Web2, curs05

I. MySql

1. SQL – elemente de baza
    - Bazele de date sunt reprezentate asemenea directoarelor pe disc iar tabelele asemenea fisierelor din interiorul directoarelor, astfel numele bazelor de date respecta restrictiile SO impuse asupra numelor de directoare iar numele tabelelor respecta restrictiile SO impuse asupra numelor de fisiere
        - Numele bazelor de date nu pot depasi 64 de caractere
        - Se recomanda sa contina numai litere , cifre (caractere alfanumerice) si "_".
        - Pot contine si caractere speciale (de exemplu spatiu)
        - Numele care contin caractere speciale sau contin numai cifre sau sunt cuvinte cheie, trebuie incluse intre ` ` (back quotes).
        - De preferat este insa sa nu folositi spatii (sau alte caractere speciale)
        - Numele tabelelor sau coloanelor se stabilesc la fel ca cele ale bazelor de date
        - Pentru a evita problemele la trecerea de la un SO la altul, denumiti bazele de date si tabelele folosind numai litere mici; folositi _ in loc de spatiu
        - La denumirea coloanelor e bine sa respectati un standard usor de tinut minte, cum ar fi **camel-case** (incepe cu litera mica, si daca e compusa din mai multe cuvinte, fiecare incepe cu litera mare; De exemplu: idProdus sau numePersoana) sau stilul **underline**..
    - Fiecare instructiune sql se termina cu punct si virgula (;)
    - Comentarii
        - # sau - - pentru comentariu pe un rand
        - /* */ pentru comentariu pe mai multe randuri
    - Valorile ce au ca tip de date un **sir de caractere sau data calendaristica**, trebuie incluse intre ghilimele simple **' '** sau ghilimele duble **" "**.
    - Desi nu este obligatoriu, se foloseste standardul conform caruia cuvintele cheie SQL se scriu uppercase (cu litere mari) pentru a le diferentia mai usor de numele de tabele, coloane, valori, etc.
    - Valoarea NULL indica lipsa unei valori in campul respectiv. Ea este diferita fata de 0 (zero) pentru numere, sau sirul gol ('') pentru siruri de caractere. Se poate specifica pentru anumite coloane faptul ca nu accepta aceasta valoare prin atributul NOT NULL.

2. SQL – tipuri de date
    - Numere intregi
        - **TINYINT -** numere intregi pe 1 octet (tiny = minuscul)
        - **SMALLINT -** numere intregi pe 2 octeti (SMALL = mic)
        - **MEDIUMINT -** numere intregi pe 3 octeti
        - **INT –** numere intregi pe 4 octeti
        - **BIGINT -** numere intregi pe 8 octeti
        - Domeniul de valori pentru reprezentare pe x octeti:
            - cu semn
                - [-a, a-1]
            - fara semn
                - [0,2a-1]
            - a= (2^(8x))/2=2^(8x-1); ^ = putere; x = numsrul de octeti
            - Ex: pentru SMALLINT x=2
                - domeniul de valori pentru numere de tip SMALLINT
                    - a=(2^16)/2 = 65536/2= 32768
                    - pentru numere cu semn: [-32768, 32767]
                    - pentru numere fara semn: [0,65535]
    - Numere reale
        - **DECIMAL –** numere reale cu precizie exacta (se recomanda pt valori monetare)
            - Ex: salariu DECIMAL(5,2) – poate memora valori intre -999.99 si 999.99
            - 999.009 nu poate fi memorat

