

IDŹ DO

PRZYKŁADOWY ROZDZIAŁ



SPIS TREŚCI

KATALOG KSIĄŻEK

KATALOG ONLINE

ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK

DODAJ DO KOSZYKA

CENNIK I INFORMACJE

ZAMÓW INFORMACJE
O NOWOŚCIACH

ZAMÓW CENNIK

CZYTELNIA

FRAGMENTY KSIĄŻEK ONLINE

MySQL

Autor: Leon Atkinson

Tłumaczenie: Jarosław Dobrzański, Tomasz Żmijewski

ISBN: 83-7361-170-3

Tytuł oryginału: [Core MySQL](#)

Format: B5, stron: 608



Zaawansowani programiści na całym świecie wybierają MySQL jako podstawę swoich aplikacji opartych na WWW. Książka, którą trzymasz w ręku, dostarczy Ci wiedzy i przykładowego kodu; elementów niezbędnych do szybkiego pisania własnych aplikacji, niezależnie od stopnia ich skomplikowania.

Książka rozpoczyna się od omówienia podstaw MySQL-a: zapytań SQL, zasad projektowania baz danych, normalizacji, transakcji i przetwarzania równoległego. Następnie w usystematyzowany sposób opisuje szczegółowe możliwości MySQL oraz przedstawia efektywne techniki dostępu do baz MySQL-a z poziomu C, Javy, PHP, Perla, Pythona i innych środowisk programistycznych.

W książce opisano między innymi:

- Instalację i korzystanie z MySQL-a – wraz ze szczegółowym opisem implementacji SQL w MySQL-u
- Typy danych, zmienne, funkcje wbudowane i narzędzia dostępne z linii komend API MySQL w języku C
- Administrację bazami MySQL, wykonywanie kopii zapasowych i usuwanie skutków awarii
- Optymalizację i zabezpieczanie aplikacji
- Tworzenie rozproszonych baz danych
- Rozszerzanie funkcjonalności MySQL-a
- Tworzenie baz danych przenośnych na inne platformy

Dokładnie przestudiowana i zalecana przez twórcę MySQL-a, Michaela Wideniusa, książka „MySQL” dostarcza profesjonalnym programistom tego, czego poszukiwali: dogłębnej, przemyślanej wiedzy, potrzebnej do tworzenia zaawansowanych aplikacji.

Leon Atkinson jest autorem książki „PHP. Programowanie”. Jest też inicjatorem projektu FreeTrade: zestawu narzędzi open source wykorzystującego MySQL i PHP. Atkinson od 1997 roku używa MySQL w wielu aplikacjach sieciowych i e-commerce.



Spis treści

Wstęp	11
Część I MySQL i model relacyjny	13
Rozdział 1. Wprowadzenie do MySQL	15
Jak się korzysta z baz danych?.....	16
Dlaczego bazy danych są lepsze niż pliki?	23
Bazy danych nie są lekiem na całe zło	25
Dlaczego MySQL?	26
Historia MySQL.....	27
Rozdział 2. Instalacja MySQL	29
Pobieranie plików instalacyjnych.....	29
Linux RPM	30
Windows	31
Kompilowanie kodu źródłowego.....	33
Nadawanie uprawnień	33
Rozdział 3. Interakcja z MySQL	35
Połączenie klient-serwer poprzez TCP/IP	35
Narzędzia wiersza poleceń	36
Interfejsy graficzne	38
ODBC.....	39
Interfejsy WWW.....	40
Rozdział 4. Modele baz danych	41
Historia	41
Terminologia.....	43
DBMS — system zarządzania bazą danych	43
Bazy danych oparte na zwykłych plikach.....	44
Hierarchiczne bazy danych	45
Sieciowe bazy danych.....	46
Relacyjne bazy danych.....	47
Obiektowe bazy danych	48
Obiektowo-relacyjne bazy danych.....	49
Rozdział 5. Model relacyjny	51
Algebra relacyjna	51
Tabele, wiersze i kolumny.....	52
Klucze.....	53
Powiązania	55

Operacje relacyjne.....	56
Czy MySQL to prawdziwy system RDMBS?	62

Rozdział 6. Język zapytań strukturalnych..... 63

SQL jako język czwartej generacji	63
Definiowanie danych	64
Wstawianie wierszy	66
Aktualizacja wierszy.....	67
Usuwanie wierszy.....	67
Zapytania	68
Złączenia	69
Porządkowanie.....	71
Grupowanie	71
Stosowanie ograniczeń.....	72
Zmiana formy tabeli	73

Rozdział 7. Projektowanie bazy danych 75

Specyfikacja wymagań.....	76
Specyfikacja projektowa	79
Tworzenie diagramów	81
Języki modelowania.....	83
Diagramy ER.....	84
Tworzenie diagramu	85
Implementacja projektu	85
Testowanie.....	88
Planowanie rozwoju.....	88

Rozdział 8. Normalizacja 91

Zasadność normalizacji	92
Pierwsza postać normalna	93
Druga postać normalna	94
Trzecia postać normalna.....	96
Postać normalna Boyce-Codda	97
Czwarta postać normalna	98
Denormalizacja	99

Rozdział 9. Transakcje i współbieżność 101

Współbieżność	102
Transakcje	102
Blokowanie.....	106
Sekwencje.....	107

Część II Encyklopedia MySQL 109**Rozdział 10. Typy danych, zmienne i wyrażenia..... 111**

Typy danych.....	111
Zmienne.....	114
Operatory	116
Wyrażenia	123
Nazwy ze spacjami	124

Rozdział 11. Typy kolumn i indeksów	125
Liczby	126
Łańcuchy.....	128
Wartości opisujące czas	131
Aliasy typów kolumn	133
Indeksy	133
Rozdział 12. Funkcje wewnętrzne	135
Diagnostyka i konfiguracja	136
Sterowanie przebiegiem egzekucji	138
Grupowanie	142
Funkcje matematyczne	146
Łańcuchy.....	154
Czas	169
Pozostałe funkcje.....	181
Procedury	183
Rozdział 13. Instrukcje SQL	185
Komentarze	185
Alter Table.....	186
Analyze Table	190
Backup Table.....	191
Begin [Work].....	192
Change Master	192
Check Table	193
Commit.....	193
Create Database.....	194
Create Function	194
Create Index.....	194
Create Table.....	195
Delete.....	201
Describe	202
Drop Database	203
Drop Function	204
Drop Index.....	204
Drop Table	204
Explain	204
Flush	206
Grant	207
Insert.....	209
Kill.....	210
Lock Tables.....	210
Load Data Infile	211
Load Table	213
Optimize Table.....	214
Purge Master Logs.....	214
Rename Table	214
Repair Table	215
Replace	215
Reset Master	216
Reset Slave.....	216

Restore Table	216
Revoke.....	216
Rollback.....	217
Select.....	217
Set	222
Set Transaction	226
Show Columns.....	226
Show Create Table.....	227
Show Databases.....	228
Show Grants.....	228
Show Index.....	229
Show Logs	229
Show Processlist	230
Show Status.....	230
Show Table Status.....	232
Show Tables.....	233
Show Variables.....	233
Slave	237
Truncate	237
Unlock Tables.....	237
Update.....	238
Use	239

Rozdział 14. Narzędzia uruchamiane z wiersza poleceń.....241

Zmienne środowiskowe	241
Pliki opcji	242
comp_err.....	244
isamchk	244
make_binary_distribution.....	244
msql2mysql.....	244
my_print_defaults	245
myisamchk.....	246
myisamlog.....	252
myisampack.....	254
mysql.....	256
mysql_install_db.....	268
mysqlaccess	268
mysqladmin.....	269
mysqlbinlog.....	277
mysqlbug	279
mysqlc	279
mysqld.....	280
mysqld-max	296
mysqld-nt	296
mysqld-opt	297
mysqld_multi	297
mysqldump	299
mysqldumpslow	306
mysqlhotcopy	308
mysqlimport	311
mysqlshow	315

pack_isam	317
perror	317
replace	318
safe_mysqlld	319
Rozdział 15. API C	321
Typy danych	321
Funkcje klienckie	325
Funkcje obsługi tablic	350
Funkcje zestawów znaków	350
Funkcje obsługi plików	352
Funkcje obsługi błędów	354
Funkcje mieszające	354
Funkcje list	355
Funkcje zarządzające pamięcią	356
Funkcje opcji	357
Funkcje obsługi haseł	357
Funkcje łańcuchowe	358
Funkcje obsługi wątków	360
Część III Pisanie programów klienckich MySQL	361
Rozdział 16. Programowanie w API języka C	363
Przygotowanie programu	363
Pobieranie danych	365
Przetwarzanie danych	367
Rozdział 17. JDBC	371
Przygotowanie programu	371
Pobieranie danych	373
Przetwarzanie danych	375
Rozdział 18. VBScript i ODBC	377
Przygotowanie programu	377
Pobieranie danych	379
Przetwarzanie danych	381
Rozdział 19. PHP	385
Przygotowanie programu	385
Pobieranie danych	386
Przetwarzanie danych	388
Rozdział 20. Perl	391
Przygotowanie programu	391
Pobieranie danych	392
Przetwarzanie danych	393
Rozdział 21. Python	397
Przygotowanie programu	397
Pobieranie danych	398
Przetwarzanie danych	400

Rozdział 22. API MySQL++	403
Przygotowanie programu	403
Pobieranie danych.....	404
Przetwarzanie danych	406
Część IV Zagadnienia zaawansowane	409
Rozdział 23. Administracja bazą danych	411
Zakres odpowiedzialności	411
Udostępnianie danych	412
Zapewnienie spójności bazy danych.....	413
W razie awarii	414
Pomoc użytkownikom	414
Tworzenie i wdrażanie standardów.....	415
Rozdział 24. Fizyczne struktury danych	417
Model bazy danych i tabel.....	417
Partycje dedykowane	418
Typy tabel.....	418
Kolumny.....	425
Blokowanie tabel	426
Indeksy	426
Deskryptory plików	428
Pamięć systemowa	429
Dzienniki	429
Rozdział 25. Jak radzić sobie z awarią	435
Sprawdzanie tabel i ich naprawianie	435
Kopie bezpieczeństwa, odzyskiwanie danych.....	438
Rozdział 26. Optymalizacja	445
Zanim zajmiesz się optymalizacją	445
Badanie wydajności.....	446
Optymalizacja na etapie projektu.....	450
Optymalizacja w aplikacjach	451
Optymalizacja zapytań	452
Optymalizacja zapytań SQL	455
Troska o tabele.....	456
Dostrajanie serwera	457
Rekompilacja MySQL.....	459
Rozdział 27. Bezpieczeństwo	463
System uprawnień	463
Nadawanie uprawnień	469
Zapewnianie bezpieczeństwa	471
Rozdział 28. Zmiana bazy danych	475
Zmiana serwerów baz danych.....	475
Uzupełnianie możliwości MySQL	476
Użycie trybu ANSI	481
Wyjątkowe cechy MySQL.....	482

Rozdział 29. Rozproszone bazy danych	487
Pojęcie rozproszonych baz danych	487
Synchronizacja opóźniona	490
Replikacja w MySQL	492
Uruchamianie wielu serwerów	497
Rozdział 30. Odzworowanie na obiekty	499
Model obiektowy	500
Serializacja obiektów	500
Odzworowanie obiektowości i relacyjności	503
Rozdział 31. Rozszerzanie możliwości	513
Biblioteka do analizy błędów	513
Dodawanie zestawów znaków	515
Funkcje	520
Procedury	523
Dodatki	525
Dodatek A Zasoby internetowe	527
Oficjalne listy mailingowe	527
Nieoficjalne listy wysyłkowe	529
Archiwa list wysyłkowych	529
Witryny	530
Dodatek B Dalsza lektura	533
Dodatek C Biznes i kwestie prawne	535
Licencja MySQL	535
Publiczna licencja GNU	535
Stabilność	542
Pomoc	542
Dodatek D Słowa zastrzeżone	545
Dodatek E Kody błędów MySQL	549
Dodatek F Styl kodowania w SQL	561
Zalecenia ogólne	561
Identyfikatory	562
Tabele	562
Instrukcje	563
Dodatek G Projekt przykładowej bazy danych	565
Diagramy	565
Schemat SQL	567
Skorowidz	579

12

Funkcje wewnętrzne

W tym rozdziale:

- Diagnostyka i konfiguracja
- Sterowanie przebiegiem egzekucji
- Grupowanie
- Funkcje matematyczne
- Łącuchy
- Czas
- Pozostałe funkcje
- Procedury

Funkcje zwracają wartości, które czasami zależą od parametrów wejściowych. Funkcję można wywołać wszędzie tam, gdzie możliwe jest zastosowanie wyrażenia. Dotyczy to również samych parametrów funkcji — innymi słowy, możliwe jest zagnieżdżanie wywołań funkcji.

