Cuprins

7. MySQL

7.1. SQL Data Definition Language

Data Definition Language (pe scurt DML) este partea limbajului SQL care permite crearea, modificarea şi ştergerea bazelor de date, a tabelelor din baza de date, a indecşilor, legăturilor între tabele şi a constrângerilor care se impun datelor şi legăturilor între tabele.

Instrucţiunile din limbajul SQL se numesc cereri ("query").

7.1.1. Crearea şi selectarea unei baze de date

Sintaxa comenzii prin care se creează o bază de date nouă este:

```
CREATE DATABASE nume_baza_de_date;
```

Pentru a vedea bazele de date existente se foloseşte comanda:

```
SHOW DATABASES;
```

Selectarea bazei de date de lucru curente se face prin:

```
USE nume_baza_de_date;
```

Exemplu:

```
mysql> CREATE DATABASE exemplu;
Query OK, 1 row affected (0.45 sec)

mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| cdcol              |
| exemplu            |
| mysql              |
| pensiuni           |
| phpmyadmin         |
| test               |
| webauth            |
+--------------------+
8 rows in set (0.00 sec)

mysql> USE exemplu;
Database changed
```

7.1.2. Crearea şi ştergerea tabelelor

Crearea unei baze de date este uşoară dar aceasta este goală, după cum se poate vedea rulând comanda **`SHOW TABLES;`**

Crearea unei tabele noi se face cu comanda **`CREATE TABLE`**.

```
mysql> SHOW TABLES;
Empty set (0.00 sec)
```

Vom crea tabela *judete* cu două coloane:

- *id* – această coloană va servi ca şi cheie primară pentru tabelă. Acestei coloane îi vom specifica faptul că vrem să se actualizeze automat în momentul în care introducem o linie nouă în tabelă.
- *nume* – coloană care va conţine numele judeţului.

```
mysql> CREATE TABLE judete (
    -> id int(11) NOT NULL auto_increment,
    -> nume varchar(100) NOT NULL,
    -> PRIMARY KEY (id)
    -> );
Query OK, 0 rows affected (0.11 sec)

mysql> SHOW TABLES;
+------------------+
| Tables_in_exemplu |
+------------------+
| judete            |
+------------------+
1 row in set (0.00 sec)
```

Ştergerea unei tabele se face cu comanda **DROP** *nume_tabelă* :

```
mysql> CREATE TABLE de_sters (id int);
Query OK, 0 rows affected (0.08 sec)

mysql> SHOW TABLES;
+----------------+
| Tables_in_test |
+----------------+
| de_sters       |
+----------------+
1 row in set (0.00 sec)

mysql> DROP TABLE de_sters;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW TABLES;
Empty set (0.00 sec)
```

7.1.2.1. Tipurile de date MySQL

Tipurile de dată utilizate în MySQL se pot împărţi în trei categorii:

- de tip text – acestea se folosesc pentru specificarea coloanelor ce vor reţine şiruri de caractere sau texte mai lungi;
- numerice – coloane care reţin valori numerice întregi, flotante (în virgulă mobilă) sau zecimale;
- de tip dată.

	Nume tip	Utilizare	Descriere
Text	`char`	`char(14)`	Şir de caractere de lungime fixă, L < 255 caractere.

	varchar	varchar(17)	Ca şi *char* dar lungimea este variabilă. Un caracter suplimentar va fi adăugat sistematic pentru a indica lungimea efectivă.
	tinytext	tinytext	Text de lungime variabilă, L < 255. Ca şi varchar(255).
	text	text	Text de lungime variabilă, L < 65.535 caractere. Pot servi la indexare, luându-se în consid. primele 255 caractere.
	mediumtext	mediumtext	Ca şi text, L < 16.777.215 caractere.
	longtext	longtext	Ca şi text, L < 4.292.967.295 caractere.
	enum	enum ('v1', 'v2', ...) [default 'v1']	Limitează domeniul valorilor posibile. Max. 65535 valori.
Numeric	int / integer	int(6) [unsigned] [zero fill] [auto_increment]	-2.147.483.648 < n < 2.147.483.647 (0 < n < 4.294.967.295 pentru unsigned). Numărul dintre paranteze indică lungimea câmpului folosit la imprimare. Autoincrement se foloseşte frecvent la definirea câmpului ca fiind cheie primară.
	tinyint	tinyint(4) [unsigned] [zero fill]	-127 < n < 127 (0 < n < 255 pentru unsigned)
	mediumint	mediumint(5) [unsigned] [zero fill]	-8.388.608 <n < 8.388.608 (0 < n < 16.772.615 pentru unsigned).
	bigint	bigint(12) [unsigned] [zerofill]	-9 E18 < n < 9 E18
	float	float(m,d) [zerofill]	m = mărimea câmpului la imprimare şi d = numărul de cifre zecimale dorit (d < m).
	double / real	double(m,d)	idem, dublă precizie.
	decimal / numeric	decimal [(m [,d])] [zerofill]	Număr păstrat ca şir de caractere, pe m poziţii. Dacă d=0 nu se va păstra o poziţie pentru punctul zecimal.
Dată	date	date	data în format yyyy-mm-dd. 1000-01-01 < d < 9999-12-31
	datetime	datetime [null / not null] [default]	data şi ora în format yyyy-mm-dd hh:mm:ss
	time	time	ora în format hh:mm:ss
	year	year(4)	anul în format yyyy

7.1.3. Modificarea tabelelor

Se poate ca în anumite cazuri să fie necesar să modificăm structura tabelelor existente în baza de date, fără a pierde datele deja existente în tabelă.

MySQL ne permite să facem acest lucru prin comanda cu următoarea sintaxă simplificată:

```
ALTER TABLE nume_tabelă
    ADD [COLUMN] definiţie_coloană [FIRST | AFTER coloană]
  | DROP [COLUMN] coloană
  | RENAME [TO] nume_nou_tabelă
  | CHANGE [COLUMN] coloană definiţie_nouă_coloană
```

Pentru a exemplifica comanda **ALTER TABLE** vom creea o tabelă de lucru cu următoarea structură:

```
mysql> CREATE TABLE t1 (
    ->   id int NOT NULL auto_increment,
    ->   c1 varchar(100),
    ->   PRIMARY KEY (id)
    -> );
```

7.1.3.1. Adăugarea / ştergerea unei coloane

Pentru a adăuga şi a şterge o coloană, folosim clauzele **ADD** [**COLUMN**] şi **DROP** [**COLUMN**].

```
mysql> ALTER TABLE t1 DROP COLUMN c1;
Query OK, 0 rows affected (0.13 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE t1 ADD COLUMN c2 varchar(200);
Query OK, 0 rows affected (0.48 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESCRIBE t1;
+-------+--------------+------+-----+---------+----------------+
| Field | Type         | Null | Key | Default | Extra          |
+-------+--------------+------+-----+---------+----------------+
| id    | int(11)      | NO   | PRI | NULL    | auto_increment |
| c2    | varchar(200) | YES  |     | NULL    |                |
+-------+--------------+------+-----+---------+----------------+
2 rows in set (0.02 sec)
```

