Lecţia 3 – Limbajul de descriere a datelor (LDD)

3.1 Introducere

Recapitulând noţiunile prezentate în capitolele anterioare şi raportându-ne concret la o bază de date MySQL, avem următoarele corespondenţe:

- o *entitate* reprezintă o *tabelă* din baza de date;
- un *atribut* reprezintă un *câmp* sau o *coloană* din baza de date;
- un *tuplu* reprezintă o înregistrare din baza de date;
- **UID** reprezintă *cheia primară* a unei tabele.

Limbajul de Desciere a Datelor (LDD, în limba engleză **Data Description Language**, prescurtat **DDL)** conţine comenzi pentru:

- crearea unei baze de date;
- ştergerea unei baze de date;
- crearea unei tabele;
- ştergerea unei tabele;
- modificarea structurii unei tabele.

În continuare vom prezenta elemente ce ţin de sintaxa limbajului **MySQL**. *Sintaxa limbajului* se referă la un *set de reguli* ce trebuie respectate atunci când scriem o *instrucţiune*. Pentru *instrucţiune* vom mai întâlni denumirile de *comandă*, respectiv, de *frază SQL*.

Aşadar, pe parcursul acestui curs vom întâlni în foarte multe rânduri termenii de ***instrucţiune SQL***, ***comandă SQL*** sau ***frază SQL***. De asemenea, trebuie precizat că, spunem la modul generic ***comandă SQL*** sau ***frază SQL*** sau ***instrucţiune SQL***, deşi, în cazul nostru, este limpede că ne vom referi la ***instrucţiuni/comenzi/fraze MySQL***. După cum am precizat şi în lecţia precedentă, **MySQL** este un **S**istem de **G**estiune al **B**azelor de **D**ate (**SGBD**) bazat pe standardul **SQL (S**tructured **Q**uery **L**anguage**).**

Deşi prezintă particularităţi, ca de altfel orice **SGBD**, totuşi instrucţiunile **MySQL** sunt foarte asemănătoare cu cele ale standardului **SQL**. Din acest motiv, în mod uzual, vorbim de instrucţiuni **SQL**. Cu alte cuvinte, putem spune că **SQL** este un nucleu de la care s-au dezvoltat aceste aplicaţii complexe utilizate pentru interacţiunea cu bazele de date, denumite **SGBD**-uri.

Aşa cum spuneam, în **MySQL**, ca de altfel în orice alt **SGBD**, pentru fiecare instrucţiune/comandă avem o sintaxă, adică un set de reguli de scriere care trebuie respectat pentru a nu fi generate erori la rularea instrucţiunilor.

Astfel, există un set de **cuvinte rezervate** ale limbajului care definesc anumite acţiuni şi care trebuie specificate în cadrul unei instrucţiuni **MySQL**. Aceste cuvinte speciale (rezervate) se numesc **cuvinte cheie**. Atunci când scriem o instrucţiune **MySQL** trebuie, aşadar, să folosim aceste cuvinte cheie, plus alte cuvinte (nume de tabele, nume de câmpuri, valori, operatori, etc.) scrise într-o anumită ordine.

Important de reţinut este şi faptul că orice instrucţiune **MySQL** se încheie cu **caracterul „;”**.

În general, clienţii **MySQL**, adică programele cu ajutorul cărora se realizează conectarea la o bază de date **MySQL** şi se execută instrucţiuni **MySQL,** recunosc aceste *cuvinte cheie* şi le *colorează* diferit de celelalte cuvinte comune (nerezervate) utilizate în cadrul instrucţiunilor.

Acest fapt uşurează în mare parte scrierea unei instrucţiuni. O altă menţiune importantă este aceea că **nu este indicat** să denumim baze de date, tabele sau coloane (câmpuri) ale tabelelor cu nume care să fie cuvinte din limbajul rezervat al **MySQL**. Dacă se întâmplă acest lucru există riscul să avem erori la execuţia unor comenzi ce conţin aceste denumiri. Un câmp denumit astfel ar trebui prefixat de numele tabelei în fiecare instrucţiune pentru a nu fi eroare, deci s-ar complica mult codul instrucţiunii scrise.