W przypadku większości funkcji parametry są przesyłane w nawiasach i nie może być odstepu pomiędzy nazwą funkcji a nawiasem otwierającym. Pomaga to interpreterowi MySQL odróżnić nazwy funkcji od nazw kolumn. Wymóg ten można znieść za pomocą odpowiednich argumentów podanych z wiersza poleceń, ale w konsekwencji używanie nazw funkcji do nazywania kolumn będzie niemożliwe.

Do kilku wymienionych w tym rozdziale funkcji lepiej pasowałoby określenie „operator”. Zostały jednak umieszczone tutaj, ponieważ ich działanie przypomina w większym stopniu funkcję. Z ich opisów jasno wynika, że nie wymagają nawiasów.

Każdy opis funkcji rozpoczyna się od nagłówka, prezentującego formę wywołania funkcji. Wyrazy pisane dużymi literami to słowa kluczowe. Wyrazy pisane małymi literami to przekazywane funkcji wartości, które mogą mieć formę stałych, nazw kolumn lub wyrażeń. MySQL rozpoznaje nazwy funkcji wpisane dowolną kombinacją dużych i małych liter.

Wielokropek (...) oznacza, że funkcja pobiera listę wartości oddzielonych przecinkami. Wszystko, co umieszczone zostało w nawiasach kwadratowych, jest opcjonalne. Niektóre funkcje mają kilka poziomów parametrów opcjonalnych.

Jak pamiętamy, wartością większości wyrażeń, które zawierają wartość `null`, będzie również wartość `null`. Dotyczy to również wywołań funkcji. Na przykład konkatencja wartości `null` z dowolnym łańcuchem daje `null`.

Diagnostyka i konfiguracja

Opisane tu funkcje zwracają informacje o serwerze i pomagają w diagnostyce.

BENCHMARK(rundy, wartość)

Funkcja oblicza wartość wyrażenia tyle razy, na ile wskazuje pierwszy argument. Zwraca zawsze zero, ale używając klienta `mysql`, można sprawdzić, ile czasu zajęło jej wykonanie. Rysunek 12.1 pokazuje, ile czasu zabiera MySQL przeprowadzenie 100 000 wywołań funkcji `SOUNDEX`.

Rysunek 12.1.
Funkcja
`BENCHMARK`

```
mysql> SELECT BENCHMARK(100000, SOUNDEX('Leon'));
+-----+
| BENCHMARK(100000, SOUNDEX('Leon')) |
+-----+
|                                     0 |
+-----+
1 row in set (0.25 sec)
```

Należy pamiętać, że zgłoszony czas to czas wykonania wywołania, a nie czas korzystania z procesora. Tłok na serwerze lub w sieci może znacząco spowolnić czas egzekucji.

CONNECTION_ID()

Funkcja zwraca identyfikator połączenia dla połączenia bieżącego — patrz: rysunek 12.2.

Rysunek 12.2.
Funkcja
`CONNECTION_ID`

```
mysql> SELECT CONNECTION_ID();
+-----+
| CONNECTION_ID() |
+-----+
|                 1 |
+-----+
1 row in set (0.01 sec)
```

DATABASE()

Funkcja zwraca nazwę domyślnej bazy danych. Jeżeli nie została wybrana żadna domyślna baza danych, zwracana jest wartość „null” — patrz: rysunek 12.3.

Rysunek 12.3.

Funkcja
DATABASE

```
mysql> SELECT DATABASE();
+-----+
| DATABASE() |
+-----+
| test      |
+-----+
1 row in set (0.00 sec)
```

LAST_INSERT_ID([klucz_główny])

MySQL dla każdego połączenia przechowuje wartość ostatnio użytego klucza głównego. Funkcja `LAST_INSERT_ID` zwraca tę wartość. Jeżeli prześlemy jej argument, wartość zostanie zgodnie z nim ustawiona.

Istnieją dwa sposoby użycia tej funkcji. Najczęściej stosowany dotyczy kolumn `AUTO_INCREMENT`. Jeżeli kolumnie nadana została wartość w wyniku automatycznej inkrementacji, to funkcji `LAST_INSERT_ID` przypisana zostaje właśnie ta wartość. Z kolei automatycznie inkrementowana kolumna, której nadamy wartość ręcznie zamiast podawania zera lub `null`, nie spowoduje stosownej aktualizacji wartości zwracanej przez `LAST_INSERT_ID`. Rysunek 12.4 przedstawia, jak zmienia się wartość `LAST_INSERT_ID` po wstawieniu wiersza z wartością `null` dla kolumny automatycznie inkrementowanej.

Rysunek 12.4.

Funkcja
LAST_INSERT_ID

```
mysql> SELECT LAST_INSERT_ID();
+-----+
| last_insert_id() |
+-----+
| 0                |
+-----+
1 row in set (0.01 sec)

mysql> INSERT INTO produkty VALUES (NULL, 'Mydło', 3.75, 'trójpak',
200);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT LAST_INSERT_ID();
+-----+
| last_insert_id() |
+-----+
| 5                |
+-----+
1 row in set (0.00 sec)
```

Za pomocą funkcji `LAST_INSERT_ID` można również symulować sekwencje, poprzez skojarzenie funkcji z jednokolumnową tabelą. Opis sekwencji i sposobu ich symulowania znajduje się w rozdziale 9. „Transakcje i współbieżność”.

SESSION_USER()

Jest to alias funkcji USER.

SYSTEM_USER()

Jest to alias funkcji USER.

USER()

Funkcja zwraca nazwę użytkownika, który zainicjował sesję, wraz z domeną komputera, z którego następuje połączenie — patrz: rysunek 12.5.

Rysunek 12.5.

*Funkcja
USER*

```
mysql> SELECT USER();
+-----+
| USER() |
+-----+
| ODBC@localhost |
+-----+
1 row in set (0.00 sec)
```

VERSION()

Funkcja zwraca numer wersji serwera MySQL — patrz: rysunek 12.6.

Rysunek 12.6.

*Funkcja
VERSION*

```
mysql> SELECT VERSION();
+-----+
| VERSION() |
+-----+
| 4.0.12-nt |
+-----+
1 row in set (0.02 sec)
```

Sterowanie przebiegiem egzekucji

Opisane tu funkcje sterują przebiegiem egzekucji. IF oraz CASE zostały zaimplementowane na wzór instrukcji języka trzeciej generacji i mimo że nie używa się z nimi nawiasów, korzysta się z nich tak, jak z funkcji. Dwie funkcje blokujące oraz funkcja wymuszająca oczekiwanie bazy nadrzędnej na podrzędną ułatwiają zarządzanie współbieżnością.

CASE wartość_testowa WHEN przypadek1 THEN wartość_zwracana1 [WHEN przypadek2 THEN wartość_zwracana2] ... [ELSE wartość_domyślna] END

Instrukcja pobiera wartość testową i porównuje ją z jednym lub z kilkoma „przypadkami”. Jeżeli wartość testowa odpowiada któremuś przypadkowi, zwracana jest wartość z nim skojarzona. Jeżeli wartość testowa nie jest równa żadnemu z przypadków, instrukcja CASE zwraca opcjonalną wartość domyślną.

Rysunek 12.7 przedstawia instrukcję CASE zastosowaną dla tabeli o pięciu wierszach. Dla kolumn o liczbie porządkowej 1 i 2 zwracany jest numer kolumny słownie, a dla pozostałych domyślny łańcuch Inna wartość.

Rysunek 12.7.
Instrukcja
CASE

```
mysql> SELECT Lp, CASE Lp
->   WHEN 1 THEN 'Jeden'
->   WHEN 2 THEN 'Dwa'
->   ELSE 'Inna wartość'
->   END AS 'Rezultaty działania CASE'
-> FROM produkty;
```

Lp	Rezultaty działania CASE
1	Jeden
2	Dwa
3	Inna wartość
4	Inna wartość
5	Inna wartość

5 rows in set (0.01 sec)

CASE WHEN przypadek1 THEN wartość_zwracana1 [WHEN przypadek2 THEN wartość_zwracana2] ... [ELSE wartość_domyślna] END

Alternatywna wersja instrukcji CASE oblicza wartość każdego wyrażenia, zamiast porównywać je do pojedynczych wartości. Przykład przedstawiony na rysunku 12.8 pokazuje, że wyrażenia będące poszczególnymi przypadkami oraz wartości zwracane mogą być dowolnym prawidłowym wyrażeniem, w tym wywołaniem funkcji lub odwołaniem do innej kolumny.

Ponieważ instrukcja CASE może zmieniać typy dla każdego wiersza, przyjmuje się, że kolumna ma typ zgodny z wartością zwracaną przez pierwszy warunek WHEN.

GET_LOCK(nazwa, limit_czasu)

Przez określony czas mierzony w sekundach funkcja GET_LOCK stara się uzyskać dostęp do blokady o podanej nazwie. Wywołanie zakończy się dopiero z chwilą upływu zadanego czasu lub uzyskania dostępu do blokady.

Rysunek 12.8.

Instrukcja
CASE WHEN

```
mysql> SELECT Lp, CASE
->     WHEN Lp < 3 THEN POWER(Lp,2)
->     WHEN Lp = 4 THEN Lp
->     ELSE Nazwa
->     END AS 'Rezultaty działania CASE'
-> FROM produkty;
+-----+-----+
| Lp | Rezultaty działania CASE |
+-----+-----+
| 1 |          1.000000 |
| 2 |          4.000000 |
| 3 |          Szczotka |
| 4 |             4 |
| 5 |           Mydło |
+-----+-----+
5 rows in set (0.00 sec)
```

Jedno połączenie może utrzymywać tylko jedną blokadę naraz. Wywołanie `GET_LOCK` uwalnia blokadę utrzymywaną dotychczas, ale dobrym nawykiem jest używanie `RELEASE_LOCK` z chwilą, gdy blokada nie jest już potrzebna. Blokada jest zwalniana również w chwili przerwania połączenia.

