MySQL

Trumpas kalbos aprašas

Aprašą sudaro dvi dalys. Pirmojoje dalyje (MySQLlt1.doc) rasite tokius skyrelius:

Antrojoje dalyje (šiame faile) rasite tokius skyrelius:

1. Literalai

Eilutė

Eilutė – simbolių seka, įrėminta paprastomis (`'') arba dvigubomis kabutėmis (`"').:

```
'a string'
"another string"
```

Sveikieji skaičiai

Sveikieji skaičiai vaizduojami skaitmenų sekomis. Slankaus kablelio skaičiuose naudojamas dešimtainės dalies skirtukas `.'.

Sveikųjų skaičių pavyzdžiai:

```
1221        0           -32
```

Slankaus kablelio skaičių pavyzdžiai

```
294.42          -32032.6809e+10  148.00
```

NULL

NULL reiškia “duomenų nėra”. Tai ne tas pats, kaip 0 skaičiams ar tuščia eilutė.

NULL gali būti atvaizduotas kaip \N, jei naudojame tekstinio failo importą ar eksportą (**LOAD DATA INFILE**, **SELECT** ... **INTO OUTFILE**).

Duomenų bazės, lentelės, indekso, stulpelių vardai ir pseudonimai

Duomenų bazės, lentelės, indekso, stulpelių vardai ir pseudonimai sudaromi pagal vieningas taisykles:

Vardas	max ilgis	Leisti simboliai
Duomenų bazės	64	Bet kokie katalogų varduose leisti simboliai, išskyrus /.
Lentelės	64	Bet kokie failų varduose leisti simboliai, išskyrus / arba .
Stulpelio	64	All characters
Pseudonimas	255	All characters

Jei vardas sutampa su rezervuotu žodžiu ar jame yra specialus simbolis, tokį vardą įrėminkite kabutėmis:

```
SELECT * from `select` where `select`.id > 100;
```

MySQL galimos tokios stulpelių nuorodos:

Stulpelio nuoroda	***Reiškia***
col_name	Stulpelis col_name bet kurioje lentelėje, panaudotas užklausje
tbl_name.col_name	Stulpelis col_name tbl_name šioje
db_name.tbl_name.col_name	Stulpelis col_name iš lentelės tbl_name duomenų bazėje db_name. Priimta MySQL 3.22 ar vėlesnėse versijose.
`column_name`	Stulpelis, kuris nurodytas kaip raktas ar jo pavadinime panaudoti spec. simboliai

Užrašas .tbl_name reiškia, kad lentelė tbl_name yra šioje duomenų bazėje. Tokia sintaksė priimta ODBC suderinamumui, nes kai kurios ODBC programos prieš lentelių pavadinimus rašo `.' simbolį

Ar varduose mažosios raidės skiriamos nuo didžiųjų - priklauso nuo OS.

Į pradžią

2. Vartotojo kintamieji

Kintamųjų inicializuoti nereikia. Jie prilyginami **NULL** pagal nutylėjimą. Gijoje panaudoti kintamieji automatiškai atlaisvinami pabaigus giją.

Kintamųjų reikšmes galima nustatyti naudojant **SET**:

SET @variable= { integer expression | real expression | string expression }[,@variable= ...].

Kintamųjų reikšmes galima nustatyti ir išraiškoje, panaudojus sintaksę @**variable**:=**expr**:

```
select @t1:=(@t2:=1)+@t3:=4,@t1,@t2,@t3;
```

| pradžią

3. Stulpeliai

3.1. Stulpelių tipai

MySQL palaiko tris stulpelių tipus: skaitinius, datų ir laiko bei eilučių tipo.

Aprašydami naudosime tokius žymėjimus:

M - nurodo rodomą ilgą (max 255)
D - nurodo slankaus kablelio skaičiaus skaitmenų skaičių po kablelio. Maksimali reikšmė - 30, tačiau ji negali viršyti M-2.

Kvadratiniai skliaustai { ir } nurodo nebūtinus elementus.

Jei stulpeliui n nurodysite **ZEROFILL**, MySQL automatiškai pridės atributą **UNSIGNED**.

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

Labai mažas sveikasis: su ženklu – nuo -128 iki 127; be ženklo – nuo 0 iki 255.

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

Mažas sveikasis: su ženklu – nuo -32768 iki 32767; be ženklo – nuo 0 iki 65535.

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

Vidutinio dydžio sveikasis: su ženklu – nuo -8388608 iki 8388607; be ženklo – nuo 0 iki 16777215.

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

Normalaus dydžio sveikasis: su ženklu – nuo -2147483648 iki 2147483647; be ženklo – nuo 0 iki 4294967295.

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

Tai yra **INT** sinonimas.

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

Didelis sveikasis: su ženklu – nuo -9223372036854775808 iki 9223372036854775807; be ženklo – nuo 0 iki 18446744073709551615.

FLOAT[(M,D)] [ZEROFILL]

Mažasis (normalaus tikslumo) slankaus kablelio skaičius. Būtinai turi ženklą. Galimos reikšmės nuo -3.402823466E+38 iki -1.175494351E-38, 0 ir nuo 1.175494351E-38 iki 3.402823466E+38. **FLOAT** be argumento arba su argumentu <= 24 reiškia normalaus tikslumo slankaus kablelio skaičių.

DOUBLE[(M,D)] [ZEROFILL]

Normalaus dydžio (dvigubo tikslumo) slankaus kablelio skaičius. Būtinai turi ženklą. Galimos reikšmės nuo -1.7976931348623157E+308 iki -2.2250738585072014E-308, 0 ir nuo 2.2250738585072014E-308 iki 1.7976931348623157E+308. Čia M nurodo skilčių skaičių, o D trupmeminės dalias ilgį. **DOUBLE** be argumento arba **FLOAT**(X), kai 25 <= X <= 53, reiškia dvigubo tikslumo slankaus kablelio skaičių.

DOUBLE PRECISION[(M,D)] [ZEROFILL]
REAL[(M,D)] [ZEROFILL]

Tai yra **DOUBLE** sinonimas.

DECIMAL[(M[,D])] [ZEROFILL]

Nepakuotas slankaus kablelio skaičius. Būtinai turi ženklą. Vaizduojamas kaip **CHAR** stulpelis: “Nepakuota”' reiškia, kad skaičius saugomas kaip eilutė, vienam skaitmenius skiriant vieną simbolį. Į M įeina dešimtainis taškas (kablelis) ir `-' ženklas (neigiamiems skaičiams). Jei D lygu 0, turime tik sveiką dalį. Galimas reikšmes apsprendžia M ir D. Jei D nenurodytas, laikoma, kad jis lygus 0. Jei M nenurodytas, laikoma, kad jis lygus 10.

NUMERIC(M,D) [ZEROFILL]

Tai yra **DECIMAL** sinonimas.

DATE

Data. Galimos reikšmės nuo '1000-01-01' iki '9999-12-31'. MySQL **DATE** reikšmes rodo 'YYYY-MM-DD' formatu, bet jas galima priskirti kaip eilutes ar skaičius.

DATETIME

Datos ir laiko kombinacija. Galimos reikšmės nuo '1000-01-01 00:00:00' iki '9999-12-31 23:59:59'. MySQL **DATETIME** reikšmes rodo 'YYYY-MM-DD HH:MM:SS' formatu, bet jas galima priskirti kaip eilutes ar skaičius.

TIMESTAMP[(M)]

Laiko žyma. Galimos reikšmės nuo '1970-01-01 00:00:00' iki 2037 metų. MySQL **TIMESTAMP** reikšmes rodo YYYYMMDDHHMMSS, YYMMDDHHMMSS, YYYYMMDD arba YYMMDD formatu, priklausomai nuo M: 14 (ar nenurodytas), 12, 8 arba 6, bet jas galima priskirti kaip eilutes ar skaičius.

TIMESTAMP stulpelis naudingas datai ir laikui įterpti **INSERT** arba **UPDATE** operacijomis, nes leidžia automatiškai fiksuoti paskutinio veiksmo datą ir laiką.

TIME

Laikas. Galimos reikšmės nuo '-838:59:59' iki '838:59:59'. MySQL **TIME** reikšmes rodo 'HH:MM:SS' formatu, bet jas galima priskirti kaip eilutes ar skaičius.

YEAR[(2|4)]

Metai 2- arba 4- skaitmenų formatu (pagal nutylėjimą – 4). Galimos reikšmės nuo 1901 iki 2155, ir 0000 4 skaitmenų formatu ir 1970-2069 – 2 skaitmenų formatu (70-69). MySQL **YEAR** reikšmes rodo YYYY formatu, bet jas galima priskirti kaip eilutes ar skaičius.

CHAR(M) [BINARY]

Fiksuoto ilgio eilutė, įrašant glaudžiama prie dešiniojo krašto. M = 1-255. **CHAR** rikiuojamos ir palyginamos nepaisant registro.

[NATIONAL] VARCHAR(M) [BINARY]

Kintamo ilgio eilutė. M = 1-255. **VARCHAR** rikiuojamos ir palyginamos nepaisant registro (jei nenurodyta **BINARY**).

TINYBLOB
TINYTEXT

BLOB (*Binary Large OBject*) arba **TEXT** stulpelis, kurio maksimalus ilgis 255 (2^8 - 1) simboliai. **BLOB** rikiuojant ir palyginant atsižvelgiama į registrą.

BLOB
TEXT

BLOB arba **TEXT** stulpelis, kurio maksimalus ilgis 65535 (2^16 - 1) simboliai.

MEDIUMBLOB
MEDIUMTEXT

BLOB arba **TEXT** stulpelis, kurio maksimalus ilgis 16777215 (2^24 - 1) simboliai.

LONGBLOB
LONGTEXT

BLOB arba **TEXT** stulpelis, kurio maksimalus ilgis 4294967295 (2^32 - 1) simboliai.

ENUM('value1','value2',...)

Išvardijimas. Eilutės tipo objektas, galintis įgyti tik vieną reikšmę iš sąrašo **'value1'**, **'value2'**, ..., arba **NULL**. Maksimalus skirtingų reikšmių skaičius – 65535.

SET('value1','value2',...)

Rinkinys. Eilutės tipo objektas, galintis įgyti 0 ar kelias reikšmes iš sąrašo **'value1'**, **'value2'**, ... Maksimalus skirtingų reikšmių skaičius – 64.

Į 3 skyrelio pradžią

Į pradžią

3.2. Stulpelio tinkamo tipo parinkimas

Norint maksimaliai efektyviai naudoti atmintį, reikia kiekvieną kartą pasirinkti tinkamiausią stulpelio tipą. Pavyzdžiui, jei stulpelyje bus sveikieji skaičiai nuo 1 iki 99999, pasirinkite **MEDIUMINT UNSIGNED**.

Daugeliu atvejų svarbu pasirinkti tinkamą stulpelio tipą pinigų reikšmėms saugoti. MySQL naudokite tipą **DECIMAL**. Ji saugoma eilutės pavidalu, tad nebus paklaidos dėl tikslumo.

Į 3 skyrelio pradžią

Į pradžią

3.3. Stulpelio indeksai

Visų MySQL tipų stulpeliai gali būti indeksuoti. Tai padidina **SELECT** operacijų vykdymo efektyvumą.

Lentelėje gali būti indeksuota iki 16 stulpelių. Maksimalus indekso ilgis – 256 baitai.

CHAR ir **VARCHAR** tipo stulpeliams galima indeksuoti stulpelio prefiksą. Tai žymiai sparčiau ir reikalauja našiau vietos diske. **CREATE TABLE** operatoriuje tai nurodoma taip:

```
KEY index_name (col_name(length))
```

Žemiau pateiktame pavyzdyje sukuriamas pirmųjų 10 simbolių indeksas stulpeliui ***name***:

```
mysql> CREATE TABLE test (name CHAR(200) NOT NULL,
                          KEY index_name (name(10)));
```

BLOB ir **TEXT** stulpeliams visada reikia taip daryti.

| 3 skyrelio pradžią

| pradžią

3.4. Kelių stulpelių indeksai

MySQL gali suformuoti kelių (iki 15) stulpelių indeksus. (**CHAR** ir **VARCHAR** stulpeliams kaip indekso dalį galima naudoti prefiksus). Kelių stulpelių indeksas suprantamas kaip jų reikšmių konkatenacijos bendras indeksas.

MySQL kelių stulpelių indeksą panaudoja greitai užklausų realizacijai.

Tarkime, sukurta lentelė tokiu būdu:

```
mysql> CREATE TABLE test (
           id INT NOT NULL,
           last_name CHAR(30) NOT NULL,
           first_name CHAR(30) NOT NULL,
           PRIMARY KEY (id),
           INDEX name (last_name,first_name));
```

Čia indeksas ***name*** sukurtas stulpeliams ***last_name*** ir ***first_name***. Jį panaudosime užklausose, kuriose bus apibrėžtos ***last_name*** reikšmės, arba abiejų stulpelių (***last_name*** ir ***first_name***) reikšmės. Taigi, ***name*** indeksą panaudosime tokiose užklausose:

```
mysql> SELECT * FROM test WHERE last_name="Widenius";
mysql> SELECT * FROM test WHERE last_name="Widenius"
                          AND first_name="Michael";
mysql> SELECT * FROM test WHERE last_name="Widenius"
          AND (first_name="Michael" OR first_name="Monty");
mysql> SELECT * FROM test WHERE last_name="Widenius"
               AND first_name >="M" AND first_name < "N";
```

Tačiau ***name*** indekso negalėsime naudoti tokiose užklausose:

```
mysql> SELECT * FROM test WHERE first_name="Michael";
mysql> SELECT * FROM test WHERE last_name="Widenius"
                          OR first_name="Michael";
```

3.5. Kitų DBVS stulpelių tipai

Kitų DBVS panaudojimui SQL užklausose palengvinti MySQL tokiu būdu atvaizduoja tipus:

Kitose DBVS	*MySQL tipas*
BINARY(NUM)	CHAR(NUM) BINARY
CHAR VARYING(NUM)	VARCHAR(NUM)
FLOAT4	FLOAT
FLOAT8	DOUBLE
INT1	TINYINT
INT2	SMALLINT
INT3	MEDIUMINT
INT4	INT
INT8	BIGINT
LONG VARBINARY	MEDIUMBLOB
LONG VARCHAR	MEDIUMTEXT
MIDDLEINT	MEDIUMINT
VARBINARY(NUM)	VARCHAR(NUM) BINARY

Atitikimas realizuojamas sukuriant lentelę.

