Bazy danych
przed 5 wykładem

Andrzej Giniewicz

05.04.2024

W tym tygodniu uzupełnimy informacje o typach danych, między innymi o typy do zapisu dat oraz czasu, wyliczeń oraz zbiorów.

1 Typy do zapisu dat i czasu

W MariaDB i MySQL istnieje wiele typów do zapisu dat i czasu. Istnieją typy do zapisu roku, całej daty, czasu, czasu z datą, przy czym wiele z nich ma więcej niż jeden wariant. W standardowym SQL są tylko trzy typy do dat: `DATE`, `TIME` oraz `TIMESTAMP`, zatem spora część opisanych tu typów jest specyficzna dla MySQL i MariaDB. Większość silników baz danych ma jednak podobne dodatkowe typy, które mają zbliżone działanie do opisanych tutaj. Niemniej jednak należy zawsze sprawdzić dokumentację. W opisie poniżej, gdy mowa o SQL, mamy na myśli dialekt MariaDB.

Oprócz sposobu zapisu, w SQL jest wiele funkcji do obsługi dat. Ich listę można sprawdzić w dokumentacji na stronie `https://mariadb.com/kb/en/date-time-functions/`. Znajdują się tam zarazem funkcje pozwalające formatować daty, wyciągać fragmenty dat, na przykład rok, miesiąc, dzień, tydzień roku, dzień tygodnia, albo wykonywać operacje na datach, na przykład odejmować daty, aby otrzymać przedział czasu lub dodawać przedział czasu do daty, aby otrzymać nową datę.

1.1 Rok

W SQL istnieją dwa warianty typu do zapisu roku, jest to `YEAR(2)` oraz `YEAR(4)`. Oba typy zajmują w pamięci 1 bajt. Wariant dwucyfrowy pozwala zapisać lata z zakresu 1970–2069, natomiast wariant czterocyfrowy zakres 1901–2155 oraz dodatkowo rok 0. Wariant dwucyfrowy jest niezalecany i może być wycofany w przyszłych wersjach serwera. Typ `YEAR` jest synonimem dla `YEAR(4)`, czyli wariantu czterocyfrowego. Jeśli potrzebujemy dat spoza tego zakresu, należy użyć innego typu danych do dat lub jednego z typów liczb całkowitych.

Wstawianie wartości do kolumn roku można wykonać na jeden z dwóch sposobów: za pomocą liczby albo za pomocą napisu. Można podawać:

- czterocyfrową liczbę, np. 2021,

- czteroznakowy napis, np. '2021',
- napis długości do 2 znaków — wartości od '0' do '69' są tłumaczone na lata 2000–2069, natomiast wartości od '70' do '99' na 1970–1999, wartość '00' i '0' jest tożsama,
- liczbę naturalną mniejszą od 100 w typie `YEAR(2)` — od 0 do 69 są tłumaczone na lata 2000–2069, natomiast wartości od 70 do 99 na 1970–1999,
- liczbę naturalną mniejszą od 100 w typie `YEAR(4)` — wartość 0 jest tłumaczona na rok 0, wartości od 1 do 69 są tłumaczone na lata 2001–2069, natomiast wartości od 70 do 99 na 1970–1999.

Na przykład rok 2021 możemy zapisać jako `2021`, `'2021'`, `21` lub `'21'`. Rok 2000 zapiszemy jako `2000`, `'2000'`, `'0'`, `'00'` lub (tylko w typie `YEAR(2)`) `0`.

1.2 Daty z miesiącem i dniem

Do zapisu dat służy standardowy typ `DATE`. Typ ten pozwala zapisać daty od pierwszego stycznia 1000 roku do 31 grudnia 9999 roku oraz specjalną nieistniejącą datę 0000-00-00 zwaną datą zerową. Do przechowywania daty z miesiącem i dniem SQL wykorzystuje 3 bajty.

Stałe dat wpisujemy za pomocą napisu lub liczby. Można podawać:

- datę postaci napisu 'YYYY/MM/DD', przy czym '/' można zastąpić innym znakiem interpunkcyjnym, na przykład '-', '.' lub pominąć;
- datę postaci 'YY/MM/DD', przy czym '/' można zastąpić innym znakiem interpunkcyjnym, a interpretacja roku napisu zapisanego za pomocą dwóch znaków jest taka, jak dla typu `YEAR(2)`;
- datę postaci liczby YYYYMMDD;
- datę postaci liczby YYMMDD, przy czym interpretacja roku napisu zapisanego za pomocą dwóch znaków jest taka, jak dla typu `YEAR(2)`.

