|  |  |  |
| --- | --- | --- |
| UnivBacau nou |  |  |

**Proiectul privind Învăţământul Secundar (ROSE)  
 SCHEMA DE GRANTURI PENTRU UNIVERSITĂŢI – NECOMPETITIVE  
Beneficiar: Universitatea „Vasile Alecsandri” din Bacău**

**Titlul subproiectului:**

**Matematică, Informatică şi Biologie - educaţie şi profesii pentru viitor.**

**Rămâi Integrat şi Total Motivat (Mate-Info-Bio-RITM)**

**Acord de grant nr. 191/SGU/NC/II din 12.09.2019**

**Material didactic pentru anul I, programul de studii INFORMATICĂ**

**pentru disciplina**

**BAZE DE DATE**

**conf. univ. dr. Cerasela Crişan**

**2019**

**Baze de date**

În perioada actuală este greu de găsit un domeniu al informaticii care să fie la fel de prezent în viaţa noastră, aşa cum sunt bazele de date. În interacţiunea cu organismele financiare şi bancare, persoanele sunt identificate în sistemele de calcul care gestionează taxele, impozitele, ratele; când achiziţionează bunuri sau servicii în mediul *online*, fiecare cumpărător generează o tranzacţie care este gestionată şi cu ajutorul bazelor de date ale vânzătorului. Cu greu ne putem imagina că există utilizator al internetului care să nu aibă un cont de email, sau care să nu fie prezent în *social media* (Facebook, Instagram, Twitter, LinkedIn, etc.)

În toate aceste activităţi şi în multe altele care utilizează sistemele de calcul actuale, se accesează şi se actualizează înregistrări în diverse baze de date. De aceea, cunoştinţele şi abilităţile din domeniul bazelor de date sunt extrem de importante în meseriile legate de Informatică, Tehnologia Informaţiilor şi Comunicaţii, Ştiinţa Calculatoarelor.

**CUPRINS**

|  |  |
| --- | --- |
| 1. **Concepte fundamentale ale bazelor de date** | **5** |
| 1. **Algebra relaţională** | **13** |
| 1. **Normalizarea bazelor de date relaţionale** | **21** |
| 1. **Tranzacţii** | **27** |
| 1. **SQL** | **31** |
| 1. **Baze de date distribuite** | **39** |
| 1. **Stocarea masivă** | **43** |
| 1. **NoSQL** | **47** |
| **Resurse disponibile online** | **49** |

**I. Concepte fundamentale ale bazelor de date**

Resursele lumii actuale sunt: materie, energie, informaţie. Societatea consumă aceste resurse în procesele de producţie, pentru a-şi asigura existenţa şi dezvoltarea.

*Informaţia* este un mesaj obiectiv care înlătură o nedeterminare. *Data* este o reprezentare a informaţiei. Datele se prelucrează manual sau automat (cu ajutorul calculatorului). În cele ce urmează, ne vom referi la prelucrarea automată a datelor.

*Informatica* este ştiinţa care se ocupă cu prelucrarea eficientă a informaţiei, folosind sistemele de calcul. Este unul dintre puţinii termeni româneşti care provin din limba franceză, prin transfer din limba germană (apare pentru prima oară în anul 1957 într-un articol publicat în revista *SEL-Nachrichten* de *Karl Steinbuch*: *Informatik: Automatische Informationsverarbeitung*, titlu care în limba română înseamnă *Informatica: Procesarea automată a informaţiei*).

Prelucrarea datelor se realizează de către specialiştii în Informatică. Aceştia proiectează, realizează şi întreţin pachetele de aplicaţii necesare, folosind resursele hardware pe care le au la dispoziţie. Este deci nevoie de trei componente pentru a asigura succesul prelucrării datelor:

* personal specializat
* software: pachetele de aplicaţii şi datele
* hardware: sistemele de calcul şi comunicaţii.

Datele se clasifică astfel:

* elementare (simple, scalare, atomice): indivizibile, tratate în mod unitar de aplicaţiile care le gestionează; de exemplu, datele de tip *int* din limbajul *C*;
* compuse (complexe): divizibile, care se pot descompune în componente; aceste componente pot fi tratate la nivel individual prin aplicaţii; de exemplu, datele de tip *struct* din limbajul *C*.

Într-o aplicaţie, data este caracterizată prin:

* identificator: numele care permite adresarea datei;
* tip de dată: mulţimea posibilă de valori şi setul posibil de prelucrări;
* valoare: un element al mulţimii de valori.

De exemplu, linia de cod scris în limbajul *C*:

*int x = 3;*

declară variabila *identificată* prin *x* ca fiind de tip *int* şi o iniţializează cu *valoarea* 3. *Mulţimea posibilă de valori* pentru acest tip de dată include numerele întregi aflate în intervalul [-32.767, 32.767]; în funcţie de procesorul pe care se execută codul, această mulţime poate conţine şi alte valori. Exemple de *prelucrări* care se pot aplica unei date de tip *int* sunt cele *numerice* (adunare, scădere, înmulţire, împărţire, modulo, incrementare, decrementare, etc.) sau cele de *comparare* (identic, diferit, mai mare, mai mic, mai mare sau egal, mai mic sau egal).

O *colecţie de date* este un set de date despre o anumită categorie de obiecte. De exemplu, colecţia de date care reprezintă catalogul de produse ale unei firme. În acest caz, pentru fiecare produs sunt stocate: denumirea, caracteristici, preţul, unitatea de măsură, câteva imagini, modul de livrare.

Stocarea datelor se realizează:

* *în memoria internă*, atunci când datele sunt definite şi utilizate de aplicaţii. Aceste date sunt *volatile*, deoarece se pierd la ieşirea din aplicaţie. De exemplu, după ieşirea din programul care utilizează linia de cod

*int x = 3;*

data identificată prin *x* nu mai poate fi utilizată de alte aplicaţii.

* *în memoria externă*, sub formă de *fişiere* sau *baze de date*. În acest caz, datele sunt *permanente*, deoarece se află pe suportul magnetic extern, existând şi când nu sunt prelucrate de aplicaţii.

Un *fişier* este o colecţie de date omogene atât din punctul de vedere al conţinutului, cât şi al prelucrării. Într-un fişier, datele sunt organizate:

* *secvenţial*: înregistrările se stochează în ordinea introducerii lor; căutarea se face în aceeaşi ordine;
* *indexat*: înregistrările se stochează în ordinea introducerii lor, dar o înregistrare se poate accesa direct, pe baza unui fişier auxiliar, numit *index*. Acest index este construit pe baza unei *chei* care identifică în mod unic fiecare înregistrare. Indexul conţine perechi de valori (*cheie*, *adresa înregistrării*) ordonate crescător după *cheie*. Căutarea se poate face şi secvenţial, prin parcurgerea înregistrărilor începând cu prima.
* *relativ*: înregistrările se stochează în ordinea introducerii lor; regăsirea se poate face direct, după o valoare numită *numărul înregistrării*, sau secvenţial.
* *direct*: înregistrările se stochează într-o ordine dată de un algoritm care stabileşte adresa fizică de stocare. Căutarea unei înregistrări se face prin apelarea aceluiaşi algoritm, care va furniza adresa unde se află înregistrarea respectivă.

Stocarea datelor în fişiere are următoarele neajunsuri:

* datele sunt dependende de aplicaţii, fiind definite şi prelucrate de acestea;
* definirea şi validarea datelor este neriguroasă, deoarece aplicaţiile pot avea erori sau pot scăpa din vedere diverse excepţii;
* dacă sunt necesare datele stocate în mai multe fişiere, atunci fiecare fişier trebuie deschis, accesat şi închis;
* datele care au legături logice dar sunt stocate în mai multe fişiere sunt izolate, astfel că aceste legături pot fi alterate pe măsură ce fiecare fişier este actualizat separat;
* fişierele care conţin date netradiţionale (de exemplu imagini, sunete, filme, etc.) pot deveni prea mari pentru a fi manevrate rapid;
* pe măsură ce se prelucrează, datele se pot copia în mod repetat, determinând o *redundanţă* ridicată.

Aceste neajunsuri în prelucrarea datelor au fost depăşite prin apariţia bazelor de date. În anii 60 ai secolului trecut au început dezvoltarea teoretică şi implementarea în practică a modelelor teoretice ale bazelor de date. Acum managementul unei firme, cumpărarea din magazinele virtuale, plata în mediul online a facturilor nu se mai pot face fără gestionarea unor baze de date.

*Baza de date* este o colecţie structurată de date accesibile calculatorului, destinată utilizării concurente. *Concurenţa* înseamnă ca mecanismele de gestionare a bazei de date asigură pentru mai mulţi utilizatori accesul simultan, în timp real. Într-o bază de date există mai multe fişiere între care există legături (dependenţe).