        - **FLOAT –** numere reale cu simpla precizie (reprezentare pe 4 octeti)
            - Ex: salariu FLOAT(5,2)
            - 999.009 va fi memorat ca 999.01
        - **DOUBLE–** numere reale cu dubla precizie (reprezentare pe 8 octeti)
        - **REAL**
- Date calendaristice
    - **DATE -** este tipul de date folosit pentru memorarea datei calendaristice. In baza de date formatul datei este **YYYY-MM-DD** (Y - year (an) , M - month (luna), d - day (zi)). Ex: 2010-10-19 Valoarea goala (empty) pentru acest tip de date este '0000-00-00'.
    - **DATETIME -** similara cu DATE, insa stocheaza data si ora exacta, la secunda, deci inclusiv timpul. Ex: 2010-10-19 19:40:21. Valoarea goala pentru acest tip de date este 0000-00-00 00:00:00
    - **TIMESTAMP** - formatul este similar cu DATETIME, insa are un interval de valori mai mic (poate tine date intre 1970-01-01 00:00:01 si 2038-01-09 03:14:07). Diferenta mai importanta poate este ca are capcitatea de actualizare automata la operatii de INSERT si UPDATE. Daca vrem ca valoarea sa se actualizeze cu data curenta doar la INSERT, specificam atributul DEFAULT CURRENT_TIMESTAMP. Daca dorim sa se actualizeze si la UPDATE putem specifica atributul ON_UPDATE CURRENT_TIMESTAMP. Daca nu specificam nici unul din aceste atribute, se considera ambele, deci se stocheaza data curenta atat la insert cat si la update.
    - **TIME** - formatul este HH:MM:SS . Ex: 19:40:21. Pot fi specificate si intervale mai mari de 24h. Ex: 98:59:59 (98 de ore, 59 de minute si 59 de secunde).
    - **YEAR** - formatul este YYYY daca este declarat YEAR(4) sau YY daca este declarat YEAR(2); valori acceptate: 1901 - 2155

- Siruri de caractere
    - **CHAR**
        - are dimensiune fixa cuprinsa intre 0 si 255
        - de ex., CHAR(30) poate memora un text de cel mult 30 de caractere; se va aloca acelasi spatiu, indiferent de lungimea textului memorat
        - se va folosi CHAR pentru campuri care memoreaza text de aceeasi lungime, de exemplu 'Y' sau 'N', 'DA' sau 'NU'
    - **VARCHAR**
        - are dimensiune variabila cuprinsa intre 0 si 65535
        - de ex., VARCHAR(30) poate memora un text de cel mult 30 de caractere; se va aloca spatiu in functie de lungime textului memorat
        - se va folosi VARCHAR pentru campuri care memoreaza text de lungime variabila
    - **TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT**
        - pentru blocuri mari de text in format ASCII
    - **BINARY, VARBINARY, TINYBLOB, BLOB (Binary Large Object), MEDIUMBLOB si LONGBLOB**
        - Siruri de octeti (date in format binar), de exemplu fisiere
- **ENUM**
    - permite memorarea unei valori dintr-o lista predefinita. Pot fi definite maxim 65535 valori distincte.
    - Daca aceasta coloana permite valoarea NULL, atunci valoarea NULL este si cea implicita.
    - Altfel, se poate specifica o valoare implicita ca fiind una dintre valorile listei.
    - In caz ca nu se specifica o valoare implicita, si nu se completeaza o valoare la insert, in mod implicit se va alege prima valoare din lista.
- **SET**
    - asemanator cu ENUM doar ca se poate memora orice combinatie dintre elementele unei liste predefinite
- !Daca incercati sa memorati o valoare mai mare decat maximul acceptat de tipul campului, MySQL va

trunchea sau va rotunji automat valoarea la valoarea maxima permisa.

