Lecţia 5 – Operatori şi funcţii MySQL

5.1 Tipuri de operatori

În această lecţie vom aborda subiectul referitor la **operatori** şi **funcţii predefinite** în **MySQL**. În ceea ce priveşte operatorii, în **MySQL** avem trei tipuri de operatori:

- **matematici**;
- **logici**;
- **de comparare**;

Expresiile care apar împreună cu aceşti operatori se numesc *operanzi*. Operanzii pot fi coloane ale tabelelor, valori sau expresii.

5.2 Operatori matematici

Operatorii matematici pe care îi întâlnim în **MySQL** sunt: **+, -, *, /, %.**

Astfel, „+" este operatorul pentru adunare, „**–**" este operatorul pentru scădere, „*****" este operatorul pentru înmulţire, „/" este operatorul pentru împărţire iar „**%**" este operatorul pentru restul împărţirii a două numere (**modulo**). Dacă avem împărţire la 0, deci folosim operatorul „/", atunci rezultatul va fi **NULL**.

5.3 Operatori logici

Operatorii logici utilizaţi în **MySQL** sunt: **AND**, **OR**, **XOR**, **NOT**.

În **MySQL** operatorii logici întorc rezultatul 1 pentru adevărat, respectiv rezultatul 0 pentru fals. O expresie evaluată din punct de vedere logic poate fi adevărată sau falsă.

Operatorul **AND** este operatorul „**şi logic"**. În **MySQL** pentru „**şi logic**", mai avem şi operatorul **&&**.

Dacă avem 2 expresii pe care le evaluăm logic, fiecare din aceste expresii poate fi adevărată sau falsă. Iată în continuare tabla de valori cu toate variantele posibile pentru 2 expresii, folosind operatorul **AND** (**&&**).

1 AND 1 = 1

1 AND 0 = 0

0 AND 1 = 0

0 AND 0 = 0

Cu alte cuvinte, tabla aceasta de valori a operatorului logic **AND** poate fi explicată astfel: dacă prima expresie este adevărată, deci întoarce rezultatul 1, iar cea de-a doua expresie este tot adevărată, atunci rezultatul 1 **&&** 1 este tot 1, deci operatorul **AND** returnează 1 (adevărat) dacă ambele expresii sunt adevărate.

Dacă una dintre expresii este adevărată iar cealaltă este falsă, atunci rezultatul este fals, deci 1 **AND** 0 = 0 şi 0 **AND** 1 = 0.

Evident că, dacă avem ambele expresii false, 0 **AND** 0 = 0.

Deci, operatorul **AND** (**&&**) va returna adevărat doar atunci când ambele expresii sunt adevărate, în orice alt caz rezultatul este fals.

Operatorul **OR** este operatorul „**sau logic**”. Mai avem pentru „**sau logic**” şi operatorul „||”. În continuare prezentăm tabla de valori cu variantele posibile pentru 2 expresii evaluate împreună cu operatorul **OR**.

1 OR 1 = 1

1 OR 0 = 1

0 OR 1 = 1

0 OR 0 = 0

Cu alte cuvinte, tabla aceasta de valori a operatorului logic **OR** poate fi explicată astfel: dacă prima expresie este adevărată, deci întoarce rezultatul 1, iar cea de-a doua expresie este tot adevărată, atunci rezultatul expresiei 1 || 1 este tot 1, deci operatorul **OR** returnează 1 (adevărat) dacă ambele expresii sunt adevărate.

Dacă una dintre expresii este adevărată iar cealaltă expresie este falsă, atunci rezultatul este adevărat, deci 1 **OR** 0 = 1 şi 0 **OR** 1 = 1. Practic, atunci când evaluăm valoarea de adevăr a două expresii logice între care am folosit operatorul **OR** rezultatul este 1 (adevărat) dacă cel puțin una din expresii este adevărată. Așadar, este suficient să fie una singură din expresii adevărată pentru ca rezultatul obţinut să fie adevărat.