Baza de date urmăreşte îndeplinirea următoarelor *obiective*:

* centralizarea datelor, care este opusă redundanţei exagerate;
* independenţa dintre date şi prelucrări: datele se pot actualiza fără a se modifica aplicaţiile care le prelucrează şi reciproc;
* se pot realiza legături între date, care reprezintă dependenţele de înţeles (*semantice*);
* integritatea datelor: corectitudinea, coerenţa datelor;
* securitatea datlor: protecţia împotriva pierderilor accidentale sau intenţionate;
* confidenţialitatea datelor: asigurarea accesului diferenţiat şi controlat la date;
* partajarea datelor: asigurarea accesului concurent la date.

Bazele de date se *clasifică* după următoarele criterii:

* după *modelul* datelor:
  + *navigaţionale*: înregistrările se caută într-o structură de tip graf. Aceste baze de date sunt ieşite din uz şi pot fi:
    - punctuale: toate înregistrările sunt pe un singur nivel, ceea ce face dificilă căutarea;
    - ierarhice: înregistrările sunt căutate prin deplasarea într-un arbore, de la rădăcină către nodurile terminale. Fiecare nod din ierarhie are deci un singur nod-părinte.
    - de tip reţea: fiecare înregistrare poate avea mai multe noduri-părinte.
  + *relaţionale*: datele sunt stocate în tabele (relaţii) între care sunt definite legături. Acesta este cel mai răspândit model actual, care foloseşte *SQL* (Structured Query Language) ca limbaj de programare.
  + *obiectuale*: înregistrările sunt obiecte, definite ca în limbajele orientate-obiect;
  + *semi-structurate*: înregistrările sunt documente care au structuri definite prin standarde de tip *XML* (eXtensible Markup Language) sau *JSON* (JavaScript Object Notation).
* după *centralizarea* datelor:
  + *centralizate*: toate înregistrările se află în acelaşi loc (din punct de vedere geografic), ele putându-se afla pe mai multe sisteme de calcul, dar care sunt gestionate unitar;
  + *distribuite*: înregistrările se află în diverse locuri (din punct de vedere geografic), putând fi chiar pe continente diferite. Din punct de vedere logic însă, baza de date este tratată unitar.
* după *intensitatea prelucrării* datelor:
  + *operaţionale* (OLTP – OnLine Transaction Processing): datele sunt actualizate masiv;
  + *analitice* (OLAP – OnLine Analytical Processing, statice): datele sunt rar actualizate.

*Sistemul de gestiune a bazei de date* (SGBD) este un pachet de aplicaţii care asigură construirea, alimentarea, actualizarea bazei de date, precum şi dezvoltarea de programe care să o întreţină.

*Funcţiile* unui SGBD sunt:

* stocarea datelor în fişiere;
* gestiunea legăturilor între fişiere;
* introducerea datelor;
* actualizarea datelor;
* consultarea datelor.

Persoanele care interacţionează cu o bază de date sunt:

* utilizatorul, care consultă (interoghează) baza de date;
* operatorul, care actualizează datele şi asigură funcţionarea bazei de date;
* programatorul, care proiectează şi realizează aplicaţiile necesare bazei de date;
* administratorul, care răspunde de proiectarea, crearea şi menţinerea bazei de date.

*Depozitul de date* este o colecţie de date specifică unei organizaţii, care conţine date istorice, arhivate la diverse momente de timp. Depozitul de date este folosit la realizarea unor analize sau rapoarte care să surprindă evoluţia în timp a organizaţiei respective. Depozitele de date furnizează astfel informaţii necesare pentru definirea strategiilor de dezvoltare ale organizaţiilor economice.

**Concepte fundamentale ale bazelor de date relaţionale**

În modelul relaţional, datele sunt stocate în tabele între care există legături. Un tabel (*relaţie*) este caracterizat prin coloanele sale, numite *atribute* sau *câmpuri*. Rândurile tabelului sunt t-upluri distincte şi se numesc *articole* sau *înregistrări*. Fiecare componentă a unui articol se numeşte *valoare* şi se află la intersecţia unui rând cu o coloană. Valorile posibile pentru un câmp formează *domeniul de valori* al câmpului respectiv.

Câmpul sau combinaţia de câmpuri care identifică în mod unic articolele formează o *cheie-candidat*. Un tabel are cel puţin o cheie-candidat: setul complet de câmpuri, deoarece înregistrările sunt unice. Administratorul alege *cheia primară* (primary key) din setul (nevid) de chei-candidat. Nu există o metodă universală pentru alegerea cheii primare; se recomandă să conţină cât mai puţine câmpuri şi să reflecte înţelesul datelor stocate în tabel.

*Cheia secundară* (cheie străină, foreign key) este un câmp sau o combinaţie de câmpuri care modelează o *legătură* între două tabele. Dacă un tabel T1 are cheia primară C1, atunci celălalt tabel T2 are cheia secundară C2 care conţine aceleaşi valori ca şi C1. *Schema* unui tabel este dată de numele tabelului, lista de câmpuri (cu domeniile lor asociate) şi cheia primară.

*Tipuri* de legături între tabelele T1 şi T2:

* 1 - 1: fiecare dintre articolele din T1 se leagă cu 0 sau 1 articole din T2 şi reciproc;
* 1 - n: fiecare dintre articolele din T1 se leagă cu 0, 1 sau mai multe articole din T2, iar articolele din T2 au cel mult un corespondent în T1;
* m - n: fiecare dintre articolele din T1 se leagă cu 0, 1 sau mai multe articole din T2.

**Integritatea bazelor de date relaţionale**

*Integritatea* datelor înseamnă corectitudinea acestora. Din punct de vedere al administrării, menţinerea integrităţii datelor asigură protecţia împotriva accidentelor (căderi ale componentelor, întreruperea alimentării cu energie electrică, erori în aplicaţii, utilizări neaşteptate ale datelor, blocarea accesului la date, activităţi de întreţinere greşit realizate, etc.) şi presupune existenţa metodelor pentru:

* detectarea,
* corectarea
* prevenirea erorilor datelor.

Aceste obiective nu pot fi atinse integral şi întotdeauna, de aceea administratorii bazelor de date caută permanent noi posibilităţi de a asigura integritatea datelor.

Una dintre metodele de asigurare a integrităţii datelor este definirea *restricţiilor de integritate*: un set de reguli care nu permit introducerea de date eronate în baza de date. Restricţiile de integritate se exprimă prin condiţii (expresii logice evaluate la valoarea logică *adevărat*) impuse datelor.

Clasificări ale relaţiilor de integritate:

* după volumul datelor care participă în condiţie:
  + generale: se aplică tuturor tabelelor din baza de date. De exemplu: valorile cheii principale sunt unice.
  + specifice: se referă la un singur tabel. Aceste restricţii de integritate pot fi:
    - pe un singur articol. De exemplu: o persoană înregistrată ca salariat nu poate avea vârsta mai mică de 16 ani.
    - pe un grup de articole. De exemplu: pentru un anumit cont bancar, suma extragerilor în numerar de la ATM într-o zi nu poate depăşi 2500 lei.
* după necesarul de timp implicat în evaluarea condiţiei:
  + statice (în timp zero). De exemplu: valoarea pentru câmpul *Grupa* dintr-un tabel care stochează studenţii unei facultăţi trebuie să fie un număr întreg cu 3 cifre, dintre care prima să fie nenulă.
  + dinamice (se referă la o evoluţie în timp a datelor). De exemplu: se renunţă la tranzacţia începută şi ATM-ul restituie cardul, dacă suma solicitată a fi eliberată este mai mare decât disponibilul aflat în cont.

O altă metodă pentru asigurarea integrităţii datelor este *arhivarea* periodică. Între două arhive consecutive se folosesc aplicaţiile de urmărire a modificărilor datelor (*jurnale*), care se află la dispoziţia administratorului bazei de date. Aceste jurnale păstrează toate transformările datelor şi care sunt aplicaţiile care au efectuat aceste transformări. La nevoie, pe baza arhivelor şi a jurnalelor, administratorul bazei de date poate aduce baza de date la o stare anterioară, care este corectă şi poate reface doar actualizările corecte.

**Securitatea bazelor de date relaţionale**

Securitatea bazelor de date înseamnă protecţia lor împotriva accesului neautorizat (atacurilor, care sunt acţiuni premeditate). Accesul neautorizat poate conduce la citiri, actualizări, ştergeri ale datelor, şi/sau instalarea de aplicaţii (de exemplu de urmărire şi raportare, etc).

Securitatea se asigură prin:

* protecţie fizică (se referă la componentele hardware): de exemplu, suportul magnetic este criptat, conexiunea *wifi* sau cea *blutooth* sunt închise;
* autentificare: se instaurează un sistem de acces alcătuit din *cont* şi *parolă*, care asigură cunoaşterea identităţii persoanei care se conectează;
* autorizare: se oferă drepturi de acces restricţionate pentru diverse roluri. Un *rol* este definit printr-un set de drepturi de acces. De exemplu, într-o firmă, rolul de *economist* permite accesul la datele contabile, dar nu şi la stocul de materiale. La angajare, o persoană primeşte datele de autentificare (cont şi parolă), dar este inclusă şi într-un anumit rol, care autorizează doar anumite acţiuni în ce priveşte baza de date..

