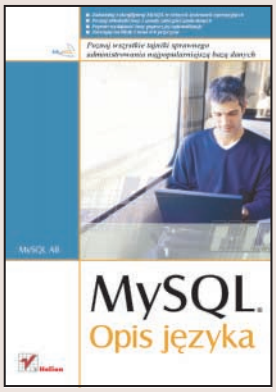
MySQL. Opis języka



Autor: Paul Dubois
Tłumaczenie: Paweł Gonera, Anna Zawadzka
ISBN: 83-7361-688-8
Tytuł oryginału: MySQL Language Reference
Format: B5, stron: 472

MySQL to najpopularniejsza na świecie baza danych dostępna na licencji open source. Dzięki swojej wydajności, szybkości działania, stabilności i rozbudowanym mechanizmom zabezpieczeń jest wykorzystywana niemal do wszystkich zadań, do jakich może być potrzebna baza danych. Stanowi zaplecze dla wielu aplikacji korporacyjnych, witryn WWW i sklepów internetowych. Bogactwo funkcji i możliwości to ogromna zaleta środowiska MySQL, jednakże zapamiętanie składni, parametrów i sposobów stosowania wszystkich poleceń i rozszerzeń jest praktycznie niemożliwe. Każdy, kto chce biegle posługiwać się tą bazą, powinien mieć pod ręką materiały opisujące stosowany w niej język SQL.

Książka „MySQL. Opis języka" to najlepsze źródło takich informacji. Stworzona, zredagowana i sprawdzona przez pracowników działów pomocy i produkcji firmy MySQL AB publikacja zawiera wszystko, co może być potrzebne użytkownikowi bazy danych MySQL. Przedstawia zarówno zagadnienia podstawowe, takie jak: struktura języka, typy danych i zasady konstruowania zapytań, jak i tematy zaawansowane: transakcje, replikację oraz administrowanie bazą. Szczegółowo opisane są również wszystkie zagadnienia związane z zaimplementowanym w niej językiem SQL.

- Podstawowe informacje o MySQL
- Język SQL – instrukcje, typy danych, funkcje i operatory
- Zapytania
- Definiowanie danych
- Obsługa transakcji
- Zarządzanie kontami użytkowników
- Konserwacja tabel
- Replikacja baz danych
- Rozszerzenia MySQL dla systemów GIS
- Procedury składowane
- Obsługa błędów

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Rozdział 4.
Typy kolumn

W MySQL dostępnych jest wiele typów danych, które można pogrupować w kilku podstawowych kategoriach, takich jak typy liczbowe, typy związane z datą i czasem oraz typy łańcuchowe. Na początku tego rozdziału przedstawiony zostanie przegląd wspomnianych przed chwilą typów kolumn, po czym nastąpi bardziej szczegółowy opis ich właściwości z podziałem na poszczególne kategorie i wreszcie zamieszczone zostanie podsumowanie informacji dotyczących wymagań co do rozmiaru pamięci potrzebnej do przechowania danej kolumny. Sam przegląd jest z zamierzenia krótki. Dodatkowe informacje o poszczególnych typach kolumn, na przykład dopuszczalny format, w którym można deklarować wartości, zostały uwzględnione w szczegółowych omówieniach.

Od wersji 4.1 MySQL rozpoznaje rozszerzenia do obsługi danych przestrzennych. Informacje o typach przestrzennych znajdują się w rozdziale 7. tego podręcznika.

W części opisu typów kolumn zastosowane zostały następujące reguły:

- *M*

  Wskazuje maksymalny rozmiar wyświetlania liczby lub łańcucha. Maksymalnym dopuszczalnym rozmiarem wyświetlania jest 255.

- *D*

  Dotyczy typów zmiennoprzecinkowych i stałoprzecinkowych. Określa liczbę cyfr po kropce dziesiętnej. Maksymalna dopuszczalna wartość wynosi 30, ale nie powinna przekraczać *M*-2.

- Nawiasy kwadratowe ([ ]) identyfikują opcjonalne fragmenty deklaracji typu.

4.1. Przegląd typów dla kolumny

4.1.1. Omówienie typów liczbowych

Poniżej znajduje się przegląd liczbowych typów dla kolumn. Bardziej szczegółowe informacje można znaleźć w podrozdziale 4.2. Wymogi dotyczące rozmiaru pamięci potrzebnej do przechowywania kolumn znajdują się w podrozdziale 4.5.

Jeśli dla kolumny liczbowej zostanie określony atrybut `ZEROFILL`, MySQL automatycznie doda do kolumny atrybut `UNISIGNED`.

Należy uważać podczas odejmowania dwóch wartości liczbowych, z których jedna ma typ `UNISIGNED`. Uzyskany wynik będzie również tego typu! — patrz podrozdział 5.7.

- `TINYINT[(M)] [UNSIGNED] [ZEROFILL]`

  Bardzo mała liczba całkowita. Dla liczb ze znakiem zakres wynosi od `-128` do `127`. W przypadku liczb bez znaku zakres obejmuje wartości od `0` do `255`.

- `BIT`, `BOOL`, `BOOLEAN`

  To są synonimy `TINYINT(1)`. Synonim `BOOLEAN` został dodany w MySQL 4.1.0. Wartość zerowa to inaczej fałsz. Wartości niezerowe traktowane są jako prawda.

  W przyszłości wprowadzona zostanie pełna, zgodna ze standardem SQL obsługa typu boolowskiego.

- `SMALLINT[(M)] [UNSIGNED] [ZEROFILL]`

  Mała liczba całkowita. Dla liczb ze znakiem zakres wynosi od `-32768` do `32767`. W przypadku liczb bez znaku zakres obejmuje wartości od `0` do `65535`.

- `MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]`

  Średnia liczba całkowita. Dla liczb ze znakiem zakres wynosi od `-8388608` do `8388607`. W przypadku liczb bez znaku zakres obejmuje wartości od `0` do `16777215`.

- `INT[(M)] [UNSIGNED] [ZEROFILL]`

  Standardowa liczba całkowita. Dla liczb ze znakiem zakres wynosi od `-2147483648` do `2147483647`. W przypadku liczb bez znaku zakres obejmuje wartości od `0` do `4294967295`.

- `INTEGER[(M)] [UNSIGNED] [ZEROFILL]`

  Jest to synonim dla typu `INT`.

- `BIGINT[(M)] [UNSIGNED] [ZEROFILL]`

  Duża liczba całkowita. Dla liczb ze znakiem zakres wynosi od `-9223372036854775808` do `9223372036854775807`. W przypadku liczb bez znaku zakres obejmuje wartości od `0` do `18446744073709551615`.

Oto kilka spraw, o których należy wiedzieć odnośnie kolumn typu BIGINT:

- Wszystkie obliczenia wykonywane są przy użyciu wartości typu BIGINT lub DOUBLE ze znakiem. Oznacza to, że dużych liczb całkowitych bez znaku, większych niż 9223372036854775807 (63 bity), można używać tylko z funkcjami bitowymi! Niezastosowanie się do tej zasady może doprowadzić do uzyskania w wyniku liczby, której kilka ostatnich cyfr jest błędnych z powodu błędów w zaokrąglaniu podczas konwersji wartości BIGINT na DOUBLE.
- MySQL 4.0 potrafi obsługiwać BIGINT w następujących przypadkach:
  - Podczas wykorzystywania liczb całkowitych do przechowywania dużych wartości bez znaku w kolumnie typu BIGINT.
  - W MIN(*nazwa_kol*) lub MAX(*nazwa_kol*), gdzie *nazwa_kol* odnosi się do kolumny BIGINT.
  - Podczas korzystania z operatorów (między innymi +, -, *), gdy oba argumenty są liczbami całkowitymi.
- Zawsze istnieje możliwość zapisania w kolumnie BIGINT dokładnej wartości liczby całkowitej. Wystarczy zapisać ją przy użyciu łańcucha. W takim przypadku zostanie dokonana konwersja łańcucha na liczbę; konwersja nie będzie wymagać pośredniego etapu przekształcającego liczbę na jej odpowiednik o podwójnej precyzji.
- Podczas korzystania z operatorów +, - i * stosowana jest arytmetyka BIGINT, jeśli oba argumenty mają wartość liczby całkowitej! To oznacza, że pomnożenie dwóch dużych liczb całkowitych (lub wyników funkcji, która zwraca liczby całkowite) może spowodować nieoczekiwane rezultaty, jeśli wynik przekroczy liczbę 9223372036854775807.

- FLOAT(*p*) [UNSIGNED] [ZEROFILL]

  Liczba zmiennoprzecinkowa. Parametr *p* oznacza precyzję i może przyjmować wartość od 0 do 14 dla liczb zmiennoprzecinkowych o dokładności do jednego znaku i od 25 do 53 dla liczb zmiennoprzecinkowych o dokładności do dwóch znaków. To przypomina opisane poniżej typy FLOAT i DOUBLE. Typ FLOAT(*p*) ma ten sam zakres, jaki posiadają odpowiadające mu typy FLOAT i DOUBLE, ale w jego przypadku niezdefiniowany jest maksymalny rozmiar wyświetlania i liczba miejsc po przecinku.

  Od wersji 3.23 MySQL jest to prawdziwa wartość zmiennoprzecinkowa. W wersjach wcześniejszych liczba typu FLOAT(*p*) była zawsze liczbą o dwóch miejscach po przecinku.

  Ta składnia została dodana w celu zachowania zgodności ze standardem ODBC.

  Stosowanie typu FLOAT może spowodować wystąpienie pewnych nieoczekiwanych problemów, ponieważ wszystkie obliczenia w MySQL są wykonywane z podwójną precyzją patrz punkt A.1.7).

- FLOAT[(*M*, *D*)] [UNSIGNED] [ZEROFILL]

Mała liczba zmiennoprzecinkowa (o dokładności do jednego znaku). Liczba ta może przyjąć wartości: od -3.402823466E+38 do -1.175494351E-38, 0, i od 1.175494351E-38 do 3.402823466E+38. Jeśli podany jest atrybut UNISIGNED, niedozwolone są wartości ujemne. Parametr *M* definiuje maksymalną szerokość wyświetlanej liczby, a *D* liczbę miejsc po przecinku. Typ FLOAT bez argumentów lub typ FLOAT(*p*) (gdzie *p* jest liczbą z zakresu od 0 do 24) oznacza liczbę zmiennoprzecinkową o pojedynczej precyzji.

♦ DOUBLE[(*M*, *D*)] [UNSIGNED] [ZEROFILL]

Standardowa liczba zmiennoprzecinkowa (o dokładności do dwóch znaków). Liczba ta może przyjąć wartości: od -1.7976931348623157E+308 do -2.2250738585072014E-308, 0, i od 2.2250738585072014E-308 do 1.7976931348623157E+308. Jeśli podany jest atrybut UNISIGNED, wartości ujemne są niedozwolone. Parametr *M* definiuje maksymalną szerokość wyświetlanej liczby, a *D* — liczbę miejsc po przecinku. Typ DOUBLE bez argumentów lub FLOAT(*p*) (gdzie *p* jest liczbą z zakresu od 25 do 53) oznacza liczbę zmiennoprzecinkową o podwójnej precyzji.