3. **Atributele unui camp**
   - NULL versus NOT NULL
     - NULL – lipsa informatiei in campul respectiv
     - NOT NULL
       - daca nu introduc nicio valaore la INSERT, se completeaza cu DEFAULT
       - Daca nu am DEFAULT, valoarea nula pentru tipul campului respectiv:" pt string, 0 pt numere, 0000-00-00 pt DATE
   - DEFAULT
     - In lipsa unei valori la INSERT pentru coloana respectiva, se memoreaza automat valoarea ce urmeaza dupa atributul DEFAULT
   - UNSIGNED
     - numere fara semn
   - ZEROFILL
     - completeaza cu 0 un numar astfel incat sa ajunga la x cifre: daca atribuim coloanei de exemplu INT(5), iar valoarea memorata este 254, se va afisa 00254.
   - AUTO_INCREMENT
     - poate fi adaugat unei coloane de tip numeric.
     - Cand nu se specifica o valoare pentru coloana respectiva la operatia de INSERT (sau valoarea specificata este 0), atunci coloana se auto-completeaza cu MAX + 1 unde MAX este cea mai mare valoare pentru acea coloana din toate inregistrarile de pana atunci.
     - Astfel, aceasta coloana va avea ca valori numere intregi, consecutive, insa daca se sterge unul din randuri, celelalte nu-si vor modifica valorile pentru aceasta coloana, deci numerele nu vor mai fi consecutive. Acest atribut se specifica in mod clasic pentru coloana de tip id, ce va fi si **cheie primara** a tabelului respectiv.
   - CHARACTER SET si COLLATE
     - Pentru coloane de tip sir de caractere (CHAR, VARCHAR, TEXT, etc) se poate specifica un set de caractere (ex: latin1 sau utf8) cu ajutorul atributului CHARACTER_SET. Atributul COLLATE specifica ceea ce se numeste "collation", termen ce desemneaza modul in care sunt comparate si ordonate caracterele dintr-un set de caractere.

4. **Indecsi**
   - **Index simplu** - folosind cuvantul cheie **INDEX** sau **KEY** (sinonime)
     Acest index nu impune nici o constrangere asupra valorilor acestei coloane, insa optimizeaza (ca orice index) pentru ordonare si cautare. Se recomanda pentru campurile care apar frecvent in clauzele WHERE, ORDER BY, GROUP BY si pentru campurile de legatura
   - **Index unic** - folosind cuvintele **UNIQUE INDEX** sau **UNIQUE KEY**
     - Folosind acest index impunem ca valorile acestei coloane sa fie unice. De exemplu, daca avem in tabelul nostru o coloana numita "email", un index unic pe aceasta coloana nu va permite existenta unor inregistrari cu aceeasi valoare pentru coloana "email". Exceptia de la aceasta regula este atunci cand o coloana nu are atributul NOT NULL (deci implicit are NULL) si poate aparea valoarea NULL de mai multe ori in cadrul acestei coloane.
   - **Cheie primara** - index ce forteaza atat **unicitatea** valorilor acestei coloane cat si prezenta unei valori (nu se accepta valoarea NULL). In mod optim, orice tabel are o coloana cu index-ul **cheie primara**, iar acea coloana este intreg si are in plus atributul **AUTO_INCREMENT**
   - In anumite situatii cheia primara poate fi compusa din mai multe campuri (**cheie primara compusa**); pentru a crea o astfel de cheie primara, se va specifica lista de campuri separate prin virgula: **PRIMARY KEY (utilizatorId, mesajId)**
   - Exista urmatoarele modalitati prin care se creeaza un index:
     - folosind sintaxa **CREATE TABLE** in momentul cand realizam tabelul

- **ALTER TABLE...ADD INDEX**
- folosind **CREATE INDEX** (similar cu ALTER TABLE... ADD INDEX) insa folosind CREATE INDEX nu se poate adauga **cheie primara**

- Se poate specifica o cheie straina intr-un tabel folosind modificatorul **FOREIGN KEY;**
- cheia straina este acceptata numai in tabelele de tip **InnoDB,** in celelate tipuri de tabele modificatorul **FOREIGN KEY** fiind ignorat.
- Cele mai des utilizate tipuri de tabel in MySQL
  - MyISAM

    este optimizat pentru viteza si securitate. Suporta tabele de dimensiuni mari si permite compresie pentru a salva spatiu
  - InnoDB

    suporta tranzactii si chei straine si permite mai multor utilizatori sa execute acelasi SELECT