În exemplul următor vom şterge coloana *c1* şi vom adăuga coloana *c2*:

```
mysql> ALTER TABLE t1 RENAME t2;
Query OK, 0 rows affected (0.01 sec)

mysql> SHOW TABLES;
+------------------+
| Tables_in_exemplu |
+------------------+
| t2               |
+------------------+
1 rows in set (0.00 sec)
```

7.1.3.2. Redenumirea unei tabele

Putem să redenumim o tabelă deja existentă prin folosirea clauzei **RENAME** [**TO**]

7.1.3.3. Modificarea unei coloane

Pentru a schimba structura unei coloane deja adăugate, fie că vrem să o redenumi doar sau vrem să schimbăm tipul de dată al acesteia, folosim clauza **CHANGE** [**COLUMN**] :

```
mysql> ALTER TABLE t2 CHANGE COLUMN c2 c3 varchar(255);
Query OK, 0 rows affected (0.53 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESCRIBE t2;
+-------+--------------+------+-----+---------+----------------+
| Field | Type         | Null | Key | Default | Extra          |
+-------+--------------+------+-----+---------+----------------+
| id    | int(11)      | NO   | PRI | NULL    | auto_increment |
| c3    | varchar(255) | YES  |     | NULL    |                |
+-------+--------------+------+-----+---------+----------------+
2 rows in set (0.02 sec)
```

7.1.4. Exerciţii SQL DDL

7.1.4.1. Exerciţiul 1

Folosind clientul de linie de comandă, creaţi o nouă bază de date *pensiuni*. În aceasta, creaţi următoarele tabele:

- pensiuni
  - **id**, *int, auto_increment, primary key, not null*
  - **nume**, *varchar(500), not null*
  - **descriere**, *varchar(1000), null*
  - **judet_id**, *tinyint, not null*
  - **proprietar_id**, *smallint, not null*
  - **nr_locuri**, *tinyint, not null*
- judete
  - **id**, *tinyint, auto_increment, primary key, not null*
  - **nume**, *varchar(50), not null*
- proprietari
  - **id**, *int, auto_increment, primary key, not null*
  - **nume**, *varchar(100), not null*
  - **adresa**, *varchar(500), null*
  - **telefon**, *varchar(20), null*
  - **email**, *varchar(50), null*
- imagini
  - **id**, *int, auto_increment, primary key, not null*
  - **nume**, *varchar(500), null*
  - **descriere**, *varchar(1000), null*
  - **mime_type,** *varchar(100), not null*
  - **cale**, *varchar(500), not null*
  - **data**, *datetime, not null*
  - **pensiune_id**, *int, not null*

7.1.4.2. Exerciţiul 2

Folsind comenzile pentru modificarea structurii tabelelor, efectuaţi următoarele schimbări asupra structurii bazei de date create la exerciţiul 1:

- adăugaţi coloana **website** la tabela pensiuni, cu tipul de dată *varchar(200)*;
- schimbaţi în *text* tipul de dată al coloanei **descriere** din tabela pensiuni;

7.2. SQL Data Manipulation Language (DML)

7.2.1. Comanda INSERT

`INSERT` ne permite să introducem înregistrări noi în tabelele din baza de date. Sintaxa simplificată a comenzii **`INSERT`** este următoarea:

```
INSERT
    [INTO] nume_tabelă [(nume_coloană,...)]
    VALUES ({expresie | DEFAULT},...),(...),...
```

Se poate opta pentru mai multe variante atunci când se inserează înregistrări într-o tabelă.

Cea mai simplă metodă este aceea în care se specifică toate coloanele de inserat.

```
mysql> INSERT INTO t1 VALUES (1,1,1);
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM t1;
+----+------+------+
| id | c1   | c2   |
+----+------+------+
|  1 |    1 |    1 |
+----+------+------+
1 row in set (0.00 sec)
```

Putem să optăm şi pentru varianta în care se specifică doar anumite coloane ale tabelei. Acest lucru este util atunci când avem o coloană cu opţiunea *auto_increment* sau care are setată o valoare *default*.

```
mysql> INSERT INTO t1 (c1, c2) VALUES (2,2);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM t1;
+----+------+------+
| id | c1   | c2   |
+----+------+------+
|  1 |    1 |    1 |
|  2 |    2 |    2 |
+----+------+------+
2 rows in set (0.00 sec)
```

Se poate observa în exemplul de mai sus că am specificat doar coloanele *c1* şi *c2*, iar MySQL a completat singur valoarea câmpului *id* datorită faptului că această coloană are specificat atributul *auto_increment.*

7.2.2. Comanda SELECT

Operaţia de regăsire a datelor în baza de date se numeşte *interogare*. Interogările se fac folosind comanda **SELECT** care este probabil cea mai utilizată comandă din tot limbajul SQL.

SELECT este totodată şi cea mai complexă comandă SQL, deoarece trebuie să permită extragerea seturilor de date în orice formă doreşte programatorul.

Comanda **SELECT** este formată dintr-o serie de *clauze* care se pot combina pentru a obţine rezultatul dorit. În continuare vom prezenta o variantă simplificată a comenzii **SELECT**, arătând cele mai folosite clauze.

Sintaxa generală simplificată comenzii **SELECT** este următoarea:

```
SELECT
    [DISTINCT]
    expresie_select, ...
    [FROM referinţe_tabele
    [WHERE condiţie_where]
```

```
    [ORDER BY nume_coloană
      [ASC | DESC], ...]
    [LIMIT {[primul_rezultat,] nr_rezultate |
      nr_rezultate OFFSET primul_rezultat}]]
```

7.2.2.1. Clauza FROM şi ţinta interogării

Fiecare *expresie_select* indică o coloană pe care vrem să o extragem din tabelă. Trebuie să existe cel puţin o *expresie_select* în orice interogare. Pentru cazul în care se doreşte selectarea tuturor coloanelor dintr-o tabelă se poate folosi operatorul special „*”.

FROM *referinţe_tabele* indică o tabelă sau mai multe tabele din care vrem să extragem date. În cazul interogării mai multor tabele aici se poate specifica lista de tabele din care selectăm, prin folosirea operatorului **JOIN**.

Cea mai simplă interogare pe care o putem scrie folosind **SELECT** este aceea care returnează toate liniile dintr-o tabelă:

```
mysql> select * from pensiuni;
+----+-------+-----------+---------+---------------+-----------+
| id | nume  | descriere | judet_id | proprietar_id | nr_locuri |
+----+-------+-----------+---------+---------------+-----------+
|  1 | Ana   | NULL      |        2 |             1 |        10 |
|  2 | Batea | NULL      |        1 |             2 |        12 |
|  3 | Man   | NULL      |        7 |             4 |        11 |
+----+-------+-----------+---------+---------------+-----------+
3 rows in set (0.00 sec)
```

Dacă vrem să afişăm doar coloanele *nume* şi *nr_locuri* va trebui să le cerem în mod explicit doar pe acestea:

```
mysql> SELECT nume, nr_locuri FROM pensiuni;
+-------+-----------+
| nume  | nr_locuri |
+-------+-----------+
| Ana   |        10 |
| Batea |        12 |
| Man   |        11 |
+-------+-----------+
3 rows in set (0.00 sec)
```

7.2.2.2. Clauza WHERE

WHERE *condiţie_where* precizează o condiţie sau un set de condiţii pe care trebuie să le îndeplinească liniile selectate. Dacă nu se precizează clauza, se extrag toate liniile.