Į 3 skyrelio pradžią

Į pradžią

4. SELECT ir WHERE operatoriuose naudojamos funkcijos

NULL turinti išraiška visada duoda reikšmę **NULL**, jei tik dokumentacijoje nenurodyta kitaip.

Pastaba: Tarp funkcijos pavadinimo ir skliausto negali būti tarpo. Argumentai gali būti atskirti tarpais.

4.1 Grupavimo funkcijos

(...)

Skliaustai. Reikalingi išraiškos skaičiavimo tvarkai apibrėžti.

```
mysql> select 1+2*3;
        -> 7
mysql> select (1+2)*3;
        -> 9
```

Į 4 skyrelio pradžią

Į pradžią

4.2 Įprastos aritmetinės operacijos

Įprastos aritmetinės operacijos:

+ sudėtis - atimtis

```
mysql> select 3+5;          mysql> select 3-5;
       -> 8                        -> -2
```

---

* daugyba / dalyba

```
mysql> select 3*5;          mysql> select 3/5;
```

```
        -> 15                          -> 0.60
```
Dalyba iš nulio duoda NULL:

Į 4 skyrelio pradžią

Į pradžią

4.3 Bitų funkcijos

Bitų operacijoms MySQL naudoja **BIGINT** (64-bit) aritmetiką, taigi, šie operatoriai duoda maksimalų ilgį 64 bitus.

	OR (pabičiui)	&	**AND** (pabičiui)

```
mysql> select 29 | 15;          mysql> select 29 & 15;
        -> 31                          -> 13
```

<<	postūmis į kairę	>>	postūmis į dešinę

```
mysql> select 1 << 2           mysql> select 4 >> 2
        -> 4                           -> 1
```

~	visų bitų inversija	BIT_COUNT(N)	vienetukų skaičius

```
mysql> select 5 & ~1           mysql> select BIT_COUNT(29);
        -> 4                           -> 4
```

Į 4 skyrelio pradžią

Į pradžią

4.4 Loginės operacijos

Jų rezultatas – 1 (**TRUE**) arba 0 (**FALSE**).

NOT arba ! loginė NE | OR arba || loginė ARBA

```
mysql> select NOT 1;           mysql> select 1 || 0;
        -> 0                          -> 1
mysql> select NOT NULL;        mysql> select 0 || 0;
        -> NULL                       -> 0
mysql> select ! (1+1);         mysql> select 1 || NULL;
        -> 0                          -> 1
mysql> select ! 1+1;
        -> 1
```

NOT NULL duoda **NULL**. | Duoda 1 jei argumentas nelygus 0 ir ne **NULL**

AND arba && loginė IR

```
mysql> select 1 && NULL;
        -> 0
mysql> select 1 && 0;
        -> 0
```

Į 4 skyrelio pradžią

Į pradžią

4.5 Palyginimo operacijos

Jų rezultatas būna lygus 1 (**TRUE**), 0 (**FALSE**) arba **NULL**. Šios funkcijos atliekamos su skaičiais ir eilutėmis. Eilutės automatiškai konvertuojamos į skaičius arba atvirkščiai, kai to reikia (kaip Perl).

Palyginimo operacijos MySQL atliekamos pagal tokias taisykles:
•jei vienas ar abu argumentai lygūs **NULL**, rezultatas yra **NULL**, išskyrus operaciją <=>;
- jei abu argumentai yra eilutės, jie palyginami kaip eilutės;

•jei abu argumentai yra skaičiai, jie palyginami kaip skaičiai;
- jei vienas argumentas yra **TIMESTAMP** ar **DATETIME** tipo, o kitas – konstanta, prieš palyginant ši konvertuojama į **TIMESTAMP** ;
- visais kitais atvejais argumentai palyginami kaip slankaus kablelio skaičiai.

Pavyzdžiai:

```
mysql> SELECT 1 > '6x';
         -> 0
mysql> SELECT 7 > '6x';
         -> 1
mysql> SELECT 0 > 'x6';
         -> 0
mysql> SELECT 0 = 'x6';
         -> 1
```

= lygu

```
mysql> select 1 = 0;
        -> 0
mysql> select '0' = 0;
        -> 1
mysql> select '0.0' = 0;
        -> 1
mysql> select '0.01' = 0;
        -> 0
mysql> select '.01' = 0.01;
        -> 1
```

<> arba **!=** nelygu

```
mysql> select '.01' <> '0.01';
-> 1
mysql> select .01 <> '0.01';
-> 0
mysql> select 'zapp' <> 'zappp';
-> 1
```

<= mažiau arba lygu

```
mysql> select 0.1 <= 2;
        -> 1
```

< mažiau

```
mysql> select 2 <= 2;
        -> 1
```

>= daugiau arba lygu

```
mysql> select 2 >= 2;
        -> 1
```

> daugiau

```
mysql> select 2 > 2;
        -> 0
```

<=> nelygu

```
mysql> select 1 <=> 1, NULL <=> NULL, 1 <=> NULL;
        -> 1 1 0
```

IS NULL

IS NOT NULL ar lygu (nelygu) **NULL**

```
mysql> select 1 IS NULL, 0 IS NULL, NULL IS NULL:
        -> 0 0 1
```

```
mysql> select 1 IS NOT NULL, 0 IS NOT NULL, NULL IS NOT NULL;
        -> 1 1 0
```

---

expr BETWEEN min AND max

Jei **expr** >= **min** ir **expr** <= **max**, **BETWEEN** duoda 1,priešingu atveju – 0.

Pirmasis (**expr**) argumentas apsprendžia palyginimo vykdymą:

- jei tai yra **TIMESTAMP**, **DATE** arba **DATETIME** stulpelis, min ir max are formatuojami vienodai, jei tai – konstantos;
- jei tai yra registrui nejautri išraiška, palyginimos kaip registrui nejautrios eilutės;
- jei tai yra registrui jautri išraiška, palyginimos kaip registrui jautrios eilutės;
- jei tai yra sveikųjų skaičių išraiška, palyginimi kaip sveikieji skaičiai;
- visais kitais atvejais palyginami kaip slankaus kablelio skaičiai. •

```
mysql> select 1 BETWEEN 2 AND 3;
        -> 0
mysql> select 'b' BETWEEN 'a' AND 'c';
        -> 1
mysql> select 2 BETWEEN 2 AND '3';
        -> 1
mysql> select 2 BETWEEN 2 AND 'x-3';
        -> 0
```

---

expr IN (value,...)

Duoda 1, jef **expr** lygi vienai iš sąraše **IN list** nurodytų reikšmių, kitaip duoda 0.

```
mysql> select 2 IN (0,3,5,'wefwf');
        -> 0
mysql> select 'wefwf' IN (0,3,5,'wefwf');
        -> 1
```

---

expr NOT IN (value,...)

Atitinka **NOT (expr IN (value,...)).**

ISNULL(expr)

Jei expr yra **NULL**, **ISNULL**() duoda 1, priešingu atveju – 0.

```
mysql> select ISNULL(1+1);
        -> 0
mysql> select ISNULL(1/0);
        -> 1
```

COALESCE(list)

Duoda pirmąjį ne-**NULL** elementą iš sąrašo.

```
mysql> select COALESCE(NULL,1);
        -> 1
mysql> select COALESCE(NULL,NULL,NULL);
        -> NULL
```

INTERVAL(N,N1,N2,N3,...)

Duoda 0, jei N < N1, ir 1, jei N < N2 ir t.t.. Visi argumentai laikomi sveikaisiais. Reikia, kad N1 < N2 < N3 < ... < Nn.

```
mysql> select INTERVAL(23, 1, 15, 17, 30, 44, 200);
        -> 3
mysql> select INTERVAL(10, 1, 10, 100, 1000);
        -> 2
mysql> select INTERVAL(22, 23, 30, 44, 200);
        -> 0
```

Į 4 skyrelio pradžią

Į pradžią

4.6 Eilučių palyginimo funkcijos

Paprastai, jei kuri nors eilutė (iš palyginamų) registrui, palyginimas atliekamas “jautriai”.

expr LIKE pat [ESCAPE 'escape-char']

Duoda 1 (**TRUE**) arba 0 (**FALSE**). **LIKE** galima naudoti du universalius simbolius:

% aititinka bet kiek (ir 0) simbolių, o _ aititinka tik vieną simbolį.

```
mysql> select 'David!' LIKE 'David_';
        -> 1
mysql> select 'David!' LIKE '%D%v%';
        -> 1
```

Jei reikia lyginti su tuo universaliu simboliu, prieš jį pridėkite \:

\% aititinka vieną % simbolį, _ aititinka vieną _ simbolį

```
mysql> select 'David!' LIKE 'David\_';
        -> 0
mysql> select 'David_' LIKE 'David\_';
        -> 1
```

LIKE galima naudoti skaitmeninėse išraiškose!

```
mysql> select 10 LIKE '1%';
        -> 1
```

expr NOT LIKE pat [ESCAPE 'escape-char']

Atitinka NOT (expr LIKE pat [ESCAPE 'escape-char']).

expr REGEXP pat
expr RLIKE pat

Eilutėje **expr** ieško fragmento **pat**. Duoda 1, jei **expr** turi **pat**, priešingu atveju – 0. **RLIKE** yra **REGEXP** sinonimas (dėl mSQL).

```
mysql> select 'Monty!' REGEXP 'm%y%%';
        -> 0
mysql> select 'Monty!' REGEXP '.*';
        -> 1
mysql> select 'new*\n*line' REGEXP 'new\\*.\\*line';
        -> 1
mysql> select "a" REGEXP "A", "a" REGEXP BINARY "A";
        -> 1  0
```

expr NOT REGEXP pat
expr NOT RLIKE pat

Atitinka **NOT** (**expr REGEXP pat**).

STRCMP(expr1,expr2)

STRCMP() duoda 0, jei eilutės sutampa, -1, jei pirmasis argumentas mažesnis už antrąjį, kitais atvejais duoda 1.

```
mysql> select STRCMP('text', 'text2');
        -> -1
mysql> select STRCMP('text2', 'text');
        -> 1
mysql> select STRCMP('text', 'text');
        -> 0
```

Į 4 skyrelio pradžią

Į pradžią

4.7 Pakeitimo operatoriai

BINARY

BINARY operatorius pakeičia nurodytą eilutę dvejetaine. Tai praverčia pereiti prie jautraus registrui palyginimo, nors stulpelis aprašytas kaip nejautrus (ne **BINARY** ar **BLOB**).

```
mysql> select "a" = "A";
        -> 1
mysql> select BINARY "a" = "A";
        -> 0
```

BINARY įvestas MySQL 3.23.0

Į 4 skyrelio pradžią

Į pradžią

4.8 Valdymo funkcijos

IFNULL(expr1,expr2)

Jei **expr1** yra ne **NULL**, **IFNULL**() duoda expr1, priešingu atveju – **expr2**. **IFNULL**() duoda skaitmeninę arba eilutės reikšmę (priklausomai nuo konteksto).

```
mysql> select IFNULL(1,0);
        -> 1
mysql> select IFNULL(0,10);
        -> 0
mysql> select IFNULL(1/0,10);
        -> 10
mysql> select IFNULL(1/0,'yes');
        -> 'yes'
```

IF(expr1,expr2,expr3)

Jei **expr1** yra **TRUE** (**expr1** <> 0 ir **expr1** <> **NULL**), tai **IF**() duoda **expr2**, priešingu atveju – **expr3**. **IF**() duoda skaitmeninę arba eilutės reikšmę (priklausomai nuo konteksto).

```
mysql> select IF(1>2,2,3);
        -> 3
mysql> select IF(1<2,'yes','no');
        -> 'yes'
mysql> select IF(strcmp('test','test1'),'yes','no');
        -> 'no'
```

expr1 paskaičiuojama kaip skaitinė reikšmė. Tai reikia įvertinti, tikrinant slankaus kablelio skaičius ar eilutes.

```
mysql> select IF(0.1,1,0);
        -> 0
mysql> select IF(0.1<>0,1,0);
        -> 1
```

Pirmame pavyzdyje **IF**(0.1) duoda 0, nes 0.1 konvertuojama į sveikąjį. Antrame pavyzdyje tikrinama slankaus kablelio skaičiaus reikšmė. Palyginimo rezultatas – sveikasis skaičius.

```
CASE value WHEN [compare-value] THEN result [WHEN [compare-value] THEN
result ...] [ELSE result] END
CASE WHEN [condition] THEN result [WHEN [condition] THEN result ...] [ELSE
result] END
```

Pirmame variante gaunamas rezultatas, kai **value**=**compare-value**. Antrame variante gaunamas rezultatas, atitinkantis patenkintą sąlygą. Jei nė viena netenkinama, gaunamas po **ELSE** nurodytas rezultatas is returned. Jei nėra **ELSE** dalies, tai **NULL**.

```
mysql> SELECT CASE 1 WHEN 1 THEN "one" WHEN 2 THEN "two" ELSE "more" END;
       -> "one"
mysql> SELECT CASE WHEN 1>0 THEN "true" ELSE "false" END;
       -> "true"
mysql> SELECT CASE BINARY "B" when "a" then 1 when "b" then 2 END;
       -> NULL
```

Į 4 skyrelio pradžią

Į pradžią

4.9 Matematinės funkcijos

Klaidos atveju visos matematinės funkcijos duoda **NULL**.