Fakt utrzymywania blokady nie stwarza żadnych ograniczeń dla innych połączeń ani nie daje żadnych korzyści dla połączenia ją utrzymującego. Blokad można jednak używać w powiązaniu, implementując w ten sposób dowolny poziom blokowania. Zastosowanie `GET_LOCK` do implementacji powiązanego blokowania wierszy zostało opisane w rozdziale 9. Prosty przykład przedstawiono na rysunku 12.9.

Rysunek 12.9.

Funkcja
GET_LOCK

```
mysql> SELECT GET_LOCK('produkty.Lp=3', 60);
+-----+-----+
| GET_LOCK('produkty.Lp=3', 60) |
+-----+-----+
|                               1 |
+-----+-----+
1 row in set (0.00 sec)

mysql> UPDATE produkty SET Cena=3.15 WHERE Lp=3;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT RELEASE_LOCK('produkty.Lp=3');
+-----+-----+
| RELEASE_LOCK('produkty.Lp=3') |
+-----+-----+
|                               1 |
+-----+-----+
1 row in set (0.00 sec)
```

IF(wartość_testowa, wartość_prawda, wartość_fałsz)

Funkcja zwraca jedną wartość, jeżeli wyrażenie testowe ma wartość true, i inną, jeżeli ma ono wartość false. Wartość testowa jest konwertowana na liczbę całkowitą, dlatego należy pamiętać o użyciu operatora porównania zamiast przesyłania jedynie liczby zmiennoprzecinkowej.

Podobnie jak w przypadku CASE, instrukcja IF może zwracać typy niezgodne z sobą w każdym z dwóch wariantów. Jeżeli obydwie wartości są łańcuchami, obydwie zostaną zwrócone jako łańcuchy. Jeżeli ani jedna wartość nie jest łańcuchem, a przynajmniej jedna jest liczbą zmiennoprzecinkową, IF zwróci obydwie jako wartości zmiennoprzecinkowe. W pozostałych przypadkach zwrócone zostaną liczby całkowite. Rysunek 12.10 demonstruje użycie instrukcji IF do oznaczenia dnia jako dnia w tygodniu lub dnia w weekendzie.

Rysunek 12.10.
Instrukcja IF

```
mysql> SELECT IF(DAYOFWEEK(NOW()) in (0, 6, 7), 'weekend', 'dzień tygodnia');
+-----+
| IF(DAYOFWEEK(NOW()) in (0, 6, 7), 'weekend', 'dzień tygodnia') |
+-----+
| dzień tygodnia |
+-----+
1 row in set (0.01 sec)
```

IFNULL(wartość_testowa, wartość_zwracana)

Funkcja zwraca wartość testową, jeżeli jest ona niepusta — w innym przypadku zwracany jest drugi argument. Przykład przedstawiony na rysunku 12.11 pokazuje, jak można za pomocą instrukcji IFNULL zamieniać wartości null stosownymi opisami.

Rysunek 12.11.
Funkcja
IFNULL

```
mysql> SELECT Nazwa,
-> IFNULL(Opis, 'Brak opisu') AS Opis
-> FROM produkty;
+-----+-----+
| Nazwa      | Opis      |
+-----+-----+
| Szczoteczka | Brak opisu |
| Grzebień   | Brak opisu |
| Szczotka   | Brak opisu |
| Pasta      | Brak opisu |
| Mydło      | trójkpak  |
+-----+-----+
5 rows in set (0.00 sec)
```

MASTER_POS_WAIT(nazwa, pozycja)

Funkcja oczekuje, aż serwer podrzędny „dogoni” serwer nadrzędny w obrębie środowiska replikacyjnego. Wymagane jest podanie nazwy pliku dziennika i pozycji w dzienniku, którą serwer podrzędny musi osiągnąć. Jeżeli serwer nie jest serwerem nadrzędnym, zwracana

jest od razu wartość `null`. Jeżeli serwer podrzędny nie został jeszcze uruchomiony, funkcja będzie trwała w zablokowaniu, oczekując na uruchomienie serwera i jego dotarcie do określonej pozycji w dzienniku.

Funkcja zwraca liczbę „przerobionych” w czasie oczekiwania zdarzeń w dzienniku. Więcej informacji na temat replikacji znajduje się w rozdziale 29. „Rozproszone bazy danych”.

NULLIF(wartość_testowa1, wartość_testowa2)

Funkcja zwraca `null`, jeżeli obydwie wartości testowe są równe. W innym przypadku zwraca pierwszą wartość testową. Funkcja ta przydaje się do zamiany wartości zerowych na `null`, co ilustruje rysunek 12.12.

Rysunek 12.12.

*Funkcja
NULLIF*

```
mysql> SELECT Nazwa, NULLIF(Magazyn, 0) AS Magazyn
-> FROM produkty;
+-----+-----+
| Nazwa      | Magazyn |
+-----+-----+
| Szczoteczka | NULL    |
| Grzebień   | NULL    |
| Szczotka   | NULL    |
| Pasta      | NULL    |
| Mydło      | 200     |
+-----+-----+
5 rows in set (0.00 sec)
```

RELEASE_LOCK(nazwa)

Funkcja zwalnia blokadę wskazaną argumentem `nazwa`, uzyskaną wcześniej za pomocą funkcji `GET_LOCK`. Jeżeli wskazana blokada nie jest utrzymywana, zwracana jest wartość `null`. Przykład zastosowania znajduje się w opisie funkcji `GET_LOCK`.

Grupowanie

Opisane tu funkcje działają na kilku wartościach w kolumnie. Jeżeli nie podano klauzuli `GROUP BY`, wówczas obejmują swoim działaniem wszystkie wiersze. Na potrzeby przykładów tu przedstawionych stworzona została nowa definicja tabeli, którą przedstawia rysunek 12.13.

Kolumny z wartością `null` są pomijane w obliczeniach, dokonywanych przez te funkcje. Użycie `null` w wyrażeniu zawsze spowoduje zwrócenie `null`, ale funkcja grupująca zastosowana na kolumnie zawierającej `null` nie powoduje tego efektu — `null` jest wówczas ignorowane.

Rysunek 12.13.
Przykład
grupowania
danych

```
mysql> CREATE TABLE testgrup (
-> Lp INT NOT NULL AUTO_INCREMENT,
-> Druzyna VARCHAR(16),
-> Wynik INT,
-> PRIMARY KEY(Lp)
-> );
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO testgrup (Druzyna, Wynik) VALUES
-> ('Czerwoni', 11),
-> ('Czerwoni', 45),
-> ('Czerwoni', 98),
-> ('Czerwoni', 19),
-> ('Czerwoni', 11),
-> ('Czerwoni', 37),
-> ('Czerwoni', 17),
-> ('Czerwoni', 75),
-> ('Niebiescy', 23),
-> ('Niebiescy', 91),
-> ('Niebiescy', 80),
-> ('Niebiescy', 63),
-> ('Niebiescy', 55),
-> ('Niebiescy', 89),
-> ('Niebiescy', 64),
-> ('Niebiescy', 5);
Query OK, 16 rows affected (0.00 sec)
Records: 16 Duplicates: 0 Warnings: 0
```

AVG(kolumna)

Funkcja zwraca średnią arytmetyczną wyników z grupy. Jest ona definiowana jako suma wszystkich wartości w grupie podzielona przez ich liczbę. Rysunek 12.14 prezentuje średnie wyniki każdej drużyny.

Rysunek 12.14.
Funkcja AVG

```
mysql> SELECT Druzyna, AVG(Wynik)
-> FROM testgrup
-> GROUP BY Druzyna;
+-----+-----+
| Druzyna | AVG(Wynik) |
+-----+-----+
| Czerwoni | 39.1250 |
| Niebiescy | 58.7500 |
+-----+-----+
2 rows in set (0.00 sec)
```

BIT_AND(kolumna)

Funkcja wykonuje na grupie bitową operację AND i zwraca dziesiętną liczbę całkowitą — patrz: rysunek 12.15.

BIT_OR(kolumna)

Funkcja wykonuje na grupie bitową operację OR i zwraca dziesiętną liczbę całkowitą. Rysunek 12.15 ukazuje, że nie ma bitów, które byłyby ustawione we wszystkich wynikach, ale pierwszych siedem jest ustawionych w co najmniej jednym wyniku.

Rysunek 12.15.

*Funkcja
BIT_OR*

```
mysql> SELECT Druzyna, BIT_AND(Wynik), BIT_OR(Wynik)
-> FROM testgrup
-> GROUP BY Druzyna;
```

Druzyna	BIT_AND(Wynik)	BIT_OR(Wynik)
Czerwoni	0	127
Niebiescy	0	127

2 rows in set (0.01 sec)

COUNT(kolumna)

Funkcja zwraca liczbę niepustych elementów w grupie. Można też zastosować znak gwiazdki (*) do sprawdzenia liczby wierszy niezależnie od tego, czy mają wartość null. Na rysunku 12.16 za każdym razem widzimy 8 wyników. W drużynie Czerwonych dwukrotnie pojawił się wynik 11, więc liczba niepowtarzalnych wyników wynosi dla niej 7.

Rysunek 12.16.

*Funkcja
COUNT*

```
mysql> SELECT Druzyna, COUNT(Wynik), COUNT(DISTINCT Wynik)
-> FROM testgrup
-> GROUP BY Druzyna;
```

Druzyna	COUNT(Wynik)	COUNT(DISTINCT Wynik)
Czerwoni	8	7
Niebiescy	8	8

2 rows in set (0.00 sec)

COUNT(DISTINCT ...)

Ta forma funkcji COUNT zwraca liczbę niepowtarzalnych kombinacji wartości kolumn podanych jako argumenty — patrz: rysunek 12.16.

MAX(...)

Funkcja zwraca największą wartość w grupie. Maksymalna wartość łańcuchowa jest definiowana jako ta, która występuje ostatnia w kolejności alfabetycznej. Ponieważ daty można porządkować, maksymalna data to ostatnia w kolejności. Dwucyfrowe oznaczenia roku

mogą nie być poprawnie sortowane, chyba że stanowią część kolumny YEAR. Konieczne może wówczas okazać się wymuszenie czterocyfrowych oznaczeń lat poprzez dodanie zera za pomocą funkcji DATE_ADD. Rysunek 12.17 przedstawia maksymalne i minimalne wyniki dla obu drużyn.

Rysunek 12.17.
Funkcja MAX

```
mysql> SELECT Druzyna, MIN(Wynik), MAX(Wynik)
-> FROM testgrup
-> GROUP BY Druzyna;
+-----+-----+-----+
| Druzyna | MIN(Wynik) | MAX(Wynik) |
+-----+-----+-----+
| Czerwoni |          11 |          98 |
| Niebiescy |           5 |          91 |
+-----+-----+-----+
2 rows in set (0.01 sec)
```

MIN(...)

Funkcja zwraca najmniejszą wartość w grupie. Minimalna wartość łańcuchowa to ta, która występuje pierwsza w kolejności alfabetycznej (patrz: rysunek 12.17).

