4. TIPURI DE DATE MySQL

Cuprins

4.1. Consecintele alegerii unui tip de date

Alegerea unui tip de date pentru o coloana a unei tabele SQL are multiple implicatii, de care trebuie tinut cont in etapa de design a bazei de date. Enumeram cateva mai importante:

- domeniul de valori posibile pentru acea coloana. In MySQL, orice valoare in afara domeniului valid este fie respinsa (generand o eroare), fie trunchiata astfel incat sa fie adusa in plaja de valori legale ale coloanei din care face parte
- precizia dorita. In cazul tipurilor de date numerice, la memorarea numerelor rationale pot exista aproximari/rotunjiri. Atunci cand avem nevoie de exactitate in operatiile aritmetice vom dori ca precizia calculelor sa se afle sub controlul nostru
- spatiul ocupat pe hard-disk. De obicei, cu cat domeniul de valori posibil este mai mare, cu atat spatiul ocupat pe hard-disk creste. Vom alege in general tipul de date cel mai mic care cuprinde toate valorile valide ale coloanei in cauza
- performanta DBMS in lucrul cu acel tip de date. Spre exemplu, valorile numerice pot fi memorate si ca siruri de caractere, insa operatiile cu ele vor fi mult mai lente comparativ cu cazul folosirii unui tip de date numeric

4.2. Tipuri de date numerice

4.2.1. Proprietati generale

Tipurile de date MySQL permit memorarea urmatoarelor tipuri de numere:

- intregi (numere fara parte zecimala)
- rationale. Acestea pot fi memorate in virgula fixa sau in virgula mobila. Nu toate numerele rationale au reprezentare numerica finita (ex: 10/3 are numar infinit de cifre), de aceea stocarea unora dintre numere in baza de date este subiect de trunchiere/rotunjire

La definirea unei coloane de tip numeric valorile vor fi memorate implicit cu semn si deci domeniul de valori posibil va fi unul care se intinde simetric in jurul valorii 0. Atunci cand specificam modificatorul UNSIGNED pentru coloana in cauza, plaja de valori posibile se muta in domeniul pozitiv, limita superioara dublandu-se pentru numerele intregi si ramanand pe loc pentru cele rationale.

4.2.2. Numere intregi

4.2.2.1. Tipuri de date si modificatori specifici

Tipurile de date MySQL folosite pentru a reprezenta numere intregi sunt TINYINT, SMALLINT, MEDIUMINT, INT si BIGINT. Ele difera prin domeniul de valori posibile si prin spatiul de stocare ocupat (vezi tabel).

Tip de date	Sinonime	Nr. octeti	Valori posibile, cu semn	Valori posibile, fara semn (unsigned)
TINYINT	-	1	-128...127	0...255
SMALLINT		2	-32768...32767	0...65535
MEDIUMINT	-	3	-8388608...8388607	0...16777215
INT	INTEGER	4	-2147483648...2147483647	0... 4294967295
BIGINT	-	8	(aprox) -9 x 10^{18}...9 x 10^{18}	0... 18 x 10^{18}

Valorile memorate intr-o coloana cu tip de date numeric intreg pot fi cu semn sau fara. In definitia coloanei pot fi specificati modificatorii SIGNED sau UNSIGNED, efectul lor fiind urmatorul:

- daca in definitia coloanei nu se specifica nici un modificator sau daca se foloseste explicit modificatorul SIGNED, valorile coloanei vor fi numere cu semn; plaja de valori posibile va fi simetrica in jurul lui 0, cuprinzand atat numere negative cat si pozitive
- daca se foloseste modificatorul UNSIGNED, valorile vor fi exclusiv pozitive. Plaja de valori se muta in domeniul pozitiv, valoarea minima admisa devenind 0, iar valoarea maxima admisa dublandu-se fata de cazul cu semn:

```
# definitia unei coloane ale carei valori se pot afla in domeniul (-128...127)
NrStudenti TINYINT
NrStudenti TINYINT SIGNED

# definitia unei coloane ale carei valori se pot afla in domeniul (0...255)
NrStudenti TINYINT UNSIGNED

# definitia unei coloane cu valori intregi intre -32768 si 32767
Abatere SMALLINT
```

4.2.2.2. „Latimea” unei coloane

In definitia unei coloane cu tip de date intreg se poate specifica, optional, „latimea” coloanei - numarul de caractere returnat de MySQL atunci cand se extrag date de pe acea coloana - prin plasarea sa intre paranteze imediat dupa tipul de date:

```
Interfon INT(3)
```

La extragerea datelor din tabela, MySQL va trimite clientului latimea impreuna cu datele, acesta din urma avand optiunea de a completa valoarea cu spatii pana la latimea coloanei. Trebuie inteles ca latimea nu afecteaza nici domeniul de valori posibile, nici spatiul de stocare ocupat: in exemplul de mai sus, valoarea 124 va fi afisata ca atare, valoarea 12345 va fi si ea memorata si afisata ca atare, iar valoarea 12 va fi probabil completata la stanga cu spatii inaintea afisarii.

Atunci cand nu specificam latimea unei coloane, MySQL alege automat aceasta valoare astfel incat sa cuprinda valoarea cea mai lunga a acelui tip de date (inclusiv eventualul semn minus). Spre exemplu, o definitie de coloana Nr TINIYINT va avea o latime implicita de 4 (3 cifre plus semn).

