Реализация языка SQL в СУБД MySQL

Студент нанимается на лето подработать в университете, его спрашивают:
- Языком владеешь?
- Я в совершенстве владею языком!
- Отлично, будешь наклеивать марки на конверты.

Обзор

СУБД MySQL предоставляет в Ваше распоряжение подмножество языка SQL, соответствующее спецификации ANSI SQL 92.

Основные цели MySQL - быстродействие и ошибкоустойчивость. Добавление транзакций принесет значительное быстродействие и повышение сложности. В настоящее время имеется проект, который должен дать подобные функциональные Это, вероятно, будет выполнено, с помощью введения атомарной модификации нескольких таблиц сразу.

Ядро, на котором сформирован MySQL - набор подпрограмм, которые использовались в высокотребовательном окружении много лет. В то время, как MySQL все еще находится в разработке, это уже предоставляет богатый и полезный функциональный набор.

ALTER TABLE

СИНТАКСИС:

ALTER [IGNORE] TABLE table_name alter_specification [, alter_specification ...]

```
  alter_specification:     ADD [COLUMN] create_definition or     CHANGE
[COLUMN] old_column_name create_definition or     ALTER [COLUMN]
column_name { SET default | DROP DEFAULT } or     DROP [COLUMN]
column_name or DROP PRIMARY KEY or DROP INDEX key_name     DROP FOREIGN
KEY key_name
```

ОПИСАНИЕ:

Команда ALTER TABLE может быть использована для изменения определения таблицы. ALTER TABLE работает с временно созданной таблицей в которую копирует все данные из текущей таблицы. Когда копия готова, старая таблица удаляется, а новая переименуется в нее. Это выполнено таким способом, что все изменения автоматически переназначаются на новую таблицу.

Пока работает ALTER TABLE, старая таблица доступна для других клиентов. Обновления и запись в таблицу останавливаются и будут выполнены только после того, как новая таблица будет готова. Если IGNORE не определен, то копирование будет прервано и процесс отработан назад в случае наличия любых уникальных ключей, дублированных в новой таблице.

- CHANGE column_name, DROP column_name и DROP INDEX являются MySQL расширениями ANSI SQL.
- [COLUMN] факультативный параметр и может быть опущен.
- Конструкция ALTER [COLUMN] может быть использована для изменения или удаления старого значения по умолчанию.
- ADD и CHANGE используют один и тот же create_definition, что и CREATE TABLE. См. CREATE TABLE.
- Если вы удаляете column_name, которое является частью составного ключа, то часть ключа будет удалена. Если все части ключа удалены, то будет удален весь ключ.
- DROP PRIMARY KEY удаляет первый уникальный ключ в таблице.
- CHANGE делает лучшее преобразование существующей информации в новый формат.
- Синтаксис DROP FOREIGN KEY пока существует для запланированных функциональных возможностей. В настоящее время не делает ничего.

Вы можете использовать функцию C API mysql_info(&MYSQL_RESULT) чтобы узнать, сколько записей скопировано и сколько удалено из-за дублированных ключей.

Для использования команды ALTER TABLE вы должны иметь права доступа *select*, *insert*, *delete*, *update*, *create* и *drop* для этой таблицы.

CREATE TABLE

СИНТАКСИС:

CREATE TABLE table_name (create_definition, ...)

Здесь create_definition имеет следующий формат:

```
  create_definition:     column_name type NOT NULL [DEFAULT
default_value] [ PRIMARY KEY ] or     column_name type [NULL] [ PRIMARY
KEY ] or     PRIMARY (KEY|INDEX) [key_name] ( column_name,... ) or
(KEY|INDEX) [key_name] ( column_name[length],...) or     INDEX
[key_name] ( column_name[length],...) or     UNIQUE
(column_name[length],...) or     FOREIGN (KEY|INDEX) [key_name]
(column_name[length],...)             REFERENCES table_name   [ON
DELETE (RESTRICT | CASCADE | SET NULL) ]
```

ОПИСАНИЕ:

В MySQL все поля имеют неявное значение по умолчанию, если объявлены, как не пустые (NOT NULL). Если вы не даете значения по умолчанию при использовании не пустого поля, оно будет назначено, исходя из типа поля.

