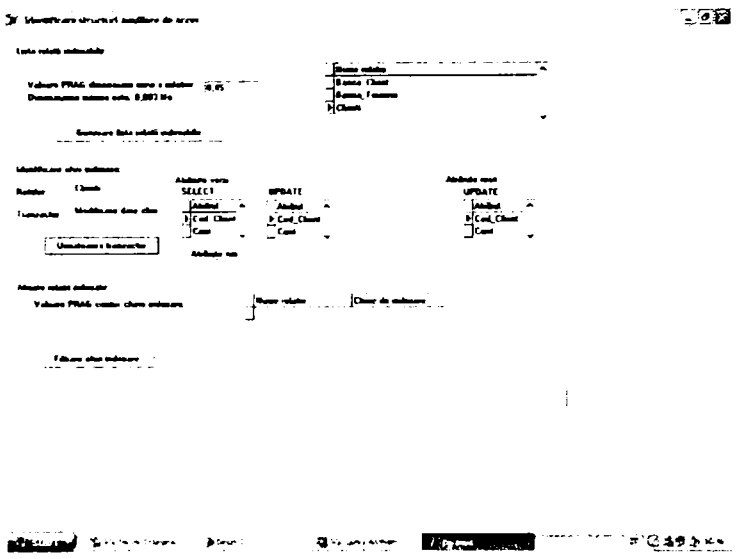


Figura 6. Forma pentru identificarea structurilor auxiliare de acces

Următoarea etapă a procesului de proiectare fizică a bazei de date urmăreşte alegerea organizării fişierelor de date (AlegOrgFisiere). Pornind de la valorile determinate anterior pentru dimensiunile estimate ale relațiilor, de la informațiile cu privire la relațiile partiționabile şi la cheile de partiționare ale acestora, de la datele cu privire la relațiile indexabile şi la cheile de indexare ale acestora, precum şi de la operațiile de compunere necesare între relațiile din sistem, programul de asistare a procesului de proiectare fizică a bazei de date stabileşte tipul de structură de date care trebuie folosită pentru implementarea fiecărei relații în parte. Aceste tipuri de structuri de date sunt alese în funcție de prelucrările executate de tranzacțiile din sistem asupra relațiilor din baza de date, astfel încât performanța sistemului să fie maximă. Algoritmul implementat respectă pasul 9 – *Alegerea organizării fişierelor de date* din cadrul metodologiei din capitolul 4. Figura 7 cuprinde captura de ecran a formei ce are ca rol prezentarea rezultatelor etapei de alegere a organizării fizice a fişierelor de date. Codul sursă se poate consulta în anexa 2.

Procesul de proiectare este continuat cu stabilirea structurii fizice a indexurilor ce vor fi implementate asupra tabelelor din baza de date (AlStructIndex). În funcție de tipul fişierului de date asupra căruia se construieşte indexul, de activitatea concurentă de la nivelul relației, de numărul estimat de valori distincte ale cheii de indexare, precum şi de operațiile de compunere necesare între relațiile din sistem, se stabileşte tipul fiecărei structuri indexate ce poate fi implementată în baza de date pentru a creşte performanța acesteia. Utilitarul foloseşte un algoritm implementat în conformitate cu specificațiile din pasul 10 – *Alegerea oranizării*

structurilor indexate din cadrul metodologiei de proiectare fizică a bazei de date propusă în capitolul anterior.

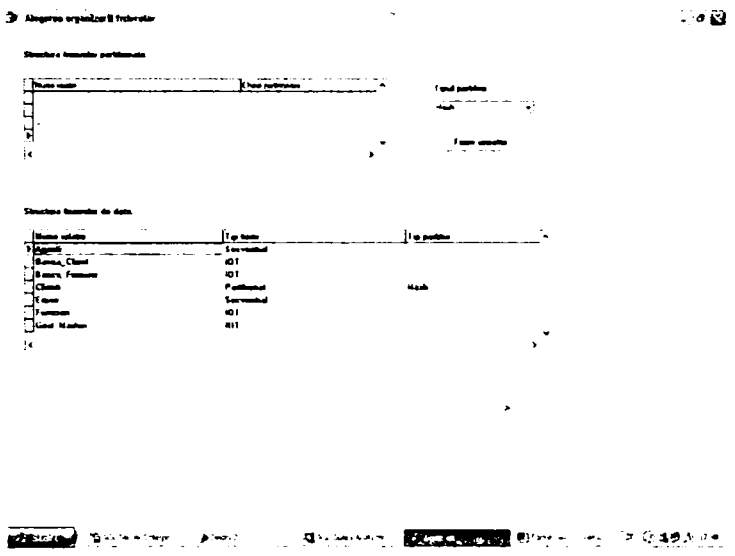


Figura 7. Forma pentru alegerea organizării fizice a fişierelor de date

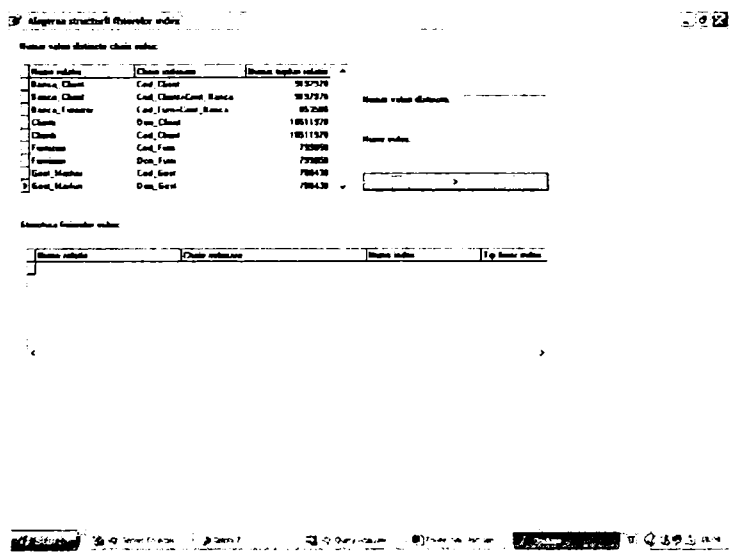


Figura 8. Forma pentru stabilirea structurii fizice a structurilor indexate

Forma din figura 8 prezintă rezultatele acestei etape de alegere a structurii indecşilor asociaţi fişierelor de date cu scopul de a micşora timpii de răspuns la prelucrările care se execută asupra bazei de date. Codul sursă aferent se găseşte în anexa 2.

Programul de asistare a procesului de proiectare fizică a bazei de date permite şi identificarea automată a relaţiilor care pot avea probleme în satisfacerea cerinţelor de performanţă (AnalIntrRedundControl). Acestea sunt relaţiile asupra cărora se poate analiza avantajul introducerii unei redundanţe controlate. Identificarea acestor relaţii se face în

conformitate cu specificațiile pasului 11 – *Analiza introducerii unei redundanțe controlate* din cadrul metodologiei propuse în capitolul 4.

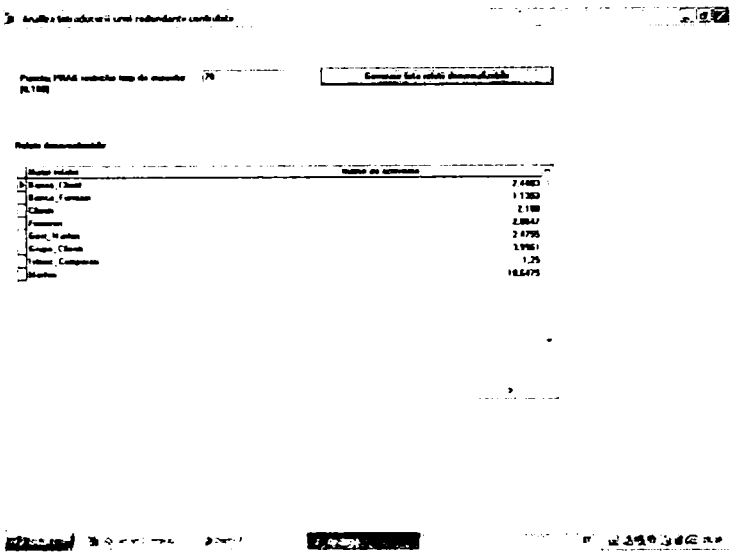


Figura 9. Forma pentru identificarea relațiilor denormalizabile

Se poate încerca o denormalizare a acestor relații, dar numai după ce s-au pus în balanță atât avantajele cât şi toate dezavantajele pe care denormalizarea le generează. Figura 9 prezintă captura de ecran a formei în care sunt afişate relațiile care pot avea probleme în satisfacerea cerințelor de performanță care le-au fost impuse. În anexa 2 se găseşte codul sursă aferent acestei etape.

La final, este realizată o estimare a spațiului de memorare necesar întregii baze de date pentru a se observa dacă nu cumva pot să apară probleme la nivelul etapei operaționale a bazei de date (EstSpatMem).

Estimarea este realizată în mod automat pornind de la datele privind durata de viață estimată a relațiilor, de la operațiile care se execută asupra acestor relații şi de la frecvențele lor de execuție, precum şi ținând cont de tipurile de structuri de date care au fost alese pentru fişierele de date şi pentru structurile lor auxiliare de acces. Mărimea totală rezultată pentru baza de date poate să determine o reanalizare a relațiilor şi a deciziilor lor de proiectare. Estimarea este realizată folosind datele din pasul 6 – *Estimarea dimensiunii relațiilor* şi pe cele din pasul 12 – *Estimarea spațiului de memorare necesar bazei de date* din cadrul metodologiei propuse în capitolul 4 al tezei. În figura 10 este prezentată captura de ecran a formei din aplicație care afişează aceste date. Codul sursă aferent este prezentat în anexa 2.

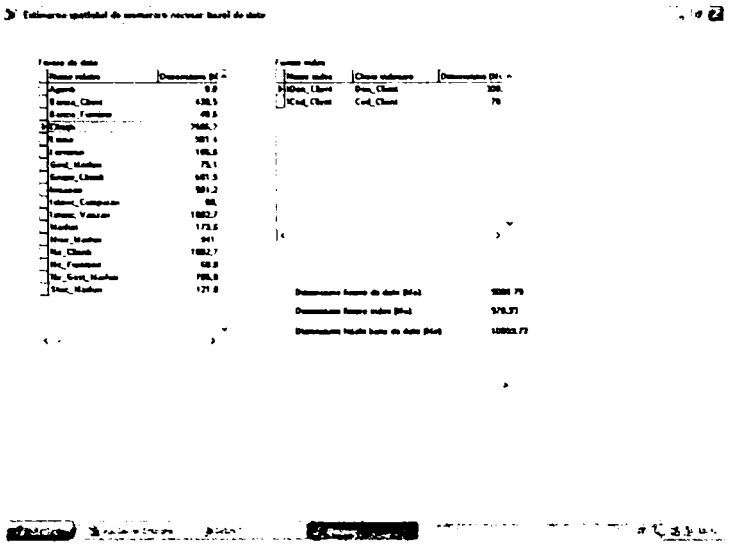


Figura 10. Forma care conține estimarea dimensiunii bazei de date

Datele finale ale procesului de proiectare a reprezentării fizice a bazei de date pot fi consultate în forma *Schema fizică a bazei de date*, care poate fi observată în figura 11. Codul sursă al acestei forme este prezentat în anexa 2 (SchFizBD).

Programul permite reluarea şi rafinarea procesului de proiectare fizică a bazei de date dacă proiectantul bazei de date decide astfel. În cazul în care se iau decizii de modificare a datelor inițiale ale relațiilor şi/sau tranzacțiilor, care operează în baza de date, procesul de proiectare trebuie refăcut.

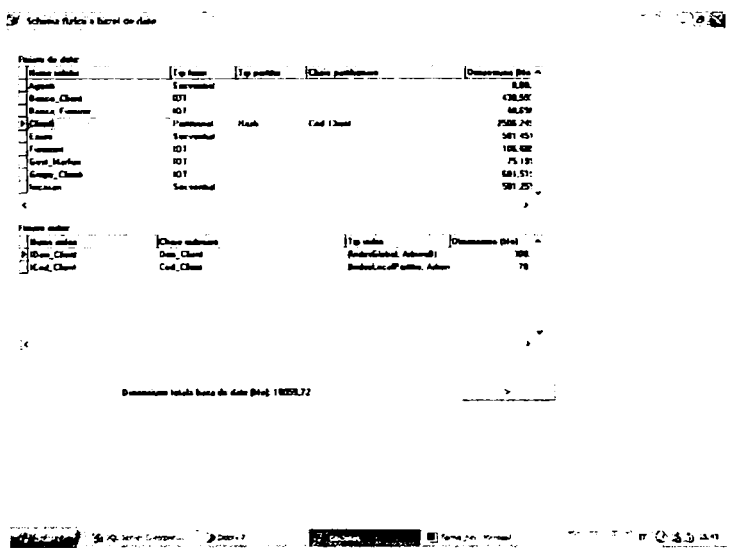


Figura 11. Forma finală care conține datele schemei fizice a bazei de date

Programul permite asistarea procesului de proiectare fizică a unei baze de date ce foloseşte modelul relațional al datelor. El urmăreşte metodologia de proiectare propusă în cadrul tezei. La elaborarea metodologiei au fost luate în calcul tipurile de structuri de date şi

metodele de acces la aceste structuri existente la nivelul celor mai evoluate sisteme de gestiune a bazelor de date.

5.2. Beneficiile utilizării metodologiei de proiectare propuse

Metodologia de proiectare fizică a unei baze de date, bazată pe modelul relațional al datelor, propusă în cadrul tezei ajută la realizarea la parametri superiori a procesului de proiectare fizică a bazei de date. Modelul de dezvoltare a procesului de proiectare fizică a bazei de date este inspirat de modelul „Cleanroom" şi foloseşte o metodă formală de specificare. Paşii propuşi în cadrul metodologiei sunt validați pe baza unor argumente matematice, care permit astfel eliminarea erorilor din procesul de proiectare. Rezultă, astfel, un produs final liber de defecte şi se scurtează şi timpii necesari parcurgerii întregului proces de dezvoltare a sistemului, deoarece etapa de testare a bazei de date este considerabil redusă ca amploare. Se obțin, în aceste condiții, beneficii în ceea ce priveşte costul final al sistemului.

Metodologia de proiectare fizică a bazei de date, propusă în cadrul tezei, a fost aplicată la realizarea etapei de proiectare fizică a unei baze de date necesară unei aplicații complexe de gestiune şi de analiză a vânzărilor, care realizează gestiunea stocului de mărfuri, a aprovizionărilor de la furnizori, a vânzărilor către clienți, a evidenței contabile a firmei, precum şi diverse analize ale vânzărilor efectuate pe o perioadă de 5 ani, pentru o firmă având ca obiect de activitate vânzarea en-gros şi en-detail de produse.

Procesul de dezvoltare al sistemului informațional a parcurs mai multe etape. După etapa de planificare a urmat cea de colectare, analiză şi specificare a cerințelor, la nivelul căreia s-au adunat şi analizat toate informațiile necesare formulării cerințelor viitoarei aplicații. Aceste cerințe au fost clasificate în cerințe funcționale şi cerințe non-funcționale şi au fost specificate în documentul numit specificația cerințelor.

O listă parțială a cerințelor identificate vis-à-vis de activitatea de gestiune a stocurilor de mărfuri şi de cea de mişcare a acestora, precum şi pentru gestionarea relației cu furnizorii şi clienții firmei este prezentată în anexa 2.

Pornind de la cerințele specificate ale aplicației şi ale bazei de date, după ce s-au parcurs etapele de proiectare conceptuală şi logică a bazei de date, a fost parcursă etapa de proiectare fizică a bazei de date, în acord cu sistemul de gestiune a bazelor de date ales pentru implementare. Această etapă a fost abordată atât clasic cât şi folosind metodologia propusă

anterior. Sistemul de gestiune a bazelor de date ales pentru implementarea bazei de date a fost Oracle 9*i*.

Porțiunea din diagrama Entitate-Relație corespunzătoare acestei secțiuni din aplicație este prezentată în figura 12.

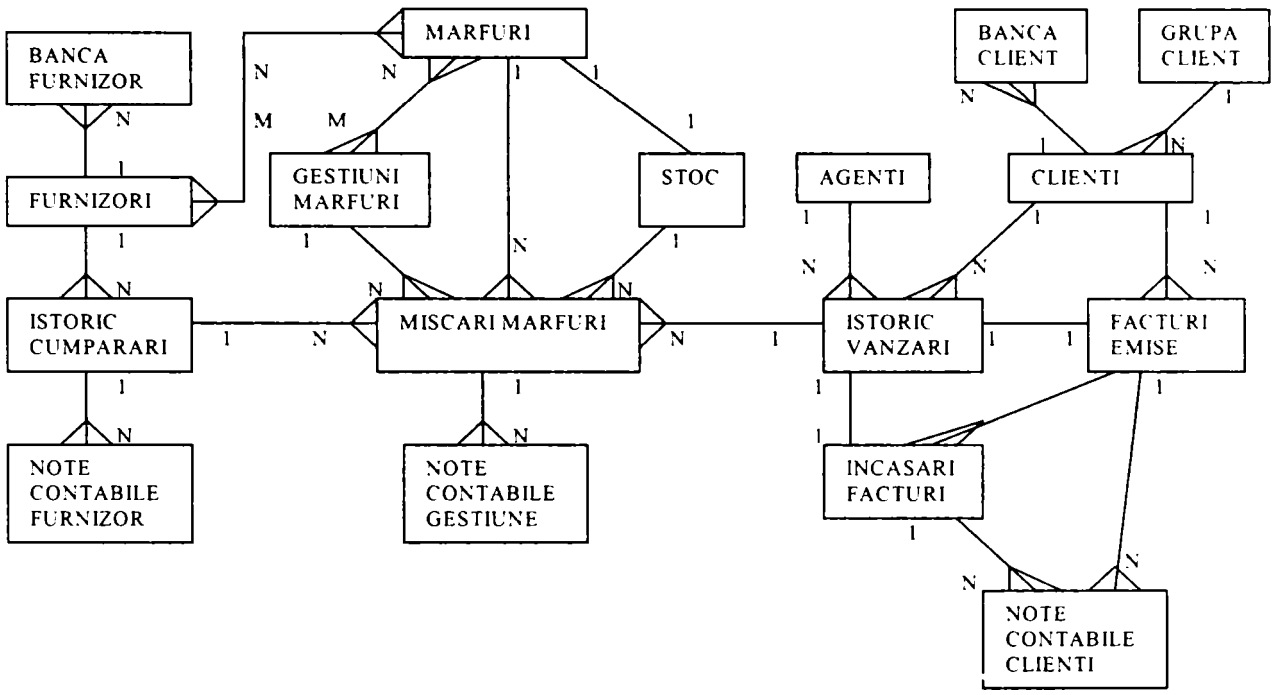


Figura 12. Diagrama Entitate-Relație.

Ea conține entitățile identificate în procesul de proiectare conceptuală a bazei de date, precum şi relațiile existente între aceste entități. Tipul fiecărei relații este, de asemenea, precizat pe diagramă.

Toate tranzacțiile identificate în etapa de specificare a cerințelor este necesar să fie analizate pentru a se stabili care dintre acestea sunt mai des utilizate, care sunt relațiile implicate la nivelul fiecăreia dintre aceste tranzacții şi care sunt operațiile care se execută asupra bazei de date în fiecare tranzacție.

Acest pas necesită, într-o primă etapă, estimarea frecvențelor de execuție a tranzacțiilor, a orelor sau intervalelor de timp de încărcare maximă pentru fiecare tranzacție în parte, precum şi a restricțiilor de timp de răspuns impuse tranzacțiilor. În anexa 2 sunt prezentate datele inițiale ale tranzacțiilor implicate de acest segment al sistemului informațional. Tot în anexa 2 sunt prezentate şi datele relațiilor din baza de date obținute în urma etapelor de proiectare conceptuală şi logică.

Alte metodologii de proiectare fizică a bazelor de date folosesc pentru analiza tranzacțiilor, care se execută asupra bazei de date, *hărțile de utilizare a tranzacțiilor*. Aceste

hărți au ca punct de plecare diagrama Entitate-Relație construită în etapa de proiectare conceptală a bazei de date şi arată care sunt relațiile accesate de fiecare tranzacție în parte indicând schematic care dintre acestea va fi, potențial, utilizată masiv. In cazul unui număr mare de tranzacții care se execută la nivelul bazei de date şi care efectuează un număr mare şi variat de operații asupra relațiilor din baza de date, hărțile de utilizare a tranzacțiilor pot deveni foarte încărcate şi, astfel, foarte greu de urmărit şi de înțeles. Prin urmare, extragerea informațiilor utile în continuate în procesul de proiectare se dovedeşte a nu fi tocmai facilă.

De exemplu, pentru tranzacțiile F1, F2, F3, F6, F7 şi F10 din anexa 2, harta de utilizare a tranzacțiilor este următoarea:

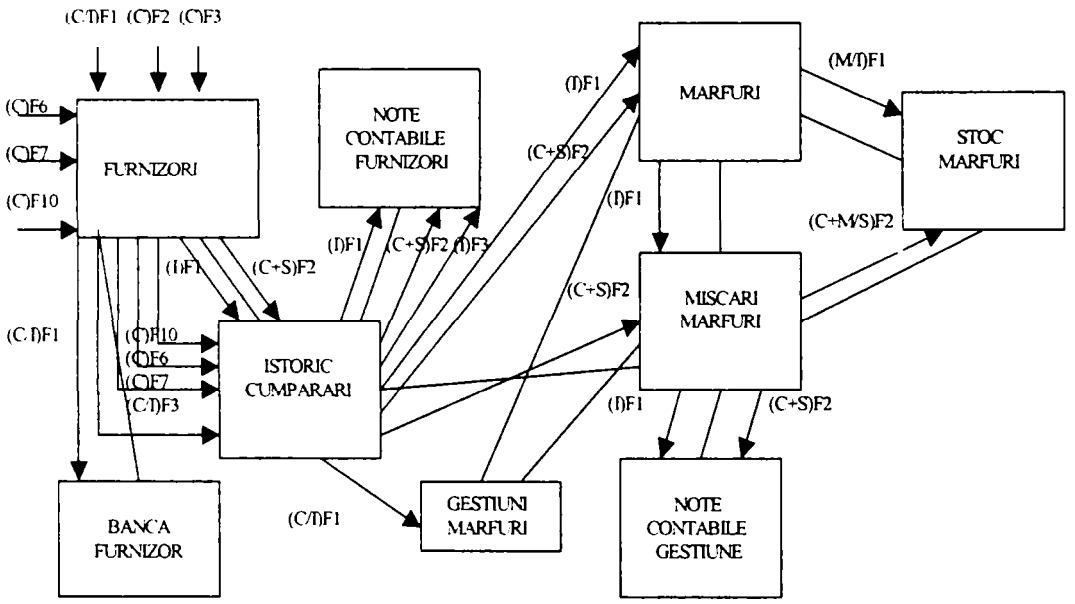


Figura 13. Harta de utilizare a tranzacțiilor pentru tranzacțiile F1, F2, F3, F6, F7, F10.

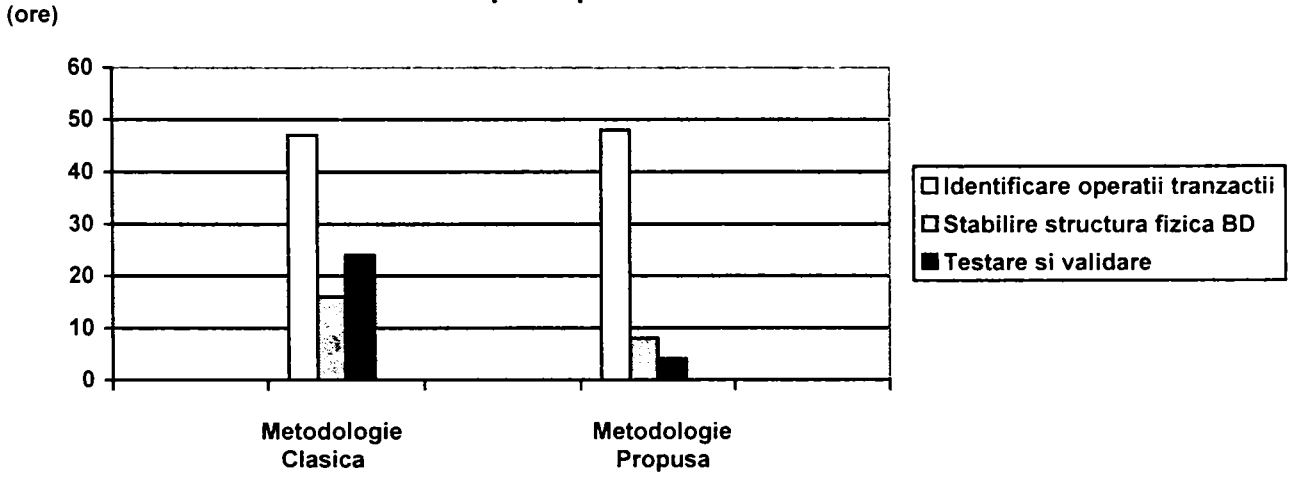
După cum se poate observa şi în figură, analiza tranzacțiilor cu ajutorul hărților de utilizare a tranzacțiilor este greoaie, deoarece diagrama E-R devine foarte încărcată după ce toate tranzacțiile şi operațiile de la nivelul acestora sunt reprezentate pe diagramă. Se poate efectua o rupere pe module a diagramei E-R, astfel încât reprezentarea tranzacțiilor şi a operațiilor de la nivelul acestora să se facă separat pentru fiecare modul în parte. In acest caz, însă, trebuie avută grijă să se însumeze valorile locale atunci când se face analiza globală a activității de la nivelul tranzacțiilor. Pentru fiecare entitate se observă câte arce vin şi pleacă din ea. Acestea dau măsura utilizării entității în cadrul tranzacțiilor care operează asupra bazei de date. Entitățile cu cele mai multe arce sunt cele mai utilizate şi cele la care accesul trebuie optimizat. Deoarece diagrama globală E-R devine deosebit de încărcată, după reprezentarea tuturor arcelor corespunzătoare operațiilor care se produc la nivelul tranzacțiilor din sistem, şi analiza este foarte greoaie, am căutat să găsesc o altă modalitate de a realiza analiza tranzacțiilor. Varianta propusă foloseşte un formalism matematic care permite validarea

deciziilor de proiectare. Procesul de analiză a tranzacțiilor şi de proiectare a reprezentării fizice a bazei de date poate fi astfel complet automatizat.

Pentru realizarea etapei de proiectare fizică a bazei de date s-au folosit două echipe, una dintre ele a utilizat metodologia de proiectare propusă în cadrul tezei şi a folosit şi aplicația de asistare a procesului de proiectare, în timp ce cealaltă a folosit doar o metodologie de proiectare similară celei descrise în capitolul 2. Schema fizică a bazei de date rezultată în urma procesului de proiectare, realizat de echipa care a folosit metodologia propusă şi utilitarul de proiectare dezvoltat pentru ea, este prezentată în anexa 2.

Au fost contorizați timpii necesari pentru efectuarea procesului de proiectare fizică a bazei de date şi au fost comparate rezultatele procesului de proiectare. După cum se poate observa şi în graficul 1 timpii totali necesari realizării proiectării fizice a bazei de date se reduc cu aproximativ 35%. Această reducere a timpilor de proiectare atrage după sine o economie în ceea ce priveşte costurile procesului de dezvoltare a sistemului.

Grafic 1 Timpii de proiectare fizica a bazei de date



Şi la nivelul structurii fizice a bazei de date s-au obținut unele diferențe între cele două echipe. Structurile fişierelor de date rezultate nu au fost într-u totul similare. S-a constatat că echipa ce nu a folosit metoda propusă de proiectare a reprezentării fizice a luat aceleaşi decizii de partiționare a tabelelor, cu toate că procesul de identificare a cheilor de partiționare a fost mai greoi. Diferențe majore sau obținut la nivelul tabelelor Banca_Client, Banca_Furnizor, Furnizori, Gest_Marfuri, Grupe_Clienti, Marfuri, Stoc_Marfuri a căror structură a fost aleasă de tip tabelă organizată indexat de către echipa care a folosit metoda propusă în cadrul tezei şi de tip secvențial de către cealaltă echipă. S-a constatat că echipa care a folosit metoda clasică a avut tendința de a crea prea multe structuri auxiliare de acces pentru fiecare tabelă în parte. La nivelul structurilor auxiliare de acces au rezultat mai multe diferențe şi se poate afirma că etapa de stabilire a acestor structuri a fost mult mai lungă şi mai anevoiasă pentru echipa care

a folosit metodologia clasică. De asemenea, pentru acestă echipă a fost necesară o perioadă mai îndelungată pentru validarea şi testarea deciziilor de proiectare luate.

La cele câteva teste la care au fost supuse cele două baze de date rezultate, teste similare celor din capitolul 3 al tezei, s-au obținut timpi mai buni pentru structura de bază de date obținută de echipa ce a folosit pentru proiectare utilitarul de proiectare dezvoltat în conformitate cu metoda de proiectare propusă în capitolul 4 al tezei. Procesul de testare a metodologiei propuse trebuie continuat şi pe alte sisteme pentru a se putea formula concluzii mai relevante.