♦ DOUBLE PRECISION[(*M*, *D*)] [UNSIGNED] [ZEROFILL]

REAL[(*M*, *D*)] [UNSIGNED] [ZEROFILL]

Są to synonimy DOUBLE. Wyjątek: Jeśli dla parametru sql_mode serwera włączono opcję REAL_AS_FLOAT, synonimem FLOAT jest REAL, a nie DOUBLE.

♦ DECIMAL[(*M*[, *D*])] [UNSIGNED] [ZEROFILL]

Niezapakowana liczba stałoprzecinkowa. Zachowanie kolumny tego typu przypomina kolumnę typu CHAR. „Niezapakowana" oznacza, że liczba jest przechowywana w postaci łańcucha przy użyciu jednego znaku na każdą cyfrę wartości. Parametr *M* definiuje maksymalną liczbę cyfr, a *D* — liczbę miejsc po przecinku. W parametrze *M* nie są brane pod uwagę: kropka dziesiętna (odpowiadająca w naszej notacji przecinkowi) i (dla liczb ujemnych) znak -, choć jest dla nich zarezerwowane miejsce. Wartości nie mają miejsc po przecinku lub części ułamkowych, jeśli *D* jest równe 0. Maksymalny zakres wartości typu DECIMAL jest taki sam, jak dla typu DOUBLE, ale rzeczywisty zakres danej kolumny DECIMAL może być ograniczony przez wybór argumentów *M* i *D*. Jeśli podany jest atrybut UNISIGNED, wartości ujemne są niedozwolone.

W przypadku braku argumentu *D*, domyślną wartością jest 0. Jeśli pominięty jest parametr *M*, domyślną wartością jest 10.

W wersjach poprzedzających MySQL 3.23 argument *M* musi być wystarczająco duży, by zawierał obszar potrzebny dla znaku i kropki dziesiętnej.

♦ DEC[(*M*[,*D*])] [UNSIGNED] [ZEROFILL]

NUMERIC[(*M*[,*D*])] [UNSIGNED] [ZEROFILL]

FIXED[(*M*[, *D*])] [UNSIGNED] [ZEROFILL]

Są to synonimy typu DECIMAL.

W MySQL 4.1.0 dodano synonim FIXED w celu osiągnięcia kompatybilności z innymi serwerami.

4.1.2. Przegląd typów związanych z datą i czasem

Poniżej znajduje się podsumowanie typów kolumn przeznaczonych dla czasu i daty. Dodatkowe informacje na ten temat można znaleźć w podrozdziale 4.3. Wymogi dotyczące pamięci potrzebnej do przechowywania danych w kolumnach zostały podane w podrozdziale 4.5.

- `DATE`

  Data. Obsługiwany zakres obejmuje daty od `'1000-01-01'` do `'9999-12-31'`. W MySQL wartości `DATE` wyświetlane są w formacie `'RRRR-MM-DD'`, ale można przypisywać wartości do kolumn `DATE` przy użyciu łańcuchów lub liczb.

- `DATETIME`

  Kombinacja daty i czasu. Obsługiwany zakres obejmuje wartości od `'1000-01-01 00:00:00'` do `'9999-12-31 23:59:59'`. W MySQL wartości `DATETIME` wyświetlane są w formacie `'RRRR-MM-DD GG:MM:SS'`, ale można przypisywać wartości do kolumn `DATETIME` przy użyciu łańcuchów lub liczb.

- `TIMESTAMP[(M)]`

  Znacznik czasu. Obsługiwany zakres rozpoczyna się od daty `'1970-01-01'` i sięga roku `2037`.

  Kolumna typu `TIMESTAMP` przydaje się do rejestracji daty i godziny operacji wstawiania lub aktualizacji. Pierwszej kolumnie `TIMESTAMP` w tabeli automatycznie przypisywana jest data i czas ostatniej operacji, jeśli tylko tych wartości nie wprowadzi ręcznie użytkownik. Dowolnej kolumnie `TIMESTAMP` można również przypisać bieżącą datę i godzinę, wpisując do niej wartość `NULL`.

  Od MySQL 4.1 wartości `TIMESTAMP` są zwracane w postaci łańcuchów w formacie `'RRRR-MM-DD GG:MM:SS'`. Aby uzyskać wartość w postaci liczby, należy dodać do kolumn tego typu `+0`. Inne szerokości wyświetlania znacznika czasu nie są obsługiwane.

  W wersjach poprzedzających MySQL 4.1 wartości `TIMESTAMP` są wyświetlane w formacie `RRRRMMDDGGMMSS`, `RRMMDDGGMMSS`, `RRRRMMDD` lub `RRMMDD` w zależności od tego, czy parametr *M* jest równy 14 (albo go brak), 12, 8 lub 6. Można jednak przypisywać wartości do kolumn `TIMESTAMP` przy użyciu łańcuchów lub liczb. Argument *M* wpływa tylko na sposób wyświetlania kolumny `TIMESTAMP`, a nie na jej przechowywanie. Wartości kolumny są zawsze zapisywane w czterech bajtach. W MySQL od wersji 4.0.12 pojawiła się opcja `--new`, dzięki której można zmienić działanie serwera, upodabniając je do MySQL 4.1.

  Należy odnotować, że kolumny `TIMESTAMP(M)`, gdzie *M* jest równe 8 lub 14, są traktowane jako liczby, natomiast reszta kolumn `TIMESTAMP(M)` jest traktowana jako łańcuchy. Jest to rodzaj gwarancji, że użytkownik będzie mógł wiarygodnie usuwać i przywracać tabele tego typu.

- `TIME`

  Godzina. Obsługiwany zakres obejmuje wartości od `'-838:59:59'` do `' 838:59:59'`. W MySQL wartości `TIME` wyświetlane są w formacie `' GG:MM:SS'`, ale można przypisać wartości do kolumn `TIME` przy użyciu zarówno łańcuchów, jak i liczb.

♦ YEAR[(2|4)]

Rok w formacie dwu- lub czterocyfrowym. Domyślnym formatem jest rok zapisany za pomocą czterech cyfr. W tym formacie dostępne wartości należą do przedziału od 1901 do 2155 oraz 0. W formacie dwucyfrowym wartości dopuszczalne to liczby z zakresu od 70 do 69, które reprezentują daty od roku 1970 do 2069. W MySQL wartości YEAR wyświetlane są w formacie RRRR, ale można przypisać wartości do kolumn YEAR przy użyciu tak łańcuchów, jak i liczb. Typ YEAR jest niedostępny dla wersji wcześniejszych niż MySQL 3.22.

4.1.3. Przegląd typów łańcuchowych

Poniżej znajduje się przegląd typów łańcuchowych. Dodatkowe informacje można znaleźć w podrozdziale 4.4. Wymogi dotyczące pamięci potrzebnej do przechowywania danych w kolumnach zostały podane w podrozdziale 4.5.

W niektórych przypadkach MySQL może zmienić kolumnę typu łańcuchowego na typ, który będzie się różnił od zadeklarowanego w instrukcji CREATE TABLE lub ALTER TABLE. Więcej informacji na ten temat znajduje się w podpunkcie 6.2.5.2.

Od wersji 4.1 w MySQL pojawiła się zmiana, która wpływa na wiele typów kolumn typu łańcuchowego. Definicja kolumn typu znakowego może zawierać atrybut CHARACTER SET, który określa zestaw znaków i opcjonalnie porządek sortowania. Dotyczy to typów CHAR, VARCHAR, różnych typów TEXT oraz ENUM i SET, na przykład:

```
CREATE TABLE t
(
   k1 CHAR(20) CHARACTER SET utf8,
   k2 CHAR(20) CHARACTER SET latin1 COLLATE latin1_bin
);
```

Ta definicja tabeli tworzy kolumnę o nazwie k1 z zestawem znaków utf8 i z domyślnym sortowaniem dla tego zestawu znaków oraz kolumnę k2 z zestawem znaków latin1 i jego binarnym sortowaniem. Podczas sortowania binarnego nie jest brana pod uwagę wielkość liter.

Sortowanie i porównywanie kolumn typu znakowego jest wykonywane na podstawie zestawu znaków, który jest przypisany do kolumny. W wersjach poprzedzających MySQL 4.1 wspomniane operacje bazują na porządku sortowania zestawu znaków serwera. Dla kolumn CHAR i VARCHAR można zadeklarować kolumnę z atrybutem BINARY, aby spowodować sortowanie i porównywanie rozróżniające wielkość liter przy użyciu wartości liczbowych kodu, a nie porządkowania leksykograficznego.

Więcej szczegółów na ten temat można znaleźć w rozdziale 3.

Dodatkowo, od wersji 4.1 MySQL interpretuje deklaracje długości przechowywanych danych w definicjach kolumn typu znakowego w znakach (w wersjach wcześniejszych interpretowanie wykonywane było w bajtach).

Oto dostępne w MySQL typy łańcuchowe:

- `[NATIONAL] CHAR (M) [BINARY | ASCII | UNICODE]`

  Łańcuch o stałej długości, który jest zawsze w celu przechowania dopełniany po prawej stronie odpowiednią liczbą spacji. Parametr *M* reprezentuje długość kolumny. Zakres *M* wynosi od 0 do 255 znaków (w wersjach poprzedzających MySQL 3.23 był to przedział od 1 do 255).

> Końcowe spacje zostają usunięte w czasie pobierania wartości `CHAR`.

  Od MySQL 4.1.0 kolumna `CHAR` o specyfikacji długości przekraczającej 255 zostanie przekształcona na najmniejszy typ `TEXT`, który może przechowywać wartości takiej długości. Wartość `CHAR(500)` zostanie na przykład przekształcona na typ `TEXT`, zaś `CHAR(200000)` na typ `MEDIUMTEXT`. Jest to spowodowane wymogami kompatybilności. Jednak taka konwersja wpływa na kolumny, które stają się kolumnami zmiennej długości, a także powoduje usuwanie końcowych spacji.

  Słowo kluczowe `CHAR` to skrót od `CHARACTER`. Typ `NATIONAL CHAR` (lub odpowiadająca mu forma skrócona `NCHAR`) to należąca do standardu SQL metoda definiowania, opierająca się na założeniu, że kolumna `CHAR` powinna użyć domyślnego zestawu znaków. W MYSQL to ustawienie jest domyślne.

  Atrybut `BINARY` powoduje rozróżnianie wielkości liter podczas operacji sortowania i porównywania.

  Od MySQL 4.1.0 można określać atrybut `ASCII`. Powoduje on przypisanie do kolumny typu `CHAR` zestawu znaków `latin1`.

  Od wersji 4.1.1 MySQL można także podawać atrybut `UNICODE`. Powoduje on przypisanie do kolumny `CHAR` zestawu znaków `ucs2`.

  MySQL pozwala tworzyć kolumnę typu `CHAR(0)`. Jest to szczególnie przydatne, w sytuacjach gdy zachodzi potrzeba osiągnięcia zgodności z jakimiś starymi aplikacjami, które zależą od istnienia kolumny, choć samej wartości w rzeczywistości nie używają. Jest to również praktyczne, gdy potrzebna jest kolumna, która może przyjąć tylko dwie wartości: kolumna `CHAR(0)`, jeśli nie zostanie zdeklarowana jako `NOT NULL`, zajmuje tylko jeden bit i może przybierać tylko wartości `NULL` i `''` (pusty łańcuch).

- `CHAR`

  Jest to synonim typu `CHAR(1)`.