Daca in cazul unei coloane numerice este specificat si modificatorul ZEROFILL, valorile vor fi completate cu cifre 0 in loc de spatii atunci cand au numar de caractere mai mic decat latimea coloanei.

```
CREATE TABLE Apartamente (Nr TINYINT UNSIGNED, Interfon TINYINT(3) UNSIGNED ZEROFILL);
INSERT INTO Aparatmente VALUES(2,2);
SELECT * FROM Apartamente;
+------+----------+
| Nr   | Interfon |
+------+----------+
|    2 |      002 |
+------+----------+
```

Nota: *la aplicarea lui ZEROFILL MySQL adauga automat si modificatorul UNSIGNED pentru coloana in cauza, chiar daca acesta nu este specificat explicit!*

4.2.2.3. Modificatorul AUTO_INCREMENT

Adaugat unei coloane cu tip de date numeric, atributul suplimentar AUTO_INCREMENT face ca, la fiecare incercare de adaugare a unei inregistrari ce nu are valoare pentru acea coloana, sa fie generata si memorata in coloana o noua valoare numerica egala cu valoarea maxima existenta plus unu. In acest fel, fiecare inregistrare va avea o valoare unica pe coloana in cauza, fara ca programatorul sa trebuiasca sa ia vreo masura suplimentara in acest scop.

Exemplu:

```
CREATE TABLE Numere (ID TINYINT AUTO_INCREMENT UNIQUE, Nr INT)
INSERT INTO NumereUnice(Nr) VALUES(13);
INSERT INTO NumereUnice(Nr) VALUES(205);
INSERT INTO NumereUnice(Nr) VALUES(23);
```

ID	Nr
1	13
2	205
3	23

Nota: *o tabela SQL poate avea o singura coloana de tip AUTO_INCREMENT si acea coloana trebuie declarata ca index (vezi sectiunea despre indecsi).*

A nu se intelege ca valorile dintr-o coloana AUTO_INCREMENT vor fi intotdeauna consecutive! Intr-adevar, in etapa de introducere a inregistrarilor, valorile vor fi generate una dupa alta, insa, daca mai apoi se sterg inregistrari din tabela, rezulta „goluri” in valorile coloanei in cauza. Serverul nu va incerca sa umple aceste goluri, ci va genera intotdeauna valorile in continuare. Mai mult, chiar si atunci cand este stearsa inregistrarea ce contine valoarea maxima de pe coloana AUTO_INCREMENT, la urmatoarea operatie INSERT nu va fi refolosita valoarea stearsa, ci se va genera cea urmatoare. Aceasta deoarece urmatoarea valoare de AUTO_INCREMENT este un parametru memorat in meta-informatia tabelei – el poate fi vizualizat cu comanda SHOW TABLE STATUS.

4.2.3. Numere fractionare

4.2.3.1. Modalitati de reprezentare a numerelor fractionare

Numerele fractionare sunt cele care cuprind atat parte intreaga, cat si zecimale (ex: 53,214). Ele corespund conceptului de numar rational din matematica, insa nu toate numerele rationale pot fi memorate exact intr-o tabela SQL.

In matematica, un numar rational poate avea o reprezentare finita sau infinita. Spre exemplu, numarul 7/2 are o reprezentare numerica finita, si anume 3,5, pe cand numarul 10/3 are o reprezentare cu numar infinit de cifre, si anume 3.333...., notat prescurtat 3.(3). Un sistem de calcul poate memora numai un numar finit de digiti, de aceea numerele cu reprezentare infinita nu pot fi reprezentate exact, ci este necesara o ajustare a valorii, fiind memorat in baza de date cel mai apropiat numar cu reprezentare finita permis de precizia specificata in definitia coloanei (ex: 3,33 daca definitia coloanei permite maxim 2 zecimale).

Memorarea numerelor fractionare intr-un sistem de calcul se poate face in doua moduri:

- **in virgula fixa**. Aceasta presupune stabilirea de la bun inceput a numarului de cifre aflate in stanga si in dreapta separatorului zecimal (care este virgula, in sistem romanesc, si punctul in cel american). Numarul de cifre al partii intregi si numarul de zecimale fiind prestabilite, virgula are pozitie fixa in cadrul numarului
- **in virgula mobila**. Pozitia virgulei in cadrul numarului nu mai este prestabilita, ci poate varia in functie de valoarea memorata. Numarul total de cifre ce pot fi memorate ramane acelasi, insa prin schimbarea pozitiei virgulei pot fi obtinute numere de diferite anverguri si precizii.

Spre exemplu, daca dispunem de 5 cifre, putem reprezenta numere in mod diferit:

- in virgula fixa, suntem fortati sa stabilim de la bun inceput numarul de zecimale. Daca alegem 3, atunci numarul maxim ce va putea fi reprezentat este 99.999, iar precizia nu va depasi niciodata 3 zecimale
- in virgula mobila, cu 5 cifre putem reprezenta numarul 43768 (pozitia implicita a virgulei este in acest caz dupa ultima cifra), dar si numarul 4.3768 (mutand virgula imediat dupa prima cifra) sau numarul 437680000 (mutand virgula cu 4 pozitii la dreapta). Ceea ce am numit pana acum „mutarea virgulei” are ca echivalent matematic inmultirea cu 10 la puterea corespunzatoare numarului de pozitii: numarul 437680000 reprezinta numarul initial (43768) inmultit cu 10 la puterea a patra.