Ca o concluzie finală, se poate spune că beneficiile aduse de folosirea metodologiei de proiectare fizică a bazelor de date propusă în cadrul tezei sunt multiple, şi anume:

- se realizează o proiectare pe baze fundamentate nu euristice;
- se uşurează munca echipei de proiectare;
- se scurtează timpii totali de realizare a procesului de proiectare;
- se elimină erorile de proiectare din cadrul procesului de proiectare fizică a bazei de date;
- se poate reveni cu mult mai mare uşurință la nivelul procesului de proiectare;
- se micşorează costurile totale ale procesului de dezvoltare a bazei de date.

C *Concluzii, contribuții şi perspective*

6.1. Concluzii

Teza de față a urmărit aducerea unor contribuții în domeniul prciectării bazelor de date. Deoarece majoritatea sistemelor informatice folosesc astăzi o bază de date, este important pentru succesul general al sistemului ca această bază de date să fie corect proiectată. Performanța sistemului este direct influențată de performanța în exploatare a bazei de date. Caracteristicile sistemului de gestiune a bazelor de date sunt evident importante, dar ele nu sunt în mod exclusiv garantul unor rezultate corespunzătoare în ceea ce priveşte performanța obținută de baza de date în etapa operațională a ciclului său de viață. Un rol deosebit de important în asigurarea unei performanțe sporite la nivelul bazei de date îl joacă structura fizică a acesteia. Două implementări diferite ale aceleiaşi baze de date folosind acelaşi sistem de gestiune a bazelor de date pot fi foarte departe una de alta ca nivel al performanței pe care o asigură în etapa de operare. Tipurile de stucturi de date alese pentru a implementa baza de date, precum şi metodele auxiliare de acces la aceste structuri au o influență hotărâtoare asupra timpilor de execuție din sistem. O alegere neinspirată va cauza performanțe foarte scăzute în exploatare, care inevitabil vor conduce la insatisfacția beneficiarului sistemului informatic şi posibil chiar la respingerea acestuia. Pentru a nu apărea astfel de situații este foarte important ca procesul de proiectare a bazei de date să fie efectuat în mod corect. În cadrul tezei mi-am concentrat atenția asupra etapei de proiectare fizică a

bazelor de date, încercând să aduc contribuții la rezolvarea acestei probleme, astfel încât performanțele obținute de baza de date în etapa operațională a ciclului său de viață să fie superioare.

Mi-am început studiul prin analizarea unora dintre cele mai importante sisteme de gestiune a bazelor de date existente astăzi pe piață şi anume: Oracle 9*i*, SQL Server 2000 şi DB2. Am analizat în principal tipurile de structuri de date puse la dispoziție de aceste sisteme pentru implementarea bazei de date, precum şi metodele auxiliare de acces către aceste structuri. Pentru a identifica plusurile şi minusurile fiecăreia dintre structurile oferite s-au efectuat numeroase teste pe o baza de date test implementată în toate cele trei variante. Testele au fost împărțite pe categorii, fiecare urmărind să studieze un anumit tip de comportament al bazei de date şi să identifice anumite şabloane pe baza cărora să se poată desfăşura procesul de proiectare a bazei de date. Rezultatele obținute la testele efectuate mi-au permis să stabilesc criterii de evaluare a deciziilor de proiectare fizică a bazei de date. Aceste rezultate au fost încorporate într-o metodă proprie de proiectare a reprezentării fizice a bazei de date. A fost propusă o metodologie de proiectare fizică a bazelor de date, care se bazează pe noua metodă de proiectare a reprezentării fizice a bazei de date. Pentru îmbunătățirea calității procesului de proiectare sunt folosite specificațiile formale şi validările deciziilor de proiectare pe baze matematice. Metoda propusă pentru proiectarea fizică a bazelor de date permite dezvoltarea de utilitare de asistare a procesului de proiectare. A fost dezvoltat un astfel de utilitar, pentru a susține beneficiile metodei propuse. El a fost utilizat la proiectarea unei baze de date pentru un sistem informatic folosit în domeniul activităților de vânzare şi analiză a vânzărilor. Structura fizică rezultată s-a dovedit a fi mai performantă decât aceea obținută de o echipă de proiectare care nu a folosit metoda propusă în cadrul tezei.

6.2. Contribuții

În cadrul tezei au fost aduse următoarele contribuții:

- *A fost propusă o nouă metodă de proiectare* a reprezentării fizice a bazelor de date ce folosesc modelul de date relațional sau relațional-obiectual. Metoda îşi propune să realizeze proiectarea bazei de date într-un „mediu curat", adică prin eliminarea erorilor de proiectare din cadrul procesului de proiectare fizică a bazei de date. Acest lucru este realizat prin folosirea unui formalism matematic în specificarea paşilor de proiectare şi prin utilizarea unor validări pe baze matematice a deciziilor de proiectare.

Metoda propusă permite o analiză cantitativă şi calitativă a tranzacțiilor care se desfăşoară în sistem, pornind de la estimările frecvențelor de execuție ale tranzacțiilor, de la orele de încărcare de vârf ale tranzacțiilor, de la restricțiile timpilor de execuție de la nivelul fiecărei tranzacții în parte şi de la operațiile executate asupra bazei de date de către tranzacții. Metoda propune calcularea unor punctaje asociate tranzacțiilor şi relațiilor din sistem, care permit identificarea tranzacțiilor şi relațiilor critice ale sistemului. Sunt folosite validări pe baze matematice ale deciziiilor de proiectare. Sunt identificate relațiile ce pot fi partiționate pentru a îmbunătăți performanța sistemului, precum şi cheile de partiționare optime. Sunt identificate structurile auxiliare de acces menite să îmbunătățească performanța sistemului, precum şi cheile de indexare ale acestora. La final, metoda permite obținerea organizării fişierelor de date şi a structurilor indexate asociate acestora, astfel încât performanțele în exploatare ale bazei de date să fie maxime. Este realizată şi o analiză a oportunității introducerii unei redundanțe controlate la nivelul bazei de date. Metoda propusă permite identificarea relațiilor care pot constitui puncte critice în sistem şi la nivelul cărora se poate pune problema unei denormalizări. La final, metoda permite şi o estimare a spațiului de memorare necesar întregii baze de date.

- *Au fost propuşi indicatori* care au ca scop evaluarea calitativă a operațiilor care au loc la nivelul bazei de date şi care permit validarea deciziilor de proiectare. Aceşti indicatori sunt *indicele de activitate* şi *indicele tipului de activitate*, ce dau măsura calitativă a operațiilor desfăşurate asupra relațiilor bazei de date şi care ajută la luarea deciziilor de proiectare pe baze matematice nu euristice. Au fost definite, de asemenea, două noi noțiuni care ajută la identificarea soluțiilor optime privind structurile auxiliare de acces. Acestea sunt *atribute verzi*, adică atributele unei relații care este indicat să facă parte din cheile de indexare ale relației, şi *atribute roşii*, adică acele atribute ale unei relații care nu este indicat să aparțină cheilor de indexare ale relației. Au fost definite totodată reguli pe baza cărora să fie făcută încadrarea atributelor relațiilor în categoria atributelor verzi sau a atributelor roşii. Atributele verzi şi atributele roşii permit specificarea cheilor de indexare pentru fiecare relație a bazei de date.

- *A fost propusă o metodologie de proiectare fizică a bazelor de date*, care încorporează metoda propusă de proiectare a reprezentării fizice a bazei de date. Metodologia de

proiectare fizică a bazei de date stabileşte paşii care trebuie urmați de proiectantul bazei de date în vederea obținerii schemei fizice a bazei de date pornind de la schema logică a acesteia. Cele trei mari etape ale metodologiei sunt: transformarea modelului de date logic global pentru sistemul SGBD ales, proiectarea reprezentării fizice şi proiectarea mecanismelor de securitate. Cea de a doua etapă a metodologiei a fost adaptată pentru a putea încorpora metoda proprie de proiectare a reprezentării fizice pentru o bază de date ce foloseşte un model de date relațional sau relațional-obiectual. Modelul de dezvoltare a bazei de date este bazat pe filosofia modelului „Cleanroom" de dezvoltare a sistemelor software, care urmăreşte proiectarea într-un mediu curat, astfel încât erorile de proiectare să fie eliminate încă din etapa de proiectare, rezultând astfel un produs final liber de defecte. Costul pe ansamblu al procesului de proiectare este redus, deoarece etapa de testare este de mult mai scurtă durată sau poate fi chiar complet elimitată.

- *S-au introdus specificațiile formale* la nivelul etapelor procesului de proiectare fizică a bazei de date cu scopul de a îmbunătăți calitatea procesului de proiectare. Etapele procesului de proiectare fizică a bazei de date sunt specificate formal folosind un limbaj de descriere structurată a funcțiilor necesare la nivelul fiecărui pas din cadrul metodologiei de proiectare propuse. Descrierea funcțiilor s-a făcut folosind un pseudo-cod pentru ca procesul de proiectare să poată fi automatizat mai facil. Folosirea specificațiilor formale aduce o serie de beneficii, deoarece permite o mai corectă înțelegere a cerințelor sistemului şi a proiectării acestuia, permite dovedirea consistenței şi a caracterului complet al specificațiilor, precum şi a faptului că implementarea este conformă cu acestea şi, de asemenea, permite construirea unor unelte software care să asiste la dezvotarea sistemului. Calitatea procesului de proiectare se îmbunătățeşte, ceea ce conduce la un produs final superior, adică la o bază de date superioară.

- *S-a dezvoltat o aplicație* pentru asistarea procesului de proiectare fizică a bazei de date ce urmăreşte metoda de proiectare a reprezentării fizice a bazei de date propusă în cadrul tezei. Aplicația este dezvoltată în Delphi7 şi foloseşte o bază de date de gen repository implementată în SQL Server 2000. Aplicația a fost dezvoltată astfel încât să urmeze paşii specificați formal ai metodologiei de proiectare fizică a bazelor de date, respectând metoda proprie de proiectare a reprezentării fizice a bazei de date.

Avantajul utilizării unui astfel de utilitar de asistare a procesului de proiectare fizică a bazei de date este evident.

6.3. Perspective

Activitatea desfăşurată de-a lungul elaborării acestei teze, în domeniul proiectării bazelor de date relaționale şi relațional-obiectuale, nu poate fi considerată complet încheiată.

Metoda propusă de proiectare a reprezentării fizice a bazei de date va trebui validată în continuare prin aplicarea ei la proiectarea unor baze de date specifice unor domenii de aplicații variate şi prin analizarea rezultatelor obținute în urma procesului de proiectare. În acest scop colaborarea cu firma S.C. Industial Software S.R.L. va fi continuată şi voi încerca extinderea colaborării şi cu alte firme producătoare de software. Metoda şi rezultatele aplicării ei vor fi prezentate şi la alte evenimente ştiințifice în țară şi în străinătate..

Metoda de proiectare fizică a bazei de date poate fi folosită cu succes şi în scopuri didactie în cadrul facultăților de profil la cursurile de baze de date. Este important ca studenții să-şi însuşească cunoştiințele necesare despre proiectarea bazelor de date şi implicit şi despre etapa de proiectare fizică a bazei de date. Aceasta nu trebuie ignorată, pentru că ea este deosebit de importantă în cazul sistemelor software care prelucrează volume mari de date complexe. Chiar dacă până astăzi mare parte dintre aplicațiile dezvoltate de firmele româneşti producătoare de software nu presupun cerințe deosebite la nivelul bazei de date şi nu prelucrează volume foarte mari de date complexe, lucrurile au început deja să ia o altă turnură şi este important ca viitorii specialişti să fie bine pregătiți.

O altă direcție care va fi urmărită pe viitor va fi aceea de a încerca integrarea utilitarului de proiectare fizică a bazei de date cu alte unelte de proiectare asistată a procesului de proiectare a bazei de date, utilitare care efectuează doar proiectarea conceptuală a bazei de date. Astfel se va obține o unealtă completă de asistare a procesului de proiectare a bazei de date. Aceasta ar putea fi chiar înglobată într-un produs software de tip CASE.

A *Anexe*

ANEXA 1

1.1. *Structura tabelelor bazei de date TEST*

FURNIZORI

COD_FURN	VARCHAR(7)
DEN_FURN	VARCHAR(30)
LOC_FURN	VARCHAR(20)
ADR_FURN	VARCHAR(30)
TAR_FURN	VARCHAR(15)
NR_FISC	NUMERIC(8,0)
CONT	CHAR(11)
SOLD_CR_AN	NUMERIC(14,2)
RUL_DB_AN	NUMERIC(14,2)
RUL_CR_AN	NUMERIC(14,2)
RUL_DB_LUN	NUMERIC(14,2)
RUL_CR_LUN	NUMERIC(14,2)

MARFURI

CODP	VARCHAR(10)
DENP	VARCHAR(50)
GRUPA	VARCHAR(8)
UM	CHAR(3)
PRET_ACHIZ	NUMERIC(20,2)
CONT	CHAR(11)
COTA_TVA	NUMERIC(5,2)
COD_FURN	VARCHAR(7)

MISC_MARFURI

COD	VARCHAR(10)
GEST	CHAR(2)
COD_OPER	CHAR(2)
CANTITATE	NUMERIC(15,3)
DATA_DOC	DATE
FEL_DOC	CHAR(3)
NR_DOC	NUMERIC(8,0)
PARTENER	VARCHAR(7)
FELDOCPAR	CHAR(1)
NRDOCPAR	NUMERIC(8,0)
DATADOCPAR	DATE
PRET_VANZ	NUMERIC(20,2)
COMANDA	NUMERIC(10,0)
COD_PRODUS	VARCHAR(10)
CANT_BRUT	NUMERIC(15,3)
SURSA_PROF	VARCHAR(15)

REZULTATE

NR_OP	NUMBER(3,0)
OPERATIE	VARCHAR(200)
TOTALINREG	NUMBER(15,0)
INREGPREL	NUMBER(12,0)
TIMP	NUMBER(10,0)
VALOARE	VARCHAR(50)
INDEX	CHAR(1)
TIPINDEX	VARCHAR(20)
NRINDEX	NUMERIC(2,0)

REZULTATE_FINALE

NR_OP	NUMBER(3,0)
OPERATIE	VARCHAR(200)
TOTALINREG	NUMBER(15,0)
INREGPREL	NUMBER(12,0)
TIMP	NUMBER(10,0)

VALOARE	VARCHAR(50)	TIPINDEX	VARCHAR(20)
INDEX	CHAR(1)	NRINDEX	NUMERIC(2,0)

1.2. *Rezultatele parțiale ale testelor efectuate*

REZULTATE

NR_OPER	OPERATIE	TOTALINREG	INREGPREL	TIMP	VALOARE	INDEX	TIPINDEX	NRINDEX
1	SELECT * FROM Furnizori WHERE Cod_Furn=	424	1	16	FLARO	N	N	0
1	SELECT * FROM Furnizori WHERE Cod_Furn=	424	1	15	COTNARI	N	N	0
1	SELECT * FROM Furnizori WHERE Cod_Furn=	424	1	16	JACOBS	N	N	0
1	SELECT * FROM Furnizori WHERE Cod_Furn=	424	1	16	WRIGLEY	N	N	0
1	SELECT * FROM Furnizori WHERE Cod_Furn=	424	1	25	FLARO	D	Clasic	1
1	SELECT * FROM Furnizori WHERE Cod_Furn=	424	1	15	FLARO	N	N	0
1	SELECT * FROM Furnizori WHERE Cod_Furn=	424	1	16	FLARO	N	N	0
1	SELECT * FROM Furnizori WHERE Cod_Furn=	424	1	25	FLARO	D	Clasic	1
1	SELECT * FROM Furnizori WHERE Cod_Furn=	848	1	16	FLARO	N	N	0
1	SELECT * FROM Furnizori WHERE Cod_Furn=	848	1	16	COTNARI	N	N	0
1	SELECT * FROM Furnizori WHERE Cod_Furn=	848	1	17	WRIGLEY	N	N	0
1	SELECT * FROM Furnizori WHERE Cod_Furn=	848	1	25	COTNARI	D	Clasic	1
1	SELECT * FROM Furnizori WHERE Cod_Furn=	6360	1	52	FLARO	N	N	0
1	SELECT * FROM Furnizori WHERE Cod_Furn=	6360	1	25	FLARO	D	Clasic	1
1	SELECT * FROM Furnizori WHERE Cod_Furn=	6360	1	54	FLARO	N	N	0
1	SELECT * FROM Furnizori WHERE Cod_Furn=	6360	1	50	FLARO	N	N	0
1	SELECT * FROM Furnizori WHERE Cod_Furn=	6360	1	25	FLARO	D	Clasic	1
2	SELECT * FROM Furnizori WHERE Loc_Furn=	424	12	16	SIBIU	N	N	0
2	SELECT * FROM Furnizori WHERE Loc_Furn=	424	14	16	BUCURESTI	N	N	0
2	SELECT * FROM Furnizori WHERE Loc_Furn=	424	12	15	ALBA	N	N	0
2	SELECT * FROM Furnizori WHERE Loc_Furn=	424	12	31	SIBIU	D	Clasic	1
2	SELECT * FROM Furnizori WHERE Loc_Furn=	848	24	31	SIBIU	D	Clasic	1
2	SELECT * FROM Furnizori WHERE Loc_Furn=	4240	120	38	SIBIU	N	N	0
2	SELECT * FROM Furnizori WHERE Loc_Furn=	4240	120	31	SIBIU	D	Clasic	1
2	SELECT * FROM Furnizori WHERE Loc_Furn=	4664	132	31	SIBIU	D	Clasic	1
2	SELECT * FROM Furnizori WHERE Loc_Furn=	6360	180	39	SIBIU	D	Clasic	1
3	SELECT * FROM Marfuri WHERE DenP=	1263	12	15	SILVANA MENTA 80G	N	N	0
3	SELECT * FROM Marfuri WHERE DenP=	1263	14	15	VINARS MILCOV	N	N	0
3	SELECT * FROM Marfuri WHERE DenP=	1263	12	16	POIANA 100G	N	N	0
3	SELECT * FROM Marfuri WHERE DenP=	1263	12	31	POIANA 100G	D	Clasic	1

3	SELECT * FROM Marfuri WHERE DenP=	4240	40	35	ZAHAR	N	N	0
3	SELECT * FROM Marfuri WHERE DenP=	4240	40	31	ZAHAR	D	Clasic	1
3	SELECT * FROM Marfuri WHERE DenP=	93462	888	110	POIANA 100G	N	N	0
3	SELECT * FROM Marfuri WHERE DenP=	93462	1036	117	VINARS MILCOV	N	N	0
3	SELECT * FROM Marfuri WHERE DenP=	214710	2040	614	POIANA 100G	N	N	0
3	SELECT * FROM Marfuri WHERE DenP=	214710	2040	78	POIANA 100G	D	Clasic	1
3	SELECT * FROM Marfuri WHERE DenP=	4234680	40236	2340	POIANA 100G	N	N	0
3	SELECT * FROM Marfuri WHERE DenP=	4234680	40236	320	POIANA 100G	D	Clasic	1
4	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi	848	24	15	NULL	N	N	0
4	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi	848	24	15	NULL	D	Clasic	1
4	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi	848	24	16	NULL	D	Clasic	2
4	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi	848	24	16	NULL	D	Clasic	3
4	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi	848	24	16	NULL	D	Clasic	4
4	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi	1272	38	16	NULL	N	N	0
4	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi	1272	38	16	NULL	D	Clasic	1
4	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi	1272	38	16	NULL	D	Clasic	2
4	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi	1272	38	16	NULL	D	Clasic	3
4	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi	1272	38	16	NULL	D	Clasic	4
5	INSERT INTO Marfuri SELECT * FROM Marfuri_Noi	395736	11818	31	NULL	N	N	0
5	INSERT INTO Marfuri SELECT * FROM Marfuri_Noi	395736	11818	31	NULL	D	Clasic	1
5	INSERT INTO Marfuri SELECT * FROM Marfuri_Noi	395736	11818	31	NULL	D	Clasic	2
5	INSERT INTO Marfuri SELECT * FROM Marfuri_Noi	395736	11818	32	NULL	D	Clasic	3
5	INSERT INTO Marfuri SELECT * FROM Marfuri_Noi	395736	11818	31	NULL	D	Clasic	4
5	INSERT INTO Marfuri SELECT * FROM Marfuri_Noi	4243680	127040	612	NULL	N	N	0
5	INSERT INTO Marfuri SELECT * FROM Marfuri_Noi	4243680	127040	12934	NULL	D	Clasic	1
5	INSERT INTO Marfuri SELECT * FROM Marfuri_Noi	4243680	127040	26078	NULL	D	Clasic	2
5	INSERT INTO Marfuri SELECT * FROM Marfuri_Noi	4243680	127040	29854	NULL	D	Clasic	3
5	INSERT INTO Marfuri	4243680	127040	34455	NULL	D	Clasic	4

	SELECT * FROM Marfuri_Noi							
5	INSERT INTO Marfuri SELECT * FROM Marfuri_Noi	4243680	127040	45817	NULL	D	Clasic	5
5	INSERT INTO Marfuri SELECT * FROM Marfuri_Noi	4243680	127040	62315	NULL	D	Clasic	6
6	UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Grupa=	395736	11790	31	BOMBOANE	N	N	0
6	UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Grupa=	395736	11790	31	BOMBOANE	D	Clasic	1
6	UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Grupa=	395736	11790	31	BOMBOANE	D	Clasic	2
6	UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Grupa=	395736	11790	32	BOMBOANE	D	Clasic	3
6	UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Grupa=	395736	11790	31	BOMBOANE	D	Clasic	4
6	UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Grupa=	395736	11870	31	RAUCH	N	N	0
6	UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Grupa=	395736	11870	31	RAUCH	D	Clasic	1
6	UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Grupa=	395736	11870	31	RAUCH	D	Clasic	2
6	UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Grupa=	395736	11870	32	RAUCH	D	Clasic	3
6	UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Grupa=	395736	11870	31	RAUCH	D	Clasic	4
6	UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Grupa=	1457184	41305	13736	POIANA	N	N	0
6	UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Grupa=	1457184	41305	12157	POIANA	D	Clasic	1
6	UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Grupa=	1457184	41305	12173	POIANA	D	Clasic	2
6	UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Grupa=	1457184	41305	16110	POIANA	D	Clasic	3
6	UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Grupa=	1457184	41305	18316	POIANA	D	Clasic	4
6	UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Grupa=	1457184	41305	19156	POIANA	D	Clasic	5
6	UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Grupa=	1457184	41305	21937	POIANA	D	Clasic	6
7	UPDATE Furnizori SET Tar_Furn= WHERE Tar_Furn=	1272	36	16	ITL	N	N	0
7	UPDATE Furnizori SET Tar_Furn= WHERE Tar_Furn=	1272	36	16	ITL	D	Clasic	1
7	UPDATE Furnizori SET Tar_Furn= WHERE Tar_Furn=	1272	36	16	ITL	D	Clasic	2
7	UPDATE Furnizori SET Tar_Furn= WHERE Tar_Furn=	1272	36	16	ITL	D	Clasic	3
7	UPDATE Furnizori SET Tar_Furn= WHERE Tar_Furn=	1272	36	16	ITL	D	Clasic	4

8	DELETE FROM Furnizori WHERE Tar_Furn=	1272	38	15	AUSTRIA	N	N	0
8	DELETE FROM Furnizori WHERE Tar_Furn=	1272	38	15	AUSTRIA	D	Clasic	1
8	DELETE FROM Furnizori WHERE Tar_Furn=	1272	38	16	AUSTRIA	D	Clasic	2
8	DELETE FROM Furnizori WHERE Tar_Furn=	1272	38	16	AUSTRIA	D	Clasic	3
8	DELETE FROM Furnizori WHERE Tar_Furn=	1272	38	16	AUSTRIA	D	Clasic	4
8	DELETE FROM Furnizori WHERE Tar_Furn=	1272	36	15	ITALIA	N	N	0
8	DELETE FROM Furnizori WHERE Tar_Furn=	1272	36	16	ITALIA	D	Clasic	1
8	DELETE FROM Furnizori WHERE Tar_Furn=	1272	36	16	ITALIA	D	Clasic	2
8	DELETE FROM Furnizori WHERE Tar_Furn=	1272	36	16	ITALIA	D	Clasic	3
8	DELETE FROM Furnizori WHERE Tar_Furn=	1272	36	16	ITALIA	D	Clasic	4
9	DELETE FROM Marfuri WHERE Tar_Furn=	395736	11080	61	TOBLERON	N	N	0
9	DELETE FROM Marfuri WHERE Tar_Furn=	395736	11080	63	TOBLERON	D	Clasic	1
9	DELETE FROM Marfuri WHERE Tar_Furn=	395736	11080	61	TOBLERON	D	Clasic	2
9	DELETE FROM Marfuri WHERE Tar_Furn=	395736	11080	62	TOBLERON	D	Clasic	3
9	DELETE FROM Marfuri WHERE Tar_Furn=	395736	11080	63	TOBLERON	D	Clasic	4
9	DELETE FROM Marfuri WHERE Tar_Furn=	4243680	127310	47534	TOBLERON	N	N	0
9	DELETE FROM Marfuri WHERE Tar_Furn=	4243680	127310	35485	TOBLERON	D	Clasic	1
9	DELETE FROM Marfuri WHERE Tar_Furn=	4243680	127310	50910	TOBLERON	D	Clasic	2
9	DELETE FROM Marfuri WHERE Tar_Furn=	4243680	127310	60765	TOBLERON	D	Clasic	3
9	DELETE FROM Marfuri WHERE Tar_Furn=	4243680	127310	67082	TOBLERON	D	Clasic	4
9	DELETE FROM Marfuri WHERE Tar_Furn=	4243680	127310	89932	TOBLERON	D	Clasic	5
9	DELETE FROM Marfuri WHERE Tar_Furn=	4243680	127310	99520	TOBLERON	D	Clasic	6
10	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE Tar_Furn=	4234680	635320	90446	ITL	N	N	0
10	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE Tar_Furn=	4234680	635320	402340	ITL	D	Clasic	1
10	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE Tar_Furn=	4234680	635320	870933	ITL	D	Clasic	2
10	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE Tar_Furn=	4234680	635320	999010	ITL	D	Clasic	3
10	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE Tar_Furn=	4234680	635320	1002101	ITL	D	Clasic	4
10	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE Tar_Furn=	4234680	635320	1202204	ITL	D	Clasic	5
10	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE Tar_Furn=	4234680	635320	1590778	ITL	D	Clasic	6
10	INSERT INTO Furnizori SELECT * FROM	4234680	1058634	92740	ROM	N	N	0

	Furnizori_Noi WHERE Tar_Furn=							
10	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE Tar_Furn=	4234680	1058634	477509	ROM	D	Clasic	1
10	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE Tar_Furn=	4234680	1058634	991546	ROM	D	Clasic	2
10	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE Tar_Furn=	4234680	1058634	1080524	ROM	D	Clasic	3
10	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE Tar_Furn=	4234680	1058634	1238852	ROM	D	Clasic	4
10	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE Tar_Furn=	4234680	1058634	1474002	ROM	D	Clasic	5
10	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE Tar_Furn=	4234680	1058634	1768455	ROM	D	Clasic	6
12	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE Tar_Furn= OR Tar_Furn=	4234680	994782	185005	ITL OR AUT	N	N	0
12	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE Tar_Furn=	4234680	994782	541010	ITL OR AUT	D	Clasic	1
12	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE Tar_Furn=	4234680	994782	1308004	ITL OR AUT	D	Clasic	2
12	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE Tar_Furn=	4234680	994782	1385085	ITL OR AUT	D	Clasic	3
12	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE Tar_Furn=	4234680	994782	1460944	ITL OR AUT	D	Clasic	4
12	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE Tar_Furn=	4234680	994782	1875992	ITL OR AUT	D	Clasic	5
12	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE Tar_Furn=	4234680	994782	2270344	ITL OR AUT	D	Clasic	6
14	UPDATE Furnizori SET Tar_Furn= WHERE Tar_Furn=	4234680	1058634	880344	ROM	D	Clasic	1
14	UPDATE Furnizori SET Tar_Furn= WHERE Tar_Furn=	4234680	1058634	1022458	ROM	D	Clasic	2
14	UPDATE Furnizori SET Tar_Furn= WHERE Tar_Furn=	4234680	1058634	1542340	ROM	D	Clasic	3
14	UPDATE Furnizori SET Tar_Furn= WHERE Tar_Furn=	4234680	1058634	1974533	ROM	D	Clasic	4
14	UPDATE Furnizori SET Tar_Furn= WHERE Tar_Furn=	4234680	1058634	2502310	ROM	D	Clasic	5
14	UPDATE Furnizori SET Tar_Furn= WHERE Tar_Furn=	4234680	1058634	2992010	ROM	D	Clasic	6
14	UPDATE Furnizori SET Tar_Furn= WHERE	4234680	1058634	882012	ROM	N	N	0