- `[NATIONAL] VARCHAR (M) [BINARY]`

  Łańcuch o zmiennej długości. Parametr *M* reprezentuje maksymalną długość kolumny. Zakres *M* wynosi od 0 do 255 znaków (w wersjach wcześniejszych od MySQL 4.0.2 wynosił od 1 do 255).

> Końcowe spacje zostają usunięte w momencie zapisywania wartości `VARCHAR`, co nie jest zgodne ze specyfikacją standardu SQL.

Od MySQL 4.1.0 kolumna VARCHAR o specyfikacji długości przekraczającej 255 zostanie przekształcona na najmniejszy typ TEXT, który może przechowywać wartości takiej długości. Wartość VARCHAR(500) zostanie na przykład przekształcona na typ TEXT, natomiast VARCHAR(200000) na typ MEDIUMTEXT. Jest to spowodowane wymogami kompatybilności. Taka konwersja wpływa jednak na usuwanie końcowych spacji.

Słowo kluczowe VARCHAR jest skrótem od słów CHARACTER VARYING.

Atrybut BINARY powoduje rozróżnianie wielkości liter podczas operacji sortowania i porównywania.

- TINYBLOB, TINYTEXT

  Kolumna BLOB lub TEXT z maksymalną długością wynoszącą 255 (2^8–1) znaków.

- BLOB, TEXT

  Kolumna BLOB lub TEXT z maksymalną długością wynoszącą 65 535 (2^{16}–1) znaków.

- MEDIUMBLOB, MEDIUMTEXT

  Kolumna BLOB lub TEXT z maksymalną długością wynoszącą 16 777 215 (2^{24}–1) znaków.

- LONGBLOB, LONGTEXT

  Kolumna BLOB lub TEXT z maksymalną długością wynoszącą 4 294 967 295 lub 4 GB (2^{32}–1) znaków. Do wersji 3.23 MySQL protokół klient-serwer i tabele MyISAM miały nałożone ograniczenie do 16 MB na pakiet komunikacyjny i wiersz tabeli. Od MySQL 4.0 maksymalna dopuszczalna długość kolumn LONGBLOB lub LONGTEXT zależy od skonfigurowanego maksymalnego rozmiaru pakietu w protokole klient-serwer i dostępnej pamięci.

- ENUM('*wartość1*','*wartość2*',...)

  Wyliczenie. Obiekt łańcuchowy, który może składać się tylko z jednej, wybranej z listy, wartości: '*wartość1*', '*wartość2*', ..., NULL lub specjalnej wartości błędu ''. Kolumna ENUM może zawierać maksymalnie do 65 535 różnych wartości. Elementy ENUM są reprezentowane wewnętrznie jako liczby całkowite.

- SET('*wartość1*','*wartość2*',...)

  Zbiór. Obiekt łańcuchowy, który może składać się dowolnej liczby wartości, z których każda musi pochodzić z listy wartości '*wartość1*', '*wartość2*', ... Kolumna SET może liczyć maksymalnie 64 elementy. Wartości SET są reprezentowane wewnętrznie jako liczby całkowite.

4.2. Typy liczbowe

MySQL obsługuje wszystkie liczbowe typy danych standardu SQL. Są to dokładne typy liczbowe danych (INTEGER, SMALLINT, DECIMAL i NUMERIC), jak również przybliżone (FLOAT, REAL i DOUBLE PRECISION). Słowo kluczowe INT jest synonimem dla typu INTEGER, natomiast słowo kluczowe DEC jest synonimem typu DECIMAL.

Jako rozszerzenie do standardu SQL, MySQL obsługuje również między innymi takie typy dla liczb całkowitych jak TINYINT, MEDIUMINT i BIGINT. Zostało to przedstawione w poniższym zestawieniu.

Typ	Bajty	Wartość minimalna (ze znakiem)	Wartość maksymalna (ze znakiem)
TINYINT	1	-128	127
SMALLINT	2	-32768	32767
MEDIUMINT	3	-8388608	8388607
INT	4	-2147483648	2147483647
BIGINT	8	-9223372036854775808	9223372036854775807

MySQL posiada także rozszerzenie służące do opcjonalnego określania dla liczby całkowitej maksymalnej szerokości wyświetlania, której wartość podawana jest w nawiasach po słowie kluczowym nazwy typu (na przykład INT(4)). Pozwala to z lewej strony uzupełniać spacjami wyświetlane wartości o szerokości mniejszej niż szerokość określona dla kolumny.

Szerokość wyświetlania nie ogranicza zakresu wartości, które można przechowywać w kolumnie, ani też liczby wyświetlanych cyfr tych wartości, których szerokość przekroczy maksimum określone dla kolumny.

Jeśli wraz z opcjonalnym rozszerzeniem użyty zostanie atrybut ZEROFILL, domyślne dopełnianie spacjami zostanie zastąpione zerami. Dla kolumny zadeklarowanej na przykład jako INT(5) ZEROFILL, wartość 4 po pobraniu będzie miała postać 00004. Należy zauważyć, że w przypadku przechowywania w kolumnie typu liczby całkowitej wartości, która przekracza szerokość wyświetlania, może wystąpić problem podczas generowania przez MySQL tymczasowej tabeli dla pewnych skomplikowanych złączeń. W takich wypadkach zakłada się, że dane zmieściły się w oryginalnej szerokości kolumny.

Wszystkie typy dla liczb całkowitych mogą opcjonalnie korzystać z (niestandardowego) atrybutu UNSIGNED. Wartość bez znaku można stosować w sytuacjach, gdy chcemy dopuścić w kolumnie tylko liczby nieujemne, ale potrzebujemy wyższej górnej granicy zakresu liczbowego.

Od MySQL 4.0.2 atrybut UNSIGNED można także określać dla typów zmiennoprzecinkowych i stałoprzecinkowych. Podobnie jak w przypadku typów dla liczb całkowitych, atrybut ten zapobiega przechowywaniu w kolumnie wartości ujemnych. Jednak w przeciwieństwie do typów dla liczb całkowitych górna granica zakresu wartości kolumny pozostaje bez zmian.

Jeśli dla kolumny liczbowej określony zostanie atrybut ZEROFILL, MySQL automatycznie doda do kolumny atrybut UNSIGNED.

Typy DECIMAL i NUMERIC implementowane są w MySQL jako ten sam typ. Służą do przechowywania wartości, dla których ważne jest zachowanie dokładności, takich jak dane walutowe. Podczas deklarowania kolumny jednego z tych typów można określić (i zazwyczaj się to robi) dokładność oraz skalę, na przykład:

```
pensja DECIMAL(5,2)
```

W tym przykładzie 5 jest dokładnością, a 2 skalą. Dokładność oznacza tutaj precyzyjne określenie liczby przechowywanych dla wartości znaczących cyfr dziesiętnych, a skala — liczby przechowywanych cyfr po kropce dziesiętnej.

MySQL przechowuje wartości `DECIMAL` i `NUMERIC` w postaci łańcuchów, a nie binarnych liczb zmiennoprzecinkowych. Pozwala to zachować ich dokładność dziesiętną. Każda cyfra wartości, kropka dziesiętna (jeśli skala jest większa niż zero) i znak minusa (dla liczb ujemnych) jest reprezentowany przez jeden znak. Jeśli skala ma wartość 0, wartości `DECIMAL` i `NUMERIC` nie zawierają kropki dziesiętnej lub części ułamkowej.

Standard SQL wymaga, by kolumna `pensja` mogła przechowywać wszystkie wartości składające się z pięciu cyfr i dwóch miejsc po przecinku. Dlatego w tym przypadku zakres wartości, które mogą być przechowywane w tej kolumnie, to przedział od `-999.99` do `999.99`. Trzeba jednak pamiętać o dwóch sprawach:

- Na dodatniej granicy zakresu kolumna może w rzeczywistości przechowywać liczby do `9999.99`. Dla liczb dodatnich MySQL rozszerza górną granicę zakresu, ponieważ używa bajta zarezerwowanego dla znaku.
- W wersjach poprzedzających MySQL 3.23 kolumny typu `DECIMAL` są przechowywane inaczej i nie mogą reprezentować wszystkich wartości wymaganych przez standard SQL. To dlatego, że dla typu `DECIMAL(M,D)` wartość *M* obejmuje bajty dla znaku i kropki dziesiętnej. W ten sposób do zakresu kolumny `pensja` przed MySQL 3.23 należałyby liczby od `-9.99` do `99.99`.

W standardzie SQL składnia `DECIMAL(M)` jest równoważna składni `DECIMAL(M,0)`. Podobnie składnia `DECIMAL` jest odpowiednikiem `DECIMAL(M,0)`, jeśli dozwolone jest definiowanie wartości *M*. Od MySQL 3.23.6 obsługiwane są obie formy typów danych `DECIMAL` i `NUMERIC`. Wartość domyślna parametru *M* wynosi 10. W wersjach poprzedzających 3.23.6 konieczne jest wyraźne określenie obu parametrów *M* i *D*.

Maksymalny zakres wartości `DECIMAL` i `NUMERIC` jest taki sam jak dla typu `DOUBLE`, jednak na rzeczywisty zakres tych kolumn może wpływać zadeklarowana dla danej kolumny dokładność lub skala. Jeśli kolumna takiego typu ma przypisaną wartość o liczbie miejsc po przecinku przekraczającej zezwalaną skalę, wartość zostaje przekształcona do tej skali (zależy to od systemu operacyjnego, ale zazwyczaj jest to przekształcenie do dozwolonej liczby cyfr). Gdy kolumnie `DECIMAL` lub `NUMERIC` przypisana jest wartość, która przekracza zakres wskazany przez określoną w definicji (lub domyślną) dokładność i skalę, MySQL przechowuje wartość reprezentującą koniec tego zakresu.

W przypadku typów dla kolumn z liczbami zmiennoprzecinkowymi MySQL przechowuje wartości o pojedynczej precyzji przy użyciu czterech bajtów, a wartości o podwójnej precyzji, wykorzystując osiem bajtów.

Typ `FLOAT` służy do reprezentowania przybliżonych liczbowych typów danych. Standard SQL pozwala na opcjonalne deklarowanie dokładności (ale nie zakresu wykładnika) w nawiasach w bitach po słowie kluczowym `FLOAT`. Implementacja MySQL obsługuje również tę opcjonalną specyfikację dokładności, ale jej wartość służy jedynie do określenia rozmiaru pamięci. Dokładność od 0 do 23 powoduje uzyskanie czterobajtowej kolumny `FLOAT` o pojedynczej precyzji. Dokładność od 24 do 53 daje w rezultacie ośmiobajtową kolumnę `DOUBLE` o podwójnej precyzji.

Jeśli słowo kluczowe FLOAT zostaje użyte do zadeklarowania typu kolumny bez specyfikacji dokładności, wówczas wartości przechowywane są przy wykorzystaniu czterech bajtów. MySQL obsługuje również odmianę tej składni, w której po słowie kluczowym FLOAT znajdują się dwie umieszczone w nawiasach liczby. Pierwsza reprezentuje szerokość wyświetlania, a druga liczbę przechowywanych i wyświetlanych cyfr po kropce dziesiętnej (podobnie jak w przypadku typów DECIMAL i NUMERIC). Kiedy zachodzi potrzeba przechowania w takiej kolumnie liczby, której ilość cyfr po kropce dziesiętnej przekracza określony dla kolumny limit, to przechowywana wartość zostanie zaokrąglona, aby usunąć dodatkowe cyfry.