STD(...)

Funkcja zwraca odchylenie standardowe dla grupy — patrz: rysunek 12.18.

Rysunek 12.18.
Funkcja STD

```
mysql> SELECT Druzyna, STD(Wynik)
-> FROM testgrup
-> GROUP BY Druzyna;
+-----+-----+
| Druzyna | STD(Wynik) |
+-----+-----+
| Czerwoni |    30.1431 |
| Niebiescy |    28.7956 |
+-----+-----+
2 rows in set (0.00 sec)
```

STDDEV(...)

Jest to alias funkcji STD.

SUM(...)

Funkcja zwraca sumę elementów grupy. Łańcuchy i daty są przekształcane na liczby całkowite — patrz: rysunek 12.19.

Rysunek 12.19.
Funkcja SUM

```
mysql> SELECT Druzyna, SUM(Wynik)
-> FROM testgrup
-> GROUP BY Druzyna;
+-----+-----+
| Druzyna | SUM(Wynik) |
+-----+-----+
| Czerwoni |          313 |
| Niebiescy |          470 |
+-----+-----+
2 rows in set (0.00 sec)
```

Funkcje matematyczne

Opisane tu funkcje wykonują operacje matematyczne. Większość z nich pobiera i zwraca liczby zmiennoprzecinkowe.

ABS(liczba)

Funkcja zwraca wartość bezwzględną z liczby — patrz: rysunek 12.20.

Rysunek 12.20.
Funkcja ABS

```
mysql> SELECT ABS(-17);
+-----+
| ABS(-17) |
+-----+
|          17 |
+-----+
1 row in set (0.00 sec)
```

ACOS(liczba)

Funkcja zwraca arcus cosinus z liczby. Arcus cosinus z liczb większych od 1 lub mniejszych od 0 jest nieokreślony — patrz: rysunek 12.21.

Rysunek 12.21.
Funkcja ACOS

```
mysql> SELECT ACOS(1);
+-----+
| ACOS(1) |
+-----+
| 0.000000 |
+-----+
1 row in set (0.00 sec)
```

ASIN(liczba)

Funkcja zwraca arcus sinus z liczby. Arcus sinus z liczb większych od 1 lub mniejszych od 0 jest nieokreślony — patrz: rysunek 12.22.

Rysunek 12.22.
Funkcja ASIN

```
mysql> SELECT ASIN(1);
+-----+
| ASIN(1) |
+-----+
| 1.570796 |
+-----+
1 row in set (0.00 sec)
```

ATAN(liczba)

Funkcja zwraca arcus tangens z liczby — patrz: rysunek 12.23.

Rysunek 12.23.
Funkcja ATAN

```
mysql> SELECT ATAN(1);
+-----+
| ATAN(1) |
+-----+
| 0.785398 |
+-----+
1 row in set (0.00 sec)
```

ATAN2(liczba, liczba)

Funkcja zwraca kąt między odcinkiem łączącym dany punkt z początkiem układu współrzędnych a osią x, wyrażony w radianach — patrz: rysunek 12.24.

Rysunek 12.24.
Funkcja ATAN2

```
mysql> SELECT ATAN2(3,7);
+-----+
| ATAN2(3,7) |
+-----+
| 0.404892 |
+-----+
1 row in set (0.00 sec)
```

CEILING(liczba)

Funkcja zaokrągla liczbę zmiennoprzecinkową do najbliższej większej liczby całkowitej — patrz: rysunek 12.25.

Rysunek 12.25.

Funkcja
CEILING

```
mysql> SELECT CEILING(1.3);
+-----+
| CEILING(1.3) |
+-----+
|           2 |
+-----+
1 row in set (0.00 sec)
```

COS(liczba)

Funkcja zwraca cosinus z liczby wyrażony w radianach — patrz: rysunek 12.26.

Rysunek 12.26.

Funkcja COS

```
mysql> SELECT COS(1);
+-----+
| COS(1) |
+-----+
| 0.540302 |
+-----+
1 row in set (0.00 sec)
```

COT(liczba)

Funkcja zwraca cotangens z liczby — patrz: rysunek 12.27.

Rysunek 12.27.

Funkcja COT

```
mysql> SELECT COT(1);
+-----+
| COT(1) |
+-----+
| 0.64209262 |
+-----+
1 row in set (0.00 sec)
```

DEGREES(liczba)

Funkcja zamienia radiany na stopnie — patrz: rysunek 12.28.

Rysunek 12.28.

Funkcja
DEGREES

```
mysql> SELECT DEGREES(1);
+-----+
| DEGREES(1) |
+-----+
| 57.295779513082 |
+-----+
1 row in set (0.00 sec)
```

EXP(liczba)

Funkcja zwraca podstawę logarytmu naturalnego (e) podniesioną do podanej potęgi — patrz: rysunek 12.29.

Rysunek 12.29.
Funkcja EXP

```
mysql> SELECT EXP(2);
+-----+
| EXP(2) |
+-----+
| 7.389056 |
+-----+
1 row in set (0.00 sec)
```

FLOOR(liczba)

Funkcja zwraca najbliższą mniejszą liczbę całkowitą dla podanej liczby zmiennoprzecinkowej — patrz: rysunek 12.30.

Rysunek 12.30.
Funkcja FLOOR

```
mysql> SELECT FLOOR(1.7);
+-----+
| FLOOR(1.7) |
+-----+
|          1 |
+-----+
1 row in set (0.00 sec)
```

GREATEST(...)

Funkcja zwraca największą wartość z podanej listy, działając zarówno na łańcuchach, jak i na liczbach — patrz: rysunek 12.31.

Rysunek 12.31.
Funkcja GREATEST

```
mysql> SELECT GREATEST(1,2,3);
+-----+
| GREATEST(1,2,3) |
+-----+
|          3 |
+-----+
1 row in set (0.00 sec)
```

LEAST(...)

Funkcja zwraca najmniejszą wartość z podanej listy, działając zarówno na łańcuchach, jak i na liczbach — patrz: rysunek 12.32.

Rysunek 12.32.
Funkcja LEAST

```
mysql> SELECT LEAST(1,2,3);
+-----+
| LEAST(1,2,3) |
+-----+
|             1 |
+-----+
1 row in set (0.00 sec)
```

LOG(liczba)

Funkcja zwraca logarytm naturalny z liczby — patrz: rysunek 12.33.

Rysunek 12.33.
Funkcja LOG

```
mysql> SELECT LOG(10);
+-----+
| LOG(10) |
+-----+
| 2.302585 |
+-----+
1 row in set (0.00 sec)
```

LOG10(liczba)

Funkcja zwraca logarytm dziesiętny z liczby — patrz: rysunek 13.34.

Rysunek 13.34.
Funkcja LOG10

```
mysql> SELECT LOG10(1234);
+-----+
| LOG10(1234) |
+-----+
|    3.091315 |
+-----+
1 row in set (0.00 sec)
```

MOD(liczba, liczba)

Funkcja zwraca resztę z dzielenia, podobnie jak operator % — patrz: rysunek 12.35.

Rysunek 12.35.
Funkcja MOD

```
mysql> SELECT MOD(35, 4);
+-----+
| MOD(35, 4) |
+-----+
|             3 |
+-----+
1 row in set (0.00 sec)
```

PI()

Funkcja zwraca kilka pierwszych cyfr liczby π — patrz: rysunek 12.36. Wewnętrznie MySQL przechowuje pełną, podwójnej precyzji wartość π .

Rysunek 12.36.
Funkcja PI

```
mysql> SELECT PI();
+-----+
| PI()   |
+-----+
| 3.141593 |
+-----+
1 row in set (0.00 sec)
```

POW(liczba, liczba)

Funkcja zwraca pierwszą liczbę podniesioną do potęgi liczby drugiej — patrz: rysunek 12.37.

Rysunek 12.37.
Funkcja POW

```
mysql> SELECT POW(2,10);
+-----+
| POW(2,10) |
+-----+
| 1024.000000 |
+-----+
1 row in set (0.00 sec)
```

POWER(liczba, liczba)

Jest to alias funkcji POW.

RADIANS(liczba)

Funkcja zamienia stopnie na radiany — patrz: rysunek 12.38.

Rysunek 12.38.
Funkcja RADIANS

```
mysql> SELECT RADIANS(45);
+-----+
| RADIANS(45) |
+-----+
| 0.78539816339745 |
+-----+
1 row in set (0.01 sec)
```

RAND([zarodek])

Funkcja zwraca liczbę pseudolosową z zakresu od 0 do 1. Zarodka należy używać, aby rozpocząć generowanie liczb pseudolosowych od określonego punktu w sekwencji. Jeżeli nie podamy zarodka, MySQL użyje jako zarodka dla generatora liczb pseudolosowych wartość pobraną z zegara systemowego — patrz: rysunek 12.39.

Rysunek 12.39.
Funkcja RAND

```
mysql> SELECT RAND(12345);
+-----+
| RAND(12345) |
+-----+
| 0.66570343232313 |
+-----+
1 row in set (0.01 sec)
```

ROUND(liczba[, dokładność])

Funkcja zaokrągla liczbę zmiennoprzecinkową do całkowitej lub opcjonalnie do podanej liczby cyfr po przecinku — patrz: rysunek 12.40.

Rysunek 12.40.
Funkcja ROUND

```
mysql> SELECT ROUND(15.666, 2);
+-----+
| ROUND(15.666, 2) |
+-----+
| 15.67 |
+-----+
1 row in set (0.00 sec)
```

SIGN(liczba)

Funkcja zwraca -1 , jeżeli liczba jest ujemna, lub 1 , jeżeli jest dodatnia — patrz: rysunek 12.41.

Rysunek 12.41.
Funkcja SIGN

```
mysql> SELECT SIGN(-10);
+-----+
| SIGN(-10) |
+-----+
| -1 |
+-----+
1 row in set (0.00 sec)
```

SIN(liczba)

Funkcja zwraca sinus z liczby — patrz: rysunek 12.42.

Rysunek 12.42.
Funkcja SIN

```
mysql> SELECT SIN(1);
+-----+
| SIN(1) |
+-----+
| 0.841471 |
+-----+
1 row in set (0.00 sec)
```

SQRT(liczba)

Funkcja zwraca pierwiastek kwadratowy z liczby — patrz: rysunek 12.43.

Rysunek 12.43.
Funkcja SQRT

```
mysql> SELECT SQRT(15);
+-----+
| SQRT(15) |
+-----+
| 3.872983 |
+-----+
1 row in set (0.00 sec)
```

TAN(liczba)

Funkcja zwraca tangens z kąta wyrażonego w radianach — patrz: rysunek 12.44.

Rysunek 12.44.
Funkcja TAN

```
mysql> SELECT TAN(1);
+-----+
| TAN(1) |
+-----+
| 1.557408 |
+-----+
1 row in set (0.00 sec)
```

TRUNCATE(liczba, dokładność)

Funkcja skracza część ułamkową liczby do podanej dokładności — patrz: rysunek 12.45.

