8. Tabele virtuale. Tabele temporare. Indexare

8.1 Tabele virtuale (vederi, view-uri)

**Vederile** sunt, de fapt, nişte **tabele virtuale**, adică vederile conţin interogări care regăsesc în mod dinamic datele atunci când sunt utilizate. Ele nu conţin date, ci fac referire la date din tabelele bazei de date. **Vederile** (**tabelele virtuale**) sunt stocate alături de tabele în baza de date.

Vederile pot fi folosite pentru simplificarea operaţiunilor **SQL** complexe. După ce a fost scrisă o interogare aceasta poate fi refolosită cu uşurinţă dacă a fost creată o vedere pe baza acelei interogări. În plus, rezultatele din tabela virtuală se actualizează dinamic, în funcţie de ceea ce se întâmplă în tabelele din care au fost extrase datele în interogarea care a fost declarată la crearea tabelei virtuale.

**Vederile** sau **tabelele virtuale** sunt adesea denumite cu termenul în limba engleză, este vorba de termenul **view**. Mai trebuie precizat şi faptul că,**vederile** (**view**-urile) pot fi folosite pentru a folosi doar părţi din anumite tabele, iar nu tabele complete.

După ce au fost create **vederile**, se pot efectua operaţii de interogare (instrucţiuni **SELECT**) şi pe acestea. Ca şi tabelele, vederile trebuie să aibă un nume unic în cadrul unei baze de date. O **vedere** **nu** poate primi **numele unei tabele** din acea bază de date.

O **tabelă virtuală** în **SQL** este creată utilizând comanda **CREATE VIEW**. De asemenea, trebuie specificat un nume pentru tabela virtuală imediat după comanda **CREATE VIEW**.

Putem spune că o **vedere** reprezintă o instrucţiune **SELECT** stocată în baza de date. **Vederile** reprezintă un mod de accesare a datelor, ele nu stochează date efectiv. Sunt legate de tabela sau tabelele pe baza cărora au fost create. Ele pot fi folosite din motive de securitate. De exemplu, se pot crea părţi din tabelă vizibile anumitor utilizatori.

**Vederile** pot fi actualizabile sau neactualizabile. Cele actualizabile permit instrucţiunile **INSERT**, **UPDATE**, **DELETE**. Pe cele neactualizabile nu se pot efectua aceste operaţii. Pentru a fi actualizabile trebuie să fie o corespondenţă linie cu linie între o tabelă virtuală şi tabela pe baza căreia a fost creată.

Este evident acest lucru, dacă nu există această corespondenţă între view şi tabelă nu se pot face actualizări în  tabela virtuală.

De asemenea, este important de menţionat că nu trebuie să avem în instrucţiunea care creează tabela virtuală actualizabilă **funcţii de grup**, clauzele **DISTINCT**, **GROUP BY** şi **HAVING**. Nu putem avea nici**subinterogări**, **reuniuni** sau **OUTER JOIN**. În schimb, dacă avem **INNER JOIN** în instrucţiunea de creare, atunci este permisă actualizarea.

# 8.1.1 Crearea unei tabele virtuale

Sintaxa folosită pentru a crea o vedere (view) este următoarea:

**CREATE VIEW** *nume\_vedere* **AS**

**SELECT** ....;

# 8.1.2 Redefinirea unei tabele virtuale

O **vedere** (**view**) poate fi redefinită. În acest sens, avem o instrucţiune specială, similară cu cea de creare a tabelei virtuale.

Sintaxa folosită pentru **redefinirea** unui **view** este următoarea:

**ALTER VIEW** *nume\_vedere* **AS**

**SELECT** ....;

# 8.1.3 Ștergerea unei tabele virtuale

Evident, avem şi posibilitatea de a şterge o tabelă virtuală din baza de date. Instrucţiunea care permite ştergerea unei vederi din baza de date este următoarea:

**DROP VIEW** *nume\_vedere*;

Alte menţiuni pe care le vom face în legătură cu vederile se referă la faptul că o vedere poate fi creată şi pe baza unei alte vederi, deci, nu este obligatoriu să fie realizată pe baza unei tabele. Tabelele virtuale şi tabelele din baza de date sunt stocate în acelaşi loc pe disc, din acest motiv, este necesar ca ele să aibă nume diferite.