Evident că, dacă avem ambele expresii false, 0 **OR** 0 = 0.

Deci, operatorul **OR** (||) va returna adevărat atunci când cel puţin una dintre expresii este adevărată, iar în cazul în care ambele expresii sunt false şi rezultatul este fals.

Operatorul **XOR** este operatorul „**sau exclusiv**”. Acest operator returnează adevărat atunci când unul dintre operanzi este adevărat iar celălalt este fals. Atunci când ambii operanzi sunt adevăraţi sau ambii operanzi sunt falşi, rezultatul este fals. Iată în continuare tabla de valori cu variantele posibile pentru 2 expresii evaluate folosind operatorul **XOR** (**sau exclusiv**):

1 XOR 1 = 0

1 XOR 0 = 1

0 XOR 1 = 1

0 XOR 0 = 0

Deci, tabla de valori a operatorului logic **XOR** (**sau exclusiv**) poate fi explicată astfel: atunci când ambele expresii evaluate sunt fie adevărate, fie false, rezultatul este fals, în timp ce rezultatul adevărat va fi returnat doar atunci când una din expresii returnează adevărat iar cealaltă returnează fals.

Acest operator, **XOR**, este mai rar folosit în evaluarea valorii de adevăr a unor expresii.

Ultimul operator logic este **NOT**, operatorul de **negaţie**. În **MySQL** pentru negarea unei expresii mai avem şi operatorul „**!**”.

Acest operator de negaţie este foarte simplu de folosit şi de înţeles. Practic, dacă avem o expresie adevărată şi o negăm (îi aplicăm acest operator **NOT**) ea devine falsă, şi reciproc, dacă o expresie este falsă şi ea este negată, atunci rezultatul expresiei va fi adevărat. Deci, tabla de valori posibile pentru acest operator este următoarea:

NOT 1 = 0

NOT 0 = 1

5.4 Operatori de comparare

De asemenea, în **MySQL** mai avem şi operatorii de comparare. În această categorie avem următorii operatori:

< - compară dacă o expresie este **mai mică** decât altă expresie;

> - compară dacă o expresie este **mai mare** decât altă expresie;

<= - compară dacă o expresie este **mai mică sau egală** decât altă expresie;

>= - compară dacă o expresie este **mai mare sau egală** decât altă expresie;

= - compară dacă două expresii sunt **egale;**

!=, < > - compară dacă două expresii sunt **diferite;**

LIKE - testează dacă un şir de caractere are o anumită formă: dacă este prefixat respectiv postfixat sau nu de un anumit subşir, dacă acesta conţine un anumit subşir. Important de reţinut este şi faptul că, în **MySQL,** simbolul „**_**” (**underline**) ţine loc unui singur caracter, în timp ce simbolul „**%**” ţine loc oricâtor caractere. Acestea se mai numesc caractere de înlocuire.

IS NULL - testează dacă o valoare este **NULL**

IS NOT NULL - testează dacă o valoare nu este **NULL**

O menţiune importantă este aceea că atunci când se testează anumite expresii (valori, câmpuri ale unei tabele) pentru a determina dacă valoarea lor este **NULL** nu putem folosi operatorii **=, !=, <, <=, >, >=.**
Pentru a testa dacă o expresie este nulă se folosesc exclusiv cei doi operatori prezentaţi anterior și anume **IS NULL**, respectiv **IS NOT NULL**.

BETWEEN - testează dacă o valoare se găseşte între 2 valori date; forma este următoarea ***expresie* BETWEEN *valoare_minimă* AND *valoare_maximă*;** deci, **BETWEEN** verifică dacă ***expresie*** se găseşte în intervalul închis cu capetele ***valoare_minimă***, respectiv, ***valoare_maximă***;
Operatorul **BETWEEN** poate fi înlocuit cu operatorii >= şi <=. Forma ar fi următoarea:

expresie* >= *valoare_minimă* AND *expresie* <= *valoare_maximă

Astfel, deducem mai limpede, că operatorul **BETWEEN** ia în considerare atunci când evaluează expresia inclusiv valorile limită ale intervalului, în acest caz denumite **valore_minimă**, repsectiv, **valoare_maximă**.

