Lecţia 4 – Limbajul de manipulare a datelor (LMD)

4.1 Introducere

Limbajul de manipulare a datelor conţine comenzile de actualizare a datelor în interiorul tabelelor dintr-o bază de date, precum şi comanda de regăsire a datelor.

Cele trei comenzi de actualizare a datelor dintr-o tabelă sunt: **INSERT**, **UPDATE** şi **DELETE**.

Iată în continuare prezentată sintaxa **MySQL** a acestor comezi:

Comanda pentru inserarea (adăugarea/introducerea) datelor într-o tabelă a bazei de date este **INSERT** şi are următoarea sintaxa:

INSERT INTO *nume_tabelă* (*câmp*$_1$, *câmp*$_2$, ..., *câmp*$_n$) VALUES (*valoare*$_1$, *valoare*$_2$, ..., *valoare*$_n$);

Instrucţiunea **INSERT** mai are şi alte forme în care poate fi utilizată:

INSERT INTO *nume_tabelă* VALUES (*valoare*$_1$, *valoare*$_2$, ..., *valoare*$_n$);

Această formă poate fi folosită pentru a insera valori în toate câmpurile.

INSERT INTO *nume_tabelă* VALUES ();

Această formă a instrucţiunii de introducere date va insera valorile default (implicite) în toate câmpurile.

INSERT INTO *nume_tabelă* (*câmp*$_1$, *câmp*$_3$) VALUES (*valoare*$_1$, *valoare*$_3$);

Această formă inserează valori doar în câmpurile specificate.

INSERT INTO *nume_tabelă* SET *nume_câmp*$_1$ = *valoare*$_1$, ..., *nume_câmp*$_n$ = *valoare*$_n$;

INSERT INTO *nume_tabelă* VALUES (*valoare*$_1$, *valoare*$_2$, ..., *valoare*$_n$), (*valoare*$_1$, *valoare*$_2$, ..., *valoare*$_n$),... (*valoare*$_1$, *valoare*$_2$, ..., *valoare*$_n$);

Această formă permite inserearea mai multor înregistrări printr-o singură instrucţiune **INSERT**.

Comanda pentru actualizarea sau modificarea unei înregistrări este **UPDATE**.

Sintaxa comenzii de actualizare a datelor este următoarea:

UPDATE *nume_tabelă* SET *nume_câmp* = *valoare* [WHERE condiţie];

Dacă lipseşte clauza **WHERE** înseamnă că se vor actualiza toate înregistrările din tabelă.

Comanda pentru ştergerea datelor dintr-o tabelă este **DELETE**.

Sintaxa comenzii de ştergere a datelor este următoarea:

DELETE FROM *nume_tabelă* [WHERE condiţie];

Dacă lipseşte clauza **WHERE** se vor şterge toate înregistrările din tabelă.

Comanda de regăsire a datelor este **SELECT**.

Sintaxa comenzii **SELECT** este următoarea:

SELECT *câmp*$_1$, *câmp*$_2$,...,*câmp*$_n$ [FROM *nume_tabelă*] [WHERE condiţie] [GROUP BY *câmp*] [HAVING condiţie] [ORDER BY *câmp*] [LIMIT nr_rânduri];

Aşadar, acestea sunt cele 4 instrucţiuni ce compun limbajul de manipulare a datelor. Ele au o sintaxă relativ simplă şi uşor de înţeles. În continuare vom explica fiecare comandă în parte şi vom prezenta câteva exemple de utilizare practică.

4.2 Instrucţiunea INSERT

Reluăm aici sintaxa instrucţiunii **INSERT**:

INSERT INTO *nume_tabelă* (*câmp$_1$, câmp$_2$, ..., câmp$_n$*) VALUES (*valoare$_1$, valoare$_2$, ..., valoare$_n$*);

De asemenea, instrucţiunea de adăugare de informaţii în baza de date mai are următoarele forme:

INSERT INTO *nume_tabelă* VALUES (*valoare$_1$, valoare$_2$, ..., valoare$_n$*);

INSERT INTO *nume_tabelă* SET *nume_câmp$_1$ = valoare$_1$, ..., nume_câmp$_n$ = valoare$_n$*;

Prezentăm în continuare câteva exemple concrete de utilizare a acestei comenzi.

După ce am creat o tabelă într-o bază de date, următorul pas pe care îl facem este să populăm tabela respectivă cu date, deci vom insera înregistrări în tabelă. Aşa cum am prezentat mai sus, comanda **INSERT** este folosită pentru acastă operaţie.

Considerăm tabela *angajaţi* ce conţine informaţii despre angajaţii unei companii (nume, prenume, data naşterii, data angajării, salariul).

