5.3 Modificarea datelor în SQL

Pentru modificarea conţinutului unei baze de date SQL pune la dispoziţie instrucţiunile *insert*, *delete* şi *update*.

5.3.1 Inserări în baza de date

Sintaxa instrucţiunii *insert* este:

insert into *NumeTabel* [(*ListaAtribute*)] <values (*ListaValori*) | *InterogareSQL*>

Prima variantă permite inserarea unei singure linii în tabele. Argumentul clauzei *values* reprezintă valorile atributelor pentru linia inserată.

Exemplu

```
insert into Departament (Dept, Oras) values (’Producţie’, ’Suceava’)
```

A doua variantă permite adăugarea unei mulţimi de linii, care sunt extrase mai întâi din baza de date.

Exemplu

```
insert into Departamentelasi (select Dept, Adresa
                              from Departament
                              where Oras = ’Iasi’)
```

Comanda anterioară inserează în tabelul *Departamentelasi* liniile din tabelul *Departament* care au *Iasi* ca valoare a atributului *Oras*.

---

insert into *NumeTabel* [(*ListaAtribute*)] <values (*ListaValori*) | *InterogareSQL*>

Dacă valorile pentru anumite atribute nu sunt specificate în momentul inserării, se vor asigna valori implicite sau valori NULL în cazul în care nu sunt definite valori implicite.

Corespondenţa între atributele tabelului şi valorile ce urmează a fi inserate este dictată de ordinea în care termenii apar în definiţia tabelului.

Astfel primul element din *ListaValori* (în cazul primei variante) sau primul element din lista ţintă (în cazul celei de-a doua variante) trebuie să corespundă primului element ce apare în *ListaAtribute* (sau în definiţia tabelului dacă *ListaAtribute* este omisă) şi aşa mai departe pentru celelalte atribute.

5.3.2 Ştergerea înregistrărilor

Sintaxa instrucţiunii *delete* este:

delete from *NumeTabel* [where *Conditie*]

Dacă nu se specifică clauza *where* se vor şterge toate înregistrările din tabel.

În cazul în care instrucţiunea *delete* conţine clauza *where* se vor şterge doar acele înregistrări ce satisfac condiţia precizată.

Condiţia poate conţine şi interogări imbricate ce fac referire la conţinutul altor tabele.

În cazul în care există constrângeri de referinţă cu opţiunea *cascade* în care tabelul este referit ştergerea unor linii din tabel poate duce la ştergerea unor linii aparţinând altor tabele.

Exemplu

```
delete  from Departament where Dept not in (select Dept from Angajati)
```

Comanda de mai sus şterge departamentele fără angajaţi.

Este de notat diferenţa dintre comanda *delete* şi comanda *drop* definită în secţiunea 5.1.8.

O comandă de genul

```
delete from Departament
```

şterge toate liniile din tabelul DEPARTAMENT, şi posibil toate liniile tabelelor care sunt legate prin constrângeri de referinţă de acesta, dacă opţiunea *cascade* este precizată ca eveniment la ştergere; schema bazei de date rămâne neschimbată, comanda modificând doar instanţa bazei de date.

Comanda

```
drop table Departament cascade
```

are acelaşi efect ca şi comanda anterioară, dar în acest caz schema bazei de date se modifică, tabelul DEPARTAMENT fiind şters, la fel ca şi vederile sau tabelele care se referă la el în definiţiile lor.

Comanda

```
drop table Departament restrict
```

eşuează dacă există linii în tabelul DEPARTAMENT.

5.3.3 Actualizarea înregistrărilor

Sintaxa instrucţiunii *update* este

update *NumeTabel*
set *Atribut* = <*Expresie* | *InterogareSQL* | NULL | default>
{, *Atribut* = <*Expresie* | *InterogareSQL* | NULL | default>}
[where *Conditie*]

Instrucţiunea *update* face posibilă actualizarea unuia sau a mai multor atribute din liniile tabelului *NumeTabel* ce satisfac o posibilă *Conditie*.

Dacă nu apare clauza *where* se vor actualiza toate liniile din tabel.

Noua valoare ce va fi asignată unui atribut poate fi:

- rezultatul evaluării unei expresii, definită pe atributele din tabel;
- rezultatul unei interogări SQL;
- valoarea NULL;
- valoarea implicită a domeniului de definiţie.