Rysunek 12.45.
Funkcja
TRUNCATE

```
mysql> SELECT TRUNCATE(1.2345, 2);
+-----+
| TRUNCATE(1.2345, 2) |
+-----+
| 1.23 |
+-----+
1 row in set (0.00 sec)
```

Łańcuchy

Opisane tu funkcje wykonują operacje na łańcuchach i zwracają łańcuchy. Jak pamiętamy, MySQL udostępnia operatory do porównywania łańcuchów, w tym LIKE i REGEXP.

ASCII(znak)

Funkcja zwraca kod ASCII pierwszego znaku w danym łańcuchu — patrz: rysunek 12.46.

Rysunek 12.46.
Funkcja ASCII

```
mysql> SELECT ASCII('a');
+-----+
| ASCII('a') |
+-----+
|          97 |
+-----+
1 row in set (0.00 sec)
```

BIN(liczba_calk)

Funkcja zwraca reprezentację binarną podanej liczby całkowitej — patrz: rysunek 12.47.

Rysunek 12.47.
Funkcja BIN

```
mysql> SELECT BIN(13);
+-----+
| BIN(13) |
+-----+
| 1101    |
+-----+
1 row in set (0.00 sec)
```

BINARY łańcuch

Słowo kluczowe BINARY sprawia, że dany łańcuch jest traktowany jako binarny, co oznacza, że wielkość liter ma znaczenie przy porównaniach. Słowo BINARY ma wyższy priorytet niż operatory porównania (wykonywane jest przed nimi). Rysunek 12.48 ilustruje zjawisko rozróżniania dużych i małych liter w łańcuchach binarnych i nierozróżniania w łańcuchach normalnych.

Rysunek 12.48.
Słowo kluczowe
BINARY

```
mysql> SELECT 'a'='A', BINARY 'a'='A';
+-----+-----+
| 'a'='A' | BINARY 'a'='A' |
+-----+-----+
|          1 |                0 |
+-----+-----+
1 row in set (0.00 sec)
```


CONCAT_WS(separator, ...)

Funkcja dokonuje połączenia łańcuchów, rozdzielając je separatorem — patrz: rysunek 12.52. W odróżnieniu od funkcji CONCAT wartość null jest tu ignorowana. Separator o wartości null powoduje, że funkcja zwraca null.

Rysunek 12.52.
Funkcja
CONCAT_WS

```
mysql> SELECT CONCAT_WS('<>', 'a', 'b', 'c');
+-----+
| CONCAT_WS('<>', 'a', 'b', 'c') |
+-----+
| a<>b<>c                          |
+-----+
1 row in set (0.01 sec)
```

CONV(liczba_calk, z_podstawy, do_podstawy)

Funkcja dokonuje konwersji liczby całkowitej z jednego systemu liczenia na inny. Funkcja ta może zastępować funkcje BIN, HEX i OCT. Na rysunku 12.53 przedstawiono konwersję liczby ósemkowej na dziesiętną.

Rysunek 12.53.
Funkcja
CONV

```
mysql> SELECT CONV('100', 8, 10);
+-----+
| CONV('100', 8, 10) |
+-----+
| 64                   |
+-----+
1 row in set (0.01 sec)
```

DECODE(tekst_zaszyfrowany, hasło)

Funkcja odszyfrowuje łańcuch zaszyfrowany za pomocą funkcji ENCODE — patrz: rysunek 12.54.

Rysunek 12.54.
Funkcja
DECODE

```
mysql> SELECT DECODE(ENCODE('MySQL', 'hasło'), 'hasło');
+-----+
| DECODE(ENCODE('MySQL', 'hasło'), 'hasło') |
+-----+
| MySQL                                       |
+-----+
1 row in set (0.00 sec)
```

ELT(element, ...)

Funkcja zwraca element o indeksie wskazanym przez pierwszy argument — patrz: rysunek 12.55. Pierwszy element ma numer 1.

Rysunek 12.55.
Funkcja *ELT*

```
mysql> SELECT ELT(3, 'a','b','c','d');
+-----+
| ELT(3, 'a','b','c','d') |
+-----+
| c                          |
+-----+
1 row in set (0.00 sec)
```

ENCODE(tekst, hasło)

Funkcja zwraca łańcuch zaszyfrowany za pomocą podanego hasła. Zwrócony łańcuch jest łańcuchem binarnym o tej samej długości, co pierwotny tekst. Łańcuch można odszyfrować za pomocą funkcji *DECODE*. Funkcji tej nie należy stosować dla kolumny *Password* w *tabeli uprawnień* (ang. *grant table*) MySQL. Do tego celu służy funkcja *PASSWORD*. Rysunek 12.56 prezentuje zastosowanie *ENCODE* do ukrycia pól zawierających hasła w aplikacji.

Rysunek 12.56.
Funkcja *ENCODE*

```
mysql> UPDATE uzytkownik
-> SET haslo = ENCODE('sekret', 'haslo')
-> WHERE Lp=1;
Query Ok., 1 row affected (0.00 sec)
Rows matched 1 Changed 1 Warnings 0
```

ENCRYPT(łańcuch[, domieszka])

Funkcja odbudowuje funkcję *crypt* języka *C* — patrz: rysunek 12.57. Jest to szyfrowanie nieodwracalne. Drugi argument może być dwuznakowym łańcuchem poprawiającym losowość szyfrowania. Dłuższe łańcuchy są skracane. Funkcja ta nie jest kompatybilna z funkcją *PASSWORD* i każdy system operacyjny może mieć odmienną implementację *crypt* (w bazach MySQL instalowanych w systemach Windows funkcja może zwracać wartość *NULL*).

Rysunek 12.57.
Funkcja *ENCRYPT*

```
mysql> SELECT ENCRYPT('haslo', 'ab');
+-----+
| ENCRYPT('haslo', 'ab') |
+-----+
| abJnggxhB/ywI         |
+-----+
1 row in set (0.00 sec)
```

EXPORT_SET(pole_bitowe, opis_wlaczanej, opis_wylaczanej[, separator[,liczba_bitow]])

Funkcja zwraca łańcuch opcji odpowiadający bitom pierwszego argumentu. Bity, skonwertowane kolejno od najmłodszego do najstarszego, budują łańcuch od lewej do prawej. Drugi i trzeci argument to ciągi tekstowe reprezentujące odpowiednio włączone i wyłączone bity.

Domyślnym separatorem jest przecinek, ale może być nim dowolny łańcuch. Domyślnie używane są wszystkie 64 bity reprezentacji liczby całkowitej, ale można to ograniczyć za pomocą piątego argumentu. Na rysunku 12.58 znaczki T i N są rozdzielane pionową kreską. Ukazano 8 pól (bitów) — pola aktywne definiuje liczba 9 (dwójkowo 1001).

Rysunek 12.58.

Funkcja
EXPORT_SET

```
mysql> SELECT EXPORT_SET(9, 'T', 'N', '|', 8);
+-----+
| EXPORT_SET(9, 'T', 'N', '|', 8) |
+-----+
| T|N|N|T|N|N|N|N |
+-----+
1 row in set (0.04 sec)
```

FIELD(element, ...)

Funkcja zwraca indeks pierwszego argumentu w obrębie listy, która po nim następuje — patrz: rysunek 12.59. Elementy listy są indeksowane od 1. Jeżeli szukany ciąg nie zostanie odnaleziony, na liście zwracane jest 0.

Rysunek 12.59.

Funkcja
FIELD

```
mysql> SELECT FIELD('b', 'a','b','c','d');
+-----+
| FIELD('b', 'a','b','c','d') |
+-----+
| 2 |
+-----+
1 row in set (0.00 sec)
```

FIND_IN_SET(łańcuch, lista)

Funkcja zwraca indeks elementu listy łańcuchowej — listy elementów oddzielonych przecinkami, będącej łańcuchem (patrz: rysunek 12.60). Elementy listy są numerowane od 1.

Rysunek 12.60.

Funkcja
FIND_IN_SET

```
mysql> SELECT FIND_IN_SET('c', 'a,b,c,d');
+-----+
| FIND_IN_SET('c', 'a,b,c,d') |
+-----+
| 3 |
+-----+
1 row in set (0.00 sec)
```

FORMAT(liczba, dokładność)

Funkcja zwraca liczbę zmiennoprzecinkową z przecinkami wstawionymi co trzy potęgi liczby 10 i z podaną liczbą cyfr w części ułamkowej — patrz: rysunek 12.61.

Rysunek 12.61.

*Funkcja
FORMAT*

```
mysql> SELECT FORMAT(12345678.909112, 2);
+-----+
| FORMAT(12345678.909112, 2) |
+-----+
| 12,345,678.91              |
+-----+
1 row in set (0.00 sec)
```

HEX(liczba_całk)

Funkcja zwraca szesnastkową reprezentację liczby całkowitej — patrz: rysunek 12.62.

Rysunek 12.62.

*Funkcja
HEX*

```
mysql> SELECT HEX(563823);
+-----+
| HEX(563823) |
+-----+
| 89A6F       |
+-----+
1 row in set (0.00 sec)
```

INET_ATON(adres)

Funkcja konwertuje adres internetowy zapisany jako łańcuch do postaci numerycznej. Adresy cztero- i ośmiobitowe są akceptowane — patrz: rysunek 12.63.

Rysunek 12.63.

*Funkcja
INET_ATON*

```
mysql> SELECT INET_ATON('64.28.67.70');
+-----+
| INET_ATON('64.28.67.70') |
+-----+
| 1075594054                |
+-----+
1 row in set (0.00 sec)
```

INET_NTOA(adres)

Funkcja zwraca łańcuchową reprezentację numerycznego adresu internetowego — patrz: rysunek 12.64.

Rysunek 12.64.

*Funkcja
INET_NTOA*

```
mysql> SELECT INET_NTOA('1075594054');
+-----+
| INET_NTOA('1075594054') |
+-----+
| 64.28.67.70             |
+-----+
1 row in set (0.00 sec)
```

INSERT(ciąg, pozycja, długość, podciąg)

Funkcja wstawia łańcuch, opcjonalnie nadpisując istniejący podciąg. Drugi argument wskazuje znak, a trzeci określa, ile znaków usunąć od wskazanego znaku przed wstawieniem łańcucha. Rysunek 12.65 ilustruje wstawienie ciągu ABC w pozycji 3 z nadpisaniem jednego znaku.

Rysunek 12.65.
Funkcja
INSERT

```
mysql> SELECT INSERT('abcdefg', 3, 1, 'ABC');
+-----+
| INSERT('abcdefg', 3, 1, 'ABC') |
+-----+
| abABCdefg                       |
+-----+
1 row in set (0.00 sec)
```

INSTR(ciąg, podciąg)

Funkcja zwraca wskaźnik do pierwszego wystąpienia podciągu w danym ciągu — patrz: rysunek 12.66.

Rysunek 12.66.
Funkcja
INSTR

```
mysql> SELECT INSTR('samołot', 'łot');
+-----+
| INSTR('samołot', 'łot') |
+-----+
|                          5 |
+-----+
1 row in set (0.00 sec)
```

LCASE(ciąg)

Funkcja zwraca łańcuch z wielkimi literami zamienionymi na małe — patrz: rysunek 12.67. Aliasem jest LOWER.

Rysunek 12.67.
Funkcja
LCASE

```
mysql> SELECT LCASE('AbCd');
+-----+
| LCASE('AbCd') |
+-----+
| abcd          |
+-----+
1 row in set (0.00 sec)
```

LEFT(ciąg, długość)

Funkcja zwraca podciąg z danego ciągu o podanej długości — patrz: rysunek 12.68.