Блок FOREIGN нужен только для совместимости. Ключевое слово REFERENCE тоже не выполняет в данной версии никаких действий.

Команда MySQL CREATE TABLE не поддерживает ключевое слово SQL CHECK.

Для создания таблицы Вы должны иметь права доступа *create*.

Замечания:

- Номер столбца может иметь дополнительное ключевое слово AUTO_INCREMENT, чтобы автоматически получить номер = самый большой номер столбца + 1 для каждой вставки, в которой номер столбца = 0 или NULL. ТО ЕСТЬ, если Вы попробуете вставить значение ноля в числовой столбец, который имеет атрибут AUTO_INCREMENT, Вы получите номер столбца, который на 1 большим, чем самый большой предварительно использованный номер.

  Если Вы желаете начать отсчет не с ноля, просто вставьте желательное стартовое значение в первой записи, которую Вы вставляете в данную таблицу. В настоящее время нет никакого другого способа достичь этого эффекта.

  ВНИМАНИЕ:

Если Вы используете AUTO_INCREMENT, Вы можете использовать его только в одном поле таблицы. Обратите внимание также, что это поле должно быть объявлено как первичный ключ, и должно быть числовым.

- ZEROFILL означает, что значение дополняется слева нулями до максимальной длины поля.

  ПРИМЕР:

  `INT(5) ZEROFILL;` значение `5` превращается в `"00005"`

- Столбцы ключа и столбцы TIMESTAMP не могут быть пустыми. Для столбцов ключа атрибут NULL тихо удаляется.
- Вы можете вставить NULL для полей типа TIMESTAMP и числовых полей с атрибутом AUTO_INCREMENT.
- BLOB столбцы не мог быть ключами. Вы не можете группировать на BLOB. Однако, можно использовать строковые функции MySQL, чтобы группировать на подразделах BLOB.
- Теперь можно использовать BLOB столбцы в предложении WHERE.
- Удаленные записи находятся в связанном списке, и последующие вставки будут повторно использовать старые позиции.
- Каждый столбец, который может принять значение NULL, берет 1 бит дополнительного пространства.
- Если нет никаких VARCHAR столбцов, и BLOBs, то MySQL использует фиксированный формат записей. Вы можете ожидать существенно лучшую эффективность, в этом случае. Также не нужно оптимизировать ваши таблицы с помощью isamchk, когда используется фиксированный формат записи.
- Если Вы используете записи переменной длины и делаете много модификаций, Вы должны выполнять время от времени '`isamchk -r table_name`' на таблице, чтобы получить лучшее размещение. Попробуйте команду '`isamchk -ei table_name`' для сбора статистики.
- Максимальная длина записи может быть найдена так:
  1 + сумма длин столбцов + null_columns/8 + число столбцов переменной длины.
- В некоторых случаях атрибуты могут тихо меняться после создания: VARCHAR столбцы с длиной 1 или 2 изменяется на CHAR. При использовании одних VARCHAR столбцов все CHAR столбцы более длинные, чем 2 изменяется на VARCHARS.
- При INSERT/UPDATE все строки (CHAR и VARCHAR) приводятся к максимальной длине, заданной, CREATE. Все хвостовые пробелы автоматически удаляются.
  Например, VARCHAR(10) задает, что столбец может содержать строки с длиной до 10 символов.
- Что угодно/0 дает значение NULL.
- REGEXP использует кодировку ISOLATIN1 при использовании функций символьного типа, подобно [[:ALPHA:]].