Pentru găsi doar pensiunile care au numărul de locuri mai mare decât 10 vom folosi clauza **WHERE**:

```
mysql> SELECT nume, nr_locuri
    -> FROM pensiuni
    -> WHERE nr_locuri > 11;
+-------+-----------+
| nume  | nr_locuri |
+-------+-----------+
| Batea |        12 |
+-------+-----------+
1 row in set (0.00 sec)
```

Expresia *condiţie_where* din clauza **WHERE** acceptă următorii operatori:

Operator	Descriere
=	Egal
<>, !=	Diferit
>	Mai mare
<	Mai mic
>=	Mai mare sau egal
<=	Mai mic sau egal
BETWEEN	Între două limite inclusive
LIKE	Căutare după şabloane

7.2.2.3. Clauza ORDER BY

ORDER BY *nume_coloană* [**ASC** | **DESC**] permite ordonarea crescătoare sau descrescătoare rezultatelor după coloanele specificate.

Putem sorta pensiunile selectate după numărul de locuri, spre exemplu:

```
mysql> SELECT nume, nr_locuri
    -> FROM pensiuni
    -> ORDER BY nr_locuri DESC;
+-------+-----------+
| nume  | nr_locuri |
+-------+-----------+
| Batea |        12 |
| Man   |        11 |
| Ana   |        10 |
+-------+-----------+
3 rows in set (0.00 sec)
```

7.2.2.4. Clauza LIMIT

De obicei, în aplicaţiile reale, numărul înregistrărilor dintr-o tabelă poate uşor să atingă numărul sutelor, miilor sau chiar mai mult. În acest caz este util să cerem bazei de date doar o parte din rezultatele care respectă criteriile interogării.

Clauza **LIMIT** se poate scrie în două moduri, amândouă având acelaşi efect:

- **LIMIT** *nr_rezultate* **OFFSET** *primul_rezultat*
- **LIMIT** *primul_rezultat*, *nr_rezultate*. Această formă este specifică MySQL.

În exemplul de mai jos folosim **LIMIT** pentru a obţine succesiv câte o pensiune:

```
mysql> SELECT nume, nr_locuri FROM pensiuni LIMIT 0, 1;
+------+-----------+
| nume | nr_locuri |
+------+-----------+
| Ana  |        10 |
+------+-----------+
1 row in set (0.00 sec)

mysql> SELECT nume, nr_locuri FROM pensiuni LIMIT 1, 1;
+-------+-----------+
| nume  | nr_locuri |
+-------+-----------+
| Batea |        12 |
+-------+-----------+
```

```
1 row in set (0.00 sec)
```

7.2.2.5. Aliasuri pentru coloane şi tabele

Deoarece interogările pot să devină destul de complexe, MySQL ne pune la dispoziţia o modalitate de etichetare a numelor coloanelor şi tabelelor în cadrul unei interogări. Această facilitate se numeşte *aliasing* şi este o unealtă puternică atunci când avem de specificat condiţii mai complexe sau efectuăm interogări care implică mai multe tabele.

Operaţia de *aliasing* se face folosind expresia **AS** *nume_alias* imediat după coloana sau tabela căreia vrem să îi asignăm aliasul.

În exemplul de mai jos folosim aliasuri atât pentru coloane (*NumePensiune* şi *NumarDeLocuri*) cât şi pentru tabele (*p*):

```
mysql> SELECT nume AS NumePensiune, nr_locuri AS NumarDeLocuri
    -> FROM pensiuni AS p
    -> WHERE p.nr_locuri > 10;
+--------------+---------------+
| NumePensiune | NumarDeLocuri |
+--------------+---------------+
| Batea        |            12 |
| Man          |            11 |
+--------------+---------------+
2 rows in set (0.00 sec)
```

Sunt de notat două lucruri la exemplul de mai sus:

- în antetul rezultatului selecţiei apar aliasurile coloanelor selectate;
- la specificarea condiţiilor de selecţie putem să specificăm aliasul tabelelor; acest lucru va fi util atunci când vom selecta din mai multe tabele şi vrem să facem distincţie între coloanele acestora.

7.2.2.6. Selecţie din mai multe tabele

În realitate, pentru a folosi la maxim informaţiile din baza de date, avem nevoie să facem interogări pe mai multe tabele, ţinând cont de relaţiile dintre acestea.

De exemplu, pentru a obţine lista pensiunilor şi a judeţelor din care fac parte acesta, vom scrie:

```
mysql> SELECT p.nume AS NumePensiune, j.nume AS NumeJudet
    -> FROM pensiuni AS p
    ->   JOIN judete AS j ON j.id = p.judet_id;
+--------------+-----------+
| NumePensiune | NumeJudet |
+--------------+-----------+
| Ana          | Alba      |
| Batea        | Cluj      |
| Man          | Maramures |
| Scarisoara   | Alba      |
+--------------+-----------+
4 rows in set (0.00 sec)
```

Primul lucru de remarcat este folosirea aliasurilor pentru tabelele *pensiuni* şi *judeţe*. Acest lucru este necesar pentru că ambele tabele au coloana *nume*, iar dacă nu am folosi în mod explicit aliasuri, nu am putea distinge între coloanele care vrem să apară ca şi rezultat în urma

interogării. Din moment ce am etichetat *pensiuni* şi *judeţe* cu *p*, respectiv *j*, expresia de selectat se va scrie folosind aceste etichete:

```
SELECT p.nume AS NumePensiune, j.nume AS NumeJudet
```

În traducere, MySQL primeste instrucţiuni să afişeze coloana *nume* din *pensiuni* şi coloana *nume* din *judeţe*, iar acestea să fie afişate ca şi *NumePensiune*, respectiv *NumeJudeţ*.

Al doilea lucru pe care îl remarcăm este folosirea operatorului **JOIN**. Dacă aliasurile ne ajutau să speficicăm *ce* să fie selectat, operatorul **JOIN** ne spune *cum* să se efectueze selecţia. În cazul nostru, expresia:

```
JOIN judete AS j ON j.id = p.judet_id
```

spune interpretorului MySQL să asocieze tabela *pensiuni* cu tabela *judeţe* după criteriul: "coloana *id* din *judeţe* corespunde coloanei *judet_id* din pensiuni".

7.2.2.6.1. INNER JOIN

Tipul de JOIN din exemplul de mai sus se numeşte *inner join*. Tablele asociate cu acest tip de join vor returna doar acele valori care sunt *asociabile* între cele două tabele. Acest tip de join se mai întâlneşte şi sub numele de *cross join*.

Cu alte cuvinte, pe exemplul nostru, dacă în tabela *pensiuni* avem înregistrări la care valoarea câmpului *judet_id* este **NULL**, acestea nu vor apărea la rezultate. La fel, nu vor apărea la rezultate nici judeţele pentru care nu există introdusă nici o pensiune în baza de date.