Comanda pentru crearea acestei tabele este următoarea:

```
CREATE TABLE angajati (
    id int(11) not null auto_increment primary key,
    nume varchar(70),
    prenume varchar(100),
    data_nasterii date,
    data_angajarii date,
    salariu double(5,2)
);
```

În continuare vom introduce date în această tabelă.

```
INSERT INTO angajati (id, nume, prenume, data_nasterii, data_angajarii, salariu)
VALUES (null, 'Popescu', 'Maria' , '1981-06-08, '2010-02-15', 2000);
```

Se observă că în câmpul ***id*** care este definit cu restricţia **auto_increment** nu este introdusă nici o valoare, deoarece se va insera automat un id întreg care creşte la fiecare înregistrare.

În cazul în care una din înregistrări este ştearsă nu se va aloca id-ul ei unei înregistrări nou introdusă în tabelă.

În câmpul *nume* şi în câmpul *prenume* se introduc şiruri de caractere, în câmpurile *data_nasterii* şi *data_angajarii* se introduc valori de tip **date**, iar în câmpul *salariu* valori de tip **double**.

Însă, nu este obligatoriu ca toate câmpurile să fie prezente în instrucţiunea **INSERT** de introducere a datelor în tabelă, mai ales că nu am aplicat restricţia **NOT NULL** pe câmpurile tabelei (doar cîmpul **id**,

care este cheie primară are această restricţie, dar acest câmp se şi auto incrementează şi atunci nu trebuie specificat în comanda **INSERT**).

În continuare avem un exemplu în care introducem o înregistrare doar cu numele şi prenumele unui angajat:

INSERT INTO *angajati* (nume, prenume) VALUES ('Ionescu', 'George');

Iată şi o instrucţiune care inserează mai multe înregistră în tabelă:

INSERT INTO *angajati* VALUES (null, 'Cristescu', 'Ionut' , '1991-11-28, '2014-01-10', 1500), (null, 'Georgescu', 'Elena' , '1987-01-21', '2015-02-01', 1700), (null, 'Popescu', 'Florin' , '1986-04-16', '2014-07-01', 2000);

Această instrucţiune va introduce 3 înregistrări în tabelă. Observăm faptul că lipseşte partea instrucţiunii în care sunt specificate câmpurile în care se introduc date. Acest lucru este posibil deoarece sunt inserate date în toate câmpurile tabelei. Atenţie, însă, în momentul în care sunt completate valorile care se introduc, trebuie păstrată ordinea în care avem definite câmpurile (coloanele) în tabelă. În caz contrar, putem să avem erori la execuţia comenzii (de exemplu, dacă încercăm să introducem un şir de caractere într-un câmp de tip int, sau o dată într-un câmp de tip double).

Dacă execuţia instrucţiunii va genera o eroare, datele nu sunt inserate în tabelă, chiar dacă o parte din ele sunt date care corespund tipurilor specificate. Instrucţiunea este evaluată în întregime, dacă ea generează o eroare atunci nu se inserează nimic.

4.3 Instrucţiunea UPDATE

Modificarea datelor stocate într-o tabelă a unei baze de date se realizează folosind instrucţiunea **UPDATE**.

Sintaxa unei instrucţiuni **UPDATE** este următoarea:

UPDATE *nume_tabelă* SET *nume_câmp* = *valoare* [WHERE condiţie];

În continuare explicăm această instrucţiune folosită pentru actualizarea/modificarea datelor dintr-o tabelă a unei baze de date, precum şi câteva exemple concrete de utilizare.

Considerând tabela *angajati* pe care am utilizat-o şi la exemplul precedent, pentru actualizarea salariului angajatului cu id-ul 2 se va folosi comanda:

UPDATE *angajati* SET *salariu* = 2000 WHERE *id* = 2;

În cazul în care, într-o instrucţiune de actualizare (modificare), lipseşte clauza **WHERE** se vor modifica informaţiile din toate înregistrările tabelei. Deci, trebuie să fim atenţi atunci când folosim această instrucţiune de actualizare deoarece, în cele mai multe din cazuri, nu se doreşte actualizarea tuturor înregistrărilor dintr-o tabelă.

Pot fi actualizate valorile din mai multe câmpuri printr-o singură instrucţiune **UPDATE**. De asemenea, pot exista mai multe condiţii care se doresc a fi îndeplinite pentru a realiza actualizarea (deci, clauza **WHERE** va avea mai multe condiţii).

În comanda de actualizare **UPDATE** mai poate să apară clauza **LIMIT** care determină aplicarea instrucţiunii de modificare la un număr limitat de înregistrări specificat în clauza **LIMIT**.