- **Operatori aritmetici :** +,-, *, / (daca se face impartire la 0 rezultatul va fi NULL;)
- **Operatori logici:** !/NOT, AND/&&, OR/||
- **Operatori de comparatie**
  - returneaza 1 pentru adevarat, 0 pentru fals si NULL daca nu se poate efectua comparatia.
  - pot compara atat numere cat si siruri de caractere. La compararea unui numar cu un sir se incearca transformarea sirului in numarul pe care il contine.
  - La compararea a doua siruri nu se va tine cont de litere mari/mici.
  - = egalitate; la compararea cu NULL intoarce NULL;
  - <> / != inegalitate;
  - <, <= , >, >=,
  - <=> - echivalenta; la compararea cu NULL intoarce 0 sau 1 (1 doar la NULL <=> NULL); daca ambii operanzi sunt diferiti de NULLse comporta ca si =;
  - IS NULL - testeaza daca operandul are valoarea NULL;
  - IS NOT NULL - testeaza daca operandul este diferit de NULL;
  - ISNULL(expr) - testeaza daca expresia are valoarea NULL;
  - expr BETWEEN min AND max - testeaza daca valoarea expresiei este in intervalul [min..max];
  - expr IN (value,...) - testeaza daca valoarea expresiei este in lista de valori specificata;
  - expr NOT IN (value,...) - testeaza daca valoarea expresiei nu este in lista de valori specificata;
  - IF(expr1,expr2,expr3) - testeaza valoarea de adevar a expr1 (falsa daca este 0 sau NULL) si intoarce expr2 pentru adevarat respectiv expr3 pentru fals

5. **Functii predefinite**

Functii pe siruri de caractere

Functiile pe siruri considera implicit primul caracter pe pozitia 1.

- ASCII(str) - intoarce codul ASCII al caracterului de pe pozitia 1 din sir. Daca sirul este vid intoarce 0.
- CONV(N, from_base, to_base) - converteste numarul N considerat in baza from_base in valoarea sa in baza to_base.
- CHAR(N,...) - converteste sirul de numere primite intr-un sir de caractere ale caror coduri ASCII sunt egale cu cele din sirul initial.
- CONCAT(str1,str2,...) - concateneaza sirurile primite ca argument, returnand sirul rezultat.

- LENGTH(str) - intoarce lungimea (numarul de caractere) a sirului primit ca si argument.
- LOCATE(substr,str)
  POSITION(substr IN str) - cauta prima aparitie a sirului substr in sirul str. Daca il gaseste returneaza pozitia, daca nu, returneaza 0.
- LEFT(str,len) - intoarce un sir format din primele len caractere din sirul dat.
- RIGHT(str,len) - intoarce un sir format din ultimele len caractere din sirul dat.
- SUBSTRING(str,pos,len)
  MID(str,pos,len) - intorc subsirul de pe pozitia pos, de lungime len caractere din sirul str.
- SUBSTRING_INDEX(str,delim,count) - intoarce subsirul pana la count aparitii ale caracterului delimitator delim. Daca delim este pozitiv intoarce prima parte a sirului, daca este negativ, ultima parte.
- LTRIM(str), RTRIM(str), TRIM(str) - intoarce sirul str fara spatiile de la inceput (LTRIM), sfarsit (RTRIM) sau din ambele parti (TRIM).
- SPACE(N) - intoarce un sir format din N spatii.
- LOWER(str), UPPER(str) - intorc sirul str cu toate caracterele convertite la litere mici, respectiv litere mari.
- sir (NOT) LIKE tipar - compara sirul sir cu tiparul dat. Intoarce adevarat (1) daca se potrivesc. Tiparele pot include caracterele speciale: '_' care se potriveste cu orice caracter si '%' care poate inlocui orice secventa de 0 sau mai multe caractere. Ex. 'ana' LIKE 'an_', 'ana' LIKE 'a%', 'ana' NOT LIKE 'a%z'.

Functii matematice

Toate functiile matematice intorc NULL in caz de eroare.