MATCH (...) AGAINST (ciąg)

Słowo kluczowe MATCH dopasowuje łańcuch do listy kolumn i zwraca liczbę zmiennoprzecinkową od 0,0 do 1,0. Wskazane kolumny muszą mieć przypisany indeks FULLTEXT. Omówienie tego typu indeksów znajduje się w rozdziale 11. „Typy kolumn i indeksów”.

MySQL dzieli ciąg tekstowy przekazany do instrukcji MATCH na poszczególne wyrazy. Znakami separującymi wyrazy jest spacja. Wyrazy mogą być również otoczone apostrofami lub wzięte w cudzysłów. Wyrazy trzyliterowe oraz krótsze zostają odrzucone. Instrukcja powoduje sortowanie wierszy według ich malejącej istotności. Wiersze nieistotne nie zostaną w ogóle zwrócone. Rysunek 12.75 przedstawia wyszukiwanie treści wiadomości w MySQL.

Rysunek 12.75.

Funkcja
MATCH

```
mysql> SELECT Lp
FROM wiadomosc
WHERE MATCH (tresc) AGAINST ('MySQL');
```

MD5(ciąg)

Funkcja zwraca 32-znakową wartość będącą efektem *mieszania* (ang. hash) zgodnie z dokumentem RFC 1321 — patrz: rysunek 12.76. Identyfikatory MD5 teoretycznie są niepowtarzalne dla każdego łańcucha.

Rysunek 12.76.

Funkcja MD5

```
mysql> SELECT MD5('Kim jest Lew Rywin?');
+-----+
| MD5('Kim jest Lew Rywin?') |
+-----+
| 2b6bb069b4321019d65928493117d54a |
+-----+
1 row in set (0.00 sec)
```

MID(ciąg, pozycja, długość)

Jest to alias funkcji SUBSTRING.

OCT(liczba_calk)

Funkcja zwraca ósemkową reprezentację dziesiętnej liczby całkowitej — patrz: rysunek 12.77.

Rysunek 12.77.

Funkcja OCT

```
mysql> SELECT OCT(16);
+-----+
| OCT(16) |
+-----+
| 20      |
+-----+
1 row in set (0.00 sec)
```

OCTET_LENGTH(ciąg)

Jest to alias funkcji LENGTH.

ORD(ciąg)

Funkcja zwraca liczbę porządkową dla pierwszego znaku z lewej strony w danym ciągu. W odróżnieniu od ASCII, ORD bierze pod uwagę znaki wielobajtowe.

PASSWORD(ciąg)

Funkcja szyfruje podane w formacie tekstowym hasło. Proces szyfrowania jest nieodwracalny. Funkcja ta jest przeznaczona do wstawiania wartości do kolumny haseł w `mysql.user` — tabeli uprawnień użytkowników. Rysunek 12.78 przedstawia kod SQL zmieniający hasło na „sekret”.

Rysunek 12.78.

*Funkcja
PASSWORD*

```
mysql> UPDATE user
SET Password=PASSWORD('sekret')
WHERE User='leon';
```

POSITION(podciąg IN ciąg)

Jest to inna forma funkcji LOCATE. Różnica polega na innej postaci argumentów — patrz: rysunek 12.79.

Rysunek 12.79.

*Funkcja
POSITION*

```
mysql> SELECT POSITION('b' IN 'abcabc');
+-----+
| POSITION('b' IN 'abcabc') |
+-----+
|                          2 |
+-----+
1 row in set (0.00 sec)
```

REPEAT(ciąg, powtórzenia)

Funkcja zwraca łańcuch złożony z danego ciągu, powtórnego określoną liczbę razy — patrz: rysunek 12.80.

Rysunek 12.80.

*Funkcja
REPEAT*

```
mysql> SELECT REPEAT('a', 10);
+-----+
| REPEAT('a', 10) |
+-----+
| aaaaaaaaaa      |
+-----+
1 row in set (0.00 sec)
```

REPLACE(w_ciągu, stary_ciąg, nowy_ciąg)

Funkcja zamienia każde wystąpienie starego ciągu w ciągu na nowy ciąg — patrz: rysunek 12.81.

Rysunek 12.81.
Funkcja
REPLACE

```
mysql> SELECT REPLACE('a-b-c-d', '-', '/');
+-----+
| REPLACE('a-b-c-d', '-', '/') |
+-----+
| a/b/c/d                        |
+-----+
1 row in set (0.00 sec)
```

REVERSE(ciąg)

Funkcja odwraca kolejność znaków w ciągu — patrz: rysunek 12.82.

Rysunek 12.82.
Funkcja
REVERSE

```
mysql> SELECT REVERSE('abcdef');
+-----+
| REVERSE('abcdef') |
+-----+
| fedcba             |
+-----+
1 row in set (0.00 sec)
```

RIGHT(ciąg, liczba)

Funkcja zwraca określoną liczbę znaków od prawej strony danego ciągu. Do pobierania znaków z lewej strony ciągu służy funkcja LEFT — patrz: rysunek 12.83.

Rysunek 12.83.
Funkcja
RIGHT

```
mysql> SELECT RIGHT('abcdef', 3);
+-----+
| RIGHT('abcdef', 3) |
+-----+
| def                 |
+-----+
1 row in set (0.00 sec)
```

RPAD(ciąg, długość, wypełnienie)

Funkcja dopełnia ciąg do określonej długości, dodając z prawej strony znaki podane w ciągu wypełnienia. Do wypełniania ciągu po lewej stronie służy funkcja LPAD. Rysunek 12.84 przedstawia prawostronne dopełnianie łańcucha 15 kropkami.

Rysunek 12.84.

Funkcja
RPAD

```
mysql> SELECT RPAD('abc', 15, '.');
+-----+
| RPAD('abc', 15, '.') |
+-----+
| abc.....          |
+-----+
1 row in set (0.00 sec)
```

RTRIM(ciąg)

Funkcja usuwa spacje po prawej stronie ciągu. Na rysunku 12.85 z ciągu usunięte zostają spacje z prawej strony, po czym jest on łączony z innym ciągiem, aby pokazać brak spacji.

Rysunek 12.85.

Funkcja
RTRIM

```
mysql> SELECT CONCAT(RTRIM('abc '), 'def');
+-----+
| CONCAT(RTRIM('abc '), 'def') |
+-----+
| abcdef                        |
+-----+
1 row in set (0.00 sec)
```

SOUNDEX(ciąg)

Funkcja zwraca ciąg będący rezultatem mieszania, oparty na tym, jakie jest brzmienie danego słowa. Donald Knuth opisał ten algorytm w trzecim tomie książki „Sztuka programowania” (WNT, 2003). Rezultaty mieszania są czteroznakowe i rozpoczynają się od litery. Rysunek 12.86 ilustruje niewielką różnicę w angielskiej wymowie słów „lion” i „lying”.

Rysunek 12.86.

Funkcja
SOUNDEX

```
mysql> SELECT SOUNDEX('lion'), SOUNDEX('lying');
+-----+-----+
| SOUNDEX('lion') | SOUNDEX('lying') |
+-----+-----+
| L500           | L520             |
+-----+-----+
1 row in set (0.00 sec)
```

SPACE(liczba)

Funkcja zwraca łańcuch złożony z podanej liczby spacji — patrz: rysunek 12.87. W zamian można zastosować funkcję REPEAT.

Rysunek 12.87.

Funkcja
SPACE

```
mysql> SELECT CONCAT('a', SPACE(10), 'b');
+-----+
| CONCAT('a', SPACE(10), 'b') |
+-----+
| a          b          |
+-----+
1 row in set (0.00 sec)
```

STRCMP(ciąg, ciąg)

Funkcja porównuje dwa łańcuchy i zwraca 0, jeżeli są one równe, -1 — jeżeli pierwszy poprzedza drugi w kolejności alfabetycznej, i 1 — jeżeli pierwszy występuje w kolejności alfabetycznej po drugim. Rysunek 12.88 pokazuje, że „abc” stoi przed „abd”.

Rysunek 12.88.

Funkcja
STRCMP

```
mysql> SELECT STRCMP('abc', 'abd');
+-----+
| STRCMP('abc', 'abd') |
+-----+
| -1 |
+-----+
1 row in set (0.00 sec)
```

SUBSTRING(ciąg FROM pozycja [FOR długość])

Funkcja zwraca podciąg. W tej wersji do separacji parametrów używane są słowa kluczowe. Jest to zgodne z normą ANSI.

SUBSTRING(ciąg, pozycja[, długość])

Funkcja zwraca podciąg rozpoczynający się od podanej pozycji — patrz: rysunek 12.89. Znaki w ciągu są numerowane od 1 i jeżeli nie została podana długość, zwrócona zostaje reszta ciągu.

Rysunek 12.89.

Funkcja
SUBSTRING

```
mysql> SELECT SUBSTRING('abcdef', 3, 3);
+-----+
| SUBSTRING('abcdef', 3, 3) |
+-----+
| cde          |
+-----+
1 row in set (0.00 sec)
```

SUBSTRING_INDEX(ciąg, separator, liczba)

Funkcja zwraca podciąg, który zawiera podaną liczbę elementów oddzielonych separatorami. Jeżeli ilość jest dodatnia, podciąg jest pobierany od lewej. Jeżeli ilość jest ujemna, podciąg jest pobierany od prawej. Na rysunku 12.90 zwrócone zostają pierwsze dwie wartości z listy elementów oddzielonych przecinkami w ciągu źródłowym.

Rysunek 12.90.
Funkcja
SUBSTRING_INDEX

```
mysql> SELECT SUBSTRING_INDEX('a,b,c,d', ',', 2);
+-----+
| SUBSTRING_INDEX('a,b,c,d', ',', 2) |
+-----+
| a,b                                |
+-----+
1 row in set (0.00 sec)
```

TRIM([BOTH|LEADING|TRAILING] wypełnienie FROM] ciąg)

Funkcja usuwa znaki wypełniające z ciągu. Domyślnie usuwa spacje po obu stronach łańcucha — patrz: rysunek 12.91.

Rysunek 12.91.
Funkcja
TRIM

```
mysql> SELECT TRIM('   abc   ');
+-----+
| TRIM('   abc   ') |
+-----+
| abc                |
+-----+
1 row in set (0.00 sec)
```

UCASE(ciąg)

Funkcja zwraca łańcuch z małymi literami zamienionymi na wielkie — patrz: rysunek 12.92. Zamianę ciągu na małe litery umożliwia LCASE.

Rysunek 12.92.
Funkcja
UCASE

```
mysql> SELECT UCASE('AbCd');
+-----+
| UCASE('AbCd') |
+-----+
| ABCD          |
+-----+
1 row in set (0.00 sec)
```

UPPER(ciąg)

Jest to alias funkcji UCASE.

Czas

Funkcje tu opisane operują na wartościach opisujących czas. W zależności od kontekstu wartości te przybierają formy kolumn, łańcuchów i liczb całkowitych. Na przykład funkcja NOW zwraca domyślnie łańcuch typu 2001-04-20 12:59:58, ale jeżeli oczekiwana jest liczba całkowita, to zwraca 20010420125958. Każda funkcja, która oczekuje daty lub czasu, zaakceptuje połączony czas i datę.

