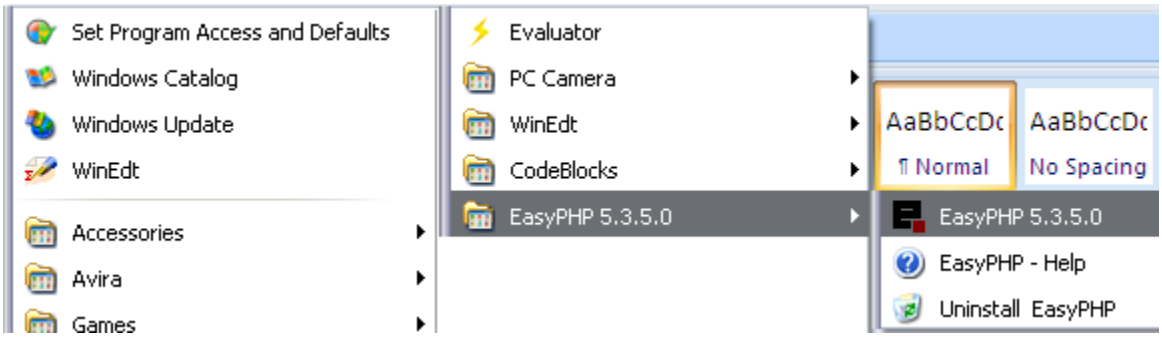
MySQL

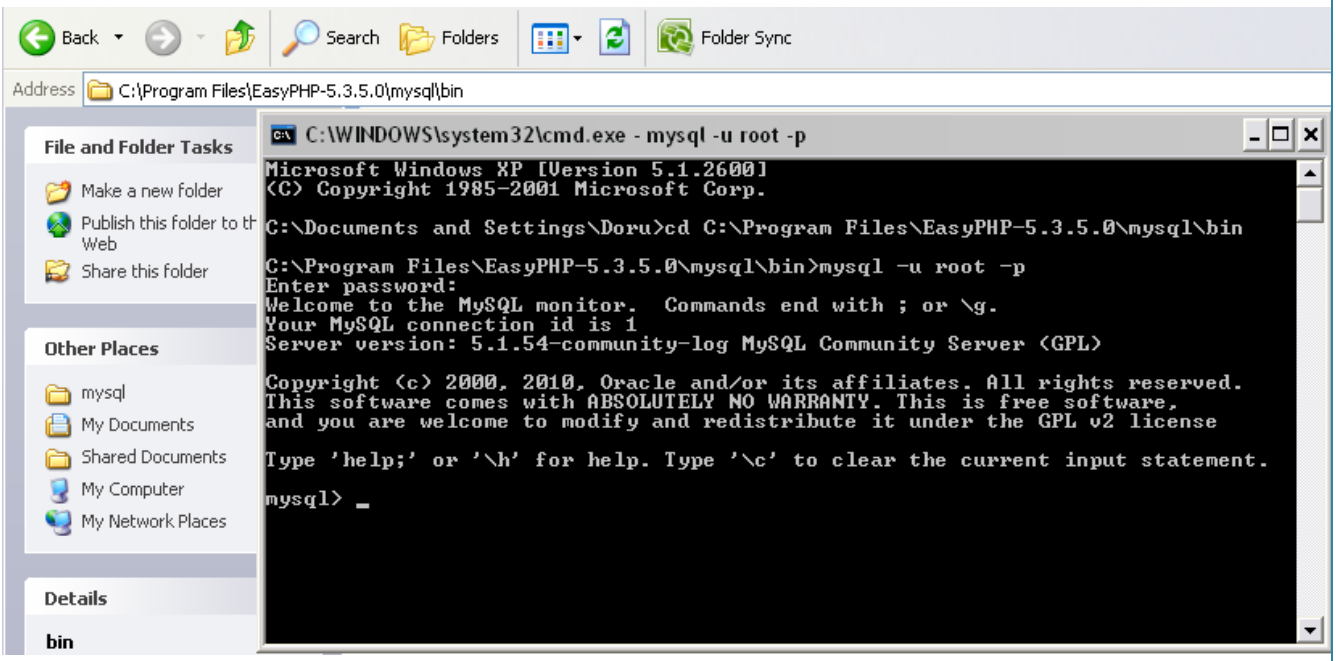
Instalare

EasyPHP-5.3.5.0

Dupa instare pentru a folosi MySql se porneste server-ul MySql:



Apoi se procedeaza ca mai jos:



Crearea unei baze de date

CREATE DATABASE nume_data;

In urma acestei comenzi se va crea baza de date cu numele indicat. In fapt se creaza un folder cu numele bazei de date in subfolder-ul data al folderului MySQL: C:\ Program Files\EasyPHP1-8\mysql\data.

Daca se doreste lucrul cu o anumita baza de date, se va da comanda:

USE nume_baza;

Pentru a sterge o baza de date se foloseste comanda:

DROP DATABASE nume_baza;

Pentru a afisa o lista a bazelor de date existente se utilizeaza comanda:

SHOW DATABASES

Tabele. Notiuni elementare

In tabel coloanele sunt identificate prin nume, iar liniile prin valorile pe care le memoreaza (inregistrari).

O instructiune prin care se poate crea un tabel este prezentata mai jos:

CREATE TABLE nume_tabel(

nume_coloana 1 tip_data [specificatori],

nume_coloana 2 tip_data [specificatori],

…

nume_coloana n tip_data [specificatori],

);

Specificatorii se refera la cheia primara, valori distincte, valori implicite, autoincrementare, daca printre valorile retinute se poate gasi sau nu valoare NULL.