	Tar_Furn=							
18	DELETE FROM Furnizori WHERE Tar_Furn=	4234680	1058634	356844	ROMANIA	D	Clasic	1
18	DELETE FROM Furnizori WHERE Tar_Furn=	4234680	1058634	485466	ROMANIA	D	Clasic	2
18	DELETE FROM Furnizori WHERE Tar_Furn=	4234680	1058634	744020	ROMANIA	D	Clasic	3
18	DELETE FROM Furnizori WHERE Tar_Furn=	4234680	1058634	783500	ROMANIA	D	Clasic	4
18	DELETE FROM Furnizori WHERE Tar_Furn=	4234680	1058634	849660	ROMANIA	D	Clasic	5
18	DELETE FROM Furnizori WHERE Tar_Furn=	4234680	1058634	927335	ROMANIA	D	Clasic	6
18	DELETE FROM Furnizori WHERE Tar_Furn=	4234680	1058634	172782	ROMANIA	N	N	0
23	SELECT count(*) FROM Marfuri WHERE Um=	4234680	1136640	1320	BUC	N	N	NULL
23	SELECT count(*) FROM Marfuri WHERE Um=	4234680	1136640	82	BUC	D	Clasic	NULL
23	SELECT count(*) FROM Marfuri WHERE Um=	4234680	1136640	19	BUC	D	Bitmap static	NULL
23	SELECT count(*) FROM Marfuri WHERE Um=	4234680	723444	1221	PAC	N	N	NULL
23	SELECT count(*) FROM Marfuri WHERE Um=	4234680	723444	71	PAC	D	Clasic	NULL
23	SELECT count(*) FROM Marfuri WHERE Um=	4234680	723444	13	PAC	D	Bitmap static	NULL
23	SELECT count(*) FROM Marfuri WHERE Um=	4234680	624456	1102	ST	N	N	NULL
23	SELECT count(*) FROM Marfuri WHERE Um=	4234680	624456	68	ST	D	Clasic	NULL
23	SELECT count(*) FROM Marfuri WHERE Um=	4234680	624456	10	ST	D	Bitmap static	NULL
25	SELECT Gest, count(*) FROM MiscMarfuri GROUP BY Gest	4540232	125	22142	ST	N	N	NULL
25	SELECT Gest, count(*) FROM MiscMarfuri GROUP BY Gest	4540232	125	19345	ST	D	Clasic	NULL
25	SELECT Gest, count(*) FROM MiscMarfuri GROUP BY Gest	4540232	125	510	ST	D	Bitmap static	NULL
27	SELECT count(*) FROM MiscMarfuri WHERE Cod_Op= AND Partener=	4540232	84560	22550	E AND ANA SRL	N	N	NULL
27	SELECT count(*) FROM MiscMarfuri WHERE Cod_Op= AND Partener=	4540232	84560	17104	E AND ANA SRL	D	Clasic	NULL
27	SELECT count(*) FROM MiscMarfuri WHERE Cod_Op= AND Partener=	4540232	84560	357	E AND ANA SRL	D	Bitmap static	NULL
27	SELECT count(*) FROM MiscMarfuri WHERE Cod_Op= AND Partener=	4540232	126452	22273	I AND VICTORIA SA	N	N	NULL
27	SELECT count(*) FROM MiscMarfuri WHERE Cod_Op= AND Partener=	4540232	126452	17109	I AND VICTORIA SA	D	Clasic	NULL
27	SELECT count(*) FROM MiscMarfuri WHERE Cod_Op= AND Partener=	4540232	126452	361	I AND VICTORIA SA	D	Bitmap static	NULL
29	SELECT * FROM Marfuri WHERE Um= OR Um=	4234680	1761096	6904	BUC OR ST	N	N	NULL
29	SELECT * FROM Marfuri WHERE Um= OR Um=	4234680	1761096	2340	BUC OR ST	D	Clasic	NULL
29	SELECT * FROM Marfuri WHERE Um= OR Um=	4234680	1761096	12106	BUC OR ST	D	Bitmap static	NULL
29	SELECT * FROM Marfuri WHERE Um= OR Um=	4234680	1347900	6702	PAC OR ST	N	N	NULL
29	SELECT * FROM Marfuri WHERE Um= OR Um=	4234680	1347900	2100	PAC OR ST	D	Clasic	NULL
29	SELECT * FROM Marfuri WHERE Um= OR Um=	4234680	1347900	2002	PAC OR ST	D	Bitmap static	NULL
31	UPDATE Marfuri SET Um= WHERE Um=	4234680	1136640	85203	BUC	N	N	NULL

31	UPDATE Marfuri SET Um= WHERE Um=	4234680	1136640	192478	BUC	D	Clasic	NULL
31	UPDATE Marfuri SET Um= WHERE Um=	4234680	1136640	81980	BUC	D	Bitmap static	NULL
31	UPDATE Marfuri SET Um= WHERE Um=	4234680	1136640	86370	buc	N	N	NULL
31	UPDATE Marfuri SET Um= WHERE Um=	4234680	1136640	194231	buc	D	Clasic	NULL
31	UPDATE Marfuri SET Um= WHERE Um=	4234680	1136640	82678	buc	D	Bitmap static	NULL
32	SELECT * FROM Marfuri WHERE CodP=	1525176	1	6990	ZAHA----00	N	Heap	NULL
32	SELECT * FROM Marfuri WHERE CodP=	1525176	1	53	ZAHA----00	N	IOT	NULL
32	SELECT * FROM Marfuri WHERE CodP=	1525176	1	97	ZAHA----00	D	Primar	NULL
32	SELECT * FROM Marfuri WHERE CodP=	1525176	1	6899	PATE----01	N	Heap	NULL
32	SELECT * FROM Marfuri WHERE CodP=	1525176	1	51	PATE----01	N	IOT	NULL
32	SELECT * FROM Marfuri WHERE CodP=	1525176	1	95	PATE----01	D	Primar	NULL
34	SELECT * FROM Marfuri WHERE Grupa=	1525176	234678	878	BOMBOANE	N	Heap	NULL
34	SELECT * FROM Marfuri WHERE Grupa=	1525176	234678	1507	BOMBOANE	N	IOT	NULL
34	SELECT * FROM Marfuri WHERE Grupa=	1525176	234678	878	BOMBOANE	D	Primar	NULL
34	SELECT * FROM Marfuri WHERE Grupa=	1525176	234678	328	BOMBOANE	D	Secundar	NULL
34	SELECT * FROM Marfuri WHERE Grupa=	1525176	234678	467	BOMBOANE	D	IOTSecundar	NULL
34	SELECT * FROM Marfuri WHERE Grupa=	1525176	79856	778	POIANA	N	Heap	NULL
34	SELECT * FROM Marfuri WHERE Grupa=	1525176	79856	1454	POIANA	N	IOT	NULL
34	SELECT * FROM Marfuri WHERE Grupa=	1525176	79856	767	POIANA	D	Primar	NULL
34	SELECT * FROM Marfuri WHERE Grupa=	1525176	79856	304	POIANA	D	Secundar	NULL
34	SELECT * FROM Marfuri WHERE Grupa=	1525176	79856	452	POIANA	D	IOTSecundar	NULL
34	SELECT * FROM Marfuri WHERE Grupa=	1525176	79856	452	POIANA	D	IOTSecundar	NULL
36	UPDATE Marfuri SET CodP= WHERE CodP=	1525176	1	9782	ZAHA----00	N	Heap	NULL
36	UPDATE Marfuri SET CodP= WHERE CodP=	1525176	1	1392	ZAHA----00	D	Primar	NULL
36	UPDATE Marfuri SET CodP= WHERE CodP=	1525176	1	135	ZAHA----00	N	IOT	NULL
36	UPDATE Marfuri SET CodP= WHERE CodP=	1525176	1	9881	VIN-----01	N	Heap	NULL
36	UPDATE Marfuri SET CodP= WHERE CodP=	1525176	1	1423	VIN-----01	D	Primar	NULL
36	UPDATE Marfuri SET CodP= WHERE CodP=	1525176	1	145	VIN-----01	N	IOT	NULL
38	UPDATE Marfuri SET Grupa= WHERE Grupa=	1525176	29458	8227	ZAHAR	N	Heap	NULL
38	UPDATE Marfuri SET Grupa= WHERE Grupa=	1525176	29458	156	ZAHAR	D	Primar	NULL
38	UPDATE Marfuri SET Grupa= WHERE Grupa=	1525176	29458	578	ZAHAR	N	IOT	NULL
38	UPDATE Marfuri SET Grupa= WHERE Grupa=	1525176	29458	8215	zahar	N	Heap	NULL
38	UPDATE Marfuri SET Grupa= WHERE Grupa=	1525176	29458	143	zahar	D	Primar	NULL
38	UPDATE Marfuri SET Grupa= WHERE Grupa=	1525176	29458	564	zahar	N	IOT	NULL
38	UPDATE Marfuri SET Grupa= WHERE Grupa=	2567853	76510	11644	POIANA	N	Heap	NULL
38	UPDATE Marfuri SET Grupa= WHERE Grupa=	2567853	76510	162	POIANA	D	Primar	NULL
38	UPDATE Marfuri SET Grupa= WHERE Grupa=	2567853	76510	382	POIANA	N	IOT	NULL
38	UPDATE Marfuri SET Grupa= WHERE Grupa=	4543667	120915	22892	TOBLERON	N	Heap	NULL

38	UPDATE Marfuri SET Grupa= WHERE Grupa=	4543667	120915	1679	TOBLERON	D	Primar	NULL
38	UPDATE Marfuri SET Grupa= WHERE Grupa=	4543667	120915	1484	TOBLERON	N	IOT	NULL
39	DELETE FROM Marfuri WHERE CodP=	1525176	1	7689	ZAHA----00	N	Heap	NULL
39	DELETE FROM Marfuri WHERE CodP=	1525176	1	527	ZAHA----00	D	Primar	NULL
39	DELETE FROM Marfuri WHERE CodP=	1525176	1	118	ZAHA----00	N	IOT	NULL
39	DELETE FROM Marfuri WHERE CodP=	1525176	1	7692	VIN-----01	N	Heap	NULL
39	DELETE FROM Marfuri WHERE CodP=	1525176	1	532	VIN-----01	D	Primar	NULL
39	DELETE FROM Marfuri WHERE CodP=	1525176	1	124	VIN-----01	N	IOT	NULL
40	DELETE FROM Marfuri WHERE Grupa=	1525176	29458	8104	ZAHAR	N	Heap	NULL
40	DELETE FROM Marfuri WHERE Grupa=	1525176	29458	8434	ZAHAR	D	Primar	NULL
40	DELETE FROM Marfuri WHERE Grupa=	1525176	29458	16325	ZAHAR	N	IOT	NULL
40	DELETE FROM Marfuri WHERE Grupa=	2567853	76510	11734	POIANA	N	Heap	NULL
40	DELETE FROM Marfuri WHERE Grupa=	2567853	76510	399	POIANA	D	Primar	NULL
40	DELETE FROM Marfuri WHERE Grupa=	2567853	76510	434	POIANA	N	IOT	NULL

REZULTATE_FINALE

NR_OPER	OPERATIE	TOTALINREG	INREGPREL	TIMP	VALOARE	INDEX	TIPINDEX	NRINDEX
1	SELECT * FROM Furnizori WHERE Cod_Furn=	424	1	25	NULL	D	Clasic	1
1	SELECT * FROM Furnizori WHERE Cod_Furn=	848	1	25	NULL	D	Clasic	1
1	SELECT * FROM Furnizori WHERE Cod_Furn=	6360	1	25	NULL	D	Clasic	1
1	SELECT * FROM Furnizori WHERE Cod_Furn=	424	1	16	NULL	N	N	0
1	SELECT * FROM Furnizori WHERE Cod_Furn=	848	1	16	NULL	N	N	0
1	SELECT * FROM Furnizori WHERE Cod_Furn=	6360	1	52	NULL	N	N	0
2	SELECT * FROM Furnizori WHERE Loc_Furn=	424	12	31	NULL	D	Clasic	1
2	SELECT * FROM Furnizori WHERE Loc_Furn=	848	24	31	NULL	D	Clasic	1
2	SELECT * FROM Furnizori WHERE Loc_Furn=	6360	180	39	NULL	D	Clasic	1
2	SELECT * FROM Furnizori WHERE Loc_Furn=	424	12	16	NULL	N	N	0
2	SELECT * FROM Furnizori WHERE Loc_Furn=	848	24	16	NULL	N	N	0
2	SELECT * FROM Furnizori WHERE Loc_Furn=	6360	180	86	NULL	N	N	0
3	SELECT * FROM Marfuri WHERE DenP=	1263	12	31	NULL	D	Clasic	1
3	SELECT * FROM Marfuri WHERE DenP=	214710	2040	78	NULL	D	Clasic	1
3	SELECT * FROM Marfuri WHERE DenP=	4234680	40236	320	NULL	D	Clasic	1
3	SELECT * FROM Marfuri WHERE DenP=	1263	12	16	NULL	N	N	0
3	SELECT * FROM Marfuri WHERE DenP=	214710	2040	609	NULL	N	N	0
3	SELECT * FROM Marfuri WHERE DenP=	4234680	40236	2320	NULL	N	N	0
4	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi	1272	38	16	NULL	D	Clasic	1
4	INSERT INTO Furnizori	1272	38	16	NULL	D	Clasic	2

	SELECT * FROM Furnizori_Noi							
4	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi	1272	38	16	NULL	D	Clasic	3
4	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi	1272	38	16	NULL	D	Clasic	4
4	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi	1272	38	16	NULL	N	N	0
5	INSERT INTO Marfuri SELECT * FROM Marfuri_Noi	395736	11818	31	NULL	N	N	0
5	INSERT INTO Marfuri SELECT * FROM Marfuri_Noi	4243680	127040	609	NULL	N	N	0
5	INSERT INTO Marfuri SELECT * FROM Marfuri_Noi	395736	11818	31	NULL	D	Clasic	1
5	INSERT INTO Marfuri SELECT * FROM Marfuri_Noi	4243680	127040	12922	NULL	D	Clasic	1
5	INSERT INTO Marfuri SELECT * FROM Marfuri_Noi	395736	11818	31	NULL	D	Clasic	2
5	INSERT INTO Marfuri SELECT * FROM Marfuri_Noi	4243680	127040	26078	NULL	D	Clasic	2
5	INSERT INTO Marfuri SELECT * FROM Marfuri_Noi	395736	11818	32	NULL	D	Clasic	3
5	INSERT INTO Marfuri SELECT * FROM Marfuri_Noi	4243680	127040	29844	NULL	D	Clasic	3
5	INSERT INTO Marfuri SELECT * FROM Marfuri_Noi	395736	11818	31	NULL	D	Clasic	4
5	INSERT INTO Marfuri SELECT * FROM Marfuri_Noi	4243680	127040	34453	NULL	D	Clasic	4
	SELECT * FROM Marfuri_Noi							
5	INSERT INTO Marfuri SELECT * FROM Marfuri_Noi	4243680	127040	45813	NULL	D	Clasic	5
5	INSERT INTO Marfuri SELECT * FROM Marfuri_Noi	4243680	127040	62313	NULL	D	Clasic	6
6	UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Grupa=	395736	11834	31	NULL	N	N	0
6	UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Grupa=	1457184	41305	13734	NULL	N	N	0
6	UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Grupa=	395736	11834	31	NULL	D	Clasic	1
6	UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Grupa=	1457184	41305	12156	NULL	D	Clasic	1
6	UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Grupa=	395736	11834	31	NULL	D	Clasic	2
6	UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Grupa=	1457184	41305	12172	NULL	D	Clasic	2
6	UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Grupa=	395736	11834	32	NULL	D	Clasic	3
6	UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Grupa=	1457184	41305	16110	NULL	D	Clasic	3
6	UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Grupa=	395736	11834	31	NULL	D	Clasic	4

6	UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Grupa=	1457184	41305	18312	NULL	D	Clasic	4
6	UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Grupa=	1457184	41305	19156	NULL	D	Clasic	5
6	UPDATE Marfuri SET Pret_Achiz=Pret_Achiz*1.1 WHERE Grupa=	1457184	41305	21938	NULL	D	Clasic	6
8	DELETE FROM Furnizori WHERE Tar_Furn=	1272	37	15	NULL	D	Clasic	1
8	DELETE FROM Furnizori WHERE Tar_Furn=	1272	37	16	NULL	D	Clasic	2
8	DELETE FROM Furnizori WHERE Tar_Furn=	1272	37	16	NULL	D	Clasic	3
8	DELETE FROM Furnizori WHERE Tar_Furn=	1272	37	16	NULL	D	Clasic	4
8	DELETE FROM Furnizori WHERE Tar_Furn=	1272	37	15	NULL	N	N	0
10	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE Tar_Furn=	4234680	995678	475609	NULL	D	Clasic	1
10	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE Tar_Furn=	4234680	995678	990766	NULL	D	Clasic	2
10	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE Tar_Furn=	4234680	995678	1010034	NULL	D	Clasic	3
10	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE Tar_Furn=	4234680	995678	1198844	NULL	D	Clasic	4
10	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE Tar_Furn=	4234680	995678	1464032	NULL	D	Clasic	5
10	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE Tar_Furn=	4234680	995678	1768875	NULL	D	Clasic	6
10	INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE Tar_Furn=	4234680	995678	93157	NULL	N	N	0
14	UPDATE Furnizori SET Tar_Furn= WHERE Tar_Furn=	4234680	995678	879922	NULL	D	Clasic	1
14	UPDATE Furnizori SET Tar_Furn= WHERE Tar_Furn=	4234680	995678	1022006	NULL	D	Clasic	2
14	UPDATE Furnizori SET Tar_Furn= WHERE Tar_Furn=	4234680	995678	1541001	NULL	D	Clasic	3
18	DELETE FROM Furnizori WHERE Tar_Furn=	4234680	995678	849060	NULL	D	Clasic	5
18	DELETE FROM Furnizori WHERE Tar_Furn=	4234680	995678	926020	NULL	D	Clasic	6
18	DELETE FROM Furnizori WHERE Tar_Furn=	4234680	995678	171211	NULL	N	N	0
23	SELECT count(*) FROM Marfuri WHERE Um=	4234680	624456	79	NULL	D	Clasic	NULL
25	SELECT Gest, count(*) FROM MiscMarfuri GROUP BY Gest	4540232	125	19528	NULL	D	Clasic	NULL
27	SELECT count(*) FROM MiscMarfuri WHERE Cod_Op= AND Partener=	4540232	79305	17045	NULL	D	Clasic	NULL
29	SELECT * FROM Marfuri WHERE Um= OR Um=	4234680	1546034	2133	NULL	D	Clasic	NULL
31	UPDATE Marfuri SET Um= WHERE Um=	4234680	1693800	195375	NULL	D	Clasic	NULL

23	SELECT count(*) FROM Marfuri WHERE Um=	4234680	624456	15	NULL	D	Bitmap static	NULL
25	SELECT Gest, count(*) FROM MiscMarfuri GROUP BY Gest	4540232	125	521	NULL	D	Bitmap static	NULL
27	SELECT count(*) FROM MiscMarfuri WHERE Cod_Op= AND Partener=	4540232	79305	340	NULL	D	Bitmap static	NULL
29	SELECT * FROM Marfuri WHERE Um= OR Um=	4234680	1546034	2031	NULL	D	Bitmap static	NULL
31	UPDATE Marfuri SET Um= WHERE Um=	4234680	1693800	83409	NULL	D	Bitmap static	NULL
23	SELECT count(*) FROM Marfuri WHERE Um=	4234680	624456	1250	NULL	N	N	NULL
25	SELECT Gest, count(*) FROM MiscMarfuri GROUP BY Gest	4540232	125	22022	NULL	N	N	NULL
27	SELECT count(*) FROM MiscMarfuri WHERE Cod_Op= AND Partener=	4540232	79305	22552	NULL	N	N	NULL
29	SELECT * FROM Marfuri WHERE Um= OR Um=	4234680	1546034	6802	NULL	N	N	NULL
31	UPDATE Marfuri SET Um= WHERE Um=	4234680	1693800	87016	NULL	N	N	NULL
32	SELECT * FROM Marfuri WHERE CodP=	1525176	1	94	NULL	D	Primar	NULL
34	SELECT * FROM Marfuri WHERE Grupa=	1525176	125346	871	NULL	D	Primar	NULL
36	UPDATE Marfuri SET CodP= WHERE CodP=	1525176	1	1402	NULL	D	Primar	NULL
38	UPDATE Marfuri SET Grupa= WHERE Grupa=	1525176	34560	150	NULL	D	Primar	NULL
39	DELETE FROM Marfuri WHERE CodP=	1525176	1	521	NULL	D	Primar	NULL
40	DELETE FROM Marfuri WHERE Grupa=	1525176	24566	8332	NULL	D	Primar	NULL
34	SELECT * FROM Marfuri WHERE Grupa=	1525176	125346	310	NULL	D	Secundar	NULL
34	SELECT * FROM Marfuri WHERE Grupa=	1525176	125346	461	NULL	D	IOTSecundar	NULL
32	SELECT * FROM Marfuri WHERE Grupa=	1525176	1	52	NULL	N	IOT	NULL
32	SELECT * FROM Marfuri WHERE CodP=	1525176	1	52	NULL	N	IOT	NULL
34	SELECT * FROM Marfuri WHERE Grupa=	1525176	125346	1503	NULL	N	IOT	NULL
36	UPDATE Marfuri SET CodP= WHERE CodP=	1525176	1	140	NULL	N	IOT	NULL
38	UPDATE Marfuri SET Grupa= WHERE Grupa=	1525176	34560	571	NULL	N	IOT	NULL
39	DELETE FROM Marfuri WHERE CodP=	1525176	1	110	NULL	N	IOT	NULL
40	DELETE FROM Marfuri WHERE Grupa=	1525176	24566	16283	NULL	N	IOT	NULL
32	SELECT * FROM Marfuri WHERE CodP=	1525176	1	6990	NULL	N	Heap	NULL
34	SELECT * FROM Marfuri WHERE Grupa=	1525176	125346	871	NULL	N	Heap	NULL
36	UPDATE Marfuri SET CodP= WHERE CodP=	1525176	1	9874	NULL	N	Heap	NULL
38	UPDATE Marfuri SET Grupa= WHERE Grupa=	1525176	34560	8222	NULL	N	Heap	NULL
39	DELETE FROM Marfuri WHERE CodP=	1525176	1	7681	NULL	N	Heap	NULL
40	DELETE FROM Marfuri WHERE Grupa=	1525176	24566	8050	NULL	N	Heap	NULL

1.3. *Scripturi şi proceduri stocate*

```
--FurnizoriInsert1.sql
        DECLARE
         v_NrOp NUMBER(3);
         v_Operatie VARCHAR(200);
         v_Start NUMBER(10,0);
         v_End NUMBER(10,0);
         v_Timp NUMBER(10,0);
         v_NrInreg NUMBER(12,0);
         v_Index CHAR(1);
         v_TipIndex VARCHAR(20);
         v_NrIndex NUMBER(2,0);
         v_Tar_Furn VARCHAR(15);
         v_TotalInreg NUMBER(15,0);
         CURSOR c_TotalInreg IS
           SELECT count(*)
           FROM Furnizori;
        CURSOR c_NrInreg IS
          SELECT count(*)
          FROM Furnizori_Noi
          WHERE Tar_Furn=v_Tar_Furn;
       BEGIN
        v_NrOp:=&NrOp;
        v_Tar_Furn:= '&Tar_Furn';
        OPEN c_TotalInreg;
        FETCH c_TotalInreg INTO v_TotalInreg;
        CLOSE c_TotalInreg;
        OPEN c_NrInreg;
        FETCH c_NrInreg INTO v_NrInreg;
        CLOSE c_NrInreg;
        v_Index:= '&Index';
        v_TipIndex:= '&TipIndex';
        v_NrIndex:=&NrIndex
        v_Operatie:='INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE Tar_Furn=';
        v_Start:=DBMS_UTILITY.GET_TIME;
        INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE Tar_Furn=v_Tar_Furn;
        v_End:=DBMS_UTILITY.GET_TIME;
        v_Timp:=(v_End-v_Start)*10;
        INSERT INTO Rezultate(NrOp, Operatie, TotalInreg, InregPrel,Timp, Valoare, Index, TipIndex,
```

```
            NrIndex) VALUES (v_NrOp, v_Operatie, v_TotalInreg, v_NrInreg, v_Timp, v_Tar_Furn,
            v_Index, v_TipIndex, v_NrIndex);
        COMMIT;
END;
```

--FurnizoriInsertCompus.sql

```
        DECLARE
         v_NrOp NUMBER(3);
         v_Operatie VARCHAR(200);
         v_Start NUMBER(10,0);
         v_End NUMBER(10,0);
         v_Timp NUMBER(10,0);
         v_NrInreg NUMBER(12,0);
         v_Index CHAR(1);
         v_TipIndex VARCHAR(20);
         v_NrIndex NUMBER(2,0);
         v_Tar_Furn1 VARCHAR(15);
         v_Tar_Furn2 VARCHAR(15);
         v_TotalInreg NUMBER(15,0);
         CURSOR c_TotalInreg IS
          SELECT count(*)
          FROM Furnizori;
        CURSOR c_NrInreg IS
          SELECT count(*)
          FROM Furnizori_Noi
          WHERE Tar_Furn=v_Tar_Furn1 OR Tar_Furn=v_Tar_Furn2;
        BEGIN
         v_NrOp:=&NrOp;
         v_Tar_Furn1:= '&Tar_Furn1';
         v_Tar_Furn2:= '&Tar_Furn2';
         OPEN c_TotalInreg;
         FETCH c_TotalInreg INTO v_TotalInreg;
         CLOSE c_TotalInreg;
         OPEN c_NrInreg;
         FETCH c_NrInreg INTO v_NrInreg;
         CLOSE c_NrInreg;
         v_Index:= '&Index';
         v_TipIndex:= '&TipIndex';
         v_NrIndex:=&NrIndex;
```

```
        v_Operatie:='INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE Tar_Furn= OR
Tar_Furn= ';
        v_Start:=DBMS_UTILITY.GET_TIME;
        INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE Tar_Furn=v_Tar_Furn1 OR
Tar_Furn=v_Tar_Furn2;
        v_End:=DBMS_UTILITY.GET_TIME;
        v_Timp:=(v_End-v_Start)*10;
        v_Tar_Furn1:=v_Tar_Furn1|| ' OR ' ||v_Tar_Furn2
        INSERT INTO Rezultate(NrOp, Operatie, TotalInreg, InregPrel,Timp, Valoare, Index, TipIndex,
              NrIndex) VALUES (v_NrOp, v_Operatie, v_TotalInreg, v_NrInreg, v_Timp, v_Tar_Furn1,
              v_Index, v_TipIndex, v_NrIndex);
        COMMIT;
END;
```

--MarfuriInsert1.sql

```
      DECLARE
       v_NrOp NUMBER(3);
       v_Operatie VARCHAR(200);
       v_Start NUMBER(10,0);
       v_End NUMBER(10,0);
       v_Timp NUMBER(10,0);
       v_NrInreg NUMBER(12,0);
       v_Index CHAR(1);
       v_TipIndex VARCHAR(20);
       v_NrIndex NUMBER(2,0);
       v_Gupa VARCHAR(8);
       v_TotalInreg NUMBER(15,0);
       CURSOR c_TotalInreg IS
         SELECT count(*)
         FROM Marfuri;
      CURSOR c_NrInreg IS
         SELECT count(*)
         FROM Marfuri_Noi
         WHERE Grupa=v_Grupa;
      BEGIN
       v_NrOp:=&NrOp;
       v_Grupa:= '&Grupa';
       OPEN c_TotalInreg;
       FETCH c_TotalInreg INTO v_TotalInreg;
       CLOSE c_TotalInreg;
```

```
 OPEN c_NrInreg;
 FETCH c_NrInreg INTO v_NrInreg;
 CLOSE c_NrInreg;
 v_Index:= '&Index';
 v_TipIndex:= '&TipIndex';
 v_NrIndex:=&NrIndex;
 v_Operatie:='INSERT INTO Marfuri SELECT * FROM Marfuri_Noi WHERE Grupa=';
 v_Start:=DBMS_UTILITY.GET_TIME;
 INSERT INTO Marfuri SELECT * FROM Marfuri_Noi WHERE Grupa=v_Grupa;
 v_End:=DBMS_UTILITY.GET_TIME;
 v_Timp:=(v_End-v_Start)*10;
 INSERT INTO Rezultate(NrOp, Operatie, TotalInreg, InregPrel,Timp, Valoare, Index, TipIndex,
        NrIndex) VALUES (v_NrOp, v_Operatie, v_TotalInreg, v_NrInreg, v_Timp, v_Grupa,
        v_Index, v_TipIndex, v_NrIndex);
 COMMIT;
END;
```