- ABS(X) - valoarea absoluta (fara semn) a lui x.
- SIGN(X) - testeaza semnul lui x, intoarce -1 pentru x negativ, 1 pentru x pozitiv si 0 in caz contrar
- MOD(N,M) , % - modulo (restul impartirii) lui N la M.
- FLOOR(X) - cea mai mare valoare intreaga mai mica decat x (rotunjire in jos)
- CEILING(X) - cea mai mica valoare intreaga mai mare decat x (rotunjire in sus)
- ROUND(X) - rotunjire la cel mai apropiat intreg
- EXP(X) - exponentiala (e la puterea x)
- LOG(X) - logaritm natural din x
- LOG10(X) - logaritm in baza 10 din x
- POW(X,Y) - x la puterea y
- SQRT(X) - radical de ordinul 2 din x
- PI() - numarul PI; implicit se afiseaza cu 5 zecimale dar in calcule intervine cu dubla precizie
- COS(X), SIN(X),TAN(X) - cosinus, sinus si tangenta de x
- RAND() , RAND(N) - genereaza un numar pseudo-aleator in intervalul [0..1]; prin N se poate specifica o valoare de initializare calcul
- LEAST(X,Y,...) - intoarce cea mai mica valoare din lista valorilor specificate
- GREATEST(X,Y,...) - intoarce cea mai mare valoare din lista valorilor specificate

Functii de tip data calendaristica si timp

- DAYOFWEEK(date) - indexul zilei din spatamana al datei specificate (1=duminica, 2=luni etc.)
- DAYOFYEAR(date) - numarul zilei din an pentru data specificata
- DAYNAME(date) - numele (in engleza) al zilei din data specificata
- MONTHNAME(date) - numele (in engleza) al lunii din data specificata
- YEAR(date) - extrage anul din data specificata

- HOUR(time) - extrage ora din timpul precizat
- MINUTE(time) - extrage minutul din timpul precizat
- SECOND(time) - extrage secunda din timpul precizat
- CURDATE() - intoarce data curenta in format 'YYYY-MM-DD'
- CURTIME() - intoarce ora curenta in format 'HH:MM:SS
- NOW() , SYSDATE() - intoarce data si ora curenta in format 'YYYY-MM-DD HH:MM:SS'
- SEC_TO_TIME(seconds) - intoarce timpul in format 'HH:MM:SS' reprezentat de numarul de secunde specificate
- TIME_TO_SEC(time) -converteste timpul specificat in numar de secunde scurse de la ora 00:00:00

6. **Crearea unui tabel**

CREATE TABLE nume_tabel (nume_coloana1 tip_date1[(lungime)] [atribute_extra], nume_coloana2 tip_date2[(lungime)] [atribute_extra], ... [UNIQUE | PRIMARY] KEY nume_index (coloana));

Exemplu:

```
CREATE TABLE utilizatori (
                    utilizatorId int(11) unsigned NOT NULL auto_increment,
                    username varchar(20) NOT NULL,
                    email varchar(25) NOT NULL,
                    varsta tinyint(4) unsigned default NULL,
                    sex enum('m','f') NOT NULL,
                    dataAdaugarii date NOT NULL,
                    PRIMARY KEY  (utilizatorId),
                    UNIQUE KEY username (username),
                    KEY email (email)
                   );
```

7. **Stergerea unui tabel**

DROP TABLE nume_tabel;

8. **Inserarea datelor intr-un tabel;**

INSERT INTO nume_tabel (coloana1, coloana2, ...) VALUES (valoare1, valoare2, ...);

- In cazul in care omitem anumite coloane de la insert, in ele se va adauga valoarea **NULL** daca atributele coloanei permit, sau valoarea DEFAULT implicita (specificata in definitia coloanei folosind atributul DEFAULT) sau cea explicita adica valoarea goala pentru acel tip de date.

sau

INSERT INTO nume_tabel VALUES (valoare1, valoare2, ...);

numarul de valori introduse trebuie sa fie egal cu numarul coloanelor tabelului

9. **Modificarea datelor dintr-un tabel**

UPDATE tabel SET coloana1=valoare1, coloana2=valoare2... WHERE ...

Exemplu:

```
UPDATE utilizatori SET username='test_modificat', email = 'test_modificat@yahoo.com' WHERE utilizatorId = 6;
```

10. **Stergerea datelor dintr-un tabel**

DELETE FROM tabel WHERE ...