Exemplu 1

Se creaza un tabel cu 3 coloane: prima contine codul materialului, a doua denumirea materialului, iar ultima cantitatea de material.

CREATE TABLE material (

cod char(5), date formate din cel mult 5 caractere

denumire char(20), date formate din cel mult 20 caractere

cantitate int(3) date numerice (intregi) cu maxim 3 cifre

);

Pentru introducerea randurilor (inregistrarilor) intr-un tablel se utilizeaza comanda:

INSERT INTO nume_tabel **VALUES** (data _1, data_2, …, data_n);

Exemplul 2

INSERT INTO material **VALUES** (‘001’, ’panouri’, 78);

Pentru a afisa intreg tabelul se utilizeaza comanda:

SELECT * **FROM** nume_tabel;

Exemplul 3

SELECT * **FROM** material;

Exista posibilitatea sa afisam numai anumite coloane in ordinea pe care o dorim folosind comanda:

SELECT nume_coloana1, …, nume_coloanak **FROM** nume_tabel

Exemplul 4

SELECT cod, denumire **FROM** material;

SELECT cod **AS** codul, denumirea **AS** nume, cantitate **AS** nr_bucati **FROM** material;

Pentru a vizualiza numele tabelelor existente in baza de date folosim comanda:

SHOW TABLES [FROM nume_baza]

In caz ca am uitat care sunt coloanele dintr-un anumit tabel al bazei de date si tipul acestora, putem utiliza comanda:

SHOW COLUMNS FROM nume_tabel

Aplicatie

Creati un tablel in baza de date curenta cu numele preturi si campurile:

cod , pret_unitar, frurnizor

CREATE TABLE preturi (

cod char(5),	date formate din cel mult 5 caractere
pret_unitar int(10),	date numerice cu max 10 cifre
furnizor char(20)	date formate din max 20 caractere

);

Tipuri de date in MySQL

Tipuri de date care retin siruri de caractere

Tip		Descriere
CHAR (n)	-	retine siruri de caractere de lungime maxim n
VARCHAR	-	retine siruri de cel mult 255 de caractere
TINYTEXT	-	la fel ca mai sus
TEXT	-	retine siruri de cel mult 65535 de caractere
MEDIUMTEXT	-	retine siruri de cel mult 16777215 caractere
LONGTEXT	-	retine siruri de cel mult 4294967295 de caractere
ENUM	-	va retine mai multe de siruri de caractere. Un camp de acest tip poate retine un singur sir
SET	-	un camp de acest tip poate retine unul sau mai multe siruri de caractere

Exemplu

Creati tabelul “test” cu doua coloane: prima retine un sir de caractere de cel mult 65.535 de caractere si a doua , un sir din multime sirurilor “luni”,”marti”,”miercuri”. In tabel se va insera 3 inregistrari. Apoi se va afisa tabelul .

```
create table test
( v1  text,
v2 enum ('luni','marti','miercuri')
);
insert into test values ('un sir','miercuri');
insert into test values ('un sir','joi');
insert into test values ('un sir','luni');
select * from test;
```

Tipuri de date numerice

1. Tipuri intregi

2.Tipuri reale

1.Tipurile intregi

Tip	Descriere
TINYINT	Retine numere intregi cuprinse in [-128, 127], iar daca este urmat de UNSIGNED retine numere naturale cuprinse in intervalul [0,255] (adica, TINYTINT UNSIGNED).
SMALLINT	Retine nr intregi cuprinse in intervalul [-32768, 32767], iar daca este urmat de UNSIGNED retine numere naturale cuprinse in intervalul [0,65535]
MEDIUMINT	Retine nunere intregi cuprinse in [-8.388.608,8388,607], iar daca este urmat de UNSIGNED retine numere naturale cuprinse in intervalul [0,16.777.215]
INT	Retine numere intregi cuprinse in [-2.147.483.648, 2.147.483.647], iar daca este urmat de UNSIGNED retine numere naturale cuprinse in intervalul [0,4294967295]
BIGINT	Retine numere intregi din intervalul [-9.223.372.036.854.775.808, 9.223.372.036.854.775.807].

Observatie!

Puteti folosi declaratii de tip in formatul int(4). Aceasta inseamna ca in coloana se rezerva automat o latime de 4 caractere pentru afisarea numerelor. De exemplu, daca numarul retinut este 1, atunci el este afisat ca bbb1, unde prin b am notat spatiul . Aceasta nu afecteaza valorile care pot fi memorate. De exemplu, daca dorim sa memoram numarul 12345, acesta va fi memorat si se va afisa 12345. Dar astfel este afectata latimea intregii coloane.

2.Tipurile reale

Tip	-	Descriere
Float	-	Ocupa 4 octeti.
Double	-	Ocupa 8 octeti.
DECIMAL(n,d)	-	Sir de caractere. Parametrul n reprezinta numarul de cifre aflate inaintea virgulei, plus, daca este cazul, 1 pentru semnul - , iar d rerezinta numarul de zecimale. Daca numarul introdus are inaintea virgulei un numar de cifre +1 (pentru semn) mai mare decat n, acesta este trunchiat, iar daca numarul de zecimale este mai mare ca d, atunci se retin exact d zecimale. In acest din urma caz, numarul se rotunjeste la exact d zecimale.

Exemplu O coloana are tipul decimal(2,3):

Se introduce	Se memoreaza si afiseaza
1.1	1.100
23.4567	23.457
1	1.000
-5	-5.000
-15	-9.999

Tipuri de date care retin anul, data si ora

YEAR – Un camp de acest tip retine ani. O data se introduce ca sir de caractere. De exemplu pentru anul 2013 se poate introduce ‘2013’ sau ‘13’, iar pentru anul 1989 se poate introduce ‘1989’ sau ‘89’.