**II. Algebra relaţională**

*Algebra* este domeniul matematicii care se ocupă cu simbolurile matematice şi cu modul de utilizare a acestora. *Algebra relaţională* este algebra care manevrează relaţii (tabele). Cum o relaţie este o *mulţime* de articole, operatorii algebrei relaţionale vor genera tot mulţimi de articole, adică tot relaţii.

O *relaţie* *R* care are *n* atribute cu domeniile de valori A1, A2, ..., An este o submulţime a produsului cartezian :

De exemplu, relaţia

|  |  |
| --- | --- |
| **denumire** | **culoare** |
| *fular* | *galben* |
| *fes* | *verde* |
| *fular* | *verde* |
| *fes* | *alb* |

poate fi scrisă utilizând mulţimile:

şi

astfel:

fiind o submulţime a produsului cartezian , care este formată din toate cele 6 combinaţii posibile:

.

La fel, relaţia

|  |  |
| --- | --- |
| **denumire** | **culoare** |
| *fular* | *alb* |
| *fes* | *verde* |

poate fi scrisă astfel:

.

Operatorii algebrei relaţionale sunt: reuniunea, intersecţia, diferenţa, complementara, produsul cartezian, diviziunea, selecţia, proiecţia, uniunea.

1. **Reuniunea**

Cele două relaţii care *se reunesc* trebuie să aibă acelaşi număr de atribute, atributele care corespund având aceleaşi domenii de valori. Relaţia care rezultă în urma reuniunii are aceleaşi atribute ca şi relaţiile reunite şi conţine articolele aflate în prima relaţie sau în a doua relaţie, fără însă a exista duplicate.

De exemplu, reuniunea relaţiilor şi este relaţia :

|  |  |
| --- | --- |
| **denumire** | **culoare** |
| *fular* | *galben* |
| *fes* | *verde* |
| *fular* | *verde* |
| *fes* | *alb* |
| *fular* | *alb* |

1. **Intersecţia**

Cele două relaţii care *se intersectează* trebuie să aibă acelaşi număr de atribute, atributele care corespund având aceleaşi domenii de valori. Relaţia care rezultă în urma intersecţiei are aceleaşi atribute ca şi relaţiile intersectate şi conţine articolele comune.

De exemplu, intersecţia relaţiilor şi este relaţia:

|  |  |
| --- | --- |
| **denumire** | **culoare** |
| *fes* | *verde* |

1. **Diferenţa**

Cele două relaţii care *se scad* trebuie să aibă acelaşi număr de atribute, atributele care corespund având aceleaşi domenii de valori. Relaţia care rezultă în urma diferenţei are aceleaşi atribute ca şi relaţiile şi şi conţine articolele din care nu se află şi în .

De exemplu, este relaţia :

|  |  |
| --- | --- |
| **denumire** | **culoare** |
| *fular* | *galben* |
| *fular* | *verde* |
| *fes* | *alb* |

1. **Complementara**

*Complementara* a unei relaţii care are *n* atribute cu domeniile de valori A1, A2, ..., An are aceleaşi atribute ca şi , iar articolele sale sunt cele aflate în produsul cartezian şi care lipsesc din .

De exemplu, complementara relaţiei este relaţia :

|  |  |
| --- | --- |
| **denumire** | **culoare** |
| *fular* | *alb* |
| *fes* | *galben* |

1. **Produsul cartezian**

Considerăm relaţia care are *n* atribute cu domeniile de valori A1, A2, ..., An şi relaţia care are *m* atribute cu domeniile de valori B1, B2, ..., Bm. Relaţia , *produsul cartezian* al celor două, are atribute care au domeniile de valori A1, A2, ..., An, B1, B2, ..., Bm şi articolele sale sunt constituite prin concatenarea tuturor articolelor din cu toate articolele din .

De exemplu, dacă relaţia este:

|  |  |
| --- | --- |
| **denumire** | **culoare** |
| *fular* | *galben* |
| *fes* | *verde* |

şi relaţia este:

|  |
| --- |
| **preţ** |
| *20* |
| *24* |

atunci relaţia este:

|  |  |  |
| --- | --- | --- |
| **denumire** | **culoare** | **preţ** |
| *fular* | *galben* | *20* |
| *fes* | *verde* | *20* |
| *fular* | *galben* | *24* |
| *fes* | *verde* | *24* |

1. **Diviziunea**

Considerăm relaţia care are *n* atribute cu domeniile de valori A1, A2, ..., An şi relaţia care are *k* atribute () cu domeniile de valori A1, A2, ..., Ak. Relaţia , care se obţine din *diviziunea* relaţiei prin relaţia are atribute care au domeniile de valori Ak+1, Ak+2, ..., An şi articolele sale sunt constituite din valorile care concatenate cu toate articolele din se află în .

De exemplu, dacă este relaţia:

|  |  |  |
| --- | --- | --- |
| **denumire** | **culoare** | **preţ** |
| *fular* | *galben* | *20* |
| *fes* | *verde* | *20* |
| *fular* | *verde* | *19* |
| *fes* | *alb* | *20* |
| *fular* | *galben* | *24* |
| *fes* | *verde* | *25* |
| *fular* | *verde* | *24* |

şi relaţia este:

|  |  |
| --- | --- |
| **denumire** | **culoare** |
| *fular* | *galben* |
| *fes* | *verde* |

atunci este relaţia:

|  |
| --- |
| **preţ** |
| *20* |

1. **Selecţia**

*Selecţia* operează pe o relaţie şi produce o relaţie cu aceleaşi atribute, iar articolele sale sunt cele din care îndeplinesc o condiţie .

De exemplu, dacă relaţia este:

|  |  |  |
| --- | --- | --- |
| **denumire** | **culoare** | **preţ** |
| *fular* | *galben* | *20* |
| *fes* | *verde* | *20* |
| *fular* | *verde* | *19* |
| *fes* | *alb* | *20* |
| *fular* | *galben* | *24* |
| *fes* | *verde* | *25* |
| *fular* | *verde* | *24* |

şi *C* este „valoarea stocată în câmpul *preţ* este cel puţin *22* şi valoarea stocată în câmpul *culoare* este *galben*”, atunci relaţia este:

|  |  |  |
| --- | --- | --- |
| **denumire** | **culoare** | **preţ** |
| *fular* | *galben* | *24* |

1. **Proiecţia**

*Proiecţia* operează pe o relaţie care are *n* atribute cu domeniile de valori A1, A2, ..., An şi generează o relaţie care are *k* atribute () cu domeniile de valori A1, A2, ..., Ak. Articolele din sunt constituite din valorile aflate în în cele *k* atribute, fără dubluri.

De exemplu, dacă relaţia este:

|  |  |  |
| --- | --- | --- |
| **denumire** | **preţ** | **culoare** |
| *fular* | *20* | *galben* |
| *fular* | *24* | *galben* |
| *fular* | *19* | *verde* |
| *fes* | *20* | *alb* |
| *fes* | *20* | *verde* |
| *fes* | *25* | *verde* |
| *fular* | *24* | *verde* |

şi aceasta se proiectează pe atributele *denumire* şi *culoare*, atunci relaţia

este:

|  |  |
| --- | --- |
| **denumire** | **culoare** |
| *fular* | *galben* |
| *fes* | *verde* |
| *fular* | *verde* |
| *fes* | *alb* |

1. **Uniunea**

Considerăm relaţia care are *n* atribute cu domeniile de valori A1, A2, ..., An, relaţia care are *m* atribute cu domeniile de valori B1, B2, ..., Bm şi o condiţie *C* care se referă la valorile stocate în ambele relaţii. Relaţia , *uniunea* celor două, are atribute care au domeniile de valori A1, A2, ..., An, B1, B2, ..., Bm şi articolele sale sunt cele din care îndeplinesc condiţia *C*.

De exemplu, dacă relaţia este:

|  |  |
| --- | --- |
| **denumire** | **culoare** |
| *fular* | *galben* |
| *fes* | *verde* |
| *fular* | *verde* |
| *fular* | *alb* |
| *fes* | *galben* |

relaţia este:

|  |  |
| --- | --- |
| **denumire** | **culoare** |
| *mănuşi* | *galben* |
| *căciulă* | *verde* |

şi condiţia *C* este: „culorile sunt diferite”

atunci relaţia este:

|  |  |  |  |
| --- | --- | --- | --- |
| **denumire** | **culoare** | **denumire** | .**culoare** |
| *fular* | *galben* | *căciulă* | *verde* |
| *fes* | *verde* | *mănuşi* | *galben* |
| *fular* | *verde* | *mănuşi* | *galben* |
| *fular* | *alb* | *mănuşi* | *galben* |
| *fular* | *alb* | *căciulă* | *verde* |
| *fes* | *galben* | *căciulă* | *verde* |

Dacă există în cele două relaţii atribute care au aceleaşi domenii de valori şi dacă uniunea se realizează pentru valori identice ale respectivelor câmpuri, atunci operaţia se numeşte *uniune naturală*. Relaţia care rezultă exclude câmpurile care ar trebui să apară de două ori.