Astfel, considerând că tabela *angajati* conţine câteva sute de înregistrări, dacă dorim actualizarea salariului la valoarea 2500 pentru primii 10 de angajaţi înregistraţi vom folosi în instrucţiunea **UPDATE** clauza **LIMIT**, după cum urmează:

UPDATE *angajati* SET *salariu* = 2500 LIMIT 10;

Observăm că în instrucţiunea **UPDATE** nu mai există clauza **WHERE**, dar, totuşi actualizarea nu se face pentru toate înregistrările tabelei, ci doar pentru primele 10 întrucât s-a specificat această limită prin clauza **LIMIT**.

Cu toate că avem şi această variantă de limitare la un anumit număr de înregistrări, totuşi cea mai folosită formă a instrucţiunii **UPDATE**, este cea care conţine una sau mai multe condiţii care trebuie să fie îndeplinite pentru a realiza actualizarea datelor. În acest mod vom şti cu certitudine că nu au fost actualizate înregistrări care nu îndeplinesc condiţiile dorite pentru a se realiza modificarea.

4.4 Instrucţiunea DELETE

Instrucţiunea folosită pentru ştergerea înregistrărilor din baza de date este **DELETE**.

Sintaxa instrucţiunii de ştergere a înregistrărilor dintr-o tabelă este următoarea:

DELETE FROM *nume_tabelă* [WHERE condiţie];

Iată un exemplu de folosire a acestei comenzi, considerăm că avem aceeaşi tabelă pe care am utilizat-o mai înainte, *angajati*, ştergerea înregstrării cu id-ul 3 se face prin următoarea comandă:

DELETE FROM *angajati* WHERE *id* = 3;

La fel ca în cazul instrucţiunii **UPDATE** şi, într-o instrucţiunea **DELETE**, dacă lipseşte clauza **WHERE**, care stabileşte condiţia ce trebuie să fie îndeplinită pentru a se executa ştergerea înregistrărilor, se vor şterge toate înregistrările din tabelă. Deci, trebuie folosită cu atenţie această instrucţiune, astfel încât să fim siguri că am stabilit condiţiile necesare a fi îndeplinite pentru a şterge anumite înregistrări.

În general, se evită folosirea instrucţiunii de ştergere din tabelele unei baze de date a unei aplicaţii aflată în utilizare. Comanda **DELETE** va fi utilizată atunci când ştim sigur că datele respective nu ne mai sunt necesare în baza de date.

În concluzie, acestea sunt cele 3 instrucţiuni folosite pentru actualizarea datelor din tabelele unei baze de date. Sunt instrucţiuni cu sintaxă destul de simplă şi cu o logică uşor de înţeles. Dacă instrucţiunea **INSERT** are mai multe forme, în schimb, instrucţiunile **UPDATE** şi **DELETE** au o singură formă asemănătoare şi uşor de înţeles şi de utilizat.

4.5 Ştergerea tuturor datelor dintr-o tabelă şi resetarea auto incrementului

Comanda care se foloseşte pentru a şterge toate datele dintr-o tabelă este următoarea:

TRUNCATE TABLE *nume_tabelă*;

Această comandă va reseta şi valorile din câmpul unei tabele care se incrementează automat. Astfel, în momentul în care se vor adăuga din nou informaţii, câmpul care are definită proprietatea de auto

incrementare va începe să ia valori de la 1. Această instrucțiune poate fi utilă pentru curățarea datelor de test introduse într-o tabelă, înainte de a porni aplicația cu date reale în tabelele bazei de date.

4.6 Instrucţiunea SELECT

Cea de-a patra comandă care aparţine limbajului de manipulare a datelor este comanda de regăsire a datelor din tabelele unei baze de date. Aceasta este comanda **SELECT** care realizează o selecţie (regăsire) a datelor care îndeplinesc anumite condiţii.

Sintaxa acestei comenzi a fost prezentată în prima parte a lecţiei, dar o vom relua şi aici, urmând ca apoi să explicăm fiecare clauză care poate să apară într-o astfel de instrucţiune de interogare a tabelelor dintr-o bază de date. Aşadar, sintaxa instrucţiunii de regăsire a datelor este următoarea:

SELECT *câmp*$_1$*, câmp*$_2$*,...,câmp*$_n$ [FROM *nume_tabelă*] [WHERE condiţie] [GROUP BY *câmp*] [HAVING condiţie] [ORDER BY *câmp*] [LIMIT nr_rânduri];

După cum se observă din prezentarea completă a sintaxei, instrucţiunea **SELECT** are mai multe clauze pe care le vom explica în continuare.