--MarfuriSelect1.sql

```
DECLARE
 v_NrOp NUMBER(3);
 v_Operatie VARCHAR(200);
 v_Start NUMBER(10,0);
 v_End NUMBER(10,0);
 v_Timp NUMBER(10,0);
 v_Valoare VARCHAR(50);
 v_NrInreg NUMBER(12,0);
 v_Index CHAR(1);
 v_TipIndex VARCHAR(20);
 v_Um Marfuri.Um%TYPE;
 v_TotalInreg NUMBER(15,0);
 CURSOR c_Marfuri IS
  SELECT count(*)
  FROM Marfuri
  WHERE Um=v_Um;
 CURSOR c_TotalInreg IS
  SELECT count(*)
  FROM Marfuri;
BEGIN
 v_NrOp:=&NrOp;
 OPEN c_TotalInreg;
```

```
  FETCH c_TotalInreg INTO v_TotalInreg;
  CLOSE c_TotalInreg;
  v_Um:='&Um';
  v_Index:= '&Index';
  v_TipIndex:= '&TipIndex';
  v_Operatie:='SELECT count(*) FROM Marfuri WHERE Um=';
  v_Start:=DBMS_UTILITY.GET_TIME;
  OPEN c_Marfuri;
  v_End:=DBMS_UTILITY.GET_TIME;
  v_Timp:=(v_End-v_Start)*10;
  FETCH c_Marfuri INTO v_NrInreg;
  CLOSE c_Marfuri;
  v_Valoare:=v_Um
  INSERT INTO Rezultate(NrOp, Operatie, TotalInreg, InregPrel,Timp, Valoare, Index, TipIndex,
        NrIndex) VALUES (v_NrOp, v_Operatie, v_TotalInreg, v_NrInreg, v_Timp, v_Valoare,
        v_Index, v_TipIndex, null);
  COMMIT;
 END;
```

--MarfuriSelect2sql

```
DECLARE
 v_NrOp NUMBER(3);
 v_Operatie VARCHAR(200);
 v_Start NUMBER(10,0);
 v_End NUMBER(10,0);
 v_Timp NUMBER(10,0);
 v_Valoare VARCHAR(50);
 v_NrInreg NUMBER(12,0);
 v_Index CHAR(1);
 v_TipIndex VARCHAR(20);
 v_Um1 Marfuri.Um%TYPE;
 v_Um Marfuri.Um%TYPE;
 v_TotalInreg NUMBER(15,0);
 CURSOR c_Marfuri IS
  SELECT DISTINCT Um
  FROM Marfuri
  WHERE Um<>v_Um;
 CURSOR c_TotalInreg IS
  SELECT count(*)
  FROM Marfuri;
```

```
BEGIN
 v_NrOp:=&NrOp;
 OPEN c_TotalInreg;
 FETCH c_TotalInreg INTO v_TotalInreg;
 CLOSE c_TotalInreg;
 v_Um:='&Um';
 v_Index:= '&Index';
 v_TipIndex:= '&TipIndex';
 v_Operatie:='SELECT DISTINCT Um FROM Marfuri WHERE Um<>';
 v_Start:=DBMS_UTILITY.GET_TIME;
 OPEN c_Marfuri;
 v_End:=DBMS_UTILITY.GET_TIME;
 v_Timp:=(v_End-v_Start)*10;
 LOOP
   FETCH c_Marfuri INTO v_Um1;
   EXIT WHEN c_Marfuri%NOTFOUND;
 END LOOP;
 v_NrInreg:=c_Marfuri%ROWCOUNT;
 CLOSE c_Marfuri;
 v_Valoare:=v_Um
 INSERT INTO Rezultate(NrOp, Operatie, TotalInreg, InregPrel,Timp, Valoare, Index, TipIndex,
        NrIndex) VALUES (v_NrOp, v_Operatie, v_TotalInreg, v_NrInreg, v_Timp, v_Valoare,
        v_Index, v_TipIndex, null);
  COMMIT;
END;
```

--MarfuriSelect4sql

```
DECLARE
 v_NrOp NUMBER(3);
 v_Operatie VARCHAR(200);
 v_Start NUMBER(10,0);
 v_End NUMBER(10,0);
 v_Timp NUMBER(10,0);
 v_Valoare VARCHAR(50);
 v_NrInreg NUMBER(12,0);
 v_Index CHAR(1);
 v_TipIndex VARCHAR(20);
 v_DenP Marfuri.DenP%TYPE;
 v_Um Marfuri.Um%TYPE;
 v_Um1 Marfuri.Um%TYPE;
```

```
 v_TotalInreg NUMBER(15,0);
 CURSOR c_Marfuri IS
  SELECT DenP, Um
  FROM Marfuri
  WHERE Um=v_Um;
 CURSOR c_TotalInreg IS
  SELECT count(*)
  FROM Marfuri;
BEGIN
 v_NrOp:=&NrOp;
 OPEN c_TotalInreg;
 FETCH c_TotalInreg INTO v_TotalInreg;
 CLOSE c_TotalInreg;
 v_Um:='&Um';
 v_Index:= '&Index';
 v_TipIndex:= '&TipIndex';
 v_Operatie:='SELECT DenP, Um FROM Marfuri WHERE Um=';
 v_Start:=DBMS_UTILITY.GET_TIME;
 OPEN c_Marfuri;
 v_End:=DBMS_UTILITY.GET_TIME;
 v_Timp:=(v_End-v_Start)*10;
 LOOP
   FETCH c_Marfuri INTO v_DenP, v_Um1;
   EXIT WHEN c_Marfuri%NOTFOUND;
 END LOOP;
 v_NrInreg:=c_Marfuri%ROWCOUNT;
 CLOSE c_Marfuri;
 v_Valoare:=v_Um
 INSERT INTO Rezultate(NrOp, Operatie, TotalInreg, InregPrel,Timp, Valoare, Index, TipIndex,
        NrIndex) VALUES (v_NrOp, v_Operatie, v_TotalInreg, v_NrInreg, v_Timp, v_Valoare,
        v_Index, v_TipIndex, null);
 COMMIT;
END;
```

--MiscMarfuriSelect1.sql

```
DECLARE
 v_NrOp NUMBER(3);
 v_Operatie VARCHAR(200);
 v_Start NUMBER(10,0);
 v_End NUMBER(10,0);
```

```
 v_Timp NUMBER(10,0);
 v_NrInreg NUMBER(12,0);
 v_Gest MiscMarfuri.Gest%TYPE;
 v_Count NUMBER(15,0);
 v_Index CHAR(1);
 v_TipIndex VARCHAR(20);
 v_TotalInreg NUMBER(15,0);
 CURSOR c_MiscMarfuri IS
  SELECT Gest, count(*)
  FROM MiscMarfuri
  GROUP BY Gest;
 CURSOR c_TotalInreg IS
  SELECT count(*)
  FROM MiscMarfuri;
BEGIN
 v_NrOp:=&NrOp;
 OPEN c_TotalInreg;
 FETCH c_TotalInreg INTO v_TotalInreg;
 CLOSE c_TotalInreg;
 v_Index:= '&Index';
 v_TipIndex:= '&TipIndex';
 v_Operatie:='SELECT Gest, count(*) FROM MiscMarfuri GROUP BY Gest';
 v_Start:=DBMS_UTILITY.GET_TIME;
 OPEN c_MiscMarfuri;
 v_End:=DBMS_UTILITY.GET_TIME;
 v_Timp:=(v_End-v_Start)*10;
 LOOP
   FETCH c_MiscMarfuri INTO v_Gest, v_Count;
   EXIT WHEN c_MiscMarfuri%NOTFOUND;
 END LOOP;
 v_NrInreg:=c_MiscMarfuri%ROWCOUNT;
 CLOSE c_MiscMarfuri;
 INSERT INTO Rezultate(NrOp, Operatie, TotalInreg, InregPrel,Timp, Valoare, Index, TipIndex,
        NrIndex) VALUES (v_NrOp, v_Operatie, v_TotalInreg, v_NrInreg, v_Timp, null,
        v_Index, v_TipIndex, null);
 COMMIT;
END;
```

--MiscMarfuriSelect2.sql

```
DECLARE
```

```
 v_NrOp NUMBER(3);
 v_Operatie VARCHAR(200);
 v_Start NUMBER(10,0);
 v_End NUMBER(10,0);
 v_Timp NUMBER(10,0);
 v_NrInreg NUMBER(12,0);
 v_Cod_Oper MiscMarfuri.Cod_Oper%TYPE;
 v_Index CHAR(1);
 v_TipIndex VARCHAR(20);
 v_TotalInreg NUMBER(15,0);
 CURSOR c_MiscMarfuri IS
  SELECT DISTINCT Cod_Oper
  FROM MiscMarfuri;
 CURSOR c_TotalInreg IS
  SELECT count(*)
  FROM MiscMarfuri;
BEGIN
 v_NrOp:=&NrOp;
 OPEN c_TotalInreg;
 FETCH c_TotalInreg INTO v_TotalInreg;
 CLOSE c_TotalInreg;
 v_Index:= '&Index';
 v_TipIndex:= '&TipIndex';
 v_Operatie:='SELECT DISTINCT Cod_Oper FROM MiscMarfuri';
 v_Start:=DBMS_UTILITY.GET_TIME;
 OPEN c_MiscMarfuri;
 v_End:=DBMS_UTILITY.GET_TIME;
 v_Timp:=(v_End-v_Start)*10;
 LOOP
   FETCH c_MiscMarfuri INTO v_Cod_Oper;
   EXIT WHEN c_MiscMarfuri%NOTFOUND;
 END LOOP;
 v_NrInreg:=c_MiscMarfuri%ROWCOUNT;
 CLOSE c_MiscMarfuri;
 INSERT INTO Rezultate(NrOp, Operatie, TotalInreg, InregPrel,Timp, Valoare, Index, TipIndex,
        NrIndex) VALUES (v_NrOp, v_Operatie, v_TotalInreg, v_NrInreg, v_Timp, null,
        v_Index, v_TipIndex, null);
 COMMIT;
END;
```

--MiscMarfuriSelect4.sql

```
DECLARE
 v_NrOp NUMBER(3);
 v_Operatie VARCHAR(200);
 v_Start NUMBER(10,0);
 v_End NUMBER(10,0);
 v_Timp NUMBER(10,0);
 v_NrInreg NUMBER(12,0);
 v_Index CHAR(1);
 v_Cod_Oper VARCHAR(50);
 v_TipIndex VARCHAR(20);
 v_TotalInreg NUMBER(15,0);
 CURSOR c_MiscMarfuri IS
  SELECT count(*)
  FROM MiscMarfuri
  WHERE Cod_Oper IN v_Cod_Oper;
 CURSOR c_TotalInreg IS
  SELECT count(*)
  FROM MiscMarfuri;
BEGIN
 v_NrOp:=&NrOp;
 OPEN c_TotalInreg;
 FETCH c_TotalInreg INTO v_TotalInreg;
 CLOSE c_TotalInreg;
 v_Cod_Oper:= '&Cod_Oper';
 v_Index:= '&Index';
 v_TipIndex:= '&TipIndex';
 v_Operatie:='SELECT count(*) FROM MiscMarfuri WHERE Cod_Oper IN ';
 v_Start:=DBMS_UTILITY.GET_TIME;
 OPEN c_MiscMarfuri;
 v_End:=DBMS_UTILITY.GET_TIME;
 v_Timp:=(v_End-v_Start)*10;
 FETCH c_MiscMarfuri INTO v_NrInreg;
 CLOSE c_MiscMarfuri;
 INSERT INTO Rezultate(NrOp, Operatie, TotalInreg, InregPrel,Timp, Valoare, Index, TipIndex,
        NrIndex) VALUES (v_NrOp, v_Operatie, v_TotalInreg, v_NrInreg, v_Timp, v_Cod_Oper,
        v_Index, v_TipIndex, null);
 COMMIT;
END;
```

--MarfuriIOTSelect2.sql

```
DECLARE
 v_NrOp NUMBER(3);
 v_Operatie VARCHAR(200);
 v_Start NUMBER(10,0);
 v_End NUMBER(10,0);
 v_Timp NUMBER(10,0);
 v_Valoare VARCHAR(50);
 v_NrInreg NUMBER(12,0);
 v_Index CHAR(1);
 v_TipIndex VARCHAR(20);
 v_NrIndex NUMBER(2,0);
 v_Marfuri Marfuri%ROWTYPE;
 v_CodP VARCHAR(50);
 v_TotalInreg NUMBER(15,0);
 CURSOR c_Marfuri IS
  SELECT *
  FROM Marfuri
  WHERE CodP IN v_CodP;
 CURSOR c_TotalInreg IS
  SELECT count(*)
  FROM Marfuri;
BEGIN
 v_NrOp:=&NrOp;
 OPEN c_TotalInreg;
 FETCH c_TotalInreg INTO v_TotalInreg;
 CLOSE c_TotalInreg;
 v_CodP:='&CodP';
 v_Index:= '&Index';
 v_TipIndex:= '&TipIndex';
 v_NrIndex:= '&NrIndex';
 v_Operatie:='SELECT * FROM Marfuri WHERE CodP IN ';
 v_Start:=DBMS_UTILITY.GET_TIME;
 OPEN c_Marfuri;
 v_End:=DBMS_UTILITY.GET_TIME;
 v_Timp:=(v_End-v_Start)*10;
 LOOP
   FETCH c_Marfuri INTO v_Marfuri;
   EXIT WHEN c_Marfuri%NOTFOUND;
 END LOOP;
```

```
 v_NrInreg:=c_Marfuri%ROWCOUNT;
 CLOSE c_Marfuri;
 v_Valoare:=v_CodP
 INSERT INTO Rezultate(NrOp, Operatie, TotalInreg, InregPrel,Timp, Valoare, Index, TipIndex,
        NrIndex) VALUES (v_NrOp, v_Operatie, v_TotalInreg, v_NrInreg, v_Timp, v_Valoare,
        v_Index, v_TipIndex, null);
 COMMIT;
END;
```

--MarfuriIOTSelect3.sql

```
DECLARE
 v_NrOp NUMBER(3);
 v_Operatie VARCHAR(200);
 v_Start NUMBER(10,0);
 v_End NUMBER(10,0);
 v_Timp NUMBER(10,0);
 v_Valoare VARCHAR(50);
 v_NrInreg NUMBER(12,0);
 v_Index CHAR(1);
 v_TipIndex VARCHAR(20);
 v_NrIndex NUMBER(2,0);
 v_Marfuri Marfuri%ROWTYPE;
 v_Grupa Marfuri.Grupa%TYPE;
 v_TotalInreg NUMBER(15,0);
 CURSOR c_Marfuri IS
  SELECT *
  FROM Marfuri
  WHERE Grupa=v_Grupa;
 CURSOR c_TotalInreg IS
  SELECT count(*)
  FROM Marfuri;
BEGIN
 v_NrOp:=&NrOp;
 OPEN c_TotalInreg;
 FETCH c_TotalInreg INTO v_TotalInreg;
 CLOSE c_TotalInreg;
 v_Grupa:='&Grupa';
 v_Index:= '&Index';
 v_TipIndex:= '&TipIndex';
 v_NrIndex:= '&NrIndex';
```

```
    v_Operatie:='SELECT * FROM Marfuri WHERE Grupa= ';
    v_Start:=DBMS_UTILITY.GET_TIME;
    OPEN c_Marfuri;
    v_End:=DBMS_UTILITY.GET_TIME;
    v_Timp:=(v_End-v_Start)*10;
    LOOP
      FETCH c_Marfuri INTO v_Marfuri;
      EXIT WHEN c_Marfuri%NOTFOUND;
    END LOOP;
    v_NrInreg:=c_Marfuri%ROWCOUNT;
    CLOSE c_Marfuri;
    v_Valoare:=v_Grupa
    INSERT INTO Rezultate(NrOp, Operatie, TotalInreg, InregPrel,Timp, Valoare, Index, TipIndex,
          NrIndex) VALUES (v_NrOp, v_Operatie, v_TotalInreg, v_NrInreg, v_Timp, v_Valoare,
          v_Index, v_TipIndex, null);
    COMMIT;
  END;
```

--MarfuriIOTModificare1.sql

```
  DECLARE
    v_NrOp NUMBER(3);
    v_Operatie VARCHAR(200);
    v_Start NUMBER(10,0);
    v_End NUMBER(10,0);
    v_Timp NUMBER(10,0);
    v_NrInreg NUMBER(12,0);
    v_Vechi_CodP Marfuri.CodP%TYPE;
    v_Nou_CodP Marfuri.CodP%TYPE;
    v_Index CHAR(1);
    v_TipIndex VARCHAR(20);
    v_NrIndex NUMBER(2,0);
    v_TotalInreg NUMBER(15,0);
    CURSOR c_TotalInreg IS
      SELECT count(*)
      FROM Marfuri;
  CURSOR c_NrInreg IS
      SELECT count(*)
      FROM Marfuri
      WHERE CodP=v_Vechi_CodP;
  BEGIN
```

```
v_NrOp:=&NrOp;
OPEN c_TotalInreg;
FETCH c_TotalInreg INTO v_TotalInreg;
CLOSE c_TotalInreg;
v_Vechi_CodP:=&CodPVechi;
v_Nou_CodP:=&CodPNou;
v_Index:= '&Index';
v_TipIndex:= '&TipIndex';
v_NrIndex:=&NrIndex;
OPEN c_NrInreg
FETCH c_NrInreg INTO v_NrInreg;
CLOSE c_NrInreg;
v_Operatie:='UPDATE Marfuri SET CodP= WHERE CodP=';
v_Start:=DBMS_UTILITY.GET_TIME;
UPDATE Marfuri SET CodP=v_Nou_CodP WHERE CodP=v_Vechi_CodP;
v_End:=DBMS_UTILITY.GET_TIME;
v_Timp:=(v_End-v_Start)*10;
INSERT INTO Rezultate(NrOp, Operatie, TotalInreg, InregPrel,Timp, Valoare, Index, TipIndex,
      NrIndex) VALUES (v_NrOp, v_Operatie, v_TotalInreg, v_NrInreg, v_Timp,
      v_Vechi_CodP, v_Index, v_TipIndex, v_NrIndex);
COMMIT;
END;
```

--MarfuriIOTModificare2.sql

```
DECLARE
 v_NrOp NUMBER(3);
 v_Operatie VARCHAR(200);
 v_Start NUMBER(10,0);
 v_End NUMBER(10,0);
 v_Timp NUMBER(10,0);
 v_NrInreg NUMBER(12,0);
 v_Grupa Marfuri.Grupa%TYPE;
 v_CodP VARCHAR(50);
 v_Index CHAR(1);
 v_TipIndex VARCHAR(20);
 v_NrIndex NUMBER(2,0);
 v_TotalInreg NUMBER(15,0);
 CURSOR c_TotalInreg IS
  SELECT count(*)
  FROM Marfuri;
```

```
CURSOR c_NrInreg IS
  SELECT count(*)
  FROM Marfuri
  WHERE CodP IN v_CodP;
BEGIN
 v_NrOp:=&NrOp;
 OPEN c_TotalInreg;
 FETCH c_TotalInreg INTO v_TotalInreg;
 CLOSE c_TotalInreg;
 v_Grupa:=&Grupa;
 v_CodP:=&CodP;
 v_Index:= '&Index';
 v_TipIndex:= '&TipIndex';
 v_NrIndex:=&NrIndex;
 OPEN c_NrInreg
 FETCH c_NrInreg INTO v_NrInreg;
 CLOSE c_NrInreg;
 v_Operatie:='UPDATE Marfuri SET Grupa= WHERE CodP IN ';
 v_Start:=DBMS_UTILITY.GET_TIME;
 UPDATE Marfuri SET Grupa=v_Grupa WHERE CodP IN v_CodP;
 v_End:=DBMS_UTILITY.GET_TIME;
 v_Timp:=(v_End-v_Start)*10;
        INSERT INTO Rezultate(NrOp, Operatie, TotalInreg, InregPrel,Timp, Valoare, Index, TipIndex,
              NrIndex) VALUES (v_NrOp, v_Operatie, v_TotalInreg, v_NrInreg, v_Timp,
              v_CodP, v_Index, v_TipIndex, v_NrIndex);
        COMMIT;
       END;
```

/*procedura stocată FurnizoriInsert1*/

```
       CREATE PROCEDURE dbo.FurnizoriInsert1
       (      @nrOp NUMERIC(18,0), @vTar_Furn VARCHAR(15), @index CHAR(1), @tipIndex
VARCHAR(20), @nrIndex NUMERIC(2,0)
       )
       AS
       SET NOCOUNT ON
       BEGIN TRANSACTION
              DECLARE @op VARCHAR(200),@t1 NUMERIC(18,0),@t2 NUMERIC(18,0),@s1
INT,@s2 INT,@total NUMERIC(18,0),@inreg NUMERIC(18,0),@dif NUMERIC(18,6)
              SET @op = 'INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE Tar_Furn='
              SELECT @total=count(*) FROM Furnizori
```

```
        SELECT @inreg = count(*) FROM Furnizori_Noi WHERE Tar_Furn=@vTar_Furn
SELECT @t1=datepart(ms,getdate()),@s1=datepart(s,getdate())
        INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE Tar_Furn=@vTar_Furn
SELECT @t2=datepart(ms,getdate()),@s2=datepart(s,getdate())
        if @s2>@s1
                SET @dif=(@s2*1000+@t2)-(@s1*1000+@t1)
        else
                SET @dif=@t2-@t1
        INSERT Rezultate VALUES
                (@nr,@op,@total,@inreg,@dif,@vTar_Furn,@index,@tipIndex,@nrIndex)
if @@error<>0
BEGIN
        ROLLBACK TRANSACTION
        RAISERROR ('Operatiunea nu s-a putut realiza din cauza unei erori!!',16,1) with NOWAIT
END
ELSE
BEGIN
        COMMIT TRANSACTION
        RETURN(0)
END
GO
```

```
/*procedura stocată FurnizoriInsertCompus*/
        CREATE PROCEDURE dbo.FurnizoriInsertCompus
        (       @nrOp NUMERIC(18,0), @vTar_Furn1 VARCHAR(15), @vTar_Furn2 VARCHAR(15),
@index CHAR(1), @tipIndex VARCHAR(20), @nrIndex NUMERIC(2,0)
        )
        AS
        SET NOCOUNT ON
        BEGIN TRANSACTION
                DECLARE @op VARCHAR(200),@t1 NUMERIC(18,0),@t2 NUMERIC(18,0),@s1
INT,@s2 INT,@total NUMERIC(18,0),@inreg NUMERIC(18,0),@dif NUMERIC(18,6)
                SET @op = 'INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE Tar_Furn=
OR Tar_Furn='
                SELECT @total=count(*) FROM Furnizori
                SELECT @inreg = count(*) FROM Furnizori_Noi WHERE Tar_Furn=@vTar_Furn1 OR
Tar_Furn=@vTar_Furn2
                SELECT @t1=datepart(ms,getdate()),@s1=datepart(s,getdate())
                INSERT INTO Furnizori SELECT * FROM Furnizori_Noi WHERE
Tar_Furn=@vTar_Furn1 OR Tar_Furn=@vTar_Furn2
```

```
        SELECT @t2=datepart(ms,getdate()),@s2=datepart(s,getdate())
        if @s2>@s1
                SET @dif=(@s2*1000+@t2)-(@s1*1000+@t1)
        else
                SET @dif=@t2-@t1
        SET @vTar_Furn1=@vTar_Furn1+' OR '+@vTar_Furn2
        INSERT Rezultate VALUES
                (@nr,@op,@total,@inreg,@dif,@vTar_Furn1,@index,@tipIndex,@nrIndex)
if @@error<>0
BEGIN
        ROLLBACK TRANSACTION
        RAISERROR ('Operatiunea nu s-a putut realiza din cauza unei erori!!',16,1) with NOWAIT
END
ELSE
BEGIN
        COMMIT TRANSACTION
        RETURN(0)
END
GO
```

/*procedura stocată MarfuriInsert1*/

```
        CREATE PROCEDURE dbo.MarfuriInsert1
        (       @nrOp NUMERIC(18,0), @vGrupa VARCHAR(8), @index CHAR(1), @tipIndex
VARCHAR(20), @nrIndex NUMERIC(2,0)
        )
        AS
        SET NOCOUNT ON
        BEGIN TRANSACTION
                DECLARE @op VARCHAR(200),@t1 NUMERIC(18,0),@t2 NUMERIC(18,0),@s1
INT,@s2 INT,@total NUMERIC(18,0),@inreg NUMERIC(18,0),@dif NUMERIC(18,6)
                SET @op = 'INSERT INTO Marfuri SELECT * FROM Marfuri_Noi WHERE Grupa='
                SELECT @total=count(*) FROM Marfuri
                SELECT @inreg = count(*) FROM Marfuri_Noi WHERE Grupa=@vGrupa
                SELECT @t1=datepart(ms,getdate()),@s1=datepart(s,getdate())
                INSERT INTO Marfuri SELECT * FROM Marfuri_Noi WHERE Grupa=@vGrupa
        SELECT @t2=datepart(ms,getdate()),@s2=datepart(s,getdate())
                if @s2>@s1
                        SET @dif=(@s2*1000+@t2)-(@s1*1000+@t1)
                else
                        SET @dif=@t2-@t1
```

```
        INSERT Rezultate VALUES
                (@nr,@op,@total,@inreg,@dif,@vGrupa,@index,@tipIndex,@nrIndex)
    if @@error<>0
    BEGIN
        ROLLBACK TRANSACTION
        RAISERROR ('Operatiunea nu s-a putut realiza din cauza unei erori!!',16,1) with NOWAIT
    END
    ELSE
    BEGIN
        COMMIT TRANSACTION
        RETURN(0)
    END
    GO
```

/*procedura stocată MarfuriIOTSelect2*/

```
    CREATE PROCEDURE dbo.MarfuriIOTSelect2
    (   @nrOp NUMERIC(18,0), @vCodP VARCHAR(50), @index CHAR(1), @tipIndex
VARCHAR(20), @nrIndex NUMERIC(2,0)
    )
    AS
    SET NOCOUNT ON
    BEGIN TRANSACTION
        DECLARE @op VARCHAR(200),@t1 NUMERIC(18,0),@t2 NUMERIC(18,0),@s1
INT,@s2 INT,@total NUMERIC(18,0),@inreg NUMERIC(18,0),@dif NUMERIC(18,6)
        SET @op = 'SELECT * FROM Marfuri WHERE CodP IN '
        SELECT @total=count(*) FROM Marfuri
        SELECT @inreg = count(*)  FROM Marfuri WHERE CodP IN @vCodP
        SELECT @t1=datepart(ms,getdate()),@s1=datepart(s,getdate())
        SELECT *  FROM Marfuri WHERE CodP IN @vCodP
        SELECT @t2=datepart(ms,getdate()),@s2=datepart(s,getdate())
        if @s2>@s1
            SET @dif=(@s2*1000+@t2)-(@s1*1000+@t1)
        else
            SET @dif=@t2-@t1
        INSERT Rezultate VALUES
            (@nr,@op,@total,@inreg,@dif,@vCodP,@index,@tipIndex,@nrIndex)
    if @@error<>0
    BEGIN
        ROLLBACK TRANSACTION
        RAISERROR ('Operatiunea nu s-a putut realiza din cauza unei erori!!',16,1) with NOWAIT
```

```
        END
        ELSE
        BEGIN
                COMMIT TRANSACTION
                RETURN(0)
        END
GO
```

/*procedura stocată MarfuriIOTSelect3*/

```
        CREATE PROCEDURE dbo.MarfuriIOTSelect3
        (       @nrOp NUMERIC(18,0), @vGrupa VARCHAR(8), @index CHAR(1), @tipIndex
VARCHAR(20), @nrIndex NUMERIC(2,0)
        )
        AS
        SET NOCOUNT ON
        BEGIN TRANSACTION
                DECLARE @op VARCHAR(200),@t1 NUMERIC(18,0),@t2 NUMERIC(18,0),@s1
INT,@s2 INT,@total NUMERIC(18,0),@inreg NUMERIC(18,0),@dif NUMERIC(18,6)
                SET @op = 'SELECT * FROM Marfuri WHERE Grupa='
                SELECT @total=count(*) FROM Marfuri
                SELECT @inreg = count(*)  FROM Marfuri WHERE Grupa=@vGrupa
                SELECT @t1=datepart(ms,getdate()),@s1=datepart(s,getdate())
                SELECT *  FROM Marfuri WHERE Grupa=@vGrupa
                SELECT @t2=datepart(ms,getdate()),@s2=datepart(s,getdate())
                if @s2>@s1
                        SET @dif=(@s2*1000+@t2)-(@s1*1000+@t1)
                else
                        SET @dif=@t2-@t1
                INSERT Rezultate VALUES
                        (@nr,@op,@total,@inreg,@dif,@vGrupa,@index,@tipIndex,@nrIndex)
if @@error<>0
BEGIN
        ROLLBACK TRANSACTION
        RAISERROR ('Operatiunea nu s-a putut realiza din cauza unei erori!!',16,1) with NOWAIT
END
ELSE
BEGIN
        COMMIT TRANSACTION
        RETURN(0)
END
```

```
GO

/*procedura stocată MarfuriIOTModificare1*/
        CREATE PROCEDURE dbo.MarfuriIOTModificare1
        (       @nrOp NUMERIC(18,0), @vVechiCodP VARCHAR(10), @vNouCodP VARCHAR(10),
@index CHAR(1), @tipIndex VARCHAR(20), @nrIndex NUMERIC(2,0)
        )
        AS
        SET NOCOUNT ON
        BEGIN TRANSACTION
                DECLARE @op VARCHAR(200),@t1 NUMERIC(18,0),@t2 NUMERIC(18,0),@s1
INT,@s2 INT,@total NUMERIC(18,0),@inreg NUMERIC(18,0),@dif NUMERIC(18,6)
                SET @op = 'UPDATE Marfuri SET CodP= WHERE CodP='
                SELECT @total=count(*) FROM Marfuri
                SELECT @t1=datepart(ms,getdate()),@s1=datepart(s,getdate())
                UPDATE Marfuri SET CodP=@vNouCodP WHERE Grupa=@vVechiCodP
                SELECT @t2=datepart(ms,getdate()),@s2=datepart(s,getdate())
                if @s2>@s1
                        SET @dif=(@s2*1000+@t2)-(@s1*1000+@t1)
                else
                        SET @dif=@t2-@t1
                SELECT @inreg = count(*) FROM Marfuri WHERE CodP=@vNouCodP
                INSERT Rezultate VALUES
                        (@nr,@op,@total,@inreg,@dif,@vVechiCodP,@index,@tipIndex,@nrIndex)
        if @@error<>0
        BEGIN
                ROLLBACK TRANSACTION
                RAISERROR ('Operatiunea nu s-a putut realiza din cauza unei erori!!',16,1) with NOWAIT
        END
        ELSE
        BEGIN
                COMMIT TRANSACTION
                RETURN(0)
        END
GO

/*procedura stocată MarfuriIOTModificare2*/
        CREATE PROCEDURE dbo.MarfuriIOTModificare2
        (       @nrOp NUMERIC(18,0), @vGrupa VARCHAR(8), @vCodP VARCHAR(50), @index
CHAR(1), @tipIndex VARCHAR(20), @nrIndex NUMERIC(2,0)
```

```
)
AS
SET NOCOUNT ON
BEGIN TRANSACTION
        DECLARE @op VARCHAR(200),@t1 NUMERIC(18,0),@t2 NUMERIC(18,0),@s1
INT,@s2 INT,@total NUMERIC(18,0),@inreg NUMERIC(18,0),@dif NUMERIC(18,6)
        SET @op = 'UPDATE Marfuri SET Grupa= WHERE CodP IN '
        SELECT @total=count(*) FROM Marfuri
        SELECT @t1=datepart(ms,getdate()),@s1=datepart(s,getdate())
        UPDATE Marfuri SET Grupa=@vGrupa WHERE CodP IN @vCodP
        SELECT @t2=datepart(ms,getdate()),@s2=datepart(s,getdate())
        if @s2>@s1
                SET @dif=(@s2*1000+@t2)-(@s1*1000+@t1)
        else
                SET @dif=@t2-@t1
        SELECT @inreg = count(*)  FROM  Marfuri WHERE CodP IN @vCodP
        INSERT Rezultate VALUES
                (@nr,@op,@total,@inreg,@dif,@vCodP,@index,@tipIndex,@nrIndex)
if @@error<>0
BEGIN
        ROLLBACK TRANSACTION
        RAISERROR ('Operatiunea nu s-a putut realiza din cauza unei erori!!',16,1) with NOWAIT
END
ELSE
BEGIN
        COMMIT TRANSACTION
        RETURN(0)
END
GO
```