W standardzie SQL typy REAL i DOUBLE PRECISION nie zezwalają na określanie dokładności. Przez MySQL obsługiwany jest wariant składni, w którym po nazwie typu można podać w nawiasie dwie liczby. Pierwsza reprezentuje maksymalną szerokość wyświetlania, a druga — liczbę przechowywanych i wyświetlanych cyfr po kropce dziesiętnej. Jako rozszerzenie standardu SQL, DOUBLE rozpoznawane jest przez MySQL jako synonim typu DOUBLE PRECISION. W odróżnieniu od nakładanego przez standard wymogu, by dokładność dla typu REAL była mniejsza od określonej dla DOUBLE PRECISION, oba typy implementowane są przez MySQL jako ośmiobajtowe wartości zmiennoprzecinkowe o podwójnej precyzji (chyba że dla parametru sql-mode serwera włączono opcję REAL_AS_FLOAT).

Dla uzyskania maksymalnej przenośności, w kodzie, który wymaga przechowywania przybliżonych wartości danych liczbowych, powinien być używany typ FLOAT lub DOUBLE PRECISION bez określania dokładności lub liczby miejsc po kropce dziesiętnej.

Jeśli w kolumnie typu liczbowego ma być przechowywana wartość spoza dopuszczalnego zakresu dla danego typu, wartość zostanie przycięta do odpowiedniej granicy zakresu i dopiero w ten sposób uzyskana liczba będzie przechowana.

Kolumna INT ma na przykład zakres od -2147483648 do 2147483647. Gdyby spróbować wstawić do niej liczbę -9999999999, wartość ta zostałaby przycięta do dolnego krańca zakresu i w zamian zachowana liczba -2147483648. Podobnie, gdyby spróbować wstawić liczbę 9999999999, wartość ta zostałaby przycięta do górnego krańca zakresu i w zamian zachowana liczba 2147483647.

Jeśli kolumna INT ma atrybut UNSIGNED, wielkość zakresu kolumny pozostaje bez zmian, ale jego krańce przesuwają się, tworząc przedział od 0 do 4294967295. Gdyby spróbować przechować liczby -9999999999 i 9999999999, w kolumnach zostałyby zapisane wartości 0 i 4294967295.

Konwersje, spowodowane odcinaniem, są dla instrukcji ALTER TABLE, LOAD DATA INFILE, UPDATE i wielowierszowych INSERT poprzedzane ostrzeżeniami.

4.3. Typy związane z datą i czasem

Typami, które służą do reprezentowania wartości związanych z datą i czasem, są DATE ➥TIME, DATE, TIMESTAMP, TIME i YEAR. Wszystkie mają zakres dopuszczalnych wartości oraz wartość zerową, wykorzystywaną w razie podania wartości niedozwolonej, której

MySQL nie może reprezentować. Typ TIMESTAMP posiada także opisaną w dalszej części rozdziału specjalną możliwość automatycznej aktualizacji.

MySQL pozwala przechowywać pewne „niekoniecznie poprawne" wartości dat, takie jak '1999-11-31'. To dlatego, że sprawdzanie daty należy do czynności wykonywanych przez aplikację, a nie serwer MySQL. Aby przyspieszyć sprawdzanie dat, przez MySQL weryfikowane jest tylko, czy podany miesiąc należy do przedziału od 0 do 12, a dzień do przedziału od 0 do 31. Zakresy te zawierają zero, ponieważ MySQL pozwala na przechowywanie w kolumnie DATE i DATETIME danych, w których dzień lub miesiąc jest zerem. Jest to szczególnie praktyczne dla aplikacji, za pomocą których zapisywana jest data urodzin, chociaż data taka nie jest dokładnie znana. W takim przypadku wystarczy zapisać datę w postaci '1999-00-00' lub '1999-01-00'. Przechowując datę zapisaną w ten sposób, nie należy oczekiwać uzyskania poprawnych wyników z takimi funkcjami, jak DATE_SUB() lub DATE_ADD, które wymagają podania pełnych dat.

Oto ogólne warunki, o których warto pamiętać, pracując z typami daty i czasu:

- Wartości dla danego typu daty lub czasu pobierane są przez MySQL w standardowym formacie wyjściowym, ale podejmowana jest próba interpretacji różnych formatów dla wartości wejściowych, dostarczanych przez użytkownika (na przykład gdy użytkownik poda wartość do przypisania do typu związanego z datą lub godziną lub do porównania z nim). Obsługiwane są tylko formaty opisane w poniższych podrozdziałach. Oczekuje się od użytkownika dostarczania poprawnych wartości. Użycie wartości w innych formatach może spowodować pojawienie się nieoczekiwanych wyników.
- Daty zawierające dwucyfrowe wartości dla roku są niejednoznaczne, ponieważ nie jest znany wiek. Takie dwucyfrowe wartości interpretowane są przez MySQL za pomocą następujących zasad:
  - Wartości roku w zakresie od 00-69 są przekształcane na lata 2000-2069.
  - Wartości roku w zakresie 70-99 są przekształcane na lata 1970-1999.
- Choć wartości mogą być interpretowane przez MySQL w wielu formatach, daty muszą być zawsze podawane w kolejności rok-miesiąc-dzień (na przykład '98-09-04'), a nie w powszechnie używanej postaci miesiąc-dzień-rok lub dzień-miesiąc-rok (na przykład '09-04-98', '04-09-98').
- MySQL powoduje automatyczne przekształcenie wartości typu daty lub czasu na liczbę, jeśli są one używane w kontekście liczbowym i na odwrót.
- Kiedy MySQL napotka wartość dla typu związanego z datą lub czasem, która wychodzi poza zakres lub jest w inny sposób niedopuszczalna (w sposób opisany na początku tego punktu), przekształca tę wartość na wartość „zero" danego typu. Wyjątkiem są wychodzące poza zakres wartości TIME, które są przycinane do odpowiedniej wartości stanowiącej granicę zakresu tego typu.

  Poniższe zestawienie przedstawia formaty wartości „zerowej" dla każdego typu:

Typ kolumny	Wartość „zerowa”
DATETIME	'0000-00-00 00:00:00'
DATE	'0000-00-00'
TIMESTAMP	00000000000000
TIME	'00:00:00'
YEAR	0000

- Wartości „zerowe” są specjalne, ale można je przechowywać lub odwoływać się do nich w sposób jawny, używając wartości przedstawionych w tabeli. Można również wykonywać to, korzystając z wartości '0' lub 0, co jest w sumie łatwiejsze do zapisania.
- Zerowe wartości daty lub czasu, stosowane w interfejsie Connector/ODBC, zostają automatycznie przekształcone do wartości NULL w interfejsie Connector/ODBC 2.50.12, ponieważ ODBC nie potrafi obsługiwać takich wartości.

4.3.1. Typy DATETIME, DATE i TIMESTAMP

Typy DATETIME, DATE i TIMESTAMP są ze sobą związane. W tym punkcie opisana zostanie ich charakterystyka, a także omówione będą podobieństwa i różnice między nimi.

Typ DATETIME stosuje się, gdy potrzebne są wartości składające się z informacji o dacie i czasie. Wartości DATETIME pobierane są i wyświetlane przez MySQL w formacie 'RRRR-MM-DD GG:MM:SS'. Obsługiwany zakres obejmuje wartości od '1000-01-01 00:00:00' do '9999-12-31 23:59:59' („obsługiwany” oznacza, że choć wartości dla dat wcześniejszych mogą być prawidłowo zinterpretowane, nie ma gwarancji, że tak będzie).

Typ DATE stosuje się, gdy potrzebna jest tylko wartość daty bez części podającej godzinę. Wartości DATE wyświetlane są przez MySQL w formacie 'RRRR-MM-DD'. Obsługiwany zakres obejmuje daty od '1000-01-01' do '9999-12-31'.

Właściwości typu kolumny TIMESTAMP różnią się w zależności od wersji MySQL i trybu SQL, w którym działa serwer. Zostały one podane w dalszej części tego punktu.

Wartości DATETIME, DATE i TIMESTAMP można deklarować za pomocą dowolnego z popularnych zestawów formatów:

- Jako łańcuch w formacie 'RRRR-MM-DD GG:MM:SS' lub 'RR-MM-DD GG:MM:SS'. Dozwolona jest nieco mniej rygorystyczna składnia: znakiem rozdzielającym elementy daty i czasu może być dowolny znak interpunkcyjny. Równoważne będą na przykład zapisy '98-12-31 11:30:45', '98.12.31 11+30+45', '98/12/31 11*30*45' i '98@12@31 11^30^45'.
- Jako łańcuch w formacie 'RRRR-MM-DD' lub 'RR-MM-DD'. I tu dozwolona jest mniej rygorystyczna składnia, na przykład '98-12-31', '98.12.31', '98/12/31' i '98@12@31'.

- Jako łańcuch bez znaków rozdzielających elementy daty w formacie `'RRRRMMDDGGMMSS'` lub `'RRMMDDGGMMSS'`, jeśli tylko dany ciąg odzwierciedla rozsądną datę. Zapisy `'19970523091528'` i `'970523091528'` interpretowane są na przykład jako `'1997-05-23 09:15:28'`, ale `'971122129015'` jest wartością niedozwoloną (dziwna liczba w elemencie reprezentującym minuty) i zostanie przekształcona do wartości `'0000-00-00 00:00:00'`.
- Jako łańcuch bez znaków rozdzielających elementy daty w formacie `'RRRRMMDD'` lub `'RRMMDD'`, jeśli tylko dany ciąg odzwierciedla rozsądną datę. Na przykład `'19970523'` i `'970523'` są interpretowane jako `'1997-05-23'`, ale `'971332'` jest wartością niedozwoloną (dziwny numer miesiąca i dnia) i zostanie przekształcona do wartości `'0000-00-00'`.
- Jako liczba w formacie `RRRRMMDDGGMMSS` lub `RRMMDDGGMMSS`, jeśli tylko dany ciąg odzwierciedla rozsądną datę. Na przykład `19830905132800` i `830905132800` są interpretowane jako `'1983-09-05 13:28:00'`.
- Jako liczba w formacie `RRRRMMDD` lub `RRMMDD`, jeśli tylko dany ciąg odzwierciedla rozsądną datę. Na przykład `19830905` i `830905` są interpretowane jako `'1983-09-05'`.
- Jako wynik funkcji, która zwraca wartość akceptowaną przez typy `DATETIME`, `DATE` i `TIMESTAMP`, na przykład `NOW()` lub `CURRENT_DATE`.

Niedozwolone wartości `DATETIME`, `DATE` i `TIMESTAMP` są przekształcane na wartości zerowe odpowiedniego typu (`'0000-00-00 00:00:00'`, `'0000-00-00'` lub `00000000000000`).

W przypadku wartości określonych jako łańcuchy, które zawierają znaki rozdzielające poszczególne części daty, nie ma konieczności podawania dwóch cyfr dla wartości miesiąca i dnia nie przekraczających 10. `'1979-6-9'` jest tym samym, czym jest `'1979-06-09'`. Podobnie dla wartości określanych jako łańcuchy ze znakami rozdzielającymi części godziny, nie ma konieczności podawania dwóch cyfr dla wartości godziny, minuty i sekundy mniejszych od 10. `'1979-10-30 1:2:3'` jest tym samym, czym jest `'1979-10-30 01:02:03'`.