# 8.2 Tabele temporare

# 8.2.1 Noțiuni generale

Vom defini, în continuare, noțiunea de **tabelă temporară**. Vom prezenta sintaxa de creare a unei tabele temporare în bazele de date **MySQL** și vom explica ce înseamnă și la ce folosesc.

*O tabelă temporară stochează date doar pe durata unei tranzacții sau a unei sesiuni de lucru.* *Tabelelor temporare nu li se alocă spațiu la creare decât dacă în cadrul instrucțiunii****CREATE****de creare se specifică clauza****AS******SELECT***. Altfel, spațiul va fi alocat la prima instrucțiune **INSERT** în această tabelă temporară.

Subliniem faptul că nu trebuie făcută confuzia între tabele temproare și tabele virtuale sau vederi. Vederilor sau tabelelor virtuale au fost prezentate și exemplificate în modulul precedent. O tabelă virtuală este cunoscută și sub denumirea de view. *O tabelă virtuală nu poate face referire la tabele temporare, ea va face referire doar la tabele stocate permanent în baza de date.* De asemenea, nici triggere (declanșatoare) nu pot fi definite pentru tabele temporare.

La crearea tabelelor temporare, în sintaxa instrucțiunii **CREATE TABLE** apare cuvântul cheie **TEMPORARY** care specifică faptul că tabela creată este temporară. *Tabela temporară va fi ștearsă automat în momentul închiderii conexiunii sau sesiunii de lucru în care a fost creată*.

*Două conexiuni diferite la o bază de date pot crea tabele temporare cu același nume, ele nu vor interfera*. *Datele din tabelele temporare sunt vizibile doar sesiunii care le inserează (una singură la un moment dat)*.

O tabelă temporară este utilă atunci când este imposibil sau foarte dificil să fi obținute date printr-o singură instrucțiune **SELECT**. Se folosesc în special în cadrul rutinelor memorate în baza de date (proceduri stocate) pentru a stoca seturi de rezultate imediate în vederea utilizării ulterioare.

Tabelele temporare sunt create prin utilizarea instrucțiunii **CREATE TEMPORARY TABLE**, astfel, se observă că față de crearea unei tabele permanente, între cuvintele cheie **CREATE** și **TABLE** apare cuvântul cheie**TEMPORARY** care specifică faptul că tabele ce va fi creată este temporară.

Așa cum am precizat, *tabelele temporare sunt șterse în mod automat la închiderea sesiunii de lucru*, dar ele pot fi și *șterse explicit, înainte de închiderea sesiunii în care au fost create, prin utilizarea comenzii****DROP TABLE***. O tabelă temporară este disponibilă și accesibilă doar utilizatorului care a creat-o.

Utilizatori diferiți pot crea tabele temporare cu același nume deoarece doar clientul care a creat o tabelă temporară are acces la ea și poate să o utilizeze, deci, ele nu vor interfera chiar dacă au același nume. În aceeași sesiune, însă, două tabele temporare create de același utilizator nu pot avea același nume.

De asemenea, o tabelă temporară poate avea același nume cu al unei tabele permanent stocate în baza de date. În această situație, pe parcursul sesiunii curente, tabela permanentă devine inaccesibilă. Fiecare instrucțiune care face referire la tabela cu acel nume se va referi la tabela temporară.

Prin urmare, nu este recomandată crearea tabelelor temporare cu același nume ca al tabelelor permanente deoarece se poate crea confuzie la utilizare.

# 8.2.2 Modalități de creare a unei tabele temporare

Definirea unei tabele temporare se poate realiza similar cu definirea unei tabele permanente, cu apariția suplimentară a cuvântului **TEMPORARY**, deci, în instrucțiunea **CREATE** se vor specifica numele coloanelor din tabela temporară, tipul lor de date, dimensiunea, constrângerile, etc.