Exemplu

Comanda

```
update  Angajati set Salariu = Salariu * 1.1
where Dept = 'Administratie'
```

produce o creştere cu 10% a salariilor angajaţilor din departamentul Administratie.

Natura SQL, care este orientat pe operaţiile cu mulţimi, trebuie luată în considerare când se scriu comenzi de actualizare.

Să presupunem că se doreşte modificarea salariilor angajaţilor astfel: creşterea salariilor sub 30 mii cu 10% şi a salariilor peste 30 mii cu 15%. O cale de a face actualizarea este execuţia următoarelor comenzi:

```
update  Angajati set Salariu = Salariu * 1.1
where Salariu <= 30

update  Angajati set Salariu = Salariu * 1.15
where Salariu > 30
```

Să presupunem că avem un angajat care câştigă 28 mii, deci satisface condiţia din prima comanda de actualizare şi atributul Salariu va fi setat la 30.8 mii. În acest moment, linia satisface de asemenea şi condiţia celei de-a doua actualizări, deci salariul va fi modificat din nou. Creşterea pentru salariatul respectiv va fi de 26.5%.

Această problemă particulară poate fi rezolvată prin inversarea celor două operaţii de actualizare.

În situaţiile mai complexe soluţia ar putea să necesite introducerea unor actualizări intermediare sau utilizarea unui limbaj de programare de înalt nivel care foloseşte *cursori*.

5.4 Alte definiţii de date în SQL

Având descrise în acest moment interogările în SQL se poate completa prezentarea componentelor unei scheme a unei baze de date cu clauza *check*, a *aserţiilor* şi a primitivelor pentru definirea *vederilor*.

5.4.1 Constrângerea de integritate de tip check

Pentru specificarea altor constrângeri decât cele discutate până acum, SQL-2 a introdus constrângerea *check*, care are următoarea sintaxă:

check (Condiţie)

Condiţiile ce pot fi utilizate sunt cele ce pot apărea în clauza *where* a unei interogări SQL.

Condiţia impusă trebuie verificată întotdeauna pentru a menţine corectitudinea bazei de date.

În acest fel pot fi specificate toate constrângerile pe tuplu discutate anterior, deoarece condiţia din constrângerea *check* poate face referire la alte atribute.

Exemplu. Vom redefini schema tabelului ANGAJATI din secţiunea 5.1.7:

```
create table Angajati
(NrInreg    character(6) check ( NrInreg is not NULL and
                                 1 = (select count(*)
                                      from Angajati a
                                      where NrInreg = a.NrInreg)),
Nume        character(20) not NULL  check ( Nume is not NULL),
Prenume     character(20) check ( Prenume is not NULL and
                                 1 = (select count(*)
                                      from Angajati a
                                      where Prenume = a.Prenume
                                      and Nume = a.Nume)),
Dept        character(15) check (Dept in (select NumeDept
                                          from Departament))
)
```

- constrângerile predefinite permit o reprezentare compactă şi mai uşor de citit;
- prin utilizarea clauzei *check* se pierde posibilitatea specificării unei reacţii în cazul încălcării constrângerii;
- când se utilizează constrângeri predefinite, sistemul le recunoaşte imediat şi le poate verifica mult mai eficient.

5.4.2 Aserţii

Aserţiile reprezintă constrângeri ce nu sunt asociate unei anumite linii sau unui anumit tabel în particular şi fac parte din schema bazei de date.

Aserţiile permit definirea tuturor constrângerilor prezentate şi, în plus, permit definirea unor constrângeri care nu pot fi exprimate altfel (de exemplu constrângeri între mai multe tabele, constrângeri ce impun ca un tabel să aibă o anumită cardinalitate).

Aserţiile au un nume şi pot fi şterse cu ajutorul comenzii *drop*.

Sintaxa ce permite definiţia aserţiilor este:

create assertion *NumeAsertie* check (*Conditie*)

Exemplu

```
create assertion CelPutinUnAngajat
      check ( 1<= (select count(*)
                   from Angajati))
```

Această constrângere impune ca tabelul ANGAJATI sa aibă cel puţin o înregistrare.

Constrângerile de integritate *check* sau *aserţie* pot fi imediate sau întârziate.

- Constrângerile imediate sunt verificate după fiecare modificare a bazei de date.
- Constrângerile întârziate sunt verificate la sfârşitul unei secvenţe de modificări a bazei de date, numită tranzacţie.