Exemplu:

```
DELETE FROM mesaje WHERE subiect LIKE '%test%';
```

11. **Interogarea datelor – SELECT**
    a) Sintaxa simplificata

    SELECT lista_coloane FROM tabel [WHERE conditie] [ORDER BY coloana1,...[ASC|DESC]] [LIMIT a,b]

- **lista_coloane** poate fi de fapt una din urmatoarele expresii:
  - **o lista de coloane separate prin virgula**;
  - ***** simbol ce semnifica selectarea tuturor coloanelor din tabel
  - **functii SQL predefinite**, aplicate unor coloane sau nu neaparat. Prezenta acestora se poate combina (sau nu) cu prezenta altor coloane in lista selectului.
- **WHERE** specifica conditia de includere a datelor in selectie
- **ORDER BY** specifica coloana sau coloanele dupa care se face sortarea datelor; implicit, sortarea este crescatoare; daca se doreste sortare descrescatoare, se va specifica clauza DESC
  - Exemplu:

  SELECT * FROM utilizatori ORDER BY anNastere, username DESC;
- **LIMIT m** - selecteaza m inregistrari
- **LIMIT n,m** - selecteaza m inregistrari incepand cu inregistrarea numarul n (numerotarea incepe de la 0). ex: LIMIT 1, 5 selecteaza 5 inregistrari excluzand prima inregistrare. (se porneste de la inregistrarea 1, dar prima inregistrare este inregistrarea 0)

b) Folosirea aliasurilor

1. alias pentru tabel

   SELECT u.username, u.anNastere FROM utilizatori u;
   - tabelul utilizatori are aliasul u
2. alias pentru coloana

   SELECT nume, prenume,Year(datan) AS an FROM Deponent ORDER BY an;
   - pentru coloana Year(datan) s-a stabilit aliasul an

c) **Extragerea informatiilor de tip total**

SELECT lista_coloane
FROM tabel
[WHERE conditie]
[GROUP BY coloana|expresie]
[HAVING conditie_includere_grup]
[ORDER BY coloana1,...[ASC|DESC]] [LIMIT a,b]

- Clauza GROUP BY specifica dupa ce valoare se va face gruparea. In majoritatea cazurilor expresia de grupare este reprezentata de o singura coloana. Toate inregistrarile cu aceeasi valoare pentru expresia de grupare vor fi considerate ca facand parte din acelasi grup.
  Daca clauza GROUP BY lipseste, dar in lista_campuri apar informatii de sumarizare se considera implicit ca toate inregistrarile fac parte din acelasi grup.
- Clauza HAVI NG permite selectarea grupurilor care sunt luate in considerare. Conditia din HAVING se aplica dupa constituirea grupului, pe cand conditia din clauza WHERE in timpul selectarii inregistrarilor. In plus, acesta clauza poate cuprinde doar
  referinte la campuri si aliasuri de campuri din lista de campuri a comenziiS ELECT sau functii de sumarizare pe grup.
- Lista de campuri din comanda SELECT poate include urmatoarele functii de tip total:
  - COUNT(*) - numara cate inregistrari sunt selectate in fiecare grup.
  - COUNT(DISTINCT expr) - numara cate inregistrari pentru care expresia are o valoare distincta sunt selectate in fiecare grup.
  - AVG(expr) - calculeaza media aritmetica a valorilor expresiei pentru inregistrarile din fiecare grup.
  - MIN(expr) - calculeaza valoarea minima a expresiei pentru toate inregistrarile din fiecare grup.
  - MAX(expr) - calculeaza valoarea maxima a expresiei pentru toate inregistrarile din fiecare grup.
  - SUM(expr) - calculeaza suma valorilor expresiei pentru toate inregistrarile din fiecare grup.

Atentie! Nu este permisa combinarea functiilor de sumarizare cu campurile din tabele decat in cazul in care

campurile respective au o valoare unica in cadrul grupului. In cele mai multe cazuri acestea vor fi campurile din clauza GROUP BY.