Типы данных

Поля должны иметь один из следующих типов данных:

BIGINT [(length)]	8 байт целое (если компилятор поддерживает такой тип)

[UNSIGNED] [ZEROFILL]	
BLOB	Двоичный объект (максимальная длина 65535 байт)
CHAR(NUM)	Строка фиксированной длины (1 <= NUM <= 255)
DATE	Сохраняет информацию о дате. Использует формат "YYYY-MM-DD". Может модифицироваться как строка или число, хотя Вы, вероятно, используете контекст строки для времени и даты. MySQL тип DATEпонимает по крайней мере следующие синтаксис. ○ YYYY-MM-DD (Обратите внимание что '- ' может фактически быть ЛЮБОЙ не цифрой) ○ YY-MM-DD (Обратите внимание что '- ' может фактически быть ЛЮБОЙ не цифрой) ○ YYMMDD ○ YYMM Диапазон для этого типа данных от 0000-00-00 до 9999-12-31. Так что "проблема 2000" здесь не стоит. В отличие от TIMESTAMP, DATE принимает годы и в виде двух цифр от 0000 до 0099. Это не очень полезно в большинстве случаев. Используйте задание лет четырьмя цифрами в полях типа DATE. Тип DATE имеет длину 4 байта.
DATETIME	Объединение типов DATE и TIME. Тип DATETIME идентичен типу TIMESTAMP со следующими исключениями: ○ Когда запись вставляется в таблицу, содержащую поля типа DATETIME, поле DATETIME не изменяется. ○ Диапазон для поля типа DATETIME: '0000-01-01 00:00:00' - '9999-12-31 23:59:59' при использовании в контексте строки, и '00000000000000' - '99991231235959' при использовании в контексте числа. Тип DATETIME имеет длину 8 байт.
DECIMAL (length,dec)	Десятичное число с плавающей запятой.
DOUBLE [(length,dec)]	Число (4 или 8 байт) двойной точности с максимальной длиной и фиксированном числом десятичных чисел.
FLOAT [(precision)]	Номер с плавающей запятой. FLOAT(4) и FLOAT одиночная точность. FLOAT(8) обеспечивает двойную точность.
FLOAT [(length,decimals)]	Число одиночной точности с максимальной длиной и фиксированном числом десятичных чисел (4 байта).
INT [(length)] [UNSIGNED] [ZEROFILL]	Целое (4 байта).
INTEGER [(length)] [UNSIGNED] [ZEROFILL]	Целое число 4 байта
LONGBLOB	Двоичный объект с максимальной длиной 2**32 байт.

MEDIUMBLOB	Двоичный объект с максимальной длиной 16777216 байт.
MEDIUMINT [(length)] [UNSIGNED] [ZEROFILL]	Целое (3 байта).
REAL [(length,dec)]	Идентично DOUBLE (8 байт).
SMALLINT [(length)] [UNSIGNED] [ZEROFILL]	Целое (2 байта).
TINYBLOB	Двоичный объект с максимальной длиной 255 байт.
TINYINT [(length)] [UNSIGNED] [ZEROFILL]	Целое число (1 байт).
VARCHAR(NUM)	Строка переменной длины (1 <= NUM <= 255)
TIME	Хранит информацию о времени. Использует формат "HH:MM:SS". Может использоваться как строка или число. MySQL тип TIME понимает следующий синтаксис. ○ HH:MM:DD ○ HHMMDD ○ HHMM ○ HH Данные типа TIME имеют длину 3 байта.
TIMESTAMP(NUM)	Автоматически изменяется при вставке/обновлении. Имеет формат YYMMDDHHMMSS или YYYYMMDDHHMMSS. Вы можете модифицировать поле TIMESTAMP при выполнении INSERT. Это полезно, когда Вы хотите установить произвольную дату/время для записи. В течение модификаций Вы не должны определять значение для вашего поля TIMESTAMP, или определять NULL как значение, для вставки. Иначе вы получите недопустимое значение для этого поля. Когда используете mysql с ODBC и Access Вы должны использовать значение 14 для NUM, поскольку это заставляет MySQL всегда использовать в годах четыре цифры. Значение 12 заставит MySQL использовать в году две цифры. Значение по умолчанию - 14. Обратите внимание, что в случае таблиц с несколькими полями TIMESTAMP только первое такое поле будет модифицироваться автоматически.

Длина поля определяет, сколько всего цифр может иметь число, в то время как поле dec определяет, сколько из этих цифр будет после десятичной точки. Эти значения используются только для форматирования и вычисления максимальной ширины столбца.