De mai sus se deduce ca, in cazul numerelor in virgula mobila, este necesar sa fie memorata, in afara de cifrele numarului, si pozitia virgulei (exponentul lui 10 din ultimul exemplu). De aceea un numar in virgula mobila are doua caracteristici principale:

- precizie – numarul total de cifre
- ordin de marime („scale” in engleza) – numarul de zecimale (cifrele din dreapta separatorului zecimal)

Exemplele de mai sus au fost date pentru numere in virgula mobila ce folosesc baza de numeratie 10. In sistemele de calcul se obisnuieste insa a se folosi baza 2 pentru repezentarea numerelor, ceea ce introduce complicatii suplimentare: doar o parte dintre numerele care au reprezentare finita in baza 10 vor avea reprezentare finita si in baza 2 (mai exact, numai cele care se pot scrie ca fractii ce au ca numitor o putere de 2).

De aceea, numere precum 0.1 in baza 10 au numar infinit de cifre in baza 2 si deci, la memorarea lor intr-un sistem de calcul, va fi stocata o valoare aproximativa (rotunjita).

In concluzie, memorarea in virgula fixa are avantajul stocarii valorii exacte a numerelor cu reprezentare finita, si a efectuarii rotunjirilor (aproximarilor) in felul in care am fost obisnuiti din baza de numeratie 10. Numerele in virgula mobila memoreaza valori aproximative, in schimb beneficiaza de o viteza de lucru mult mai mare, procesorul sistemului de calcul avand in general suport direct pentru operatii cu numere in virgula mobila.

4.2.3.2. Tipuri de date MySQL pentru numere in virgula fixa

Tip de date	Sinonime	Nr. oct.	Valori posibile
DECIMAL	DEC NUMERIC FIXED	1	In functie de parametrii specificati. Maxim 65 de cifre in total, din care maxim 30 de zecimale.

Exista un singur tip de date de acest fel, si anume DECIMAL. In definitia unei coloane de tip DECIMAL trebuie specificati doi parametri:

- numarul total de cifre admise pentru numerele de pe acea coloana (nu se iau in considerare separatorul zecimal si eventualul semn al numarului, ci doar cifrele ce compun partea intreaga si zecimalele)
- numarul de zecimale permise

Cei doi parametri se specifica separati prin virgula, imediat dupa tipul de date:

```
CREATE TABLE Produse (Denumire VARCHAR(100), Pret DECIMAL(5,2))
```

Cu aceasta instructiune, coloana Pret va permite numere a caror parte intreaga are 3 cifre si care au maxim doua zecimale. Plaja de valori admise se va intinde intre -999.99 si +999.99. Daca sql_mode nu specifica functionarea in modul strict, orice valoare introdusa ce depaseste acest interval va fi trunchiata la una dintre extreme (ex: incercand sa introducem valoarea 10000, va fi memorata in tabela valoarea 999.99). In mod strict, incercarea de introducere a unei valori in afara plajei admise se va solda cu o eroare.

4.2.3.3. Tipuri de date MySQL pentru numere in virgula mobila

Tip de date	Sinonime	Nr. oct.	Valori posibile, cu semn	Valori posibile, fara semn (unsigned)
FLOAT	-	4	-3.4×10^{38}...3.4×10^{38}	0...3.4×10^{38}
DOUBLE	DOUBLE PRECISION REAL (vezi nota)	8	-1.7×10^{308}...1.7×10^{308} (aprox)	0...$1.7 \times 3.4 \times 10^{308}$ (aprox.)

Nota: *REAL este implicit sinonim pentru DOUBLE; daca insa se foloseste in sql_mode optiunea REAL_AS_FLOAT, REAL devine sinonim pentru FLOAT.*

In definitia unei coloane de tip FLOAT sau DOUBLE pot fi specificate, ca si in cazul lui DECIMAL, numarul total de cifre si numarul de zecimale, cu aceeasi sintaxa:

```
CREATE TABLE Persoane (Nume VARCHAR(100), Greutate FLOAT(5,2))
```

Efectul va fi acelasi cu cel din cazul lui DECIMAL: plaja de valori posibile si precizia memorarii numerelor va fi redusa. Spre exemplu, daca in coloana Greutate de mai sus incercam sa introducem valoarea 1000 in modul non-strict, valoarea memorata va fi 999.99 si se va genera un warning. Daca incercam sa introducem valoarea 999.0052, nu va fi generata nicio eroare (deoarece numarul introdus se afla in interiorul plajei de valori posibile) insa valoarea va fi rotunjita la cel mai apropiat numar cu doua zecimale, si anume 999.01.