O altă modalitate de a defini o tabelă temporară este utilizarea unei instrucțiuni **SELECT** după cuvintele cheie **CREATE TEMPORARY TABLE** și numele dat tabelei temporare. Aceasta însemnă că tabela temporară va conține acele coloane returnate de instrucțiunea **SELECT** (cu tipurile de date aferente) și va fi și populată cu valorile corespunzătoare acelor coloanel ce au îndeplinit criteriile specificate în cadrul instrucțiunii**SELECT** din comanda **CREATE**.

Crearea unei tabele temporare poate fi realizată prin specificarea explicită a coloanelor componente ale tabelei, prin utilizarea instrucțiunii **SELECT** în comanda **CREATE**, astfel preluându-se date din tabelele de origine, sau, prin copierea structurii unei tabele cu ajutorul clauzei **LIKE**.

Avem, deci, cele trei forme posibile, a căror sintaxă este enunțată în cele ce urmează:

**CREATE TEMPORARY TABLE** *nume\_tabelă*(

                   nume\_coloană1 tip\_dată[(dimensiune)],

                   nume\_coloană2 tip\_dată[(dimensiune)],

                   ....

);

**CREATE TEMPORARY TABLE** *nume\_tabelă\_nouă* **LIKE** *nume\_tabelă\_originală*;

**CREATE TEMPORARY** **TABLE** *nume\_tabelă* [**AS**]

**SELECT** \* | column\_name(s)

**FROM** *tabelă\_origine*

                   [**WHERE** …];

# 8.2.3 Ștergerea unei tabele temporare

Am menționat faptul că o tabelă temporară poate fi ștearsă pe parcursul unei sesiuni prin utilizarea instrucțiunii **DROP TABLE** urmată de numele tabelei.

Recomandarea este să fie utilizata instrucțiunea **DROP TEMPORARY TABLE** urmată de numele tabelei temporare, tocmai pentru a nu se crea confuzie și pentru a nu exista riscul ca, în cazul în care există două tabele – una permanentă și una temporară cu același nume – să fie ștearsă tabela permanentă.

Comanda **DROP TEMPORARY TABLE** nu mai generează ambiguitate, este foarte limpede ce tabelă va fi eliminată din baza de date. Sintaxa comenzii de ștergere a unei tabele temporare este următoarea:

**DROP** [**TEMPORARY**] **TABLE** *nume\_tabelă*;

# 8.2.4 Deosebiri între tabele temporare și tabele virtuale

Așa cum am menționat mai sus, nu trebuie să confundăm tabelele temporare cu tabelele virtuale, sunt noțiuni diferite, astfel că, mai departe vom încerca să explicăm deosebirile dintre ele.

Vom prezenta, în tabelul următor, o comparație între aceste două obiecte ce pot să apară în bazele de date. Comparația are menirea de a ne lămuri pe deplin asupra a ceea ce reprezintă fiecare dintre aceste obiecte, precum și pentru a vedea principalele elemente care le caracterizează pe fiecare dintre ele.

|  |  |
| --- | --- |
| **Tabele temporare** | **Tabele virtuale (vederi, view-uri)** |
| Sunt stocate temporar în baza de date, până la încheierea sesiunii curente de lucru | Sunt stocate permanent în baza de date |
| Sunt create prin instrucțiunea**CREATE** **TEMPORARY TABLE** | Sunt create prin instrucțiunea**CREATE VIEW** |
| Conțin efectiv date atât timp cât sunt create și sesiunea curentă este activă | Nu conțin date efectiv ci fa referire la datele existente în tabela sau tabelele pe baza cărora au fost create. Tabelele virtuale sunt de două feluri: actualizabile și neactualizabile. Cele actualizabile prezintă avantajul actualizării automate și a vederii atunci când s-au făcut actualizări în tabela la care face referire vederea. |
| O tabelă temporară poate avea același nume cu al unei tabele permanente din baza de date | O tabelă virtuală nu poate să aibă același nume cu al unei tabele permanente stocate în baza de date |
| Pot exista două tabele temporare cu același nume în aceeași bază de date dacă sunt create de utilizatori diferiți | Nu pot exista două tabele virtuale cu același nume în aceeați bază de date, chiar dacă se încearcă crearea de utilizatori diferiți |