TIME – Un camp de acest tip retine ora din zi – se introduce ca sir de caractere sub forma ‘hh:mm:ss’. Exemplu: ’12:30:45’

DATE – Un camp de acest tip retine data. Aceasta se introduce sub forma ‘yyyy-mm-dd’. De exemplu ‘1999-11-24’.

DATETIME – Un camp de acest tip retine date de forma data si ora. O data se introduce ca sir de forma:

‘yyyy-mm-dd:hh:mm:ss’.

Operatori utilizati in MySQL

Incepem prin a prezenta principalii operatori in ordinea crescatoare a prioritatii lor.

0. ||, OR, XOR
1. &&, AND
2. BETWEEN, CASE WHEN, THEN, ELSE
3. ==,>=,<=,<,>,!=,<>,IS,IS,LIKE,IN
4. |
5. &
6. <<,>>
7. -,+ (OPERATORI BINARI)
8. *,/, DIV, %, MOD
9. ^
10. -, + (OPERATORI UNARI)
11. !, NOT

Pentru a putea testa un operator puteti utiliza SELECT. De exemplu, daca introduceti comanda

SELECT 1+1;

rezultatul este 2.

Observatie!

In MySQL exista o valoare speciala, numita Null. Semnificatia ei este: valoare necunoscuta. Daca Null este un operand, atunci rezultatul oricarei operatii care se efectueaza cu Null este Null. Evident, daca operandul este Null, atunci el este necunoscut, iar operatia va avea rezultat necunoscut, indifferent de celalalt operand sau de tipul operatiei.

Exemplu

1 + Null=>Null sau Null=Null=>Null.

Operatorii se impart in mai multe grupe.

Operatori aritmetici

Operatorii aritmetici actioneaza asupra tipurilor numerice si returneaza o valoare de tip numeric.

- “+”(adunare) 2+3=>1;
- “-“(scadere) 2-3=>1;
- “*”(inmultire) 3.5*2=>6;
- “/”(impartire) 5/2=>2.5;
- “DIV”(imartire intreaga) 5 DIV 2=>2; -5 DIV 2=>-1;
- “MOD sau %”(restul impartirii intregi) 5 MOD 2=>1; -5 MOD 2=>-1; 5 % 2=>1; -5 % 2 =>-1;
- “-, +” (Minus si plus,oeratori unari) --1=>1;

Operatori de comparare

Pot fi comparate doua valori numerice sau doua siruri. Sirurile se compara lexicographic (ordinea de dictionary) si nu se face distinctive intre literele mari si cele mici.Rezultatul este 1 entru adevarat si 0 entru fals.

- “<” (mai mic) 2<1=>0; 1<2=>0; ‘ab’<b=>1;
- “<=”(mai mic sau egal) 2<=2=>1;
- “>” (mai mare) 3>2=>1; ‘b’>’ab’=>1;
- “>=” (mai mare sau egal) 7>=7=>1; 7 XOR 8=>0;
- “=” (egalitate) 7=7=>1; ‘un’=’UN’=>1;
- “<> sau !=” (diferit) 1<>2=>1; 1!=2=>1; 1<>1=>0; 1!=1=>0;

Comparare lexicografica: ‘Alexandru’ < ’Alexar’ falsa pentru ca ‘n’<’r’

Observatie.

Se pot comara si date de tiul TIME, DATE, etc. De fapt, o astfel de comparare este lexicografica.

Analizati operatorii de comparare de mai jos

‘1:12:3’<2:00:00’=>1

‘1:12:3’<13:00:00’=>0 –comparare s-a facut lexicographic.motiv entru care rezultatul este eronat;’01:12:3’<13:00:00’=>1 entru ca rezultatul sa fie correct,chiar daca compararea este lexicografica,trebuie ca orele,minutele,secundele sa fie cu acelasi numar de cifre.

Operatori logici

In MySQL se considera 2 valori logice:

false, reprezentat de 0 si

true, reprezentat de 1 si de orice valoare diferita de 0.

Operatorii logici actioneaza asupra valorilor logice si returneaza 1, pentru adevarat si 0, pentru fals.

Operatori:

- || sau OR : sau logic, daca ambii operanzi sunt 0, rezultatul este 0, altfel rezultatul este 1.
  2 || 1 => 1;
  (2>3)||(3<5) => 1
- && sau AND, daca ambii operanzi sunt diferiti de 0, rezultatul este 1, altfel rezultatul este 0.
  2 && 1 => 1;
  (2>3)&&(3<5) =>0
- NOT negatie, daca operandul este 0, rezultatul este 1, altfel rezultatul este 0.
  NOT 3 => 0;
- XOR (sau exclusive), daca un operand este 0 si altul diferit de 0, rezultatul este 1, altfel, rezultatul este 0.
  7 XOR 0 => 1
  4 XOR 1 => 1

Alti operatori

1. Operatorii IS NULL, IS NOT NULL testeaza daca o valoare este sau nu NULL
   (sunt singurii operatori prin care se testeaza acest lucru)!
   Exemplu: 1 IS NULL =>0; NULL IS NULL=>1; 1 IS NOT NULL=>1; NULL IS NOT NULL=> 0;

2. Operatorii IN, NOT IN testeaza apartenenta unei valori la o multime. Valoarea poate fi de forma numerica sau de un tip care retine un sir de caractere.