Clauzele care sunt plasate între paranteze drepte „**[]**” sunt opționale, pot sa lipsească din instrucțiunea **SELECT**. Dacă aceste clauze sunt folosite ele sunt scrise fără paranteze drepte. Deci, parantezele sunt folosite doar în prezentarea sintaxei instrucțiunii cu înțelesul că acele clauze sunt opționale.

Pentru a utiliza comanda de regăsire a datelor, **SELECT**, trebuie să precizăm cel puţin două informaţii: ce anume dorim să selectăm şi locaţia de unde dorim să selectăm. Deci, imediat după cuvântul cheie **SELECT** urmează enumerarea câmpurilor (coloanelor) din tabela din care dorim să le extragem. În cazul în care se doreşte extragerea datelor din toate câmpurile unei tabele se foloseşte caracterul asterisc „*” care reprezintă selectarea tuturor câmpurilor dintr-o tabelă.

Instrucţiunea următoare va extrage şi va afişa toate înregistrările din tabela *angajati*:

SELECT * FROM *angajati*;

Continuăm cu o instrucţiune simplă în care este selectat un singur câmp dintr-o tabelă:

SELECT nume FROM *angajati*;

Această instrucţiune selectează din tabela angajaţi doar numele angajaţilor stocaţi în această tabelă. Specificarea mai multor coloane ale unei tabele într-o instrucţiune **SELECT** se face prin separarea câmpurilor (coloanelor) tabelelor prin virgulă. Pentru a selecta numele, prenumele şi salariul angajaţilor stocate în tabela *angajati* se va utiliza următoarea instrucţiune **SELECT**:

SELECT nume, prenume, salariu FROM *angajati*;

4.7 Clauza WHERE

Prezentăm în continuare clauza **WHERE** a instrucţiunii **SELECT**. Este o clauză opţională, dar este foarte des folosită şi foarte importantă. În cazul în care lipseşte clauza **WHERE** dintr-o interogare, atunci se vor afişa toate înregistrările din coloanele specificate în instrucţiunea **SELECT** din tabela respectivă.

În cele mai multe situaţii însă nu avem nevoie de extragerea tuturor înregistrărilor din tabelă, ci doar de acele înregistrări care îndeplinesc anumite condiţii. Aceste condiţii sunt specificate în clauza **WHERE** în cadrul instrucţiunii de regăsire a datelor. După clauza **WHERE**, într-o interogare sunt specficate diverse condiţii ce se cer îndeplinite pentru a extrage anumite date.

4.8 Operatori folosiţi în clauza WHERE

În interiorul clauzei WHERE putem folosi următorii operatori:

- = este operatorul de **egalitate**, poate fi egalitate între două coloane sau între valoarea dintr-o coloană şi o valoare specificată;
- **!= sau < >** este operatorul **diferit de**, deci verifică dacă două coloane sunt diferite sau o valoarea dintr-o coloană este diferită de o anumită valoare specificată;
- < este operatorul **mai mic**, acest operator compară dacă valoarea dintr-o coloană este strict mai mică decât o valoare din altă coloană sau decât o valoare specificată;
- <= este operatorul **mai mic sau egal**, acest operator compară dacă valoarea dintr-o coloană este mai mică sau egală cu o valoare din altă coloană sau cu o valoare specificată;
- > este operatorul **mai mare**, acest operator compară dacă valoarea dintr-o coloană este strict mai mare decât o valoare din altă coloană sau decât o valoare specificată;
- >= este operatorul **mai mare sau egal**, acest operator compară dacă valoarea dintr-o coloană este mai mare au egală cu o valoare din altă coloană sau cu o valoare specificată;
- **BETWEEN** – compară dacă valoarea dintr-o coloană se află în intervalul specificat în operatorul **BETWEEN**, practic, verifică dacă acea valoare din coloană se află între valorile specificate în **BETWEEN;** forma în care se foloseşte este **BETWEEN** valoare_minimă **AND** valoare_maximă**;**
- **IN** – acest operator testează dacă operandul se regăseşte printre lista de valori care este specificată între paranteze; acest operator este folosit în forma următoare: **IN(valoare$_1$, valoare$_2$,...,valoare$_n$)**; valorile testate cu operatorul **IN** pot fi obţinute şi printr-o instrucţiune **SELECT**, deci poate fi folosit în subinterogări;
- **IS NULL** – verifică dacă valoarea dintr-o coloană a tabelei este **NULL;**
- **IS NOT NULL** – verifică dacă valoarea dintr-o coloană a tabelei nu este **NULL**;

De asemenea, atunci când punem condiţii pe anumite câmpuri (coloane) ce conţin date de tip şir de caractere, mai apare un operator, **LIKE**.