Wartości zadeklarowane jako liczby powinny mieć długość 6, 8, 12 i 14 cyfr. Jeśli liczba składa się z 8 lub 14 cyfr, zakłada się, że jest w formacie `RRRRMMDD` i `RRRRMMDDGGMMSS` oraz że rok jest reprezentowany przez pierwsze cztery cyfry. Jeśli liczba składa się z 6 lub 12 cyfr, zakłada się, że jest w formacie `RRMMDD` i `RRMMDDGGMMSS` i że rok reprezentują pierwsze dwie cyfry. Liczby składające się z innej liczby cyfr interpretuje się, jakby zostały uzupełnione początkowymi zerami do najbliższej długości.

Wartości określone jako łańcuchy bez znaków rozdzielających części daty lub godziny są interpretowane z wykorzystaniem ich długości w sposób następujący: jeśli łańcuch składa się z 8 lub 14 znaków, zakłada się, że rok reprezentują pierwsze cztery cyfry. W przeciwnym razie zakłada się, ze rok został podany w postaci dwucyfrowej. Łańcuch jest interpretowany od lewej do prawej. Najpierw szukana jest wartość roku, miesiąca, dnia, godziny, minuty i sekundy dla jak największej liczby elementów obecnych w ciągu. To oznacza, że nie należy używać łańcuchów zawierających mniej niż 6 znaków. Podanie na przykład łańcucha `'9903'`, który w zamyśle miał reprezentować marzec 1999 roku, spowoduje, że do tabeli zostanie przez MySQL wstawiona wartość zerowa. Mimo że

wartości roku i miesiąca są przechowywane w postaci liczb 99 i 03, brakuje zupełnie części odnoszącej się do dnia, dlatego nie jest to poprawna wartość. Jednak począwszy od MySQL 3.23 użytkownik może brakujące części miesiąca i dnia zdefiniować w sposób jawny jako zero. Można na przykład użyć wartości '990300', aby wstawić wartość '1999-03-00'.

Istnieje pewna możliwość przypisywania wartości jednego typu danych do obiektu o innym typie. Może to jednak prowadzić do pewnych zmian w wartości lub do utraty danych:

- Jeśli obiektom DATETIME i TIMESTAMP przypisze się wartość DATE, część wartości wyjściowej, związana z czasem, wynosić będzie '00:00:00', ponieważ wartość DATE nie zawiera żadnych informacji o czasie.
- Jeśli obiektowi DATE przypisze się wartość DATETIME lub TIMESTAMP, część wartości wyjściowej, dotycząca czasu, zostanie usunięta, ponieważ typ DATA nie przechowuje żadnych informacji o czasie.
- Należy pamiętać, że choć wartości DATETIME, DATE i TIMESTAMP można określać przy użyciu tego samego zestawu formatów, nie wszystkie typy mają ten sam zakres wartości. Na przykład wartości TIMESTAMP nie mogą być datami wcześniejszymi niż rok 1970 lub starszymi niż rok 2037. To oznacza, że data taka jak '1968-01-01', choć dopuszczalna jako wartość DATETIME i DATE, nie jest poprawną wartością TIMESTAMP i zostanie przekształcona na 0 w razie przypisania do takiego obiektu.

Podczas określania wartości daty należy zapoznać się z pewnymi pułapkami:

- Uproszczony format, dozwolony dla wartości deklarowanych jako łańcuchy, może być nieco mylący. Wartość '10:11:12' może na przykład z powodu użytego znaku oddzielającego poszczególne elementy ':' przypominać godzinę, jednak użyta w kontekście daty zostanie zinterpretowana jako rok '2010-11-12'. Wartość '10:45:15' zostanie przekształcona na '0000-00-00', ponieważ '45' nie jest poprawnie zapisanym miesiącem.
- Serwer MySQL wykonuje tylko podstawowe sprawdzanie poprawności daty: zakres dla roku, miesiąca i dnia to odpowiednio od 1000 do 9999, od 00 do 12 i od 00 do 31. Każda data, zawierająca części wychodzące poza te zakresy, podlega przekształceniu na '0000-00-00'. Należy jednak pamiętać, że to nie zapobiegnie przechowywaniu takiej niepoprawnej daty, jak '2002-04-31'. Poprawność daty należy sprawdzić w aplikacji.
- Daty, zawierające wartości roku w postaci dwucyfrowej, są niejednoznaczne z powodu braku informacji o wieku. Interpretowane są przez MySQL w następujący sposób:
  - Wartości reprezentujące rok w zakresie od 00-69 zostają przekształcone na rok 2000-2069.
  - Wartości reprezentujące rok w zakresie od 70-99 zostają przekształcone na rok 1970-1999.

4.3.1.1. Właściwości TIMESTAMP przed wersją 4.1 MySQL

Typ TIMESTAMP można wykorzystać do automatycznego oznaczania bieżącą datą i czasem operacji INSERT i UPDATE. Jeśli tabela zawiera wiele kolumn TIMESTAMP, automatycznie aktualizowana jest tylko pierwsza.

Automatyczna aktualizacja pierwszej kolumny TIMESTAMP w tabeli jest wykonywana, gdy spełniony jest dowolny z poniższych warunków:

- Kolumna zostaje jawnie zadeklarowana jako NULL.
- Kolumna nie jest jawnie określona w instrukcji INSERT lub LOAD DATA INFILE.
- Kolumna nie jest jawnie określona w instrukcji UPDATE, a w jakiejś innej kolumnie zmieniana jest wartość. Jeżeli aktualizacja przypisze kolumnie wartość, którą ta już zawiera, kolumna TIMESTAMP nie będzie aktualizowana. Jeśli kolumnie zostanie przypisana jej bieżąca wartość, aktualizacja zostanie przez MySQL zignorowana z powodów wydajnościowych.

Bieżącą datę i godzinę można przypisywać nie tylko pierwszej kolumnie TIMESTAMP. Aby wpisać je do innych kolumn tego typu, wystarczy przypisać kolumnie wartość NULL lub NOW().

Dowolnej kolumnie TIMESTAMP można jawnie przypisać wartość inną niż bieżąca data i czas. Dotyczy to nawet pierwszej kolumny TIMESTAMP. Jest to przydatne, w sytuacji gdy na przykład chcemy przypisać TIMESTAMP bieżącą datę i czasy w momencie tworzenia rekordu, ale tak, by informacja ta nie była już później zmieniana:

- Należy podczas tworzenia rekordu zezwolić MySQL na przypisanie kolumnie wartości. To spowoduje zainicjowanie jej z bieżącą datą i czasem.
- Po wykonaniu kolejnych aktualizacji innych kolumn w rekordzie, należy przypisać kolumnie TIMESTAMP w sposób jawny bieżącą wartość znacznika czasu:

```
UPDATE nazwa_tabeli
   SET kol_timestamp = kol_timestamp,
      inna_kol1 = nowa_wartość1,
      inna_kol2 = nowa_wartość2, ...
```

Innym sposobem utworzenia kolumny, w której rejestrowany będzie czas utworzenia rekordu, jest przygotowanie kolumny DATETIME, inicjalizowanej za pomocą funkcji NOW() w momencie tworzenia wiersza, po czym, przy kolejnych aktualizacjach, pozostawianie jej bez zmian.

Zakres obejmowany przez wartości TIMESTAMP rozpoczyna się od roku 1970 i sięga do roku 2037 z dokładnością co do jednej sekundy. Wartości wyświetlane są jako liczby.

Format, w którym przez MySQL pobierane są i wyświetlane wartości TIMESTAMP, zależy od maksymalnej wyświetlanej liczby cyfr — zostało to przedstawione w poniższym zestawieniu. Pełny format typu TIMESTAMP liczy 14 cyfr, ale takie kolumny można także tworzyć z mniejszym rozmiarem wyświetlania:

Specyfikacja typu	Format wyświetlania
TIMESTAMP(14)	RRRRMMDDGGMMSS
TIMESTAMP(12)	RRMMDDGGMMSS
TIMESTAMP(10)	RRMMDDGGMM
TIMESTAMP(8)	RRRRMMDD
TIMESTAMP(6)	RRMMDD
TIMESTAMP(4)	RRMM
TIMESTAMP(2)	RR

Rozmiar przechowywanych wartości wszystkich kolumn TIMESTAMP jest ten sam bez względu na liczbę wyświetlanych cyfr. Najbardziej popularnymi rozmiarami wyświetlania są 6, 8, 12 i 14. Podczas tworzenia tabeli istnieje także możliwość określenia dowolnej maksymalnej liczby wyświetlanych cyfr, ale w przypadku wartości 0 lub wartości przekraczającej 14 kolumna zostanie potraktowana jako TIMESTAMP(14). Nieparzyste wartości *MM* w zakresie od 1 do 13 traktowane są jako wyższa o jeden liczba parzysta.

Kolumny TIMESTAMP przechowują poprawne wartości z zastosowaniem pełnej dokładności, z którą wartość została zadeklarowana, bez względu na rozmiar wyświetlania. Prowadzi to do kilku następstw:

- Należy zawsze deklarować rok, miesiąc i dzień, nawet gdy kolumna jest typu TIMESTAMP(4) lub TIMESTAMP(2). W przeciwnym razie wartość nie będzie poprawną datą i przechowane zostanie 0.
- Użycie instrukcji ALTER TABLE do poszerzenia zbyt wąskiej kolumny TIMESTAMP spowoduje wyświetlenie poprzednio ukrytych informacji.
- Podobnie, zwężenie kolumny TIMESTAMP nie spowoduje utraty informacji. Po prostu mniej informacji będzie wyświetlanych.
- Mimo że wartości TIMESTAMP są przechowywane z pełną dokładnością, jedyną funkcją, której działanie bezpośrednio dotyczy faktycznie przechowywanych wartości, jest UNIX_TIMESTAMP(). Inne funkcje operują na sformatowanych wartościach odczytanych. Oznacza to, że takich funkcji, jak HOUR() lub SECOND() można używać tylko wtedy, gdy sformatowana wartość znacznika zawiera stosowną część wartości TIMESTAMP. Część GG kolumny TIMESTAMP nie jest na przykład wyświetlana, jeśli rozmiar wyświetlania nie przekracza 10. Z tego powodu próba użycia funkcji HOUR() na wartościach TIMESTAMP o skróconym formacie wyświetlania zwraca niezrozumiały wynik.

4.3.1.2. Właściwości TIMESTAMP począwszy od wersji 4.1 MySQL

Od wersji 4.1.0 właściwości TIMESTAMP różnią się od tych z wcześniejszych wersji MySQL.

- Kolumny TIMESTAMP są wyświetlane w tym samym formacie, w którym wyświetlane są kolumny DATETIME.
- Szerokości wyświetlania nie są obsługiwane w sposób opisany w poprzednim podpunkcie. Innymi słowy, nie można używać takich deklaracji, jak TIMESTAMP(2) czy TIMESTAMP(4).