Exemplu: 1 IN (1,2,3,4) => 1; 5 IN(1,2,3,4)=>0; '1 sir' IN ('1sir','2sir','3sir')=>1; '4sir' IN ('1sir','2sir','3sir')=>0;

3. Operatorul BETWEEN min AND max testeaza daca o valoare se gaseste intre o valoare minima si una maxima. In caz afirmativ returneaza 1, altfel, 0. Se poate aplica si pentru date care retin siruri de caractere (ordinea lexicografica), dar si pentru datele de timp.

Exemplu: 1 BETWEEN 0 AND 4 => 1; 'mama' BETWEEN 'min' AND 'lin'=> 0; 2 BETWEEN 2 AND 2 =>1;

1 BETWEEN 2 AND 4=>0; '1999-10-31' BETWEEN '1989-12-10' AND '2000-01-01' =>1.

4. Operatorul CASE WHEN THEN ELSE. Poate fi aplicat in doua forme:

Forma 1. Se evalueaza v si daca produce valoarea v_i se returneaza valoarea val_i, iar daca nicio canditie v_i nu este indeplinita, se returneaza valoarea val_{n+1}:

CASE v

WHEN v_1 THEN val_1

WHEN v_2 THEN val_2

…

WHEN v_n THEN val_n

[ELSE val_{n+1}]

END

Exemplu. Se va afisa ‘doi’ dupa operatia:

CASE 2

WHEN 1 THEN ‘unu’

WHEN 2 THEN ‘doi’

WHEN 3 THEN ‘trei’

ELSE ‘alta valoare’

END;

Forma 2. In acesta forma, daca este indeplinita conditia i; se returneaza valoarea v_i. Daca niciuna din conditiile v_i nu este indeplinita, se returneaza valoarea v_{n+i}:

CASE

WHEN $conditie_1$ then V_1

[WHEN $conditie_2$ THEN v_2]

…

[WHEN $conditio_n$ THEN v_n]

ELSE v_{n+1}

END

Exemplu. Se afiseaza ‘adevarat’:

CASE

WHEN 1<2 THEN ‘adevarat’

ELSE ”nu se poate” END;

Functii MySQL

In MySQL exista o serie de functii predefinite. Pot fi testate prin utilizarea instructiunii SELECT.

Cateva functii matematice:

1. ABS(x) → Modul de x
   Exemplu: ABS(-2) →2;

2. CEIL(x) → Cel mai mic intreg mai mare sau egal cu x
   Exemplu: CEIL(1.7) →2;
   CEIL(2) →2;
   CEIL(-1.7) →-1;

3. FLOOR(x) →Cel mai mare intreg mai mic sau egal cu x
   Exemplu: FLOOR(-1.7) →-2;
   FLOOR(2) →2;
   FLOOR(1.7) →1;

4. EXP(x)→ e^x
   Exemplu:EXP(1)→ 2.7182…;

5. LOG(b,x)→ Logaritm in baza b din x
   Exemplu :LOG(2,4) →2;

6. PI()→π
   Exemplu:PI() →3.141593;

7. POW(x,y) →x^y
   Exemplu:POW(2,3) →8;

8. ROUND(x) →Cel mai apropiat intreg de x.
   Exemplu: ROUND(2.3) →2;
   ROUND(2.7) →3;
   ROUND(-2.8) →-3;

9. SQRT(x) →Radical din x
   Exemplu:SQRT(4) →2;

Cateva functii care prelucreaza siruri de caractere

Functie	Descriere	Rezultat
LENGTH(X)	Lungimea sirului x.	LENGTH('abc') =>3;

CONCAT(X1, X2, …)	Concateneaza sirurile x1,x2,…	CONCAT(‘ab’,’c’)=>’abc’
INSTR(x,y)	Cauta daca y este subsir pentru x. In caz afirmativ, Returneaza indicele de inceput(indicii incep cu 1), Astfel returneaza 0.	INSTR(‘abc’,’b’)=>2 INSTR(‘abc’,’d’)=>0
SUBSTRING(x,p,l)	Returneaza subsirul din x care incepe in pozitia p si are lungime l.	SUBSTRING(‘abcde’,2,3) => ’bcd’
RTRIM(x)	Returneaza sirul fara blank-urile din dreapta.	RTRIM(‘un ‘)=>’un’
LTRIM(x)	Returneaza sirul fara blank-urile din stanga.	LTRIM(‘ un‘)=>’un’
TRIM(x)	Returneaza sirul fara blank-urile din stanga si din dreapta.	TRIM(‘ un ‘)=>’un’
UPPER(x)	Returneaza sirul scris cu litere mari.	UPPER(‘Un’)=>’UN’
LOWER(x)	Returneaza sirul scris cu litere mici.	LOWER(‘UN’)=>’un’
FIND_IN _SET(x, ‘s1,s2,…,sn’)	Returneaza indicele aparitiei sirului x in sirul de siruri ‘s1,s2,…,sn’. Daca x nu este gasit,returneaza 0.	FIND_IN_SET (‘b’,’a,b,c’)=>2
FORMAT(x,d)	Converteste numarul real x la un sir cu d zecimale. Pentru ultima zecimala se face, daca este cazul, rotunjirea.	FORMAT(1.789,2)=>1.79
STRCMP(x,y)	Compara lexicografic sirurile x si y. Daca x<y returnaza -1, daca x=y returnaza 0, iar daca x>y, returnaza 1.	STRCMP(‘ma’,’mama’) =>-1

Cateva functii care prelucreaza data si ora

NOW() - Returneaza data si ora curenta sub forma yyyy-mm-dd hh:mm:ss.