ADDDATE(data, INTERVAL typ wartości)

Jest to alias funkcji DATE_ADD.

CURDATE()

Funkcja zwraca bieżącą datę — patrz: rysunek 12.93. Można w tym celu użyć również CURRENT_DATE.

Rysunek 12.93.

Funkcja
CURDATE

```
mysql> SELECT CURDATE();
+-----+
| CURDATE() |
+-----+
| 2003-05-26 |
+-----+
1 row in set (0.01 sec)
```

CURRENT_DATE

Funkcja zwraca bieżącą datę — patrz: rysunek 12.94. Jak widać, nie wymaga ona żadnych nawiasów. Zamiennie można zastosować CURDATE.

Rysunek 12.94.

Funkcja
CURRENT_DATE

```
mysql> SELECT CURRENT_DATE;
+-----+
| CURRENT_DATE |
+-----+
| 2003-05-26 |
+-----+
1 row in set (0.00 sec)
```

CURRENT_TIME

Funkcja zwraca bieżący czas — patrz: rysunek 12.95. Jak widać, nie wymaga ona żadnych nawiasów. Zamiennie można zastosować CURTIME.

Rysunek 12.95.

Funkcja
CURRENT_TIME

```
mysql> SELECT CURRENT_TIME;
+-----+
| CURRENT_TIME |
+-----+
| 05:21:26     |
+-----+
1 row in set (0.00 sec)
```

CURRENT_TIMESTAMP

Funkcja zwraca bieżącą datę i czas — patrz: rysunek 12.96. Jak widać, nie wymaga ona żadnych nawiasów. Zamiennie można zastosować NOW.

Rysunek 12.96.

Funkcja
CURRENT
_TIMESTAMP

```
mysql> SELECT CURRENT_TIMESTAMP;
+-----+
| CURRENT_TIMESTAMP |
+-----+
| 2003-05-26 05:23:27 |
+-----+
1 row in set (0.00 sec)
```

CURTIME()

Funkcja zwraca bieżący czas — patrz: rysunek 12.97. Zamiennie można użyć CURRENT_TIME.

Rysunek 12.97.

Funkcja
CURTIME

```
mysql> SELECT CURTIME();
+-----+
| CURTIME() |
+-----+
| 05:25:12   |
+-----+
1 row in set (0.00 sec)
```

DATE_ADD(data, INTERVAL wartość typ)

Funkcja dodaje czas do daty lub datownika. Czas jest określany za pomocą słowa kluczowego INTERVAL, po którym następuje liczba całkowita lub ciąg tekstowy i oznaczenie typu. Typy i wymagane przez nie formaty zostały przedstawione w tabeli 12.1. Słowa oznaczają tu pełne liczby.

Tabela 12.1. Typy interwałów

Typ	Format
DAY	dni
DAY_HOUR	'dni godziny'
DAY_MINUTE	'dni godziny:minuty'
DAY_SECOND	'dni godziny:minuty:sekundy'
HOUR	godziny
HOUR_MINUTE	'godziny:minuty'
HOUR_SECOND	'godziny:minuty:sekundy'
MINUTE	minuty
MINUTE_SECOND	'minuty:sekundy'
MONTH	miesiące
SECOND	sekundy
YEAR	lata
YEAR_MONTH	'lata-miesiące'

Interwał czasowy można też dodać do daty za pomocą operatora dodawania (+), np. `NOW() + INTERVAL 1 DAY`. Aby odjąć czas od daty, należy posłużyć się funkcją `DATE_SUB` lub znakiem minusa. Pominięcie części wartości interwału powoduje, że MySQL przyporządkowuje wartości od prawej do lewej, a brakujące wartości ustawia na zero. Na przykład podanie 1:2 dla interwału `DAY_MINUTE` jest równorzędne użyciu `MINUTE_SECOND`. Rysunek 12.98 przedstawia czas bieżący i czas sprzed dwóch tygodni.

Rysunek 12.98.

Funkcja
`DATE_ADD`

```
mysql> SELECT NOW(), DATE_ADD(NOW(), INTERVAL 14 DAY);
+-----+-----+
| NOW()          | DATE_ADD(NOW(), INTERVAL 14 DAY) |
+-----+-----+
| 2003-05-26 05:39:25 | 2003-06-09 05:39:25 |
+-----+-----+
1 row in set (0.00 sec)
```

DATE_FORMAT(data, format)

Funkcja formatuje datę zgodnie z podanym formatem. Łańcuch formatujący może zawierać dowolną liczbę kodów rozpoczynających się od znaku procenta (%), symbolizujących poszczególne elementy daty. Reszta znaków łańcucha zostaje zwrócona w postaci dosłownej. Kody zostały wymienione w tabeli 12.2.

Tabela 12.2. Kody DATE_FORMAT

Kod	Opis	Przykłady
%%	Kod ucieczki dla %	%
%a	Skrócona nazwa dnia tygodnia	SUN...SAT
%b	Skrócona nazwa miesiąca	JAN...DEC
%c	Numer miesiąca nie poprzedzony zerem	1...12
%d	Dzień miesiąca z angielskim przyrostkiem	1ST, 2ND, 3RD, 4TH, ...
%D	Dzień miesiąca poprzedzony zerem	01...31
%e	Dzień miesiąca	1...31
%f	Mikrosekundy	000000...999999
%H	Godzina na zegarze 24-godzinnym poprzedzona zerem	01...23
%h	Godzina na zegarze 12-godzinnym poprzedzona zerem	01...12
%I	Równoznaczny %h	01...12
%i	Minuty poprzedzone zerami	00...59
%j	Numer dnia w roku poprzedzony zerami	001...366
%k	Godzina na zegarze 24-godzinnym niepoprzedzona zerem	0...23
%l	Godzina na zegarze 12-godzinnym niepoprzedzona zerem	1...12
%M	Nazwa miesiąca	JANUARY...DECEMBER
%m	Numer miesiąca poprzedzony zerem	01...12
%p	AM lub PM	AM, PM
%r	Czas na zegarze 12-godzinnym	01:15:30 AM
%S	Sekundy	00...59
%s	Sekundy	00...59
%T	Czas na zegarze 24-godzinnym	15:32:00
%u	Numer tygodnia w roku, gdzie poniedziałek jest pierwszym dniem tygodnia	0...53
%U	Numer tygodnia w roku, gdzie niedziela jest pierwszym dniem tygodnia	0...53
%v	Numer tygodnia w roku, gdzie poniedziałek jest pierwszym dniem tygodnia	1...53
%V	Numer tygodnia w roku, gdzie niedziela jest pierwszym dniem tygodnia	1...53
%w	Numer dnia tygodnia	0...6
%W	Nazwa dnia tygodnia	SUNDAY...SATURDAY
%x	Rok rozpoczynający się od poniedziałku — odpowiada numerowi tygodnia zwróconemu przez %v	0000...9999
%X	Rok rozpoczynający się od niedzieli — odpowiada numerowi tygodnia zwróconemu przez %v	0000...9999
%y	Rok w obrębie wieku	00...99
%Y	Rok	0000...9999

DAYOFWEEK(data)

Funkcja zwraca dzień tygodnia jako liczbę. Niedziela jest traktowana jako pierwszy dzień tygodnia. Z rysunku 12.102 wynika, że 1 stycznia 1970 roku był piątym dniem tygodnia, czyli czwartkiem.

Rysunek 12.102.
Funkcja
DAYOFWEEK

```
mysql> SELECT DAYOFWEEK('1970-01-01');
+-----+
| DAYOFWEEK('1970-01-01') |
+-----+
|                          5 |
+-----+
1 row in set (0.00 sec)
```

DAYOFYEAR(data)

Funkcja zwraca liczbę dni od początku roku, traktując 1 stycznia jako pierwszy dzień. Z rysunku 12.103 wynika, że 6 maja 1984 roku był 127. dniem roku.

Rysunek 12.103.
Funkcja
DAYOFYEAR

```
mysql> SELECT DAYOFYEAR('1984-05-06 01:30:00');
+-----+
| DAYOFYEAR('1984-05-06 01:30:00') |
+-----+
|                          127 |
+-----+
1 row in set (0.01 sec)
```

EXTRACT(typ FROM data)

Funkcja pobiera z daty wartość o określonym typie. Dopuszczalne typy to nazwy interwałów, akceptowane przez funkcję DATE_ADD. Wymieniono je w tabeli 12.1. Rysunek 12.104 przedstawia pobieranie roku z daty.

Rysunek 12.104.
Funkcja
EXTRACT

```
mysql> SELECT EXTRACT(YEAR FROM '1970-01-01');
+-----+
| EXTRACT(YEAR FROM '1970-01-01') |
+-----+
|                          1970 |
+-----+
1 row in set (0.00 sec)
```

FROM_DAYS(dni)

Funkcja zwraca datę na podstawie liczby dni od początku danego systemu kalendarzowego. Jednak daty przed rokiem 1582 nie są przedstawiane dokładnie. MySQL nie bierze pod uwagę zmian, jakie nastąpiły z chwilą wprowadzenia kalendarza gregoriańskiego. Rysunek 12.105 demonstruje zastosowanie tej funkcji. Odwrotne działanie ma funkcja `T0_DAYS`.

Rysunek 12.105.
Funkcja
`FROM_DAYS`

```
mysql> SELECT FROM_DAYS(719528);
+-----+
| FROM_DAYS(719528) |
+-----+
| 1970-01-01        |
+-----+
1 row in set (0.00 sec)
```

FROM_UNIXTIME(sekundy[, format])

Funkcja zwraca datę na podstawie datownika UNIX-a, który jest liczbą sekund od 1 stycznia 1970 roku. Opcjonalny argument `format` definiuje format zwracanego łańcucha. Może on zawierać te same kody, których używa się z `DATE_FORMAT` — patrz: tabela 12.2. Wartości przekazane do `FROM_UNIXTIME` są przeznaczone dla czasu Greenwich (GMT). Rysunek 12.106 przedstawia rezultaty dla komputera w strefie środkowoeuropejskiej.

Rysunek 12.106.
Funkcja
`FROM_UNIXTIME`

```
mysql> SELECT FROM_UNIXTIME(28800);
+-----+
| FROM_UNIXTIME(28800) |
+-----+
| 1970-01-01 09:00:00 |
+-----+
1 row in set (0.00 sec)
```

HOUR(czas)

Funkcja zwraca godzinę na podstawie podanego czasu — patrz: rysunek 12.107.

Rysunek 12.107.
Funkcja
`HOUR`

```
mysql> SELECT HOUR('01:23:45');
+-----+
| HOUR('01:23:45') |
+-----+
|                  1 |
+-----+
1 row in set (0.00 sec)
```

MINUTE(czas)

Funkcja zwraca minutę na podstawie podanego czasu — patrz: rysunek 12.108.

Rysunek 12.108.

Funkcja
MINUTE

```
mysql> SELECT MINUTE('01:23:45');
+-----+
| MINUTE('01:23:45') |
+-----+
|                    23 |
+-----+
1 row in set (0.01 sec)
```

MONTH(data)

Funkcja zwraca miesiąc jako liczbę na podstawie podanej daty, począwszy od 1 — patrz: rysunek 12.109.