3.2 Crearea unei baze de date

Comanda care se foloseşte pentru crearea unei baze de date în MySQL este:

CREATE DATABASE *nume_bd*;

unde *nume_bd* reprzintă numele pe care vrem să îl aibă baza de date.

Denumirea bazei de date poate să conţină doar caractere alfanumerice şi semnul „_”.

3.3 Ştergerea unei baze de date

Comanda care se foloseşte pentru crearea unei baze de date în MySQL este:

DROP DATABASE *nume_bd*;

3.4 Utilizarea unei baze de date

Comanda:

USE *nume_bd*;

- specifică faptul că din momentul executării acestei instrucţiuni se foloseşte baza de date specificată (se pot realiza operaţii pe această bază de date). Comanda este utilă atunci când avem mai multe baze de date. Fiecare instrucţiune MySQL se încheie cu caracterul „;”.

3.5 Crearea unei tabele

Comanda care se foloseşte pentru crearea unei tabele într-o bază de date este următoarea:

CREATE TABLE *nume_tabelă*(

$nume\_atribut_1$ tip_dată(dimensiune) [modificatori],

$nume\_atribut_2$ tip_dată(dimensiune) [modificatori],

...

$nume\_atribut_n$ tip_dată(dimensiune) [modificatori],

[restricţii]

);

- *nume_atribut* reprezintă numele coloanelor (câmpurilor) tabelei, *tip_dată* reprezintă tipul de dată al câmpului respectiv, *modificatori* este opţional la fel ca şi *restricţiile* ce se pot aplica pe anumite câmpuri.

3.6 Ştergerea unei tabele

Comanda care se foloseşte pentru a şterge o tabelă dintr-o bază de date este următoarea:

DROP TABLE *nume_tabelă*;

3.7 Modificarea unei tabele

Comanda care se foloseşte pentru modificarea numelui unei tabele dintr-o bază de date este următoarea:

ALTER TABLE *nume_tabelă*

RENAME TO nume_*nou_tabelă*;

sau

RENAME TABLE *nume_tabelă TO nume_nou_tabelă*;

3.8 Modificarea structurii unei tabele

Comenzile folosite pentru modificarea structurii unei tabele sunt următoarele:

- pentru modificarea definiţiei unui câmp:

ALTER TABLE *nume_tabelă* CHANGE *nume_câmp nume_câmp* definiţie_câmp;

- pentru adăugarea unui câmp într-o tabelă

ALTER TABLE *nume_tabelă* ADD *nume_câmp* definiţie_câmp;

- pentru ştergerea unui câmp dintr-o tabelă

ALTER TABLE *nume_tabelă* DROP *nume_câmp*;

3.9 Tipuri de date

În MySQL întâlnim următoarele tipuri de date:

- numerice
- şiruri de caractere
- binare
- date calendaristice
- text

Tipurile de date numerice sunt următoarele:

- pentru numere întregi:
  - **TINYINT** – poate lua valori de la -128 până la 127 sau de la 0 la 255 unsigned
  - **SMALLINT** – valori în intervalul -32 768 până la 32 767 sau de la 0 la 65 535 unsigned
  - **MEDIUMINT** – de la -8 388 608 până la 8 388 607 sau de la 0 la 16 777 215 unsigned
  - **INT** – de la -2 147 483 648 până la 2 147 483 647 sau de la 0 la 4 294 967 295 unsigned
  - **BIGINT** – de la -9 223 372 036 854 775 808 până la 9 223 372 036 854 775 807 sau de la 0 la 18 446 744 073 709 551 615 unsigned

Cel mai folosit tip de date numeric întreg este **INT**, sau dacă avem valori numerice mici într-un câmp putem folosi **SMALLINT** sau **TINYINT** care ocupă spaţiu mai puţin pe disc.

Fiecărui tip de dată i se specifică şi lungimea, de exemplu dacă avem valori de la 1 la 100 într-un câmp putem aloca ca tip de dată **INT**(3), adică numere întregi cu lungimea maximă 3.

- pentru numere cu zecimală:
    - **FLOAT** – folosit pentru numere mici cu virgulă;
    - **DOUBLE** – folosit pentru numere mari cu virgulă;
    - **DECIMAL** – permite alocarea unui număr fix de zecimale.

De exemplu, un câmp ce conţine preţul unui produs poate fi definit de tip **DOUBLE**(5,2).