# 8.3 Index

# 8.3.1 Conceptul de index

La bazele de date de dimensiuni mari este foarte utilă metoda indexării pe anumite coloane sau ansambluri de coloane, astfel ca, la executarea interogărilor timpul de execuție să scadă.

Conceptul de index definește o structură de date adițională, redundantă, care dacă este atașată unei tabele dintr-o bază de date poate crește viteza de căutare și ordonare a datelor din acea tabelă. Un index poate fi definit pe una sau mai multe coloane ale unei tabele.

În momentul în care este definit un index, informația din coloana sau ansamblul de coloane pe care a fost definit indexul este copiată într-o structură separată, fiecare element al acestei noi structuri având o referință către înregistrarea de la care provine. Într-un index elementele sunt ordonate, iar aceasta face ca găsirea unei informații să fie rapidă.

Un index este util atunci când se realizează o căutare după o coloană ce a fost indexată (căutarea realizându-se în structura unde se află indexul – fișierul de index și va găsi elementele care îndeplinesc condiția enunțată, iar apoi, se va folosi de referințele pe care indexul le are către înregistrările corespunzătoare din tabelă), dar și atunci când se realizează o ordonare după o coloană ce este indexată (în acest caz, informațiile din index fiind deja ordonate, serverul va obține imediat înregistrările care corespund referințelor elementelor din index și le va afișa în ordinea din index). O ordonare descrescătoare va reprezenta o parcurgere inversă a intrărilor din index.

Există și anumite constrângeri care atunci când sunt definite pe coloanele unei tabele, creează automat un index pe coloana sau ansamblul de coloane pe care a fost impusă restricția respectivă.

Definirea unui **PRIMARY KEY** pe o coloană sau pe un ansamblu de coloane, *creează automat un index pe acea coloană sau pe ansamblul de coloane ce formează cheia primară*.

Astfel că, o interogare în care condiția de căutare din clauza **WHERE** este pusă pe o coloană **PRIMARY KEY** va genera rezultatul într-un timp rapid.

Același lucru se întâmplă și la definirea unei constrângeri **UNIQUE** pe o coloană, adică se va *genera automat un index pe coloana respectivă*.*Constrângerile de unicitate garantează faptul că toate datele dintr-o coloană sau dintr-un set de coloane sunt unice*.

# 8.3.2 Deosebiri între PRIMARY KEY și constrângerea UNIQUE

Constrângerile de unicitate sunt asemănătoare cu cheile primare, dar între ele există și deosebiri. Astfel, o tabelă poate conține mai multe constrângeri de unicitate, deci constrângerea **UNIQUE** poate fi *definită pe mai multe coloane*, pe când **cheia primară** este *unică într-o tabelă*, chiar dacă este compusă din mai multe coloane (ansamblu de coloane) și nu este simplă (o singură coloană).

Deci, atunci când *cheia primară este definită pe un ansamblu de coloane trebuie ca acele valori din coloanele respective luate împreună (combinația de valori) să fie unice*.

O altă **deosebire** între coloanele care au definită constrângerea de **cheie primară** și cele care au definită **constrângerea de unicitate** este aceea că acele *coloanele ce prezintă constrângeri de unicitate pot conține valori****NULL***. Cunoaștem faptul că în *coloanele ce prezintă constrângerea cheie primară nu pot conține valori nule*. *Coloanele cheie primară conțin valori unice și nenule.*

# 8.3.3 Considerații generale despre indexare

Există, însă, posibilitatea de a defini indecși și pe alte coloane ce nu au definite constrângeri de alt tip asupra lor. În momentul definirii unui index, sau al generaării automate a unui index, pe o coloană sau pe un grup de coloane *se crează automat un fișier, numit fișier de index care conține datele din această coloană sau din grupul de coloane*.