ANEXA 2

2.1. *Structura tabelelor bazei de date repository*

RELATII			
		Ins	numeric 10,0
		Upd	numeric 10,0
Relatie	varchar 50	Del	numeric 10,0
DurataViata	numeric 4,0	Sel	numeric 10,0
CheiePrimara	varchar 200	NT	numeric 12,0
MarMedieRand	numeric 10,0	PR	numeric 18,6

LR	numeric	1,0
Tip	char	2
NRC	numeric	10,0
PCR	numeric	18,6
Partitionabila	numeric	1,0
LR1	numeric	1,0
Iactiv	numeric	9,4
LRI	numeric	1,0
ITipActiv	numeric	9,4
CheiePartitionare	varchar	300
TipFisier	varchar	30
Dimensiune	numeric	18,3
Tupluri	numeric	12,0
TipPartitie	varchar	20

ATRIBUTE_REL

Relatie	varchar	50
Atribut	varchar	30
TipData	varchar	15
Lung	numeric	22,2

TRANZACTII

IdTranz	char	4
Nume	varchar	30
Descriere	varchar	250
Frecv	numeric	10,4
UMFrecv	char	4
FrecvAn	numeric	18,0
PondereFrecv	numeric	2,0
FT	numeric	18,6
OreMIN1	char	5
OreMAX1	char	5
OreMIN2	char	5
OreMAX2	char	5
OreMIN3	char	5
OreMAX3	char	5
PondereConc	numeric	2,0
TimpExec	numeric	2,0
UMTimp	char	3
TimpExecMs	numeric	6,0
PondereTimp	numeric	2,0
PF	numeric	18,6
PC	numeric	18,6
PTn	numeric	18,6
PT	numeric	18,6
LCT	bit	1

OP_TRANZ

IdTranz	char	4
Relatie	varchar	50
Ins	numeric	2,0
Upd	numeric	2,0
Del	numeric	2,0
Sel	numeric	2,0

TRANZ_CRITICE

IdTranz	char	4
Ins	numeric	3,0
Upd	numeric	3,0
Del	numeric	3,0
Sel	numeric	3,0
PAC	numeric	10,6
FO	numeric	18,4
LT1	bit	1

ATRIBUTE_ROSII

Relatia	varchar	50
Cheie	varchar	900
Contor	numeric	10,0

ATRIBUTE_VERZI

Relatia	varchar	50
Cheie	varchar	900
Contor	numeric	10,0
Marime	int	4
Contor_rosii	numeric	10,0

CHEI_INDEXARE

Relatia	varchar	50
CheieIndexare	varchar	900
MarimeCheie	int	4
TipIndex	varchar	50
Structura	char	6
Dimensiune	numeric	9,2
NumeIndex	varchar	30

CHEI_PARTITIONARE

Relatia	varchar	50
Cheie	varchar	300
Contor	numeric	10,0
L	numeric	1,0

JOIN1

Relatia1	varchar	50
Relatia2	varchar	50
Contor	int	4
Operatii	int	4

REDUNDANTA

Relatie	varchar	50
Iactiv	numeric	9,4

CONVERSIE

Id	int	4
Um1	char	4
Um2	char	4
Val	numeric	18,0
Spor	bit	1

INTERVAL_PUNCTAJ

Interv	int	4

T

IdTranz	char	4
Nr	int	4
DifNr	int	4
Frecv	numeric	13,7
OreMIN1	numeric	18,0
OreMAX1	numeric	18,0
NrMin	numeric	18,0
V2	numeric	18,6

2.2. *Programele sursă ale aplicației de asistare a procesului de proiectare fizică a bazei de date*

program Doctorat;

```
uses
  Forms,
  PreluareRelatii in 'PreluareRelatii.pas' {Form1},
  PreluareTranzactii in 'PreluareTranzactii.pas' {Form2},
  prima in 'prima.pas' {Form3},
  Analiza1 in 'Analiza1.pas' {Form4},
  IdRelCrit in 'IdRelCrit.pas' {Form5},
  IdRelPartit in 'IdRelPartit.pas' {Form6},
  IdStructAuxAcces in 'IdStructAuxAcces.pas' {Form7},
  AlegOrgFisiere in 'AlegOrgFisiere.pas' {Form8},
  AlStructIndex in 'AlStructIndex.pas' {Form9},
  AnalIntrRedundControl in 'AnalIntrRedundControl.pas' {Form10},
  EstSpatMem in 'EstSpatMem.pas' {Form11},
  SchFizBD in 'SchFizBD.pas' {Form12},
  Temporara in 'Temporara.pas' {Form13};

{$R *.res}

begin
  Application.Initialize;
  {var
    interv: integer;}
  Application.CreateForm(TForm3, Form3);
  Application.CreateForm(TForm1, Form1);
  Application.CreateForm(TForm2, Form2);
  Application.CreateForm(TForm4, Form4);
  Application.CreateForm(TForm5, Form5);
  Application.CreateForm(TForm6, Form6);
  Application.CreateForm(TForm7, Form7);
  Application.CreateForm(TForm8, Form8);
  Application.CreateForm(TForm9, Form9);
  Application.CreateForm(TForm10, Form10);
  Application.CreateForm(TForm11, Form11);
  Application.CreateForm(TForm12, Form12);
  Application.CreateForm(TForm13, Form13);
  Application.Run;
end.
```