Dodatkowo, jeśli serwer MySQL pracuje w trybie MAXDB, typ TIMESTAMP jest identyczny z typem DATETIME. Jeśli serwer działa w trybie MAXDB w momencie tworzenia tabeli, wszystkie kolumny TIMESTAMP zostają utworzone jako kolumny DATETIME. W rezultacie stosowany jest dla nich format wyświetlania DATETIME, ten sam zakres wartości, a ich aktualizacja nie jest wykonywana automatycznie.

MySQL można uruchamiać w trybie MAXDB od wersji 4.1.1. Aby tego dokonać, należy uruchomić serwer SQL z opcją --sql-mode=MAXDB lub podczas pracy serwera zmodyfikować zmienną globalną sql_mode:

```
mysql> SET GLOBAL sql_mode=MAXDB;
```

Aby serwer działał w trybie MAXDB dla połączenia danego klienta, wystarczy skorzystać z instrukcji:

```
mysql> SET SESSION sql_mode=MAXDB;
```

4.3.2. Typ TIME

Wartości TIME pobierane są i wyświetlane przez MySQL w formacie 'GG:MM:SS' (lub 'GGG:MM:SS' dla godzin o większych wartościach). Wartości TIME należą do zakresu od '-838:59:59' do '838:59:59'. Element reprezentujący godzinę może przybierać tak duże wartości, ponieważ typ TIME może służyć nie tylko do reprezentowania godzin w ciągu doby (która nie może przekroczyć 24 godzin), ale również upływającego czasu lub przedziału czasowego między dwoma zdarzeniami (a ten może znacznie przekroczyć 24 godziny, a nawet być liczbą ujemną).

Wartości TIME można deklarować w wielu formatach:

- Jako łańcuch w formacie 'D GG:MM:SS.ułamek'. Można także użyć jednego z nieco prostszych formatów: 'GG:MM:SS.ułamek', 'GG:MM:SS', 'GG:MM', 'D GG:MM:SS', 'D GG:MM', 'D GG' lub 'SS'. Parametr *D* reprezentuje dni i może przybierać wartość od 0 do 34. Należy zauważyć, że MySQL nie przechowuje jeszcze części ułamkowej.
- Jako łańcuch bez znaków rozdzielających elementy godziny w formacie 'GGMMSS' przy założeniu, że reprezentowana przez niego godzina istnieje. '101112' należy na przykład odczytać jako '10:11:12', natomiast '109712' jest wartością niepoprawną (część reprezentująca liczbę minut jest bezsensowna) i zostanie przekształcona na '00:00:00'.
- Jako liczba w formacie GGMMSS przy założeniu, że reprezentowana przez niego godzina istnieje, na przykład 101112 należy odczytać jako '10:11:12'. Zrozumiałe są także następujące formaty alternatywne: SS, MMSS, GGMMSS, GGMMSS.ułamek. Należy zauważyć, że MySQL nie przechowuje jeszcze części ułamkowej.
- Jako wynik funkcji, która zwraca wartość akceptowaną dla typu TIME — może nią być na przykład funkcja CURRENT_TIME.

W przypadku wartości TIME, zadeklarowanych jako łańcuchy zawierające elementy rozdzielające części godziny, nie ma potrzeby podawania dwóch cyfr dla wartości godziny, minuty i sekundy, mniejszych niż 10. Wartość '8:3:2' jest tym samym, czym jest '08:03:02'.

Należy uważać w przypadku przypisywania do kolumn TIME godzin o skróconej postaci. Podczas interpretacji wartości bez dwukropków zakłada się, że cyfry na jej prawym końcu reprezentują sekundy (MySQL interpretuje wartości TIME jako upływający czas, a nie jako godzinę dnia). Można by sądzić na przykład, że '1112' i 1112 oznaczają godzinę '11:12:00' (12 minut po 11), ale MySQL zinterpretuje je jako '00:11:22' (11 minut, 12 sekund). Podobnie '12' i 12 są interpretowane jako wartość '00:00:12'. Wartości z dwukropkami są zawsze traktowane jako godzina dnia, czyli '11:12' będzie oznaczać '11:12:00', a nie '00:11:12'.

Wartości, które są poza zakresem TIME, ale oprócz tego są poprawne, zostają przycięte do najbliższej granicy zakresu. Wartości '-850:00:00' i '850:00:00' na przykład zostaną przekształcone na '838:59:59' i '838:59:59'.

Niepoprawne wartości TIME zostają zapisane jako '00:00:00'. Warto odnotować, że ponieważ wartość '00:00:00' jest sama w sobie poprawną wartością TIME, patrząc na nią w tabeli, nie można ustalić, czy w ten sposób została zadeklarowana wartość oryginalna, czy też jest to wartość nieprawidłowa.

4.3.3. Typ YEAR

Typ YEAR jest typem jednobajtowym, służącym do przechowywania wartości lat.

MySQL pobiera i wyświetla wartości YEAR w formacie RRRR. Zakres wynosi od 1901 do 2155.

Wartości YEAR można deklarować za pomocą różnych formatów:

- W postaci czterocyfrowego łańcucha w przedziale od '1901' do '2155'.
- W postaci czterocyfrowej liczby w przedziale od 1901 do 2155.
- W postaci dwucyfrowego łańcucha w przedziale od '00' do '99'. Wartości z zakresu od '00' do '69' i od '70' do '99' są przekształcane na wartości YEAR należące do zakresów od 2001 do 2069 i od 1970 do 1999. Należy jednak pamiętać, że zakres dla liczb dwucyfrowych różni się trochę od zakresu dla dwucyfrowych łańcuchów, ponieważ nie można zadeklarować zera jako liczby, by zostało zinterpretowane jako 2000. Wstawienie wartości 00 do czterocyfrowej kolumny YEAR spowoduje przechowanie jej pod postacią 0000, a nie na 2000. Aby wartość została rozpoznana jako rok 2000, należy przesłać ją w postaci łańcucha '0' lub '00'.
- W postaci wyniku funkcji, która zwraca wartość dopuszczalną dla typu YEAR, na przykład funkcja NOW().

Niepoprawne wartości YEAR zostają przechowane pod postacią 0000.

4.3.4. Problem roku 2000 a typy związane z datą

Sam MySQL jest całkowicie odporny na problem roku 2000 (patrz punkt 1.2.5), jednak wartości wejściowe, dostarczane serwerowi MySQL, takie być nie muszą. Wszystkie daty z rokiem, wprowadzane w postaci dwucyfrowej, są niejednoznaczne, ponieważ nie wiadomo, o które stulecie chodzi. Takie wartości muszą być przekształcone na formę czterocyfrową, ponieważ wewnętrznie lata przechowywane są przez MySQL w pełnej, czterocyfrowej postaci.

Dla typów `DATETIME`, `TIMESTAMP` i `YEAR` daty interpretowane są niejednoznacznie, na podstawie następujących zasad:

- Wartości z zakresu `00-69` przekształcane są na wartości `2000-2069`.
- Wartości z zakresu `70-99` przekształcane są na wartości `1970-1999`.

Należy pamiętać, że te reguły pozwalają jedynie domyślić się, co oznaczają wprowadzone przez użytkownika wartości. W przypadku uzyskania wartości, która nie jest poprawna, należy podać rok w pełnej, czterocyfrowej postaci.

Za pomocą instrukcji `ORDER BY` można poprawnie sortować wartości `TIMESTAMP` i `YEAR`, w których rok jest podany w postaci dwucyfrowej.

Niektóre funkcje, takie jak `MIN()` i `MAX()` przekształcają wartości `TIMESTAMP` lub `YEAR` na liczby. To oznacza, że w przypadku wartości z rokiem, podanym w postaci dwucyfrowej, funkcje te dadzą błędny wynik. Aby tego uniknąć, należy przekształcić wartości `YEAR` lub `TIMESTAMP` na format, w którym rok jest w postaci czterocyfrowej, lub skorzystać w następujący sposób z funkcji `DATE_ADD`: `MIN(DATE_ADD(timestamp, INTERVAL 0 DAYS))`.

4.4. Typy łańcuchowe

Do typów łańcuchowych zalicza się `CHAR`, `VARCHAR`, `BLOB`, `TEXT`, `ENUM` i `SET`. W tym podrozdziale przedstawione zostanie ich przeznaczenie, a także omówione będą metody wykorzystywania tych typów w zapytaniach.

4.4.1. Typy CHAR i VARCHAR

Typy `CHAR` i `VARCHAR` są do siebie podobne. Różnią się tylko sposobem, w jaki są przechowywane i pobierane.

Długość kolumny `CHAR` jest stała i deklaruje się ją podczas tworzenia tabeli. Maksymalna długość kolumny może być wartością z przedziału od 0 do 255 (w wersjach poprzedzających MySQL 3.23 długość `CHAR` mogła wynosić od 1 do 255). Wartości `CHAR` podczas zapisywania uzupełniane są z prawej strony odpowiednią liczbą spacji. Spacje te są jednak odcinane w momencie pobierania ich przez klienta.

Wartości w kolumnach VARCHAR są łańcuchami o zmiennej długości. Można zadeklarować kolumnę VARCHAR o dowolnej długości z przedziału od 0 do 255, podobnie jak w przypadku kolumn CHAR (w wersjach poprzedzających MySQL 4.0.2 długość VARCHAR mogła wynosić od 1 do 255 znaków). W przeciwieństwie jednak do typu CHAR, wartości VARCHAR są przechowywane tylko w tylu bajtach, ile potrzeba ich do zapisania łańcucha; dodatkowo jeden bajt przeznaczony jest do zanotowania długości. Wartości nie są dopełniane spacjami, natomiast znajdujące się na końcu spacje są usuwane, gdy wartość jest zapisywana w tabeli. To odróżnia MySQL od specyfikacji typu VARCHAR w standardzie SQL.

Podczas zapisywania lub pobierania tych typów nie jest wykonywana także żadna konwersja związana z wielkością liter.

Jeśli do kolumny CHAR lub VARCHAR przypisze się wartość przekraczającą maksymalną długość kolumny, wartość ta zostanie obcięta w celu jej dopasowania.

Jeśli użytkownik potrzebuje kolumny, dla której nie są usuwane końcowe spacje, powinien zastanowić się nad zastosowaniem typu BLOB lub TEXT. Aby przechować wartości binarne (takie jak wynik funkcji szyfrującej lub kompresującej), które mogą zawierać przypadkowe wartości bajtów, zamiast CHAR lub VARCHAR należy użyć raczej kolumny BLOB. To pozwoli uniknąć potencjalnych problemów z usuwaniem końcowych spacji, które mogłyby zmienić wartość danych.

W poniższym zestawieniu przedstawiono różnice między tymi dwoma typami kolumn i pokazano rezultat zapisania różnych wartości łańcuchowych w kolumnach CHAR(4) i VARCHAR(4):

Wartość	CHAR(4)	Wymagana pamięć	VARCHAR(4)	Wymagana pamięć
''	' '	4 bajty	''	1 bajt
'ab'	'ab '	4 bajty	'ab'	3 bajty
'abcd'	'abcd'	4 bajty	'abcd'	5 bajtów
'abcdefgh'	'abcd'	4 bajty	'abcd'	5 bajtów

Wartości pobrane z kolumn CHAR(4) i VARCHAR(4) będą w obu przypadkach takie same, ponieważ z kolumn CHAR w momencie pobierania wartości z tabeli usuwane są końcowe spacje.