Astfel, atunci când o interogare conține *condiția pe o coloană indexată căutarea se va face în fișierul de index, fapt ce va duce la returnarea rezultatului interogăriii într-un timp mult mai rapid decât dacă nu ar fi indexate coloanele*, tabele ar fi foarte mari iar căutarea s-ar face prin parcurgerea întregii tabele.

*Indexarea* trebuie înțeleasă și, reprezintă, de fapt, o *ordonare la nivel logic a datelor*, deci, *nu este o ordonare vizibilă* așa cum se întâmplă cu rezultatul utilizării clauzei **ORDER BY** într-o interogare, ci este *o ordonare sau sortare la nivel logic a datelor*.

*Principiul unui index este acela că este sortat în mod corect*. După definirea unui index, programul **SGBD** păstrează o listă sortată a conținutului. În momentul adresării unei cereri de regăsire către **SGBD** acesta caută în indexul sortat pentru a găsi locația eventualelor valori care corespund celor ce îndeplinesc criteriile de căutare și apoi regăsește înregistrările respective.

Nu există nici o regulă universală cu privire la ce coloane trebuie să fie indexate și când trebuie indexate. Menționăm, însă, și anumite aspecte cu privire la posibilitatea ca, în anumite cazuri, utilizarea de indecși să îngreuneze anumite instrucțiuni pe baza de date.

Trebuie să ținem cont și de faptul că, doar adăugarea de indecși, operație care se realizează foarte ușor, nu reprezintă mereu o soluție completă de optimizare. Într-adevăr, prin generarea acelor fișiere de indecși care se salvează pe server și care permit realizarea mult mai **rapidă** a instrucțiunilor**SELECT**, aceste fișiere duc la **îngreunarea** execuției instrucțiunilor**INSERT**, **UPDATE** și **DELETE**.

Duc la îngreunarea execuției acestor instrucțiuni deoarece la fiecare operație de acest tip (**INSERT**, **UPDATE** sau **DELETE**) se va produce o regenerare autmată de către sistem a acestor fișiere de indecși.

Deși, așa cum am menționat mai sus, nu există nici o regulă generală cu privire la ceea ce trebuie indexat, totuși sunt coloane care nu se pretează la indexare. Practic, datele care nu prezintă o frecvență suficient de redusă nu vor beneficia la fel de mult de pe urma indexării.

Astfel, declararea unui index pe coloane cum ar fi un nume, un prenume, o denumire, un pret, o cantitate, o localitate nu sunt eficiente pentru că ar putea exista foarte multe valori identice astfel că nu ar fi relevante. Se pot indexa coloane ce conțin date calendaristice, de exemplu data nașterii stocată pentru anumite persoane ceea ce ar îmbunătăți căutarea după data nașterii pentru a identifica o persoană.

De asemenea, este recomandată reexaminarea periodică a indexărilor realizate într-o bază de date, deoarece anumite indexuri eficiente la un anumit moment, pot deveni ineficiente după o perioadă când baza de date a suferit diverse modificări (manipulări de date).

# 8.3.4 Avantaje și dezavantaje întâlnite la utilizarea indecșilor

Principalul **avantaj** al definirii de indecși este creșterea vitezei operațiilor de căutare și ordonare. Acest lucru se întâmplă, în cazul în care căutarea se realizează după o coloană indexată. Dacă se realizează o filtrare după alte coloane din tabelă care nu sunt indexate, nu are importanță faptul că în tabela respectivă există și coloane indexate. Deci, recomandarea este să fie indexate acele coloane ce sunt folosite frecvent pentru filtrarea datelor dintr-o tabelă (coloanele utilizate în clauza **WHERE** a instrucțiunii **SELECT**).