7.2.2.6.2. LEFT JOIN

Să presupunem că vrem să obţinem o listă cu toate înregistrările din tabela *pensiuni* şi numele judeţelor din care fac parte, dar vrem să apară şi acelea care nu au asociat un judeţ.

Soluţia este folosirea operatorului **LEFT JOIN**, prin care instruim interpretorul că vrem toate înregistrările din tabela din stânga operatorului **JOIN**.

```
mysql> SELECT p.nume AS NumePensiune, j.nume AS NumeJudet
    -> FROM pensiuni AS p
    ->   LEFT JOIN judete AS j ON j.id = P.judet_id;
+--------------+-----------+
| NumePensiune | NumeJudet |
+--------------+-----------+
| Ana          | Alba      |
| Batea        | Cluj      |
| Man          | Maramures |
| Fara Judet   | NULL      |
| Scarisoara   | Alba      |
+--------------+-----------+
5 rows in set (0.00 sec)
```

Observăm că în dreptul rezultatului „Fara Judet" apare valoarea **NULL** în locul numelui unui judeţ din cauză că pensiunea nu are asociată nici o valoare la tabela judeţe. Dacă am fi folosit **JOIN** simplu, acest rezultat nu ar fi apărut.

7.2.2.6.3. RIGHT JOIN

Funcţionalitatea operatorului **RIGHT JOIN** o oglindeşte pe cea a lui **LEFT JOIN**; adică va returna toate înregistrările tabelei din dreapta operatorului **JOIN**.

La exemplul nostru, putem folosi **RIGHT JOIN** pentru a obţine lista tuturor pensiunilor cu judeţul din care fac parte, iar judeţele care nu au nici o pensiune vor avea în dreptul lor **NULL**.

```
mysql> SELECT p.nume AS NumePensiune, j.nume AS NumeJudet
    -> FROM pensiuni AS p
```

```
    ->   RIGHT JOIN judete AS j ON j.id = P.judet_id;
+--------------+---------------+
| NumePensiune | NumeJudet     |
+--------------+---------------+
| Batea        | Cluj          |
| Ana          | Alba          |
| Scarisoara   | Alba          |
| NULL         | Harghita      |
| NULL         | Mures         |
| NULL         | Timis         |
| NULL         | Sibiu         |
| Man          | Maramures     |
| NULL         | Prahova       |
| NULL         | Brasov        |
| NULL         | Olt           |
+--------------+---------------+
11 rows in set (0.00 sec)
```

7.2.2.7. Clauza GROUP BY şi funcţii de agregare

Clauza GROUP BY ne permite să grupăm rezultatele unei interogări în funcţie de elementele specificate în lista de selecţie. Operaţia de grupare este utilă în special atunci când o combinăm cu funcţii de agregare, cum ar fi funcţia *count().*

Ne propunem să obţinem lista judeţelor care au pensiuni şi numărul de pensiuni în fiecare judeţ:

```
mysql> SELECT j.nume AS Judet, COUNT(p.nume) AS NrPensiuni
    -> FROM judete AS j
    ->   INNER JOIN pensiuni AS p ON j.id = p.judet_id
    -> GROUP BY j.nume;
+-----------+------------+
| Judet     | NrPensiuni |
+-----------+------------+
| Alba      |          2 |
| Cluj      |          1 |
| Maramures |          1 |
+-----------+------------+
3 rows in set (0.00 sec)
```

Se pot observa cele două elemente noi:

- funcţia de agregare **`COUNT()`** apare în lista de selecţie şi primeşte aliasul *NrPensiuni;*
- clauza **`GROUP BY`** apare la finalul interogării, cu parametrul *j.nume*; rezultatul este că fiecare nume de judeţ va apărea o singură dată în lista de rezultate, iar funcţia de agregare **`COUNT()`** se va aplica pentru fiecare judet.

Lista funcţiilor de agregare folosite frecvent:

Funcţia	Descriere	Exemple
`COUNT()`	Calculează numărul de înregistrări din interogare.	`SELECT COUNT(*) FROM t1;` (afişează numărul total de înregistrări din t1) `SELECT COUNT(*) FROM t1 WHERE c1 = 1;`

		(afişează numărul de înregistrări din t1 care au valoarea câmpului c1 egală cu 1)
SUM()	Calculează suma valorilor de pe o coloană.	`SELECT tip_produs, SUM(pret) FROM produse GROUP BY tip_produs;` (calculează suma preţurilor tuturor produselor care au acelaşi tip)
AVG()	Calculează media aritmetică a valorilor de pe o coloană.	`SELECT tip_produs, AVG(pret) FROM produse GROUP BY tip_produs;` (calculează media preţurilor tuturor produselor care au acelaşi tip)
MIN()	Calculează minimul.	`SELECT nume, tip_produs, MIN(pret) FROM produse GROUP BY tip_produs HAVING MIN(pret) < 10;` (găseşte numele, tipul şi preţul la produsele care sunt mai ieftine de 10 lei)
MAX()	Calculează maximul.	vezi MIN()

7.2.2.8. Clauza HAVING

Apare o problemă în cazul folosirii clauzei **`GROUP BY`**: coloanele după care se grupează nu mai pot să facă parte dintr-o condiţie **`WHERE`**. Spre exemplu, dacă vrem să obţinem lista judeţelor care au cel puţin două pensiuni, suntem nevoiţi să folosim clauza **`HAVING`** pentru a impune condiţia:

```
mysql> SELECT j.nume AS Judet, COUNT(p.nume) AS NrPensiuni
    -> FROM judete AS j
    ->   INNER JOIN pensiuni AS p ON j.id = p.judet_id
    -> GROUP BY j.nume
    -> HAVING NrPensiuni >= 2;
+-------+------------+
| Judet | NrPensiuni |
+-------+------------+
| Alba  |          2 |
+-------+------------+
1 row in set (0.00 sec)
```

7.2.2.9. Sub-interogări

O sub-interogare este o propoziţie SELECT imbricată în altă propoziţie select.

MySQL ne permite să scriem interogări de forma:

```
SELECT * FROM t1
WHERE column1 = (SELECT column1 FROM t2);
```

În acest exemplu, **`SELECT`** `*` **`FROM`** `t1` … este interogarea (sau propoziţia) externă, iar (**`SELECT`** `column1` **`FROM`** `t2`) este sub-interogarea. Spunem că o sub-interogare este *imbricată* în interogarea externă; interogările se pot imbrica pe multe nivele.

Avantajele principale ale sub-interogărilor sunt:

- permit ca interogările să fie structurate, izolând astfel fiecare parte a propoziţiei;
- oferă alternative la scrierea interogărilor care ar necesita altfel join-uri complexe;

- de cele mai multe ori o interogare scrisă structurat (cu sub-interogări) este mai uşor de citit de către oameni decât echivalentul ei scris cu join-uri.

O sub-interogare poate să returneze un scalar (o singură valoare, cum ar fi rezultatul aplicării unui COUNT()), un tabel cu o singură coloană şi mai multe linii, sau un tabel cu mai multe linii şi coloane. Acestea se numesc sub-interogări de tip *scalar*, *coloană*, respectiv *tabel*. Sub-interogările care returnează un anumit tip de rezultat pot fi adesea folosite doar în anumite cazuri.