IN(*val_1, ..., val_n*) - testează dacă o valoare aparţine unei mulţimi de valori trecută ca argumente între parantezele operatorului **IN**;

NOT IN(*val$_1$, ..., val$_n$*) - testează dacă o valoare nu aparţine mulţimii de valori dată între parantezele operatorului.

Aceşti operatori au mai fost explicaţi la lecţia anterioară la subcapitolul **Operatori folosiţi în clauza WHERE**, unde puteţi găsi prezentări lămuritoare pentru fiecare operator în parte.

5.5 Funcţii predefinite MySQL

În **MySQL** avem funcţii simple care prelucrează fiecare înregistrare şi pentru fiecare înregistrare returnează un rezultat şi funcţii de agregare (de grup) care prelucrează un set de înregistrări şi returnează un singur rezultat pentru acel set de înregistrări.

O funcţie primeşte şi *parametri*. *Parametrii* unei funcţii sunt valori pe care o funcţie le primeşte pentru a le prelucra şi a returna un rezultat. Parametrii unei funcţii se mai numesc şi *argumente*.

Putem clasifica funcţiile simple în mai multe tipuri:

- **matematice;**
- **de comparare;**
- **condiţionale;**
- **pentru şiruri de caractere;**
- **pentru date calendaristice.**

În continuare, vom lua fiecare din aceste tipuri de funcţii şi vom prezenta câteva dintre cele mai utilizate funcţii din fiecare tip.

5.6 Funcţii matematice

Funcţiile matematice sunt folosite pentru efectuarea de operaţii matematice. Ele pot fi utlizate fie pentru realizarea de operaţii matematice cu numele coloanelor, fie cu valori specifice.

În continuare vom prezenta câteva dintre cele mai cunoscute funcţii matematice disponibile în **MySQL**. Fiecare funcţie primeşte unul sau mai mulţi parametri care sunt trecuţi între paranteze rotunde – „**()**”. Aceşti parametri pot reprezenta numele unor coloane sau numere. Astfel, avem funcţiile:

- **ABS(*n*)** – returnează modulul sau valoarea absolută a unui număr;
- **CEILING(*n*)** – returnează cea mai mică valoare întreagă mai mare ca *n*;
- **FLOOR(*n*)** – returnează cea mai mare valoare întreagă mai mică ca *n*;
- **POW(*a*,*b*)** – returnează rezultatul ridicării la putere, adică a^b, deci primul parametru reprezintă baza, iar cel de-al doilea exponentul;
- **ROUND(*n*)** – returnează valoarea rotunjită a numărului primit ca paramentru, rotunjirea se face fără zecimale; dar această funcţie poate primi şi un al doilea parametru care reprezintă numărul de zecimale la care se face rotunjirea, deci, mai avem forma **ROUND(*n*,*d*)** care va returna valoarea rotunjită a lui *n* cu *d* zecimale;
- **TRUNCATE(*n*,*d*)** – returnează valoarea tăiată (trunchiată) a lui *n* cu *d* zecimale; de exemplu **TRUNCATE**(1.284,1) va returna 1.2;
- **RAND()** – returnează un număr aleatoriu între 0 şi 1; această funcţie nu are parametri;

- **SQRT(*n*)** – returnează valoarea rădăcinii pătrate (radicalul) unui număr;
- **MOD(*a,b*) –** returnează restul împărţirii lui a la b; acelaşi rezultat se poate obţine şi folosind operatorul „**%**” – **modulo**.