Acest operator este folosit pentru a verifica dacă valoarea de tip şir de caractere dintr-o coloană corpespunde cu un şir de caractere specificat sau, putem folosi aici şi caractere de înlocuire. Astfel avem caracterul de înlocuire **„%”** care are semnificaţia că găseşte orice caracter, indiferent de câte ori apare. De exemplu pentru a găsi toţi angajaţii al căror nume începe cu litera **A** se poate scrie următoarea frază **SELECT**:

SELECT * FROM *angajati* WHERE *nume* LIKE ‘A%’;

Mai există un caracter de înlocuire a unui singur caracter de această dată. Este vorba de caracterul **„_”**. Este mai rar folosit şi acest caracter înlocuieşte un singur caracter, nici mai mult, nici mai puţin. În schimb caracterul de înlocuire **„%”** poate să substituie un caracter, nici un caracter (zero caractere) sau oricât de multe caractere.

În continuare vom prezenta şi alţi operatori folosiţi în clauze **WHERE** mai complexe în care punem mai multe condiţii, deci combinăm mai multe condiţii simple. Astfel, intervin operatorii logici:

- **AND (&&)** – operatorul **„şi”** logic, va returna **adevărat (1)** dacă toţi operanzii sunt adevăraţi, respectiv **fals (0)** dacă cel puţin unul dintre operanzii este fals;
- **OR (||)** – operatorul **„sau”** logic, va returna **adevărat (1)** dacă cel puţin unul dintre operanzi este adevărat, respectiv **fals (0)** dacă toţi operanzii sunt falşi;
- **NOT (!)** – operatorul de negare, va returna **adevărat (1)** dacă expresia negată este falsă, respectiv **fals (0)** dacă expresia negată este adevărată;
- **XOR** – operatorul **„sau exclusiv”** logic, dacă este folosit pentru compararea a doi operanzi va returna **adevărat (1)** dacă unul şi numai unul din aceşti operanzi este adevărat iar celălalt fals, dacă ambii operatori sunt la fel rezultatul returnat va fi **fals (0)**; dacă avem mai mulţi operanzi rezultatul returnat va fi **adevărat (1)** dacă avem un număr impar de operanzi a căror valoare de adevăr este **adevărat (1)**; în caz contrar rezultatul returnat va fi **fals (0)**.

4.9 Clauza GROUP BY

Clauza **GROUP BY** se foloseşte pentru a grupa datele din una sau mai multe coloane pe baza unor criterii. Scopul grupării datelor este calcularea de valori statistice pentru fiecare grup în parte. În acest caz rezultatul cererii va conţine câte o linie pentru fiecare grup identificat. În cazul în care în clauza **GROUP BY** apar mai multe coloane, un grup va fi construit din toate înregistrările care au valori comune pe toate coloanele specificate.

Datele dintr-o tabelă pot fi grupate în funcţie de valorile dintr-o anumită coloană. Astfel, toate valorile egale dintr-o anumită coloană vor forma un grup. Prelucrările datelor din cadrul unui grup se pot face cu ajutorul funcţiilor agregate (funcţii de grup), acestea acţionând asupra datelor din fiecare grup.

Gruparea efectivă se realizează cu clauza **GROUP BY**, aplicată comenzii **SELECT**. În cazul în care dorim filtrarea interogării rezultate în urma unei grupări, nu se mai foloseşte clauza **WHERE**, ci există o nouă clauză, **HAVING**.

Datele din tabela rezultată în urma grupării vor fi sortate după coloana care realizează gruparea.

Într-o instrucţiune **SELECT** putem avea:

- nume de câmpuri (sau expresii în funcţie de acestea): în acest caz se va folosi valoarea primei linii din fiecare grup;
- funcţii agregate: acestea vor acţiona asupra tuturor valorilor coloanei din grup asupra cărora sunt aplicate;

În exemplul anterior, dacă vrem să obţinem numărul de angajaţi din fiecare departament vom executa următoarea instrucţiune **SELECT**:

SELECT *id_dept,* COUNT(*id_angajat*) FROM *angajati* GROUP BY id_dept;

În acest exemplu am folosit şi funcţia **COUNT()** care numără toate înregistrările nenule din coloana *id_dept*, coloană ce conţine id-ul fiecărui departament din baza de date.

4.10 Clauza HAVING

Dacă într-o instrucţiune **SELECT** folosim funcţii de agregare şi avem nevoie să punem condiţii pe rezultatul obţinut în urma utilizării acestor funcţii, atunci vom folosi clauza **HAVING.** Mai simplu de reţinut, clauza **HAVING** se foloseşte când avem funcţii de grup.