Rysunek 12.109.

Funkcja
MONTH

```
mysql> SELECT MONTH('1970-01-01');
+-----+
| MONTH('1970-01-01') |
+-----+
|                      1 |
+-----+
1 row in set (0.00 sec)
```

MONTHNAME(data)

Funkcja zwraca nazwę miesiąca na podstawie podanej daty — patrz: rysunek 12.110.

Rysunek 12.110.

Funkcja
MONTHNAME

```
mysql> SELECT MONTHNAME('1970-01-01');
+-----+
| MONTHNAME('1970-01-01') |
+-----+
| January                  |
+-----+
1 row in set (0.00 sec)
```

NOW()

Funkcja zwraca bieżącą datę i czas. Zwrócona wartość będzie odpowiednia do kontekstu wywołania. Rysunek 12.111 przedstawia wersję łańcuchową.

Rysunek 12.111.
Funkcja *NOW*

```
mysql> SELECT NOW();
+-----+
| NOW() |
+-----+
| 2003-05-26 07:22:12 |
+-----+
1 row in set (0.00 sec)
```

PERIOD_ADD(okres, miesiące)

Funkcja dodaje określoną liczbę miesięcy do okresu, który jest miesiącem pewnego roku. Z rysunku 12.112 wynika, że 15 miesięcy po styczniu 1970 nastąpi kwiecień 1971 roku.

Rysunek 12.112.
Funkcja
PERIOD_ADD

```
mysql> SELECT PERIOD_ADD(197001, 15);
+-----+
| PERIOD_ADD(197001, 15) |
+-----+
| 197104 |
+-----+
1 row in set (0.00 sec)
```

PERIOD_DIFF(okres, okres)

Funkcja zwraca różnicę w miesiącach pomiędzy dwoma okresami. Okres to miesiąc określonego roku. Z rysunku 12.113 wynika, że styczeń 1970 roku był o 15 miesięcy wcześniej niż kwiecień roku 1971.

Rysunek 12.113.
Funkcja
PERIOD_DIFF

```
mysql> SELECT PERIOD_DIFF(197001, 197104);
+-----+
| PERIOD_DIFF(197001, 197104) |
+-----+
| -15 |
+-----+
1 row in set (0.00 sec)
```

QUARTER(data)

Funkcja zwraca kwartał roku dla danej daty. Pierwszy kwartał rozpoczyna się wraz z początkiem roku i obejmuje pierwsze trzy miesiące. Z rysunku 12.114 wynika, że 1 stycznia 1970 roku przypada w pierwszym kwartale.

Rysunek 12.114.

*Funkcja
QUARTER*

```
mysql> SELECT QUARTER('1970-01-01');
+-----+
| QUARTER('1970-01-01') |
+-----+
| 1 |
+-----+
1 row in set (0.00 sec)
```

SECOND(czas)

Funkcja zwraca sekundy z podanego czasu — patrz: rysunek 12.115.

Rysunek 12.115.

*Funkcja
SECOND*

```
mysql> SELECT SECOND('01:23:45');
+-----+
| SECOND('01:23:45') |
+-----+
| 45 |
+-----+
1 row in set (0.00 sec)
```

SEC_TO_TIME(sekundy)

Funkcja zwraca czas na podstawie liczby sekund — patrz: rysunek 12.116. Nie ma tu ograniczenia do wartości, które mieszczą się w obrębie jednego dnia. Podanie liczby sekund większej niż 86 400 spowoduje po prostu zwrócenie wartości większej niż 24 godziny. Odwrotne działanie ma funkcja TIME_TO_SEC.

Rysunek 12.116.

*Funkcja
SEC_TO_TIME*

```
mysql> SELECT SEC_TO_TIME(5025);
+-----+
| SEC_TO_TIME(5025) |
+-----+
| 01:23:45 |
+-----+
1 row in set (0.00 sec)
```

SUBDATE(data, INTERVAL wartość typ)

Jest to alias funkcji DATE_SUB.

SYSDATE()

Jest to alias funkcji NOW.

TIME_FORMAT(czas, format)

Funkcja zwraca czas sformatowany zgodnie z kodami podanymi w argumencie format. Dopuszczalne kody są takie same, jak w przypadku funkcji DATE_FORMAT. Rysunek 12.117 przedstawia czas sformatowany dla zegara dwunastogodzinnego.

Rysunek 12.117.
Funkcja
TIME_FORMAT

```
mysql> SELECT TIME_FORMAT('23:45:01', '%r');
+-----+
| TIME_FORMAT('23:45:01', '%r') |
+-----+
| 11:45:01 PM                    |
+-----+
1 row in set (0.00 sec)
```

TIME_TO_SEC(czas)

Funkcja zwraca liczbę sekund, jaką reprezentuje podany czas — patrz: rysunek 12.118. Wykonuje ona operację odwrotną do funkcji SEC_TO_TIME.

Rysunek 12.118.
Funkcja
TIME_TO_SEC

```
mysql> SELECT TIME_TO_SEC('01:23:45');
+-----+
| TIME_TO_SEC('01:23:45') |
+-----+
| 5025                      |
+-----+
1 row in set (0.04 sec)
```

TO_DAYS(data)

Funkcja zwraca liczbę dni od początku systemu kalendarzowego do danej daty — patrz rysunek 12.119. Daty przed 1582 rokiem nie są odwzorowywane dokładnie w związku ze zmianami dokonanymi w kalendarzu gregoriańskim. Funkcja wykonuje operację odwrotną do FROM_DAYS. Za jej pomocą można obliczać liczbę dni dzielącą dwie daty. Na przykład TO_DAYS("2001-09-01")-TO_DAYS("2001-02-01") zwraca 212 — liczbę dni od 1 września 2001 do 1 lutego 2001 roku.

Rysunek 12.119.
Funkcja
TO_DAYS

```
mysql> SELECT TO_DAYS('1970-01-01');
+-----+
| TO_DAYS('1970-01-01') |
+-----+
| 719528                 |
+-----+
1 row in set (0.00 sec)
```


YEAR(data)

Funkcja zwraca rok dla podanej daty — patrz: rysunek 12.123.

Rysunek 12.123.

Funkcja
YEAR

```
mysql> SELECT YEAR('1970-06-06');
+-----+
| YEAR('1970-06-06') |
+-----+
|                1970 |
+-----+
1 row in set (0.01 sec)
```

YEARWEEK(data[, pierwszy_dzień])

Funkcja zwraca okres na podstawie daty. Drugi argument decyduje o tym, czy tygodnie zaczynają się od niedzieli, czy od poniedziałku. W pierwszym przypadku należy użyć 0, a w drugim 1. Z rysunku 12.124 wynika, że 6 czerwca 1970 przypada w 22. tygodniu tego roku.

Rysunek 12.124.

Funkcja
YEARWEEK

```
mysql> SELECT YEARWEEK('1970-06-06', 0);
+-----+
| YEARWEEK('1970-06-06', 0) |
+-----+
|                197022 |
+-----+
1 row in set (0.00 sec)
```

Pozostałe funkcje

Opisane tu funkcje nie pasują do żadnej z wcześniejszych kategorii.

BIT_COUNT(liczba_calk)

Funkcja traktuje podaną liczbę całkowitą jako liczbę dwójkową i zwraca liczbę ustawionych w niej bitów. Z rysunku 12.125 wynika, że liczba dziesiętna 19 ma w swej postaci dwójkowej ustawione 3 bity.

Rysunek 12.125.

Funkcja
BIT_COUNT

```
mysql> SELECT BIT_COUNT(19), BIN(19);
+-----+
| BIT_COUNT(19) | BIN(19) |
+-----+
|                3 | 10011 |
+-----+
1 row in set (0.01 sec)
```

COALESCE(...)

Funkcja zwraca pierwszy element, licząc od lewej strony listy, który nie jest wartością null. Jeżeli wszystkie elementy są puste, zwracana jest wartość null. Rysunek 12.126 przedstawia jej działanie.

Rysunek 12.126.

Funkcja
COALESCE

```
mysql> SELECT COALESCE(NULL, NULL, 1, NULL, 2, 3);
+-----+
| COALESCE(NULL, NULL, 1, NULL, 2, 3) |
+-----+
| 1 |
+-----+
1 row in set (0.01 sec)
```

INTERVAL(cel, ...)

Funkcja zwraca pozycję na liście elementów, od której cel jest większy, ale jednocześnie mniejszy niż kolejna wartość na liście. Elementy listy muszą być posortowane od najmniejszego do największego. W obrębie listy wykonywane jest wyszukiwanie binarne — pierwszy element ma numer 1. Z rysunku 12.127 wynika, że 9 jest większe niż piąty element, którym jest 7, i mniejsze od 11 na pozycji szóstej.

Rysunek 12.127.

Funkcja
INTERVAL

```
mysql> SELECT INTERVAL(9,1,2,3,5,7,11,13,17);
+-----+
| INTERVAL(9,1,2,3,5,7,11,13,17) |
+-----+
| 5 |
+-----+
1 row in set (0.00 sec)
```

ISNULL(wartość)

Funkcja ISNULL zwraca 1, jeżeli argument ma wartość „null”, lub 0 — w innych przypadkach. Z rysunku 12.128 wynika, że do ISNULL przesłane zostały trzy wyrażenia.

Rysunek 12.128.

Funkcja
ISNULL

```
mysql> SELECT ISNULL(1), ISNULL(null), ISNULL(1+NULL);
+-----+-----+-----+
| ISNULL(1) | ISNULL(null) | ISNULL(1+NULL) |
+-----+-----+-----+
| 0 | 1 | 1 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

Procedury

Procedury to funkcje, które działają na zbiorze wyników, zanim zostanie on przesłany do klienta. Wywołuje się je na końcu instrukcji wyboru. Do MySQL załączona jest tylko jedna procedura — `analyse`. W rozdziale 31. „Rozszerzanie możliwości” opisano tworzenie własnych procedur.

`analyse`([Elementy L, pamięć])

Procedura zwraca analizę tabeli. Możliwe jest ograniczenie maksymalnej liczby elementów branych pod uwagę dla każdej kolumny. Można również podać maksymalną ilość pamięci do wykorzystania podczas analizy — patrz: rysunek 12.129.

Rysunek 12.129.
Procedura `analyse`

```
mysql> SELECT Nazwa, Cena
-> FROM produkty
-> WHERE Lp IN (1,2,3)
-> PROCEDURE analyse() \G;
***** 1. row *****
      Field_name: produkty.Nazwa
      Min_value: Grzebień
      Max_value: Szczotka
      Min_length: 8
      Max_length: 11
      Empties_or_zeros: 0
      Nulls: 0
      Avg_value_or_avg_length: 9.0000
      Std: NULL
      Optimal_fieldtype: ENUM('Grzebień','Szczoteczka','Szczotka') NOT NULL
***** 2. row *****
      Field_name: produkty.Cena
      Min_value: 1.25
      Max_value: 3.15
      Min_length: 4
      Max_length: 4
      Empties_or_zeros: 0
      Nulls: 0
      Avg_value_or_avg_length: 2.30
      Std: 0.79
      Optimal_fieldtype: ENUM('1.25','2.50','3.15') NOT NULL
2 rows in set (0.01 sec)
```