Ключи

MySQL таблица может иметь до 16 ключей, каждый из которых может иметь до 15 полей. Максимальная поддерживаемая длина ключа 120 байт. Вы можете

увеличить длину ключа, изменяя N_MAX_KEY_LENGTH в файле nisam.h и перекомпилировав пакет. Обратите внимание, что длинные ключи могут привести к низкой эффективности.

Ключи могут иметь имена. В случае первичного ключа имя будет всегда PRIMARY. Если имя ключа не задано в процессе создания таблицы, то заданное по умолчанию имя ключа - первое имя столбца с факультативным суффиксом (_2, _3, и т. д.) чтобы сделать это имя уникальным. Имя ключа может использоваться с командой ALTER TABLE, чтобы удалить ключ.

При создании ключа Вы можете факультативно определить, что только первые N символов поля будут использоваться. Например, если Вы хотите создавать уникальный ключ на поле, в котором только первые 40 символов уникальны, можно сделать следующее.

```
CREATE TABLE SomeTable (composite CHAR(200), INDEX
comp_idx(composite(40)));
```

Хорошая идея - использовать эту опцию на неуникальных полях, поскольку эта мера значительно уменьшит размер вашего индекса, а снижение производительности будет очень не большим.

Вы можете иметь один первичный ключ на таблицу. Если поле определено, как поле первичного ключа, то генерируется индекс. Нет никакой необходимости определять нормальный ключ. Кроме того, при определении дополнительных индексов, которые содержат первичный ключ не будет иметь смысла, поскольку первичный ключ сделает индекс бесполезным.

Ключи с несколькими полями следует использовать для оптимизации узкоспецифических запросов. То есть, все поля в предложении WHERE запроса должны появляться в многопольном ключе.

Поскольку MySQL использует B-Tree не нужно объявлять ключи, которые являются префиксами других ключей. Оптимизатор найдет любой пригодный для использования префикс ключа и использует его, чтобы выполнить поиск. Например, если Вы объявляете следующий ключ:

```
INDEX (first, second, third, fourth)
```

Вы также неявно создали следующие ключи:

```
(first, second, third)
(first, second)
(first)
```

Объявление ненужных ключей только займет дополнительное место и замедлит ваши запросы. Ключи должны быть созданы во время создания таблицы или изменения таблицы с использованием команды ALTER TABLE.

BLOB'ы

BLOB - "Binary Large OBject" - двоичный большой объект.

Как отмечено выше, MySQL поддерживает четыре типа BLOB:

tinyblob (0-255 байт) blob (0-65535 байт) mediumblob (0-16777216 байт) longblob (0-2147483648 байт)

Обратите внимание, что могут иметься некоторые ограничения из-за размера буфера сообщения. Буфер сообщений выделяется динамически. Вы должны знать, что 'max_allowed_packet' устанавливается на сервере и клиенте. По умолчанию, это - 64КБ для сервера и 512КБ для клиента.

Вы можете сменить размер буфера, запустив `mysqld` с опцией `-O`. Но помните, что это количество памяти будет выделяться каждому потоку!

ПРИМЕР:

```
mysqld -O max_allowed_packet=max_blob_length
```

MySQL WIN95 ODBC драйвер определяет BLOB как LONGVARCHAR.

Двоичные данные в BLOBS

Если Вы вставляете двоичные данные в BLOB, Вы не должны применять следующие символов:

- \0
- \\
- ' или "

CREATE INDEX

СИНТАКСИС:

```
CREATE [UNIQUE] INDEX index_name ON table_name (column_name,... )
```

ОПИСАНИЕ:

В MySQL эта команда проверит был ли данный индекс создан, когда создавалась таблица. Она не создает индекс. Это предусмотрено по причинам совместимости. Если Вы хотите добавить, ключ используйте команду ALTER TABLE.