Un alt **avantaj** al indexării este oferit de faptul că indecșii pot introduce restricții asupra coloanei sau grupului de coloane pe care sunt definiți. Acest lucru se întâmplă, de exemplu, atunci când este definită o restricție**UNIQUE**, care va crea și un index pe coloana sau ansamblul de coloane respectiv..

Un **dezavantaj** constă în scăderea vitezei la celelate operații de manipulare a datelor (inserare, modificare și ștergere) deoarece, pe lângă modificările produse în tabelă, este necesar să se actualizeze de fiecare dată și indexul (actualizarea aceasta se face automat dar îngreunează aceste operații).

Un alt **dezavantaj** este acela că la crearea de indecși se alocă spațiu suplimentar care va fi ocupat. Se întâmplă acest lucru deoarece indexul reproduce părți ale înregistrărilor din tabele. Un număr mare de indecși va crește dimensiunea tabelei, fapt din care decurge că va fi ocupat mai mult spațiu pe disc.

Deci, faptul că pot ocupa o mare cantitate de spațiu reprezintă un alt aspect care poate face ineficientă utilizarea indexării.

Astfel, indexarea degradează operațiile de inserare, modificare și ștergere pentru că la execuția acestor instrucțiuni, programul **SGBD** trebuie să actualizeze fișierul de index (lista de index) în mod dinamic.

# 8.3.5 Utilitatea creării indecșilor

Fiind vorba despre optimizări aduse bazei de date care pot duce la creșterea vitezei de căutare, dar care pot produce și anumite încetiniri pe operații de actualizare a datelor, trebuie analizată cu atenție oportunitatea folosirii indexării. Nu întotdeauna avem nevoie de indexări de coloane pe tabelele unei baze de date.

Prezentăm mai jos o parte din situațiile în care este recomandat să fie folosiți indecși pentru optimizarea procesului de căutare a informațiilor:

·         Coloana respectivă este utilizată frecvent în clauza **WHERE** a instrucțiunilor**SELECT** sau în condițiile de join;

·         Coloana conține o gamă largă de valori;

·         Coloana conține un număr mare de valori nule;

·         Tabelele au dimensiuni foarte mari, iar interogările vor returna un număr mic de rezultate;

·         Două sau mai multe coloane sunt utilizate frecvent împreună în clauza**WHERE** a instrucțiunilor **SELECT** sau în condițiile de join din interogări.

Expunem, în continuare, și o serie de situații în care nu este recomandată indexarea coloanelor, deoarece indexarea ar cauza o încetinire a vitezei de lucru:

·         Tabelele au dimensiuni mic, deci, conțin un număr mic de înregitrări;

·         Coloanele sunt folosite rar în condițiile ce se pun la realizarea interogărilor;

·         Tabelele sunt modificate frecvent, deci, se execută foarte des instrucțiuni de manipulare ce afectează înregistrările din tabele;

·         Cele mai multe interogări returnează un număr mare de înregistrări.

# 8.3.6 Metode de regăsire a datelor

Există două metode ce pot fi utilizate în SQL pentru regăsirea datelor. Aceste metode sunt următoarele:

·         **Metoda secvențială;**

·         **Metoda cu acces direct;**

În cazul utilizării metodei secvențiale se realizează o parcurgere a fiecărui element din tabelă pentru a găsi înregistrările ce îndeplinesc condițiile specificate în interogare. Aceasta este o metodă ineficientă de căutare.

Pentru a înțelege mai bine ce înseamnă indexarea se poate face asocierea cu indexul situat la sfârșitul unei cărți. Astfel, dacă se dorește găsirea tuturor aparițiilor unei anumite expresii sau al unui anumit concept, metoda cea mai simplă este să se verifice indexul de la sfârșitul cărții și se va găsi acolo la ce pagini este menționată expresia respectivă. Indexul este o listă alfabetică, deci, găsirea unui anumit concept în index, oricât de stufos ar fi, este rapidă. O altă metodă ar fi parcurgerea cărții pagină cu pagină și linie cu linie pentru a găsi locurile în care apare expresia căutată. În mod evident, această metodă este total ineficientă și consumă foarte mult timp.