Chiar daca restrictiile ce pot fi impuse sunt asemanatoare cu cele de la DECIMAL, FLOAT si DOUBLE isi mentin proprietatile care tin de memorarea in virgula mobila. Iata cateva exemple de imprecizie a memorarii anumitor numere in virgula mobila:

```
CREATE TABLE `test` ( d decimal(8,6), f float(8,6))


INSERT INTO test VALUES(80.01,80.01),(56.01,56.01);
SELECT * FROM test;
+-----------+-----------+
| d         | f         |
+-----------+-----------+
| 80.010000 | 80.010002 |
| 56.010000 | 56.009998 |
+-----------+-----------+


ALTER TABLE test CHANGE d d BIGINT(10),CHANGE f f FLOAT(10,0);
TRUNCATE test;
INSERT INTO test VALUES(1234567890,1234567890);
SELECT * FROM test;
+------------+------------+
| d          | f          |
+------------+------------+
| 1234567890 | 1234567936 |
+------------+------------+
```

4.3. Tipuri de date temporale

4.3.1. Prezentare generala

Tip de date	Valori posibile
DATE	1000-01-01 ... 9999-12-31
DATETIME	1000-01-01 00:00:00 ... 9999-12-31 23:59:59
TIME	-838:59:59 ... 838:59:59
YEAR	1901 ... 2155 (pt 4 digiti) 1970 ... 2069 (pt 2 digiti)
TIMESTAMP	1970-01-01 00:00:01 ... 2038-01-09 03:14:07

Tipurile de date temporale sunt folosite pentru a reprezenta urmatoarele categorii de valori:

- **data calendaristica** – tipul de date DATE. Data este memorata si extrasa sub forma AAAA-LL-ZZ (A=an, L=luna, Z=zi). Exemplu: 2009-01-18

- **moment in timp (data calendaristica+ora)**, cu precizie de secunda
  - tipul de date DATETIME. Formatul este AAAA-LL-ZZ OO:MM:SS. (O=ora, M=minut, S=secunda)
  - tipul de date TIMESTAMP. Formatul este acelasi ca in cazul lui DATETIME, insa intr-o coloana de acest tip se poate memora automat timpul curent la fiecare operatie de INSERT sau UPDATE, folosind atributul ON_UPDATE_CURRENT_TIMESTAMP la definirea coloanei, iar in plus valoarea default a coloanei poate fi chiar data/ora curenta din momentul introducerii valorii, folosind clauza DEFAULT CURRENT_TIMESTAMP

- **interval de timp** – tipul de date TIME, care poate fi folosit pentru a reprezenta atat ora din zi, cat si intervale de timp care pot fi mai mari de 24h (spre exemplu, 35:12:41). Formatul este OO:MM:SS (sau OOO:MM:SS pt intervale de timp mari).

- **an** – tipul YEAR. Se poate specifica optional numarul de digiti folosit pentru reprezentare – 2 sau 4. (ex: YEAR(4)).

In instructiunile SQL, valorile constante pentru coloane cu tip de date temporal se specifica incadrate in apostroafe, cu exceptia tipului YEAR reprezentat ca intreg. Lunile sau zilele care au o singura cifra nu este obligatoriu sa aiba 0 in fata (ex: '2009-08-03' poate fi reprezentat si ca '2009-8-3').

```
INSERT INTO Date(D DATE, TS TIMESTAMP, T TIME, Y YEAR)
VALUES('2009-04-05', '2006-4-13 18:23:45', '104:56:89', 1986);
```

Fiecare dintre tipurile de date temporale dispune de o valoare „zero” a acelui tip de date(ex: 0000-00-00 pentru DATE, 00:00:00 pentru DATETIME etc.). Valoarea in cauza are urmatoarele caracteristici:

- in modul non-strict, la incercarea de introducere a unei valori invalide pe o coloana cu tip de date temporal, MySQL va introduce automat valoarea „zero” a tipului de date, cu semnificatia „data invalida”
- prezenta valorilor temporale „zero” (care nu sunt, la randul lor, valori valide) poate fi interzisa complet daca sql_mode specifica modul strict si in plus contine optiunea NO_ZERO_DATE

4.3.2. Particularitati ale tipului de date TIMESTAMP

O coloana de tip TIMESTAMP memoreaza momentele de timp sub forma unui intreg ce reprezinta numarul de secunde scurs de la 1 ianuarie 1970 (asa-numitul „Unix time”). Aceasta forma de reprezentare face ca TIMESTAMP sa poata memora valori intre 1 ian 1970 si 19 ian 2038.

Coloanele de tip TIMESTAMP au doua particularitati utile:

- in definitia coloanei se poate folosi ca valoare default CURRENT_TIMESTAMP, cu efectul ca, la introducerea unei noi inregistrari, cand se omite coloana in cauza in instructiunea INSERT, va fi introdusa automat pe coloana valoarea corespunzatoare momentului de timp cand a fost rulata instructiunea INSERT.
- atunci cand in definitia unei coloane TIMESTAMP nu se specifica NOT NULL, coloana NU mai permite valoarea NULL (cum se intampla de obicei), iar valoarea default va fi CURRENT_TIMESTAMP. La

incercarea de a introduce NULL pe coloana va fi introdus timestamp-ul curent. Daca dorim ca coloana sa permita NULL trebuie sa specificam acest lucru explicit:

```
SET sql_mode='traditional';
CREATE TABLE t(t1 TIMESTAMP, t2 TIMESTAMP NULL);
DESCRIBE t;
INSERT INTO t VALUES(NULL,NULL);
SELECT * FROM t;
+---------------------+------+
| t1                  | t2   |
+---------------------+------+
| 2009-08-03 13:38:25 | NULL |
+---------------------+------+
```