În paranteză este trecut numărul total de cifre al preţului (5) respectiv numărul obligatoriu de zecimale care va fi afişat (2). Prin urmare, datele introduse în câmpul preş definit de tipul **DOUBLE**(5,2) poate lua valori cuprinse în intervalul închis [-999.99,999.99]. Delimitatorul pentru un număr cu zecimale recunoscut de MySQL este „.”.

Tipurile de date folosite pentru şiruri de caractere sunt următoarele:

- **CHAR** – lungime fixă de la 0 la 255 de caractere
- **VARCHAR** – lungime variabilă de la 0 până la 65 535 de caractere. La versiunile mai vechi de 5.0.3 ale **MySQL** lungimea era variabilă de la 0 la 255 de caractere.

În practică, tipul **VARCHAR** este cel mai folosit pentru definirea câmpurilor de tip şir de caractere sau string. Într-o tabelă cu informaţii despre angajaţii unei companii, de exemplu, numele angajaţilor poate fi ţinut într-un câmp nume de tip **VARCHAR**(70), între paranteze fiind trecută dimensiunea maximă pe care o poate avea valoarea introdusă în acest câmp.

Aşadar, câmpul nume poate avea maxim 70 de caractere ceea ce considerăm a fi suficient pentru a nu avea probleme de trunchiere a vreunui nume.

Putem defini, de asemenea, şi un câmp prenume de tip **VARCHAR**, de dimensiune 100 de caractere, adică **VARCHAR**(100).

Tipurile de date **CHAR** şi **VARCHAR** acceptă şi definirea unei valori implicite (default) pe care o va avea acel câmp în cazul în care nu se introduce nici o valoare în el.

Principala diferenţă între aceste două tipuri este că şirul dintr-un tip **CHAR** va fi stocat întotdeauna ca un şir cu lungimea maximă a coloanei, folosind spaţii pentru completare, dacă şirul introdus este mai mic decât lungimea coloanei.

Tipurile de date folosite pentru stocare text sunt următoarele:

- **TINYTEXT** – un şir cu lungime maximă de 255 de caractere;
- **TEXT** – un şir cu o lungime amximă de 65 535 de caractere;
- **MEDIUMTEXT** – un şir cu o lungime maximă de 16 777 215 de caractere;
- **LONGTEXT** – un şir cu o lungime maximă de 4 294 967 295 de caractere.

Pentru câmpuri în care este necesară stocarea unui text de mari dimensiuni, în general, se foloseşte tipul **TEXT**. Unui câmp de tip text nu i se poate specifica lungimea.

De exemplu, într-o tabelă a unei baze de date în care sunt stocate articole, câmpul ce conţine conţinutul (corpul) articolului poate fi de tip **TEXT**.

Spre deosebire de **VARCHAR**, tipul de date **TEXT** nu permite definirea unei valori implicite (default) pentru acel câmp.

La ultimele versiuni de **MySQL**, a fost mărită dimensiunea maximă a tipului de date **VARCHAR**, astfel încât poate fi folosit acesta şi pentru câmpurile cu texte lungi.

Tipurile de date binare întâlnite în cadrul MySQL sunt următoarele:

- **TINYBLOB** – stochează până la 255 bytes;
- **BLOB** (**B**inary **L**arge **Ob**ject) **–** stochează până la 64 KB;
- **MEDIUMBLOB** – stochează până la 16 MB;
- **LONGBLOB** – stochează până la 4 GB.

Aceste tipuri de date sunt folosite pentru stocarea obiectelor binare de mari dimensiuni cum ar fi imaginile. Valorile din câmpurile de tip **BLOB** sunt tratate ca şiruri binare.

Ele nu au un set de caractere iar sortarea şi compararea lor se bazează pe valorile numerice ale octeţilor din valoarea câmpului respectiv definit cu acest tip.

Tipurile de date folosite pentru stocarea datelor calendaristice sunt următoarele:

- **DATE** – stochează o dată calendaristică în formatul *an-lună-zi*;
- **TIME** – stochează ora în formatul *oră-minut-secunda*;
- **DATETIME** – stochează data şi ora în formatul *an-lună-zi ora-minut-secundă*;
- **TIMESTAMP** – este util la înregistrarea unor operaţii precum inserare sau actualizare pentru că reţine implicit data efectuării ultimei operaţii.