În felul acesta funcționează și indexarea coloanelor în tabelele bazelor de date. Atunci când se realizează o căutare (**SELECT**) după o anumită coloană care este indexată se va parcurge fișierul de index (deci, facem analogia cu indexul de la finalul cărții). Avem de-a face, în situația aceasta, cu o metodă de acces direct la date.

Dacă nu există indecși definiți, atunci se va face o parcurgere linie cu linie a întregii tabele pentru a găsi înregistrările ce corespund criteriilor de căutare (deci, analogia cu parcurgerea pagină cu pagină a unei cărți). Este vorba, deci, de o metodă secvențială de acces la date,

Devine limpede faptul că este mult mai rapidă căutarea atunci când există indecși definiți iar căutarea se face în fișierul de index. Prin adăugarea de indecși se obțin performanțe superioare privind accesul la datele din baza de date.

# 8.3.7 Clasificare

Din punct de vedere al numărului de coloane pe care le cuprinde, un index poate fi clasificat astfel:

·         **index simplu** sau **index uni-coloană** (creat pe o singură coloană);

·         **index multiplu** sau **multi-coloană** (creat pe două sau mai multe coloane);

·         **index parțial** (creat pe oa numită parte dintr-un șir de caractere în cazul**MySQL**);

În situația indexării multiple, deci, crearea unui index pe un ansamblu (grup) de coloane, **MySQL** realizează indexarea prin *concatenarea (alipirea) coloanelor în ordinea specificării acestora* în instrucțiunea de crearea indexului multiplu.

De asemenea, la acest tip de index, multi-coloană, ordinea în care sunt introduse coloanele în index contează. **MySQL** se folosește de un *index multi-coloană* doar în situația în care clauza **WHERE** a instrucțiunii**SELECT** conține valori din prima sau primele coloane din index, dacă, însă, lipsește prima coloană și sunt utilizate doar valori din celelalte, atunci nu va fi utilizat indexul.

O altă clasificare a indecșilor care se poate face este din punct de vedere al constrângerilor ce sunt impuse asupra datelor din coloana sau din coloanele pe care este definit indexul respectiv.

Astfel, distingem următoarele tipuri de indecși:

·         **index simplu** – coloana respectivă este indexată fără a se impune restricții asupra ei;

·         **index unic** – valorile acestei coloane trebuie să fie unice (constrângerea**UNIQUE**). În cazul introducerii de valori duplicate în coloane ce au definit acest tip de index, sistemul va genera o eroare. Însă, într-o coloană ce are definit acest index pot fi introduse valori nule (**NULL**). Pot exista mai multe valori nule pe coloana respectivă.

·         **cheie primară** (**PRIMARY KEY**) – index care constrânge la introducerea de valori unice și nenule. Pe o tabelă poate fi definit un singur index de tip**PRIMARY KEY**, dar mai multe de tip **UNIQUE**.

Un **index unic** crește și mai mult viteza de execuție a interogărilor filtrate după o astfel de coloană, față de un index simplu. În cazul unui index simplu este parcurs întreg fișierul de index, deoarece poate exista un set de valori corespunzător căutării. În situația în care pe coloana respectivă este definit un index unic, în momentul găsirii în index a elementului respectiv, sistemul nu va căuta mai departe în index alte valori.

# 8.4 Operații

Operațilie ce se pot face cu indecși sunt: crearea, ștergerea și actualizarea lor (aceasta din urmă se realizează automat în cazul efectuării operațiilor de manipulare **INSERT**, **UPDATE**, **DELETE**); de altfel, așa cum am prezentat mai sus, și crearea se poate face automat în anumite cazuri, adică atunci când sunt definite constrângeri de tip **PRIMARY KEY**, **UNIQUE** se crează automat și indecși pe acele coloane.

Fiecare index definit trebuie să aibă un nume unic.

# 8.4.1 Crearea unui index

Indecșii pot fi creați în momentul creării tabelei, deci, în cadrul instrucțiunii**CREATE TABLE** sau, ulterior, prin executarea unei instrucțiuni **ALTER TABLE** sau **CREATE INDEX**.