- o coloana TIMESTAMP pot avea ca modificator suplimentar ON UPDATE CURRENT_TIMESTAMP. Acesta are ca efect memorarea in acea coloana a momentului de timp al ultimei modificari – si aceasta la fiecare modificare adusa *inregistrarii* in cauza! (nota: setarea valorii deja existente pe o coloana nu se considera modificare)

```
CREATE TABLE t(i INT, t1 TIMESTAMP ON UPDATE CURRENT_TIMESTAMP);
INSERT INTO t VALUES(1, NULL);
SELECT * FROM t;
+------+---------------------+
| i    | t1                  |
+------+---------------------+
|    1 | 2009-08-03 13:42:50 |
+------+---------------------+
UPDATE t SET i=5 WHERE i=1;
SELECT * FROM t;
+------+---------------------+
| i    | t1                  |
+------+---------------------+
|    5 | 2009-08-03 13:44:08 |
+------+---------------------+
```

Proprietatile suplimentare ale unei coloane de tip TIMESTAMP fac subiectul urmatoarelor restrictii:

- intr-o tabela MySQL pot exista mai multe coloane de tip TIMESTAMP, insa numai una dintre ele poate avea modificatorul ON UPDATE CURRENT_TIMESTAMP
- intr-o tabela cu mai multe coloane TIMESTAMP, nu se poate folosi DEFAULT CURRENT_TIMESTAMP pentru o coloana si ON UPDATE CURRENT_TIMESTAMP pentru alta
- daca la crearea unei tabele cu coloane TIMESTAMP nu se specifica nici DEFAULT CURRENT_TIMESTAMP, nici ON UPDATE CURRENT_TIMESTAMP, MySQL le va atribui pe amandoua primei coloane de tip TIMESTAMP

4.4. Siruri

4.4.1. Tipuri de siruri

MySQL ofera utilizatorului posibilitatea de a lucra cu doua tipuri de siruri:

- **siruri de octeti** – sunt simple succesiuni de numere. Compararea a doua astfel de siruri se efectueaza numeric, octet cu octet
- **siruri de caractere**. Un sir de caractere difera fundamental de unul de octeti prin urmatoarele aspecte:
  - un caracter poate fi reprezentat pe unul sau mai multi octeti (spre exemplu, un sir de 3 caractere poate avea chiar 9 sau mai multi octeti). Ca o consecinta, compararea a doua siruri de caractere nu mai poate fi facuta comparand octetii componenti unul cate unul, ci trebuie tinut cont de caracterele pe care le reprezinta diversele grupuri de octeti

    Exemplu: sirul de caractere Ţâr care in reprezentare UTF8 are 5 octeti:

Ţ		â		r
197	162	195	162	114

  - un sir de caractere are asociate doua proprietati care nu se regasesc la sirurile de octeti: character set si collation. Prima stabileste setul de caractere ce pot fi folosite in cadrul sirului, iar cea de-a doua modalitatea de comparare a caracterelor din acel set. Ambele vor fi detaliate mai jos

4.4.2. Siruri de octeti

Tip de date	Spatiu ocupat	Nr. maxim de octeti
BINARY(N)	N octeti	N (N<=255)
VARBINARY(N)	Nr. de octeti al valorii + 1 sau 2 pt lungime	N (N<=65535)
TINYBLOB	Nr. de octeti al valorii + 1 octet pt lungime	255
BLOB	Nr. de octeti al valorii + 2 octet pt lungime	65535
MEDIUMBLOB	Nr. de octeti al valorii + 3 octet pt lungime	16,777,215 (16M)
LONGBLOB	Nr. de octeti al valorii + 4 octet pt lungime	4,294,967,295 (4G)

Tipurile de date din tabelul alaturat sunt folosite pentru a memora siruri de octeti. Ele au urmatoarele caracteristici:

- tipul BINARY memoreaza siruri de lungime fixa, specificata in definitia coloanei. Valorile cu lungime mai mica decat cea specificata sunt completate cu spatii, astfel incat, chiar daca introducem pe o astfel de coloana valori de diferite lungimi, in final toate valorile vor ocupa spatiu de stocare egal. La extragerea unei astfel de valori din tabela, spatiile din final sunt eliminate; de aceea, daca valoarea originala se incheia cu unul sau mai multe spatii, la extragere ea nu le va mai avea
- tipul VARBINARY introduce o alta abordare a memorarii sirurilor de octeti: in definitia coloanei se specifica o lungime care reprezeinta numarul maxim de octeti admis pentru valorile coloanei, insa pentru fiecare valoare se memoreaza in tabela strict valorile introduse plus unul sau doi octeti pentru lungime. Astfel, tipul de date VARBINARY este mai eficient din punct de vedere al spatiului ocupat. In plus, nu mai este o problema daca valorile introduse pe coloana se termina cu spatii – acestea vor fi pastrate

– tipurile de date BLOB (Binary Large Object) sunt tot tipuri de date ce accepta valori cu lungime variabila – asemanatoare cu VARBINARY, dar de anvergura mult mai mare. In aceste conditii, fiecare valoare memorata intr-o coloana BLOB va ocupa exact numarul de octeti al valorii plus 1-4 octeti pentru lungime.

4.4.3. Siruri de caractere

4.4.3.1. Despre character set

Scrierile existente in diversele limbi folosesc diferite caractere posibile. Spre exemplu, engleza americana foloseste doar alfabetul standard (a-z), limba romana romana adauga diacriticele (Ş, Ţ, ...), poloneza foloseste litere precum Ł sau Ś etc. Fiecare sir de caractere are atasat un singur set de caractere, el fiind astfel constrans sa foloseasca doar caractere din acel set.