Singurul format în care **MySQL** păstrează şi afişează datele calendaristice este formatul *an-lună-zi* (*AAAA-LL-ZZ*), sau, mai cunoscut acest format după denumirea în limba engleză *year-month-day*, sau prescurtarea *YYYY-MM-DD*.

Intervalul în care poate lua valori o dată calendaristică este foarte mare, de la '1000-01-01' pînă la '9999-12-31'. Dacă avem într-o tabelă a unei baze de date stocată data naşterii unei persoane care presupunem că este 20 martie 1981. Informația cu privire la data naşterii va fi stocată în baza de date în următorul format '1981-03-20'.

Formatul în care se salvează un câmp de tip **TIME**, câmp care păstrează ora în baza de date, este *oră-minut-secundă* (*HH-MM-SS*), format mult mai cunoscut după denumirea în limba engleză hour-minute-second sau după prescurtarea *HH-MM-SS*.

Formatul în care se salvează un câmp de tip **DATETIME** care stochează atât data cât şi ora este *year-month-day hour-minute-second* (AAAA-LL-ZZ HH-MM-SS sau YYYY-MM-DD HH-MM-SS).

Domeniul de valori este între '1970-01-01 00:00:00' până în ‘2037-01-01 00:00:00’. Formatul în care păstrează valorile pentru **TIMESTAMP** este YYYYMMDDHHMMSS.

În cazul în care o dată calendaristică nu este introdusă în formatul corect sunt convertite la valoarea zero, adică '0000-00-00' dacă este cu câmp de tip **DATE** sau, dacă este şi ora, de exemplu, tipul de date **DATETIME**, '0000-00-00 00-00-00'.

3.10 Modificatori

Modificatorii sunt constrângeri ce pot fi definite pentru câmpurile tabelelor stocate în baza de date. Modificatorii se definesc prin utilizarea unor cuvinte cheie şi a unei sintaxe specifice. Modificatorii ce pot fi întâlniţi în definiţiile de descriere ale unui câmp sunt:

- **NOT NULL** – modificator sau constrângere care stabileşte pentru câmpul la care este definit să nu permită valoarea **NULL**;
- **DEFAULT** – permite stabilirea unei valori implicite pentru acel câmp care are setat acest modifcator;
- **AUTO_INCREMENT** – constrângere care este stabilită pentru un câmp care este cheie primară, în general un id care este incrementat automat la fiecare inserare de înregistrări în tabelă;
- **PRIMARY KEY** – constrângere care defineşte acel câmp ca fiind cheie primară a unei tabele;
- **FOREIGN KEY** – constrângere care defineşte o cheie externă pentru o tabelă;
- **UNIQUE** – constrângere de unicitate care impune valori unice pentru cîmpurile care au definit acest modificator;
- **INDEX** – constrângere care aplică un index pe un câmp al unei tabele.

În continuare prezentăm câteva exemple de folosire a instrucţiunilor prezentate în cadrul limbajului de descriere a datelor (**LDD**). Aceste instrucţiuni se referă doar la structura unei baze de date cu tabelele aferente şi nu au legătură cu valorile din baza de date (cu datele propriu zise).

Considerăm crearea unei aplicaţii care gestionează cărţile aflate într-o bibliotecă. Primul pas pentru realizarea acestei aplicaţii ar fi crearea bazei de date, bază de date care va primi numele *biblioteca*.

Deci instrucţiunea de creare este următoarea:

CREATE DATABASE *bibilioteca*;

Prezentăm în continuare comanda de creare a unei tabele, autori ce conţine informaţii despre autorii cărţilor din această bibliotecă.

```
CREATE TABLE autori(
        id_autor int(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
        nume varchar(200)
);
```

Aşadar, avem o instrucţiune de creare a unei tabele din baza de date, este vorba de o tabelă simplă cu doar 2 câmpuri ce conţine numele autorilor cărţilor. Primul câmp, *id_autor*, conţine valori care identifică în mod unic o înregistrare, deci acest câmp este utilizat drept cheie primară a tabelei.

În descrierea definiţiei acestui câmp se aplică modificatorii pentru cheie primară, valori nenule şi incrementare automată.