De exemplu, dacă relaţia este:

|  |  |
| --- | --- |
| **denumire** | **culoare** |
| *fular* | *galben* |
| *fes* | *verde* |
| *fular* | *verde* |
| *fular* | *alb* |
| *fes* | *galben* |

şi relaţia este:

|  |  |
| --- | --- |
| **denumire** | **preţ** |
| *fular* | *20* |
| *fes* | *24* |

atunci relaţia este:

|  |  |  |
| --- | --- | --- |
| **denumire** | **culoare** | **preţ** |
| *fular* | *galben* | *20* |
| *fes* | *verde* | *24* |
| *fular* | *verde* | *20* |
| *fular* | *alb* | *20* |
| *fes* | *galben* | *24* |

Considerând relaţia :

|  |  |
| --- | --- |
| **denumire** | **preţ** |
| *fes* | *20* |
| *mănuşi* | *25* |

observăm că valoarea *mănuşi* nu există în prima relaţie şi nici *fular* nu există în a doua relaţie în câmpurile care se presupune că au valori identice. Operaţiile care se pot aplica se numesc:

* *leftjoin* care are drept rezultat relaţia :

|  |  |  |
| --- | --- | --- |
| **denumire** | **culoare** | **preţ** |
| *fular* | *galben* | *NULL* |
| *fes* | *verde* | *20* |
| *fular* | *verde* | *NULL* |
| *fular* | *alb* | *NULL* |
| *fes* | *galben* | *20* |

* *rightjoin* care are drept rezultat relaţia *right*:

|  |  |  |
| --- | --- | --- |
| **denumire** | **culoare** | **preţ** |
| *fes* | *verde* | *20* |
| *fes* | *galben* | *20* |
| *mănuşi* | *NULL* | *25* |

* *fulljoin* care are drept rezultat relaţia :

|  |  |  |
| --- | --- | --- |
| **denumire** | **culoare** | **preţ** |
| *fular* | *galben* | *NULL* |
| *fes* | *verde* | *20* |
| *fular* | *verde* | *NULL* |
| *fular* | *alb* | *NULL* |
| *fes* | *galben* | *20* |
| *mănuşi* | *NULL* | *25* |

**III. Normalizarea bazelor de date relaţionale**

Pe parcursul exploatării unei baze de date relaţionale se pot produce actualizări *corecte*, dar cu efecte nedorite asupra datelor. De exemplu, dacă un furnizor îşi schimbă denumirea, atunci este posibil ca în baza de date care conţine stocul unui magazin să existe produse care să conţină în câmpul *Furnizor* valori diferite, deşi este vorba despre aceeaşi firmă. Astfel de situaţii se numesc *anomalii* şi o bază date bine administrată ar trebui să aibă cât mai puţine anomalii.

*Normalizarea* bazelor de date relaţionale este metoda prin care se scade numărul anomaliilor prin scăderea dependenţei datelor dintr-o relaţie prin micşorarea numărului de câmpuri şi generarea de noi relaţii, dar cu păstrarea tuturor datelor.

Pentru exprimarea dependenţei datelor stocate într-o relaţie se definesc următoarele noţiuni: dependenţă funcţională, dependenţă multi-valoare, dependenţă de tip uniune.

1. **Dependenţa funcţională**

Fie *R* o relaţie şi *X*, *Y* submulţimi disjuncte ale atributelor relaţiei *R*. Spunem că *X* *determină funcţional* *Y*, sau *Y* *depinde funcţional* de *X* dacă orice două articole identice pe *X* sunt identice şi pe *Y*. Notaţia este: .

De exemplu, dacă în departamentul *Management* lucrează persoane specializate (Alina gestionează proiectele informatice şi Bogdan pe cele artistice), atunci în relaţia:

|  |  |  |  |
| --- | --- | --- | --- |
| **salariat** | **început** | **sfârşit** | **tip proiect** |
| Alina | septembrie | decembrie | baze de date |
| Alina | august | noiembrie | magazin virtual |
| Alina | februarie | mai | baze de date |
| Bogdan | martie | mai | vernisaj |
| Bogdan | august | decembrie | spectacol |
| Bogdan | aprilie | noiembrie | vernisaj |

se poate evidenţia dependenţa funcţională:

{*tip proiect*}{*salariat*}

şi se observă că {*început*}{*salariat*}, {*sfârşit*}{*salariat*}.

1. **Dependenţa multi-valoare**

Fie *R* o relaţie şi *X*, *Y* submulţimi disjuncte ale atributelor relaţiei *R*. Spunem că *X* *determină multivaloare* *Y*, sau *Y* *depinde multivaloare* de *X* dacă pentru orice două articole identice pe *X* care diferă pe *Y* există alte două articole care păstrează aceleaşi valori pe *X* şi Y şi care interschimbă valorile pe restul atributelor. Notaţia este: .

De exemplu, dacă o persoană *P1* are două conturi bancare (*Cont1* şi *Cont2*) şi are de plătit lunar două abonamente de telefonie (*Abonament1* şi *Abonament2*), atunci plăţile sale către furnizor pot fi:

|  |  |  |
| --- | --- | --- |
| **nume** | **abonament** | **cont plătitor** |
| *P1* | *Abonament1* | *Cont1* |
| *P1* | *Abonament2* | *Cont2* |
| *P1* | *Abonament1* | *Cont2* |
| *P1* | *Abonament2* | *Cont1* |

În această situaţie, există dependenţele multivaloare:

{*nume*}{*abonament*} şi {*nume*}{*cont plătitor*}

1. **Dependenţa de tip uniune**

Fie *R* o relaţie şi *X*, *Y* submulţimi ale atributelor relaţiei *R*. Spunem că *R* satisface o *dependenţă de tip uniune* dacă . Cu alte cuvinte, *R* se decompune *fără alterarea datelor stocate* în două relaţii, fiecare cu mai puţine atribute.

De exemplu, dacă în departamentul *Management* lucrează trei persoane, la patru proiecte, atunci relaţia *R* este:

|  |  |  |
| --- | --- | --- |
| **salariat** | **termen** | **tip proiect** |
| Alina | decembrie | baze de date |
| Alina | mai | baze de date |
| Bogdan | aprilie | vernisaj |
| Bogdan | noiembrie | vernisaj |
| Clara | decembrie | baze de date |
| Clara | aprilie | vernisaj |

Dacă definim , atunci este:

|  |  |
| --- | --- |
| **salariat** | **termen** |
| Alina | decembrie |
| Alina | mai |
| Bogdan | aprilie |
| Bogdan | noiembrie |
| Clara | decembrie |
| Clara | aprilie |

Dacă definim , atunci este:

|  |  |
| --- | --- |
| **termen** | **tip proiect** |
| decembrie | baze de date |
| mai | baze de date |
| aprilie | vernisaj |
| noiembrie | vernisaj |

Construind acum obţinem relaţia:

|  |  |  |
| --- | --- | --- |
| **salariat** | **termen** | **tip proiect** |
| Alina | decembrie | baze de date |
| Alina | mai | baze de date |
| Bogdan | aprilie | vernisaj |
| Bogdan | noiembrie | vernisaj |
| Clara | decembrie | baze de date |
| Clara | aprilie | vernisaj |

care este exact relaţia iniţială, adică avem că , deci *R* satisface o dependenţă de tip uniune, bazată pe {*salariat*, *termen*} şi {*termen*, *tip proiect*}.

1. **Ierarhia de forme normale**

Normalizarea unei baze de date relaţionale este un proces iterativ, care transformă relaţiile nenormalizate în relaţii care îndeplinesc restricţiile diverselor forme normale. O relaţie în care datele nu au restricţii de dependenţă este *nenormalizată*, sau în *forma normală 0* (FN0, UNF). Un astfel de exemplu este relaţia

*JURNAL(IdStud, NumeStud, IdCarte, Titlu, Autori, Termen, Preţ,*

*IdDomeniu, DenumireDomeniu)*

în care sunt stocate împrumuturile unor cărţi de la Biblioteca Universităţii. Pentru fiecare înregistrare din relaţia *JURNAL* se stochează: codul şi numele studentului care a împrumutat, codul, titlul, autorii, preţul cărţii împrumutate, codul şi denumirea domeniul din care face parte cartea. Cheia primară este alcătuită din două câmpuri: *IdStud* şi *IdCarte*, deoarece o carte poate fi împrumutată succesiv de mai mulţi studenţi şi un student poate împrumuta simultan mai multe cărţi. Această relaţie este nenormalizată, deoarece nu s-au impus restricţii privind dependenţele datelor.