7.2.2.9.1. Sub-interogări scalar

În forma cea mai simplă, o sub-interogare întoarce o singură valoare. O sub-interogare scalară este un operand simplu care poate fi folosit aproape în orice situatie se foloseste o singură valoare de coloană sau literal.

Exemplul următor foloseşte o sub-interogare pentru a insera într-o tabelă numărul de înregistrări al altei tabele:

```
mysql> CREATE TABLE t1 (nr int);
Query OK, 0 rows affected (0.09 sec)

mysql> INSERT INTO t1
    -> VALUES ( (SELECT COUNT(*) FROM pensiuni) );
Query OK, 1 row affected (0.02 sec)

mysql> SELECT * FROM t1;
+------+
| nr   |
+------+
|    5 |
+------+
1 row in set (0.00 sec)
```

7.2.2.9.2. Sub-interogări cu ANY, IN, SOME şi ALL

Cuvântul cheie **ANY**, care trebuie să urmeze după un operator de comparaţie înseamnă: „întoarce *adevărat* dacă este *adevărată* comparaţia pentru cel puţin unul dintre valorile pe care le întoarce sub-interogarea”.

Putem scrie următoarea interogare:

```
SELECT s1 FROM t1 WHERE s1 > ANY (SELECT s1 FROM t2);
```

Interogarea de mai sus va extrage toate valorile câmpului *s1* din tabela *t1* care sunt mai mari decât cel puţin unul dintre câmpurile s1 existente în tabela t2.

Cuvântul cheie **IN** este echivalentul lui „**= ANY**”, deci următoarele două propoziţii sunt echivalente:

```
SELECT s1 FROM t1 WHERE s1 = ANY (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 IN    (SELECT s1 FROM t2);
```

Cuvântul cheie **SOME** este echivalent lui **ANY**.

Cuvântul cheie ALL se foloseşte în acelaşi context ca şi ANY şi înseamnă: „întoarce *adevărat* dacă este *adevărată* comparaţia pentru toate valorile pe care le întoarce sub-interogarea”.

În cazul acesta, interogarea:

```
SELECT s1 FROM t1 WHERE s1 > ALL (SELECT s1 FROM t2);
```

va întoarce acele câmpuri *s1* din *t1* care sunt mai mari decât toate câmpurile *s1* din *t2*.

7.2.2.9.3. Sub-interogări tabel

Până acum am discutat doar despre sub-interogări tip scalar şi tip coloană (cele cu ANY, IN şi ALL). Sub-interogările tip tabel sunt acelea care întorc mai multe linii şi mai multe coloane.

Exemplu:

```
SELECT * FROM t1
WHERE (1,2) = (SELECT c1, c2 FROM t2);
```

Această interogare va returna toate înregistrările din tabela *t1* dacă în tabela *t2* există o înregistrare cu *c1=1* şi *c2=2*.

7.2.3. Comanda UPDATE

Comanda **UPDATE** permite modificarea valorilor câmpurilor înregistrărilor dintr-un tabel.

Sintaxa comenzii **UPDATE** este următoarea:

```
UPDATE nume_tabelă
    SET nume_coloană1=expr1 [, nume_coloană2=expr2 ...]
    [WHERE condiţie_where]
    [ORDER BY ...]
    [LIMIT nr_înregistrări]
```

Clauza **WHERE** specifică o condiţie pe care trebuie să o îndeplinească înregistrările deja existente în tabelă pentru a fi actualizate. În lipsa acestei clauze se vor actualiza toate înregistrările din tabel.

Presupunem că avem următoarele înregistrări în tabela *t1*:

```
mysql> SELECT * FROM t1;
+----+------+------+
| id | c1   | c2   |
+----+------+------+
|  1 |    1 |    1 |
|  2 |    2 |    2 |
|  3 |    3 |    3 |
+----+------+------+
3 rows in set (0.00 sec)
```

Vom incrementa cu 1 valoarea câmpului *c1* în felul următor:

```
mysql> UPDATE t1 SET c1=c1+1;
Query OK, 3 rows affected (0.00 sec)
Rows matched: 3  Changed: 3  Warnings: 0

mysql> SELECT * FROM t1;
+----+------+------+
| id | c1   | c2   |
+----+------+------+
|  1 |    2 |    1 |
|  2 |    3 |    2 |
|  3 |    4 |    3 |
+----+------+------+
3 rows in set (0.00 sec)
```

De multe ori, vrem să modificăm valorile unei înregistrări specifice, caz în care folosim clauza **WHERE**:

```
mysql> UPDATE t1 SET c1=5, c2=5 WHERE id=3;
```

```
Query OK, 1 row affected (0.36 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM t1;
+----+------+------+
| id | c1   | c2   |
+----+------+------+
|  1 |    2 |    1 |
|  2 |    3 |    2 |
|  3 |    5 |    5 |
+----+------+------+
3 rows in set (0.00 sec)
```

Clauzele **ORDER BY** şi **LIMIT** funcţionează la fel ca şi în cazul comenzii **SELECT**.

7.2.4. Comanda DELETE

Comanda **DELETE** permite ştergerea înregistrărilor dintr-o tabelă, acceptând opţional un criteriu după care se aleg înregistrările care vor fi şterse.

Sintaxa comenzii **DELETE** este următoarea:

```
DELETE FROM nume_tabelă
     [WHERE condiţie_where]
     [ORDER BY ...]
     [LIMIT nr_înregistrări]
```

Clauzele **WHERE, LIMIT** şi **ORDER BY** funcţionează ca şi în cadrul comenzii **UPDATE**.

Pentru a şterge o înregistrare dintr-o tablelă, scriem:

```
mysql> DELETE FROM t1 WHERE id=1;
Query OK, 1 row affected (0.00 sec)
```

ATENŢIE!!! Comanda **DELETE FROM** *nume_tabelă;* va şterge **toate** înregistrările din tabela respectivă.

7.2.5. Exerciţii SQL DML

7.2.5.1. Exerciţiul 1

Folosind comenzile pentru inserarea datelor în tabele, populaţi baza de date creată la capitolul precedent cu următoarele date:

Nume	***Proprietar***	***Judet***	***Locuri***
Batea	Batea Valeria	Cluj	6
Chindris	Chindris Dumitru	Maramures	2
Man	Man Gheorghe	Maramures	6
Magnolia	Korosi Elisabeta	Salaj	8
Scarisoara	Nicodim Pasca	Alba	10
Ana	Ana Flueras	Alba	10

Indicaţii:

- populaţi pentru început tabela judete cu cele 4 judete necesare; opţional, puteţi insera şi alte judeţe;
- continuaţi apoi cu tabela proprietari, unde trebuie să introduceţi cei 6 proprietari din tabelul de mai sus; opţional, puteţi introduce şi alţi proprietari;

- executaţi două interogări pentru a lista conţinutul tabelelor judete şi proprietari şi observaţi care *id*-uri sunt asignate automat judeţelor şi proprietarilor introduşi în baza de date;
- continuaţi cu introducerea pensiunilor, în tabela pensiuni; folosiţi aici *id*-urile aflate la pasul precedent;
- în final, introduceţi datele pentru tabela imagini.