YEAR(x) – Daca x este de tip DATA, afiseaza anul extras din x. YEAR(NOW()) returneaza anul curent

DAY(x) – extrage ziua din data x.

MONTH(x) – extrage luna din data x.

TIME(x) – extrage ora din x.

HOUR(x) – extrage ora ca numar intre 0 si 23 din x.

MINUTE(x) – extrage minutul din x.

DATEDIFF(x,y) – x, y sunt de tipul DATETIME sau DATE si calculeaza diferenta de zile dintre x si y. (DATEDIFF(NOW(), 1987-11-20).

DATE_ADD(DATA, INTERVAL, nr, x) - Calculeaza data care se obtine daca la valoarea data se adauga nr, unde x specifica ce este nr (year, month,day). (DATE_ADD(NOW(), INTERVAL, 10, DAY) se obtine data care va fi peste 10 zile.

DATE_SUB(DATA, INTERVAL, nr, x) – La fel ca mai sus numai ca se obtine data diferentiata. (DATE_ADD(NOW(), INTERVAL, 10 DAY) se obtine data care a fost acum 10 zile.

Doua functii speciale

Functia IF are forma de mai jos, unde $expresie_1$ este de tip intreg. Daca $expresie_1$ este nenula, atunci se returneaza $expresie_2$, altfel se returneaza $expresie_3$.

Atentie!

Daca $expresie_1$ este de tip real, se rotunjeste la cel mai apropiat intreg. Daca, de exemplu, acesta este de 0.41, atunci se rotunjeste la 0, prin urmare, rezultatul va fi altul

IF($expresie_1$, $expresie_2$, $expresie_3$)

Exemplu.

Analizati exemplele urmatoare: IF(1<2, ‘adevarat’, ‘fals’) => ‘adevarat’, IF(0.45, ‘nu’, ‘da’) => ‘da’;

Functia IFNULL are forma de mai jos. Daca $expresie_1$ nu este NULL, returneaza $expresie_1$, altfel returneaza $expresie_2$.

IFNULL($expresie_1$, $expresie_2$)

Exemplu.

Analizati exemplele urmatoare: IFNULL(0, ‘este null’) => 0, IFNULL(NULL,’este null’) => ‘este null’.

Afisarea coloanelor care rezulta in urma unui calcul

O regula elementara in crearea si utilizarea bazelor de date spune ca acestea vor retine numai date care nu rezulta in urma unui calcul. Nerespectarea ei conduce la baze de date care ocupa mult spatiu. Acum vom prezenta modul in care se afiseaza datele care rezulta in urma unui calcul.

Tabelul de mai jos, numit “prod” retine produsele existente intr-un deposit. Pentru fiecare produs se cunoaste cantitatea (“cant”) si pretul unitar (“pret_unitar”), pretul unui exemplar din produsul respectiv. Sa presupunem ca dorim ca, pe langa informatiile afisate, sa aflam valoarea totala pentru fiecare produs in parte. Valoarea se obtina ca produs intre cantitate si pretul unitar.

den	cant	Pret_unitar
Produsul 1	20	2.50
Produsul 2	10	15.00
Produsul 3	12	3.00

Tabelul “prod”

Pentru aceasta, atunci cand afisam tabelul, vom crea o noua coloana sub forma: *expresie* as *nume_coloana*, unde *expresie* calculeaza valorile din coloana respective, iar “nume_coloana” va fi numele ei. Iata cum se afiseaza noul tabel, prezentat mai jos:

den	cant	pret_unitar	valoare
Produsul 1	20	2.50	50.00
Produsul 2	10	15.00	150.00
Produsul 3	12	3.00	36.00

Tabelul rezultat

L)

SELECT den, cant, pret_unitar, cant*pret_unitar AS valoare FROM prod;

Exemplu

Se da tabelul de mai jos, numit : ”persoane”. Se cere sa se separe numele de prenume si sa se afiseze ca mai jos:

nume
Andrei Popescu
Carmen Paunica

Tabelul “persoane”

prenume	nume
Andrie	Popescu
Carmen	Paunica

Tabelul dorit

Rezolvare.

Avem nevoie de functiile substring, instr si length de la siruri de caractere, pentru extragerea unei sectente de caractere dintr-un sir de caractere, pentru cautarea unui caracter intr-un sir de caractere si pentru determinarea numarului de caractere dintr-un sir de caractere.

Se utilizeaza functii care opereaza asupra sirurilor de caractere, dupa cum urmeaza:

SELECT SUBSTRING(nume,1,INSTR(nume,' ')-1) AS prenume, SUBSTRING(nume, 1+INSTR(nume,' '), LENGTH(nume)) AS nume FROM persoane;

Valoarea NULL

În MySQL exista o valoare speciala, numita NULL. Semnificatia ei este "nici o valoare". Atunci cand definim o coloana se poate trece specificatorul NULL, implicit NULL.

a) Atunci cand o coloana are trecut specificator NULL sau nu s-a trecut nimic si la introducerea unui rand, pentru coloana respectiva nu este trecuta o valoare pentru acel camp, acolo se va memora valoarea speciala NULL.

b) Atunci cand o coloana are trecut specificatorul NOT NULL si la introducerea unui rand, in coloana respectiva, nu este trecuta o valoare, acolo se va memora:

- 0, daca coloana este de tip numeric;
- Sirul vid, daca coloana este de un tip care permite retinerea sirului;
- Indicele 0 din sir,daca coloana este de un tip ENUM si acesta are efect afisarea primului sir al enumerarii;
- Multimea vida, daca coloana este de tip SET.