DELETE

СИНТАКСИС:

```
DELETE FROM table_name WHERE where_definition
```

Здесь where_definition имеет формат:

```
where_definition:   where_expr or where_expr [AND | OR]
where_expr
```

where_expr имеет формат:

```
where_expr:   column_name [> | >= | = | <> | <= | < ]
column_name_or_constant or   column_name LIKE
```

```
column_name_or_constant or   column_name IS NULL or column_name
IS NOT NULL or (where_definition)
```

ОПИСАНИЕ:

Удаляет записи из таблицы.

- Возвращает количество обработанных записей.
- Если вызван DELETE без WHERE, то таблица будет очищена. В этом случае DELETE вернет 0 для числа обработанных записей.

Замечания:

- Все строки сравниваются без учета регистра (ISO_8859_1). Если Вы должны сделать чувствительный к регистру поиск, то используйте REGEXP в предложении HAVING.
- LIKE применим на числовых столбцах.
- Сравнение с явным NULL (столбец == NULL) эквивалентно условию IS NULL, то есть использованию (столбец IS NULL). Это было сделано, для совместимости с mSQL.

Вы должны иметь права доступа *delete* для удаления записей.

DESCRIBE

СИНТАКСИС:

```
(DESCRIBE | DESC) table [column]
```

ОПИСАНИЕ:

Описывает таблицу или столбец. Эта команда подобна команде SHOW. Факультативный параметр [column] может быть именем столбца или строкой. Если [column] - строка, он может содержать символы подстановки.

DROP

СИНТАКСИС:

```
DROP TABLE table_name [table_name ...]
```

ОПИСАНИЕ:

Удаляет (в оригинальной документации почему-то сказано, что роняет) одну или несколько таблиц.

Если Вы хотите только удалить все данные в таблице и сохранить ее структуру для будущего повторного заполнения, Вы можете использовать командуDELETE.

ОСТЕРЕГАЙТЕСЬ! DROP TABLE полностью удалит именованную таблицу(ы) из вашей системы. Не предусмотрено никакого UNDO или UNERASE (если Вы не имеете резервной копии, конечно).

Вы должны иметь права доступа *delete*, чтобы использовать DROP.

DROP INDEX

СИНТАКСИС:

```
DROP INDEX index_name
```

ОПИСАНИЕ:

Эта команда ничего не делает. Чтобы удалить индекс, Вы должны использовать команду ALTER TABLE.

DROP INDEX предусмотрен по причине совместимости. Это вводит в заблуждение некоторые клиенты, которые думают, что получили то, что они просили. Прежде всего это касается тупых ODBC драйверов.

GRANT

СИНТАКСИС:

```
  GRANT (ALL PRIVILEGES | (SELECT, INSERT, UPDATE, DELETE, REFERENCES
(column list), USAGE)) ON table TO user,... [WITH GRANT OPTION]
```

ОПИСАНИЕ:

Команда GRANT ничего не делает. Она всегда возвращает истину и нужна прежде всего, чтобы ввести в заблуждение некоторые прикладные программы, которые используют ODBC и думают, что команда GRANT, которую они выдали, что-то сделала. Вообще, ODBC такая библиотека, что для совместимости с ней предусмотрена не одна функция... См. главу Администрирование пакета для получения подробностей по поводу прав доступа в MySQL.

SELECT

СИНТАКСИС:

```
  SELECT [STRAIGHT_JOIN] [DISTINCT | ALL] select_expression,...
[FROM tables... [WHERE where_definition] [GROUP BY column,...]
[ORDER BY column [ASC | DESC], ...] HAVING full_where_definition
[LIMIT [offset,] rows] [PROCEDURE procedure_name]]          [INTO
OUTFILE 'file_name'... ]
```

Здесь where_definition:

```
  where_definition:     where_expr or where_expr [AND | OR] where_expr
```
where_expr имеет формат:
```
  where_expr:     column_name [> | >= | = | <> | <= | <]
column_name_or_constant or column_name LIKE column_name_or_constant or
column_name IS NULL or column_name IS NOT NULL or (where_definition)
```

ОПИСАНИЕ:

Оператор SELECT является краеугольным камнем всего языка SQL. Он используется, чтобы выполнить запросы к базе данных. Это действительно основа языка SQL. Для хорошего общего учебника о том, как работает SELECT, посмотрите http://w3.one.net/~jhoffman/sqltut.htm#Basics of the SELECT Statement.