Al doilea câmp, *nume*, conţine un şir de caractere de lungime maximă 200 de caractere ce va stoca numele şi prenumele fiecărui autor.

3.11 Chei externe

Instrucţiunea prin care se aplică restricţia de cheie externă este următoarea:

FOREIGN KEY (*nume_câmp*$_1$) REFERENCES *nume_tabelă*(*nume_câmp*$_2$)

FOREIGN KEY şi **REFERENCES** sunt cuvinte cheie în timp ce ***nume_câmp*$_1$** reprezintă câmpul din tabelă care este cheie externă în tabelă, în timp ***nume_câmp*$_2$** reprezintă câmpul la care face referire, adică cheia primară din tabela *nume_tabelă*.

3.12 Index

Introducem în continuare noţiunea de index. Indecşii sunt folosiţi pentru sortarea logică a datelor în vederea îmbunătăţirii vitezei operaţiilor de căutare şi sortare.

Indecşii dintr-o bază de date funcţionează în maniera următoare: datele din cheile primare sunt întotdeauna sortate; este o operaţie pe care programul **SGBD** o execută. Deci, regăsirea anumitor rânduri în funcţie de cheia primară este întotdeauna o operaţie rapidă şi eficientă.

Un index se poate definii pe una sau mai multe coloane. Indecşii îmbunătăţesc performanţele operaţiilor de regăsire dar le degradează pe acelea ale operaţiilor de inserare, modificare şi ştergere a datelor. Când sunt executate aceste operaţii, programul **SGBD** trebuie să actualizeze indexul în mod dinamic. Datele din index pot ocupa o cantitate mare de spaţiu de stocare.

Atunci când se defineşte un index se creează un fişier de index, iar în momentul în care este executată o instrucţiune de interogare pe câmpul indexat, se face practic o căutare în fişierul de index, din acest motiv avem o viteză foarte mare de execuţie.

3.13 Exemple

Următoarea comandă modifică numele tabelei autori în autori_noi:

ALTER TABLE *autori* RENAME TO *autori_noi*;

Aceeaşi comandă poate fi scrisă şi în felul următor:

RENAME TABLE *autori* TO *autori_noi*;

Comanda de mai jos modifică lungimea maximă a câmpului nume:

ALTER TABLE *autori* CHANGE *nume nume* VARCHAR(70);

Iată şi o comandă care adaugă un nou câmp în această tabelă, câmpul *prenume*:

ALTER TABLE *autori* ADD *prenume* VARCHAR(100);

De asemenea, prezentă şi instrucţiunea pentru ştergerea unui cîmp:

ALTER TABLE *autori* DROP *prenume*;

Prezentăm în continuare alte exemple de utilizare a instrucţiunilor din limbajul de desciere a datelor. Astfel, vom realiza o instrucţiune de creare a unei baze de date, apoi vom crea 2 tabele în această bază de date şi vom aplica diverse constrângeri.

Presupunem crearea unei baze de date în care se păstrează informaţii despre angajaţii şi departamentele unei companii. Aşadar, vom avea o bază de date pe care o vom denumi companie şi în această bază de date vom crea două tabele pentru evidenţa departamentelor, respectiv a angajaţilor.

Instrucţiunea pentru crearea bazei de date *companie*:

CREATE DATABASE *companie*;

Instrucţiunea pentru crearea tabelei *departamente*:

CREATE TABLE *departamente*(

***id_departament* int(4) NOT NULL AUTO_INCREMENT PRIMARY KEY,**

***denumire* varchar(100),**

***manager* varchar(100)**

);

Instrucţiunea pentru crearea tabelei *angajati*:

CREATE TABLE *angajati*(

***id_angajat* int(6) NOT NULL AUTO_INCREMENT PRIMARY KEY,**

***nume* varchar(100),**

***prenume* varchar(100),**

***cnp* varchar(13),**

***data_nasterii* date,**

***data_angajarii* date,**

***id_departament* int(4),**

FOREIGN KEY(*id_departament*) REFERENCES *departamente*(*id_departament*)

);

Observăm în definiţia de creare a tabelei *angajati* aplicarea unei constrângeri de tip **FOREIGN KEY** pe câmpul *id_departament* cu referire la câmpul *id_departament* din tabela *departamente*, câmp care este cheie primară în această tabelă.