- unarinis minusas.

```
mysql> select - 2;
        -> -2
```

ABS(X) duoda X absoliutinę reikšmę.

```
mysql> select ABS(2);
        -> 2
mysql> select ABS(-32);
        -> 32
```

SIGN(X) duoda ženklą: -1, (neig.) 0 ar 1 (teig.)

```
mysql> select SIGN(-32);
        -> -1
mysql> select SIGN(0);
        -> 0
mysql> select SIGN(234);
        -> 1
```

MOD(N,M) arba **%** modulis

```
mysql> select MOD(234, 10);
          -> 4
mysql> select 253 % 7;
          -> 1
mysql> select MOD(29,9);
          -> 2
```

FLOOR(X) maks sveikasis ≤ X.

```
mysql> select FLOOR(1.23);
        -> 1
mysql> select FLOOR(-1.23);
        -> -2
```

CEILING(X) min sveikasis ≥ X.

```
mysql> select CEILING(1.23);
-> 2
mysql> select CEILING(-1.23);
-> -1
```

ROUND(X) suapvalintas X

```
mysql> select ROUND(-1.23);
```

ROUND(X,D) D - ženklų po kablelio

```
mysql> select ROUND(1.298, 1);
```

```
        -> -1                                   -> 1.3
mysql> select ROUND(-1.58);             mysql> select ROUND(1.298, 0);
        -> -2                                   -> 1
mysql> select ROUND(1.58);
        -> 2
```

EXP(X)

```
mysql> select EXP(2);
        -> 7.389056
mysql> select EXP(-2);
        -> 0.135335
```

LOG(X)

```
mysql> select LOG(2);
-> 0.693147
mysql> select LOG(-2);
-> NULL
```

LOG10(X)

```
mysql> select LOG10(2);
        -> 0.301030
mysql> select LOG10(100);
        -> 2.000000
mysql> select LOG10(-100);
        -> NULL
```

POW(X,Y) POWER(X,Y) X^Y

```
mysql> select POW(2,2);
-> 4.000000
mysql> select POW(2,-2);
-> 0.250000
```

SQRT(X)

```
mysql> select SQRT(4);
        -> 2.000000
mysql> select SQRT(20);
        -> 4.472136
```

PI()

```
mysql> select PI();
-> 3.141593
```

COS(X) kosinusas

```
mysql> select COS(PI());
        -> -1.000000
```

SIN(X) sinusas

```
mysql> select SIN(PI());
        -> 0.000000
```

TAN(X) tangentas

```
mysql> select TAN(PI()+1);
        -> 1.557408
```

ACOS(X) arkkosinusas

Jei X už intervalo -1 to 1 – **NULL**.

```
mysql> select ACOS(1);
        -> 0.000000
mysql> select ACOS(1.0001);
        -> NULL
mysql> select ACOS(0);
        -> 1.570796
```

ASIN(X) arksinusas

Jei X už intervalo -1 to 1 – **NULL**.

```
mysql> select ASIN(0.2);
-> 0.201358
mysql> select ASIN('foo');
-> 0.000000
```

ATAN(X) arktangentas

```
mysql> select ATAN(2);
        -> 1.107149
mysql> select ATAN(-2);
        -> -1.107149
```

ATAN2(X,Y) arktangentas Y/X

```
mysql> select ATAN2(-2,2);
-> -0.785398
mysql> select ATAN2(PI(),0);
-> 1.570796
```

COT(X) arkkotangentas

```
mysql> select COT(12);
        -> -1.57267341
```

```
mysql> select COT(0);
        -> NULL
```

RAND() **RAND(N)** atsitiktinis akaičius intervale 0 -1.0.

Jei duotas argumentas N, jis naudojamas kaip seed value.

```
mysql> select RAND();
        -> 0.5925
mysql> select RAND(20);
        -> 0.1811
mysql> select RAND(20);
        -> 0.1811
mysql> select RAND();
        -> 0.2079
mysql> select RAND();
        -> 0.7888
```

LEAST(X,Y,...) Mažiausias argumentas.

```
mysql> select LEAST(2,0);
        -> 0
mysql> select LEAST(34.0,3.0,5.0,767.0);
        -> 3.0
mysql> select LEAST("B","A","C");
        -> "A"
```

GREATEST(X,Y,...) Didžiausias argumentas.

```
mysql> select GREATEST(2,0);
        -> 2
mysql> select GREATEST(34.0,3.0,5.0,767.0);
        -> 767.0
mysql> select GREATEST("B","A","C");
        -> "C"
```

DEGREES(X) duoda X laipsniais.

```
mysql> select DEGREES(PI());
        -> 180.000000
```

RADIANS(X) duoda X radianais

```
mysql> select RADIANS(90);
        -> 1.570796
```

TRUNCATE(X,D) nukerpa X trupmeninę dalį iki D skaitmenų

```
mysql> select TRUNCATE(1.223,1);
        -> 1.2
mysql> select TRUNCATE(1.999,1);
        -> 1.9
mysql> select TRUNCATE(1.999,0);
        -> 1
```

Į 4 skyrelio pradžią

Į pradžią

4.10 Veiksmai su eilutėmis

Šios funkcijos duoda **NULL**, jei rezultato ilgis viršija **max_allowed_packet** serverio parametrą. Pozicijos numeruojamos nuo 1.

ASCII(str)

Duoda kairiausio simbolio ASCII kodą. Duoda NULL, jei eilutė lygi NULL.

```
mysql> select ASCII('2');
        -> 50
mysql> select ASCII(2);
        -> 50
mysql> select ASCII('dx');
        -> 100
```

BIN(N) OCT(N) HEX(N)

Duoda N dvejetainę, aštuntainę ar šešioliktainę reikšmę.

```
mysql> select BIN(12);
        -> '1100'
mysql> select OCT(12);
        -> '14'
mysql> select HEX(255);
        -> 'FF'
```

CHAR(N,...)

CHAR() argumentus interpretuoja kaip sveikuosius ir duoda eilutę.

```
mysql> select CHAR(77,121,83,81,'76');
        -> 'MySQL'
mysql> select CHAR(77,77.3,'77.3');
        -> 'MMM'
```

CONCAT(str1,str2,...)

Sukabina eilutes.

```
mysql> select CONCAT('My', 'S', 'QL');
        -> 'MySQL'
mysql> select CONCAT('My', NULL, 'QL');
        -> NULL
mysql> select CONCAT(14.3);
        -> '14.3'
```

LENGTH(str)
OCTET_LENGTH(str)
CHAR_LENGTH(str)
CHARACTER_LENGTH(str)

Duoda eilutės **str** ilgį.

```
mysql> select LENGTH('text');
        -> 4
mysql> select OCTET_LENGTH('text');
        -> 4
```

LOCATE(substr,str)
POSITION(substr IN str)

Duoda eilutės **str** poziciją, kurioje rasta pirmoji **substr**. Jei **substr** neįeina į **str**, duoda 0.

```
mysql> select LOCATE('bar', 'foobarbar');
        -> 4
mysql> select LOCATE('xbar', 'foobar');
        -> 0
```

LOCATE(substr,str,pos)

Duoda eilutės **str** poziciją, kurioje rasta pirmoji **substr**, pradedant **pos**.

```
mysql> select LOCATE('bar', 'foobarbar',5);
        -> 7
```

LPAD(str,len,padstr)

Duoda eilutę **str**, papildytą iš kairės **padstr**, kol **str** bus ilgio **len**.

```
mysql> select LPAD('hi',4,'??');
        -> '??hi'
```

RPAD(str,len,padstr)

Duoda eilutę **str**, papildytą iš dešinės **padstr**, kol **str** bus ilgio **len**.

```
mysql> select RPAD('hi',5,'?');
        -> 'hi???'
```

LEFT(str,len)

Duoda eilutę, sudarytą iš **len** pirmųjų simbolių.

```
mysql> select LEFT('foobarbar', 5);
        -> 'fooba'
```

RIGHT(str,len)

Duoda eilutę, sudarytą iš **len** paskutinių simbolių.

```
mysql> select RIGHT('foobarbar', 4);
        -> 'rbar'
```

SUBSTRING(str,pos,len)
SUBSTRING(str FROM pos FOR len)
MID(str,pos,len)

Duoda eilutės fragmentą, sudarytą iš **len** simbolių, pradedant pozicija **pos**.

```
mysql> select SUBSTRING('Quadratically',5,6);
        -> 'ratica'
```

SUBSTRING(str,pos)
SUBSTRING(str FROM pos)

Duoda eilutės fragmentą, sudarytą pradedant pozicija **pos**.

```
mysql> select SUBSTRING('Quadratically',5);
        -> 'ratically'
mysql> select SUBSTRING('foobarbar' FROM 4);
        -> 'barbar'
```

LTRIM(str)

Duoda eilutę, gaunamą atmetus tarpus priekyje.

```
mysql> select LTRIM('  barbar');
        -> 'barbar'
```

RTRIM(str)

Duoda eilutę, gaunamą atmetus tarpus gale.

```
mysql> select RTRIM('barbar   ');
        -> 'barbar'
```

TRIM([[BOTH | LEADING | TRAILING] [remstr] FROM] str)

Duoda eilutę, gaunamą atmetus nurodytus simbolius.

```
mysql> select TRIM('  bar   ');
        -> 'bar'
mysql> select TRIM(LEADING 'x' FROM 'xxxbarxxx');
        -> 'barxxx'
mysql> select TRIM(BOTH 'x' FROM 'xxxbarxxx');
        -> 'bar'
mysql> select TRIM(TRAILING 'xyz' FROM 'barxxyz');
        -> 'barx'
```

SPACE(N)

Duoda eilutę, sudarytą iš N tarpų.

```
mysql> select SPACE(6);
        -> '      '
```

REPLACE(str,from_str,to_str)

Eilutėje **str** pakeičia kiekvieną fragmentą **from_str** fragmentu **to_str**.

```
mysql> select REPLACE('www.mysql.com', 'w', 'Ww');
        -> 'WwWwWw.mysql.com'
```

REPEAT(str,count)

Pakartoja **str** **count** kartų. Jei **count** <= 0, gauname tuščią eilutę. Jei **count** =**NULL** , gauname **NULL**.

```
mysql> select REPEAT('MySQL', 3);
        -> 'MySQLMySQLMySQL'
```

REVERSE(str)

Reversas:

```
mysql> select REVERSE('abc');
        -> 'cba'
```

INSERT(str,pos,len,newstr)

Eilutėje **str** nuo pozicijos **pos** vietoj **len** simbolių įterpia fragmentą **newstr**.

```
mysql> select INSERT('Quadratic', 3, 4, 'What');
        -> 'QuWhattic'
```

ELT(N,str1,str2,str3,...)

Duoda N-ją eilutę.

```
mysql> select ELT(1, 'ej', 'Heja', 'hej', 'foo');
        -> 'ej'
mysql> select ELT(4, 'ej', 'Heja', 'hej', 'foo');
        -> 'foo'
```

FIELD(str,str1,str2,str3,...)

Duoda numerį eilutės sąraše **str1, str2, str3**, ... Jei nėra, – duoda 0.

```
mysql> select FIELD('ej', 'Hej', 'ej', 'Heja', 'hej', 'foo');
        -> 2
mysql> select FIELD('fo', 'Hej', 'ej', 'Heja', 'hej', 'foo');
        -> 0
```

FIND_IN_SET(str,strlist)

Duoda 1, jei eilutė yra sąraše. Jei nėra, – duoda 0. Jei kuris nors argumentas yra **NULL**, – duoda **NULL**.

```
mysql> SELECT FIND_IN_SET('b','a,b,c,d');
        -> 2
```

LCASE(str)
LOWER(str)

Visos raidės pakeičiamos mažosiomis.

```
mysql> select LCASE('QUADRATICALLY');
        -> 'quadratically'
```

UCASE(str)
UPPER(str)

Visos raidės pakeičiamos didžiosiomis

```
mysql> select UCASE('Hej');
        -> 'HEJ'
```

LOAD_FILE(file_name)

Skaito serveryje esantį failą ir pateikia kaip eilutę.

```
mysql> UPDATE table_name
           SET blob_column=LOAD_FILE("/tmp/picture")
           WHERE id=1;
```

Kur reikia, MySQL automatiškai pakeičia skaičius eilutėmis ir atvirkščiai:

```
mysql> SELECT 1+"1";
        -> 2
mysql> SELECT CONCAT(2,' test');
        -> '2 test'
```

Į 4 skyrelio pradžią

Į pradžią

4.11 Datos ir laiko funkcijos

Žemiau pateiktas operatorius atrenka visus įrašus, padarytus per paskutines 30 dienų:

```
mysql> SELECT something FROM table
           WHERE TO_DAYS(NOW()) - TO_DAYS(date_col) <= 30;
```

DAYOFWEEK(date)

Pagal datą nurodo savaitės dieną (1 = Sunday, 2 = Monday, ... 7 = Saturday).

```
mysql> select DAYOFWEEK('1998-02-03');
        -> 3
```

WEEKDAY(date)

Pagal datą nurodo savaitės dieną (0 = Monday, 1 = Tuesday, ... 6 = Sunday).

```
mysql> select WEEKDAY('1997-10-04 22:23:00');
        -> 5
mysql> select WEEKDAY('1997-11-05');
        -> 2
```

DAYOFMONTH(date)

Pagal datą nurodo mėnesio dieną.

```
mysql> select DAYOFMONTH('1998-02-03');
        -> 3
```

DAYOFYEAR(date)

Pagal datą nurodo metų dieną (1 iki 366).

```
mysql> select DAYOFYEAR('1998-02-03');
        -> 34
```

MONTH(date)

Pagal datą nurodo mėnesio numerį (1 iki 12).

```
mysql> select MONTH('1998-02-03');
        -> 2
```

DAYNAME(date)

Pagal datą nurodo savaitės dieną.

```
mysql> select DAYNAME("1998-02-05");
        -> 'Thursday'
```

MONTHNAME(date)

Pagal datą nurodo mėnesį.

```
mysql> select MONTHNAME("1998-02-05");
        -> 'February'
```

QUARTER(date)

Pagal datą nurodo ketvirtį (1 iki 4).

```
mysql> select QUARTER('98-04-01');
        -> 2
```

WEEK(date)
WEEK(date,first)

Pagal datą nurodo savaitę (0 iki 53) laikant, kad sekmadienis (Sunday) yra pirmoji savaitės diena. Antrasis **WEEK**() argumentas **first** nurodo, ar savaitė pradedama sekmadieniu (Sunday - 0) ar pirmadieniu (Monday - 1).

```
mysql> select WEEK('1998-02-20');
        -> 7
mysql> select WEEK('1998-02-20',0);
        -> 7
mysql> select WEEK('1998-02-20',1);
        -> 8
mysql> select WEEK('1998-12-31',1);
        -> 53
```

YEAR(date)

Pagal datą nurodo metus (1000 iki 9999).

```
mysql> select YEAR('98-02-03');
        -> 1998
```

YEARWEEK(date)
YEARWEEK(date,first)