В MySQL версии меньше 3.21.x предложение WHERE очень ограничено. HAVING будет работать там, где предложение WHERE ничего не делает. Некоторые примеры, которые не работают в предложении WHERE - REGEXP и операторе !. В основном, Вы не можете использовать функции с WHERE, но Вы можете использовать функции с HAVING.

HAVING по существу, WHERE применительно к результатам. Он используется главным образом для узкой области данных, возвращенных запросом.

Вы должны иметь права *select* для использования SELECT.

Функции

`select_expression` может содержать следующие функции и операторы:

`+ - * /`	Арифметические действия.
`%`	Остаток от деления (как в C)
`| &`	Битовые функции (используется 48 бит).
–	Смена знака числа.
`( )`	Скобки.
`BETWEEN(A,B,C)`	(A >= B) AND (A <= C).
`BIT_COUNT()`	Количество бит.
`ELT(N,a,b,c,d)`	Возвращает a, если N == 1, b, если N == 2 и т. д. a,b,c,d строки. **ПРИМЕР:** ELT(3,"First","Second","Third","Fourth") вернет "Third".
`FIELD(Z,a,b,c)`	Возвращает a, если Z == a, b, если Z == b и т. д. a,b,c,d строки. **ПРИМЕР:** FIELD("Second","First","Second","Third","Fourth") вернет "Second".
`IF(A,B,C)`	Если A истина (!= 0 and != NULL), то вернет B, иначе вернет C.
`IFNULL(A,B)`	Если A не null, вернет A, иначе вернет B.
`ISNULL(A)`	Вернет 1, если A == NULL, иначе вернет 0. Эквивалент ('A == NULL').
`NOT !`	NOT, вернет TRUE (1) или FALSE (0).

OR, AND	Вернет TRUE (1) или FALSE (0).
SIGN()	Вернет -1, 0 или 1 (знак аргумента).
SUM()	Сумма столбца.
= <> <= < >= >	Вернет TRUE (1) или FALSE (0).
expr LIKE *expr*	Вернет TRUE (1) или FALSE (0).
expr NOT LIKE *expr*	Вернет TRUE (1) или FALSE (0).
expr REGEXP *expr*	Проверяет строку на соответствие регулярному выражению expr.
expr NOT REGEXP *expr*	Проверяет строку на соответствие регулярному выражению expr.

select_expression может также содержать один или большее количество следующих математических функций.

ABS()	Абсолютное значение (модуль числа).
CEILING()	()
EXP()	Экспонента.
FORMAT(nr,NUM)	Форматирует число в формат '#,###,###.##' с NUM десятичных цифр.
LOG()	Логарифм.
LOG10()	Логарифм по основанию 10.
MIN(),MAX()	Минимум или максимум соответственно. Должна иметь при вызове два или более аргументов, иначе рассматривается как групповая функция.
MOD()	Остаток от деления (аналог %).
POW()	Степень.
ROUND()	Округление до ближайшего целого числа.
RAND([integer_expr])	Случайное число типа float, 0 <= x <= 1.0, используется integer_expr как значение для запуска генератора.
SQRT()	Квадратный корень.

select_expression может также содержать одну или больше следующих строковых функций.

CONCAT()	Объединение строк.
INTERVAL(A,a,b,c,d)	Возвращает 1, если A == a, 2, если A == b... Если совпадений нет, вернет 0. A,a,b,c,d... строки.
INSERT(org,strt,len,new)	Заменяет подстроку org[strt...len(gth)] на new. Первая позиция строки=1.
LCASE(A)	Приводит A к нижнему регистру.
LEFT()	Возвращает строку символов, отсчитывая слева.
LENGTH()	Длина строки.
LOCATE(A,B)	Позиция подстроки B в строке A.
LOCATE(A,B,C)	Позиция подстроки B в строке A, начиная с позиции C.
LTRIM(str)	Удаляет все начальные пробелы из строки str.
REPLACE(A,B,C)	Заменяет все подстроки B в строке A на подстроку C.
RIGHT()	Get string counting from right.