**Forma normală 1 (FN1)**

O relaţie este în *Forma Normală 1* dacă toate valorile stocate sunt atomice. De exemplu, relaţia *JURNAL* nu este în *FN1* deoarece valorile stocate în câmpul *Autori* pot conţine mai multe şiruri de caractere. Pentru a o aduce în *FN1*, relaţia se poate transforma astfel (considerăm că se vor stoca doar primii 3 autori, dacă există mai mulţi):

*JURNAL(IdStud, NumeStud, IdCarte, Titlu, Autor1, Autor2, Autor3, Termen, Preţ, IdDomeniu, DenumireDomeniu)*

**Forma normală 2 (FN2)**

O relaţie este în *Forma Normală 2* dacă este în FN1 şi toate atributele non-cheie depind funcţional *total* de cheie. De exemplu, relaţia precedentă nu este în *FN2* deoarece valorile stocate în câmpul *NumeStud* depind parţial de cheie (mai exact, depind doar de *IdStud*). La fel, *Titlu*, *Autor1*, *Autor2*, *Autor3*, *Preţ*, *IdDomeniu, DenumireDomeniu* depind parţial de cheie, deoarece depind doar de *IdCarte*. Trecerea în FN2 se face prin definirea unor noi relaţii:

*JURNAL(IdStud, IdCarte, Termen)*

*STUDENTI(IdStud, NumeStud)*

*CARTI( IdCarte, Titlu, Autor1, Autor2, Autor3, Preţ, IdDomeniu, DenumireDomeniu)*

Toate aceste relaţii sunt în FN2.

**Forma normală 3 (FN3)**

O relaţie este în *Forma Normală 3* dacă este în FN2 şi toate atributele non-cheie sunt independente funcţional. De exemplu, relaţia *CARTI* nu este în *FN3* deoarece valorile stocate în câmpul *DenumireDomeniu* depind de *IdDomeniu*, care nu face parte din cheia relaţiei. Trecerea în FN3 se face prin definirea unei noi relaţii:

*JURNAL(IdStud, IdCarte, Termen)*

*STUDENTI(IdStud, NumeStud)*

*CARTI( IdCarte, Titlu, Autor1, Autor2, Autor3, Preţ, IdDomeniu)*

*DOMENII( IdDomeniu, DenumireDomeniu)*

Toate aceste relaţii sunt în FN3.

**IV. Tranzacţii**

Pe parcursul exploatării unei baze de date se pot produce evenimente nedorite, numite *anomalii*, care pot altera corectitudinea datelor stocate. De exemplu, deoarece datele sunt accesate simultan de mai mulţi utilizatori, deşi fiecare utilizator interacţionează corect cu baza de date, o gestionare defectuoasă a cererilor *simultane* de acces poate conduce la pierderea integrităţii datelor. Tranzacţiile reprezintă metoda prin care se asigură excluderea anomaliilor şi se menţine consistenţa (integritatea) datelor .

O *tranzacţie* este unitatea atomică de prelucrare a datelor, care duce o bază de date dintr-o stare consistentă în altă stare consistentă. La încheierea unei tranzacţii, baza de date fie trece în altă stare consistentă (tranzacţia *se realizează cu succes*), fie se întoarce în starea în care era la iniţierea tranzacţiei (tranzacţia *se anulează*).

Considerăm că o bancă eliberează unei familii două carduri cu acces la un cont comun *C*. Mai presupunem că se încearcă simultan alimentarea cu numerar a contului, de la două ATM-uri. În tabelele următoare, săgeata arată evoluţia temporală a etapelor celor două tranzacţii. Dacă tranzacţiile sunt serializate (adică a doua începe după ce prima s-a încheiat), atunci suma din *C* creşte în mod corect cu 200 lei.

|  |  |  |
| --- | --- | --- |
| **Tranzacţia 1** | **Tranzacţia 2** | **Timp** |
| read *C* |  |  |
| *C* = *C* + 100 |  |
| write *C* |  |
|  | read *C* |
|  | *C* = *C* + 100 |
|  | write *C* |

Dacă însă etapele se intercalează, atunci se pot înregistra valori incorecte în cont. De exemplu, în situaţia următoare, valoarea din C creşte cu 100 lei. Această situaţie se numeşte *pierderea actualizării*. În acest caz, actualizarea care ar fi trebuit realizată de prima tranzacţie este ignorată.

|  |  |  |
| --- | --- | --- |
| **Tranzacţia 1** | **Tranzacţia 2** | **Timp** |
| read *C* |  |  |
| *C* = *C* + 100 |  |
|  | read *C* |
|  | *C* = *C* + 100 |
| write *C* |  |
|  | write *C* |

O altă anomalie poate apărea când o tranzacţie este anulată, dar o altă tranzacţie este în evoluţie. În exemplul următor, valoarea din *C* creşte cu 200 lei, deşi ar trebui să crească doar cu 100 lei. Această situaţie se numeşte *citire improprie* (dirty read).

|  |  |  |
| --- | --- | --- |
| **Tranzacţia 1** | **Tranzacţia 2** | **Timp** |
| read *C* |  |  |
| *C* = *C* + 100 |  |
| write *C* |  |
|  | read *C* |
|  | *C* = *C* + 100 |
|  | write *C* |
| cancel |  |

O anomalie fără efecte asupra valorii stocate în *C* este prezentată în continuare. Dacă o tranzacţie se întrerupe şi se reia ulterior, este posibil să citească din *C* valori diferite. Această situaţie se numeşte *citire irepetabilă*.

|  |  |  |
| --- | --- | --- |
| **Tranzacţia 1** | **Tranzacţia 2** | **Timp** |
| read *C* |  |  |
|  | read *C* |
|  | *C* = *C* + 100 |
|  | write *C* |
| read *C* |  |

Resursele *critice* (resurse *în conflict*) ale unei baze de date sunt datele care sunt accesate simultan de mai multe tranzacţii, cel puţin un acces fiind pentru scriere. Pentru păstrarea integrităţii datelor, se folosesc metodele de control al concurenţei: prin blocare sau prin marca de timp.

1. **Controlul concurenţei prin blocare**

Controlul concurenţei *prin blocare* asociază fiecărui articol o variabilă binară care reţine dacă acesta este blocat sau nu. Regulile care permit controlul prin blocare sunt:

* o tranzacţie blochează articolul înainte de a-l accesa;
* o tranzacţie deblochează articolul după ce a efectuat toate accesările asupra lui;
* o tranzacţie nu poate bloca un articol blocat;
* o tranzacţie nu poate debloca un articol blocat de altă tranzacţie;
* blocarea şi deblocarea sunt acţiuni indivizibile.

Controlul prin blocare poate să genereze următoarele situaţii:

* *interblocare* (deadlock): două tranzacţii se împiedică reciproc. De exemplu, fiecare dintre tranzacţiile următoare încearcă să blocheze o resursă deja blocată de cealaltă şi niciuna nu mai poate înainta:

|  |  |  |
| --- | --- | --- |
| **Tranzacţia 1** | **Tranzacţia 2** | **Timp** |
| lock *A* |  |  |
|  | lock *B* |
| lock *B* |  |
|  | lock *A* |
|  |  |

În acest caz, una dintre tranzacţii este anulată şi cealaltă îşi poate continua acţiunile.

* *aşteptare indefinită* (livelock): o tranzacţie este permanent împiedicată de alte tranzacţii care încep şi se termină, dar ocupă permanent resursa aşteptată de prima. În acest caz, se instituie un sistem de priorităţi, de exemplu vechimea tranzacţiei şi încurajarea celor mai vechi.

1. **Controlul concurenţei prin marcă de timp**

Controlul concurenţei *prin marcă de timp* asociază fiecărei tranzacţii *t* o marcă de timp (time stamp) *TS(t)*, care este momentul iniţierii sale. Fiecărui articol *x* i se ataşează două mărci: *readTS(x)*, care este marca celei mai noi tranzacţii care l-a citit şi *writeTS(x)*, care este marca celei mai noi tranzacţii care l-a scris. Strategia acestei metode este încurajarea tranzacţiilor mai noi. Aceasta se realizează prin următoarele reguli:

* dacă *t* încearcă să scrie articolul *x*, atunci:
  + dacă *readTS(x) > TS(t)* sau *writeTS(x) > TS(t)*, atunci *t* se abandonează, pentru că valoarea care ar trebui să se scrie în *x* este deja perimată;
  + altfel, *t* scrie în *x* şi setează *writeTS(x) = TS(t)*.
* dacă *t* încearcă să citească articolul *x*, atunci:
  + dacă *writeTS(x) > TS(t)*, atunci *t* se abandonează, pentru că valoarea care ar trebui să se citească din *x* este deja perimată;
  + altfel, *t* citeşte articolul *x* şi setează *readTS(x) = max {TS(t), readTS(x)}*
* o tranzacţie abandonată se relansează ulterior (deci va avea o marcă de timp mai mare), deci va avea mai multe şanse să fie finalizată cu succes.

**V. SQL**

*SQL* (Structured Query Language) este limbajul standard pentru realizarea sistemelor de gestiune a bazelor de date relaţionale (SGBDR). Standardul mai este cunoscut ca ANSI-SQL, deoarece a fost iniţiat şi dezvoltat de ANSI (American National Standards Institute). În deceniul 1970-1979 au apărut primele implementări ale sistemelor de gestiune a bazelor de date, care apoi au fost standardizate în deceniul următor. Cea mai recentă versiune ANSI-SQL datează din 2016. SQL este un limbaj de programare declarativ, care descrie *logica* programului (ce trebuie făcut), fără a descrie *fluxul de control* (cum trebuie făcut).