7.2.5.2. Exerciţiul 2

A. Scrieţi câte un query SQL pentru a returna:
   1. ID-ul şi nume pensiunilor care au cel puţin 7 locuri;
   2. Toate informaţiile disponibile despre proprietarii care nu au specificat telefon;
   3. Numele pensiunii şi numele proprietarului, pentru pensiunile care eu 6 locuri;
   4. Numele pensiunilor din judeţul Sălaj, împreună cu locurile lor.
      Atentie! Nu se cunoaşte ID-ul judeţului Sălaj, deci acesta nu se va putea folosi în query.
   5.
      a. ID-ul pensiunii, numele pensiunii şi numarul de imagini asignate, pentru fiecare pensiune.
      b. Modificaţi query-ul de la 5.a. pentru a returna acelaşi rezultat, sortat descrescător, după numarul de imagini asignate pentru o pensiune
      c. Modificaţi query-ul de la 5.a. pentru a returna doar pensiunile care au imagini specificate.
   6. Email-urile proprietarilor pensiunilor din judetul Alba.
      Atentie! Nu se cunoaste ID-ul judetului Alba, deci acesta nu se va putea folosi în query.

B. Scrieţi query-uri pentru inserarea unui judeţ şi al unui proprietar.
   Scrieţi un query pentru a insera o pensiune; obţineţi ID-urile noului judeţ şi proprietar inserat şi folosiţi-le pentru a insera o nouă pensiune.

   Scrieţi un query pentru a insera o imagine; copiaţi o nouă imagine în directorul **images**; folosiţi numele acestei imagini, obţineţi ID-ul pensiunii inserate anterior şi apoi inseraţi o nouă imagine în tabela **imagini** pentru acea pensiune.

C. Inseraţi 3 judeţe de test, în tabela **judete**. Obţineţi-le ID-urile. Scrieţi un query care să şteargă toate cele 3 judeţe, pe baza ID-ului lor.
D. Scrieţi un query care să selecteze primul proprietar din tabela **proprietari** care nu are specificată adresa de email (se va returna ID-ul şi eventual numele acelui proprietar). Pe baza rezultatului query-ului (se va folosi ID-ul proprietarului), formaţi un alt query care să modifice informaţiile despre respectivul proprietar, prin adăugarea unei adrese de email fictive.
E. Scrieţi un singur query care va asigna imaginea din fişierul **006.jpg** pensiunii cu numele *Scarisoara*.
   Indicaţii: folosiţi query-uri imbricate; va trebui să selectaţi ID-ul pensiunii cu numele specificat şi să-l transmiteţi unui query care va face modificarea (UPDATE) imaginii cu numele respectiv.

7.3. Aplicaţia phpMyAdmin

PhpMyAdmin este o aplicaţie web care oferă o interfaţă prietenoasă pentru majoritatea sarcinilor de administrare ale unui server MySQL.

Aplicaţia phpMyAdmin este scrisă în PHP şi este distribuită sub o licenţă Open Source, ceea ce înseamnă că poate fi descărcată, instalată şi folosită în mod gratuit.

PhpMyAdmin are nevoie de un server web, un interpretor PHP şi o bază de date MySQL. Din moment ce aplicaţia în sine este scrisă în PHP, aceasta rulează practic pe orice platformă care pune la dispoziţie dependinţele mai sus amintite.

Pentru convenienţă, distribuitori independenţi oferă pachete care conţin toate aceste aplicaţii la un loc: Apache, PHP, MySQL şi phpMyAdmin. Unul dintre aceste pachete se numeşte XAMPP şi poate fi descărcat la adresa http://www.apachefriends.org/en/xampp.html.

În această carte folosim XAMPP 1.5.3 care conţine Apache 2.2.3, MySQL 5.0.24a, PHP 5.1.6 şi phpMyAdmin 2.8.2.4.

Odată instalat, phpMyAdmin poate fi accesat print intermediul unui browser web, adresa depinzând de configuraţia specifică sistemului.

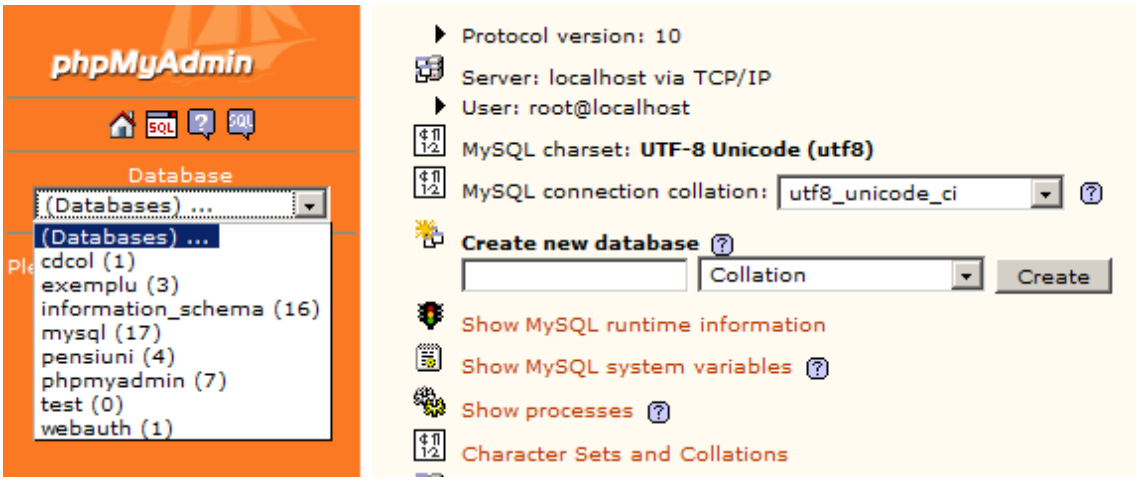
7.3.1. Utilizare XAMPP

- instalare: rulare şi extracţie *xampplite-win32-x.x.x.exe* în directorul preferat, pe harddisc (îşi creează singur subdirectorul *xampplite*) ;
- configurare: rulaţi *setup_xampp.bat*; aşteptaţi apariţia mesajului "*Press any key to continue...*", apoi apăsaţi o tastă;
- pornire: rulaţi *xampp_start.exe*; NU ÎNCHIDEŢI fereastra, eventual minimizaţi-o;
- lucraţi...;
- oprire: rulaţi *xampp_stop.exe*;
- dezinstalare: asiguraţi-vă că este oprit; ştergeţi directorul *xampplite*.