Wszystkie poniższe napisy reprezentują datę 28 marca 2021:

- `20210328`,
- `210328`,
- `'2021/03/28'`,
- `'2021.03.28'`,
- `'2021:03!28'`,
- `'20210328'`,
- `'21-03-28'`,

- `'210328'`.

Wobec tego na przykład sprawdzając, czy data to 28 marca 2021 w wyrażeniu `WHERE` możemy na przykład napisać `WHERE data = '2021/03/28'` lub dowolną inną wersję. Wybór wersji zależy od tego, jaka forma daty jest najbardziej czytelna dla piszącego zapytania.

1.3 Czas

Do zapisu czasu używamy typu `TIME` lub `TIME(n)`, gdzie `n` jest liczbą naturalną od 0 do 6. Wartość `n` oznacza liczbę miejsc po przecinku w zapisie sekundowym. Na przykład `TIME(6)` oznacza precyzję milisekundową. Typ `TIME` jest równoważny do `TIME(0)`, czyli typowi czasu bez możliwości zapisu ułamkowych części sekund. `TIME(n)` zajmuje $3+\left\lceil \frac{n}{2} \right\rceil$ bajtów. Zmienna typu `TIME` pozwala zapisać zakres od −838 godzin, 59 minut, 59 sekund do 838 godzin, 59 minut, 59 sekund. Typ `TIME(6)` pozwala na zapis czasów w zakresie `-838:59:59.999999` do `838:59:59.999999`.

Wpisywanie czasu w SQL wymaga większej precyzji zapisu. Na przykład nie możemy dowolnie zmieniać znaków przestankowych tak, jak w datach. Minuty, godziny i sekundy oddzielamy dwukropkiem, sekundy od części sekund kropką, a ewentualne dni od godzin spacją. W zapisie poniżej D oznacza liczbę dni, H oznacza cyfrę godzin, M cyfrę minut, S cyfrę sekund, F cyfrę części sekundy. Prawidłowe formaty godzin to:

- liczba z zakresu −8385959.999999–8385959.999999, oznacza czas zapisany w ten sposób, że cyfra jedności i dziesiątek odpowiada za sekundy, setek i tysięcy za minutę, wyższe cyfry za godzinę, a opcjonalna wartość po przecinku za ułamkową część sekundy. Warto zwrócić uwagę, że po 59 w tym systemie następuje 100, a liczba 60 nie reprezentuje poprawnego czasu;
- napis `'D HH:MM:SS.FFFFFF'` — jeśli podajemy dni, godziny powinny być mniejsze niż 24,
- napis `'HHH:MM:SS.FFFFFF'` — w zapisie bez dni, godziny mogą być większe niż 23,
- napis `'HH:MM.FFFFFF'`,
- napis `'SS.FFFFFF'`,
- napis `'D HH:MM:SS'` — jeśli podajemy dni, godziny powinny być mniejsze niż 24,
- napis `'HHH:MM:SS'` — w zapisie bez dni, godziny mogą być większe niż 23,
- napis `'HH:MM'`,
- napis `'SS'`.

Przykładowo godzina rozpoczęcia zajęć z baz danych może być zapisana jako:

- `111500`,
- `'11:15:00'`,

- `'11:15:00.0'`,
- `'11:15:00.00'`,
- `'11:15:00.000'`,
- `'11:15:00.0000'`,
- `'11:15:00.00000'`,
- `'11:15:00.000000'`.

1.4 Data i czas

Datę i czas moglibyśmy przechowywać w dwóch kolumnach, jednej typu `DATE`, drugiej typu `TIME(n)`. Zajęłoby to $6 + \left\lceil \frac{n}{2} \right\rceil$ bajtów. Zwróćmy jednak uwagę, że gdy pamiętamy już datę, nie będziemy zwykle potrzebowali pamiętać godzin większych niż 23 i mniejszych od zera. W efekcie sporo miejsca w liczbie będzie niewykorzystane. Do zapisu daty z godziną, służy typ `DATETIME(n)`, który łączy typ `DATE` i `TIME(n)` ale zajmuje o jeden bajt mniej, niż oba te typy z osobna, czyli $5 + \left\lceil \frac{n}{2} \right\rceil$ bajtów. `DATETIME` jest skrótem dla `DATETIME(0)`, podobnie jak w typie `TIME`. Zwykle typu `DATETIME(n)` użyjemy do zapisu dat, czyli momentu w czasie, natomiast typu `TIME` do zapisu różnic pomiędzy dwoma typami `DATETIME`, czyli do zapisu przedziału czasu — wtedy zarazem ujemne wartości godzin, jak i większe od 23 mają sens.