Dezvoltatorii software produc propriile SGBDR, care implementează cel puţin standardul SQL actual. De exemplu, *Oracle* oferă produsul *Database*, care include: *Oracle/SQL* (limbajul de programare), *SQL\*Plus* (interfaţa grafică) şi *PL/SQL* (o extensie procedurală a SQL). *Microsoft* oferă *Access* (parte a pachetului *Office*) şi *SQL Server*, iar *IBM* dezvoltă *DB2*. Dintre sistemele open-source, *MySQL* (dezvoltat de Oracle) este cel mai utilizat. Producătorii pot propune facilităţi suplimentare standardului SQL (de exemplu, tipuri proprii de date) şi de obicei au propriile:

* mesaje de eroare;
* interfeţe;
* metode de iniţializare, deschidere, conectare la baza de date.

Tipurile *standard* de date includ următoarele:

* şir de caractere:
  + de lungime fixă *n*: *CHAR(n)*;
  + de lungime variabilă *n*: *VARCHAR(n)*;
* numere întregi: *INTEGER*;
* numere reale: *DOUBLE*;
* date calendaristice şi valori de timp: *DATE*, *TIME*.

Una dintre dificultăţile SQL este evaluarea condiţiilor care implică şi valori *NULL*. În aceste cazuri, logica devine trivalentă, deoarece valorile de adevăr sunt *Adevărat* (A), *Fals* (F) şi *Necunoscut* (N - NULL). Evaluarea operatorilor logici *negaţie*, *sau*, *şi*, *implicaţie*, *echivalenţă* se realizează conform *Tabelului 1*.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *p* | *q* | *negaţie* | *sau* | *şi* | *implică* | *echivalenţă* |
| A | A | F | A | A | A | A |
| A | N | F | A | N | N | N |
| A | F | F | A | F | F | F |
| N | A | N | A | N | A | N |
| N | N | N | N | N | N | N |
| N | F | N | N | F | N | N |
| F | A | A | A | F | A | F |
| F | N | A | N | F | A | N |
| F | F | A | F | F | A | A |

*Tabelul 1. Operatorii logici în logica trivalentă*

Modelul aritmetic al evaluărilor în logica trivalentă foloseşte alocarea unor valori numerice pentru valorile de adevăr:

* 1 pentru *adevărat*;
* 0.5 pentru *necunoscut* (NULL);
* 0 pentru *fals*

şi alocă următoarele expresii aritmetice pentru operatorii logici:

* pentru ;
* pentru ;
* pentru ;
* pentru ;
* pentru .

**Comenzi SQL**

Limbajul SQL este compus din următoarele sub-limbaje:

* limbajul de definire a datelor (LDD): comenzile care definesc, actualizează şi şterg obiectele bazei de date: CREATE, ALTER, RENAME, TRUNCATE, DROP;
* limbajul de manevră a datelor (LMD): comenzile care introduc, actualizează, şterg, vizualizează datele stocate: INSERT, UPDATE, DELETE, SELECT;
* limbajul de control al datelor (LCD): comenzile care acordă şi ridică privilegiile de acces la obiectele bazei de date: GRANT, REVOKE;
* limbajul de control al tranzacţiilor (LCT): comenzile care finalizează o tranzacţie prin succes sau eşec: COMMIT, ROLLBACK, SAVEPOINT.

Sintaxa acestor comenzi este descrisă în continuare, pe baza următoarelor convenţii de scriere:

* cuvintele scrise cu majuscule sunt cuvinte-cheie ale SQL
* restul cuvintelor se înlocuiesc cu elementul de limbaj indicat
* [element] înseamnă că element este opţional
* {element\_1 | element\_2} înseamnă că se alege una dintre opţiuni
* [, element ...] înseamnă că element se poate repeta indefinit şi separarea elementelor se face prin virgulă
* ( şi ) sunt obligatorii

CREATE TABLE nume\_tabel

(nume\_coloană tip\_date [constrângere] [, nume\_coloană tip\_date [constrângere] ...])

unde *constrângere* poate fi NOT NULL

DEFAULT valoare

UNIQUE

CHECK condiţie

PRIMARY KEY

REFERENCES alt\_tabel(cheie\_primară)

Crearea unei tabele noi, câmpurile sunt specificate în ordine prin nume, tip de date şi constrângeri. NOT NULL împiedică stocarea articolelor care nu conţin valoare; DEFAULT stochează valoarea specificată dacă utilizatorul nu setează o valoare; UNIQUE împiedică dublarea valorilor; CHECK împiedică stocarea artoclelor care nu îndeplinesc condiţia specificată, PRIMARY KEY setează cheia primară, REFERENCES setează o cheie secundară prin specificarea unei legături cu un câmp cheie primară din alt tabel.

ALTER TABLE nume\_tabel parametri

unde *parametri* pot fi {ADD | MODIFY} coloană (tip\_date)

RENAME COLUMN coloană\_veche TO coloană\_nouă

DROP COLUMN coloană

ADD CONSTRAINT nume\_constrângere constrângere

DROP CONSTRAINT nume\_constrângere

Modifcarea structurii unui tabel, prin adăugarea unei coloane cu nume şi tip specificate, modificarea tipului de date pentru o coloană existentă, redenumirea unei coloane, ştergerea unei coloane, adăugarea unei constrângeri sau ştergerea unei constrângeri.

RENAME TABLE nume\_tabel\_vechi TO nume\_tabel\_nou

Redenumirea unui tabel

TRUNCATE TABLE nume\_tabel

Ştergerea tuturor articolelor din tabel

DROP TABLE nume\_tabel

Ştergerea tabelului

SELECT [DISTINCT] { \* | {coloană | expresie} [, {coloană | expresie} ...] }

[INTO tabel\_nou]

FROM nume\_tabel

[WHERE condiţie]

[GROUP BY coloană [, coloană] [HAVING condiţie\_de\_grup] ]

[ORDER BY criteriu [ASC | DESC } [, criteriu [ ASC | DESC ] ...] ]

Afişarea datelor din tabelul specificat în clauza FROM;

DISTINCT interzice dublurile rândurilor afişate;

\* afişează valorile din toate câmpurile; alternativa este să se specifice câmpurile sau expresii construite cu câmpurile tabelului;

INTO generează un nou tabel care va conţine rezultatul comenzii;

WHERE reduce rândurile afişate doar la cele care îndeplinesc condiţia specificată;

GROUP BY grupează articolele care sunt identice pe coloanele specificate;

HAVING reduce rândurile afişate doar la cele care corespund grupurilor care îndeplinesc condiţia de grup specificată;

ORDER BY produce sortarea după criteriilor specificate.

INSERT INTO nume\_tabel [coloană [, coloană ...] ) ]

{ VALUES (valoare [, valoare ...] ) | *o comandă SELECT validă*}

Inserează în tabel un articol cu valorile specificate, care trebuie să respecte ordinea coloanelor; alternativa este să se insereze un grup de articole care este rezultatul unei alte comenzi SELECT.

UPDATE nume\_tabel SET coloană = expresie [, coloană = expresie ...]

[WHERE condiţie]

Actualizează valorile stocate în coloanele specificate conform expresiilor specificate;

WHERE reduce rândurile modificate doar la cele care îndeplinesc condiţia specificată.

DELETE FROM nume\_tabel

[WHERE condiţie]

Şterge articolele din tabel;

WHERE reduce rândurile şterse doar la cele care îndeplinesc condiţia specificată.

GRANT {privilegiu [, privilegiu ...] | ALL PRIVILEGES}

ON nume\_tabel

TO {PUBLIC | cont\_utilizator | rol}

[ WITH GRANT OPTION]

unde *rol* este un set de privilegii

*privilegiu* {CREATE | ALTER | DROP | SELECT | INSERT | DELETE | UPDATE | EXECUTE}

Acordă privilegiile specificate sau toate privilegiile privind un tabel către toţi utilizatorii, sau unui singur cont, sau unui grup de utilizatori;

WITH GRANT OPTION permite celor care au primit privilegiile să le delege la rândul lor.

REVOKE {privilegiu [, privilegiu ...] | ALL PRIVILEGES}

ON nume\_tabel

TO {PUBLIC | cont\_utilizator | rol}

Ridică privilegiile specificate sau toate privilegiile privind un tabel pentru toţi utilizatorii, sau pentru un singur cont, sau pentru un grup de utilizatori.

COMMIT

Finalizarea cu succes a tranzacţiei curente.

SAVEPOINT nume\_punct\_salvare

Setarea unui punct de salvare în evoluţia tranzacţiei curente.

ROLLBACK [TO nume\_punct\_salvare]

Anularea tranzacţiei curente;

TO permite întoarcerea la punctul de salvare specificat.

**Funcţii de grup**

Funcţiile de grup permit agregarea datelor dintr-un grup de articole. Aceste funcţii returnează pentru câmpul specificat, pentru fiecare grup:

* COUNT (câmp): numărul de valori diferite;
* MIN(câmp): cea mai mică valoare;
* MAX(câmp): cea mai mare valoare;
* AVG(câmp): media aritmetică a valorilor;
* SUM(câmp): suma valorilor.