unit PreluareRelatii;

```
  { Preluare date initiale relatii }

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls, DB, DBTables, dblookup, DBCtrls, Grids,
  DBGrids;

type
  TForm1 = class(TForm)
    p1: TPanel;
    Edit1: TEdit;
    Label1: TLabel;
    Edit2: TEdit;
    Label2: TLabel;
    Edit3: TEdit;
    Label3: TLabel;
    Edit4: TEdit;
    Label4: TLabel;
    p2: TPanel;
    Edit5: TEdit;
    Label6: TLabel;
    Edit6: TEdit;
    Label7: TLabel;
    Edit7: TEdit;
    Label8: TLabel;
    b2: TButton;
    b3: TButton;
    qRel: TQuery;
    b4: TButton;
    DataSource1: TDataSource;
    TAtr: TTable;
    l: TLabel;
    dg1: TDBGrid;
    dg2: TDBGrid;
    DataSource2: TDataSource;
    rg1: TRadioGroup;
    e1: TEdit;
    b1: TButton;
    TRel: TTable;
    b5: TButton;
    b6: TButton;
    b7: TButton;
    spSterg: TStoredProc;
    b8: TButton;
    spEditare: TStoredProc;
    Label5: TLabel;
    Label9: TLabel;
    Label10: TLabel;
    Button1: TButton;
    procedure FormCreate(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure b3Click(Sender: TObject);
    procedure b2Click(Sender: TObject);
    procedure b4Click(Sender: TObject);
    procedure rg1Click(Sender: TObject);
    procedure b1Click(Sender: TObject);
    procedure b6Click(Sender: TObject);
    procedure b5Click(Sender: TObject);
```

```
    procedure dg2CellClick(Column: TColumn);
    procedure b7Click(Sender: TObject);
    procedure StAtr();
    procedure StRel();
    procedure b8Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  contor: integer;


implementation

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
begin
    qRel.Open;
    TAtr.Open;
    contor:=0;
end;

procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  qRel.Close;
  TAtr.Close;
end;

procedure TForm1.b3Click(Sender: TObject);
begin
  p1.Enabled:=true;
  Edit1.Text:='';
  Edit2.Text:='';
  Edit3.Text:='';
  Edit4.Text:='';
end;

procedure TForm1.b2Click(Sender: TObject);
begin
if b2.Caption<>'Salvare editare' then
begin
  if (Edit1.Text='') or (Edit2.Text='') or (Edit3.Text='') or (Edit4.Text='')
     or (Edit5.Text='') or (Edit6.Text='') or (Edit7.Text='') then
  begin
    showmessage('Pentru a efectua aceasta operatiune, toate datele cerute trebuie completate!');
  end
  else
     try
       if contor = 0 then
       begin
         if not TRel.Active then
           TRel.Open;
         if TRel.Locate('relatie',Edit1.Text,[loPartialKey]) then
         begin
```

```
        showmessage('Relatia inserata trebuie sa fie unica!');
        exit;
       end
       else
        TRel.AppendRecord([Edit1.text,StrToInt(Edit2.Text),Edit4.Text,StrToInt(Edit3.Text),0,0,0,0]);
       TRel.Close;
      end;
      TAtr.AppendRecord([Edit1.text,Edit5.Text,Edit6.Text,StrToInt(Edit7.Text)]);
      p1.Enabled:=false;
      contor:=contor+1;
      if qRel.Active then
       qRel.Close;
      qRel.Open;
      qRel.Locate('relatie',Edit1.Text,[loPartialKey]);
     except
      on E: Exception do
      showmessage('Eroare: '+ E.Message);
     end;
     Edit5.Text:='';
     Edit6.Text:='';
     Edit7.Text:='';
     b3.Enabled:=true;
end
else
begin
  if (qRel.FieldByName('PR').IsNull) or (MessageDlg('Aceasta operatiune duce la pierderea datelor analizei
efectuate pentru aceasta relatie!Continuati?!', mtInformation, [mbYes, mbNo], 0) = mrYes) then
begin
  try
    spEditare.ParamByName('@rel').AsString := trim(Edit1.Text);
    spEditare.ParamByName('@dur').AsFloat := StrToFloat(Edit2.Text);
    spEditare.ParamByName('@che').AsString := trim(Edit4.Text);
    spEditare.ParamByName('@mar').AsFloat := StrToFloat(Edit3.Text);
    spEditare.ParamByName('@r').AsString := trim(qRel.Fields[0].AsString);
    TAtr.Edit;
    TAtr.FieldByName('Atribut').AsString:= trim(Edit5.Text);
    TAtr.FieldByName('TipData').AsString:= trim(Edit6.Text);
    TAtr.FieldByName('Lungime').AsFloat:= StrToFloat(Edit7.Text);
    TAtr.Post;
    qRel.Close;
    spEditare.Prepare;
    spEditare.ExecProc;
    qRel.Open;
    qRel.Locate('relatie',Edit1.Text,[loPartialKey]);
  except
    on E: Exception do
       showmessage('Operatiunea nu s-a putut efectua din cauza urmatoarei erori: '+#13#10+ E.Message);
  end;
end;
    Edit1.Text:='';
    Edit2.Text:='';
    Edit3.Text:='';
    Edit4.Text:='';
    Edit5.Text:='';
    Edit6.Text:='';
    Edit7.Text:='';
    b2.Caption:='Atribut nou';
    b7.Visible:=false;
    button1.Visible:=false;
    b3.Enabled:=true;
```

```
end;
end;

procedure TForm1.b4Click(Sender: TObject);
begin
  form1.Close;
end;

procedure TForm1.rg1Click(Sender: TObject);
begin
  If rg1.ItemIndex =1 then
  begin
    l.Visible:=true;
    b1.Visible:=true;
    e1.Visible:=true;
    e1.Text:='';
  end
  else
  begin
    l.Visible:=false;
    b1.Visible:=false;
    e1.Visible:=false;
    qRel.Close;
    qRel.SQL.Clear;
    qRel.SQL.Add('select Relatie,DurataViata,MarMedieRand,CheiePrimara,PR from relatii');
    qRel.Open;
  end;
end;

procedure TForm1.b1Click(Sender: TObject);
begin
    try
      qRel.Close;
      qRel.SQL.Clear;
      qRel.SQL.Add('select Relatie,DurataViata,MarMedieRand,CheiePrimara,PR from relatii where
Relatie='''+trim(e1.Text)+'''');
      qRel.Open;
    except
    end;
end;

procedure TForm1.b6Click(Sender: TObject);
begin
  if (qRel.FieldByName('PR').IsNull) then
    StRel()
  else
  begin
    if MessageDlg('Aceasta operatiune duce la pierderea datelor analizei efectuate pentru aceasta
relatie!Continuati?!', mtInformation, [mbYes, mbNo], 0) = mrYes then
      StRel;
  end;
end;
procedure TForm1.StRel();
begin
  try
    spSterg.ParamByName('@r').AsString := trim(qRel.Fields[0].AsString);
    spSterg.ParamByName('@a').AsString := '#';
    qRel.Close;
    spSterg.Prepare;
    spSterg.ExecProc;
```

```
   qRel.Open;
  except
   on E: Exception do
      showmessage('Operatiunea nu s-a putut efectua din cauza urmatoarei erori: '+#13#10+ E.Message);
  end;
end;
procedure TForm1.b5Click(Sender: TObject);
begin
  p1.Enabled:=true;
  Edit1.Text:=qRel.Fields[0].AsString;
  Edit2.Text:=qRel.Fields[1].AsString;
  Edit3.Text:=qRel.Fields[2].AsString;
  Edit4.Text:=qRel.Fields[3].AsString;
  Edit5.Text:=TAtr.Fields[1].AsString;
  Edit6.Text:=TAtr.Fields[2].AsString;
  Edit7.Text:=TAtr.Fields[3].AsString;
  b2.Caption:='Salvare editare';
  b7.Visible:=true;
  button1.Visible:=true;
end;

procedure TForm1.dg2CellClick(Column: TColumn);
begin
  if (b2.Caption='Salvare editare') and (qRel.Fields[0].AsString = Edit1.Text) then
  begin
    Edit5.Text:=TAtr.Fields[1].AsString;
    Edit6.Text:=TAtr.Fields[2].AsString;
    Edit7.Text:=TAtr.Fields[3].AsString;
  end;
end;

procedure TForm1.b7Click(Sender: TObject);
begin
  Edit1.Text:='';
  Edit2.Text:='';
  Edit3.Text:='';
  Edit4.Text:='';
  Edit5.Text:='';
  Edit6.Text:='';
  Edit7.Text:='';
  b2.Caption:='Atribut nou';
  b7.Visible:=false;
  button1.Visible:=false;
end;

procedure TForm1.b8Click(Sender: TObject);
begin
  if (qRel.FieldByName('PR').IsNull) then
    StAtr()
  else
  begin
    if MessageDlg('Aceasta operatiune duce la pierderea datelor analizei efectuate pentru aceasta
relatie!Continuati?!', mtInformation, [mbYes, mbNo], 0) = mrYes then
      StAtr;
  end;
end;

procedure TForm1.StAtr();
var
  rel: string;
```

```
begin
  rel:=qRel.Fields[0].AsString;
  if dg2.DataSource.DataSet.RecordCount = 1 then
  begin
    if MessageDlg('Stergerea acestui atribut va determina stergerea intregii relatii!', mtInformation, [mbYes,
mbNo], 0) = mrYes then
      b6.Click;
  end
  else
    try
      spSterg.ParamByName('@r').AsString := trim(qRel.Fields[0].AsString);
      spSterg.ParamByName('@a').AsString := trim(TAtr.Fields[1].AsString);
      qRel.Close;
      spSterg.Prepare;
      spSterg.ExecProc;
      qRel.Open;
      qRel.Locate('relatie',rel,[loPartialKey]);
    except
      on E: Exception do
        showmessage('Operatiunea nu s-a putut efectua din cauza urmatoarei erori: '+#13#10+ E.Message);
    end;
  end;

procedure TForm1.Button1Click(Sender: TObject);
begin
if button1.Caption='Atribut nou' then
begin
  Edit5.Text:='';
  Edit6.Text:='';
  Edit7.Text:='';
  b2.Enabled:=false;
  button1.Caption:='Salvare atribute';
end
else
begin
  if (Edit5.Text='') or (Edit6.Text='') or (Edit7.Text='') then
  begin
    showmessage('Pentru a efectua aceasta operatiune, toate datele cerute referitoare la atribute trebuie
completate!');
    exit;
  end
  else
  begin
    TAtr.AppendRecord([Edit1.text,Edit5.Text,Edit6.Text,StrToInt(Edit7.Text)]);
    if qRel.Active then
      qRel.Close;
    qRel.Open;
    qRel.Locate('relatie',Edit1.Text,[loPartialKey]);
    b2.Caption:='Atribut nou';
    b7.Visible:=false;
    button1.Visible:=false;
    b3.Enabled:=true;
    b2.Enabled:=true;
    button1.Caption:='Atribut nou';
    Edit1.Text:='';
    Edit2.Text:='';
    Edit3.Text:='';
    Edit4.Text:='';
  end;
end;
```

```
  Edit5.Text:='';
  edit6.Text:='';
  edit7.Text:='';
end;
end.

unit PreluareTranzactii;
   { Preluare date initiale tranzactii }

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  DateUtils,Dialogs, DBTables, DB, StdCtrls, ExtCtrls, Grids, DBGrids;

type
  TForm2 = class(TForm)
    l: TLabel;
    p1: TPanel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    b3: TButton;
    b4: TButton;
    dg1: TDBGrid;
    rg1: TRadioGroup;
    e1: TEdit;
    b1: TButton;
    b5: TButton;
    b6: TButton;
    qTranz: TQuery;
    DataSource1: TDataSource;
    TTranz: TTable;
    spSalvTranz: TStoredProc;
    Label5: TLabel;
    Edit6: TEdit;
    Label6: TLabel;
    Edit7: TEdit;
    Edit8: TEdit;
    Edit9: TEdit;
    Label7: TLabel;
    Label8: TLabel;
    Label9: TLabel;
    Edit10: TEdit;
    Edit11: TEdit;
    Edit12: TEdit;
    Label10: TLabel;
    Label11: TLabel;
    Label12: TLabel;
    Edit13: TEdit;
    Edit14: TEdit;
    Label13: TLabel;
    Label14: TLabel;
    Button1: TButton;
    Edit15: TEdit;
    Label15: TLabel;
    cb1: TComboBox;
```

```
    Label16: TLabel;
    cb2: TComboBox;
    qSt: TQuery;
    Edit4: TEdit;
    Label17: TLabel;
    Label18: TLabel;
    Label1: TLabel;
    Label19: TLabel;
    procedure FormCreate(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure Button1Click(Sender: TObject);
    procedure b3Click(Sender: TObject);
    procedure b4Click(Sender: TObject);
    function ora(s1: string):boolean;
    procedure rg1Click(Sender: TObject);
    procedure b1Click(Sender: TObject);
    procedure b5Click(Sender: TObject);
    procedure b6Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form2: TForm2;
  h1,h2,m1,m2,contor: integer;

implementation

{$R *.dfm}

procedure TForm2.FormCreate(Sender: TObject);
begin
  qTranz.Open;
end;

procedure TForm2.FormClose(Sender: TObject; var Action: TCloseAction);
begin
    qTranz.Close;
end;

procedure TForm2.Button1Click(Sender: TObject);
begin
  Form2.Close;
end;

function TForm2.ora(s1: string):boolean;
var
 h,m,s: string;
 i,dp,c : integer;
begin
  dp:=0;
  c:=1;
  s:=':';
  for i:=0 to length(s1)-1 do
  begin
    if s1[i]=s then
    begin
      if c=1 then
```

```
    begin
      h:=copy(s1,0,i-1);
      dp:=i+1;
    end;
    c:=c+1;
  end;
  if i=length(s1)-1 then
  begin
    c:= Length(s1)-dp+1;
    m:=copy(s1,dp,c);
  end;
end;
result:=true;
if (strtoint(h)<24) and (strtoint(m)<60) and (length(m)>1)
    and (length(m)<3) and (length(h)<3) then
begin
  if contor =1 then
  begin
    contor:=2;
    h2:= strtoint(h);
    m2:=strtoint(m);
  end
  else
  begin
    contor:=1;
    h1:= strtoint(h);
    m1:=strtoint(m);
  end;
  if contor=2 then
  begin
    if h2>h1 then
      result:=true
    else
    begin
      if (h2=h1) and (m2 > m1) then
        result:=true
      else
        result:=false;
    end;
   end;
 end
 else
    result :=false;
end;

procedure TForm2.b3Click(Sender: TObject);
var
  s: string;
begin
  if (Edit1.Text='') or (Edit2.Text='') or (Edit3.Text='') or (Edit4.Text='')
     or (Edit6.Text='') or (Edit7.Text='') or(Edit8.Text='')
     or (Edit13.Text='') or (Edit14.Text='') or (Edit15.Text='') then
  begin
    showmessage('Pentru a efectua aceasta operatiune, trebuie completate toate datele cerute (*)!');
  end
  else
  begin
     s:= copy(cb1.Text,0,3);
     contor:=0;
     try
```

```
  if (ora(Edit7.text)) and (ora(Edit8.text)) then
  begin
   if (edit9.Text <>'') or (edit10.Text<>'') then
   begin
    if not (ora(Edit9.text)) or not (ora(Edit10.text)) then
    begin
     ShowMessage('Valorile pentru intervalele orare sunt inserate gresit!');
     exit;
    end;
   end;
   if (edit11.Text <>'') or (edit12.Text<>'') then
   begin
    if not (ora(Edit11.text)) or not (ora(Edit12.text)) then
    begin
     ShowMessage('Valorile pentru intervalele orare sunt inserate gresit!');
     exit;
    end;
   end;
  end
  else
  begin
   ShowMessage('Valorile pentru intervalele orare sunt inserate gresit!');
   exit;
  end;
 except
  on E: Exception do
  begin
   ShowMessage('Valorile pentru intervalele orare sunt inserate gresit!');
   exit;
  end;
 end;
if (qtranz.FieldByName('PF').IsNull) or (MessageDlg('Aceasta operatiune duce la pierderea datelor analizei
efectuate pentru aceasta tranzactie!Continuati?!', mtInformation, [mbYes, mbNo], 0) = mrYes) then
  begin
  if (StrToInt(edit6.Text)+StrToInt(edit13.Text)+StrToInt(edit15.Text))=100 then
  begin
   spSalvTranz.ParamByName('@id').AsString := trim(Edit1.text);
   spSalvTranz.ParamByName('@num').AsString := trim(Edit2.text);
   spSalvTranz.ParamByName('@des').AsString := trim(Edit3.text);
   spSalvTranz.ParamByName('@fre').AsInteger := StrToInt(Edit4.text);
   spSalvTranz.ParamByName('@umf').AsString := s;
   spSalvTranz.ParamByName('@pof').AsInteger := StrToInt(Edit6.text);
   spSalvTranz.ParamByName('@mi1').AsString := Edit7.Text;
   spSalvTranz.ParamByName('@ma1').AsString := Edit8.Text;
   spSalvTranz.ParamByName('@mi2').AsString := Edit9.Text;
   spSalvTranz.ParamByName('@ma2').AsString := Edit10.Text;
   spSalvTranz.ParamByName('@mi3').AsString := Edit11.Text;
   spSalvTranz.ParamByName('@ma3').AsString := Edit12.Text;
   spSalvTranz.ParamByName('@poc').AsInteger := StrToInt(Edit13.text);
   spSalvTranz.ParamByName('@tex').AsInteger := StrToInt(Edit14.text);
   spSalvTranz.ParamByName('@umt').AsString := cb2.Text;
   spSalvTranz.ParamByName('@pot').AsInteger := StrToInt(Edit15.text);
   if b3.Caption='Salvare tranzactie' then
    spSalvTranz.ParamByName('@op').AsInteger := 0
   else
    spSalvTranz.ParamByName('@op').AsInteger := 1;
   try
    qTranz.Close;
    spSalvTranz.Prepare;
    spSalvTranz.ExecProc;
```

```
    qTranz.Open;
    qTranz.Locate('idtranz',Edit1.Text,[loPartialkey]);
    {qTranz.Close;
    qTranz.Open;}
   except
       on e:exception do
       begin
           showmessage(e.Message);
           qTranz.Open;
           exit;
       end;
   end;
  end
  else
  begin
   showmessage('EROARE!'+#13#10+'Suma ponderilor trebuie sa fie 100%!');
   exit;
  end;
  end;
Edit1.Text:='';
Edit2.Text:='';
Edit3.Text:='';
Edit4.Text:='';
Edit6.Text:='';
Edit7.Text:='';
Edit8.Text:='';
Edit9.Text:='';
Edit10.Text:='';
Edit11.Text:='';
Edit12.Text:='';
Edit13.Text:='';
Edit14.Text:='';
Edit15.Text:='';
b3.Caption:='Salvare tranzactie';
b4.Caption:='Anulare tranzactie';
end;

procedure TForm2.b4Click(Sender: TObject);
begin
Edit1.Text:='';
Edit2.Text:='';
Edit3.Text:='';
Edit4.Text:='';
Edit6.Text:='';
Edit7.Text:='';
Edit8.Text:='';
Edit9.Text:='';
Edit10.Text:='';
Edit11.Text:='';
Edit12.Text:='';
Edit13.Text:='';
Edit14.Text:='';
Edit15.Text:='';
cb1.Text:='';
cb2.Text:='';
b3.Caption := 'Salvare tranzactie';
b4.Caption:='Anulare tranzactie';
end;
```

```
procedure TForm2.rg1Click(Sender: TObject);
begin
 If rg1.ItemIndex =1 then
 begin
  l.Visible:=true;
  b1.Visible:=true;
  e1.Visible:=true;
  e1.Text:='';
 end
 else
 begin
  l.Visible:=false;
  b1.Visible:=false;
  e1.Visible:=false;
  qTranz.Close;
  qTranz.SQL.Clear;
  qTranz.SQL.Add('select IdTranz,Nume,Descriere,Frecv,UMFrecv, ');
  qTranz.SQL.Add('PondereFrecv,OreMin1,OreMax1,OreMin2,OreMax2,OreMin3,');
  qTranz.SQL.Add('OreMax3, PondereConc,TimpExec,PondereTimp,UMTimp,PF');
  qTranz.SQL.Add( 'from tranzactii');
  qTranz.Open;
 end;
end;

procedure TForm2.b1Click(Sender: TObject);
begin
 try
  qTranz.Close;
  qTranz.SQL.Clear;
  qTranz.SQL.Add('select IdTranz,Nume,Descriere,Frecv,UMFrecv, ');
  qTranz.SQL.Add('PondereFrecv,OreMin1,OreMax1,OreMin2,OreMax2,OreMin3,');
  qTranz.SQL.Add('OreMax3, PondereConc,TimpExec,PondereTimp,UMTimp,PF');
  qTranz.SQL.Add( 'from tranzactii where IdTranz = '''+trim(e1.Text)+'''');
  qTranz.Open;
 except
 end;
end;

procedure TForm2.b5Click(Sender: TObject);
begin
 Edit1.Text:=qTranz.Fields[0].AsString;
 Edit2.Text:=qTranz.Fields[1].AsString;
 Edit3.Text:=qTranz.Fields[2].AsString;
 Edit4.Text:=qTranz.Fields[3].AsString;
 cb1.Text:=qTranz.Fields[4].AsString;
 Edit6.Text:=qTranz.Fields[5].AsString;
 Edit7.Text:=qTranz.Fields[6].AsString;
 Edit8.Text:=qTranz.Fields[7].AsString;
 Edit9.Text:=qTranz.Fields[8].AsString;
 Edit10.Text:=qTranz.Fields[9].AsString;
 Edit11.Text:=qTranz.Fields[10].AsString;
 Edit12.Text:=qTranz.Fields[11].AsString;
 Edit13.Text:=qTranz.Fields[12].AsString;
 cb2.Text:=qTranz.Fields[15].AsString;
 Edit14.Text:=qTranz.Fields[13].AsString;
 Edit15.Text:=qTranz.Fields[14].AsString;
 b3.Caption:='Salvare editare';
 b4.Caption:='Anulare editare';
end;
```

```
procedure TForm2.b6Click(Sender: TObject);
begin
 if (qtranz.FieldByName('PF').IsNull) or (MessageDlg('Aceasta operatiune duce la pierderea datelor analizei
efectuate pentru aceasta tranzactie!Continuati?!', mtInformation, [mbYes, mbNo], 0) = mrYes) then
 begin
    qSt.Close;
    qSt.SQL.Clear;
    qSt.SQL.Add('delete from tranzactii where IdTranz = '''+qtranz.Fields[0].AsString+'''');
    qTranz.Close;
    qSt.ExecSQL;
    qTranz.Open;
 end;
 Edit1.Text:='';
 Edit2.Text:='';
 Edit3.Text:='';
 Edit4.Text:='';
 Edit6.Text:='';
 Edit7.Text:='';
 Edit8.Text:='';
 Edit9.Text:='';
 Edit10.Text:='';
 Edit11.Text:='';
 Edit12.Text:='';
 Edit13.Text:='';
 Edit14.Text:='';
 Edit15.Text:='';
 cb1.Text:='';
 cb2.Text:='';
 b3.Caption := 'Salvare tranzactie';
 b4.Caption:='Anulare tranzactie';
end;

end.
```

```
unit Analiza1;
   { Identificare tranzactii critice }

interface
uses
 Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
 Dialogs, ExtCtrls, StdCtrls, DBCtrls, DB, DBTables, Grids, DBGrids,IdRelCrit;
type
 TForm4 = class(TForm)
   gb3: TGroupBox;
   Label6: TLabel;
   Label7: TLabel;
   chb1: TCheckBox;
   chb2: TCheckBox;
   chb3: TCheckBox;
   chb4: TCheckBox;
   b2: TButton;
   e6: TEdit;
   p1: TPanel;
   gb2: TGroupBox;
   Label9: TLabel;
   Label10: TLabel;
   e3: TEdit;
   cb1: TComboBox;
   gb1: TGroupBox;
   Label11: TLabel;
```

```
Label12: TLabel;
e1: TEdit;
e2: TEdit;
b1: TButton;
b5: TButton;
TRel: TTable;
qTr: TQuery;
spP1: TStoredProc;
DataSource1: TDataSource;
Memo1: TMemo;
Label11: TLabel;
chb5: TCheckBox;
Label2: TLabel;
gb4: TGroupBox;
Label3: TLabel;
Edit1: TEdit;
Button1: TButton;
g1: TDBGrid;
lcb1: TDBLookupComboBox;
lcb2: TDBLookupComboBox;
spOp: TStoredProc;
spPac: TStoredProc;
spLt: TStoredProc;
qLista: TQuery;
DataSource2: TDataSource;
Label4: TLabel;
gb5: TGroupBox;
e4: TEdit;
Label5: TLabel;
b3: TButton;
Label8: TLabel;
spPr: TStoredProc;
  Button2: TButton;
  Edit2: TEdit;
  procedure b1Click(Sender: TObject);
  procedure FormCreate(Sender: TObject);
  procedure FormClose(Sender: TObject; var Action: TCloseAction);
  procedure chb5Click(Sender: TObject);
  procedure b2Click(Sender: TObject);
  procedure Button1Click(Sender: TObject);
  procedure b5Click(Sender: TObject);
  procedure b3Click(Sender: TObject);
  procedure Button2Click(Sender: TObject);
 private
  { Private declarations }
 public
  { Public declarations }
 end;

var
 Form4: TForm4;
 c,interv: integer;

implementation

{$R *.dfm}

procedure TForm4.b1Click(Sender: TObject);
begin
If (e1.text='') or (e2.text='') or (e3.text='') then
```

```
 showmessage('Eroare! Inserati toate datele cerute')
else
begin
 if (StrToInt(e2.Text)- StrToInt(e1.Text)>0) then
 begin
  interv:=StrToInt(e2.Text)- StrToInt(e1.Text);
  spP1.ParamByName('@intervPunct').AsInteger:=interv;
  spP1.ParamByName('@umt').AsString:=copy(cb1.Text,0,3);
  spP1.ParamByName('@v').AsInteger:= StrToInt(e3.Text);
  try
   spP1.Prepare;
   spP1.ExecProc;
   label8.Caption:='['+spP1.ParamByName('@min').AsString+','+spP1.ParamByName('@max').AsString+']';
  except
   on e: Exception do Showmessage('Operatiunea nu s-a putut efectua din cauza urmatoarei erori:'+E.Message);
  end;
 end
 else
  ShowMessage('Eroare!Maxim trebuie sa fie mai mare decat minim!')
end;
end;

procedure TForm4.FormCreate(Sender: TObject);
begin
 Trel.Open;
 c:=0;
end;

procedure TForm4.FormClose(Sender: TObject; var Action: TCloseAction);
begin
 TRel.Close;
 qLista.Close;
 qTr.Close;
end;

procedure TForm4.chb5Click(Sender: TObject);
begin
 if chb5.Checked then
 begin
  label2.Visible:=True;
  lcb2.Visible:=true;
 end
 else
 begin
  label2.Visible:=false;
  lcb2.Visible:=false;
 end;
end;

procedure TForm4.b2Click(Sender: TObject);
var
 i,u,d,s,j: integer;
begin
 if chb1.Checked then
 begin
  chb1.Checked:=false;
  i:=1;
 end
 else
  i:=0;
```

```
if chb2.Checked then
begin
  chb2.Checked:=false;
  u:=1;
end
else
  u:=0;
if chb3.Checked then
begin
  chb3.Checked:=false;
  d:=1;
end
else
  d:=0;
if chb4.Checked then
begin
  chb4.Checked:=false;
  s:=1;
end
else
  s:=0;
if chb5.Checked then
begin
  chb5.Checked:=false;
  j:=1;
  spOp.ParamByName('@r2').AsString:= lcb2.Text
end
else
begin
  j:=0;
  spOp.ParamByName('@r2').AsString:='';
end;
  spOp.ParamByName('@id').AsString:= qtr.Fields[0].AsString;
  spOp.ParamByName('@r1').AsString:= lcb1.Text;
  spOp.ParamByName('@i').AsInteger:= i;
  spOp.ParamByName('@u').AsInteger:= u;
  spOp.ParamByName('@d').AsInteger:= d;
  spOp.ParamByName('@s').AsInteger:= s;
  spOp.ParamByName('@j').AsInteger:= j;
if c=0 then
begin
try
  spOp.Prepare;
  spOp.ExecProc;
except
  on e: Exception do Showmessage(E.Message);
end;
  qtr.Next;
  c:=1;
  if not qTr.Eof then
  begin
    e6.Text := qtr.Fields[1].AsString;
    Edit2.Text := qtr.Fields[0].AsString;
    memo1.Text := qtr.Fields[2].AsString;
  end
  else
  begin
    gb3.enabled := false;
    b2.Enabled:=false;
    gb4.Enabled:=true;
```

```
 end;
end
else
begin
try
 spOp.Prepare;
 spOp.ExecProc;
except
 on e: Exception do Showmessage(E.Message);
end;
 qtr.Next;
 if not qTr.Eof then
 begin
  e6.Text := qtr.Fields[1].AsString;
  Edit2.Text := qtr.Fields[0].AsString;
  memo1.Text := qtr.Fields[2].AsString;
 end
 else
 begin
  gb3.enabled := false;
  b2.Enabled:=false;
  spPac.Prepare;
  spPac.ExecProc;
  gb4.Enabled:=true;
  label4.Caption:='['+spPac.ParamByName('@mi').AsString+','+spPac.ParamByName('@ma').AsString+']';
 end;
end;
end;

procedure TForm4.Button1Click(Sender: TObject);
begin
 if edit1.Text='' then
  showmessage('Inserati punctajul prag al activitatii concurente!')
 else
 begin
  spLt.ParamByName('@prag').AsFloat:=strToFloat(Edit1.Text);
  try
   spLt.Prepare;
   spLt.ExecProc;
  except
   on e:exception do showmessage('Eroare:'+ #13#10 + e.message);
   end;
  gb4.Enabled:=true;
  qLista.Close;
  qLista.Open;
  Edit1.Enabled:=false;
  button1.Enabled:=false;
  b5.Enabled:=true;
 end;
end;

procedure TForm4.b5Click(Sender: TObject);
begin
 Form4.Close;
 Form5.Show;
end;

procedure TForm4.b3Click(Sender: TObject);
begin
if (e4.text<>'')then
```

```
begin
   spPr.ParamByName('@ppt').AsInteger:=StrToInt(e4.Text);
   spPr.Prepare;
   spPr.ExecProc;
   p1.Enabled:=false;
   b1.Enabled:=false;
   gb3.Enabled:=true;
   qtr.Open;
   qtr.First;
   e6.Text := qtr.Fields[1].AsString;
   Edit2.Text := qtr.Fields[0].AsString;
   memo1.Text := qtr.Fields[2].AsString;
end
else
  showmessage('Eroare! Inserati punctajul prag al tranzactiei!');
end;

procedure TForm4.Button2Click(Sender: TObject);
var
  i,u,d,s,j: integer;
begin
  if chb1.Checked then
  begin
    chb1.Checked:=false;
    i:=1;
  end
  else
    i:=0;
  if chb2.Checked then
  begin
    chb2.Checked:=false;
    u:=1;
  end
  else
    u:=0;
if chb3.Checked then
begin
  chb3.Checked:=false;
  d:=1;
end
else
  d:=0;
if chb4.Checked then
begin
  chb4.Checked:=false;
  s:=1;
end
else
  s:=0;
if chb5.Checked then
begin
  chb5.Checked:=false;
  j:=1;
  spOp.ParamByName('@r2').AsString:= lcb2.Text
end
else
begin
  j:=0;
  spOp.ParamByName('@r2').AsString:='';
end;
```

```
 spOp.ParamByName('@id').AsString:= qtr.Fields[0].AsString;
 spOp.ParamByName('@r1').AsString:= lcb1.Text;
 spOp.ParamByName('@i').AsInteger:= i;
 spOp.ParamByName('@u').AsInteger:= u;
 spOp.ParamByName('@d').AsInteger:= d;
 spOp.ParamByName('@s').AsInteger:= s;
 spOp.ParamByName('@j').AsInteger:= j;
 try
    spOp.Prepare;
    spOp.ExecProc;
 except
   on e: Exception do Showmessage(E.Message);
 end;
end;

end.

unit IdRelCrit;
   { Identificare relatii critice }

interface

uses
 Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
 Dialogs, StdCtrls, Grids, DBGrids, DB, DBTables,IdRelPartit;

type
 TForm5 = class(TForm)
   gb1: TGroupBox;
   Label1: TLabel;
   Edit1: TEdit;
   Button1: TButton;
   DBGrid1: TDBGrid;
   Label2: TLabel;
   GroupBox2: TGroupBox;
   DBGrid2: TDBGrid;
   b5: TButton;
   spLR: TStoredProc;
   qRc1: TQuery;
   DataSource1: TDataSource;
   qEstimare: TQuery;
   DataSource2: TDataSource;
   Label3: TLabel;
   spMaxPR: TStoredProc;
   procedure Button1Click(Sender: TObject);
   procedure FormClose(Sender: TObject; var Action: TCloseAction);
   procedure b5Click(Sender: TObject);
   procedure FormShow(Sender: TObject);
 private
   { Private declarations }
 public
   { Public declarations }
 end;

var
 Form5: TForm5;

implementation

{$R *.dfm}
```

```
procedure TForm5.Button1Click(Sender: TObject);
begin
 if edit1.Text='' then
  showmessage('Eroare! Inserati punctajul prag al relatiei!')
 else
 begin
  spLr.ParamByName('@prag').AsFloat := StrToFloat(Edit1.text);
  spLr.Prepare;
  spLr.ExecProc;
  qRc1.Open;
  qEstimare.Open;
  b5.Enabled:=true;
  button1.Enabled:=false;
  Edit1.Enabled:=false;
 end;
end;

procedure TForm5.FormClose(Sender: TObject; var Action: TCloseAction);
begin
 qRc1.Close;
 qEstimare.Close;
end;

procedure TForm5.b5Click(Sender: TObject);
begin
 form5.Close;
 form6.Show;
end;

procedure TForm5.FormShow(Sender: TObject);
begin
 spMaxPR.Prepare;
 spMaxPR.ExecProc;
label3.Caption:='['+spMaxPR.ParamByName('@min').AsString+','+spMaxPR.ParamByName('@max').AsString
+']';
end;

end.

unit IdRelPartit;
   { Identificare relatii partitionate }

interface

uses
 Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
 Dialogs, StdCtrls, Grids, DBGrids, DBTables, DB,IdStructAuxAcces;

type
 TForm6 = class(TForm)
  GroupBox1: TGroupBox;
  Label1: TLabel;
  Label2: TLabel;
  Label3: TLabel;
  e1: TEdit;
  e2: TEdit;
  e3: TEdit;
  Button1: TButton;
  g1: TDBGrid;
```

```
    gb2: TGroupBox;
    Label4: TLabel;
    e4: TEdit;
    e5: TEdit;
    Label5: TLabel;
    l1: TLabel;
    l4: TLabel;
    l5: TLabel;
    g2: TDBGrid;
    g5: TDBGrid;
    g6: TDBGrid;
    Button2: TButton;
    gb3: TGroupBox;
    b5: TButton;
    g7: TDBGrid;
    l2: TLabel;
    g3: TDBGrid;
    l3: TLabel;
    g4: TDBGrid;
    qRP: TQuery;
    DataSource1: TDataSource;
    spRP: TStoredProc;
    qAtr: TQuery;
    qTr: TQuery;
    DataSource2: TDataSource;
    qAfis: TQuery;
    DataSource3: TDataSource;
    spCheie: TStoredProc;
    spLista: TStoredProc;
    spMax: TStoredProc;
    Label6: TLabel;
    Button3: TButton;
    spRp1: TStoredProc;
    Label7: TLabel;
    Label8: TLabel;
    qUpd: TQuery;
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure Button2Click(Sender: TObject);
    procedure IncarcaAtribute;
    procedure InserareChei;
    procedure gr(g: TDBGrid);
    procedure FormCreate(Sender: TObject);
    procedure b5Click(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form6: TForm6;
  sl : TStringList;
  contor: integer;
  s: string;

implementation
```

```
{$R *.dfm}

procedure TForm6.IncarcaAtribute;
begin
  qAtr.Close;
  qAtr.SQL.Clear;
  qAtr.SQL.Add('select a.atribut,o.sel,o.upd,o.del from');
  qAtr.SQL.Add(' atribute_rel a inner join op_tranz o on a.relatie=o.relatie');
  qAtr.SQL.Add(' where a.relatie='"+qRP.Fields[0].AsString+'" and o.idtranz='"+qtr.Fields[0].AsString+'"');
  qAtr.Open;
  if qAtr.Fields[1].AsInteger=1 then
  begin
    g2.Visible:=true;
    l1.Visible:=true;
    g3.Visible:=true;
    l2.Visible:=true;
    g4.Visible:=true;
    l3.Visible:=true;
  end
  else
  begin
    g2.Visible:=false;
    l1.Visible:=false;
    g3.Visible:=false;
    l2.Visible:=false;
    g4.Visible:=false;
    l3.Visible:=false;
  end;
  if qAtr.Fields[2].AsInteger=1 then
  begin
    g5.Visible:=true;
    l4.Visible:=true;
  end
  else
  begin
    g5.Visible:=false;
    l4.Visible:=false;
  end;
  if qAtr.Fields[3].AsInteger=1 then
  begin
    g6.Visible:=true;
    l5.Visible:=true;
  end
  else
  begin
    g6.Visible:=false;
    l5.Visible:=false;
  end;
end;

procedure TForm6.FormClose(Sender: TObject; var Action: TCloseAction);
begin
    qRP.Close;
    qAfis.Close;
    sl.Free;
end;

procedure TForm6.gr(g: TDBGrid);
var
 i,j,index: Integer;
```

```
begin
 if g.SelectedRows.Count = 0 then
    exit;
 if sl.Count>0 then
  j:=sl.Count
 else
  j:=0;
 if g.SelectedRows.Count>0 then
 begin
  with g.DataSource.DataSet do
   for i:=0 to g.SelectedRows.Count-1 do
   begin
    GotoBookmark(pointer(g.SelectedRows.Items[i]));
    sl.Add(Fields[0].AsString);
   end;
   for i:=j to sl.Count-1 do
   begin
    if i=0 then
     s:=sl[0]
    else
     s:=s + '+'+ sl[i];
   end;
 end;
 spCheie.ParamByName('@r').AsString := qRP.Fields[0].AsString;
 spCheie.ParamByName('@ch').AsString := s;
 spCheie.ParamByName('@contor').AsInteger := contor;
 spCheie.Prepare;
 spCheie.ExecProc;
 sl.Clear;
end;

procedure TForm6.InserareChei;
begin
 if g2.Visible then
   gr(g2);
 if g3.Visible then
  gr(g3);
 if g4.Visible then
  gr(g4);
 if g5.Visible then
  gr(g5);
 if g6.Visible then
  gr(g6);
end;

procedure TForm6.Button2Click(Sender: TObject);
begin
 if not qtr.Eof then
 begin
  InserareChei;
  qtr.Next;
  if qtr.Eof then
  begin
   if not qRP.Eof then
   begin
    qRP.Next;
    qTr.Close;
    qtr.SQL.Clear;
    qTr.SQL.Add('select t.idtranz,t.nume from tranzactii t inner join op_tranz o on t.idtranz=o.idtranz where
o.relatie='"+qRP.Fields[0].AsString+'");
```

```
      qTr.Open;
      qTr.First;
      IncarcaAtribute;
      e4.Text:=qRP.Fields[0].AsString;
      e5.Text:=qtr.Fields[1].AsString;
     end;
     if qRP.Eof then
     begin
      gb2.Enabled:=false;
      gb3.Enabled:=true;
      button2.Enabled:=false;
      spLista.Prepare;
      spLista.ExecProc;
      qAfis.Open;
      b5.Enabled:=true;
     end;
    end
    else
    begin
     e5.Text:=qtr.Fields[1].AsString;
     IncarcaAtribute;
    end;
  end;
end;

procedure TForm6.FormCreate(Sender: TObject);
begin
  sl:=TStringList.Create;
  sl.Sorted:=true;
  contor:=1;
end;

procedure TForm6.b5Click(Sender: TObject);
begin
  qUpd.ExecSQL;
  Form6.Close;
  Form7.Show;
end;

procedure TForm6.FormShow(Sender: TObject);
begin
  spMax.Prepare;
  spMax.ExecProc;
  label6.Caption:='['+
spMax.paramByName('@min1').AsString+','+spMax.paramByName('@max1').AsString+']';
  label8.Caption:='['+
spMax.paramByName('@min2').AsString+','+spMax.paramByName('@max2').AsString+']';
end;

procedure TForm6.Button3Click(Sender: TObject);
begin
  spRp.ParamByName('@PDM').AsFloat:=strtofloat(e1.Text);
  spRp.ParamByName('@PPC').AsFloat:=strtofloat(e2.Text);
  try
    spRp.Prepare;
    spRp.ExecProc;
    e3.Enabled:=true;
    button1.Enabled:=true;
    label7.Caption:='['+ spRp.paramByName('@mi').AsString+','+spRp.paramByName('@ma').AsString+']';
  except
```

```
   on e:exception do showmessage('Eroare:'+ #13#10 + e.message);
  end;
end;

procedure TForm6.Button1Click(Sender: TObject);
begin
  spRp1.ParamByName('@PCR').AsFloat:=strtofloat(e3.Text);
  try
    spRp1.prepare;
    spRp1.ExecProc;
    qRP.Open;
    qRp.First;
    e4.Text:=qRP.Fields[0].AsString;
    qTr.Close;
    qtr.SQL.Clear;
    qTr.SQL.Add('select t.idtranz,t.nume from tranzactii t inner join op_tranz o on t.idtranz=o.idtranz where
o.relatie='"+qRP.Fields[0].AsString+"');
    qTr.Open;
    qTr.First;
    e5.Text:=qtr.Fields[1].AsString;
    gb2.Enabled:=true;
    IncarcaAtribute;
  except
    on e:exception do showmessage('Eroare:'+ #13#10 + e.message);
  end;
end;

end.
```

```
unit IdStructAuxAcces;
   { Identificare structuri auxiliare de acces }
interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Grids, DBGrids, StdCtrls, DB, DBTables,AlegOrgFisiere;

type
  TForm7 = class(TForm)
    gb1: TGroupBox;
    Label1: TLabel;
    e1: TEdit;
    Button1: TButton;
    DBGrid1: TDBGrid;
    gb2: TGroupBox;
    Label4: TLabel;
    Label5: TLabel;
    e4: TEdit;
    e5: TEdit;
    Button2: TButton;
    gb3: TGroupBox;
    DBGrid5: TDBGrid;
    b5: TButton;
    GroupBox4: TGroupBox;
    l1: TLabel;
    l2: TLabel;
    l3: TLabel;
    g2: TDBGrid;
    g3: TDBGrid;
```

```
    g4: TDBGrid;
    GroupBox5: TGroupBox;
    l4: TLabel;
    g5: TDBGrid;
    Label8: TLabel;
    e7: TEdit;
    Button3: TButton;
    Button4: TButton;
    l11: TLabel;
    spMin: TStoredProc;
    qRI: TQuery;
    DataSource1: TDataSource;
    spListaI: TStoredProc;
    qTr: TQuery;
    qAtr: TQuery;
    DataSource2: TDataSource;
    spCheie: TStoredProc;
    TAtrRos: TTable;
    spChIn: TStoredProc;
    qChIndex: TQuery;
    DataSource3: TDataSource;
    qMin: TQuery;
    l: TLabel;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure IncarcaAtribute;
    procedure InserareChei;
    procedure gr(g: TDBGrid);
    procedure Button2Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure b5Click(Sender: TObject);
    procedure FormShow(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form7: TForm7;
  i,s: string;
  sl: TStringList;
  sl1: TStringList;
  contor,c_ros,v: integer;

implementation

{$R *.dfm}

procedure TForm7.FormCreate(Sender: TObject);
begin
  sl := TStringList.Create;
  sl.Sorted:=true;
  sl1 := TStringList.Create;
  sl1.Sorted := true;
  sl1.Duplicates:=dupIgnore;
  contor:=1;
  c_ros:=0;
```

```
  qAtr.Open;
end;

procedure TForm7.Button1Click(Sender: TObject);
begin
   spListaI.ParamByName('@m').AsFloat := StrToFloat(e1.Text);
   spListaI.Prepare;
   spListaI.ExecProc;
   qRI.Open;
   qRI.First;
   e4.Text:=qRI.Fields[0].AsString;
   qTr.Close;
   qtr.SQL.Clear;
   qTr.SQL.Add('select t.idtranz,t.nume from tranzactii t inner join op_tranz o on t.idtranz=o.idtranz where
o.relatie='"+qRI.Fields[0].AsString+"');
   qTr.Open;
   qTr.First;
   e5.Text:=qtr.Fields[1].AsString;
   gb2.Enabled:=true;
   IncarcaAtribute;
end;

procedure TForm7.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  TAtrRos.Close;
  qRI.Close;
  qTr.Close;
  qChIndex.Close;
  qAtr.Close;
  qMin.Close;
  sl.Free;
  sl1.Free;
end;
procedure TForm7.gr(g: TDBGrid);
var
  n,marime: Integer;
begin
  marime:=0;
  sl.Clear;
  if g.SelectedRows.Count>0 then
  begin
    with g.DataSource.DataSet do
      for n:=0 to g.SelectedRows.Count-1 do
      begin
        GotoBookmark(pointer(g.SelectedRows.Items[n]));
        sl.Add(Fields[0].AsString);
        marime:=marime+ Fields[5].AsInteger;
      end;
      for n:=0 to sl.Count-1 do
      begin
        if n=0 then
          s:=sl[0]
        else
          s:=s + '+'+ sl[n];
      end;
    spCheie.ParamByName('@r').AsString := qRI.Fields[0].AsString;
    spCheie.ParamByName('@ch').AsString := s;
    spCheie.ParamByName('@contor').AsInteger := 1;
    spCheie.ParamByName('@v').AsInteger := v;
    spCheie.ParamByName('@m').AsFloat := marime;
```

```
   spCheie.ParamByName('@id').AsString := qTr.Fields[0].AsString;
   spCheie.Prepare;
   spCheie.ExecProc;
   sl.Clear;
   s:='';
   contor:=1;
  end;
end;

procedure TForm7.IncarcaAtribute;
begin
   qAtr.Close;
   qAtr.SQL.Clear;
   qAtr.SQL.Add('select a.atribut,o.sel,o.upd,o.del,o.ins,a.lungime from');
   qAtr.SQL.Add(' atribute_rel a inner join op_tranz o on a.relatie=o.relatie');
   qAtr.SQL.Add(' where a.relatie='"+qRI.Fields[0].AsString+'" and o.idtranz='"+qtr.Fields[0].AsString+'"');
   qAtr.Open;
   if qAtr.Fields[1].AsInteger=1 then
   begin
     g2.Visible:=true;
     l1.Visible:=true;
     button4.Visible:=true;
   end
   else
   begin
     g2.Visible:=false;
     l1.Visible:=false;
     button4.Visible:=false;
   end;
   if qAtr.Fields[2].AsInteger=1 then
   begin
     g3.Visible:=true;
     l2.Visible:=true;
     g5.Visible:=true;
     l4.Visible:=true;
   end
   else
   begin
     g3.Visible:=false;
     l2.Visible:=false;
     g5.Visible:=false;
     l4.Visible:=false;
   end;
   if qAtr.Fields[3].AsInteger=1 then
   begin
     g4.Visible:=true;
     l3.Visible:=true;
   end
   else
   begin
     g4.Visible:=false;
     l3.Visible:=false;
   end;
end;

procedure TForm7.InserareChei;
begin
  if g2.Visible then
  begin
    v:=1;
```

```
  gr(g2);
 end;
 if g3.Visible then
 begin
  v:=1;
  gr(g3);
 end;
 if g4.Visible then
 begin
  v:=1;
  gr(g4);
 end;
 if g5.Visible then
 begin
  v:=0;
  gr(g5);
 end;
end;

procedure TForm7.Button2Click(Sender: TObject);
begin
 if not qtr.Eof then
 begin
  begin
   if (qAtr.Fields[3].AsInteger=1) and (qAtr.Fields[4].AsInteger=1) then
    c_ros:=c_ros+2
   else
    c_ros:=c_ros+1;
  end;
  InserareChei;
  qtr.Next;
  if qtr.Eof then
  begin
   if not qRI.Eof then
   begin
    spCheie.ParamByName('@r').AsString := qRI.Fields[0].AsString;
    spCheie.ParamByName('@ch').AsString := s;
    spCheie.ParamByName('@contor').AsInteger := c_ros;
    spCheie.ParamByName('@v').AsInteger := 2;
    spCheie.ParamByName('@m').AsFloat := 0;
    spCheie.ParamByName('@id').AsString := qTr.Fields[0].AsString;
    spCheie.Prepare;
    spCheie.ExecProc;
    c_ros:=0;
   qRI.Next;
   qTr.Close;
   qtr.SQL.Clear;
   qTr.SQL.Add('select t.idtranz,t.nume from tranzactii t inner join op_tranz o on t.idtranz=o.idtranz where
o.relatie="'+qRI.Fields[0].AsString+'"');
   qTr.Open;
   qTr.First;
   IncarcaAtribute;
   e4.Text:=qRI.Fields[0].AsString;
   e5.Text:=qtr.Fields[1].AsString;
  end;
  if qRI.Eof then
  begin
   gb2.Enabled:=false;
   gb3.Enabled:=true;
   button2.Enabled:=false;
```

```
      button4.Enabled:=false;
      qMin.Open;
      l.Caption:='['+qMin.Fields[0].AsString+','+qMin.Fields[1].AsString+']';
      b5.Enabled:=true;
     end;
   end
   else
   begin
    e5.Text:=qtr.Fields[1].AsString;
    IncarcaAtribute;
   end;
 end;
end;

procedure TForm7.Button4Click(Sender: TObject);
begin
 v:=1;
 gr(g2);
 g2.SelectedRows.Clear;
end;

procedure TForm7.Button3Click(Sender: TObject);
begin
 spChIn.ParamByName('@c').AsInteger:=StrToInt(e7.Text);
 spChIn.Prepare;
 spChIn.ExecProc;
 qChIndex.Open;
end;

procedure TForm7.b5Click(Sender: TObject);
begin
 Form7.Close;
 Form8.Show;
end;

procedure TForm7.FormShow(Sender: TObject);
begin
 spMin.Prepare;
 spMin.ExecProc;
 l11.Caption:='Dimensiunea minima este: '+spMin.ParamByName('@m').AsString+' Mo';
end;

end.

unit AlegOrgFisiere;
   { Alegerea organizarii fisierelor }

interface

uses
 Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
 Dialogs, Grids, DBGrids, StdCtrls, DB, DBTables,AlStructIndex;

type
 TForm8 = class(TForm)
   g1: TDBGrid;
   cb1: TComboBox;
   Button1: TButton;
   gb1: TGroupBox;
   Label1: TLabel;
```

```
  gb2: TGroupBox;
  DBGrid2: TDBGrid;
  b5: TButton;
  spTipFis: TStoredProc;
  qPart: TQuery;
  DataSource1: TDataSource;
  spTipPart: TStoredProc;
  qAfStruct: TQuery;
  DataSource2: TDataSource;
  procedure FormCreate(Sender: TObject);
  procedure Button1Click(Sender: TObject);
  procedure FormClose(Sender: TObject; var Action: TCloseAction);
  procedure b5Click(Sender: TObject);
  procedure FormShow(Sender: TObject);
 private
  { Private declarations }
 public
  { Public declarations }
 end;

var
 Form8: TForm8;

implementation

{$R *.dfm}

procedure TForm8.FormCreate(Sender: TObject);
begin
 {spTipFis.Prepare;
 spTipFis.ExecProc;
 qPart.Open;
 qPart.First;
  }
end;

procedure TForm8.Button1Click(Sender: TObject);
begin
 if not qPart.Eof then
 begin
  spTipPart.ParamByName('@r').AsString := qPart.Fields[0].AsString;
  spTipPart.ParamByName('@tp').AsString :=cb1.Text;
  spTipPart.Prepare;
  spTipPart.ExecProc;
  cb1.Text:='Hash';
  qPart.Next;
  if qPart.Eof then
  begin
   gb1.Enabled:=false;
   gb2.Enabled:=true;
   qAfStruct.Open;
   b5.Enabled:=true;
  end;
 end;
end;

procedure TForm8.FormClose(Sender: TObject; var Action: TCloseAction);
begin
 qAfStruct.Close;
 qPart.Close;
```

```
end;

procedure TForm8.b5Click(Sender: TObject);
begin
  Form8.Close;
end;

procedure TForm8.FormShow(Sender: TObject);
begin
  spTipFis.Prepare;
  spTipFis.ExecProc;
  qPart.Open;
  qPart.First;
end;

end.

unit AlStructIndex;
   { Alegerea structurii fisierelor index }

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Grids, DBGrids, DB, DBTables;

type
  TForm9 = class(TForm)
    gb1: TGroupBox;
    g1: TDBGrid;
    gb2: TGroupBox;
    g2: TDBGrid;
    b5: TButton;
    DataSource1: TDataSource;
    qValori: TQuery;
    Button1: TButton;
    Label1: TLabel;
    Label2: TLabel;
    e1: TEdit;
    e2: TEdit;
    qStruct: TQuery;
    DataSource2: TDataSource;
    spIndex: TStoredProc;
    procedure FormCreate(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure Button1Click(Sender: TObject);
    procedure b5Click(Sender: TObject);
    procedure FormActivate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form9: TForm9;

implementation

{$R *.dfm}
```

```
procedure TForm9.FormCreate(Sender: TObject);
begin
 qValori.Open;
 qValori.First;
end;

procedure TForm9.FormClose(Sender: TObject; var Action: TCloseAction);
begin
 qValori.Close;
end;

procedure TForm9.Button1Click(Sender: TObject);
begin
 if not qValori.Eof then
 begin
   spIndex.ParamByName('@nv').AsInteger := StrToInt(e1.Text);
   spIndex.ParamByName('@ni').AsString := e2.Text;
   spIndex.ParamByName('@tupl').AsFloat := qValori.Fields[2].AsFloat;
   spIndex.ParamByName('@ch').AsString := qValori.Fields[1].AsString;
   spIndex.ParamByName('@r').AsString := qValori.Fields[0].AsString;
   spIndex.Prepare;
   spIndex.ExecProc;
   qValori.Next;
   if qValori.Eof then
   begin
     gb1.Enabled:=false;
     button1.Enabled:=false;
     gb2.Enabled:=true;
     qStruct.Open;
     b5.Enabled:=true;
   end;
 end;
 e1.Text:='';
 e2.Text:='';
end;

procedure TForm9.b5Click(Sender: TObject);
begin
 Form9.Close;
end;

procedure TForm9.FormActivate(Sender: TObject);
begin
 qValori.Open;
 qValori.First;
end;

end.

unit AnalIntrRedundControl;
   { Analiza introducerii unei redundante controlate }

interface

uses
 Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
 Dialogs, StdCtrls, Grids, DBGrids, DBTables, DB;

type
```

```
 TForm10 = class(TForm)
  GroupBox1: TGroupBox;
  e1: TEdit;
  Label1: TLabel;
  GroupBox2: TGroupBox;
  DBGrid1: TDBGrid;
  b5: TButton;
  spDenormalizabile: TStoredProc;
  DataSource1: TDataSource;
  TRedund: TTable;
  Button1: TButton;
  Label2: TLabel;
  spMax: TStoredProc;
  procedure b5Click(Sender: TObject);
  procedure FormClose(Sender: TObject; var Action: TCloseAction);
  procedure Button1Click(Sender: TObject);
  procedure FormCreate(Sender: TObject);
 private
  { Private declarations }
 public
  { Public declarations }
 end;

var
 Form10: TForm10;

implementation

{$R *.dfm}

procedure TForm10.b5Click(Sender: TObject);
begin
 Form10.Close;
end;

procedure TForm10.FormClose(Sender: TObject; var Action: TCloseAction);
begin
 TRedund.Close;
end;

procedure TForm10.Button1Click(Sender: TObject);
begin
 TRedund.Close;
 spDenormalizabile.ParamByName('@p').AsFloat:=StrToFloat(e1.Text);
 spDenormalizabile.Prepare;
 spDenormalizabile.ExecProc;
 TRedund.Open;
end;

procedure TForm10.FormCreate(Sender: TObject);
begin
 spMax.Prepare;
 spMax.ExecProc;
 label2.Caption:='['+ spMax.paramByName('@mi').AsString+','+spMax.paramByName('@ma').AsString+']';
end;

end.
```