Wartości wprowadzane do typu `DATETIME` powinny być wprowadzone jako napis lub liczba, przy czym w przypadku napisów należy połączyć obie części spacją, natomiast w przypadku liczb, dopisać datę z przodu jako kolejne cyfry. Na przykład datę najbliższych zajęć z baz danych zapiszemy jako `'2021-03-31 11:15:00'` lub `20210331111500`. Jest to na tyle długi zapis, że format napisowy jest zalecany dla czytelności — w zapisie w formie liczby mającej 14 cyfr łatwo się pogubić.

Wewnętrznie SQL sprawdza daty i godziny za pomocą lokalnej strefy czasowej ustawionej na serwerze. Jeśli jakaś godzina nie istnieje, na przykład z powodu tego, że w strefie czasowej serwera była zmiana czasu, wartość `DATETIME` będzie nieprawidłowa. Prowadzi to do sytuacji, w której jedna wartość `DATETIME` jest prawidłowa na jednym serwerze, ale nieprawidłowa na innym serwerze uruchomionym w innym kraju, nawet jeśli wszystkie wersje oprogramowania są identyczne.

Oprócz `DATETIME` SQL posiada również typ `TIMESTAMP(n)`, który zajmuje o jeden bajt mniej niż `DATETIME`, czyli $4 + \left\lceil \frac{n}{2} \right\rceil$ bajtów. `TIMESTAMP` jest skrótem dla `TIMESTAMP(0)`, podobnie jak w typie `TIME` i `DATETIME`. Typ ten pamięta liczbę sekund, jaka upłynęła od `'1970-01-01 00:00:00'` w strefie czasowej Greenwich. Może przechowywać dowolne wartości od `'1970-01-01 00:00:00.000001'` do `'2038-01-19 03:14:07.999999'` oraz specjalną wartość `'0000-00-00 00:00:00'`, która może być wykorzystana jako flaga. Kolumny `TIMESTAMP` mają specjalne zachowanie, ponieważ jeśli wpisujemy dane do wiersza lub je aktualizujemy, to pierwsza kolumna typu `TIMESTAMP` w tabeli będzie się automatycznie aktualizować, jeśli nie podamy jej wartości. Oznacza to, że jeśli jej nie ustawimy lub ustawimy ją na `NULL`, SQL automatycznie wpisze w jej miejsce obecny czas w strefie czasowej UTC.

Data 1 stycznia 1970 roku nazywana jest początkiem ery Unixa. Na komputerach 32--bitowych maksymalny czas, jaki można było wyrazić, to właśnie 19 stycznia 2038, godzina 3:14:07 UTC. Po przekroczeniu tego czasu, licznik „przekręci się" i zacznie wskazywać 1 stycznia 1970 roku. Co ważne, problem ten został już rozwiązany w samym systemie operacyjnym, ale dużo oprogramowania, w tym znaczna część silników SQL (również MariaDB i MySQL) korzystają z 32 bitowego typu `TIMESTAMP`. Istnieje ryzyko, że systemy informatyczne korzystające z takich wartości mogą przestać działać lub zacząć dziwnie działać. Problem ten jest realnym problemem i może doprowadzić do awarii wielu systemów. Na szczęście twórcy oprogramowania są świadomi problemu i pracują nad rozwiązaniem, które do 2038 roku na pewno ujrzy światło dzienne. Jedynym problemem mogą być systemy działające bardzo długo, które nie były aktualizowane. Problem ten przypomina tak zwaną „pluskwę milenijną", czyli problem roku 2000. Istnieje termin „problem roku 2038", który może się właśnie objawić z powodu nadużywania typu `TIMESTAMP`. Jeśli uważamy, że rok 2038 będzie dla nas problemem, czyli na przykład spodziewamy się, że system informatyczny będzie działał dłużej niż 17 lat od teraz, unikajmy tego typu.

2 Typ wyliczeń

Typ wyliczeń, czyli `ENUM` w SQL jest synonimem dla typu `TINYINT UNSIGNED`, gdy korzystamy z mniej niż 256 wartości i `SMALLINT UNSIGNED`, gdy korzystamy z więcej niż 255 wartości. Oznacza to, że zajmuje jeden lub dwa bajty i może przechować 65535 różnych wartości. Typ ten jest skrótem pozwalającym zapamiętywać skończoną liczbę napisów za pomocą tylko dwóch bajtów. I tak na przykład możemy zdefiniować typ `ENUM('red', 'green', 'blue')`. Do zapisania tych typów jako napis, potrzebowalibyśmy minimum 5 bajtów dla typu `CHAR(5) CHARACTER SET ASCII`. Do zapisu tych samych wartości jako wyliczenie potrzeba jednego bajtu, co stanowi sporą oszczędność przy jednoczesnym zachowaniu czytelności nazw. Przy definicji jak z przykładu, SQL tłumaczy wartości na liczby w sposób opisany poniższą tabelą.