Exemplu:

```
CREATE TABLE testnull1

( c1 TEXT NULL,

c2 TEXT NOT NULL ) ;

INSERT INTO TESTNULL1 (c1) VALUES ('un sir');

INSERT INTO TESTNUL1 (c2) VALUES ('alt sir');

SELECT * FROM TESTNULL1;
```

Valori implicite

Atunci cand pentru un anumit rand nu se cunoaste un camp, fie acesta va retine NULL, fie va retine o valoare implicita, convenabil aleasa. Pentru ca un camp, in absenta datelor sa aiba o valoare implicita, se foloseste specificatorul DEFAULT:

DEFAULT valoare_implicita

Pot lua valoare implicita campurile cu o lungime fixa.

L)
CREATE TABLE pers_oras
(nume CHAR(25),
Domiciliu CHAR(30) DEFAULT ‘necunoscut’);
INSERT into pers_oras (nume) VALUES (‘Gulagea Constantin’);
INSERT into pers_oras VALUES (‘Ionescu Mihai’, ‘Galati’);
SELECT * FROM pers_oras;

Cheie primara si cheie unica

A) CHEIE PRIMARA este constituita dintr-un camp si trebuie sa indeplineasca simultan conditiile:

- Valorile retinute trebuie sa fie distincte;
- Campurile trebuie sa aiba o lungime fixa.

CREATE TABLE ex_cheie_primara_un_camp

(c1 INT PRIMARY KEY,

C2 TEXT
);

B)CHEIE UNICA

Create table DISTINCTE
(c1 INT UNIQUE KEY,
C2 char(10) UNIQUE KEY
);

Autoincrementare

Este incomod, ca la introducerea unei inregistrari sa se specifice numarul de ordine pentru aceasta. Din acest motiv, s-a pus la dispozitia utilizatorilor un mecanism prin care incrementarea se face automat.

Pentru ca o coloana sa retina numerele de ordine ale inregistrarilor respective, este necesar ca la definirea campului, sa se utilizeze specificatorul AUTO_INCREMENT. Pentru a putea utiliza acest specificator, este necesar sa fie indeplinite conditiile:

a) coloana (campul) sa fie de un tip intreg;

b) coloana (campul) sa retina numai valori distincte – pentru a realiza acest lucru, coloana (campul) trebuie sa fie declarata ca fiind cheie primara sau cu cheie unica.

Exemplu: Crearea unui tabel “auto”:

```
CREATE TABLE auto(
nr_ord INT UNIQUE KEY AUTO_INCREMENT,
nume  CHAR(20)
);
INSERT INTO auto (nume) VALUES (‘Marius Oprean’);
INSERT INTO auto (nume) VALUES (‘Mihai Minca’);
```

Sortarea datelor

Uneori este necesar ca datele sa fie afisate sortate dupa valorile unei coloane sau dupa valorile mai multor coloane. Daca acestea sunt de tip sir de caractere, ele pot fi sortate in ordine lexicografica.

Exemplu: Avem un table, numit “sortare” care contine campurile “nume” si “data_n”.

Pentru a ordona crescator coloana (campul) nume vom folosi comanda:

```
SELECT * FROM tabel_sortare ORDER BY camp_de_ordonare;
```

Implicit ordonarea este crescatoare. Daca se doreste ordonare descrescatoare se adauga la comanda clauza DESC.

Exemplu:

```
SELECT * FROM tabel_sortare ORDER BY varsta DESC;
```

Un tabel se poate sorta dupa mai multe campuri.

Exemplu:

```
SELECT * FROM sortare ORDER BY nume, varsta DESC;
```

Astfel tabelul va fi ordonat crescator dupa nume si la nume egale descrescator dupa varsta.

Filtrarea datelor

Uneori nu suntem interesati sa afisam toate inregistrarile (liniile) unui tabel, ci numai cele care indeplinesc anumite conditii. Aceasta operatie se numeste *filtrare*. Pentru a se realiza operatia de filtrare se adauga instructiunii SELECT, la sfarsit, clauza WHERE *conditie*. In acest fel, sunt selectate numai liniile pentru care conditia din clauza WHERE este indeplinita. In cazul in care datele se cer sortate, clauza ORDER BY se va trece dupa WHERE.

Exemplu: CREATE TABLE studenti (nume TEXT, matematica INT, engleza INT, informatica INT);

1. Afisati numele studentilor, care au la matematica note mai mai sau egale cu 8.

```
SELECT nume FROM studenti WHERE matematica >=8;
```

2.La fel ca mai sus,numai ca se cere ca numele studentilor sa fie afisate in ordine alfabetica.

```
SELECT nume FROM studenti WHERE matematica >=8 ORDER BY name;
```

Actualizari intr-un tabel

Prin actualizarea datelor dintr-un tabel intelegem operatii precum inserarea unor linii, modificarea valorilor unor campuri, stergerea unor randuri.

Pentru inserare putem utiliza instructiunea de mai jos:

INSERT INTO nume_tabel (nume_col1, nume_col2, …) VALUES (expresie1, expresie2, ...)

Intr-un tabel se pot insera linii dintr-un alt tabel. Pentru aceasta se foloseste instructiunea urmatoare:

INSERT INTO [DISTINCT] nume_tabel (nume_col1, nume_col2, …) SELECT col1, col2, ... FROM nume_tabel_sursa WHERE *conditie*

Daca se doreste inserarea numai a liniilor distincte din tabelul sursa, utilizati specificatorul DISTINCT.