Relaţia utilizată în exemplificarea utilizării comenzilor SQL este EMP, care stochează următoarele articole:

EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO

----- ------ ------- ---- --------- -------- ------- ------

7369 SMITH CLERK 7902 13-JUL-83 800.00 20

7499 ALLEN SALESMAN 7698 15-AUG-83 1,600.00 300.00 30

7521 WARD SALESMAN 7698 26-MAR-84 1,250.00 500.00 30

7566 JONES MANAGER 7839 31-OCT-83 2,975.00 20

7654 MARTIN SALESMAN 7698 05-DEC-83 1,250.00 1.400.00 30

7698 BLAKE MANAGER 7839 11-JUN-84 2,850.00 30

7782 CLARK MANAGER 7839 14-MAY-84 2,450.00 10

7788 SCOTT ANALYST 7566 05-MAR-84 3,000.00 20

7839 KING PRESIDENT 09-JUL-84 5,000.00 10

7844 TURNER SALESMAN 7698 04-JUN-84 1,500.00 .00 30

7876 ADAMS CLERK 7788 04-JUN-84 1,100.00 20

7900 JAMES CLERK 7698 23-JUL-84 950.00 30

7902 FORD ANALYST 7566 05-DEC-83 3,000.00 20

Exemple de utilizare ale comenzilor SQL prezentate:

1. Afişarea funcţiilor, fără repetiţie.

SELECT DISTINCT JOB FROM EMP;

1. Afişarea numelor angajaţilor care lucrează în departamentul cu codul 30.

SELECT ENAME FROM EMP WHERE DEPTNO=30;

1. Afişarea numelor celor care au comision mai mare decât 500.

SELECT ENAME FROM EMP WHERE COMM>500;

1. Afişarea codulrilor şefilor şi numărul de subalterni ai fiecăruia.

SELECT MGR, COUNT(\*) FROM EMP GROUP BY MGR;

1. Afişarea numelor analiştilor şi a salariilor lor anuale.

SELECT ENAME, SAL\*12 FROM EMP WHERE JOB=”ANALYST”;

1. Afişarea în ordine alfabetică a numelor celor care nu au comision.

SELECT ENAME FROM EMP WHERE COMM IS NULL ORDER BY ENAME;

1. Introducerea unei noi coloane în structura tabelului EMP, cu numele CHILDREN, tip de date INTEGER.

ALTER TABLE EMP ADD CHILDREN (INTEGER);

1. Setarea valorii 0 pentru câmpul CHILDREN în tot tabelul.

UPDATE EMP SET CHILDREN=0;

1. Creşterea cu 10% a salariilor pentru cei care au un salariu de cel mult 1000.

UPDATE EMP SET SAL=SAL\*1.1 WHERE SAL<=1000;

1. Ştergerea salariaţilor care au comision 0.

DELETE FROM EMP WHERE COMM=0.0;

**VI. Baze de date distribuite**

*Bazele de date distribuite* accesează date aflate în diverse puncte geografice, dar care logic sunt interogate şi actualizate ca o singură colecţie de date. Dificultăţile suplimentare induse de distribuţia geografică a datelor sunt:

* necesitatea toleranţei sistemului la căderile individuale ale componentelor;
* caracteristicile hardware ale componentelor nu sunt cunoscute şi se pot schimba în timp;
* sistemele de calcul care le întreţin au vederi incomplete asupra datelor, de obicei pot accesa doar datele stocate local;
* nesiguranţa liniilor de comunicaţie;

Prima aplicaţie distribuită a fost *poşta electronică* (email), care a apărut în 1970. Acum aplicaţiile distribuite sunt omniprezente: telefonia (fixă, mobilă), reţelele de calculatoare, comunităţile virtuale, jocurile online (*massively multiplayer online games* – MMOG), sistemele de control aeronautic, calcul pe grid, calcul în cloud, internetul lucrurilor (*internet of things* – IoT), calcul omniprezent (*ubiquitous computing*).

Stocarea datelor în bazele de date distribuite se poate face:

* în fragmente:
  + pe câmpuri;
  + pe articole;
* prin replicare (copii totale, *mirrors*), pentru a asigura echilibrarea încărcării reţelei;
* mixte: unele obiecte sunt fragmentate, altele sunt replicate.

Arhitecturile bazelor de date distribuite sunt:

* de tip *client-server*, când clientul trimite interogarea tuturor serverelor şi acumulează răspunsurile;
* de tip *server colaborativ*, când clientul trimite interogarea serverului apropiat, care o execută colaborând cu celelalte servere şi trimite răspunsul clientului.

Tranzacţiile asigură o recuperare corectă a datelor după căderile sistemului de gestiune, când aplicaţiile bazei de date distribuite s-au oprit în stări incorecte ale datelor. Recuperarea datelor presupune întoarcerea la ultima stare cu date corecte (după ce tranzacţia precedentă s-a încheiat). Repartizarea datelor la distanţă este un element suplimentar de dificultate pentru gestionarea tranzacţiilor.

Metode de asigurare a consistenţei datelor în bazele de date distribuite:

* *pesimiste*: tranzacţia blochează resursele critice, lipsind celelalte tranzacţii de accesul la aceste resurse. *Nivelurile de blocaj* (care descriu *granularitatea* blocării) sunt:
  + *înalt*: blocarea întregului tabel; în cazul accesului masiv, blocajul înalt încetineşte accesul la date;
  + *scăzut*: se blochează doar articolul care se scrie; în cazul în care clientul comandă o scriere multiplă, tranzacţia care blochează va dura destul de mult;
  + *intermediar*: se blochează un grup de articole.
* *optimiste*: resursele conflictuale nu sunt blocate, dar li se setează un indicator de conflict. Dacă la sfârşitul tranzacţiei indicatorul de conflict este setat, atunci tranzacţia se anulează. În cazul magazinelor virtuale, aceste strategii sunt eficiente, deoarece conflictele sunt rare şi un mecanism de blocare ar mări nejustificat timpul de execuţie a tranzacţiilor.

O tranzacţie distribuită presupune actualizarea datelor din mai multe baze de date aflate la distanţă. O astfel de tranzacţie presupune actualizarea datelor din toate bazele de date. Serverul care iniţiază tranzacţia distribuită se numeşte *coordonator*. Protocolul care asigură încheierea tranzacţiei distribuite are două faze:

* *votare*: coordonatorul cere pregătirea celorlalte servere; acestea transmit coordonatorului posibilitatea actualizării datelor stocate local şi blochează resursele aflate în proces de actualizare;
* *încheiere*: dacă toate serverele au acceptat tranzacţiile, atunci coordonatorul încheie cu succes propria tranzacţie şi cere realizarea tranzacţiilor locale. Altfel, coordonatorul renunţă la propria tranzacţie şi transmite anularea tuturor celorlalte tranzacţii.

**Data mining**

*Data mining* este activitatea de extragere de informaţii (şabloane) din date. La momentul actual, există colecţii mari de date (baze de date, depozite de date) care se pretează explorării electronice. Aplicaţiile de data mining asigură:

* *clasificarea* datelor: împărţirea datelor în categorii predefinite;
* *clustering*: găsirea grupurilor care să asigure o împărţire cât mai eficientă a datelor în acele grupuri;
* *regresia*: găsirea unei funcţii care să asigure modelarea datelor cu o eroare cât mai mică;
* *asocierea de reguli*: găsirea de legături între date.

**Data fishing**

*Data fishing* este varianta metodelor pentru data mining aplicate colecţiilor mici de date, care nu sunt potrivite aplicării legilor statistice.

**VII. Stocarea masivă**

*Stocarea masivă* se referă la stocarea datelor pentru accesarea concurentă de către sisteme eterogene de calcul. Volumele actuale de date devin din ce în ce mai mari; de aceea este necesar ca datele să se acceseze rapid şi automat. De exemplu, astfel de sisteme de fişiere sunt: *NTFS* pentru *Windows* sau *Btrfs* pentru *Linux*. Problema stocării masive poate fi abordată pentru reţele de calculatoare sau pentru centrele de date.

1. **Stocarea masivă la nivelul unei reţele de calculatoare**

Cea mai utilizată soluţie pentru o reţea de calculatoare este sistemul *NAS* (*Network-Attached Storage*), care conţine:

* unităţi de stocare (HDD - hard disk drive, SSD – solid state drive), care constituie componenta hardware;
* o organizare logică în *RAID* (*Redundant Array of Independent Disks*), care permite acces rapid şi oferă redundanţă pentru protecţia la căderea unităţilor de stocare.