Pagal datą nurodo metus ir savaitę. Antrasis **WEEK**() argumentas **first** nurodo, ar savaitė pradedama sekmadieniu (Sunday - 0) ar pirmadieniu (Monday - 1).

```
mysql> select YEARWEEK('1987-01-01');
        -> 198653
```

HOUR(time) MINUTE(time)

Pagal laiką nurodo valandą (0 iki 23), minutę (0-59).

```
mysql> select HOUR('10:05:03');
        -> 10
mysql> select MINUTE('98-02-03 10:05:03');
        -> 5
```

PERIOD_ADD(P,N)

Prideda N mėnesių prie periodo P (formate YYMM ar YYYYMM). Reikšmę pateikia formatu YYYYMM.

```
mysql> select PERIOD_ADD(9801,2);
        -> 199803
```

PERIOD_DIFF(P1,P2)

Pateikia mėnesiais išreikštą skirtumą tarp P1 ir P2. P1 ir P2 formatas YYMM ar YYYYMM.

```
mysql> select PERIOD_DIFF(9802,199703);
        -> 11
```

DATE_ADD(date,INTERVAL expr type)
DATE_SUB(date,INTERVAL expr type)
ADDDATE(date,INTERVAL expr type)
SUBDATE(date,INTERVAL expr type)

Atlieka aritmetines funkcijas su datomis. Tai įvesta MySQL 3.22. ADDDATE() ir SUBDATE() yra DATE_ADD() ir DATE_SUB() sinonimai. MySQL 3.23 vietoj DATE_ADD() ir DATE_SUB() galima naudoti + ir - .

```
mysql> SELECT "1997-12-31 23:59:59" + INTERVAL 1 SECOND;
        -> 1998-01-01 00:00:00
mysql> SELECT INTERVAL 1 DAY + "1997-12-31";
        -> 1998-01-01
mysql> SELECT "1998-01-01" - INTERVAL 1 SECOND;
       -> 1997-12-31 23:59:59
mysql> SELECT DATE_ADD("1997-12-31 23:59:59",
                       INTERVAL 1 SECOND);
        -> 1998-01-01 00:00:00
mysql> SELECT DATE_ADD("1997-12-31 23:59:59",
                       INTERVAL 1 DAY);
        -> 1998-01-01 23:59:59
mysql> SELECT DATE_ADD("1997-12-31 23:59:59",
                       INTERVAL "1:1" MINUTE_SECOND);
        -> 1998-01-01 00:01:00
```

```
mysql> SELECT DATE_SUB("1998-01-01 00:00:00",
                       INTERVAL "1 1:1:1" DAY_SECOND);
        -> 1997-12-30 22:58:59
mysql> SELECT DATE_ADD("1998-01-01 00:00:00",
                       INTERVAL "-1 10" DAY_HOUR);
        -> 1997-12-30 14:00:00
mysql> SELECT DATE_SUB("1998-01-02", INTERVAL 31 DAY);
        -> 1997-12-02
mysql> SELECT EXTRACT(YEAR FROM "1999-07-02");
       -> 1999
mysql> SELECT EXTRACT(YEAR_MONTH FROM "1999-07-02 01:02:03");
       -> 199907
mysql> SELECT EXTRACT(DAY_MINUTE FROM "1999-07-02 01:02:03");
       -> 20102
```

TO_DAYS(date)

Pagal duotą datą pateikia dienų skaičių (nuo 0 metų).

```
mysql> select TO_DAYS(950501);
        -> 728779
mysql> select TO_DAYS('1997-10-07');
        -> 729669
```

FROM_DAYS(N)

Pagal duotą dienų skaičių pateikia datą .

```
mysql> select FROM_DAYS(729669);
        -> '1997-10-07'
```

CURDATE()
CURRENT_DATE

Pateikia šios dienos datą 'YYYY-MM-DD' ar YYYYMMDD formatu, priklausomai nuo naudojamo konteksto.

```
mysql> select CURDATE();
        -> '1997-12-15'
mysql> select CURDATE() + 0;
        -> 19971215
```

CURTIME()
CURRENT_TIME

Pateikia to momento laiką 'HH:MM:SS' ar HHMMSS formatu, priklausomai nuo naudojamo konteksto.

```
mysql> select CURTIME();
        -> '23:50:26'
mysql> select CURTIME() + 0;
        -> 235026
```

NOW()
SYSDATE()
CURRENT_TIMESTAMP

Pateikia to momento datą ir laiką 'YYYY-MM-DD HH:MM:SS' ar YYYYMMDDHHMMSS formatu, priklausomai nuo naudojamo konteksto.

```
mysql> select NOW();
        -> '1997-12-15 23:50:26'
```

```
mysql> select NOW() + 0;
        -> 19971215235026
```

UNIX_TIMESTAMP()
UNIX_TIMESTAMP(date)

Be argumento pateikia to momento Unix laikmatį (sekundes nuo '1970-01-01 00:00:00' GMT). Argumentas **date** nurodo momentą date gali būti **DATE** eilutė, **DATETIME** eilutė, **TIMESTAMP** arba formato YYMMDD ar YYYYMMDD skaičius.

```
mysql> select UNIX_TIMESTAMP();
        -> 882226357
mysql> select UNIX_TIMESTAMP('1997-10-04 22:23:00');
        -> 875996580
```

FROM_UNIXTIME(unix_timestamp)

Pateikia unix_timestamp argumenta 'YYYY-MM-DD HH:MM:SS' ar YYYYMMDDHHMMSS formatu, priklausomai nuo naudojamo konteksto.

```
mysql> select FROM_UNIXTIME(875996580);
        -> '1997-10-04 22:23:00'
mysql> select FROM_UNIXTIME(875996580) + 0;
        -> 19971004222300
```

SEC_TO_TIME(seconds)

Sekundes konvertuoja į valandas, monutes ir sekundes, 'HH:MM:SS' ar HHMMSS formatu, priklausomai nuo naudojamo konteksto.

```
mysql> select SEC_TO_TIME(2378);
        -> '00:39:38'
mysql> select SEC_TO_TIME(2378) + 0;
        -> 3938
```

TIME_TO_SEC(time)

Laiką konvertuoja į sekundes.

```
mysql> select TIME_TO_SEC('22:23:00');
        -> 80580
mysql> select TIME_TO_SEC('00:39:38');
        -> 2378
```

Į 4 skyrelio pradžią

Į pradžią

4.12 Kitos funkcijos

DATABASE()

Nurodo aktyvios DB pavadinimą.

```
mysql> select DATABASE();
        -> 'test'
```

USER()
SYSTEM_USER()
SESSION_USER()

Nurodo MySQL vartotojo vardą.

```
mysql> select USER();
```

```
        -> 'davida@localhost'
```

MySQL 3.22.11 ir vėlesnėse versijose - ir kliento hosto vardą:

```
mysql> select substring_index(USER(),"@",1);
        -> 'davida'
```

PASSWORD(str)

Užkoduoja duotą slaptažodį.

```
mysql> select PASSWORD('badpwd');
        -> '7f84554057dd964b'
```

ENCRYPT(str[,salt])

Užkoduoja duotą eilutę **str** naudodama Unix funkciją crypt(). Argumentas **salt** - dviejų simbolių eilutė. (MySQL 3.22.16 g.b. ilgesnė)

```
mysql> select ENCRYPT("hello");
        -> 'VxuFAJXVARROc'
```

ENCODE(str,pass_str)
DECODE(crypt_str,pass_str)

Užkoduoja duotą eilutę **str** naudodama **pass_str** kaip slaptažodį. Atkoduoja DECODE().

LAST_INSERT_ID([expr])

Pateikia paskutinę automatiškai sugeneruotą reikšmę, kuri buvo įterpta AUTO_INCREMENT stulpelyje.

```
mysql> select LAST_INSERT_ID();
        -> 195
```

VERSION()

Pateikia MySQL serverio versijos numerį.

```
mysql> select VERSION();
        -> '3.22.19b-log'
```

Į 4 skyrelio pradžią

Į pradžią

4.13 Funkcijos GROUP BY operatoriams

Jei naudosite grupavimo funkciją operatoriuje, kuriame nėra GROUP BY sakinio, tai atitiks grupavimą visoms eilutėms.

COUNT(expr)

Pateikia ne-NULL reikšmių skaičių eilutėse, kurias nurodo SELECT operatorius.

```
mysql> select student.student_name,COUNT(*)
           from student,course
           where student.student_id=course.student_id
           GROUP BY student_name;
```

COUNT(DISTINCT expr,[expr...])

Pateikia skirtingų reikšmių skaičių.

```
mysql> select COUNT(DISTINCT results) from student;
```

AVG(expr)

Pateikia vidutinę **expr** reikšmę

```
mysql> select student_name, AVG(test_score)
           from student
           GROUP BY student_name;
```

MIN(expr) MAX(expr)

Pateikia minimalią ar maximalią **expr** reikšmę. MIN() ir MAX() kaip argumentą gali turėti eilutę; tuomet pateikiama eilutės reikšmė.

```
mysql> select student_name, MIN(test_score), MAX(test_score)
           from student
           GROUP BY student_name;
```

SUM(expr)

Pateikia **expr** reikšmių sumą.

STD(expr) STDDEV(expr)

Pateikia standartinį **expr** reikšmių atsilenkimą.

MySQL išplėstas **GROUP BY** panaudojimas. Galima naudoti stulpelius ar skaičiavimus **SELECT** išraiškose, kurios nepasirodo **GROUP BY** dalyje. Tai galima naudoti tam, kad padidinti spartą išvengiant nereikalingų elementų rikiavimo ir grupavimo. Pavyzdžiui, nereikia grupuoti **customer.name** tokioje užklausoje:

```
mysql> select order.custid,customer.name,max(payments)
       from order,customer
       where order.custid = customer.custid
       GROUP BY order.custid;
```

Į 4 skyrelio pradžią

Į pradžią

5. CREATE DATABASE sintaksė

CREATE DATABASE db_name

CREATE DATABASE sukuria duomenų bazę nurodytu vardu.

MySQL duomenų bazės realizuojamos kaip katalogai, kuriuose saugomi DB lenteles atitinkantys failai. Tik sukūrus DB lentelių nėra.

Į pradžią

6. DROP DATABASE sintaksė

DROP DATABASE [IF EXISTS] db_name

DROP DATABASE sunaikina visas lenteles ir DB. **DROP DATABASE** pateikia sunaikintų failų skaičių. Paprastai jis lygus (3 x lentelių skaičius), nes kiekvieną lentelę atitinka 3 failai:`.MYD' failas, `.MYI' failas ir `.frm' failas.

Į pradžią

7. CREATE TABLE sintaksė

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
[(create_definition,...)] [table_options] [select_statement]
```

create_definition:

```
  col_name type [NOT NULL | NULL] [DEFAULT default_value]
[AUTO_INCREMENT] [PRIMARY KEY] [reference_definition]
 arba PRIMARY KEY (index_col_name,...)
 arba KEY [index_name] (index_col_name,...)
 arba   INDEX [index_name] (index_col_name,...)
 arba UNIQUE [INDEX] [index_name] (index_col_name,...)
arba   [CONSTRAINT symbol] FOREIGN KEY index_name
(index_col_name,...) [reference_definition]
 arba CHECK (expr)
```

type:

```
   TINYINT[(length)] [UNSIGNED] [ZEROFILL]
 arba SMALLINT[(length)] [UNSIGNED] [ZEROFILL]
 arba MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]
 arba   INT[(length)] [UNSIGNED] [ZEROFILL]
 arba INTEGER[(length)] [UNSIGNED] [ZEROFILL]
 arba BIGINT[(length)] [UNSIGNED] [ZEROFILL]
 arba REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]
 arba DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]
 arba FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]
 arba DECIMAL(length,decimals) [UNSIGNED] [ZEROFILL]
 arba NUMERIC(length,decimals) [UNSIGNED] [ZEROFILL]
 arba CHAR(length) [BINARY]
 arba VARCHAR(length) [BINARY]
 arba DATE
 arba TIME
 arba TIMESTAMP
 arba DATETIME
 arba TINYBLOB
 arba BLOB
 arba MEDIUMBLOB
 arba LONGBLOB
 arba TINYTEXT
 arba TEXT
 arba MEDIUMTEXT
 arba LONGTEXT
 arba ENUM(value1,value2,value3,...)
 arba SET(value1,value2,value3,...)
```

index_col_name:

```
        col_name [(length)]
```

reference_definition:

```
        REFERENCES tbl_name [(index_col_name,...)]
                   [MATCH FULL | MATCH PARTIAL]
                   [ON DELETE reference_option]
                   [ON UPDATE reference_option]
```

reference_option:

```
    RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT
```

table_options:

```
     TYPE = {ISAM | MYISAM | HEAP}
arba AUTO_INCREMENT = #
arba AVG_ROW_LENGTH = #
arba CHECKSUM = {0 | 1}
arba COMMENT = "string"
arba MAX_ROWS = #
arba MIN_ROWS = #
arba PACK_KEYS = {0 | 1}
arba PASSWORD = "string"
arba DELAY_KEY_WRITE = {0 | 1}
arba ROW_FORMAT= { default | dynamic | static | compressed }
```

select_statement:

```
[IGNORE | REPLACE] SELECT ...  (Some legal select statement)
```

CREATE TABLE sukuria lentelę. MySQL 3.22 ar paskesnėse versijose lentelės vardą galima nurodyti taip: **db_name.tbl_name**.

Sukuriant lentelę MySQL 3.23, galima naudoti **TEMPORARY** raktą. Tokia lentelė bus automatiškai pašalinta, nutraukus ryšį su serveriu. Tai reiškia, kad skirtingi prisijungimai gali naudoti tą patį vardą ir nebus konfliktų.

MySQL 3.22 ar paskesnėse versijose galima naudoti raktus **IF NOT EXISTS**, tada nebus fiksuojama klaida, jei tokiu vardu lentelė jau egzistuoja.

Kiekvieną lentelę atitinka 3 failai:

Failas	***Paskirtis***
tbl_name.frm	Lentelės deskriptoriaus failas
tbl_name.MYD	Duomenø failas
tbl_name.MYI	Indekso failas

- Jei nenurodyta **NULL** arba **NOT NULL**, stulpelis laikomas lyg kad būtų nurodyta **NULL**.
- Sveikųjų skaičių stulpelis gali turėti papildomą atributą **AUTO_INCREMENT**. Įterpiant į **AUTO_INCREMENT** stulpelį reikšmę **NULL** (rekomenduojama) arba 0, stulpelyje nustatoma value+1, čia value - maksimali buvusi reikšmė. **AUTO_INCREMENT** seka pradedama 1. Pašalinus eilutę, kurioje yra maksimali **AUTO_INCREMENT** stulpelio reikšmė, ši reikšmė vėl bus panaudota ISAM lentelėje, bet ne MyISAM lentelėje. Pašalinus visas lentelės eilutes komanda **DELETE FROM TABLE** (be **WHERE**), seka bus pradedama iš naujo.

  Pastaba: Lentelėje gali būti tik vienas **AUTO_INCREMENT** stulpelis, ir jis turi būti indeksuotas.
- **TIMESTAMP** stulpeliuose **NULL** reikšmės apdorojamos kitaip nei kituose. **TIMESTAMP** stulpelyje negalima išsaugoti literalo **NULL**; tokiu atveju būtų įrašyta to momento data ir laikas.

- Jei stulpelyje apibrėžta **DEFAULT** reikšmė, MySQL automatiškai ją priskiria. Jei stulpeliui gali būti suteikta reikšmė **NULL**, pagal nutylėjimą ji ir priskiriama. Jei stulpelis aprašytas kaip **NOT NULL**, pagal nutylėjimą priskiriama reikšmė priklauso nuo stulpelio tipo:
  - skaitmeninio tipo stulpelyje, kuris neturi atributo **AUTO_INCREMENT**, pagal nutylėjimą priskiriama reikšmė lygi 0; jei stulpelyje turi atributą **AUTO_INCREMENT**, pagal nutylėjimą priskiriama sekanti iš eilės reikšmė;
  - datos ir laiko tipo stulpelyje, kuris nėra **TIMESTAMP**, pagal nutylėjimą priskiriama reikšmė lygi "0"; pirmajame lentelės **TIMESTAMP** stulpelyje pagal nutylėjimą priskiriamas to momento data ir laikas;
  - eilučių tipo stulpelyje, kuris nėra **ENUM**, pagal nutylėjimą priskiriama tuščia eilutė; **ENUM** stulpelyje pagal nutylėjimą priskiriamas pirmoji numeravimo reikšmė;
- **KEY** yra **INDEX** sinonimas.
- MySQL **UNIQUE** raktas gali turėti tik skirtingas reikšmes.
- **PRIMARY KEY** yra unikalus raktas **KEY** su papildomu apribojimu – visi rakto stulpeliai turi būti apibrėžti kaip **NOT NULL**. Lentelėje toks **PRIMARY KEY** gali būti tik vienas. Jei **PRIMARY KEY** nėra, o programa reikalauja, kad lentelėje jis būtų, MySQL kaip **PRIMARY KEY** pateikia **UNIQUE** raktą, neturintį **NULL** stulpeliø.
- Tik MyISAM tipo lentelės palaiko indeksavimą stulpelių, kurie gali turėti **NULL** reikšmes. Kitais atvejais tokie stulpeliai turi būti **NOT NULL** tipo.
- Naudodami **col_name(length)** sintaksę, galima nurodyti, kad būtų indeksuojama pagal **CHAR** ar **VARCHAR** stulpelio dalį. Taip gaunamas trumpesnis indekso failas.
- Tik MyISAM tipo lentelės palaiko **BLOB** ir **TEXT** stulpelių indeksavimą. Tokiais atvejais būtina apibrėžti indekso ilgį:

  CREATE TABLE test (blob_col BLOB, index(blob_col(10)));
- **BLOB** ir **TEXT** stulpeliuose naudojant **ORDER BY** ar **GROUP BY**, naudojami tik pirmieji **max_sort_length** baitai.
- Kiekviename **NULL** stulpelyje pridedamas papildomas bitas, apvalinimui iki artimiausio baito.
- Maksimalus įrašo ilgis baitais nustatomas taip:

  eilutės ilgis = 1
  + (stulpelių pločių suma)
  + (NULL stulpelių skaičius + 7)/8
  + (kintamo ilgio stulpelių skaičius)
- **table_options** ir **SELECT** opcijos realizuotos tik MySQL 3.23 ir paskesnėse versijose. Lentelės būna tokių tipų:

  ISAM – The original table handler
  MyISAM – The new binary portable table handler
  HEAP – The data for this table is only stored in memory

  Kitos lentelių opcijos naudojamos siekiant optimizuoti lentelės darbą. Daugumoje atvejų jų net nereikia nurodyti. Opcijos naudojamos visų tipų lentelėse, jei nėra kitaip nurodyta.

  - **AUTO_INCREMENT** Sekanti auto_increment reikšmė, kurią norite nustatyti lentelei (MyISAM)
  - **AVG_ROW_LENGTH** Lentelės eilutės ilgio vidurkio aproksimacija. Tai reikia nurodyti tik lentelei su kintamo ilgio įrašais.
  - **CHECKSUM** Nustatykite 1, jei norite, kad MySQL skaičiuotų visų eilučių kontrolines sumas (tai kiek sumažina spartą) (MyISAM)
  - **COMMENT** 60 simboliø ilgio komentaras

- **MAX_ROWS** Maksimalus lentelės eilučių skaičius (planuojamas)
- **MIN_ROWS** Minimalus lentelės eilučių skaičius (planuojamas)
- **PACK_KEYS** Nustatykite 1, jei norite turėti mažesnį indeksą. Tai kiek sulėtina atnaujinimą, bet paspartina skaitymą (MyISAM, ISAM).
- **PASSWORD** Užkoduoti .frm failą su slaptažodžiu. Standartinėje MySQL versijoje nieko neduoda.
- **DELAY_KEY_WRITE** Nustatykite 1, jei norite atidėti raktų lentelės atnaujinimą iki lentelė bus uždaryta (MyISAM).
- **ROW_FORMAT** Nustato, kaip turi būti įrašomos eilutės (ateičiai).

Naudojant MyISAM lentelę, jos dydį MySQL paskaičiuoja kaip sandaugą **max_rows * avg_row_length**. Jei kuris nors iš šių parametrų nenurodytas, lentelės maksimalus dydis bus 4G (ar 2G, jei OS palaiko tik tokį).

- Jei po **CREATE STATEMENT** nurodote **SELECT**, kiekvienam **SELECT** elementui MySQL sukuria naują lauką. Pavyzdžiui:

```
mysql> CREATE TABLE test (a int not null auto_increment,
           primary key (a), key(b))
           TYPE=HEAP SELECT b,c from test2;
```

Bus sukurta **HEAP** lentelė, turinti 3 stulpelius.

7.1 Stulpelio specifikacijos "tylus" pakeitimas

Kai kuriais atvejais MySQL "tyliai" pakeičia stulpelio specifikaciją kita, nei nurodyta **CREATE TABLE** operatoriuje. (Tai gali atsitikti ir su **ALTER TABLE**.)

- **VARCHAR** stulpeliai, kurių ilgis<4, pakeičiami į **CHAR**.
- Jei kuris nors lentelės stulpelis turi kintamą ilgį, laikoma, kad visa eilutė turi kintamą ilgį. Taigi, jei kuris nors stulpelis turi kintamą ilgį (**VARCHAR**, **TEXT** ar **BLOB**), visi **CHAR** stulpeliai, kurių ilgis>3, pakeičiami į **VARCHAR**.
- **TIMESTAMP** rodomų pozicijų skaičius turi būti lyginis – nuo 2 iki 14. Jei nurodysite ilgį lygų 0 ar >14, jis bus pakeistas į 14. 1-13 intervale nurodyti nelyginiai ilgiai pakeičiami artimiausiais lyginiais.
- **TIMESTAMP** stulpelyje negalima įrašyti literalo NULL; įrašant NULL, bus įrašyta to momento data ir laikas. .

Jei norite sužinoti, ar MySQL pakeitė jūsų nurodytą stulpelio tipą, panaudokite operatorių **DESCRIBE tbl_name**.

Į pradžią

8. ALTER TABLE sintaksė

```
ALTER [IGNORE] TABLE tbl_name alter_spec [, alter_spec ...]
```

alter_specification:

```
  ADD [COLUMN] create_definition [FIRST | AFTER column_name ]
arba ADD INDEX [index_name] (index_col_name,...)
arba ADD PRIMARY KEY (index_col_name,...)
arba ADD UNIQUE [index_name] (index_col_name,...)
arba ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}
arba CHANGE [COLUMN] old_col_name create_definition
arba MODIFY [COLUMN] create_definition
arba DROP [COLUMN] col_name
```

arba **DROP PRIMARY KEY**
arba **DROP INDEX index_name**
arba **RENAME [AS] new_tbl_name**
arba **table_options**

ALTER TABLE leidžia pakeisti turimos lentelės tipą. Pavyzdžiui, galite pridėti ar išmesti stulpelius, sukurti ar panaikinti indeksus, pakeisti stulpelio tipą, pakeisti stulpelio ar lentelės pavadinimą.

Kai kuriais atvejais pakeitimai gali būti nepadaryti dėl 7.1 skyrelyje nurodytų priežasčių.

ALTER TABLE vykdoma sukuriant laikiną lentelės kopiją ir ją modifikuojant, o vėliau ja pakeičiant originalą. Pakeitimo metu kiti dirba su originalu, bet jie negali atlikti pakeitimų ar įrašymo.

- Naudojant **ALTER TABLE**, lentelėje turite pasirinkti, įterpti, panaikinti, pakeisti, sukurti ir atmesti privilegijas.
- **IGNORE** – tai MySQL išplėtimas pagal ANSI SQL92. Ji kontroliuoja, kaip vykdoma **ALTER TABLE** , jei naujojoje lentelėje yra unikaliø raktø dublikatai. Jei **IGNORE** nebus, kopija nekuriama. Jei **IGNORE** yra, paliekama tik pirmoji eilutė iš turinčių tą patį raktą (kitos pašalinamos).
- Viename **ALTER TABLE** sakinyje galite panaudoti kelis **ADD**, **ALTER**, **DROP** ir **CHANGE** operatorius.
- **CHANGE col_name**, **DROP col_name** ir **DROP INDEX** yra MySQL išplėtimas pagal ANSI SQL92.
- **MODIFY** yra Oracle išplėtimas sakiniui **ALTER TABLE**.
- Žodį **COLUMN** galima praleisti – čia jis nieko neduoda.
- Jei naudojate **ALTER TABLE tbl_name RENAME AS new_name** be kitų opcijų, MySQL paprasčiausiai pakeičia failų vardus, kurie atitinka lentelės pavadinimą **tbl_name**. Tuomet nereikia kurti laikiną lentelę.
- **create_definition** sakinys naudojama tą pačią **ADD** ir **CHANGE** sintaksę, kaip ir **CREATE TABLE**.
- Stulpelio pavadinimą galima pakeisti **CHANGE old_col_name create_definition** sakiniu. Tam reikia nurodyti senąjį ir naująjį stulpelio pavadinimus ir stulpelio tipą. Pavyzdžiui, norėdami **INTEGER** stulpelio pavadinimą a pakeisti į b, rašykite:

```
mysql> ALTER TABLE t1 CHANGE a b INTEGER;
```

  Jei norite pakeisti tik stulpelio tipą, **CHANGE** sintaksė vis tiek reikalaus nurodyti du pavadinimus. Pavyzdžiui:

```
mysql> ALTER TABLE t1 CHANGE b b BIGINT NOT NULL;
```

  Tačiau MySQL 3.22.16a to nereikalauja:

```
mysql> ALTER TABLE t1 MODIFY b BIGINT NOT NULL;
```

- Jei naudojate **CHANGE** ar **MODIFY** indeksuotam stulpeliui susiaurinti (pvz., **VARCHAR** stulpelis indeksuotas pagal pirmuosius 10 simbolių), negalite jo susiaurinti daugiau, nei reikia indeksui.
- Jei naudodami **CHANGE** ar **MODIFY** keičiate stulpelio tipą, MySQL bando konvertuoti duomenis į naująjį tipą.
- MySQL 3.22 ir paskesnėse versijose galima naudoti **FIRST** ar **ADD ... AFTER col_name**, norint pridėti stulpelį tam tikroje eilutės vietoje. Pagal nutylėjimą jis pridedamas gale.

- **ALTER COLUMN** nurodo naują stulpelio reikšmę pagal nutylėjimą arba ją pakeičia. Panaikinus senąją, jei stulpelis gali būti lygus **NULL**, naujoji reikšmė pagal nutylėjimą bus **NULL**. Jei stulpelis negali būti lygus **NULL**, MySQL priskiria naują reikšmę pagal nutylėjimą.
- **DROP INDEX** panaikina indesavimą.
- Jei stulpeliai panaikinami, jie išmetami iš indeksų, į kuriuos buvo įtraukti. Panaikinus visus indeksą formavusius stulpelius, bus panaikintas ir indeksas.
- **DROP PRIMARY KEY** panaikina pirminį indeksą. Jei tokio nėra, panaikina pirmąjį unikalų indeksą lentelėje.
- **FOREIGN KEY**, **CHECK** ir **REFERENCES** galima praleisti – čia jie nieko neduoda.

Žemiau pateiktas pavyzdys rodo, kaip gali būti panaudotas **ALTER TABLE**. Turime lentelę t1, sukurtą tokiu būdu:

```
mysql> CREATE TABLE t1 (a INTEGER,b CHAR(10));
```

Pakeisime jos pavadinimą iš t1 į t2:

```
mysql> ALTER TABLE t1 RENAME t2;
```

Pakeisime stulpelį iš **INTEGER** į **TINYINT NOT NULL** (nekeisdami pavadinimo), o **b** pakeisime iš **CHAR(10)** į **CHAR(20),** kartu pakeisime pavadinimą iš **b** į **c**:

```
mysql> ALTER TABLE t2 MODIFY a TINYINT NOT NULL, CHANGE b c
CHAR(20);
```

Pridėsime naują **TIMESTAMP** stulpelį **d**:

```
mysql> ALTER TABLE t2 ADD d TIMESTAMP;
```

Pridėsime stulpelio **d** indeksą ir stulpelį **a** padarysime pirminiu raktiniu:

```
mysql> ALTER TABLE t2 ADD INDEX (d), ADD PRIMARY KEY (a);
```

Panaikinsime stulpelį **c**:

```
mysql> ALTER TABLE t2 DROP COLUMN c;
```

Pridėsime naują **AUTO_INCREMENT** sveikųjų skaičių stulpelį **c**:

```
mysql> ALTER TABLE t2 ADD c INT UNSIGNED NOT NULL
AUTO_INCREMENT, ADD INDEX (c);
```

Pastebėsime, kad indeksavome **c**, nes **AUTO_INCREMENT** stulpeliai turi būti indeksuoti, be to, c paskelbėme kaip **NOT NULL**, nes indeksuoti stulpeliai negali būti **NULL**.

Į pradžią

9. OPTIMIZE TABLE sintaksė

```
OPTIMIZE TABLE tbl_name
```

OPTIMZE TABLE reikia naudoti tuomet, kai iš lentelės pašalinate didelę jos dalį arba lentelėje padarote daug kintamo ilgio eilučių pakeitimų (**VARCHAR, BLOB** ar **TEXT** tipo stulpeliuose). **OPTIMIZE TABLE** optimizuoja vietą lentelėje.

OPTIMIZE TABLE padaro originalios lentelės laikinąją kopiją; senoji kopijuojama į naują praleidžianrt nenaudojamas eilutes, paskui originali lentelė pašalinama, o naujoji užima jos vietą. Optimizavimo metu kiti dirba su originalu, bet jie negali atlikti pakeitimø ar įrašymo.

Į pradžią

10. DROP TABLE sintaksė

```
DROP TABLE [IF EXISTS] tbl_name [, tbl_name,...]
```

DROP TABLE pašalina vieną ar kelias lenteles. Visi duomenys ir deskriptorius *išmetami*, yad šią komandą naudokite **atsargiai**!

Į pradžią

11. DELETE TABLE sintaksė

```
DELETE [LOW_PRIORITY] FROM tbl_name
       [WHERE where_definition] [LIMIT rows]
```

DELETE iš lentelės tbl_name pašalina eilutes, kurios tenkina nurodytą sąlygą where_definition, ir nurodo pašalintų įrašų numerius.

Pateikus **DELETE** komandą be **WHERE** dalies, bus pašalintos visos eilutės. MySQL lentelę padaro tuščia, ir tai žymiai greičiau, nei išmestume eilutes po vieną. Tokiu atveju **DELETE** parodo, kad pašalinta 0 eilučių. Kol egzistuoja lentelę aprašantis failas `**tbl_name.frm'**, lentelė gali būti atkurta, nors duomenų ar indekso failai ir sugadinti.

Jei nurodytas raktas **LOW_PRIORITY**, komandos **DELETE** vykdymas atidedamas iki nė vienas klientas neskaitys iš jos.

Siekiant po pašalinimo geriau panaudoti atmintį ir sumažinti failo dydį, panaudokite komandą **OPTIMIZE TABLE** arba myisamchk utilitą. **OPTIMIZE TABLE** yra lengvesnė, bet myisamchk – greitesnė.

DELETE opcija **LIMIT rows** serveriui nurodo maksimalų skaičių eilučių, kurias pašalinus grąžinamas valdymas klientui. Tai daroma norint įsitikinti, kad **DELETE** komanda neužima per daug laiko. **DELETE** komandą galima tuomet vėl pakartoti.

Į pradžią

12. SELECT sintaksė

```
SELECT [STRAIGHT_JOIN] [SQL_SMALL_RESULT] [SQL_BIG_RESULT]
[HIGH_PRIORITY]
    [DISTINCT | DISTINCTROW | ALL]
    select_expression,...
    [INTO {OUTFILE | DUMPFILE} 'file_name' export_options]
    [FROM table_references
       [WHERE where_definition]
       [GROUP BY col_name,...]
       [HAVING where_definition]
       [ORDER BY {unsigned_integer | col_name | formula} [ASC |
DESC] ,...]
       [LIMIT [offset,] rows]
       [PROCEDURE procedure_name] ]
```

SELECT naudojama eilutėms atrinkti iš vienos ar kelių lentelių. **select_expression** nurodo, kurios eilutės turi būti atrinktos. **SELECT** galima naudoti ir informacijai gauti nesikreipiant į lentelę, pavyzdžiui:

```
mysql> SELECT 1 + 1;
         -> 2
```

Visi parametrai turi būti nurodyti tiksliai ta tvarka, kaip jie aukščiau nurodyti. Pavyzdžiui, parametras **HAVING** turi eiti po **GROUP BY** ir prieš **ORDER BY**.

- **SELECT** išraiškai gali būti duotas pseudonimas panaudojant **AS**. Pseudonimas naudojamas kaip išraiškos stulpelio pavadinimas ir šį pseudonimą galima naudoti **ORDER BY** ar **HAVING** dalyse. Pavyzdžiui:

```
mysql> select concat(last_name,', ',first_name) AS full_name
       from mytable ORDER BY full_name;
```

- **FROM table_references** parodo, iš kurių lentelių turi būti atrinktos eilutės.
- Stulpelis gali būti nurodytas taip: **col_name, tbl_name.col_name** ar **db_name.tbl_name.col_name**. **tbl_name** ar **db_name.tbl_name** prefiksą nurodyti nebūtina, jei nėra nevienareikšmiškumo.
- Lentelė gali būti nurodyta naudojant pseudonimą **tbl_name [AS] alias_name**.

```
mysql> select t1.name, t2.salary from employee AS t1, info AS
t2
           where t1.name = t2.name;
mysql> select t1.name, t2.salary from employee t1, info t2
           where t1.name = t2.name;
```

- Išvedimui atrinkti stulpeliai **ORDER BY** ir **GROUP BY** gali būti nurodomi naudojant stulpelių pavadinimus, stulpelių pseudonimus arba stulpelių pozicijas. Šios numeruojamos nuo 1.

```
mysql> select college, region, seed from tournament
           ORDER BY region, seed;
mysql> select college, region AS r, seed AS s from tournament
           ORDER BY r, s;
mysql> select college, region, seed from tournament
           ORDER BY 2, 3;
```

  Jei reikia išrikiuoti atvirkščia tvarka, **ORDER BY** sakinyje prie stulpelio pavadinimo pridėkite raktą **DESC** (descending). Pagal nutylėjimą rikiuojama didėjimo tvarka; ir tai aiškiai galima nurodyti raktu **ASC**.

- **HAVING** dalyje galima nurodyti bet kurį stulpelį ar pseudonimą, paminėtą **select_expression**. Nenaudokite **HAVING** tiems dalykams, kurie turi būti **WHERE** dalyje. Pavyzdžiui, nerašykite taip:

```
mysql> select col_name from tbl_name HAVING col_name > 0;
```

  Vietoj to rašykite:

```
mysql> select col_name from tbl_name WHERE col_name > 0;
```

  MySQL 3.22.5 ar paskesnėse versijose galima rašyti taip:

```
mysql> select user,max(salary) from users
           group by user HAVING max(salary)>10;
```

  Vietoj to ankstesnėse MySQL versijose galima rašyti taip:

```
mysql> select user,max(salary) AS sum from users
           group by user HAVING sum>10;
```

- **SQL_SMALL_RESULT, SQL_BIG_RESULT, STRAIGHT_JOIN** ir **HIGH_PRIORITY** skirti MySQL išplėtimui pagal ANSI SQL92.
- **STRAIGHT_JOIN** nurodo, kad optimizatorius turi apjungti lenteles ta tvarka, kaip jos nurodytos **FROM** sakinyje. Tai gali padidinti spartą.

- **SQL_SMALL_RESULT** galima naudoti **GROUP BY** ar **DISTINCT** dalyse nurodant optimizatoriui, kad gautasis rinkinys bus mažas. Tokiu atveju MySQL vietoj rikiavimo naudos greitas laikinąsias lenteles formuojamai lentelei saugoti.
- **SQL_BIG_RESULT** galima naudoti **GROUP BY** ar **DISTINCT** dalyse nurodant optimizatoriui, kad gautasis rinkinys turės daug eilučių. Tokiu atveju MySQL naudos laikinąsias lenteles diske (jei to prireiks). MySQL šiuo atveju pirmenybę teiks rikiavimui vietoj laikinųjų lentelių naudojimo.
- **HIGH_PRIORITY** suteiks **SELECT** aukštesnį prioritetą nei operatoriui atnaujinančiam lentelę. Tai reikia naudoti tik tose užklausose, kurios yra greitos ir turi būti atliktos iš karto.
- **LIMIT** dalį galima naudoti norint apriboti **SELECT** sakiniu pateikiamų eilučių skaičių. **LIMIT** naudoja vieną ar du skaitinius argumentus. Jei nurodyti du argumentai, pirmasis nurodo pirmosios pateikiamos eilutės "poslinkį" (offset), o antrasis –maksimalų pateikiamų eilučių skaičių. Pradinės eilutės "poslinkis" yra 0 (ne 1).

```
mysql> select * from table LIMIT 5,10;  # Retrieve rows 6-15
```

Jei nė vienas argumentas nėra nurodomas, bus pateiktos visos eilutės.

```
mysql> select * from table LIMIT 5;  # Retrieve first 5 rows
```

- **SELECT ... INTO OUTFILE 'file_name'** atveju atrinktos eilutės surašomos į failą. Tam reikia turėti atitinkamas teises.

Jei vietoj **INTO OUTFILE** naudojate **INTO DUMPFILE,** MySQL į failą įrašys tik vieną eilutę, be stulpelio ar eilutės pabaigos žymių. Tai naudinga įrašant blob.

Į pradžią

13. JOIN sintaksė

MySQL palaiko tokias **JOIN** sintakses **SELECT** sakiniuose**:**

```
table_reference, table_reference
table_reference [CROSS] JOIN table_reference
table_reference INNER JOIN table_reference
table_reference STRAIGHT_JOIN table_reference
table_reference LEFT [OUTER] JOIN table_reference ON
conditional_expr
table_reference LEFT [OUTER] JOIN table_reference USING
(column_list)
table_reference NATURAL LEFT [OUTER] JOIN table_reference
{ oj table_reference LEFT OUTER JOIN table_reference ON
conditional_expr }
```

Paskutinysis **LEFT OUTER JOIN** sintakses variantas skirtas suderinamumui su ODBC.

- Lentelės nuoroda gali būti pakeista pseudonimu **tbl_name AS alias_name** ar **tbl_name alias_name**.

```
mysql> select t1.name, t2.salary from employee AS t1, info AS
t2
            where t1.name = t2.name;
```

- **INNER JOIN** ir , (kablelis) yra semantiškai ekvivalentiški. Abu gali būti naudojami lentelėms apjungti. Paprastai lentelių apjungimas apibrėžiamas **WHERE** sąlyga.
- **ON** sąlyga gali būti nurodyta bet kuria forma, priimta **WHERE** sakinyje.

- Jei **LEFT JOIN** dešinėje nurodytai lentelei nėra įrašo, visuose eilutės stulpeliuose įrašoma **NULL** If there is no matching record for the right table in a LEFT JOIN, a row with all columns set to NULL is used for the right table. You can use this fact to find records in a table that have no counterpart in another table:

```
mysql> select table1.* from table1
           LEFT JOIN table2 ON table1.id=table2.id
           where table2.id is NULL;
```

  Čia bus rastos visos **table1** eilutės su **id** reikšme, kurios nėra lentelėje **table2** (t.y., visos **table1** eilutės kurioms nėra atitinkamos eilutės lentelėje **table2**). Taip daroma prielaida, kad **table2.id** yra paskelbtas **NOT NULL**.

- **USING (column_list)** dalis nurodo sąrašą stulpelių, kurie privalo būti abiejose lentelėse. Toks **USING**:

```
A LEFT JOIN B USING (C1,C2,C3,...)
```

  yra semantiškai identiškas **ON** išraiškai:
  `A.C1=B.C1 AND A.C2=B.C2 AND A.C3=B.C3,...`

- Dviejų lentelių **tables NATURAL LEFT JOIN** padarytas semantiškai ekvivalentišku veiksmui **LEFT JOIN** su **USING** dalimi, nurodančia stulpelius, kurie yra abiejose lentelėse.

- **STRAIGHT_JOIN** identiškas **JOIN**, išskyrus tai, kad kairioji lentelė visada skaitoma anksčiau nei dešinioji.

  Keletas pavyzdžių:

```
mysql> select * from table1,table2 where table1.id=table2.id;
mysql> select * from table1 LEFT JOIN table2 ON
table1.id=table2.id;
mysql> select * from table1 LEFT JOIN table2 USING (id);
mysql> select * from table1 LEFT JOIN table2 ON
table1.id=table2.id
           LEFT JOIN table3 ON table2.id=table3.id;
```

Į pradžią

14. INSERT sintaksė

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
    [INTO] tbl_name [(col_name,...)]
    VALUES (expression,...),(...),...
```

arba **INSERT [LOW_PRIORITY | DELAYED] [IGNORE]**
[INTO] tbl_name [(col_name,...)]
SELECT ...
arba

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
    [INTO] tbl_name
    SET col_name=expression, col_name=expression, ...
```

INSERT įterpia eilutes jau sukurtoje lentelėje:

- **INSERT ... VALUES** įterpia eilutes, sukurtas pagal nurodytas reikšmes.
- **INSERT ... SELECT** įterpia eilutes, parinktas kitoje lentelėje ar lentelėse.
- **INSERT ... VALUES** įterpia eilutes, sukurtas pagal reikšmių sąrašus (MySQL 3.22.5 ar paskesnė versija).

tbl_name nurodo lentelę, į kurią įterpiame. **SET** dalyje nurodomas sąrašas stulpelių, kuriems reikšmės bus suteiktos.

- Jei nenurodytas sąrašas stulpelių **INSERT ... VALUES** ar **INSERT ... SELECT**, reikia nurodyti visų stulpelių reikšmes sąraše **VALUES()** ar **SELECT**.
- Jei stulpelio reikšmė nenurodyta aiškiai, jam priskiriama reikšmė pagal nutylėjimą.
- Išraiškoje galima panaudoti tik jau apibrėžto stulpelio vardą. Galima rašyti:

  ```
  mysql> INSERT INTO tbl_name (col1,col2) VALUES(15,col1*2);
  ```

  Tačiau taip – ne:
  ```
  mysql> INSERT INTO tbl_name (col1,col2) VALUES(col2*2,15);
  ```
- **LOW_PRIORITY** nurodo, kad **INSERT** komandos vykdymas atidedamas iki kiti nebeskaitys iš lentelės.
- Jei **INSERT** komandoje nurodytas raktas **IGNORE,** ignoruojamos tos eilutės, kurios turi tą pačią **PRIMARY** ar **UNIQUE** rakto reikšmę (jos nėra įterpiamos). Jei raktas **IGNORE** nenurodytas**,** minėtu atveju įterpimas nutraukiamas.
- Jei MySQL konfigūruotas naudojant **DONT_USE_DEFAULT_FIELDS** opciją, **INSERT** operatorius generuos klaidą, jei nebus aiškiai užduotos tų stulpelių reikšmės, kurie reikalauja ne-**NULL** reikšmių.
- **INSERT INTO ... SELECT** operatoriui galioja tokios sąlygos:
  - Užklausoje negali būti naudojama **ORDER BY** dalis.
  - **INSERT** operatoriuje nurodyta lentelė statement negali būti nurodyta užklausos **SELECT** dalies **FROM** dalyje, nes tai draudžia ANSI SQL (negalima **SELECT** iš tos pačios lentelės, į kurią įterpiame).
  - **AUTO_INCREMENT** stulpeliai funkcionuoja įprastu būdu.

Jei naudojate **INSERT ... SELECT** arba **INSERT ... VALUES** operatorių su daugeliu reikšmių sąrašais, galima panaudoti C API funkciją **mysql_info()** informacijai apie užklausą gauti. Ji pateikiama taip:

Records: 100 Duplicates: 0 Warnings: 0

DELAYED opcija **INSERT** operatoriuje labai naudinga tuo, jei kurie nors klientai negali laukti, kol pasibaigs **INSERT** operacija. Kai naudojate **INSERT DELAYED**, klientas gaus “ok” iš karto ir eilutė bus įterpta, jei lentelė nenaudijama tuo metu kitoje gijoje.

Į pradžią

15. REPLACE sintaksė

```
    REPLACE [LOW_PRIORITY | DELAYED]
        [INTO] tbl_name [(col_name,...)]
       VALUES (expression,...)
arba  REPLACE [LOW_PRIORITY | DELAYED]
        [INTO] tbl_name [(col_name,...)]
        SELECT ...
arba  REPLACE [LOW_PRIORITY | DELAYED]
        [INTO] tbl_name
        SET col_name=expression, col_name=expression,...
```

REPLACE vykdoma lygiai taip pat, kaip ir **INSERT**, išskyrus tai, kad jei senas lentelės įrašas turėjo tą pačią reikšmę, kaip ir naujasis su unikaliu indeksu, senasis pašalinamas prieš įterpiant naują.

Į pradžią

16. LOAD DATA INFILE sintaksė

```
LOAD DATA [LOW_PRIORITY] [LOCAL] INFILE 'file_name.txt'
[REPLACE | IGNORE]
        INTO TABLE tbl_name
        [FIELDS
            [TERMINATED BY '\t']
            [OPTIONALLY] ENCLOSED BY '']
            [ESCAPED BY '\\' ]]
        [LINES TERMINATED BY '\n']
        [IGNORE number LINES]
        [(col_name,...)]
```

LOAD DATA INFILE operatorius skaito eilutes iš tekstinio failo į lentelę dideliu greičiu.Jei nurodytas raktas **LOCAL**, failas skaitomas iš lokalios kliento vietos, priešingu atveju – iš serverio.

Pavyzdžiui, šis **LOAD DATA** operatorius skaito failą `**data.txt**' iš DB **db1** katalogo ir rašo į DB **db2** lentelę:

```
mysql> USE db1;
mysql> LOAD DATA INFILE "data.txt" INTO TABLE db2.my_table;
```

REPLACE ir **IGNORE** raktai valdo dubliuojančių įrašų tvarkymą.

Žemiau pateiktas pavyzdys, kai įkraunami visi **persondata** lentelės stulpeliai:

```
mysql> LOAD DATA INFILE 'persondata.txt' INTO TABLE persondata;
```

Laukų sąrašo nėra, todėl **LOAD DATA INFILE** laukia, kad bus pateiktos visų stulpelių reikšmės.

Jei norite įkrauti tik kai kuriuos lentelės stulpelius, pateikite jų sąrašą:

```
mysql> LOAD DATA INFILE 'persondata.txt'
           INTO TABLE persondata (col1,col2,...);
```

Jei reikšmių trūksta, jos užpildomos pagal nutylėjimą. Jei eilutėje yra per daug laukų, pertekliniai laukai ignoruojami.

Į pradžią

17. UPDATE sintaksė

```
UPDATE [LOW_PRIORITY] tbl_name SET col_name1=expr1,
col_name2=expr2,...
        [WHERE where_definition] [LIMIT #]
```

UPDATE atnaujina anksčiau sukurtos lentelės stulpelių reikšmes. **SET** dalis nurodo, kurie stulpeliai turi būti pakeisti ir kokias reikšmes jie įgys. **WHERE** dalis (jei ji yra) nurodo, kurios eilutės turi būti keičiamos. Jei **WHERE** dalies nėra - atnaujinamos visos eilutės.

LOW_PRIORITY nurodo, kad operatoriaus **UPDATE** vykdymas atidedamas, kol kiti klientai skaito iš lentelės.

Jei lentelės **tbl_name** stulpelis panaudotas išraiškoje, **UPDATE** naudoja esamą jo reikšmę. Pavyzdžiui, šis operatorius **age** stulpelio reikšmę padidina vienetu:

```
mysql> UPDATE persondata SET age=age+1;
```

UPDATE išraiškos skaičiuojamos iš kairės į dešinę. Pavyzdžiui, šis operatorius padvigubina **age** stulpelio reikšmę, o po to ją dar padidina vienetu:

```
mysql> UPDATE persondata SET age=age*2, age=age+1;
```

UPDATE parodo, kiek eilučių buvo iš tiesų atnaujinta.

Į pradžią

18. USE sintaksė

```
USE db_name
```

USE db_name nurodo MySQL, kad toliau aktyviaja DB bus laikoma **db_name**:

```
mysql> USE db1;
mysql> SELECT count(*) FROM mytable;          # selects from
db1.mytable
mysql> USE db2;
mysql> SELECT count(*) FROM mytable;          # selects from
db2.mytable
```

Žinoma, galima kreiptis ir į kitas DB, pavyzdžiui į **db2** DB:

```
mysql> USE db1;
mysql> SELECT author_name,editor_name FROM author,db2.editor
           WHERE author.editor_id = db2.editor.editor_id;
```

Į pradžią

19. FLUSH sintaksė (kešų valymas)

```
FLUSH flush_option [,flush_option]
```

FLUSH komandą naudokite tuomet, kai norėsite, kad būtų išvalyti vidiniai MySQL kešai. **FLUSH** vykdymui reikia, kad turėtumėte perkrovimo teises (reload privilege).

Į pradžią

20. KILL sintaksė

```
KILL thread_id
```

Kiekvienas prisijungimas prie MySQL vykdomas atskiroje gijoje.Gijas galite pamatyti **SHOW PROCESSLIST** komanda, o konkrečią giją nutraukti - **KILL thread_id** komanda.

Jei turite proceso teises (process privilege), galite matyti ir nutraukti bet kurią giją, priešingu atveju - tik savo.

Į pradžią

21. SHOW sintaksė (gauti informaciją apie lenteles, stulpelius,...)

```
   SHOW DATABASES [LIKE wild]
arba SHOW TABLES [FROM db_name] [LIKE wild]
arba SHOW COLUMNS FROM tbl_name [FROM db_name] [LIKE wild]
arba SHOW INDEX FROM tbl_name [FROM db_name]
arba SHOW STATUS
arba SHOW VARIABLES [LIKE wild]
arba SHOW [FULL] PROCESSLIST
arba SHOW TABLE STATUS [FROM db_name] [LIKE wild]
arba SHOW GRANTS FOR user
```

SHOW pateikia informaciją apie duomenų bazes, lenteles, stulpelius arba serverį. **LIKE wild** dalyje gali būti naudojami simboliai `%' ir `_'.

Galima naudoti **db_name.tbl_name** vietoj **tbl_name FROM db_name**. Šie du operatoriai yra ekvivalentiški:

```
mysql> SHOW INDEX FROM mytable FROM mydb;
mysql> SHOW INDEX FROM mydb.mytable;
```

SHOW STATUS pateikia informaciją apie serverio būseną. Pavyzdys:

```
+--------------------------+--------+
| Variable_name            | Value  |
+--------------------------+--------+
| Aborted_clients          | 0      |
| Aborted_connects         | 0      |
| Connections              | 17     |
| Created_tmp_tables       | 0      |
| Delayed_insert_threads   | 0      |
| Delayed_writes           | 0      |
| Delayed_errors           | 0      |
| Flush_commands           | 2      |
| Handler_delete           | 2      |
| Handler_read_first       | 0      |
| Handler_read_key         | 1      |
| Handler_read_next        | 0      |
| Handler_read_rnd         | 35     |
| Handler_update           | 0      |
| Handler_write            | 2      |
| Key_blocks_used          | 0      |
| Key_read_requests        | 0      |
| Key_reads                | 0      |
| Key_write_requests       | 0      |
| Key_writes               | 0      |
| Max_used_connections     | 1      |
| Not_flushed_key_blocks   | 0      |
| Not_flushed_delayed_rows | 0      |
| Open_tables              | 1      |
| Open_files               | 2      |
| Open_streams             | 0      |
| Opened_tables            | 11     |
| Questions                | 14     |
| Slow_queries             | 0      |
| Threads_connected        | 1      |
| Threads_running          | 1      |
| Uptime                   | 149111 |
+--------------------------+--------+
```

SHOW PROCESSLIST parodo vykdomas gijas.

Į pradžią

22. EXPLAIN sintaksė (duoda informaciją apie SELECT)

```
EXPLAIN tbl_name
```

arba **`EXPLAIN SELECT select_options`**

EXPLAIN tbl_name yra **DESCRIBE tbl_name** ar **SHOW COLUMNS FROM tbl_name** sinonimas.

Jei prieš **SELECT** operatorių įrašote **EXPLAIN**, MySQL paaiškina, kaip bus vykdomas **SELECT**, nurodydama, kaip bus apjungtos (join) lentelės ir kuria tvarka.

Naudodami **EXPLAIN**, galite pastebėti, kada reikia indeksuoti lenteles **SELECT** spatrai padidinti. Taip pat galite pastebėti, ar optimizatorius apjungia lenteles optimalia tvarka. Kad jis tai darytų, **SELECT** operatoriuje pridėkite **STRAIGHT_JOIN** dalį.

EXPLAIN pateikia informaciją aoie jo darbą tokiuose stulpeliuose:

table - vardas lentelės, apie kurią informuoja ši eilutė

type - apjungimo tipas

possible_keys - galimi raktai - kuriuos indeksus MySQL naudoja eilutėms atrinkti

key - raktas, kurį MySQL pasirinko

key_len - rakto, kurį MySQL pasirinko, ilgis

ref - stulpeliai ar konstantos, kurias MySQL pasirinko eilutėms atrinkti

rows - skaičius eilučių, kurias MySQL ketina patikrinti, vykdydamas užklausą

Extra - papildoma informacija

Skirtingi apjungimo tipai, pradedant geriausiais:

system - lentelė turi tik vieną eilutę (= sisteminė lentelė).

const - lentelė turi maksimum vieną tinkančią eilutę, kuri perskaitoma užklausos pradžioje. Jos stulpeliai gali būti optimizatoriaus laikomi konstantomis. Vykdoma labai greitai.

eq_ref - viena eilutė iš šios lentelės bus perskaitoma kiekvienai ankstesnių lentelių eilučių kombinacijai. Tai geriausias iš nekonstantinių apjungimo tipų. Jis naudojamas, kai visos indekso dalys gali būti naudojamos apjungimui, ir indeksas yra **UNIQUE** ar **PRIMARY KEY**.

ref - visos eilutės su sutampančiais indeksais iš šios lentelės bus perskaitoma. Jis naudojamas, kai apjungimas naudoja tik kairiausią rakto prefiksą arba raktas yra ne **UNIQUE** ar **PRIMARY KEY**.

range - bus imamos eilutės tik iš nurodyto diapazono, atrinkimui naudojant indeksą. Nurodo panaudotą indeksą.

index - tas pats kaip ir **ALL**, išskyrus tai, kad skenuojamas tik indeksų medis. Tai dirba sparčiau, nei **ALL**, nes indekso failas yra trumpesnis nei duomenų.

ALL - bus skenuojama visa lentelė kiekvienai ankstesnių lentelių eilučių kombinacijai. Tai paprastai nėra gerai, ir to išvengti galima padidinant indeksų skaičių.

Kiek gerai atliekamas apjungimas, galima sužinoti sudauginus **EXPLAIN** pateiktų visų eilučių **rows** stulpelio reikšmes.

Žemiau pateiktas pavyzdys rodo, kiek gali būti optimizuotas **JOIN** panaudojant informaciją, gautą iš **EXPLAIN**.

Tarkime, turime tokį **SELECT** operatorių, kurį analizuosime su **EXPLAIN**:

```
EXPLAIN SELECT tt.TicketNumber, tt.TimeIn,
            tt.ProjectReference, tt.EstimatedShipDate,
            tt.ActualShipDate, tt.ClientID,
            tt.ServiceCodes, tt.RepetitiveID,
            tt.CurrentProcess, tt.CurrentDPPerson,
            tt.RecordVolume, tt.DPPrinted, et.COUNTRY,
```