Un set de caractere (character set) reprezinta o lista de caractere posibile impreuna cu codurile numerice asignate lor. Fiecare caracter este pus in corespondenta cu un cod numeric. Codurile pot diferi de la un set de caractere la altul; in plus, exista coduri care in unele seturi de caractere au semnificatie si in altele nu. Cel mai cunoscut set de caractere este probabil ASCII, unde, spre exemplu, litera A are codul 65, B codul 66 etc., dar care, fiind un set de caractere pentru limba engleza americana, nu contine caracterele romanesti cu diacritice.

Unele seturi de caractere au un numar de elemente suficient de mic astfel incat fiecare caractere sa poata fi reprezentat pe un singur octet (ex: ASCII), insa in cazul altora fiecare caracter poate fi memorat pe 2 sau mai multi octeti. In plus, numarul de octeti pe care este memorat fiecare caracter poate fi fix sau variabil: spre exemplu, in UCS2 fiecare caracter ocupa cate 2 octeti, pe cand in UTF8 un caracter poate ocupa intre 1 si 4 octeti.

Set de caractere	Descriere
ASCII	Set de caractere folosit in engleza americana; fiecare caracter poate fi reprezentat pe 7 biti (codurile au valori <128)
ISO-8859-2 (numit si latin2)	Character set ce contine caracterele folosite in zona central si est europeana (inclusiv Romania)
UCS2	Character set ce contine majoritatea caracterelor Unicode, codul fiecarui caracter avand 2 octeti
UTF8	Character set pentru Unicode, cu fiecare caracter ocupand intre 1 si 4 octeti

In MySQL, lista seturilor de caractere disponibile poate fi vizualizata cu comanda SHOW CHARACTER SET.

Nota: *CHARACTER SET si CHARSET sunt sinonime si pot fi folosite interschimbabil oriunde apar (in instructiuni SHOW, in definitii de baza de date/tabela/coloana, etc.). Ex: SHOW CHARSET.*

```
mysql> SHOW CHARSET;
+----------+-----------------------------+---------------------+--------+
| Charset  | Description                 | Default collation   | Maxlen |
+----------+-----------------------------+---------------------+--------+
[...output omis...]
```

```
| latin2    | ISO 8859-2 Central European | latin2_general_ci    |      1 |
| swe7      | 7bit Swedish                | swe7_swedish_ci      |      1 |
| ascii     | US ASCII                    | ascii_general_ci     |      1 |
[...output omis...]
| utf8      | UTF-8 Unicode               | utf8_general_ci      |      3 |
[...output omis...]
```

Lista poate fi filtrata folosind clauza LIKE. Pentru fiecare character set este prezentat collation-ul implicit si numarul maxim de octeti pe care il poate ocupa un caracter.

4.4.3.2. Despre collation

Collation-ul unui sir de caractere reprezinta setul de reguli ce specifica in ce fel se compara unul cu altul caracterele lui componente:

- literele mici si mari pot fi considerate sau nu echivalente. Daca sunt echivalente, spunem ca collation-ul este „case-insensitive”, in caz contrar este „case sensitive”. Astfel, pentru un sir de caractere cu collation de tip case-insensitive, in MySQL expresia 'A' = 'a' va avea ca rezultat 1.
- putem considera literele cu sau fara accente sau diacritice ca fiind sau nu echivalente. Spre exemplu, in romana literele a si â pot fi considerate sau nu egale, in franceza e si é pot fi de asemenea considerate echivalente etc.

Dat fiind faptul ca exista mai multe modalitati de a compara caracterele aceluiasi set, este firesc ca in general pentru acelasi character sa existe mai multe collation-uri posibile. Spre exemplu, pentru setul de caractere latin2, ce reuneste caractere folosite in mai multe limbi din zona Europei centrale si de est, exista un collation generic si inca cateva pentru limba ceha, maghiara, croata etc. De remarcat insa ca un sir de caractere nu poate avea decat un singur character set si un singur collation.

Nota: *nu este posibil ca doua seturi de caractere diferite sa aiba acelasi collation. Fiecare set de caractere are propria lista de collation-uri disponibile.*

In MySQL denumirea fiecarui collation incepe cu numele setului de caractere urmat de _ si se termina cu tipul de collation precedat de _ . Acesta din urma poate fi:

- **_ci** – indica un collation case-insensitive
- **_cs** – indica un collation case sensitive
- **_bin** – indica un collation de tip binary. Intr-un astfel de collation caracterele sunt comparate exclusiv pe baza codului lor. Dat fiind faptul ca codurile de caractere sunt unice, toate caracterele vor fi considerate diferite intre ele, si implicit comparatia va fi una case-sensitive.

Nota: *diferenta intre un collation case-sensitive si unul binary este ca, in cazul celui case sensitive, desi literele mici si literele mari sunt diferite, pot exista totusi relatii de egalitate intre litere cu sau fara accent sau diacritice; in cazul unui collation binary nu se mai intampla acest lucru.*

In MySQL, vizualizarea listei de collation-uri disponibile pentru diversele seturi de caractere poate fi realizata cu comanda SHOW COLLATION urmata de o posibila clauza LIKE:

```
# collation-urile disponibile pentru setul de caractere latin2
mysql> SHOW COLLATION LIKE 'latin2%';
+--------------------+---------+----+---------+----------+---------+
| Collation          | Charset | Id | Default | Compiled | Sortlen |
+--------------------+---------+----+---------+----------+---------+
| latin2_czech_cs    | latin2  |  2 |         | Yes      |       4 |
| latin2_general_ci  | latin2  |  9 | Yes     | Yes      |       1 |
| latin2_hungarian_ci| latin2  | 21 |         | Yes      |       1 |
| latin2_croatian_ci | latin2  | 27 |         | Yes      |       1 |
| latin2_bin         | latin2  | 77 |         | Yes      |       1 |
+--------------------+---------+----+---------+----------+---------+
```

Precum observam si in output-ul de mai sus, dintre collation-urile disponibile pentru un character set, unul este marcat ca default si va fi cel folosit implicit atunci cand se alege acel set de caractere fara a se specifica si collation-ul dorit.