Począwszy od MySQL 4.1 wartości w kolumnach CHAR i VARCHAR są sortowane i porównywane według porządku sortowania zestawu znaków przypisanego do kolumny. W wersjach wcześniejszych sortowanie i porównania były wykonywane na podstawie porządku sortowania zestawu znaków serwera. Użytkownik może deklarować kolumnę z atrybutem BINARY, aby podczas sortowania i porównywania brana była pod uwagę wielkość liter, ponieważ użyte zostaną wartości liczbowe kodu, a nie porządek leksykalny. Atrybut BINARY nie wpływa na sposób, w jaki kolumna jest przechowywana i pobierana.

Od MySQL 4.1.0 typ CHAR BYTE jest aliasem typu CHAR BINARY. Jest to spowodowane wymogami kompatybilności.

Atrybut BINARY jest trwały. Oznacza to, że jeśli kolumna z atrybutem BINARY zostanie użyta w wyrażeniu, całe wyrażenie będzie traktowane jako wartość BINARY.

Od wersji 4.1.0 można określać dla typu CHAR atrybut ASCII. Przypisuje on zestaw znaku latin1.

Od wersji 4.1.0 można określać dla typu CHAR atrybut UNICODE. Przypisuje on zestaw znaku ucs2.

Podczas tworzenia tabeli typ kolumny CHAR lub VARCHAR może zostać zmieniony bez informowania o tym użytkownika (patrz podpunkt 6.2.5.2).

4.4.2. Typy BLOB i TEXT

BLOB jest skrótem angielskiej nazwy *binary large object* (wielki obiekt binarny). Cztery typy BLOB: TINYBLOB, BLOB, MEDIUMBLOB i LONGBLOB różnią się tylko maksymalną długością wartości, które mogą przechowywać (patrz podrozdział 4.5).

Cztery typy TEXT: TINYTEXT, TEXT, MEDIUMTEXT i LONGTEXT odpowiadają czterem typom BLOB i mają te same maksymalne długości i wymagania co do pamięci.

Kolumny typu BLOB są traktowane jako łańcuchy binarne. Kolumny TEXT są traktowane zgodnie z ich zestawem znaków. Podczas sortowania i porównywania wartości BLOB nie jest brana pod uwagę wielkość liter. Od wersji 4.1 MySQL operacje sortowania i porównywania wartości w kolumnach typu TEXT są wykonywane według porządku sortowania zestawu znaków przypisanego do kolumny. Wcześniej sortowanie i porównywanie tych kolumn było wykonywane na podstawie porządku sortowania zestawu znaków serwera.

Podczas zapisu i pobierania nie jest wykonywana żadna konwersja związana z wielkością liter.

Jeśli do kolumny BLOB lub TEXT zostanie przypisana wartość, która przekroczy maksymalną długość tego typu kolumny, wartość zostanie przycięta.

W pewnym sensie kolumnę TEXT można traktować jako kolumnę VARCHAR o dowolnej wielkości. Podobnie kolumnę BLOB można traktować jako kolumnę VARCHAR BINARY. Typy BLOB i TEXT różnią się od typów CHAR i VARCHAR następującymi cechami:

- Kolumny BLOB i TEXT mogą mieć indeksy dopiero od wersji 3.23.2 MySQL. Starsze wersje programu nie obsługiwały indeksowania tych typów kolumn.
- Dla indeksów na kolumnach BLOB i TEXT należy określić długość przedrostka indeksu. Dla typów CHAR i VARCHAR jest to opcjonalne.
- Dla kolumn BLOB i TEXT nie ma usuwania końcowych spacji podczas zapisywania lub pobierania wartości. To odróżnia je od kolumn CHAR (spacje końcowe są usuwane w momencie pobierania wartości) i od kolumn VARCHAR (spacje końcowe są usuwane w momencie zapisywania wartości).
- Kolumny typu BLOB i TEXT nie mogą mieć wartości DEFAULT.

Od MySQL 4.1.0 typ LONG i LONG VARCHAR odpowiada typowi danych MEDIUMTEXT. Jest to mechanizm dodany dla kompatybilności.

Interfejs Connector/ODBC deklaruje wartości BLOB jako LONGVARBINARY, a wartości TEXT jako LONGVARCHAR.

Ponieważ wartości BLOB i TEXT mogą być niezwykle długie, podczas ich stosowania można spotkać się z pewnymi ograniczeniami:

- Jeśli na kolumnie typu BLOB lub TEXT ma być wykonana operacja GROUP BY lub ORDER BY, wartość kolumny należy przekształcić na obiekt o stałej długości. Standardowo wykonuje się to za pomocą funkcji SUBSTRING, na przykład:

```
mysql> SELECT comment FROM nazwa_tabeli,SUBSTRING(comment,20) AS substr
    ->              ORDER BY substr;
```

  W przeciwnym razie podczas sortowania użyte zostaną tylko pierwsze bajty max_sort_length kolumny. Domyślna wartość zmiennej max_sort_length wynosi 1024 i można ją zmienić przy użyciu opcji --max_sort_length podczas uruchamiania serwera mysqld.

  Istnieje możliwość wykonania operacji grupowania na wyrażeniu, które jest związane z wartościami BLOB lub TEXT — za pomocą aliasu lub przez określenie położenia kolumny:

```
mysql> SELECT id,SUBSTRING(kolumna_blob,1,100) AS b
   ->  FROM nazwa_tabeli GROUP BY b;
mysql> SELECT id,SUBSTRING(kolumna_blob,1,100)
   ->  FROM nazwa_tabeli GROUP BY 2;
```

- Maksymalny rozmiar obiektu BLOB lub TEXT określa jego typ, ale o tym, jaką największą wartość można w rzeczywistości przesłać między klientem a serwerem, decyduje ilość dostępnej pamięci i rozmiar buforów komunikacyjnych. Rozmiar bufora wiadomości można zastąpić, zmieniając wartość zmiennej max_allowed_packet, ale trzeba to zrobić dla serwera i programu klienckiego. Na przykład na zmianę wartości opcji max_allowed_packet po stronie klienta pozwalają mysql i mysqldump.

Każda wartość BLOB i TEXT jest wewnętrznie reprezentowana przez obiekty o oddzielnie przydzielanej pamięci. To odróżnia je od wszystkich innych typów kolumn, dla których pamięć jest przydzielana raz na całą kolumnę w momencie otwarcia tabeli.

4.4.3. Typ Enum

ENUM to obiekt łańcuchowy z wartością wybieraną z listy dozwolonych wartości, które są w sposób jawny wyliczone w deklaracji kolumny podczas tworzenia tabeli.

W pewnych warunkach wartość może być również pustym łańcuchem ('') lub wartością NULL:

- Jeśli do kolumny ENUM zostanie wstawiona niepoprawna wartość (czyli łańcuch nienależący do listy dopuszczalnych wartości), to zostanie zastąpiona pustym

łańcuchem reprezentującym błędną wartość. Można go odróżnić od zwykłego pustego łańcucha, ponieważ jego wartość liczbowa jest równa 0. Więcej o tym za chwilę.

- Jeśli w deklaracji kolumny ENUM dopuszczone będzie użycie wartości NULL, będzie ona wartością prawidłową i wtedy wartością domyślną jest NULL. Jeśli kolumna ENUM zostanie zadeklarowana jako NOT NULL, jej wartością domyślną będzie pierwszy element na liście dozwolonych wartości.

Każda wartość wyliczenia ma indeks:

- Wartości należące do listy dopuszczalnych elementów w deklaracji kolumny są ponumerowane, rozpoczynając od 1.
- Wartość indeksu pustego łańcucha reprezentującego błąd wynosi 0. Oznacza to, że poniższa instrukcja SELECT pomoże odszukać wiersze, którym przypisano niepoprawne wartości ENUM:

```
mysql> SELECT * FROM nazwa_tabeli WHERE kol_enum=0;
```

- Indeksem wartości NULL jest NULL.

Kolumna zadeklarowana jako ENUM ('jeden', 'dwa', 'trzy') może na przykład zawierać dowolną z przedstawionych w zestawieniu wartości. Dodatkowo zaprezentowana została wartość indeksu każdej wartości:

Wartość	Indeks
NULL	NULL
''	0
'jeden'	1
'dwa'	2
'trzy'	3

Wyliczenie może zawierać maksymalnie 65 535 elementów.

Począwszy od wersji 3.23.51 MySQL wartości należące do ENUM mają automatycznie usuwane spacje końcowe w momencie tworzenia tabeli.

Podczas przypisywania wartości do kolumny ENUM nieistotna jest wielkość liter. Wartości pobrane później z takiej kolumny są jednak wyświetlane przy użyciu liter o wielkości użytej w definicji kolumny.

Jeśli wartość ENUM zostanie pobrana w postaci liczbowej, zwrócony zostanie indeks wartości kolumny. W następujący sposób można na przykład pobrać wartość liczbową z kolumny ENUM:

```
mysql> SELECT kol_enum+0 FROM nazwa_tabeli;
```

Jeśli do kolumny ENUM zostanie wpisana liczba, będzie ona traktowana jako wartość indeksu, a przechowywana wartość będzie elementem wyliczenia o tym indeksie (nie zadziała to jednak z instrukcją LOAD DATA, ponieważ traktuje ona wszystkie dane wej-

ściowe jako łańcuchy). Nie zaleca się deklarowania kolumn ENUM z wartościami wyliczenia, które wyglądem przypominają liczby, ponieważ może to łatwo doprowadzić do pomyłki. Poniższa kolumna zawiera na przykład elementy wyliczenia, których wartości łańcucha — '0', '1' i '2' — mają wartości liczbowe indeksu równe 1, 2 i 3:

```
liczby ENUM('0', '1', '2')
```

Sortowanie wartości ENUM odbywa się na podstawie porządku, w którym elementy wyliczenia zostały wymienione w deklaracji kolumny (innymi słowy wartości ENUM są sortowane według przypisanych im wartości indeksu), na przykład 'a' zostanie ustawione przed 'b' dla ENUM('a', 'b'), ale 'b' stanie przed 'a' dla ENUM ('b', 'a'). Pusty łańcuch zostaje ustawiony przed łańcuchem niepustym, wartości NULL zostaną umiejscowione przed wszystkimi innymi wartościami wyliczenia. Aby zapobiec nieoczekiwanym wynikom, należy w deklaracji kolumny wymienić wartości wyliczenia listy ENUM w porządku alfabetycznym. Można również zastosować GROUP BY CAST(*kol* AS VARCHAR) lub GROUP BY CONCAT(*kol*), aby upewnić się, że kolumna zostanie uporządkowana alfabetycznie, a nie według liczb w indeksie.

Aby ustalić wszystkie możliwe wartości dla kolumny ENUM, należy użyć instrukcji SHOW COLUMNS FROM *nazwa_tabeli* LIKE *kol_enum* i przetworzyć definicję ENUM w drugiej kolumnie danych wyjściowych.

4.4.4. Typ SET

SET to obiekt łańcuchowy składający się z zera lub większej liczby wartości, z których każda musi być wybrana z listy dozwolonych wartości, wyliczonych w deklaracji kolumny podczas tworzenia tabeli. W przypadku wartości kolumny typu SET, składających się z wielu elementów zestawu, elementy te są oddzielane przecinkami (,). Z tego powodu wartości elementów SET nie mogą zawierać przecinków.