Aşa cum am precizat anterior, o cheie externă poate fi definită în instrucţiunea de creare a unei tabele, după ce au fost enumerate toate coloanele tabelei sau poate fi creată prin instrucţiuni de modificare a structurii unei tabele din baza de date.

Pentru a prezenta şi cea de-a doua modalitate, mai întâi vom elimina (şterge) constrângerea de cheie externă aplicată câmpului *id_departament* din tabela *angajati*.

Instrucţiunea pentru eliminarea unei constrângeri de tip **FOREIGN KEY** este următoarea:

ALTER TABLE *angajati* DROP FOREIGN KEY *nume_cheie*;

Fiecare **FOREIGN KEY** primeşte automat un nume dacă noi nu am asociat un nume în definirea constrângerii. Pentru a defini un nume unei chei externe, în faţa instrucţiunii de creare şi referire trebuie să mai avem cuvântul cheie **CONSTRAINT** urmat de numele dat cheii externe. De exemplu, în cazul nostru, vom denumi cheia externă *fk_deptAng*.

Deci am fi avut următoarea instrucţiune:

CONSTRAINT *fk_deptAng* FOREIGN KEY(*id_departament*) REFERENCES *departamente*(*id_departament*)

Aşadar, instrucţiunea de ştergere a cheii externe, în cazul nostru concret, va fi:

ALTER TABLE *angajati* DROP FOREIGN KEY *fk_deptAng*;

În continuare, vom crea din nou, constrângerea de tip **FOREIGN KEY** pentru tabela *angajati*:

ALTER TABLE *angajati* ADD CONSTRAINT *fk_deptAng* FOREIGN KEY(*id_departament*) REFERENCES *departamente*(*id_departament*);

De asemenea, avem şi forma în care nu atribuim un nume constrângerii, iar, în acest caz, **SGBD-ul** va atribui un nume automat pentru constrângere:

ALTER TABLE *angajati* ADD FOREIGN KEY(*id_departament*) REFERENCES *departamente*(*id_departament*);

Important de precizat este faptul că, dacă tabela *departamente* nu ar fi fost creată înaintea tabelei *angajati* ar fi aparut o problemă de integritate a datelor în momentul în care încercam să creăm o constrângere de tip cheie externă care ar fi făcut referire la un câmp dintr-o tabelă care nu exista.

În continuare prezentăm alte câteva exemple de instrucţiuni de modificare a structurii unei tabele din baza de date. De exemplu, eliminarea cheii primare din tabela *departamente* se face cu instrucţiunea:

ALTER TABLE *departamente* DROP PRIMARY KEY;

Adăugarea unei chei primare la o tabelă deja creată se poate face cu instrucţiunea:

ALTER TABLE *departamente* ADD PRIMARY KEY (*id_departament*);

Schimbarea dimensiunii unui câmp dintr-o tabelă se face cu instrucţiunea:

ALTER TABLE *angajati* CHANGE *prenume prenume* VARCHAR(150);

Ştergerea unui câmp dintr-o tabelă se poate face cu următoarea instrucţiune:

ALTER TABLE *departamente* DROP *manager*;

Adăugarea unui câmp într-o tabelă se poate face cu următoarea instrucţiune:

ALTER TABLE *departamente* ADD *manager* VARCHAR(150) NOT NULL;

De asemenea, observăm în instrucţiunea anterioară şi stabilirea unei constrângeri de tip **NOT NULL** pentru câmpul adăugat în tabelă.

Redenumirea unui câmp dintr-o tabelă se face astfel:

ALTER TABLE *angajati* CHANGE *data_angajarii data_ang DATE*;

Adăugarea unei constrângeri pe un câmp deja existent se face cu instrucţiunea:

ALTER TABLE *angajati* CHANGE *cnp cnp* VARCHAR(13) NOT NULL;

Aşadar, în această lecţie, au fost prezentate instrucţiuni şi am trecut în revistă, comenzi din limbajul de descriere a datelor (**LDD**). Aceste comenzi afectează structura bazei de date, în timp ce limbajul de manipulare a datelor (**LMD**), de care ne vom ocupa în lecţia următoare, conţine instrucţiuni care afectează înregistrările din tabelele unei baze de date.