La crearea tabelei, în cadrul sintaxei **CREATE TABLE** se adaugă clauze suplimentare: **PRIMARY KEY**, **UNIQUE** sau **INDEX**. Clauza se adauagă în dreptul coloanei respective, sau la final, după enumerarea coloanelor, în zona de restricții (în această zonă coloana sau coloanele vor fi trecute după numele constrângerii între paranteze).

Sintaxa instrucțiunii **ALTER TABLE** de adăugare a unui index după ce tabela a fost creată este următoarea:

**ALTER TABLE** *nume\_tabelă* **ADD** tip\_index(nume\_coloană1[, nume\_coloană2, ...]);

Dacă tabela este deja populată cu înregistrări pot să apară unele erori în cazul definirii unui index de tip **PRIMARY KEY** sau **UNIQUE**. Sistemul va genera eroare la definirea unui astfel de index, dacă în coloana respectivă există valori duplicate (**UNIQUE**) sau valori duplicate sau nule (**PRIMARY KEY**).

Comanda **CREATE INDEX** se folosește pentru crearea unui index la o tabelă, index care va conține câte o intrare pentru fiecare valoare ce apare specificată în tabelă în coloana respectivă. Această coloană poartă numele de coloană de index.

Sintaxa comenzii de creare a unui index este următoarea:

**CREATE** [**UNIQUE**] **INDEX** *nume\_index*

**ON** *nume\_tabelă* (nume\_coloană1 [**ASC**|**DESC**][, nume\_coloană2 [**ASC**|**DESC**], …]);

Cu ajutorul comenzii **CREATE INDEX** se pot crea indecși simpli sau unici, dar nu se pot crea indecși de tip **PRIMARY KEY**.

# 8.4.2 Vizualizarea indecșilor existenți

Vizualizarea indecșilor definiți asupra coloanelor unei tabele se poate face prin executarea comenzilor:

·         **SHOW CREATE TABLE** – ce va afișa indecșii ca parte a definiției tabelei;

·         **DESCRIBE** – care afișează structura unei tabele, inclusiv indecșii;

·         **SHOW INDEX** – care va afișa doar indecșii definiți.

Primele două comenzi au fost deja prezentate, așa că aici vom reda sintaxa instrucțiunii **SHOW INDEX** care este următoarea:

**SHOW INDEX FROM** *nume\_tabelă*;

Această comandă este importantă mai ales pentru afișarea numelor pe care le are fiecare index. Numele unui index este important dacă se încearcă ștergerea lui, deoarece va fi utilizat în instrucțiunea de ștergere.

8.4.3 Ștergerea unui index se poate realiza fie printr-o instrucțiune **ALTER TABLE** care va îndepărta o constrângere care este și index, fie printr-o instrucțiune **DROP INDEX**.

Eliminarea unei constrângeri care este și index, prin executarea unei instrucțiuni **ALTER TABLE,** se face prin comanda următoarea:

**ALTER TABLE** *nume\_tabelă* **DROP** tip\_index [(nume\_index)];

Instrucțiunea de ștergere a unui index simplu este următoarea:

**DROP INDEX** *nume\_index* **ON** *nume\_tabelă*;

# 8.5 Concluzii

În concluzie*, indexarea este o metodă deosebit de eficientă pentru creșterea eficienței execuției instrucțiunilor de regăsire a datelor (****SELECT****)*, dar, în același timp, poate duce la *îngreunarea execuției instrucțiunilor de manipulare a datelor ce afectează datele din tabelele unei baze de date* (este vorba de instrucțiunile **INSERT**, **UPDATE** și **DELETE**).

*Indexarea nu schimbă ordinea fizică a înregistrărilor din tabele*. *Prin definirea indecșilor se crează automat o corespondență între ordinea fizică a înregitrărilor și ordinea logică*. Indecșii ordonează la nivel logic după anumite criterii stabilite la definirea (crearea) lor.