7.3.2. Crearea şi selectarea unei baze de date

Ecranul de întâmpinare al aplicaţiei phpMyAdmin ne pune la dispoziţie două opţiuni:

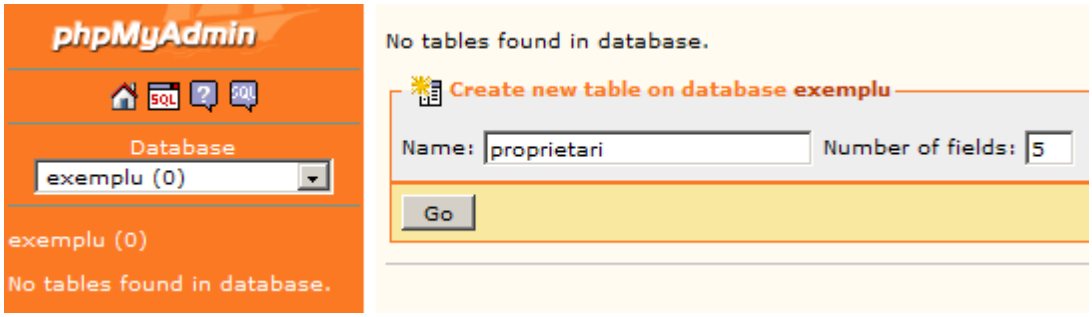
- Crearea unei noi baze de date. Pentru asta scriem numele noii baze de date în căsuţa text de sub „Create new database” şi dăm click pe „Create”.
- Folosirea unei baze de date deja existente prin selectarea ei din căsuţa drop-down din stânga ecranului.



Fie că facem o nouă bază de date sau folosim una existentă, suntem duşi la ecranul de administrare al bazei de date curente. Lista bazelor de date din partea stângă rămâne tot timpul vizibilă, astfel încât putem naviga uşor de la o bază de date la alta.

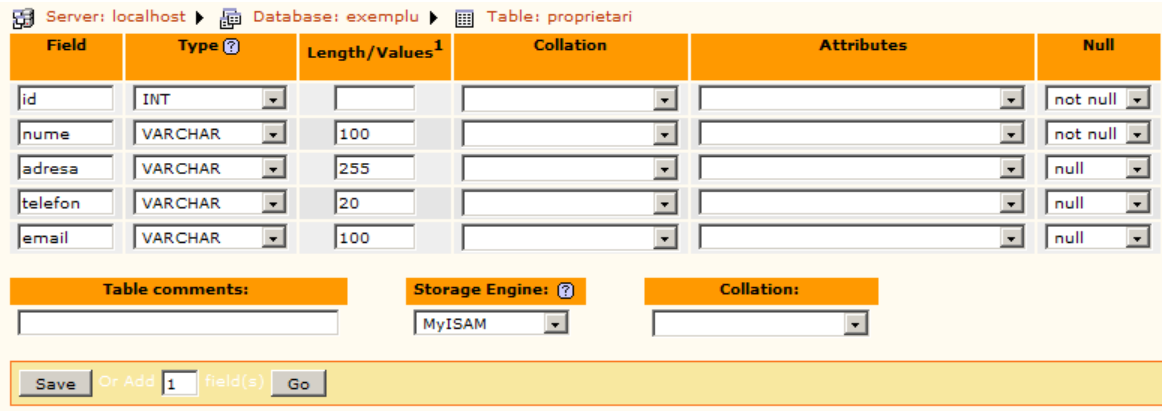
7.3.3. Crearea tabelelor

Crearea unei tabele se face din ecranul principal de administrare al unei baze de date. Întâi ni se cere să specificăm numele şi numărul de coloane al noii tabele. Le introducem şi dam click pe „Go”.



Pasul următor ne lasă să definim structura noii tabele cerându-ne să introducem următoarele informaţii:

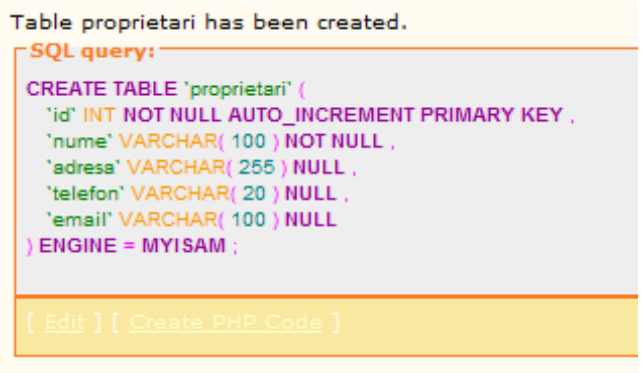
- numele fiecărei coloane;
- tipul de dată al coloanelor (*int, varchar, date* etc.);
- dimensiunea câmpurilor (ex. 100 de caractere pentru *varchar*);
- anumite constrângeri impuse asupra coloanelor – valori unice, nenule, cheie primară etc.;
- opţiunea *auto_increment*.



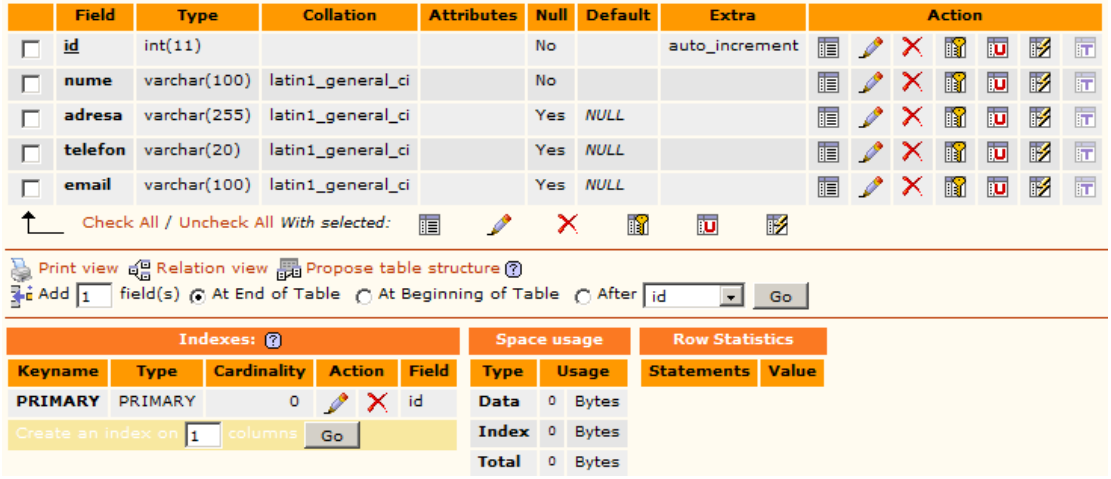
După ce am specificat toate opţiunile pentru toate coloanele, putem să dăm click pe „Save”.

În cazul în care opţiunile introduse sunt nu sunt valide, ni se cere să corectăm erorile pentru a putea continua.

Dacă toate datele introduse sunt corecte, tabela este creată şi ne este confirmată comanda SQL folosită la crearea tabelei.