Principalele funcţii de grup sunt:

- **COUNT()** – funcţie de numărare;
- **SUM()** – funcţie care returnează suma valorilor din coloana trecută ca argument;
- **MIN()** – funcţie care returnează valoarea minimă din coloana trecută ca argument;
- **MAX()** – funcţie care returnează valoarea maximă din coloana trecută ca argument;
- **AVG()** – funcţie care întoarce media aritmetică a valorilor din coloana primită ca argument.

Funcţia **COUNT()** are mai multe forme:

- **COUNT(*) –** întoarce numărul total de înregistrări din tabelă;
- **COUNT(expr) –** întoarce numărul de valori nenule pentru expresia primită ca argument;
- **COUNT(DISTINCT expr) –** întoarce numărul de valori distincte pentru expresia primită ca argument.

Funcţia **SUM()** întoarce suma valorilor unor expresii care sunt primite ca argument de către funcţie. Valorile nule nu sunt luate în considerare la calculul sumei. Dacă grupul pentru care se calculează suma este vid atunci rezultatul funcţiei **SUM()** va fi **NULL.**

Funcţia **AVG()** întoarce media aritmetică a valorilor din expresia primită ca argument şi poate primi ca argument o coloană a unei tabele sau o expresie.

Funcţia **MIN()** întoarce valoarea minimă dintr-o expresie primită ca argument.

Funcţia **MAX()** întoarce valoarea maximă dintr-o expresie primită ca argument.

Funcţiie de grup **MIN()** şi **MAX()** se pot aplica atât expresiilor numerice, cât şi şirurilor de caractere. În cazul în care se aplică şirurilor de caractere se va folosi ordinea lexicografică pentru determinarea valorii minime, respectiv valorii maxime din expresie.

Dacă instrucţiunea **SELECT,** în care au fost utilizate funcţii de agregare, nu conţine clauza **GROUP BY**, atunci valoarea funcţiilor de agregare va fi calculată pentru întreaga tabelă specificată în clauza **FROM** a instrucţiunii de interogare a bazei de date.

Pentru exemplul anterior, dacă vrem doar afişarea departamentelor cu cel puţin 2 angajaţi instrucţiunea **SELECT** prezentată anterior se transformă astfel:

SELECT *id_dept,* COUNT(*id_angajat*) FROM *angajati* GROUP BY id_dept HAVING COUNT(*id_angajat*) > 1;

De asemenea, un câmp, o expresie sau o tabelă poate primi un **alias**. Un **alias** reprezintă o denumire prin care acea expresie poate fi utilizată în cadrul interogării. De exemplu, în instrucţiunea **SELECT** de mai sus, expresia **COUNT(*id_angajat*)** poate primi un **alias**, adică îi putem asocia un nume pe care să-l folosim mai departe în comanda de regăsire a datelor.

Pentru a defini un **alias** unei expresii se foloseşte cuvântul cheie **AS** urmat de numele asociat acelei expresii, în cazul nostru putem asocia **alias-ul *număr_angajati*** expresiei **COUNT(*id_angajat*).**

Prin urmare, instrucţiunea SELECT anterioară poate fi rescrisă astfel:

SELECT *id_dept,* COUNT(*id_angajat*) AS *nr_angajati* FROM *angajati* GROUP BY id_dept HAVING *nr_ angajati* > 1;

Astfel, putem folosi mai uşor o expresie în cadrul clauzei **HAVING.** În plus, rezultatul acestei interogări, care este o tabelă, va avea ca antet (cap de tabel) sau câmpuri ale tabelei rezultat coloanele ***id_dept*** şi ***nr_angajati***. Dacă nu asociem un **alias** expresiei de numărare, coloanele rezultate ar fi ***id_dept*** şi **COUNT(*id_angajat*)**

4.11 Clauza ORDER BY

Rezultatele obţinute în urma unei instrucţiuni **SELECT** pot fi ordonate în funcţie de anumite câmpuri. Ordonarea acestor rezultate poate fi crescătoare sau descrescătoare. În cazul în care câmpurile folosite pentru ordonare sunt de tip şir de caractere, atunci ordonarea este alfabetică sau în ordine inversă a alfabetului.