unit EstSpatMem;

```
  { Estimarea spatiului de memorare necesar bazei de date }
```

```
interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Grids, DBGrids, DB, DBTables;

type
  TForm11 = class(TForm)
    GroupBox1: TGroupBox;
    DBGrid1: TDBGrid;
    DBGrid2: TDBGrid;
    Label1: TLabel;
    Label2: TLabel;
    b5: TButton;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    l1: TLabel;
    l3: TLabel;
    Label8: TLabel;
    l2: TLabel;
    qDate: TQuery;
    DataSource1: TDataSource;
    DataSource2: TDataSource;
    TIndex: TTable;
    spDimensiune: TStoredProc;
    procedure FormCreate(Sender: TObject);
    procedure b5Click(Sender: TObject);
    procedure FormActivate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form11: TForm11;

implementation

{$R *.dfm}

procedure TForm11.FormCreate(Sender: TObject);
begin
  qDate.Open;
  qDate.First;
  TIndex.Open;
  spDimensiune.Prepare;
  spDimensiune.ExecProc;
  l1.Caption := FloatToStr(spDimensiune.ParamByName('@Ddate').AsFloat);
  l2.Caption := FloatToStr(spDimensiune.ParamByName('@Dindex').AsFloat);
  l3.Caption := FloatToStr(spDimensiune.ParamByName('@Dtot').AsFloat);
end;

procedure TForm11.b5Click(Sender: TObject);
begin
  Form11.Close;
end;
```

```
procedure TForm11.FormActivate(Sender: TObject);
begin
  qDate.Open;
  qDate.First;
  TIndex.Open;
end;

end.

unit SchFizBD;
   { Schema fizica a bazei de date }

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Grids, DBGrids, DB, DBTables;

type
  TForm12 = class(TForm)
    GroupBox1: TGroupBox;
    Label1: TLabel;
    Label2: TLabel;
    dg1: TDBGrid;
    dg2: TDBGrid;
    b5: TButton;
    Label3: TLabel;
    l1: TLabel;
    qDate: TQuery;
    DataSource1: TDataSource;
    Tindex: TTable;
    DataSource2: TDataSource;
    sp1: TStoredProc;
    procedure FormCreate(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure b5Click(Sender: TObject);
    procedure FormActivate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form12: TForm12;

implementation

{$R *.dfm}

procedure TForm12.FormCreate(Sender: TObject);
begin
  qDate.Open;
  Tindex.Open;
  sp1.Prepare;
  sp1.ExecProc;
  l1.Caption := sp1.parambyname('@Dtot').AsString;
end;

procedure TForm12.FormClose(Sender: TObject; var Action: TCloseAction);
```

```
begin
  qDate.Close;
  Tindex.Close;
end;

procedure TForm12.b5Click(Sender: TObject);
begin
  Form12.Close;
end;

procedure TForm12.FormActivate(Sender: TObject);
begin
  qDate.Open;
  Tindex.Open;
end;

end.
```

2.3. *Structura tabelelor bazei de date proiectate*

BANCA_FURNIZOR

COD_FURN	C	7
CONT_BANCA	C	22
DEN_BANCA	C	30

FURNIZORI

COD_FURN	C	7
CONT	C	11
DEN_FURN	C	30
LOC_FURN	C	20
ADR_FURN	C	30
TAR_FURN	C	15
NR_FISC	N	8
SOLD_CR_AN	N	14,2
RUL_DB_AN	N	14,2
RUL_CR_AN	N	14,2
RUL_DB_LUN	N	14,2
RUL_CR_LUN	N	14,2

ISTORIC_CUMPARARI

COD_FURN	C	7
FEL_DOC	C	1
DATA_DOC	D	8
NR_DOC	N	8
FEL_OPER	C	1
DATA_OPER	D	8
SUMA	N	15,2
SIMB_MON	C	4
CONT_PLATA	C	11
FEL_DOC_PL	C	3
NR_DOC_PL	N	8
CONTAB	L	1
DATA_CURS	D	8
DATA_SCAD	D	8

NC_FURNIZORI

DEBIT	C	11
CREDIT	C	11
VALOARE	N	15,2
FEL_DOC	C	3
NR_DOC	N	8
DATA_DOC	D	8
CONTAB	L	1
EXPLIC	C	45
SURSA_PROFITC	15	

GEST_MARFURI

COD_GEST	C	2
DEN_GEST	C	30
LOC_GEST	C	20
ADR_GEST	C	30
GESTIONAR	C	50

MARFURI

CODP	C	10
DENP	C	50
GRUPA	C	8
UM	C	3
PRET_ACHIZ	N	20,2
CONT	C	11
COTA_TVA	N	5,2
COD_FURN	C	7

MISC_MARFURI

COD	C	10
GEST	C	2
COD_OPER	C	2
CANTITATE	N	15,3
DATA_DOC	D	8
FEL_DOC	C	3
NR_DOC	N	8
PARTENER	C	7
FELDOCPAR	C	1
NRDOCPAR	N	8
DATADOCPAR	D	8
PRET_VANZ	N	20,2
COMANDA	N	10
COD_PRODUS	C	10
CANT_BRUT	N	15,3
SURSA_PROF	C	15

NC_GEST_MARFURI

DEBIT	C	11
CREDIT	C	11
VALOARE	N	15,2
FEL_DOC	C	3
NR_DOC	N	8
DATA_DOC	D	8
CONTAB	L	1
EXPLIC	C	45
SURSA_PROFITC	15	

STOC_MARFURI

COD	C	10
DEN	C	30
UM	C	3
GEST	C	2
PRET_ACHIZ	N	20,2
PRET_VANZ	N	20,2
STOC	N	15,3
STOC_BRUT	N	15,3
STOC_INIT	N	15,3
STOC_BR_IN	N	15,3
INT_AN	N	15,3
IES_AN	N	15,3
INT_LUN	N	15,3
IES_LUN	N	15,3
ULTIMA_INT	D	8
ULTIMA_IES	D	8

AGENTI

COD_AGENT	C	8
DEN_AGENT	C	20
DIVIZIE	C	10

CLIENTI

COD_CLIENT	C	7
CONT	C	11
DEN_CLIENT	C	30
LOC_CLIENT	C	20
ADR_CLIENT	C	30
TAR_CLIENT	C	15
NR_FISC	N	8
PLATIT_TVA	L	1
GRUPA	C	7
AUT_ALCOOL	C	20
AUT_TUTUN	C	20
AUT_CAFEA	C	20
BLOCAT	L	1
ZILE_SCAD	N	3
LIMITA_CR	N	15,2
SOLD_DB_AN	N	14,2
RUL_DB_AN	N	14,2
RUL_CR_AN	N	14,2
RUL_DB_LUN	N	14,2
RUL_CR_LUN	N	14,2

BANCA_CLIENT

COD_CLIENT	C	7
CONT_BANCA	C	22
DEN_BANCA	C	30

EMISE

NR_FACT	N	8
COD_FACT	C	3
TIP_RAND	C	1
SECV	N	1
COD_CLIENT	C	7
DATA_EMI	D	8
DEBIT	C	11
CREDIT	C	11
VALOARE	N	15,2
STARE	N	1
SCUTIT_TVA	L	1
COTA_TVA	N	5,2
CONTAB	L	1
SIMB_MON	C	4
VAL_VALUTA	N	15,2
SURSA_PROF	C	15,2

GRUPE_CLIENTI

COD_GRUPA	C	7
DEN_GRUPA	C	30
CARACTERISTICI	C	250

INCASARI

NR_FACT	N	8
DATA_EMI	D	8
DATA_INCAS	D	8
DEBIT	C	11
CREDIT	C	11
VALOARE	N	15,2
STARE	N	1
CONTAB	L	1
SIMB_MON	C	4

VAL_VALUTA	N	15,2
FEL_DOC_IN	C	3
NR_DOC_IN	N	8
SURSA_PROF	C	15

ISTORIC_VANZARI

COD_CLIENT	C	7
DATA_EMI	D	8
NR_FACT	N	8
FEL_OP	C	1
DATA_OP	D	8
SIMB_MON	C	4
DATA_CURS	D	8
SUMA	N	15,2
CONTAB	L	1
FEL_DOC_IN	C	3
NR_DOC_IN	N	8
DATA_SCAD	D	8
COD_AGENT	C	8
CONT_IN	C	11
COD_FACT	C	3

NC_CLIENTI

DEBIT	C	11
CREDIT	C	11
VALOARE	N	15,2
FEL_DOC	C	3
NR_DOC	N	8
DATA_DOC	D	8
CONTAB	L	1
EXPLIC	C	45
SURSA_PROFITC	15	

VANZARI

CODP	C	10
COD_FURN	C	7
COD_CLIENT	C	7
COD_AGENT	C	8
CODT	C	10

TIMPI

CODT	C	10
NUME_ZI	C	10
ZI_DIN_SAPT	N	1
ZI_DIN_LUNA	N	2
SAPT_DIN_LUNA	N	1
SAPT_DIN_AN	N	2
LUNA_DIN_AN	N	2
NUME_LUNA	C	10
TRIMESTRU	N	1
AN	N	4

2.4. *Lista parțială a cerințelor*

GESTIUNI

- întocmirea unui nomenclator de mărfuri– acesta va suporta operațiile:
  - vizualizare
  - adăugare
  - modificare
  - eliminare
- evidențierea gestiunilor existente – operații suportate:
  - vizualizare
  - adăugare
  - modificare
  - eliminare
- lista furnizorilor existenți
- lista clienților existenți
- intrările de mărfuri :
  - de la furnizori cu factură
  - de la furnizori fară factură
  - de la terți
  - de la subunități
  - din plus de inventar
- ieşirile de mărfuri:
  - prin consum de materiale
  - prin transfer la terți
  - prin transfer la subunități
  - prin transfer în obiecte de inventar în folosință
  - prin transfer în mărfuri en-detail în unitate

- prin minus de inventar
- prin perisabilități sau deprecieri
- prin protocol
- prin investiții
- prin casare obiecte de inventar

- generarea fişei de magazie
- calcularea stocurilor de mărfuri:
  - pe articole la preț de achiziție
  - pe articole la preț de vânzare
  - pe grupe de articole
  - pe grupe de articole şi pe preţuri
  - inventarul stocurilor de mărfuri
- afişarea pe ecran şi la imprimantă a intrărilor de mărfuri:
  - pe lună
  - pe articol
  - pe document
- afişarea pe ecran şi la imprimantă a ieşirilor de mărfuri:
  - pe lună
  - pe articol
  - pe cod document
  - pe cote TVA
  - pe magazin
- datele necesare despre gestiuni:
  - cod gestiune – unic
  - denumirea
  - localizare
  - gestionar
- datele necesare despre mărfuri:
  - cod marfă – unic
  - denumire
  - unitate de măsură
  - preț de achiziție
  - cota de tva
  - cod furnizor
  - grupa de produse din care face parte
  - contul
- datele necesare cu privire la operațiunile de mişcare a mărfurilor:
  - cod mărfă
  - cod gestiune
  - cod operație:
    - intrare
    - ieşire
    - retur
  - cantitate operată
  - dată document de intrare
  - fel document de intrare
  - număr document de intrare
  - cod partener
  - dată document de ieşire
  - fel document de ieşire
  - număr document de ieşire
  - preț de vânzare
- datele necesare notelor contabile care evidențiază operațiile de mişcare a mărfurilor:
  - cont de debit
  - cont de credit
  - valoarea operației
  - dată document
  - felul documentului ce stă la baza notei contabile
  - număr document
  - explicație operațiune efectuată

- sursă de profit
- contabilizată sau nu

- datele necesare pentru evidențierea stocului de mărfuri:
  - cod marfă
  - denumire
  - unitate de măsură
  - gestiune
  - preț de achiziție
  - preț de vânzare
  - stoc
  - stoc brut
  - stoc inițial
  - stoc brut inițial
  - intrări anuale
  - ieşiri anuale
  - intrări lunare
  - ieşiri lunare
  - data ultimei intrări
  - data ultimei ieşiri

CLIENTI

- emitere de facturi către clienți
- anulare factură client
- încasare factură client
- generarea Jurnalului de Vânzări
- borderoul cu vânzările cu amănuntul pe societăți sau pe gestiuni
- lista facturilor emise
- lista chitanțelor emise
- lista clienților:
  - fişa clientului
  - clienți debitori
  - facturi scadente
  - balanța clienților
  - lista tuturor clienților
  - vânzările către clienți:
    - dela .... până la ....
    - pe cont
    - pe furnizor:
      - cu total
      - fără total
    - pe agent:
      - cu total
      - fără total
    - pe client:
      - cu total
      - fără total
    - pe grupe de mărfuri:
      - cu total
      - fără total
    - pe articole:
      - cu total
      - fără total
    - pe gestiuni:
      - cu total
      - fără total
    - pe grupe clienți:
      - cu total
      - fără total
    - raport la preț mediu

  - raport centralizat
  - sortare valorică
- istoricul facturilor emise
- facturile emise
- facturile încasate
- datele necesare despre agenții de vânzări:
  - cod agent – unic
  - denumire
  - divizie
- datele necesare despre clienți:
  - cod client
  - denumire
  - localitate
  - adresă
  - tară/județ de origine
  - cod fiscal
  - plătitor tva
  - grupă client
  - autorizație alcool
  - autorizație tutun
  - autorizație cafea
  - zile scadență acordate
  - limita creditului acordat
  - client blocat sau nu
  - cont
  - sold debitor anual
  - rulaj debitor anual
  - rulaj creditor anual
  - rulaj debitor lunar
  - rulaj creditor lunar
- datele despre conturile bancare ale clienților:
  - cod client
  - cont bancar
  - banca
- datele despre facturile emise:
  - număr factură
  - cod factură
  - tip rând
  - secvența
  - cod client
  - data emiterii
  - cont de debit
  - cont de credit
  - valoarea operației
  - scutit tva
  - cota tva
  - simbol monetar
  - valoarea valutei
  - sursă de profit
  - contabilizată sau nu
- datele despre grupele de clienți:
  - cod grupă
  - denumirea grupă
  - caracteristici
- datele despre încasările de la clienți:
  - număr factură
  - data emiterii
  - data încasării
  - cont de debit
  - cont de credit

- valoarea operației
- simbol monetar
- valoarea valutei
- fel document încasare
- număr document încasare
- sursă de profit
- contabilizat sau nu

- datele istorice despre operațiile efectuate cu clienții:
  - cod client
  - fel operație:
    - intrare (încasare factură)
    - ieşire (emitere factură)
  - data operației
  - data emiterii
  - număr factură
  - cod factură
  - simbol monetar
  - dată curs valutar
  - suma
  - fel document încasare
  - număr document încasare
  - data scadenței
  - cod agent vânzări
  - cont încasare
  - contabilizat sau nu
- datele necesare notelor contabile care evidențiază operațiile efectuate cu clienții:
  - cont de debit
  - cont de credit
  - valoarea operației
  - dată document
  - felul documentului ce stă la baza notei contabile
  - număr document
  - explicație operațiune efectuată
  - sursă de profit
  - contabilizată sau nu

FURNIZORI

- înregistrare factură nouă
- stornare factură existentă
- plată a unei facturi
- stornare plată factură
- generarea Jurnalului de Cumpărări
- lista facturilor primite de la furnizori
- lista facturilor primite şi neachitate pe furnizori
- furnizori de plătit:
  - pe facturi
  - pe solduri
- plăți scadente către furnizori
- plăți întârziate către furnizori
- balanța furnizorilor
- note de recepție
- datele despre conturile bancare ale furnizorilor:
  - cod furnizor
  - cont bancar
  - banca
- datele necesare despre furnizori sunt:
  - cod furnizor – unic
  - denumire
  - localitate

- adresă
- țară/județ de origine
- cod fiscal
- cont
- sold creditor anual
- rulaj debitor anual
- rulaj creditor anual
- rulaj debitor lunar
- rulaj creditor lunar

- datele necesare despre operațiunile cu furnizorii sunt:
  - cod furnizor
  - fel operație:
    - cumpărare
    - plată
  - dată operație
  - sumă
  - simbol monetar
  - dată curs valutar
  - fel document
  - dată document
  - număr document
  - data scadenței
  - pentru plăți:
  - cont de plată
    - fel document plată
    - număr document plată
  - contabilizată sau nu
- datele notelor contabile care evidențiază în contabilitate operațiunile efectuate cu furnizorii:
  - cont de debit
  - cont de credit
  - valoare operațiune
  - felul documentului ce stă la baza notei contabile
  - număr document
  - dată document
  - explicația operațiunii efectuate
  - sursă de profit
  - contabilizată sau nu

ANALIZA VANZARILOR

- top clienți
- top solduri
- top vânzări/ agenți
- top vânzări/agenți/clienți
- top vânzări/agenți/divizii
- top produse - cantitativ
- top produse – valoric
- solduri/clienți
- solduri/termene
- solduri/agenți/divizii
- solduri/agenți/termene
- clienți/agenți/furnizori
- clienți/agenți/grupe
- clienți/agenți/produse
- vânzări/produse
- vânzări/furnizori
- vânzări/furnizori/grupe
- istoric vânzări/luni
- vânzări/divizii/luni

- incasări/divizii/agent

2.5. *Lista parțială a relațiilor şi a tranzacțiilor*

Relații de bază

Nume Relație	Atribute	Domeniu	Valoarea NULL	Valori Implicite	Atribut derivat	Cheie Primară	Chei Alternative	Chei Externe	Restricția Cheii Externe	DELETE pe PK	UPDATE pe PK	Durata de viață
Banca_Furnizor	Cod_Furn Cont_Banca Den_Banca	C(7) C(22) C(30)	Nu Nu Nu			Cod_Furn+Cont_Banca	Cont_Furn+Den+Banca	Cod_Furn	Furnizori(Cod_Furn)	Cascadă	Cascadă	5 ani
Furnizori	Cod_Furn Cont Den_Furn Loc_Furn Adr_Furn Tar_Furn Nr_Fisc Sold_Cr_An Rul_Db_An Rul_Cr_An Rul_Db_Lun Rul_Cr_Lun	C(7) C(11) C(30) C(20) C(30) C(15) N(8) N(14,2) N(14,2) N(14,2) N(14,2) N(14,2)	Nu Nu Nu Nu Nu Nu Nu Nu Nu	 0 0 0 0 0	Sold_Cr_An(Rul_Cr_An – Rul_Db_An)	Cod_Furn	Cont					5 ani
Istoric_Cumparari	Cod_Furn Fel_Doc Data_Doc Nr_Doc Fel_Oper Data_Oper Suma Simb_Mon Cont_Plata Fel_Doc_Pl Nr_Doc_Pl Contab Data_Curs Data_Scad	C(7) C(1) D N(8) Op_Furn D N(15,2) C(4) C(11) C(3) N(8) L D D	Nu Nu Nu Nu Nu Nu Nu Nu Nu Nu Nu Nu	 LEI F		Cod_Furn+Fel_Doc+Data_Doc+Nr_Doc+Fel_Oper+Fel_Doc_Pl+Nr_Doc_Pl		Cod_Furn	Furnizori(Cod_Furn	Interzis	Cascadă	1 an
Nc_Furnizori	Debit Credit Valoare Fel_Doc Nr_Doc Data_Doc Contab Explic Sursa_Profit	C(11) C(11) N(15,2) C(3) N(8) D L C(45) C(15)	Nu Nu Nu Nu Nu Nu Nu	 0 F		Debit+Credit+Fel_Doc+Nr_Doc+Data_Doc		Fel_Doc+Nr_Doc+Data_Doc	Istoric_Furnizori(Fel_Doc+Nr_Doc+Data_Doc)	Cascadă	Cascadă	1 an
Gest_Marfuri	Cod_Gest Den_Gest Loc_Gest Adr_Gest Gestionar	C(2) C(30) C(20) C(30) C(50)	Nu Nu			Cod_Gest						5 ani