5.7 Funcţii de comparare

În continuare vom prezenta câteva funcţii de comparare:

- **LEAST(val1,val2,...)** – returnează cel mai mic parametru dintr-o listă de parametri; această funcţie trebuie să aibă cel puţin 2 parametri; dacă unul din argumente este **NULL** atunci rezultatul este **NULL**;
- **GREATEST(val1,val2,...)** – returnează cel mai mare parametru dintr-o listă de parametri; similar cu funcţia **LEAST()** şi această funcţie trebuie să aibă cel puţin 2 parametri;
- **INTERVAL(n,n1,n2,...)** – această funcţie compară valoarea lui **n** cu setul de valori care urmează – **n1**, **n2**, **...**; este necesar ca setul de valori ce va fi comparat cu **n** să fie ordonat crescător, deci **n1 < n2 < n3 ...** ; funcţia va returna 0 dacă **n<n1,** 1 dacă **n<n2**, deci va returna poziţia pe care este găsită prima valoare mai mare decât primul parametru al funcţiei, considerând că primul parametru cu care se compară această valoare se află pe poziţia 0, al doilea pe poziţia 1; dacă **n** este **NULL** funcţia va returna -1.

5.8 Funcţii condiţionale

Continuăm prezentarea cu funcţiile condiţionale:

- **IF(expresie_testată,expr1,expr2)** – această funcţie primeşte trei parametri, se verifică valoarea de adevăr a primului argument al funcţiei, **expresie_testată**, dacă valoarea de adevăr este **TRUE (adevărat)** atunci funcţia returnează cel de-al doilea parametru, **expr1**, altfel, adică valoarea de adevăr a evaluării expresiei de testat este **FALSE (fals)** atunci funcţia va returna **expr2**.
- **IFNULL(expr1,expr2)** – această funcţie returnează **expr1** dacă **expr1** este diferit de **NULL** sau va returna **expr2** dacă **expr1** este **NULL**. Valoarea returnată de această funcţie poate fi numerică sau şir de caractere, în funcţie de contextul în care este folosită.
- **NULLIF(expr1,expr2)** – această funcţie compară cele 2 expresii primite ca parametri, dacă sunt egale funcţia va returna **NULL**, iar dacă cele 2 expresii sunt diferite va returna **expr1**, deci va returna primul parametru primit.

5.9 Valoarea NULL

După cum am văzut în sintaxa comenzii care creează o tabelă (**CREATE TABLE**), după fiecare coloană a tabelei se pot trece nişte specificatori. Ne vom ocupa aici de specificatorii **NULL** respectiv **NOT NULL**. Primul, care este şi implicit (dacă nu-l trecem, se asumă automat că acea coloană are valoarea **NULL**), se referă la faptul că în coloana respectivă pot să apară şi valori de tip **NULL**. Astfel, dacă într-o comandă **INSERT** este omisă o valoare pentru o anumită coloană, în acea coloană se va trece automat valoarea **NULL**.

Al doilea specificator, **NOT NULL**, **nu** permite înregistrări cu valori **NULL** în acea coloană. În acest caz, dacă este omisă valoarea acelei coloane, în ea se va trece automat:

- 0 dacă este de tip numeric;

- şirul vid dacă acea coloană este de tip şir de caractere;

- 0000-00-00 dacă acea coloană este de tip dată calendaristică, etc.

Valoarea **NULL** reprezintă, de fapt, **lipsa unei valori**. Foarte important este de menţionat aspectul că nu trebuie să se facă confuzie între valoarea **NULL** şi 0, de exemplu pentru coloane numerice, sau şirul vid ("") pentru coloane de tip şir de caractere. Acestea sunt valori, şi 0 şi şirul vid sunt valori, pe când **NULL** specfică de fapt **lipsa unei valori** în acel câmp.