Clauza utilizată pentru ordonarea datelor rezultate în urma unei selecţii este **ORDER BY**, după această clauză se specifică numele câmpului după care se face ordonarea şi tipul de sortare (crescător sau descrescător). Pentru sortare în ordine crescătoare avem cuvâtnul cheie **ASC**, iar pentru sortare descrescătoare avem cuvântul cheie **DESC**. De asemenea, se poate face sortare după mai multe câmpuri. În cazul în care, în clauza **ORDER BY,** sunt specificate mai multe câmpuri sortarea se realizează astfel: se sortează datele după valorile din primul câmp, iar în cazul în care în acest câmp avem valori egale (identice) se trece la sortare după următorul câmp specificat în clauza **ORDER BY**, şi aşa mai departe pentru toate câmpurile din clauză. De asemenea, sortarea se poate face crescător după anumite câmpuri şi descrescător după alte câmpuri.

Sortarea implicită a unei interogări este crescătoare, deci, dacă dorim o sortare crescătoare nu este necesar să mai specificăm cuvântul cheie **ASC** după numele coloanei stabilită drept criteriu de sortare.

Dacă vrem să selectăm toţi angajaţii din baza de date sortaţi după nume şi prenume vom realiza următoarea interogare:

SELECT * FROM *angajati* ORDER BY *nume, prenume*;

După cum se observă lipseşte specificarea ordinii de sortare, deci, implicit, se consideră sortare în ordine crescătoare. Interogarea următoare este echivalentă cu cea anterioară, va returna aceleaşi rezultate:

SELECT * FROM *angajati* ORDER BY *nume ASC, prenume ASC*;

În cazul în care se doreşte o sortare descrescătoare a valorilor returnate de interogare, specificarea ordinii de sortare este obligatorie. Avem astfel, următorul exemplu:

SELECT * FROM *angajati* ORDER BY *nume DESC*;

Următoarea interogare realizează o ordonare combinată, descrescătoare după nume şi crescătoare după prenume, adică angajaţii sunt sortaţi după nume în ordine inversă, iar dacă există mai mulţi angajaţi cu acelaşi nume se va realiza o ordonare a acestora după prenume, în ordine alfabetică:

SELECT * FROM *angajati* ORDER BY *nume DESC, prenume ASC*;

4.12 Clauza LIMIT

Ultima clauză a unei instrucţiuni **SELECT** este **LIMIT**. Această clauză, dacă este folosită limitează numărul de înregistrări returnate de interogarea **SELECT**. În clauza **LIMIT** se poate specifica fie un singur număr, care reprezintă numărul de înregistrări pe care instrucţiunea **SELECT** le va întoarce, în acest caz fiind returnate primele *n* înregistrări din totalul de înregistrări returnate, unde *n* este numărul specificat în cadrul clauzei **LIMIT**, fie se pot specifica 2 numere, în acest caz primul reprezintă poziţia de la care va începe returnarea înregistrărilor rezultate în urma interogării, iar cel de-al doilea număr reprezintă numărul de înregistrări care vor fi returnate (cu alte cuvinte poziţia de unde începe şi câte înregistrări vor fi returnate de interogare).

Clauza **LIMIT**, atunci când este utilizată, este ultima în cadrul unei instrucţiuni **SELECT**.

Afişarea primilor 10 angajaţi din tabela în care sunt salvaţi, ordonaţi alfabetic se realizează cu următoarea instrucţiune:

SELECT * FROM *angajati* ORDER BY *nume* LIMIT *10*;

Sintaxa acesteia are una dintre următoarele două forme, aşa cum am precizat şi anterior:

- **LIMIT *n*** – din ceea ce s-ar afişa în mod normal, se afişează doar primele ***n*** linii (înregistrări);
- **LIMIT *m,n*** – din ceea ce s-ar afişa în mod normal, se afişează doar începând de la a ***m+1***-a linie (înregistrare) un număr de ***n*** linii (înregistrări).

Important de reţinut este faptul că prima linie este numerotată cu 0. Aşadar, instrucţiunea exemplu prezentată mai sus ar putea fi rescrisă astfel:

SELECT * FROM *angajati* ORDER BY *nume* LIMIT *0,10*;

Din tabela *angajati* vor fi selectate 10 înregistrări, începând de la poziţia 0. Deci, prima înregistrare rezultată în urma unei selecţii se află pe poziţia 0.

Dacă am fi scris următoarea instrucţiune:

SELECT * FROM *angajati* ORDER BY *nume* LIMIT *1,10*;

Rezultatul întors ar fi tot 10 înregistrări, însă nu va fi afişat primul angajat, ci vor fi afişaţi angajaţii, începând cu al doilea în ordine alfabetică până la al 11-lea.