Astfel se copiaza in campurile nume_col1, nume_col2, … din tabelul nume_tabel toate inregistrarile (campurile col1, col2, …) din tabelul nume_tabel_sursa ce indeplinesc *conditie*.

Pentru a modifica valorile retinute intr-un camp sau mai multe, se utilizeaza instructiunea de mai jos:

UPDATE nume_tabel SET coloana1=expresie1, …, coloanak=expresiek [WHERE conditie];

In urma executarii acestei comenzi, pentru fiecare inregistrare (rand) din tabel care indeplineste conditia din WHERE, se actualizeaza coloanele indicate de SET cu expresiile cotespunzatoare.

Observatie. In absenta clauzei WHERE sunt afectate toate randurile din tabel.

Stergerea unui rand sau a mai multor randuri dintr-un tabel se realizeaza cu comanda:

DELETE FROM nume_tabel [WHERE conditie]

Observatie. In absenta clauzei WHERE sunt sterse toate randurile din tabel.

Modificarea numelui unui tabel acestuia se realizeza prin comanda:

RENAME TABLE nume_vechi TO nume_nou

Pentru stergerea unei coloane se foloseste:

ALTER TABLE nume_ tabel DROP COLUMN nume_coloana

Pentru adaugarea unei coloane cu un anumit tip de date se foloseste:

ALTER TABLE nume_tabel ADD nume_col tip;

Exemplu: Se cere sa se insereze in tabelul ”prs” acele persoane din prs1 care sunt din Slatina. Tabelele prs si prs1 au ambele campuri nume si oras de acelasi tip.

INSERT INTO prs (nume, oras) SELECT nume, oras FROM prs1 WHERE oras=’Slatina’;

Functii agregate

Toate calculele realizate pana in present au avut ca operanzi doar campurile unui aceluiasi rand. Intrebarea este: se pot efectua calcule cu valorile retinute de o coloana? Raspunsul este afirmativ. Pentru astfel de calcule se utilizeaza asa-numitele “functii-agregate”. In cele ce urmeaza, prezentam cateva dintre aceste functii, iar pentru exemplificare utilizam tabelul ‘studenti‘, tabel creat astfel:

CREATE TABLE studenti (nume TEXT, matematica INT, engleza INT, informatica INT);

Functii agregate:

COUNT() – Sub forma COUNT(*), afiseaza numarul de linii ale tabelului. Sub forma COUNT(nume_coloana), numara valorile, din coloana specificata, care nu sunt egale cu NULL.

MIN() – Sub forma MIN(nume_coloana), calculeaza cea mai mica valoare din coloana specificata. Valorile NULL sunt ignorate.

MAX() – Sub forma MAX(nume_coloana), calculeaza cea mai mare valoare din coloana specificata. Valorile NULL sunt ignorate.

SUM() – Sub forma SUM(nume_coloana), calculeaza suma valorilor din coloana specificata. Valorile NULL sunt ignorate.

AVG() – Sub forma AVG(nume_coloana), calculeaza media aritmetica a valorilor din coloana specificata. Valorile NULL sunt ignorate.

Exemplul 1. Cati studenti sunt in tabel?

```
SELECT COUNT(*) AS numar_studenti FROM studenti;
```

Exemplul 2. Care este numarul de note la engleza?

```
SELECT COUNT(engleza) AS numar_note_engleza FROM studenti;
```

Exemplul 3. Care este media notelor la informatica?

```
SELECT AVG(informatica) AS media_note_informatica FROM studenti;
```

Exemplul 4. Cati studenti au nota 10 la engleza?

```
SELECT COUNT(engleza) AS studenti_de_zece From studenti WHERE engleza=10;
```

Exemplul 5. Cati studenti au media generala peste 8?

```
SELECT COUNT(nume) AS studenti_peste_8 WHERE (matematica+engleza+informatica)/3 >8;
```

Utilizarea subinterogarilor

La modul general, o interogare poate returna:

A) **O coloana, eventual un singur rand;**
B) **Mai multe coloane**

A) Interogarea subordonata returneaza o coloana cu un singur rand (o singura valoare)

Exemplul 1

Se cere sa se determine numele studentilor care au cea mai mare nota la matematica.

```
SELECT nume, matematica FROM  studenti WHERE matematica=(SELECT MAX(matematica) FROM studenti);
```

Mai intai, se executa interogarea subordonata, apoi cea care o subordoneaza. In acest caz, interogarea subordonata calculeaza nota maxima obtinuta la matematica, iar cealalta interogare returneaza numele studentilor cu aceasta nota.

B) Interogarea subordonata returneaza un tabel

In acest caz, tabelul returnat de interogarea subordonata trebuie sa aiba un nume. Aceasta se asociaza cu ajutorul clauzei AS.

Exemplu Care este media generala la matematica si la informatica a studentilor care au cel putin 8 la una dintre discipline?

SELECT AVG(virtual.matematica) AS medie_mate, AVG(virtual.informatica) AS medie_info FROM

(SELECT matematica, informatica FROM studenti WHERE matematica>=8 or informatica>=8) AS virtual;

Gruparea datelor

Datele dintr-un tabel pot fi grupate in functie de valorile dintr-o anumita coloana.

Exemplul 1

Consideram tabelul ‘vanzari’ cu structura din exemplu, pe care il vom crea astfel:

CREATE TABLE vanzari(data DATE, produs CHAR(20), valoare INT);

data	produs	valoare
2012-03-15	Produs 1	12
2012-03-15	Produs 2	18
2012-03-15	Produs 1	28
2012-03-16	Produs 3	100
2012-03-16	Produs 1	15
2012-03-17	Produs 2	155
2012-03-17	Produs 3	11
NULL	Produs 2	111
NULL	Produs 1	12

De exemplu, datele pot fi grupate dupa valorile din coloana data. Astfel un grup este alcatuit din randurile de tabel corespunzatoare datei 2012-03-15, altul datei 2012-03-16, si asa mai departe.