Wartość	Reprezentacja
`NULL`	`NULL`
`''`	`0`
`'red'`	`1`
`'green'`	`2`
`'blue'`	`3`

Liczby te będą użyte do porównywania, będą też użyte w wyrażeniach arytmetycznych oraz sortowaniu — napisy w typie wyliczeniowym są jedynie skrótem do wewnętrznej reprezentacji.

3 Typ bitowy i zbiory

Typ bitowy jest skrótem dla typów całkowitych. Może zajmować 1, 2, 3, 4 lub 8 bajtów, zależnie od wymaganej liczby bitów. W szczególności typ `BIT(n)` jest implementowany za pomocą najmniejszego z typów całkowitych mieszczących n bitów. Oznacza to, że w zależności od n będzie to.

Typ `BIT(n)`	Reprezentacja	Rozmiar
$1 \leqslant n \leqslant 8$	`TINYINT UNSIGNED`	1 bajt
$9 \leqslant n \leqslant 16$	`SMALLINT UNSIGNED`	2 bajty
$17 \leqslant n \leqslant 24$	`MEDIUMINT UNSIGNED`	3 bajty
$25 \leqslant n \leqslant 32$	`INT UNSIGNED`	4 bajty
$33 \leqslant n \leqslant 64$	`BIGINT UNSIGNED`	8 bajtów

Typ `BIT` jest synonimem dla `BIT(1)`. Wartości bitowe zapisujemy za pomocą napisów poprzedzonych literą `b`. Na przykład `b'101101'` jest wartością, którą możemy zapisać w typie `BIT(6)` lub większym. Najbardziej znaczący bit jest po lewej stronie, najmniej znaczący po prawej. Jeśli wypisujemy mniej znaków niż w typie, najbardziej znaczące bity zostaną wypełnione zerami, na przykład `b'101101'` w typie `BIT(8)` jest tożsama wartości `b'00101101'`. Na typach bitowych możemy wykonywać następujące operacje.

Operator	Znaczenie
`~`	negacja (bit po bicie)
`|`	suma logiczna (bit po bicie)
`&`	iloczyn logiczny (bit po bicie)
`^`	różnica symetryczna (bit po bicie)
`>>`	przesunięcie w prawo
`<<`	przesunięcie w lewo

Typ `BIT` może służyć do zapisu tak zwanych flag. Jeśli wprowadzimy umowę, że idąc od prawej bity oznaczają „prawo do odczytu", „prawo do zapisu", „prawo do skasowania" — wtedy wartość `b'011'` oznacza prawo do odczytu i zapisu, ale brak praw do kasowania. Taki system zapisu pozwala za pomocą zer i jedynek szybko zapisać, które „flagi" są ustawione, czyli na co pozwalamy. Również dzięki operacjom logicznym możemy sprawdzić, które flagi są ustawione. Aby sprawdzić, czy w zmiennej x zapisanej w standardzie z przykładu, ktoś ma uprawnienia do zapisu, wykorzystamy warunek `WHERE x & b'010' > 0`.

Koncepcyjnie flagi są podobne do zbiorów. Możemy myśleć, o zbiorze flag. Jeśli myślenie o zbiorach jest dla nas wygodniejsze, możemy skorzystać z typu `SET`. Typ `SET` pozwala wypisać maksymalnie 64 elementy, po czym każdemu z elementów zbioru przyporządkowuje jeden bit typie `BIT` użytym do jego implementacji. Przykład z flagami moglibyśmy zaimplementować też typem `SET('odczyt', 'zapis', 'usuwanie')`, co byłoby reprezentowane przez `BIT(3)` a tym samym zajmowało jeden bajt na każdą wartość. Wewnętrznie wartości tego zbioru `SET` będą reprezentowane przez następujące wartości typu `BIT`.

Wartość SET	Wartość BIT	Wartość TINYINT UNSIGNED
'odczyt'	b'001'	1
'zapis'	b'010'	2
'usuwanie'	b'100'	4

Do sprawdzania, czy zbiór posiada pewien element, używa się funkcji FIND_IN_SET, używanej jako FIND_IN_SET(wartość, zbiór), która zwraca wartość dodatnią, jeśli element jest w zbiorze i zero, w przeciwnym wypadku. Zachowanie takie jest związane z tym, że wewnętrznie funkcja ta jest zaimplementowana za pomocą operatora iloczynu logicznego, podobnie jak w przykładzie dla typu BIT.