Marfuri	CodP DenP Grupa Um Pret_Achiz Cont Cota_TVA Cod_Furn	C(10) C(50) C(8) Unit_Mas N(20,2) C(11) N(5,2) C(7)	Nu Nu Nu Nu Nu Nu	 BUC 0 0.19 		CodP		Cod_Furn	Furnizori(Cod_Furn)	Interzis	Cascadă	5 ani
Misc_Marfuri	Cod Gest Cod_Oper Cantitate Data_Doc Fel_Doc Nr_Doc Partener FelDocPar NrDocPar DataDocPar Pret_Vanz Comanda Cod_Produs Cant_Brut Sursa_Profit	C(10) C(2) Op_Misc N(15,3) D C(3) N(8) C(7) C(1) N(8) D N(20,2) N(10) C(10) N(15,3) C(15)	Nu Nu Nu Nu Nu Nu Nu Nu Nu Nu Nu Nu 	 0 		Cod+Gest+Data_Doc+Fel_Doc+Nr_Doc		Cod Gest Partener+FelDocPar+NrDocPar+DataDocPar	Marfuri(CodP) Gest_Marfuri(Cod_Gest) Istoric_Cumparari(Cod_Furn+Fel_Doc+Data_Doc+Nr_Doc) IstoricVanzari(Cod_Client+Data_Emi+Nr_Fact)	Interzis Interzis Interzis	Cascadă Cascadă Interzis	1 an
Nc_Gest_Marfuri	Debit Credit Valoare Fel_Doc Nr_Doc Data_Doc Contab Explic Sursa_Profit	C(11) C(11) N(15,2) C(3) N(8) D L C(45) C(15)	Nu Nu Nu Nu Nu Nu Nu 	 0 F 		Debit+Credit+Fel_Doc+Nr_Doc+Data_Doc		Fel_Doc+Nr_Doc+Data_Doc	Misc_Marfuri(Fel_Doc+Nr_Doc+Data_Doc)	Cascadă	Cascadă	1 an
Stoc_Marfuri	Cod Den Um Gest Pret_Achiz Pret_Vanz Stoc Stoc_Brut Stoc_Init Stoc_Br_In Int_An Ies_An Int_Lun Ies_Lun Ultima_Int Ultima_Ies	C(10) C(30) Unit_Mas C(2) N(20,2) N(20,2) N(15,3) N(15,3) N(15,3) N(15,3) N(15,3) N(15,3) N(15,3) N(15,3) D D	Nu Nu Nu Nu Nu Nu Nu 	 	Stoc=Stoc_Init+Int_An-Ies_An	Cod		Cod Gest	Marfuri(CodP) Gest_Marfuri(Cod_Gest)	Cascadă Interzis	Cascadă Cascadă	1 an
Agenti	Cod_Agent Den_Agent Divizie	C(8) C(20) C(10)	Nu Nu 	 		Cod_Agent						5 ani
Clienti	Cod_Client Cont Den_Client Loc_Clie	C(7) C(11) C(30) C(20)	Nu Nu Nu 	 	Sold_Db_An (Rul_Db_An – Rul_Cr_An)	Cod_Client	Cont	Grupa	Grupe_Clienti(Cod_Grupa)	Set NULL	Cascadă	5 ani

	nt											
	Adr_Client	C(30)										
	Tar_Client	C(15)										
	Nr_Fisc	N(8)										
	Platit_TVA	L		T								
	Grupa	C(7)										
	Aut_Alcool	C(20)										
	Aut_Tutun	C(20)										
	Aut_Cafea	C(20)										
	Blocat	L		F								
	Zile_Scad	N(3)		30								
	Limita_Cr	N(15,2)		0								
	Sold_Db_An	N(14,2)	Nu	0								
	Rul_Db_An	N(14,2)	Nu	0								
	Rul_Cr_An	N(14,2)	Nu	0								
	Rul_Db_Lun	N(14,2)	Nu	0								
	Rul_Cr_Lun	N(14,2)	Nu	0								
Banca_Client	Cod_Client	C(7)	Nu			Cod_Client+Cont_Banca	Cont_Client+Den+Banca	Cod_Client	Clienti(Cod_Client)	Cascadă	Cascadă	5 ani
	Cont_Banca	C(22)	Nu									
	Den_Banca	C(30)	Nu									
Emise	Nr_Fact	N(8)	Nu			Nr_Fact+Data_Emi+Cod_Fact+Secv		Cod_Client	Clienti(Cod_Client)	Interzis	Cascadă	1 an
	Cod_Fact	Cod_Fact	Nu	INT								
	Tip_Rand	C(1)	Nu									
	Secv	N(1)	Nu									
	Cod_Client	C(7)	Nu									
	Data_Emi	D	Nu									
	Debit	C(11)	Nu									
	Credit	C(11)	Nu									
	Valoare	N(15,2)	Nu	0								
	Stare	N(1)	Nu									
	Scutit_TVA	L	Nu	F								
	Cota_TVA	N(5,2)	Nu	0.19								
	Contab	L	Nu	F								
	Simb_Mon	C(4)	Nu	LEI								
	Val_Valuta	N(15,2)		0								
	Sursa_Prof	C(15)										
Grupe_Clienti	Cod_Grupa	C(7)	Nu			Cod_Grupa						5 ani
	Den_Grupa	C(30)	Nu									
	Caracteristici	C(250)										
Incasari	Nr_Fact	N(8)	Nu			Nr_Fact+Data_Emi+Data_Incas+Nr_Doc_In+Fel_Doc_In		Nr_Fact+Data_Emi	Emise(Nr_Fact+Data_Emi)	Interzis	Cascadă	1 an
	Data_Emi	D	Nu									
	Data_Incas	D	Nu									
	Debit	C(11)	Nu									
	Credit	C(11)	Nu									
	Valoare	N(15,2)	Nu	0								
	Stare	N(1)	Nu									
	Contab	L	Nu	F								
	Simb_Mon	C(4)		LEI								
	Val_Valu	N(15,2)										

	ta											
	Fel_Doc_In	C(3)	Nu									
	Nr_Doc_In	N(8)	Nu									
	Sursa_Profit	C(15)										
Istoric_Vanzari	Cod_Client	C(7)	Nu			Nr_Fact+Data_Emi+Nr_Doc_In+Fel_Doc_In+Dat_Op		Cod_Client Nr_Fact+Data_Emi+Cod_Fact Nr_Fact+Data_Emi+Nr_Doc_In+Fel_Doc_In+Data_Op Cod_Agent	Clienti(Cod_Client) Emise(Nr_Fact+Data_Emi+Cod_Fact) Incasari(Nr_Fact+Data_Emi+Nr_Doc_In+Fel_Doc_In+Data_Incas) Agenti(Cod_Agent)	Interzis Cascadă Cascadă Set NULL	Cascadă Cascadă Cascadă Cascad	1 an
	Data_Emi	D	Nu									
	Nr_Fact	N(8)	Nu									
	Fel_Op	Op_Cli	Nu									
	Data_Op	D	Nu									
	Simb_Mon	C(4)										
	Data_Curs	D										
	Suma	N(15,2)	Nu									
	Contab	L	Nu									
	Fel_Doc_In	C(3)	Nu									
	Nr_Doc_In	N(8)	Nu									
	Data_Scad	D	Nu									
	Cod_Agent	C(8)										
	Cont_In	C(11)	Nu									
	Cod_Fact	C(3)	Nu									
Nc_Clienti	Debit	C(11)	Nu			Debit+Credit+Fel_Doc+Nr_Doc+Data_Doc		Fel_Doc+Nr_Doc+Data_Doc	Emise(Cod_Fact+Nr_Fact+Data_Emi)	Cascadă	Cascadă	1 an
	Credit	C(11)	Nu									
	Valoare	N(15,2)	Nu	0								
	Fel_Doc	C(3)	Nu									
	Nr_Doc	N(8)	Nu					Fel_Doc+Nr_Doc+Data_Doc	Incasari(Fel_Doc_In+Nr_Doc_In+Data_Incas)	Cascadă	Cascadă	
	Data_Doc	D	Nu									
	Contab	L	Nu	F								
	Explic	C(45)										
	Sursa_Profit	C(15)										
Vânzări	CodP	C(10)	Nu			CodP+Cod_Furn+Cod_Client+Cod_Agent+CodT		CodP	Mărfuri(CodP)	Cascadă	Cascadă	5 ani
	Cod_Furn	C(7)	Nu					Cod_Furn	Furnizori(Cod_Furn)	Cascadă	Cascadă	
	Cod_Client	C(7)	Nu					Cod_Client	Clienti(Cod_Client)	Cascadă	Cascadă	
	Cod_Agent	C(8)	Nu					Cod_Agent	Agenti(Cod_agent)	Cascadă	Cascadă	
	CodT	C(10)	Nu					CodT	Timpi(CodT)	Cascadă	Cascadă	
Timpi	CodT	C(10)	Nu			CodT						5 ani
	Nume_Zi	C(10)	Nu									
	Zi_Din_Sapt	N(1)	Nu									
	Zi_Din_Luna	N(2)	Nu									
	Sapt_Din_Luna	N(1)	Nu									
	Sapt_Din_An	N(2)	Nu									
	Luna_Din_An	N(2)	Nu									
	Nume_Luna	C(10)	Nu									
	Trimestru	N(1)	Nu									
	An	N(4)	Nu									

Domenii.

Nume Domeniu	Tip de Date	Lungime	Set Posibil de Valori	Interval Posibil de Valori
Op_Furn	C	1	C,P	
Unit_Mas	C	3	BUC,ST,KG PAC,BAX,NAV,BID,ML,MP,MC	

Op_Misc	C	1	I,E,R	
Op_Cli	C	1	I,E	
Cod_Fact	C	3	INT,EXT	

Tranzacţii

Identificator Tranzacţie	Descriere Tranzacţie	Frecvenţa Tranzacţiei	Orele cu Incărcare de Vârf	Restricţie timp răspuns
F1	Preluarea datelor unei noi facturi	30/oră	10-16	
F2	Stornarea unei facturi	3/oră	10-16	2 s
F3	Plata unei facturi	80/zi	12-14	
F4	Stornare plată factură	8/zi	12-14	2 s
F5	Generare Jurnal de Cumpărări	10/zi	12-12:30	30 s
F6	Lista facturilor primite de la un anume furnizor	80/zi	10-12	10 s
F7	Lista facturilor primite si neachitate unui anume furnizor	80/zi	10-12	10 s
F8	Furnizori de plătit pe facturi	10/zi	8-10	1 min
F9	Furnizori de plătit pe solduri	10/zi	8-10	1 min
F10	Plaţi scadente către furnizori	10/zi	16-18	20 s
F11	Plăţi întârziate către furnizori	10/zi	16-18	20 s
F12	Balanţa furnizorilor	10/zi	14-14:15	5 min
F13	Generare Note de Recepţie	30/oră	10-16	2 s
F14	Istoricul operaţiilor cu furnizorii	10/zi	16-18	5 min
F15	Vizualizare date generale furnizor	50/zi	10-14	2 s
F16	Adăugare date generale furnizor nou	10/zi	10-14	
F17	Modificare date generale furnizor	50/lună	10-14	2 s
F18	Eliminare date generale furnizor	100/an	10-14	2 s
F19	Vizualizare bănci furnizor	50/zi	10-14	2 s
F20	Adăugare date bancă furnizor	50/zi	10-14	
F21	Modificare date bancă furnizor	50/lună	10-14	2 s
F22	Eliminare date bancă furnizor	100/an	10-14	2 s
G1	Adăugare marfă în nomenclator mărfuri	50/oră	10-16	
G2	Vizualizare marfă din nomenclatorul de mărfuri	500/oră	10-16 17-19	2 s
G3	Modificare marfă din nomenclatorul de mărfuri	5/oră	10-16	2 s
G4	Eliminare marfă din nomenclatorul de mărfuri	20/lună	14-16	2 s
G5	Stocul de mărfuri pe gestiuni	50/lună	16-18	10 min
G6	Adăugare gestiune nouă	20/an	10-14	
G7	Vizualizare gestiune existentă	30/oră	10-16	2 s
G8	Modificare date gestiune	5/lună	10-14	2 s
G9	Eliminare gestiune	10/an	10-14	2 s
G10	Lista furnizorilor existenţi	40/lună	8-10	2 min
G11	Lista clienţilor existenţi	40/lună	8-10	2 min
G12	Lista intrărilor de mărfuri de la un anumit furnizor	200/lună	8-10	5 min
G13	Lista intrărilor de mărfuri de la terţi	40/lună	8-10	5 min
G14	Lista intrărilor de mărfuri de la subunităţi	200/lună	8-10	5 min
G15	Lista intrărilor de mărfuri din plus de inventar	20/an	10-14	5 min
G16	Lista ieşirilor de mărfuri către un anumit client	400/lună	8-12	5 min
G17	Lista ieşirilor de mărfuri prin consum de materiale	120/lună	14-16	5 min
G18	Lista ieşirilor de mărfuri prin transfer la terţi	40/lună	12-14	5 min
G19	Lista ieşirilor de mărfuri prin transfer la subunităţi	20/zi	8-14	5 min
G20	Lista ieşirilor de mărfuri prin transfer în obiecte de inventar în folosinţă	10/zi	14-16	5 min
G21	Lista ieşirilor de mărfuri prin transfer en-detail unitate	10/zi	16-18	5 min
G22	Lista ieşirilor de mărfuri prin minus de inventar	20/an	10-14	5 min
G23	Lista ieşirilor de mărfuri prin perisabilităţi	10/lună	10-14	5 min
G24	Lista ieşirilor de mărfuri prin protocol	10/lună	10-14	5 min
G25	Lista ieşirilor de mărfuri prin investiţii	10/lună	10-14	5 min
G26	Lista ieşirilor de mărfuri prin casare obiecte inventar	10/lună	10-14	5 min
G27	Fişa de magazie a unei anumite mărfi	200/zi	10-16	1 min
G28	Stocul pe articole la preţ de achiziţie	50/zi	10-18	2 min
G29	Stocul pe articole la preţ de vânzare	50/zi	10-18	2 min
G30	Stocul pe grupe de articole	50/zi	10-18	2 min
G31	Stocul pe grupe şi preţuri de achiziţie	50/zi	10-18	2 min
G32	Generare inventar stocuri de mărfuri	60/an	16-18	10 min
G33	Intrări pe lună	30/lună	12-14	5 min
G34	Intrări pe articole	20/lună	12-14	5 min
G35	Intrări pe document	150/lună	08-14	20 s
G36	Ieşiri pe lună	20/lună	12-14	5 min
G37	Ieşiri pe articole	20/lună	12-14	5 min
G38	Ieşiri pe document	150/lună	12-14	20 s
G39	Ieşiri pe cote TVA	20/lună	12-14	5 min

G40	Ieşiri pe magazin	20/săpt	12-14	5 min
G43	Stornare operaţie de intrare mărfuri	2/oră	10-14 17-19	2 s
G44	Stornare operaţie de ieşire mărfuri	20/oră	10-14 17-19	2 s
G45	Balanţa analitică a mărfurilor la preţ de achiziţie	15/lună	16-18	10 min
G46	Balanţa analitică a mărfurilor la preţ de vânzare	15/lună	16-18	10 min
G47	Balanţa analitică a mărfurilor la preţ devize	15/lună	16-18	10 min
G48	Balanţa analitică a mărfurilor la preţ devize/lei	15/lună	16-18	10 min
G49	Jurnalul de mişcare pe gestiune de la ... până la ...	75/lună	16-18	10 min
C1	Emitere factură client	300/oră	10-14 17-19	10 s
C2	Stornare factură client	10/oră	10-14 17-19	2 s
C3	Incasare factură	250/oră	10-14 17-19	5 s
C4	Lista facturilor emise	30/zi	17-18	10 s
C5	Lista chitanţelor emise	30/zi	17-18	10 s
C6	Generare Jurnal de Vânzări	30/zi	17-18	30 s
C7	Borderou vânzări cu amănuntul pe societate	40/lună	17-18	5 min
C8	Borderou vânzări cu amănuntul pe gestiuni	30/zi	17-18	5 min
C9	Lista clienţilor	40/lună	08-10	2 min
C10	Fişa unui client	50/zi	08-10	20 s
C11	Lista clienţilor debitori	20/zi	08-10	1 min
C12	Lista facturilor scadente	20/zi	08-10	20 s
C13	Balanţa clienţilor	10/zi	08-10	5 min
C14	Lista vânzărilor către clienţi de la ... până la....	20/zi	12-14	2 min
C15	Lista vânzărilor către un anumit client (Cu Total/Fără Total)	30/zi	10-14	2 min
C16	Lista vânzărilor de produse ce provin de la acelaşi furnizor (Cu Total/Fără Total)	40/lună	12-14	5 min
C17	Lista vânzărilor de produse efectuate de un agent de vânzări (Cu Total/Fără Total)	400/lună	12-14	2 min
C18	Lista vânzărilor de produse pe grupe de clienţi (Cu Total/Fără Total)	40/lună	12-14	5 min
C19	Lista vânzărilor de produse pe articole (Cu Total/Fără Total)	400/lună	12-14	2 min
C20	Lista vânzărilor de produse pe gestiuni (Cu Total/Fără Total)	400/lună	12-14	2 min
C21	Raportul vânzărilor la preţ mediu	40/lună	16-18	5 min
C22	Raportul vânzărilor centralizat	40/lună	16-18	5 min
C23	Sortarea valorică a clienţilor	10/zi	17-18	2 min
C24	Istoricul facturilor emise	10/sapt	19-19:30	1 min
C25	Vizualizare factură emisă	100/zi	10-14 17-19	2 s
C26	Lista facturilor neîncasate	10/zi	08-10	20 s
C27	Lista facturilor încasate parţial	10/zi	08-10	20 s
C28	Lista încasărilor din vânzări cu amănuntul	10/zi	19-19:30	30 s
C29	Lista bonurilor de vânzare din gestiuni en-detail	10/zi	19-19:30	1 min
C30	Lista tuturor clienţilor sortată după denumirea clientului	10/lună	16-18	2 min
C31	Lista tuturor clienţilor sortată după localitatea clientului	10/lună	16-18	2 min
C32	Lista vânzărilor de mărfuri către un client	50/lună	12-14	2 min
C33	Lista jurnalului de vânzare pe agenţi comerciali de la .data ... până la data ...	150/lună	16-18	10 min
C34	Lista clienţilor debitori pe agenţi comerciali la data de... mai vechi de ... zile	150/lună	16-18	5 min
C35	Lista încasării facturilor pe agenţi comerci de la data ... până la data ...	150/lună	16-18	5 min
C36	Lista vânzărilor de mărfuri pe clienţi	10/lună	16-18	10 min
C37	Lista vânzărilor de mărfuri pe agenţi comerciali	10/lună	16-18	10 min
C38	Lista vânzărilor de mărfuri pe agenţi comerciali şi pe clienţi	10/lună	16-18	10 min
C39	Vizualizare date client	200/oră	10-14 17-19	2 s
C40	Adăugare date client nou	100/oră	10-14 17-19	
C41	Modificare date client	10/zi	10-14 17-19	2 s
C42	Eliminare date client	10/an	10-12	2 s
C43	Vizualizare date grupă clienţi	30/zi	10-14 17-19	2 s
C44	Adăugare date grupă nouă clienţi	30/an	10-12	
C45	Modificare date grupă clienţi	50/an	10-12	2 s
C46	Eliminare date grupă clienţi	10/an	10-12	2 s
C47	Vizualizare date agent comercial	5/zi	14-16	2 s
C48	Adăugare date agent comercial	5/lună	14-16	
C40	Modificare date agent comercial	2/lună	14-16	2 s
C50	Eliminare date agent comercial	2/lună	14-16	2 s

2.6. *Schema fizică a bazei de date*

Nume Relaţie	Tip Fişier	Tip Partiţie	Cheie Partiţionare	Dimesiune	Nume Index	Cheie Indexare	Tip Index	Dimensiune Index
Agenti	Secvential			0,003				
Banca_Client	IOT			438,593	ICod_BancaClient	Cod_Client	IndexSecundar,Arbore B	61,4
Banca_Furnizori	IOT			40,698				
Clienti	Partitionat	Hash	Cod_Client	2506,249	IDen_Client	Den_Client	IndexGlobal,Arbore B	300,75
					ICod_Client	Cod_Client	IndexLocalPartiţie,Arbore B	70,17
Emise	Secvential			581,451				
Furnizori	IOT			106,685	IDen_Furnizori	Den_Furn	IndexSecundar,Arbore B	22,86
Gest_Marfuri	IOT			75,191	IDen_Gest	Den_Gest	IndexSecundar,Arbore B	22,56
Grupe_Clienti	IOT			601,515	IDen_Grupa	Den_Grupa	IndexSecundar,Arbore B	75,19
Incasari	Secvential			501,251				
Istoric_Cumparari	Secvential			706,012				
Istoric_Vanzari	Partitionat	Hash	Cod_Client	1082,703				
Marfuri	IOT			173,616	IDen_Marfuri	DenP	IndexSecundar,Arbore B	96,45
Misc_Marfuri	Partitionat	List	Cod_Oper	941,35				
Nc_Clienti	Secvential			60,15				
Nc_Furnizori	Secvential			60,818				
Nc_Gest_Marfuri	Partitionat	Range	Data_Doc+Fel_Doc+Nr_Doc	1082,703				
Stoc_Marfuri	IOT			121,804	IDen_Stoc	Den	IndexSecundar,Arbore B	20,3

F *Lista figurilor*

Capitolul 2

Capitolul 5

T *Lista tabelelor*

Capitolul 3

Capitolul 4

G *Lista graficelor*

Capitolul 3

Capitolul 5

B *Bibliografie*

[Agres 86] Agresti, W.W., *New paradigms for software development*, IEEE Press, 1986

[Bella 00–1] Bellatreche, L., Karlapalem, K., Mohania, M., *Some Issues in Design of Data Warehousing Systems*, Distributed and Parallel Databases Journal, 2000

[Bella 00-2] Bellatreche, L., Karlapalem, K., *Logical and Physical Design in Data Warehousing Enviroments*, IDEAS'2000, 2000

[Bloor 02] *Databases: an Evaluation and Comparison*, Bloor Research, 2002

[Bloor 03-1] *Database Performance IBM, Oracle & Microsoft*, Bloor Research, 2003

[Bloor 03-2] *DB2 UDB V8.1 fom IBM*, Bloor Research, 2003

[Chen 76] Chen, P.P., *The Entity-Relationship Model – Toward a Unified View of Data*, ACM transactions on Database Systems, 1,1, 1976

[Cobb 90] Cobb, R.H., Mills, H.D., *Engineering software under statistical quality control*, IEEE Software 7, 6, 1990, pag.44-54

[Codd 70] Codd, E.F., *A Relational model for Large Shared data Banks*, Communications of the ACM, 13, 1970

[Codd 79] Codd, E.F., *Extending the Database Relational model to capture more Meaning*, ACM Transactions on Database Systems, 4, 4, 1979

[Cohen 86] Cohen, B., Harwood, W.T., Jackson, M.I., *The specification of Complex Systems*, Addison-Wesley Publishing Company, 1986

[Conno 01] Connolly, T., Begg, C., Strachan, A., *Database Systems – A practical approach to design, implementation and management*, Addison-Wesley

Publishing Company, Editie Teora, 2001

[Date 03] Date, C.J., *An Introduction to database systems*, Addison-Wesley Publishing Company, 2003

[Datta 90] Datta, A., Ramamritham, K., Thomas, H., *Curio: A Novel Solution for Efficient Storage and Indexing in Data Warehouses*, Proceedings of the International Conference on Very Large Databases, 1990, pag.730-733

[DeMar 78] DeMarco, T., *Structured Analysis and System Specification*, New York Yourdon Press, 1978

[Elbra 92] Elbra, R.A., *Computer Security Handbook*, Oxford: NCC Blackwell, 1992

[Fagan 86] Fagan, M.E., *Advances in software inspections*, IEEE Transactions on Software Engineering, 12, 7, 1986

[Flemi 89] Fleming, C., Von Halle, B., *Handbook of Relational Database Design*, Addison-Wesley Publishing Company, 1989

[Gladd 82] Gladden, G.R., *Stop the life cycle – I want to get off*, ACM Software Engineering Notes 7, 2, 1982, pag.35-39

[Hamme 81] Hammer, M., McLeod, D., *Database descriptions with SDM: a semantic database model*, ACM Transactions on Database Systems, 6, 3, 1981, pag.351-386

[Herbe 90] Herbert, A.P., *Computer Security: Policy, planning and practice*, Roberts D.W. Blenheim Online, 1990

[Honou 98] Honour, E., Dalberth, P., Kaplan, A., Mehta, A., *Oracle8 How to*, Waite Group Press, 1998.

[IBM 02] *DB2 Concepts*, IBM Corporation, Release UDB V8.1, 2002

[Ince 87] Ince, D.C., Hekmatpour, S., *Software prototyping – progress and prospects*, Information and Software Technology, 29, 1, 1987, pag.8-14

[Kochh 00] Kochhar, N., *SQL*, Oracle Corparation, 2000

[Koopm 03] Koopmann, J.F., *Key Performance Metrics*, Database Journal, Ian.2003

[Lejeu 02-1] Lejeune, H., *Technical Comparison of Oracle 9i Database vs. IBM DB2 UDB V8.1*, Oracle Corporation, 2002

[Lejeu 02-2] Lejeune, H., *Technical Comparison of Oracle 9i Database vs. SOL Server 2000*, Oracle Corporation, 2002

[Linger 94] Linger, R.C., *Cleanroom process model*, IEEE Software 11, 2, 1994, pag.50-58

[Luers 95] Luers, T., *Essential Oracle7*, Sams Publishing, 1995

[Lumpk 03] Lumpkin, G., Jakobsson, H., *Query Optimization in Oracle 9i*, Oracle Corporation, 2003

[McCra 82] McCracken, D.D., Jackson, M.A., *Lifecycle concept considered harmful*, ACM Software Engineering Notes 7, 2, 1982, pag.28-32

[Medin 97] Medina, E.F., Piattini, M., *A Methodology for Multilevel Database Design*, Proceedings of ACM SIGMOD International Conference on Management of Data, 1997, pag.458-470

[Micro 00] *SQL Server 2000 Concept*, Microsoft Corporation, 2000

[Micro 02-1] *Tips on Optimizing SQL Server Composite Indexes*, Microsoft Corporation, 2002

[Micro 02-2] *Tips on Performance Tuning SQL Server Database Clustered Indexes*, Microsoft Corporation, 2002

[Micro 03-1] *Tips on Rebuilding SQL Server Database Indexes for Maximum Performance*, Microsoft Corporation, 2003

[Micro 03-2] *Tips on Performance Tuning SQL Server Database Indexes*, Microsoft Corporation, 2003

[Micro 03-3] *Tips on Performance Tuning SQL Server Database Non-Clustered Indexes*, Microsoft Corporation, 2003

[Micro 03-4] *Tips on Performance Tuning SQL Server Database Covering Indexes*, Microsoft Corporation, 2003

[Micro 03-5] *VLDB Performance Tuning and Optimization Tips*, Microsoft Corporation, 2003

[Micro 03-6] *How to Select Indexes for Your SQL Server Tables,* Microsoft Corporation, 2003

[Micro 03-7] *How to Perform a SQL Server Database Index Performance Audit,* Microsoft Corporation, 2003

[Micro 03-8] *SQL Server Indexes Are Not Created Equal,* Microsoft Corporation, 2003

[Mills 87-1] Mills, H.D., O'Neill, D., Linger, R.C., Dyer, M., Quinnan, R.E., *The management of software engineering*, IBM Syst. J.24, 2, 1987, pag.414-477

[Mills 87-2] Mills, H.D., Dyer, M., Linger, R.C., *Cleanroom software engineering*, IEEE Software 4, 5, 1987, pag.19-25

[Mills 90] Mills, H.D., *Engineering software under statistical quality control*,

IEEE Software, 7, 6, 1990

[Mitea 98-1] Mitea, A.C., - *Some aspects about database design - normalization approach*, Acta Universitatis Cibiniensis vol.XII(1) Technical series A. Electronics, Electrotehnics and Computer Science, Sibiu, 1998

[Mitea 98-2] Mitea, A.C., - *Some aspects about database design –object-oriented approach*, Acta Universitatis Cibiniensis vol.XII(1) Technical series A. Electronics, Electrotehnics and Computer Science, Sibiu, 1998

[Mitea 99-1] Mitea, A.C., - *Implementing persistence in object oriented databases*, Beyond 2000: Engineering Research Strategies – International Conference, Sibiu, 1999

[Mitea 99-2] Volovici, D., Mitea, A.C., -*Some aspects about object identity in object-oriented databases*, Acta Universitatis Cibiniensis vol.XXXVIII Technical series E. Computer Science and Automatic Control, Sibiu, 1999, pag.131-134

[Mitea 00] Mitea, A.C., - *Baze de Date Orientate pe Obiecte - Stadiul Actual*, Referat doctorat, Universitatea „Politehnica" Timişoara, 2000

[Mitea 01] Mitea, A.C., - *Baze de Date Relaţionale şi Orientate pe Obiecte - Studiu Comparativ al Performanţelor*, Referat doctorat, Universitatea „Politehnica" Timişoara, 2001

[Mitea 02-1] Mitea, A.C., *Baze de date relaţionale*, Ed."Alma Mater", Sibiu, 2002

[Mitea 02-2] Mitea, A.C., *Dezvoltarea sistemelor informatice*, Ed.Univ."Lucian Blaga" din Sibiu, 2002

[Mitea 02-3] Mitea, A.C., *Baze de date relaţionale şi orientate-obiect*, Ed.Univ."Lucian Blaga" din Sibiu, 2002

[Mitea 04-1] Mitea, A.C., Jian, I., *A Physical Design Methodology for Databases*, Scientific Bulletin of „Politehnica" University of Timişoara, Transactions on Automatic Control and Computer Science, Vol. 49(63) No.3, 2004, pag.11-17

[Mitea 04-2] Mitea, A.C., *A Different Aproach to Transaction Analysis*, SACCS 2004 – 8th International Symposium on Automatic Control and Computer Science, Iaşi, Romania, 2004, pag.79-84

[Mitea 04-3] Mitea, A.C., *A proposal for a mltiple join index*, Proceeding of the IASTED International Conference on Software Engineering and Applications, Cambridge, USA, 2004, pag.1-7.

[O'Neil 97] O'Neil, P., Quass, D., *Improved query performance with variant indexes*, Proceedings of ACM SIGMOD International Conference on Management of Data, 1997, pag.38-49

[Oracl 92] *Oracle7 Server Concepts Manual*, Oracle Corporation, Redwood City, CA, 1992

[Oracl 99] *Oracle8i Concepts*, Oracle Corporation, Release 8.1.5, 1999

[Oracl 02] *Oracle9i Concepts*, Oracle Corporation, Release 9.1.2, 2002

[Popes 01] Popescu, I., *Modelarea Bazelor de Date*, Editura Tehnică, 2001

[Rob 95] Rob, P., Coronel, C., *Database Systems: design, implementation and management*, Boyd & Fraser Publishing Company, 1995

[Roger 89] Rogers, U., *Denormalization: Why, What and How?* Database Programming and Design, 2(12), 1989, pag.46-53

[Royce 70] Royce, W.W., *Managing the development of large software systems: Concepts and techniques*, In Proceedings of IEEE WESTCON, Los Angeles, CA, 1970, pag.1-9

[Rusin 95] Rusinkiewicz, M., Sheth, A., *Specification and Execution of Transactional Workflows*, ACM Press Addison-Wesley Publishing Company, 1995, pag. 592-620

[Schwi 00] Schwinn, U., Venkatachalam, V., *Oracle8: Database Administration*, Oracle Corporation, 2000

[Selby 95] Selby, R.W., Basili, V.R., Baker, F.T., *Cleanroom software development: An empirical evaluation*, IEEE Transactions on Software Engineering SE-13, 9,1995, pag.1027-1037

[Spive 92] Spivey, J.M., *The Z Notation: A reference manual*, London, Prentice Hall International, 1992

[Somme 96-1] Sommerville, I., *Software Process Models*, ACM Computing Surveys, Vol.28, No.1, March 1996

[Somme 96-2] Sommerville, I., *Software Engineering*, Addison-Wesley Publishing Company, 1996

[Steph 00] Stephens, R., Plew, R., *Database Design*, Sams Publishing Company, 2000

[Thomb 01] Thombre, N., Ozbutun, C., *Key Data Warehousing Features in Oracle 9i*, Oracle Corporation, 2001

[Valdu 87] Valduriez, P., *Join indices*, ACM Transactions on Database Systems,

12(2), 1987, pag.218-246

[Wiede 83] Wiederhold, G, *Database design*, New York McGraw-Hill, 1983

300 000 lei
30 lei