Dacă în tabela noastră presupunem că am avea 100 de înregistrări, afişarea ultimilor 10 angajaţi sortaţi în ordine alfabetică după nume s-ar realiza cu instrucţiunea:

SELECT * FROM *angajati* ORDER BY *nume* LIMIT 9*0,10*;

Întrucât prima poziţie este 0, dacă avem 100 de linii în tabela rezultat, atunci ultima înregistrare, cea de-a 100, se află la linia 99. Deci, forma corectă a clauzei **LIMIT** pentru cerinţa anterioară este **LIMIT *90,10***, iar nu **LIMIT *91,10***. A doua variantă ar fi afişat doar 9 înregistrări, întrucât începând cu linia 91 nu mai există 10 înregistrări ăn tabelă. Deci, atunci când numărul de înregistrări care ar trebui afişate, specificat în clauza **LIMIT** este mai mare decât numrăul de înregistrări care există în tabelă, de la poziţia (linia) dată, atunci se afişează toate înregistrările rămase. Nu va fi generată nici o eroare din faptul că nu mai sunt în tabelă atâtea înregistrări câte au fost specificate în clauza **LIMIT** pentru afişare, ci vor fi afişate atâtea câte există.

4.13 Clauza DISTINCT

Într-o tabelă, unele coloane pot conţine valori duplicate. Adică, pentru mai multe înregistrări, pe acelaşi câmp, vom avea aceeaşi valoare. Aceasta nu este o problemă, dar uneori vrem să extragem dintr-o tabelă doar valorile diferite (distincte) din tabelă. În acest caz se va folosi clauza **DISTINCT** în cadrul unei interogări **SELECT**. Astfel, în instrucţiunea **SELECT** mai apare un cuvânt cheie, şi anume **DISTINCT** plasat imediat după **SELECT**, după care trebuie specificat câmpul pentru care valorile returnate trebuie să fie distincte (diferite).

Sintaxa este următoarea:

SELECT DISTINCT *nume_câmp* FROM *nume_tabelă*;

De asemenea, trebuie reţinut că această clauză **DISTINCT** poate fi utilzată şi în cadrul funcţiilor de agregare. În acest caz, cuvântul cheie **DISTINCT** este utilizat ca argument al funcţiei de agregare. De exemplu, pentru a număra doar valorile distincte dintr-o coloană.

Un exemplu în acest sens ar fi următoarea instrucţiune **SELECT**, care afişează localităţile de domiciliu ale angajaţilor salvaţi în baza de date a unei companii, în tabela *angajati*. Este evident faptul că, există posibilitatea ca mai mulţi angajaţi să aibă aceeaşi localitate de domiciliu. Deci, pentru a extrage toate localităţile din care avem angajaţi, vom folosi o instrucţiune **SELECT** în care vom avea specificată o clauză **DISTINCT** pentru câmpul *localitate*:

SELECT DISTINCT *localitate* FROM *angajati*;

Fără utilizarea clauzei **DISTINCT**, interogarea ar fi returnat un număr de rezultate egal cu numărul înregistrărilor din tabelă, iar localităţile care se regăsesc de mai multe ori în tabelă ar fi fost afişate de fiecare dată.

Următoarea instrucţiune va returna toate localităţile de domiciliu, la fel ca mai sus, dar va returna şi judeţul pentru fiecare localitate în parte (în acest caz vor exista judeţe care se repetă):

SELECT DISTINCT *localitate,judet* FROM *angajati*;

Iată şi un exemplu de folosire a clauzei **DISTINCT** ca argument într-o funcţie de agregare. De exemplu, dacă într-o tabelă în care sunt salvate spre evidenţă facturile unor clienţi ai unei companii, vrem

să ştim câţi clienţi au facturi emise de companie, avem nevoie de folosirea acestui argument, **DISTINCT**, în cadrul funcţiei **COUNT**:

SELECT COUNT(DISTINCT *cod_client*) FROM *facturi*;

Observăm foarte limpede că absenţa argumentului **DISTINCT** din cadrul funcţiei **COUNT** ar duce la numărarea tuturor înregistrărilor din tabela *facturi* unde câmpul *cod_client* este nenul. Dar dacă am fi avut mai multe facturi emise aceluiaşi client, ceea ce este foarte posibil, rezultatul obţinut ar fi fost alterat, adică nu ar fi corespuns cerinţei noastre de a afla numărul de clienţi unici pentru care există facturi emise de către companie.

În această lecţie am tratat pe larg comenzile aparţinând **L**imbajului de **M**anipulare a **D**atelor, iar accentul a fost pus pe instrucţiunea de regăsire a datelor pentru care au fost precizate şi explicate toate clauzele posibile. În continuare vor fi tratate aspecte legate de operatorii întâlniţi în **MySQL**, o parte din ei au fost prezentaţi şi în cadrul acestei lecţii, precum şi de funcţiile predefinte pe care **MySQL** le pune la dispoziţia utilizatorilor.