Orice sir de caractere MySQL are un character set si un collation. Faptul ca pana acum am putut scrie definitii de coloane care nu specificau aceste doua proprietati se datoreaza faptului ca exista valori implicite care actioneaza automat. Valorile implicite pot fi definite independent la nivel de server, baza de date si tabela.

4.4.3.3 Tipuri de date MySQL pentru siruri de caractere

Tip de date	Numar maxim de caractere
CHAR(N)	N (N<=255)
VARCHAR(N)	N (N<=65535)
TINYTEXT	255
TEXT	65535
MEDIUMTEXT	16,777,215 (16M)
LONGTEXT	4,294,967,295 (4G)

Ca si in cazul tipurilor de date pentru siruri de octeti, dispunem de doua categorii de tipuri de date:

- cele de lungime fixa, in care numarul de caractere al sirului este stabilit in definitia coloanei si orice valoare va ocupa intotdeauna acelasi spatiu de stocare, fiind completata cu spatii daca este prea scurta. Este cazul tipului de date CHAR
- cele de lungime variabila, in cazul carora valorile memorate pe coloana vor ocupa spatiul corespunzator numarului lor de caractere plus cativa octeti suplimentari pentru memorarea lungimii. Tipurile de date corespunzatoare sunt VARCHAR si tipurile de date TEXT.

Nota: *subliniem inca o data faptul ca nu avem o corespundenta 1 la 1 intre octeti si caractere: pentru o coloana definita ca CHAR(3), o singura valoare poate ocupa chiar si 9 octeti!*

La definirea unei coloane de tip sir de caractere se specifica lungimea, character set-ul si collation-ul dupa cum urmeaza:

```
CREATE TABLE Persoane(Nume VARCHAR(100) CHARACTER SET utf8 COLLATE utf8_romanian_ci);
```

- cele folosite pentru memorarea de siruri de caractere (*CHAR, *TEXT). Pentru coloanele de acest fel se poate specifica un set de caractere si un mod de comparare si ordonare (asa-numitul *collation*)
- cele folosite pentru memorarea de siruri de octeti (*BINARY, *BLOB). Datele din astfel de campuri vor fi comparate strict pe baza valorii octetilor componenti

4.4.4. Tipuri de date enumerate

Tipurile de date enumerate sunt folosite atunci cand o coloana poate lua valori numai dintr-o multime fixa de valori discrete (ex: zilele saptamanii, notele muzicale, sexul unei persoane etc). MySQL ofera doua tipuri de date in acest scop:

- **ENUM** – o coloana cu acest tip de date poate contine numai valori dintr-o multime prestabilita, specificata in definitia coloanei. Fiecare inregistrare va putea avea pe coloana in cauza numai una dintre valorile din lista, sau NULL daca acest lucru este permis. Pot fi definite maxim 65535 valori distincte.

```
CREATE TABLE Disponibilitati(ID INT(10), Perioada ENUM('dimineata', 'pranz', 'seara'))
INSERT INTO Disponibilitati VALUES(3, 'pranz')
```

Daca coloana in cauza permite NULL, valoarea default a coloanei va fi NULL. In caz contrar, daca nu se specifica explicit valoarea default la crearea coloanei, in modul non-strict va fi folosit implicit primul element din lista de valori permise.

Nota: *Atunci cand pe o coloana de tip ENUM se introduce o valoare invalida (alta decat cele din lista permisa) si serverul functioneaza in modul non-strict, valoarea memorata in baza de date va fi '' (sirul vid) pentru a indica eroarea. Aceastei valori particulare ii corespunde codul 0 (vezi explicatie mai jos)*

- **SET** – asemanator cu ENUM, insa permite ca valoarea coloanei pentru o anumita inregistrare sa fie o combinatie a valorilor din set (asemanator cu selectia multipla de la elementele de formular HTML). Pot fi specificate maxim 64 de valori.

```
CREATE TABLE Disponibilitati(ID INT(10), Perioada SET('dimineata', 'pranz', 'seara'))
INSERT INTO Disponibilitati VALUES(4, 'pranz,seara')
```

Nota: *valoarea '' (sirul vid) este una valida in cazul tipului de date SET, putand fi memorata explicit in cadrul coloanei.*

Diferenta principala intre cele doua tipuri de date de mai sus este ca, in cazul primului, pe acea coloana nu se poate memora decat unul dintre intervalele de disponiblitate, pe cand pentru cel de-al doilea este posibila, de exemplu, memorarea combinatiei dimineata+pranz.