```
    et_1.COUNTRY, do.CUSTNAME
FROM tt, et, et AS et_1, do
WHERE tt.SubmitTime IS NULL
    AND tt.ActualPC = et.EMPLOYID
    AND tt.AssignedPC = et_1.EMPLOYID
    AND tt.ClientID = do.CUSTNMBR;
```

- Cia buvo priimta, kad palyginami stulpeliai buvo aprašyti šitaip:

Lentele	Stulpelis	Stulpelio tipas
tt	ActualPC	CHAR(10)
tt	AssignedPC	CHAR(10)
tt	ClientID	CHAR(10)
et	EMPLOYID	CHAR(15)
do	CUSTNMBR	CHAR(15)

- Leenteles turi ¸emiau pateiktus indeksus:

Lentele	Indeksas
tt	ActualPC
tt	AssignedPC
tt	ClientID
et	EMPLOYID (primary key)
do	CUSTNMBR (primary key)

- **tt.ActualPC** reikšmes nera paskirstytos evenly.

Prieš pradedant optimizuoti **EXPLAIN** operatorius pateikia tokia informacija:

```
    table type possible_keys                key   key_len ref   rows
Extra
    et    ALL  PRIMARY                      NULL NULL    NULL 74
    do    ALL  PRIMARY                      NULL NULL    NULL 2135
    et_1  ALL  PRIMARY                      NULL NULL    NULL 74
    tt    ALL  AssignedPC,ClientID,ActualPC NULL NULL    NULL 3872
          range checked for each record (key map: 35)
```

Kadangi kiekviena lentele yra **ALL** tipo, ši informacija rodo, kad MySQL atlieka pilna lenteliu apjungima! Tai u¸truks gana ilgai, nes bus tikrinama 74 * 2135 * 74 * 3872 = 45,268,558,720 eiluciu.

Susiduriame su tokia problema - MySQL (dar) negali efektyviai naudoti stulpeliu indeksu, jei jie apibre¸ti skirtingai. Šiame kontekste **VARCHAR** ir **CHAR** yra vienodi, jei tik apibre¸iant nera nurodytas skirtingas ilgis. Kadangi **tt.ActualPC** apibre¸ta kaip **CHAR(10)**, o **et.EMPLOYID** – kaip **CHAR(15)**, turime ilgiu nesutapima.

Tokiu atveju panaudokite **ALTER TABLE** , kad **ActualPC** ilgis butu padidintas nuo 10 simboliu iki 15:

```
mysql> ALTER TABLE tt MODIFY ActualPC VARCHAR(15);
```

Į pradžią

23. DESCRIBE sintakse (duoda informaciją apie stulpelius)

```
{DESCRIBE | DESC} tbl_name {col_name | wild}
```

DESCRIBE pateikia informacija apie lenteles stulpelius. **col_name** - stulpelio pavadinimas arba eilute su SQL `%' ir `_' simboliais.

Į pradžią

24. LOCK TABLES/UNLOCK TABLES sintakse

```
LOCK TABLES tbl_name [AS alias] {READ | [READ_LOCAL] | [LOW_PRIORITY] WRITE}
        tbl_name {READ | [LOW_PRIORITY] WRITE} ...]
...
UNLOCK TABLES
```

LOCK TABLES "u¸rakina" lenteles šioje gijoje. **UNLOCK TAB**LES – "atrakina". Visos "u¸rakintos" lenteles automatiškai "atrakinamos" u¸rašius kita **LOCK TABLES** arba atsijungus nuo serverio.

Jei nurodyta **READ**, ta gija (ir visos kitos gijos) gali iš lenteles tik skaityti. Jai nurodyta **WRITE**, tik ta gija gali atlikti **READ** ar **WRITE**. Kitos gijos blokuojamos. **READ_LOCAL** skiriasi nuo **READ** tuo, kad **READ_LOCAL** leid¸ia vykdyti nekonfliktinius **INSERT** operatorius.

Į pradžią

25. SET sintakse

```
SET [OPTION] SQL_VALUE_OPTION= value, ...
```

SET OPTION nustato ivairius parametrus, kurie turi itakos serverio arba kliento darbui.

CHARACTER SET character_set_name | DEFAULT

Tai nustato naudojamus šriftus. Kol kas vienintele **character_set_name** reikšme yra cp1251_koi8, bet galima prideti naujas koreguojant **`sql/convert.cc'** fail! **MySQL** distribucijoje.

PASSWORD = PASSWORD('some password')

Tai nustato to vartotojo slapta¸odi. Bet kuris neanonimas gali ji pasikeisti!

PASSWORD FOR user = PASSWORD('some password')

Tai nustato nurodyto vartotojo slapta¸odi.

SQL_AUTO_IS_NULL = 0 | 1

Jei nustatytas 1 (pagal nutylejima), tai kiekvienas gali rasti paskutine iterpta eilute.

SQL_BIG_TABLES = 0 | 1

Jei nustatytas 1, visos laikinosios lenteles greiciau isimenamos diske mei atmintyje.

SQL_BIG_SELECTS = 0 | 1

Jei nustatytas 0, MySQL bus "nuimtas", jei **SELECT** darbas truktu per ilgai.

SQL_LOW_PRIORITY_UPDATES = 0 | 1

Jei nustatytas 1, visi **INSERT, UPDATE, DELETE** ir **LOCK TABLE WRITE** operatoriai lauks iki bus ivykdyti atideti **SELECT** ar **LOCK TABLE READ**.

SQL_SAFE_MODE = 0 | 1

Jei nustatytas 1, MySQL bus "nuimtas", jei **UPDATE** ar **DELETE** nenaudoja rakto ar **LIMIT WHERE** dalyje.

SQL_SELECT_LIMIT = value | DEFAULT

Maksimalus skaičius irašų, kuriuos pateikia **SELECT** operatorius. **SELECT** nurodyta **LIMIT** reikšmė turi viršenybę **SQL_SELECT_LIMIT** atžvilgiu.

TIMESTAMP = timestamp_value | DEFAULT

Nustato **time** reikšmę tam klientui.

INSERT_ID = #

Nustato reikšmę, kuria naudos kitos **INSERT** komandos naudodamos **AUTO_INCREMENT** reikšmę.

Į pradžią

26. GRANT ir REVOKE sintaksė

```
    GRANT  priv_type  [(column_list)]  [,  priv_type  [(column_list)]
...]
         ON {tbl_name | * | *.* | db_name.*}
         TO user_name [IDENTIFIED BY 'password']
             [, user_name [IDENTIFIED BY 'password'] ...]
         [WITH GRANT OPTION]
  REVOKE priv_type [(column_list)] [, priv_type [(column_list)] ...]
         ON {tbl_name | * | *.* | db_name.*}
         FROM user_name [, user_name ...]
```

GRANT įvestas MySQL 3.22.11.

GRANT ir **REVOKE** komandos leidžia sistemos administratoriams suteikti teises MySQL vartotojams ir jas atimti privilrgijų lygiais:

1. Globalinis lygis

Šis privilegijų lygis taikomas visoms tame serveryje esančioms DB. Informacija saugoma lentelėje **mysql.user**.

2. Duomenų bazių lygis

Šis privilegijų lygis taikomas visoms tos DB lentelėms. Informacija saugoma lentelėse **mysql.db** ir **mysql.host**.

3. Lentelės lygis

Šis privilegijų lygis taikomas visiems tos lentelės stulpeliams. Informacija saugoma lentelėje **mysql.tables_priv**.

4. Stulpelio lygis

Šis privilegijų lygis taikomas atiskiriems tos lentelės stulpeliams. Informacija saugoma lentelėje **mysql.columns_priv**.

GRANT ir **REVOKE** operatoriuose **priv_type** gali būti nurodytas vienu iš tokių būdų:

ALL PRIVILEGES	FILE	RELOAD
ALTER	INDEX	SELECT
CREATE	INSERT	SHUTDOWN
DELETE	PROCESS	UPDATE
DROP	REFERENCES	USAGE

Norėdami atimti vartotojui suteiktas teises, naudokite **priv_type** reikšmę **GRANT OPTION**:

REVOKE GRANT OPTION ON ... FROM ...;

Bet kuriuose hostuose esančių vartotojų teisėms valdyti MySQL leidžia nurodyti **user_name** reikšmę tokia forma: **user@host**. Jei vartotojo ar hosto varde naudojami specialūs simboliai (pvz., `-'), vartotojo ar hosto vardus galima įrėminti kabutėmis (pvz., 'test-user'@'test-hostname').

Be to, galima rašyti **user@"%.loc.gov"**, nurodant visus **loc.gov** domeno vartotojus, arba **user@"144.155.166.%"** , nurodant vartotoją visuose hostuose, esančiuose **144.155.166** klasės C potinklyje.

Lentelės ar stulpelio teisės formuojamos logine **OR** operacija apjungiant kiekvieną iš keturių privilegijų lygių. Pavyzdžiui, jei **mysql.user** lentelė apibrėžia, kad vartotojas turi globalią select privilegiją, jos negalima paneigti DB, lentelės ar stulpelio lygiais.

WITH GRANT OPTION dalis vartotojui leidžia suteikti kitiems tas privilegijas, kokias turi jis pats.

Į pradžią

27. CREATE INDEX sintaksė

```
CREATE [UNIQUE] INDEX index_name ON tbl_name (col_name[(length)],... )
```

CREATE INDEX įvestas nuo MySQL versijos 3.22. **CREATE INDEX** skirtas indeksams kurti (kai naudojama **ALTER TABLE**). Paprastai visi lentelės indeksai sukuriami tuo pat metu, kai k`uriama lentelė CREATE TABLE. **CREATE INDEX** prideda indeksus jau sukurtoms lentelėms.

Stulpelių sąrašas **(col1,col2,...)** sukuria kelis stulpelius apimantį indeksą (*multiple-column index*). Indekso reikšmės formuojamos sukabinant stulpelių reikšmes.

CHAR ir **VARCHAR** stulpelių indeksams galima naudoti stulpelių dalį; pavyzdžiui, 10 pirmųjų stulpelio **name** simbolių:

```
mysql> CREATE INDEX part_of_name ON customer (name(10));
```

Į pradžią

28. DROP INDEX sintaksė

```
DROP INDEX index_name ON tbl_name
```

DROP INDEX panaikina **index_name** indeksą lentelei **tbl_name**. **DROP INDEX** įvestas nuo MySQL versijos 3.22.

Į pradžią

29. Komentarų sintaksė

MySQL serveris palaiko tokius komentarų stilius: nuo # iki eilutės galo, nuo -- iki eilutės galo ir /* bet kokio ilgio */:

```
mysql> select 1+1;        # Šis komentaras baigiasi eilutės gale
mysql> select 1+1;        -- Šis komentaras baigiasi eilutės gale
mysql> select 1  /* šis komentaras yra eilutės vidule*/ + 1;
mysql> select 1+
/*
Šis komentaras tęsiasi
per kelias eilutes
*/
1;
```

Jei komentaras pradedamas “--“, po jų turi būti bent vienas tarpas!

Jei komentaras apskliaudžiamas /* ... */ , turi būti laikomasi tokių taisyklių:

- ką nors jo viduje apskliaudžiant paprastomis ir dvigubomis kabutėmis, būtina, kad jos eitų poromis net ir komentaro viduje;
- ";" žymi SQL operatoriaus pabaigą.

Į pradžią

30. CREATE FUNCTION/DROP FUNCTION sintaksė

```
CREATE [AGGREGATE] FUNCTION function_name RETURNS
{STRING|REAL|INTEGER}
      SONAME shared_library_name
DROP FUNCTION function_name
```

Vartotojo apibrėžtos funkcoijos (UDF) naudojamos MySQL išplėsti naujomis funkcijomis, kurios veikia kaip ir tikrosios (built in) MySQL funkctijos, pavyzdžiui, kaip **ABS()** ar **CONCAT().**

AGGREGATE - nauja opcija MySQL 3.23. **AGGREGATE** funkcija veikia kaip ir tikroji MySQL **GROUP** funcija **SUM** ar **COUNT()????**

CREATE FUNCTION išsaugo mysql.func sisteminėje lentelėje funkcijos pavadinimą, tipą ir bendrąjį bibliotekos pavadinimą. Kad galėtumėt kurti ir išmesti funkcijas, turite turėti įterpimo ir naikinimo teises (privileges) mysql duomenų bazėje.

Į pradžią

31. Apie rezervuotus žodžius MySQL

Žemiau išvardinti žodžiai yra rezervuotieji MySQL. Daugumos iš jų negalima naudoti lentelių ar stulpelių vardams:

action	add	aggregate	all	alter	after
and	as	asc	avg	avg_row_length	
auto_increment	between	bigint	bit	binary	blob
bool	both	by	cascade	case	char
character	change	check	checksum	columncolumns	
comment	constraint	create	cross	current_date	
current_time	current_timestamp		data	database databases	
date	datetime	day	day_hour	day_minute	
day_second	dayofmonth	dayofweek	dayofyear	dec	decimal
default	delayed	delay_key_write	delete	desc	describe
distinct	distinctrow	double	drop	end	else
escape	escaped	enclosed	enum	explain	exists
fields	file	first	float	float4	float8
flush	foreign	from	for	full	function
global	grant	grants	group	having	heap
high_priority	hour	hour_minute	hour_second	hosts	identified
ignore	in	index	infile	inner	insert
insert_id	int	integer	interval	int1	int2
int3	int4	int8	into	if	is
isam	join	key	keys	kill	
last_insert_id	leading	left	length	like	lines
limit	load	local	lock	logs	long
longblob	longtext	low_priority max	max_rows	match	
mediumblob	mediumtext	mediumint	middleint	min_rows	minute
minute_second	modify	month	monthname	myisam	natural
numeric	no	not	null	on	optimize
option	optionally	or	order	outer	outfile

pack_keys	partial	password	precision	primary	procedure
process	processlist	privileges	read	real	references
reload	regexp	rename	replace	restrict	returns
revoke	rlike	row	rows	second	select
set	show	shutdown	smallint	soname	
sql_big_tables		sql_big_selects	sql_low_priority_updates		sql_log_off
sql_log_update		sql_select_limit	sql_small_result		sql_big_result
sql_warnings	straight_join	starting	status	string	table
tables	temporary	terminated	text	then	time
timestamp	tinyblob	tinytext	tinyint	trailing to	
type	use	using	unique	unlock unsigned	
update	usage	values	varchar	variables	varying
varbinary	with	write	when	where	year
year_month	zerofill				

Žemiau išvardinti žodžiai yra rezervuotieji ANSI SQL, tačiau MySQL juos leista naudoti lentelių ar stulpelių vardams. Taip padaryta dėl jų natūralumo.

•ACTION •BIT •DATE •ENUM •NO •TEXT •TIME •TIMESTAMP