În funcţie de gradul de redundanţă avut în vedere, cele mai utilizate organizări RAID sunt *nivelurile*:

* *RAID 0*, care repartizează datele pe mai multe suporturi de memorie, având redundanţă zero. O defecţiune duce la pierderea ireversibilă a datelor.
* *RAID 1*, care oferă multiplicarea integrală (*mirror*) a datelor pe două discuri. Dacă un disc se defectează, atunci datele se pot accesa de pe celălalt disc.
* *RAID 5*, care repartizează blocuri de date şi asigură recuperarea datelor în cazul defectării unui suport de memorie. Pentru a îndeplini această condiţie, sunt necesare cel puţin 3 discuri. În cazul general (stocare pe *n* discuri), blocurile de date se repartizează ciclic pe discuri şi celălalt este destinat stocării parităţilor la nivel de bit. Un exemplu pentru 4 discuri este prezentat în continuare.
* *RAID 6*, care repartizează blocuri de date şi asigură recuperarea datelor în cazul defectării simultane a două discuri. Pentru a îndeplini această condiţie, sunt necesare cel puţin 4 discuri.

*Exemplu de stocare la nivel RAID 5 pe 4 discuri*

Blocuri consecutive de date se stochează pe discuri diferite. Pentru fiecare set de 3 blocuri stocate pe 3 discuri, pe al patrulea disc se stochează parităţile la nivel de bit (adică se aplică operatorul pe biţi *XOR* – *sau exclusiv*). De exemplu, dacă blocurile consecutive de date dintr-un fişier sunt notate A, B, ... şi parităţile cu P1, P2, ..., atunci stocarea se realizează ca în Figura 1.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *disc 1* |  | *disc 2* |  | *disc 3* |  | *disc 4* |
| A |  | B |  | C |  | P1 |
| D |  | E |  | P2 |  | F |
| G |  | P3 |  | H |  | I |
| P4 |  | J |  | K |  | L |
| M |  | N |  | O |  | P5 |
| ... |  | ... |  | ... |  | ... |

*Fig. 1. Stocarea blocurilor consecutive de date pe 4 discuri în nivelul RAID 5.*

Dacă presupunem că un bloc de date este format dintr-un octet şi că:

* A este octetul 01001110
* B este octetul 11001000
* C este octetul 00010011

atunci P1 este octetul 01001110 *XOR* 11001000 *XOR* 00010011 = 10010101.

În cazul defectării unui singur disc, formulele de refacere a datelor sunt:

* A = P1 *XOR* B *XOR* C
* B = P1 *XOR* A *XOR* C
* C = P1 *XOR* A *XOR* B
* P1 = A *XOR* B *XOR* C

De exemplu, dacă se defectează *discul 1*, atunci octetul A care a fost pierdut se obţine prin efectuarea calculului:

10010101 *XOR* 11001000 *XOR* 00010011 = 01001110

Dacă în timpul refacerii unui disc se mai defectează încă unul, atunci datele sunt pierdute definitiv.

1. **Stocarea masivă în centrele de date**

Volumele mari de date (*Big Data*) se stochează în centre de date repartizate pe întreg globul pământesc. Marile companii din industria IT (Google, Facebook, Amazon, etc.) construiesc şi întreţin astfel de centre de date, care includ sisteme de calcul, de stocare şi de comunicaţii, precum şi sisteme auxiliare pentru alimentare cu energie, pentru răcire şi protecţie. Eficienţa unui centru de date este dată de *eficienţa energetică* (*PUE* - Power Usage Effectiveness):

Viteza de regăsire a datelor este cel mai important parametru al unui centru de date. Căutarea datelor se face pe baza *tabelelor hash* (hash tables, hash maps). Un *tabel hash* este o structură care pune în corespondenţă o mulţime de chei *K* cu o mulţime de valori *V*, pe baza unei funcţii *h*. Valorile date de o astfel de funcţie trebuie să se calculeze rapid, pentru a permite stocarea/regăsirea rapidă a datelor asociate cheii. În mod ideal, funcţia hash este injectivă; există însă cazuri când pentru două chei diferite asociază aceeaşi valoare, producându-se astfel o *coliziune*. Un exemplu de coliziune în agenda telefonică este tentativa de stocare a unei noi persoane care are acelaşi nume cu o altă persoană deja aflată în agendă. Coliziunile se rezolvă prin:

* extinderea tabelei hash cu liste înlănţuite; această metodă necesită memorie suplimentară;
* stocarea noii valori în următoarele poziţii din tabelă, dacă există spaţiu nealocat; această metodă duce la umplerea rapidă a staţiului disponibil.

Într-o tabelă hash se execută doar trei operaţii: inserare, ştergere, citire. Fiecare operaţie necesită calcularea valorii funcţiei hash şi actualizarea valorilor stocate. Cum funcţia se calculează uşor, operaţiile se execută *aproape* de fiecare dată (în lipsa colizunilor) instantaneu.

De exemplu, *Google* foloseşte pentru indexarea web-ului stocarea pe niveluri, dată de sistemul proprietar *Bigtable*.

**VIII. NoSQL**

Aşa cum am văzut, SQL este limbajul dedicat realizării sistemelor de gestiune ale bazelor de date relaţionale tradiţionale. S-a observat că problemele actuale legate de gestiunea datelor au soluţii mai eficiente dacă nu se utilizează bazele de date relaţionale. A apărut astfel abordarea *NoSQL*, care nu înseamnă *No SQL*, ci *Not Only SQL*.

Un exemplu de proprietate a sistemelor de gestiune a bazelor de date relaţionale care nu mai poate fi îndeplinită este integritatea datelor, deoarece unele date pot conţine erori, sau tipurile de date nu mai sunt atât de restrictive. Volumele foarte mari de date care se colectează în bazele de date actuale nu mai pot fi prelucrate astfel încât să se furnizeze rezultate exacte. Este de dorit ca în ipoteza pierderii accesului la unele sisteme care depozitează date să nu se refuze prelucrările, ci să se furnizeze rezultate aproximative.

De asemenea, date semi-structurate, cum sunt textele din bibliotecile virtuale sau enciclopedii/dicţionare pot fi gestionate cu astfel de sisteme. Volumul mare de date care nu au tipuri de date stricte nu permit procesări SQL eficiente.

Nebazându-se doar pe SQL, sistemele de gestiune bazate pe NoSQL pot beneficia de programarea tradiţională (procedurală) a aplicaţiilor.

**Abordări NoSQL**

*MapReduce* (*Google, Hadoop*)

Datele sunt stocate în fişiere. Programatorii scriu funcţiile specifice, care iniţial sunt doar interfeţe: *map*, *reduce*, *reader*, *writer*, *combiner*. Sistemul oferă un mediu tolerant la erori, care permite manevra volumelor mari de date. Dacă se doreşte un volum mai mic al erorilor, atunci se pot utiliza intrumente precum *Hive* (care structurează datele) sau *Pig* (care lucrează imperativ).

*Bigtable (Google)*

Datele sunt perechi (*key*, *value*), iar operaţiile de bază sunt *insert (key, value), fetch (key), update (key), delete (key)*. Datele sunt distribuite pe bază de cheie în sistemele de stocare.

*MongoDB (opensource)*

Datele sunt perechi (*key*, *document*), iar operaţiile operaţiile de bază sunt *insert (key, document), fetch (key), update (key), delete (key)*. Documentele pot fi semi-structurate, de exemplu în limbajul *XML* (eXtensible Markup Language).

*Neo4j (opensource)*

Datele sunt noduri şi muchii ale grafurilor, cu informaţii ataşate. Codul sursă este în Java.

**Big Data**

*Big Data* desemnează colecţii de date care nu se mai pot gestiona în mod tradiţional. Societatea produce date în ritm accelerat: reţelele de socializare, sistemele industriale sau meteorologice de supraveghere şi control sunt surse de date de mare complexitate, care sunt dificil de procesat cu metodele şi limbajele specifice secolului trecut.

Prin Big Data se înţeleg datele de volum mare, viteză mare de creare şi diversitate mare, care necesită noi forme de prelucrare pentru a permite luarea deciziilor în timp corespunzător, extragerea de informaţii şi optimizarea proceselor. O caracterizare a Big Data se referă la *4V*: *volum*, *viteză*, *varietate*, *veridicitate*.

Sistemele de prelucrare pentru Big Data sunt cunocute sub denumirea generică *New SQL*, care adaugă la NoSQL securitatea tranzacţiilor, deoarece unele domenii necesită această caracteristică (sistemele financiar-bancare, de exemplu).

**Resurse disponibile online**

1. <http://cursuri.cs.pub.ro/~radulescu/bd/sql7/>
2. <http://cursuri.cs.pub.ro/~radulescu/bd/sql9/>
3. <https://livesql.oracle.com/apex/f?p=590:1000>::::::
4. <https://ocw.mit.edu/courses/find-by-topic/#cat=engineering&subcat=computerscience>
5. <https://online.stanford.edu/courses/soe-ydatabases-databases>
6. <https://online-learning.harvard.edu/subject/sql>
7. <https://www.coursera.org/courses?query=database>
8. <https://www.edx.org/learn/databases>
9. <https://www.ibm.com/support/knowledgecenter/en/ssw_ibm_i_73/rzahg/rzahgdb.htm>
10. <https://www.oracletutorial.com/>
11. <https://www.w3resource.com/sql/tutorials.php>