Kolumna zadeklarowana na przykład jako SET('jeden', 'dwa') NOT NULL może mieć jedną z następujących wartości:

```
''
'jeden'
'dwa'
'jeden, dwa'
```

Typ SET może zawierać maksymalnie 64 różne elementy.

Od wersji 3.23.51 MySQL podczas tworzenia tabeli z elementów należących do zestawu wartości automatycznie usuwane są spacje końcowe.

W MySQL wartości SET przechowywane są w postaci liczbowej, w której bit mniej znaczący przechowywanej wartości odpowiada pierwszemu elementowi zestawu. Wartość SET pobrana w postaci liczbowej ma ustawione bity odpowiadające elementom zestawu, które tworzą wartość kolumny. Można na przykład pobrać wartości liczbowe z kolumny SET w następujący sposób:

```
mysql> SELECT kol_set+0 FROM nazwa_tabeli;
```

Jeśli liczba jest zapisana w kolumnie SET, bity włączone w binarnej reprezentacji liczby określają elementy zestawu w wartości kolumny. Dla kolumny zadeklarowanej jako SET('a', 'b', 'c', 'd') elementy mają następujące wartości dziesiętne i binarne:

Element SET	Wartość dziesiętna	Wartość binarna
'a'	1	0001
'b'	2	0010
'c'	4	0100
'd'	8	1000

Jeśli do takiej kolumny przypisze się wartość 9, odpowiada to binarnej wartości 1001, czyli wybrane są pierwszy i czwarty element wartości SET 'a' i 'd', dlatego w wyniku otrzymamy wartość 'a,d'.

W przypadku wartości zawierającej więcej niż jeden element SET, nie ma znaczenia kolejność, w jakiej elementy są wymienione podczas wstawiania wartości. Nie ma również znaczenia, ile razy dany element jest wymieniony w wartości. Kiedy później wartość jest pobierana, każdy element w wartości będzie pojawiać się raz z elementami wymienionymi zgodnie z porządkiem, w którym zostały podane podczas tworzenia tabeli. Jeśli kolumna została zadeklarowana jako SET('a','b','c','d'), wtedy wartości 'a,d', 'd,a' i 'd,a,a,d,d' pojawią się w momencie pobrania jako 'a,d'.

Jeśli do kolumny SET zostanie przypisana wartość nieobsługiwana., zostanie ona zignorowana.

Wartości SET są sortowane według porządku liczbowego. Wartości NULL są ustawiane przed niezerowymi wartościami SET.

Zazwyczaj element zestawu wyszukuje się za pomocą funkcji FIND_IN_SET() lub operatora LIKE:

```
mysql> SELECT * FROM nazwa_tabeli WHERE FIND_IN_SET('wartość',kol_set)>0;
mysql> SELECT * FROM nazwa_tabeli WHERE kol_set LIKE '%wartość%';
```

Pierwsza instrukcja znajduje wiersze, w których kolumna *kol_set* zawiera element zestawu *wartość*. Druga jest podobna, ale nie taka sama: odszukuje wiersze, w których kolumna *kol_set* zawiera wartość w dowolnym miejscu, nawet jako podłańcuch innego elementu zestawu.

Poniższe instrukcje są również prawidłowe:

```
mysql> SELECT * FROM nazwa_tabeli WHERE kol_set & 1;
mysql> SELECT * FROM nazwa_tabeli WHERE kol_set = 'war1,war2';
```

Pierwsza z tych instrukcji powoduje wyszukanie wartości zawierających pierwszy element zestawu, druga — wyszukanie elementu, który dokładnie pasuje do wzorca. Z porównaniami drugiego typu należy uważać. Porównywanie wartości zestawu do *'war1,war2'* zwróci inny wynik niż porównanie wartości do *'war2,war1'*. Wartości należy podawać dokładnie w tej samej kolejności, w której są zadeklarowane w definicji kolumny.

Aby ustalić wszystkie możliwe wartości dla kolumny SET, należy użyć instrukcji SHOW COLUMNS FROM *nazwa_tabeli* LIKE *kol_set* i przetworzyć definicję SET w drugiej kolumnie wyniku.

4.5. Rozmiar pamięci potrzebnej dla typów kolumn

Rozmiar pamięci dla każdego z typów kolumn, obsługiwanych przez MySQL, został wymieniony według kategorii.

Maksymalny rozmiar wiersza w tabeli MyISAM wynosi 65 534 bajty. Każda kolumna typu BLOB lub TEXT stanowi tylko 5 do 9 bajtów tego rozmiaru.

Jeśli tabela MyISAM lub ISAM zawiera jakieś typy kolumn o zmiennej długości, także i format rekordu będzie zmiennej długości. Podczas tworzenia tabeli, MySQL może spowodować w pewnych okolicznościach zmianę typu kolumny zmiennej długości na kolumnę stałej długości lub na odwrót (patrz podpunkt 6.2.5.2).

Rozmiar pamięci potrzebnej dla typów liczbowych

Typ kolumny	Wymagany rozmiar pamięci
TINYINT	1 bajt
SMALLINT	2 bajty
MEDIUMINT	3 bajty
INT, INTEGER	4 bajty
BIGINT	8 bajtów
FLOAT(*p*)	4 bajty — jeśli 0<= *p* <= 24, 8 bajtów — jeśli 25 <= *p* <= 53
FLOAT	4 bajty
DOUBLE [PRECISION], REAL	8 bajtów
DECIMAL(*M*,*D*), NUMERIC(*M*,*D*)	*M*+2 bajty — jeśli *D* > 0, *M*+1 bajt — jeśli *D* = 0 (*D*+2 — jeśli *M*<*D*)

Rozmiar pamięci potrzebnej dla typów związanych z datą i czasem

Typ kolumny	Wymagany rozmiar pamięci
DATE	3 bajty
DATETIME	8 bajtów
TIMESTAMP	4 bajty
TIME	3 bajty
YEAR	1 bajt

Rozmiar pamięci potrzebnej dla typów łańcuchowych

Typ kolumny	Wymagany rozmiar pamięci
`CHAR(M)`	*M* bajtów, `0 <= M <= 255`
`VARCHAR(M)`	*L*+1 bajtów, gdzie *L* <= *M* i 0<= *M* <=255
`TINYBLOB`, `TINYTEXT`	*L*+1 bajtów, gdzie $L < 2^8$
`BLOB`, `TEXT`	*L*+2 bajty, gdzie $L < 2^{16}$
`MEDIUMBLOB`, `MEDIUMTEXT`	*L*+3 bajty, gdzie $L < 2^{24}$
`LONGBLOB`, `LONGTEXT`	*L*+4 bajty, gdzie $L < 2^{32}$
`ENUM('wartość1', 'wartość2',...)`	1 lub 2 bajty w zależności od liczby wartości wyliczenia (maksymalnie 65 535)
`SET('wartość1', 'wartość2',...)`	1, 2, 3, 4 lub 8 bajtów w zależności od liczby elementów zestawu (maksymalnie 64)

Typy `VARCHAR`, `BLOB` i `TEXT` mają zmienną długość. Dla każdego ilość pamięci potrzebnej do przechowania wartości będzie zależeć od faktycznej długości wartości kolumny (reprezentowanej w ostatniej tabeli przez *L*), a nie od maksymalnego możliwego rozmiaru danego typu. Kolumna `VARCHAR(10)` na przykład może przechowywać łańcuch o maksymalnej długości 10 znaków. Faktyczna ilość wymaganej pamięci to długość łańcucha (*L*) plus 1 bajt do zapisania długości łańcucha. Dla łańcucha `'abcd'` parametr *L* jest równy 4, a wymagany rozmiar pamięci to 5 bajtów.

Typy `BLOB` i `TEXT` wymagają 1, 2, 3 lub 4 bajtów do zapisu długości wartości kolumny w zależności od maksymalnej możliwej długości tego typu (patrz punkt 4.4.2).

Rozmiar obiektu `ENUM` jest ustalany na podstawie liczby różnych wartości wyliczenia. Przy wyliczeniach do 255 możliwych wartości używany jest jeden bajt, a przy wyliczeniach do 65 535 wartości używane są 2 bajty (patrz punkt 4.4.3).

Rozmiar obiektu `SET` jest określany przez liczbę różnych elementów zestawu. Jeśli zestaw ma rozmiar *N*, obiekt zajmuje `(N+7)/8` bajtów, którą to wartość zaokrągla się do 1, 2, 3, 4 lub 8 bajtów. Obiekt `SET` może mieć do 64 elementów (patrz punkt 4.4.4).

4.6. Wybieranie odpowiedniego typu dla kolumny

Aby jak najbardziej wydajnie korzystać z pamięci, należy we wszystkich przypadkach starać się stosować typ, który będzie jak najbardziej dokładny. Jeśli na przykład w kolumnie liczb całkowitych będą przechowywane wartości z zakresu od `1` do `99999`, najlepszym wyborem będzie typ `MEDIUMINT UNSIGNED`. Ze wszystkich typów, które reprezentują wszystkie wymagane wartości, zajmuje najmniej pamięci.

Powszechnym problemem jest dokładne przedstawianie wartości walutowych. W MySQL należy do tego używać typu DECIMAL, za pomocą którego wartości zapisywane są w formie łańcucha. Dzięki temu nie traci się na dokładności (mimo to obliczenia z wartościami DECIMAL mogą być wciąż wykonywane przy użyciu operacji działających na liczbach z podwójną precyzją). Jeśli dokładność nie jest zbyt ważna, być może wystarczająco dobrym typem będzie DOUBLE.

Aby uzyskać większą dokładność, można zawsze przekształcić wartość na typ stałoprzecinkowy, przechowywany w BIGINT. Pozwoli to wykonywać wszystkie obliczenia na liczbach całkowitych i tylko w razie potrzeby przekształcać wynik z powrotem na wartości zmiennoprzecinkowe.

4.7. Wykorzystywanie typów kolumn z innych mechanizmów baz danych

Aby ułatwić użytkownikowi zastosowanie kodu pochodzącego od innych producentów, możliwe jest odwzorowanie typów kolumn w sposób przedstawiony w poniższej tabeli. Odwzorowania te upraszczają operację importu do MySQL definicji tabel z innych mechanizmów baz danych:

Typ innych producentów	Typ MySQL
BINARY(*M*)	CHAR(*M*) BINARY
CHAR VARYING(*M*)	VARCHAR(*M*)
FLOAT4	FLOAT
FLOAT8	DOUBLE
INT1	TINYINT
INT2	SMALLINT
INT3	MEDIUMINT
INT4	INT
INT8	BIGINT
LONG VARBINARY	MEDIUMBLOB
LONG VARCHAR	MEDIUMTEXT
LONG	MEDIUMTEXT (od MySQL 4.1.0)
MIDDLEINT	MEDIUMINT
VARBINARY(*M*)	CHAR(*M*) BINARY

Odwzorowanie typów kolumn jest wykonywane podczas tworzenia tabeli, po czym oryginalne deklaracje typu zostają usunięte. Jeśli użytkownik utworzy tabelę z typami stosowanymi przez innych producentów, po czym wyda polecenie DESCRIBE *nazwa_tabeli*, zostanie przedstawiona struktura tabeli z odpowiadającymi im typami MySQL.