Încălcarea unei constrângeri imediate de o instrucţiune reface starea bazei de date din momentul anterior execuţiei instrucţiunii.

Dacă o constrângere întârziată este încălcată se va reface starea bazei de date din momentul anterior începerii tranzacţiei.

În interiorul unui program se poate seta tipul unei constrângeri la imediată sau întârziată cu ajutorul comenzilor

set constraints [*NumeConstrangere*] immediate

sau

set constraints [*NumeConstrangere*] deferred

5.4.3 Vederi

În capitolul 3 vederile au fost introduse ca fiind tabele „virtuale” al căror conţinut depinde de conţinutul altor tabele din baza de date.

În SQL vederile sunt definite prin asocierea unui nume şi a unei liste de atribute cu rezultatul execuţiei unei interogări.

O vedere se defineşte folosind comanda:

create view *NumeVedere* [(*ListaAtribute*)] as *InterogareSQL*
[with [<local | cascaded>] check option]

Interogarea SQL şi schema vederii trebuie să aibă acelaşi număr de atribute.

Exemplu

Să se definească vederea AngajatiAdmin care va conţine toţi angajaţii din departamentul Administraţie şi care au salariul mai mare ca 10.

```
create view AngajatiAdmin (NrInreg, Nume, Prenume, Salariu) as
    select NrInreg, Nume, Prenume, Salariu
    from Angajati
    where Dept = 'Administratie' and Salariu > 10
```

În cazul anumitor vederi se pot efectua operaţii de actualizare, dar aceste operaţii trebuie translate în instrucţiuni de modificare a tabelelor ce stau la baza vederii.

Nu întotdeauna se pot găsi soluţii de modificare a tabelelor de bază, mai ales în situaţiile în care vederea se defineşte pe baza unei joncţiuni între mai multe tabele.

În general sistemele comerciale permit modificarea unei vederi doar dacă este definită pe un singur tabel; alte sisteme cer ca atributele vederii să conţină măcar o cheie primară a tabelului de bază.

Clauza *check option* specifică faptul că operaţiile de actualizare se pot face numai asupra liniilor ce aparţin vederii şi după actualizare liniile continuă să aparţină vederii.

Când o vedere este definită pe baza altor vederi, opţiunile *local* sau *cascaded* specifică dacă ştergerea liniilor se face la nivel local sau trebuie propagată la toate vederile de care depinde vederea în cauză.

Opţiunea implicită este *cascaded*.

Exemplu

Să se definească vederea AngajatiAdmin1, bazată pe vederea AngajatiAdmin, care va conţine toţi angajaţii din departamentul Administraţie şi care au salariul între 10 şi 50.

```
create view AngajatiAdmin1 as
    select *
    from AngajatiAdmin
    where Salariu < 50
    with check option
```

Încercarea de a da valoarea 8 atributului Salariu nu va fi acceptată de definiţia curentă a vederii, dar ar fi fost validată dacă *check option* ar fi fost definită ca *local*. Încercarea de a modifica valoarea atributului Salariu pentru o linie din vedere la valoarea 60 nu ar fi validată nici cu opţiunea *local*.

Vederile pot fi utilizate în SQL pentru formularea unor interogări care altfel ar fi imposibil de exprimat.

În general, vederile pot fi considerate ca fiind unelte ce măresc posibilitatea creării interogărilor imbricate.

Exemplu

Interogarea 24: Să se găsească departamentul cu cel mai mare buget alocat salariilor.

```
create view BugetSalarii (Dept, SalariuTotal) as
    select Dept, sum(Salariu)
    from Angajati
    group by Dept

    select Dept
    from BugetSalarii
    where SalariuTotal = (select max(SalariuTotal) from BugetSalarii)
```

5.5 Controlul accesului la baza de date

Mecanismele de protecţie a datelor reprezintă un aspect important al aplicaţiilor moderne ce lucrează cu baze de date.

Administratorul bazei de date are posibilitatea de a alege şi de a implementa politici adecvate de control al accesului la baza de date.

SQL a fost proiectat astfel încât fiecare utilizator poate fi identificat în două moduri:

- utilizator al sistemului de operare;
- utilizator al bazei de date.

5.5.1 Resurse şi privilegii

Resursele protejate de sistem sunt în general tabelele, dar pot fi protejate şi alte componente, cum ar fi atributele unor tabele, vederi sau domenii.