Exemplu:

1. varsta minima si maxima:

```
SELECT MIN(YEAR(CURDATE())-YEAR(dataN)) AS varstaMinima, MAX(YEAR(CURDATE())-
    YEAR(dataN)) AS varstaMaxima  FROM deponenti;
```

2. Varsta minima si maxima pentru domni si varsta minima si maxima pentru doamne

```
SELECT MIN(YEAR(CURDATE())-YEAR(dataN)) AS varstaMinima, MAX(YEAR(CURDATE())-
    YEAR(dataN)) AS varstaMaxima  FROM deponenti
GROUP BY sex;
```

3.
```
SELECT tip_cont, COUNT(*) AS NrConturi, MIN(sold), MAX(sold), AVG(sold), SUM(sold)
FROM conturi
GROUP BY tipCont
HAVING SUM(sold)>10000;
```

d) **Selectarea datelor din mai multe tabele**

- **Produs cartezian**

```
/*
fara a folosi sintaxa JOIN, se extrag informatii din ambele tabele, fara nici o conditie.
fiecare inregistrare din primul tabel este combinata cu o inregistrare din cel de-al doilea
*/
SELECT * FROM mesaje, utilizatori
```

- **INNER JOIN**
  - Folosind sintaxa INNER JOIN, extragem fiecare inregistrare din primul tabel ce are ca pereche o inregistrare din cel de-al doilea tabel conform unei conditii.
  - De obicei, conditia folosita este: valoarea cheii primare din primul tabel (sa zicem coloana utilizatorId) este egala cu valoarea aceleiasi coloane (cheie straina) din al doilea tabel (coloana utilizatorId din al doilea tabel)
  - Exemplu:

```
SELECT
 *
FROM
 mesaje m
 INNER JOIN utilizatori u  ON m.utilizatorId = u.utilizatorId
ORDER BY m.dataMesaj DESC
```

- **LEFT JOIN**
  - Sintaxa **LEFT JOIN** difera de **INNER JOIN** prin faptul ca selecteaza toate inregistrarile din primul tabel, indiferent ca au corespondent in cel de-al doilea sau nu.
    Similar cu **INNER JOIN** se foloseste conditia de combinare a inregistrarilor din primul tabel cu cele din al doilea tabel, dar daca nu este gasita o inregistrare corespondenta in al doilea tabel se va afisa inregistrarea din primul tabel iar pe pozitiile campurilor din al doilea tabel se va afisa valoarea NULL.
  - Exemplu:

```
/*
adaug intai un mesaj ce nu are corespondenta in tabelul utilizatori pentru coloana utilizatorId = 999
*/
INSERT INTO mesaje (mesajId, utilizatorId, subiect, mesaj, dataMesaj) VALUES (NULL, 999, 'subiect
test', 'mesaj test', now());


SELECT
 *
FROM
```

```
 mesaje m
LEFT JOIN utilizatori u  ON m.utilizatorId = u.utilizatorId
ORDER BY m.dataMesaj DESC
```

- **Alte exemple folosind INNER JOIN si LEFT JOIN:**

```
/*
cu ajutorul LEFT JOIN vrem sa gasim DOAR inregistrarile din primul tabel ce nu au corespondenta in al doilea tabel folosind operatorul IS NULL
Putem folosi acest query pentru a identifica si a sterge mesajele utilizatorilor ce nu mai sunt in baza de date.
Desi, in mod normal, cand stergem un utilizator, trebuie sa cautam si sa stergem si mesajele aferente (de exemplu, stergem utilizatorul cu utilizatorId = 999, apoi stergem din tabelul mesaje toate mesajele ce au utilizatorId = 999
*/

SELECT
 m.*,
 u.utilizatorId
 FROM
 mesaje m
 LEFT JOIN utilizatori u ON m.utilizatorId = u.utilizatorId
 WHERE
 u.utilizatorId IS NULL

/*
extragem doar numele de utilizator, subiectul si corpul mesajului
vrem doar mesajele scrise dupa anumita data
*/

SELECT
 u.username,
 m.subiect,
 m.mesaj
 FROM
 mesaje m
 INNER JOIN utilizatori u ON m.utilizatorId = u.utilizatorId
 WHERE
 m.dataMesaj > '2009-04-01'
 ORDER BY m.dataMesaj DESC
```