In instructiunile MySQL, valorile de tip ENUM sau SET sunt reprezentate ca stringuri, incadrate intre apostroafe. Stocarea lor in baza de date se face insa eficient: fiecare valoare a unui tip de date enumerat este

memorata de catre MySQL sub forma unui intreg corespunzator. Valorile sunt numerotate de la 1; valoarea 0 este rezervata pentru a indica incercarea de introducere a unei valori invalide pe coloana in cauza. Valoarea intreaga corespunzatoare conteaza atunci cand coloanele de tip ENUM/SET intervin in expresii in contexte numerice:

```
CREATE TABLE Disponibilitati(ID INT(10), Perioada ENUM('dimineata', 'pranz', 'seara'))
INSERT INTO Disponibilitati VALUES(15, 'pranz')
SELECT Perioada+1 FROM Disponibilitati;
+-------------+
| Perioada+1  |
+-------------+
|    3        |
+-------------+
# a fost folosit intregul 2 ce corespunde valorii 'pranz'
```

4.5. BIBLIOGRAFIE

- MySQL 5 Certification Study Guide – Cap. 5 „Data Types”
- Numere in virgula mobila: http://en.wikipedia.org/wiki/Floating-point
- Tipuri de date: http://dev.mysql.com/doc/refman/5.1/en/data-types.html
- MySQL Character Sets and Collations: http://dev.mysql.com/doc/refman/5.1/en/charset-general.html
- MySQL Collation Charts: http://www.collation-charts.org/mysql60/mysql604.latin2_general_ci.html

Conventie

-referitoare la conditiile de desfasurare a examenului intermediar-

1. **Caracteristici generale ale examenului Intermediar teoretic :**

   a. are ca scop verificarea cunostintelor acumulate in prima parte a fiecarui modul
   b. este compus din intrebari selectate din examenele partiale 1-4
   c. este de tip test grila si are in medie 15-20 de intrebari cu raspuns unic sau multiplu
   d. dureaza 30 de minute
   e. poate fi sustinut numai la sediul academiei „InfoAcademy" in prezenta instructorului sau a unui supraveghetor delegat de InfoAcademy
   f. este accesibil pe http://www.infoacademy.net → Login → Student Home → NumeClasa → TakeAssesment
   g. se considera promovat daca nota obtinuta este cel putin egala cu 75% la prima încercare, sau cel puțin egala cu 70% la a doua incercare
   h. poate fi sustinut de maxim 2 ori, la interval de 1 saptamana intre cele 2 incercari

2. **Data la care se sustine examenul intermediar**

   a. examenul intermediar (prima incercare) se sustine in cadrul celei de-a 5-a sedinte de curs
   b. eventuala repetare a examenului intermediar (cea de-a doua incercare) va avea loc in urmatoarea zi de vineri de dupa prima sustinere a examenului, intre orele 8-11,30.

3. **Conditii de participare la examenul intermediar teoretic :**

   Poate participa la examenul intermediar teoretic orice student Infoacademy care:
   a. a promovat examenele partiale 1-4 cu cel putin 75%
   b. a achitat obligatiile financiare convenite la inscriere sau se obliga sa le achite pana la data examenului intermediar inclusiv
   c. daca exista obligatii financiare restante, acestea pot fi achitate cu 10-15 minute inaintea examenului intermediar
   d. are asupra sa un act cu fotografie care ii atesta identitatea
   e. se obliga sa respecte conditiile de desfasurare a examenului intermediar incluse in aceasta Conventie

4. **In timpul desfasurarii examenului intermediar, studentul se obliga :**

   a. sa nu comunice direct sau prin dispozitive electronice cu alte persoane
   b. sa nu salveze pe nici un fel de suport intrebarile/imaginile/paginile incluse in examenul sustinut
   c. sa nu incerce sa transmita prin Intranet/Extranet/Internet continutul intrebarilor, imagini sau pagini web continute in examen sau aflate pe calculatorul de pe care sustine examenul
   d. sa nu utilizeze materiale scrise/inregistrate pentru a afla raspunusuri la intrebarile incluse in examen
   e. sa nu utilizeze dispozitive de calcul (de tip calculatorul inclus in telefon, ceas, sistemul de operare, etc..) pentru a efectua operatii de orice tip
   f. sa nu aiba asupra sa dispozitive de inregistrare audio/video (de tip telefon celular,etc..)
   g. sa nu afecteze prin tinuta sau comportament desfasurarea examenului final

5. **Dupa terminarea examenului intermediar, studentul :**

   a. iese din cont (Logout)
   b. preda ciorna
   c. paraseste sala de examen

6. **Este obligatoriu examenul intermediar?**

   a. conform contractului examenul intermediar este obligatoriu.
   b. cei care nu sustin examenul intermediar si nu achita rata a 2-a (in cazul in care au platit partial cursul) nu mai sunt eligibili pentru participarea la cursuri/laboratoare si examen final.

7. **Incalcarea Conventiei**

   a. incalcarea prevederilor articolului 4 a,b,c,d,e,f duce la eliminarea din examenul intermediar si din academie cu pierderea oricaror drepturi asupra modulului al carui examen intermediar se sustine.
   b. incalcarea prevederilor articolului 4.g duce la eliminarea studentului din examenul intermediar cu pierderea uneia din cele doua sanse de sustinere a acestuia

Participarea la examenul intermediar echivaleaza cu acceptarea deplina a acestei conventii, acordul de a o respecta întocmai şi supunerea la sanctiunile prevazute în cazul nerespectarii ei.