5.10 Funcţii pentru şiruri de caractere

Funcţiile pentru şiruri de caractere sunt folosite atunci când se lucrează cu şiruri de caractere, deci parametrii acestor funcţii pot fi coloane sau valori de tip şir de caractere. Astfel, prezentăm în continuare câteva dintre aceste funcţii pentru text:

- **CONCAT(s1,s2,s3,...)** – va concatena (alipi) toate şirurile date ca argumente ale funcţiei; dacă unul dintre argumente este **NULL** atunci rezultatul funcţiei va fi **NULL**.
- **CONCAT_WS(separator,s1,s2,s3,...)** – va concatena (alipi) toate şirurile date ca argumente ale funcţiei cu separatorul dat ca prim argument între ele; dacă separatorul este **NULL** atunci rezultatul funcţiei va fi **NULL**, dar dacă unul din celelalte argumente este **NULL,** atunci funcţia va returna celelate şiruri concatenate, ignorând **NULL**.

Atenţie, în **MySQL NU** există un operator de concatenare (ca în PHP, spre exemplu). Deci, în MySQL, pentru concatenare vom folosi una din aceste funcţii.

- **CHAR_LENGTH(s) – returnează lungimea şirului *s*;**
- **LENGTH(s) - lungimea (nr. de caractere) şirului *s*;**
- **LPAD(s,lungime_finală,şir_completare) – şirul *s* este completat la stânga cu şirul *şir_completare* până ajunge la lungimea *lungime_finală*;**
- **RPAD(s,lungime_finală,şir_completare) – şirul *s* este completat la dreapta cu şirul *şir_completare* până ajunge la lungimea *lungime_finală*;**
- **LTRIM(s) - întoarce şirul obţinut din *s* prin eliminarea spaţiilor inutile din stânga;**
- **RTRIM(s) - întoarce şirul obţinut din *s* prin eliminarea spaţiilor inutile din dreapta;**
- **TRIM(s) - întoarce şirul obţinut din *s* prin eliminarea spaţiilor inutile atât din dreapta cât şi din stânga;**
- **SUBSTR(s,pos,nr_caractere) – întoarce din şirul *s*, un subşir începând de la poziția *pos*, subşir de lungime *nr_caractere*; dacă lipsește al treilea parametru care reprezintă câte caractere vor fi extrase în subșir, atunci subșirul va extrage toate caracterele începând de la poziția *pos* până la final; și funcția SUBSTRING este similară funcției SUBSTR;**

- **LEFT(s,nr_caractere) – întoarce primele *nr_caractere* din șirul *s*;**
- **RIGHT(s,nr_caractere) – întoarce ultimele *nr_caractere* din șirul *s*;**
- **LOCATE(s1,s) – returnează poziția primei apariții a subșirului *s1* în șirul *s*, respectiv, 0 dacă *s1* nu se găsește în șirul *s*;**
- **LOCATE(s1,s,pos) – returnează poziția primei apariții a subșirului *s1* în șirul *s*, începând de la poziția *pos*; returnează 0 dacă *s1* nu se găsește în șirul *s*;**
- **UPPER(s) - întoarce şirul obţinut din *s* prin convertirea tuturor literelor mici la litere mari;**
- **LOWER(s) - întoarce şirul obţinut din *s* prin convertirea tuturor literelor mari la litere mici;**

5.11 Funcţii pentru date calendaristice

O altă categorie de funcţii **MySQL** este cea a funcţiilor pentru date calendaristice. Trebuie să ne amintim că singurul format de dată acceptat de **MySQL** este *an-lună-zi* (AAAA-LL-ZZ), sau, mai cunoscut acest format după denumirea în limba engleză *year-month-day*, sau prescurtarea YYYY-MM-DD. Formatul în care se păstrează ora în baza de date este *oră-minut-secundă* (HH-MM-SS), format mult mai cunoscut după denumirea în limba engleză hour-minute-second sau după prescurtarea HH-MM-SS. Formatul în care se stochează atât data cât şi ora este *year-month-day hour-minute-second* (AAAA-LL-ZZ HH-MM-SS sau YYYY-MM-DD HH-MM-SS). Astfel avem următoarele funcţii în această categorie:

- **CURDATE()** – returnează data curentă;
- **CURRENT_DATE()** - returnează data curentă;
- **CURTIME()** – returnează ora curentă;
- **CURRENT_TIME()** – returnează ora curentă;
- **CURRENT_TIMESTAMP()** – returnează data şi ora curentă;
- **NOW()** – returnează data şi ora curentă;
- **YEAR(data)** – returnează anul din data introdusă ca parametru;
- **MONTH(data)** – returnează valoarea numerică a lunii din data primită ca argument (valori posibile de la 1 la 12);
- **DAY(data)** - returnează valoarea numerică a zilei din data primită ca argument (valori posibile de la 1 la 31);
- **DAYOFMONTH(data)** - returnează valoarea numerică a zilei din data primită ca argument (valori posibile de la 1 la 31);
- **HOUR(time)** – returnează ora din valoarea parametrului funcţiei (intervalul de valori este de la 0 la 23 dacă parametrul reprezintă o oră dar poate lua şi valori mai mari dacă parametrul reprezintă un timp contorizat – număr de ore, minute şi secunde contorizate);

- **MINUTE(time)** - returnează minutul din valoarea parametrului funcţiei (intervalul de valori care poate fi returnat este de la 0 la 59);
- **SECOND(time)** - returnează seunda din valoarea parametrului funcţiei (intervalul de valori care poate fi returnat este de la 0 la 59);
- **DAYNAME(data)** - returnează numele, în limba engleză, al zilei săptămânii din data care este trecută ca parametru;
- **MONTHNAME(data)** - returnează numele, în limba engleză, al lunii din data care este primită ca parametru;
- **DAYOFWEEK(data)** – returnează indexul zilei din săptămână din data primită ca parametru (returnează 1 pentru duminică, 2 pentru luni, ..., 7 pentru sâmbătă);
- **DAYOFYEAR(data)** - returnează indexul zilei din anul din data primită ca parametru (valoare posibilă returnată de la 1 la 366);
- **LAST_DAY(date)** – returnează ultima zi din luna datei primită ca parametru;

5.12 Funcţii de agregare

De asemenea, în **MySQL** mai avem o categorie de funcţii care operează asupra mai multor înregistrări (linii) dintr-o tabelă a unei baze de date, calculând şi returnând o singură valoare. Aceste funcţii poartă denumirea de funcţii agregat sau funcţii de agregare. Avem funcţii de agregare matematice şi o funcţie agregat pentru numărare.

Funcţiile de agregare matematice sunt:

- **MIN(coloană) –** returnează valoarea minimă pentru un set de înregistrări din coloana primită ca parametru;
- **MAX(coloană) –** returnează valoarea pentru un set de înregistrări din coloana primită ca parametru;
- **SUM(coloană) –** returnează suma valorilor din coloana specificată ca parametru;
- **AVG(coloană) –** returnează media aritmetică a valorilor din coloana specificată ca parametru;

Prezentăm în continuare şi funcţia de agregare de numărare. Este vorba de funcţia:

- **COUNT(coloană) –** returnează numărul de înregistrări nenule din coloana primită ca parametru pentru un set de înregistrări;

Aceste funcţii de agregare au fost prezentate pe larg, cu exemple concrete de utilizare, în cadrul lecţiei numărul 4 la prezentarea clauzelor instrucţiunii **SELECT.** În instrucţiunea de regăsire a datelor, instrucţiunea **SELECT**, pentru a utiliza funcţiile de agregare trebuie folosită clauza **HAVING**, deci, funcţiile de agregare trebuie folosite împreună cu această clauză **HAVING**.

Pe parcursul acestei lecţii am prezentat operatorii întâlniţi în **MySQL** şi câteva dintre cele mai cunoscute funcţii predefinte ale **SGBD**-ului.

În lecţia următoare ne vom ocupa de **uniunile** între tabele sau **join**-uri, tipuri de **join**-uri, precum şi de reuniuni. Aşadar, vom trece la noţiuni mai complexe ale **MySQL**.