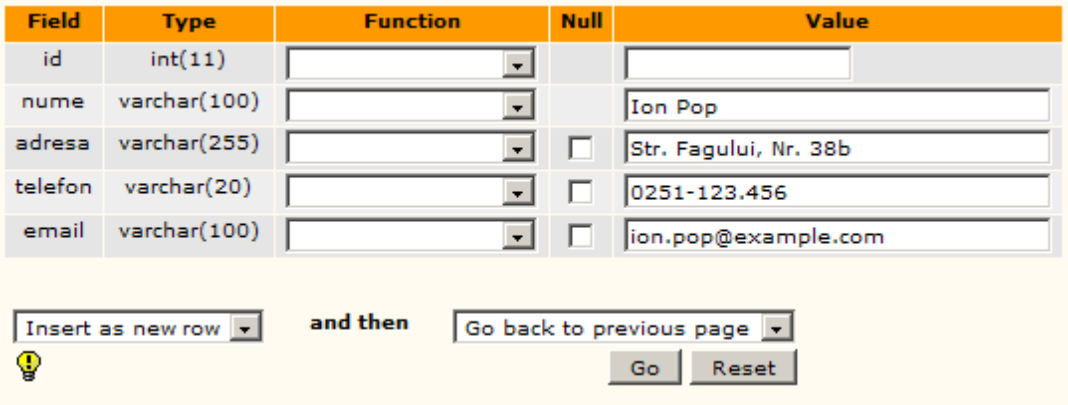
De asemenea putem vedea structura noii tabele create, împreună cu o serie de icoane şi legături permit să efectuăm operaţii asupra tabelei.



7.3.4. Inserarea înregistrărilor

Opţiunea „Insert” apare în meniul din partea superioară a ecranului în cazul în care vizualizăm structura unei tabele (asa cum apare imediat după ce am creat o tabelă nouă), sau sub formă de icoană în dreptul tabelei, dacă vizualizăm lista de tabele din structura bazei de date.

Ne este prezentat un ecran în care putem să introducem valorile fiecărui câmp pentru una sau două înregistrări, cu opţiunea de a reveni la acest ecran după inserare, în cazul în care vrem să introducem mai mult de două înregistrări.



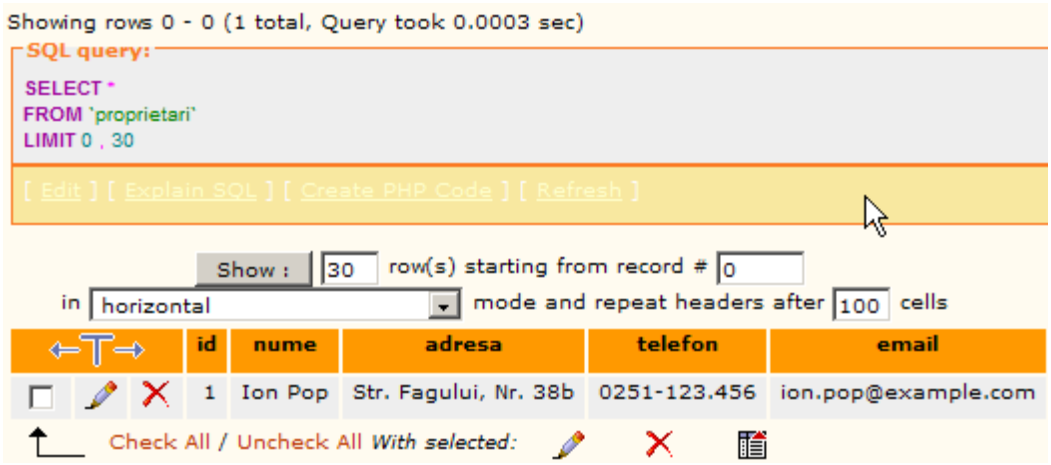
Odată completate câmpurile, dăm cilck pe „Go”. Din nou, inserarea ne este confirmată prin listarea comenzii SQL care a fost folosită pentru operaţia de inserare.

7.3.5. Efectuarea interogărilor

Pentru efectuarea interogărilor phpMyAdmin ne pune la dispoziţie 3 facilităţi:

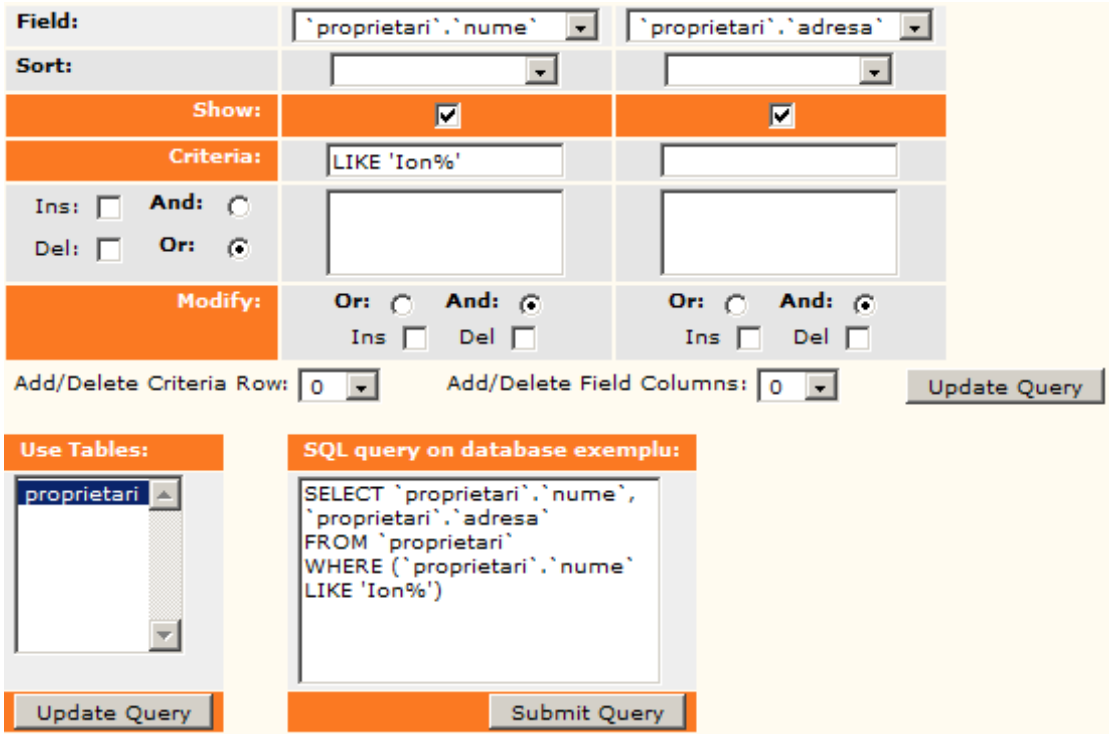
7.3.5.1. Browse

Permite explorarea înregistrărilor fără nici un criteriu de filtrare sau sortare. Acesta este echivalent comenzii `SELECT * FROM` *`nume_tabelă`*. Opţiunea apare în dreptul tabelei dacă vizualizăm baza de date sau în meniul de sus dacă vizualizăm structura tabelei respective.



7.3.5.2. Query

Pune la dispoziţie un formular care ne ajută să construim interogări. O interogare se construieste prin selectarea câmpurilor care vor face parte din interogare, criteriile de filtrare şi de sortare pentru fiecare câmp. Opţiunea este disponibilă doar în meniul principal când vizualizăm baza de date.



7.3.5.3. SQL

Este o metodă generică de interogare. Ne pune la dispoziţie o căsuţă de text în care putem să introduce orice interogare SQL. Opţiunea este disponibilă din meniul principal.