Prelucrarile datelor din fiecare grup se pot realiza cu ajutorul functiilor agregate. Pentru a grupa datele din tabel dupa valorile unei coloane se utilizeaza clauza GROUP BY nume_coloana, adaugata dupa numele tabelului. Astfel pentru exemplul anterioar putem raspunde la urmatoarele intrebari:

1. Care este numarul de vanzari in fiecare zi?
   Grupam datele dupa campul data si numaram vanzarile pentru fiecare grup.

SELECT data, COUNT(*) AS numar_vanzari FROM vanzari GROUP BY data;

2. Care este suma vanzarilor zilnice?
   Grupam datele dupa data si insumam vanzarile pentru fiecare data in parte:
   SELECT data, SUM(valoare) AS suma_vanzari FROM vanzari GROUP BY data;

3. Care este numarul de vanzari si suma vanzarilor in fiecare zi?
   SELECT data, COUNT(*) AS nr_vanzari, SUM(valoare) AS suma_incasata FROM vanzari GROUP BY data;

Uniuni de tabele

O uniune este alcatuita din doua sau mai multe tabele intre care exista o legatura. De cele mai multe ori, legatura este data de valorile existente in cate o coloana a fiecarui tabel din uniune. Pentru tabelele ce alcatuiesc uniunea se pot utiliza alias-uri, sub forma nume_tabel AS nume_alias, introduse in clauza FROM. Alias-urile pot fi utilizate in orice parte a instructiunii SELECT. In aceste cazuri, adresarea unei coloane a tabelului se face sub forma: **nume_alias.nume_coloana**.

Avantajele folosirii alias-urilor sunt:

- Se pot utilize nume scurte: A, B, …
- Un sigur tabel poate avea mai multe alias-uri.

1) O prima forma de realizare a unei uniuni este utilizarea instructiunii SELECT in care la clauza FROM se trec, pe rand, toate tabelele care o alcatuiesc, iar legatura este precizata cu ajutorul clauzei WHERE.

Exemplu

Consideram tabelele cu numele **produse** si **date_produse** cu urmatoarele inregistrari:

produse

cod	cantitate
001	20
003	10
002	2
004	1
006	8

date_produse

cod	denumire	producator

001	Produs 1	Producator 2
003	Produs 3	Producator 5
002	Produs 2	Producator 4
005	Produs 4	Producator 2
006	Produs 6	Producator 1
008	Produs 7	Producator 2

Pentru fiecare produs existent in magazine, care este trecut in tabelul date_produse se cere sa se afiseze denumirea si cantitatea.

SELECT B.denumire, A.cantitate FROM produse AS A, date_produse AS B WHERE A.cod=B.cod

Dupa scrierea acestei comenzi se obtine tabelul:

denumire	Cantitate
Produs 1	20
Produs 3	10
Produs 2	2
Produs 6	8

2) Atunci cand uniunea este alcatuita din doua tabele se poate preciza o legatura de tip INNER sub forma urmatoare:

   Nume_tabel1 AS … INNER JOIN nume_tabel2 AS … ON conditia_de_uniune

   In astfel de cazuri se afiseaza datele din tabel1 si tabel2, pentru care este indeplinita conditia din ON. O astfel de uniune se numeste si uniune interioara.

   Rezolvarea cerintei din exemplul anterior poate fi realizata si astfel:
   SELECT B.denumire, A.cantitate FROM produse AS A INNER JOIN date_produse AS B ON A.cod=B.cod;

3) Atunci cand uniunea este alcatuita din doua tabele, se poate preciza o legatura de tip LEFT OUTER sub forma:

   Nume_tabel1 AS … LEFT OUTER JOIN nume_tabel2 AS … ON conditia_de_uniune

   In acest caz se afiseaza toate datele din tabel1, iar datele din tabel2 se afiseaza numai daca este indeplinita conditia din ON, altfel, in locul lor se afiseaza valoarea implicita (NULL de cele mai multe ori). O astfel de uniune se numeste si uniune exterioara stanga.

   Daca folosim tabelele din exemplul anterior, comanda:

SELECT B.denumire, A.cantitate FROM produse AS A LEFT OUTER JOIN date_produse AS B ON A.cod=B.cod;

Se va afisa:

denumire	Cantitate
Produs 1	20
Produs 3	10
Produs 2	2
NULL	1
Produs 6	8

4) Atunci cand uniunea este alcatuita din doua tabele, se poate preciza o legatura de tip RIGHT OUTER sub forma:

Nume_tabel1 AS … RIGHT OUTER JOIN nume_tabel2 AS … ON conditia_de_uniune

In acest caz se afiseaza toate datele din tabel2, iar datele din tabel1 se afiseaza numai daca este indeplinita conditia din ON, altfel, in locul lor se afiseaza valoarea implicita (NULL de cele mai multe ori). O astfel de uniune se numeste si uniune exterioara stanga.

Daca folosim tabelele din exemplul anterior comanda:

SELECT B.denumire, A.cantitate FROM produse AS A RIGHT OUTER JOIN date_produse AS B ON A.cod=B.cod;

Se va afisa:

denumire	Cantitate
Produs 1	20
Produs 3	10
Produs 2	2
Produs 4	NULL
Produs 6	8
Produs 7	NULL