Ca regulă generală, utilizatorul care creează o resursă este proprietarul ei şi este autorizat să efectueze orice operaţie asupra acelei resurse.

Din cauza acestei limitări, SQL pune la dispoziţie mecanisme de organizare ce permit administratorului să specifice acele resurse la care utilizatorii au acces şi cele la care nu au acces.

Prin intermediul acestor mecanisme utilizatorii dispun de *privilegii de acces* la resursele sistemului.

Fiecare privilegiu de acces este caracterizat de:

- resursa la care face referire;
- utilizatorul ce acordă privilegiul;
- utilizatorul ce primeşte privilegiul;
- operaţia permisă asupra resursei;
- posibilitatea acordării privilegiului altor utilizatori.

În momentul creării unei resurse, sistemul acordă, în mod automat, toate privilegiile asupra resursei creatorului său.

Există un utilizator predefinit, _system, asociat administratorului bazei de date, ce deţine toate privilegiile asupra tuturor resurselor.

Tipurile de privilegii disponibile sunt:

1) insert – permite inserarea unui obiect nou în resursă (aplicabil numai tabelelor şi vederilor);
2) update – permite modificarea valorii unui obiect (poate fi utilizat cu tabele, vederi şi atribute);
3) delete – permite eliminarea unui obiect din resursă (doar tabele sau vederi);

4) select – permite utilizatorului să citească resursa cu scopul de a o utiliza în interogări (tabele, vederi sau atribute);

5) references – permite crearea unei referinţe către o resursă în contextul definirii unui tabel. Poate fi asociat cu tabele sau atribute.

Acordarea acestui privilegiu asupra unei resurse poate conduce la limitarea posibilităţii de modificare a resursei.

Să considerăm că utilizatorul Paul este proprietarul tabelului DEPARTAMENT, iar utilizatorul Ştefan deţine privilegiul de referinţă. Ştefan are posibilitatea să definească o constrângere de tip *foreign key* pe tabelul său ANGAJATI, referind resursa indicată de privilegiu (de exemplu cheia tabelului DEPARTAMENT).

Dacă Ştefan adoptă o politică *no action* la definirea constrângerii, Paul va fi pus în situaţia de a nu putea şterge sau modifica linii din tabelul său dacă aceste operaţii au ca efect încălcarea constrângerii.

6) usage – se aplică domeniilor şi permite utilizarea lor, spre exemplu, pentru definirea schemei unui tabel.

Privilegiul de a efectua operaţiile de *drop* sau *alter* nu poate fi acordat. Acest tip de privilegiu este deţinut doar de proprietarul resursei.

Privilegiile se acordă sau se revocă cu ajutorul instrucţiunilor grant şi revoke.

5.5.2 Comenzi pentru acordarea şi revocarea privilegiilor

Sintaxa comenzii de acordare de privilegii este:

grant *Privilegii* on *Resursă* to *Utilizatori* [with grant option]

Această instrucţiune permite acordarea de *Privilegii* asupra *Resursei* către *Utilizatori*.

Exemplu

grant select on Departament to Stefan

Clauza *with grant option* indică posibilitatea propagării privilegiului către alţi utilizatori.

Se pot utiliza cuvintele cheie *all privileges* pentru acordarea tuturor privilegiilor.

Exemplu

grant all privileges on Departament to Stefan, Paul

Sintaxa comenzii de revocare de privilegii este:

revoke *Privilegii* on *Resursă* from *Utilizatori* [restrict | cascade]

Printre privilegiile ce pot fi revocate unui utilizator se găseşte şi privilegiul *grant option*, ce derivă din utilizarea opţiunii *with grant option*.

Revocarea privilegiilor poate fi făcută doar de utilizator care, într-o primă fază, a acordat aceste privilegii.

Opţiunea *restrict* împiedică execuţia instrucţiunii *revoke* dacă retragerea privilegiului are ca efect o retragere în lanţ de privilegii. O astfel de comportare poate apărea în situaţia în care utilizatorul a primit privilegiul cu opţiunea *with grant option* şi a propagat privilegiul către alţi utilizatori.

Opţiunea *cascade* în schimb va avea ca rezultat revocarea tuturor privilegiilor din lanţ şi în plus va elimina toate obiectele din baza de date ce au fost construite pe baza acestor privilegii.