RTRIM(str)	Удаляет хвостовые пробелы из строки str.
STRCMP()	Возвращает 0, если строки одинаковые.
SUBSTRING(A,B,C)	Возвращает подстроку из A, с позиции B до позиции C.
UCASE(A)	Переводит A в верхний регистр.

И наконец несколько просто полезных функций, которые тоже можно применить в select_expression.

CURDATE()	Текущая дата.
DATABASE()	Имя текущей базы данных из которой выполняется выбор.
FROM_DAYS()	Меняет день на DATE.
NOW()	Текущее время в форматах YYYYMMDDHHMMSS или "YYYY-MM-DD HH:MM:SS". Формат зависит от того в каком контексте используется NOW(): числовом или строковом.
PASSWORD()	Шифрует строку.
PERIOD_ADD(P:N)	Добавить N месяцев к периоду P (в формате YYMM).
PERIOD_DIFF(A,B)	Возвращает месяцы между A и B. Обратите внимание, что PERIOD_DIFF работает только с датами в форме YYMM или YYYMM.
TO_DAYS()	Меняет DATE (YYMMDD) на номер дня.
UNIX_TIMESTAMP([*date*])	Возвращает метку времени unix, если вызвана без *date* (секунды, начиная с GMT 1970.01.01 00:00:00). При вызове со столбцом TIMESTAMP вернет TIMESTAMP. *date* может быть также строкой DATE, DATETIME или числом в формате YYMMDD (или YYYMMDD).
USER()	Возвращает логин текущего пользователя.
WEEKDAY()	Возвращает день недели (0 = понедельник, 1 = вторник, ...).

Групповые функции в операторе select:

Следующие функции могут быть использованы в предложении GROUP:

AVG()	Среднее для группы GROUP.
SUM()	Сумма элементов GROUP.
COUNT()	Число элементов в GROUP.
MIN()	Минимальный элемент в GROUP.
MAX()	Максимальный элемент в GROUP.

Здесь MIN() и MAX() могут принимать строку или число в качестве аргумента. Эти функции не могут использоваться в выражении, хотя их параметр может быть выражением:

ПРИМЕР: "SUM(value/10)" нормально, но вот "SUM(value)/10" уже нет!

- Строки автоматически конвертируются в числа и наоборот по мере необходимости (прямо как в perl). При использовании операторов = <> <= >= < > как в инструкции WHERE, левая сторона определяет, выполняется ли тест с числами или со строками. Все сравнения строк независимы от регистра (ISO8859-1).

ПРИМЕР:

```
  "a" < "b"    ; Сравнение строк    "a" < 0      ; Сравнение строк    0  <
"a"     ; Сравнение чисел    a < 5        ; Если поле имеет тип CHAR, то
сравниваются строки,                    ; иначе сравниваются числа.
```

Если надо учитывать регистр, используйте REGEXP в HAVING.

- Имя столбца не должно иметь префикса таблицы, если данное имя столбца уникально.
- В LIKE выражения `%` и _ могут предваряться символом \ для получения символьного выражения.
- DATE является строкой с одним из синтаксисов:
  - YYMMDD (Год считается 2000, если YY < 70)
  - YYYYMMDD
  - YY.MM.DD Здесь '.' может быть любым нецифровым разделителем
  - YYYY.MM.DD Здесь '.' может быть любым нецифровым разделителем
- IFNULL() и IF() возвращает число или строку в зависимости от ситуации, в которой использованы.
- *Order* и *group* столбец может быть именем столбца, его псевдонимом или номером в операторе SELECT.
- HAVING может принимать в качестве аргумента любые поля или псевдонимы в `select_expression`. Он применяется последним перед передачей данных клиенту без какой-либо оптимизации. Не используйте его для элементов из предложения WHERE.

  Замечание: Вы не можете написать:
  `SELECT user,MAX(salary) FROM users GROUP BY users HAVING max(salary)>10`
  Вместо этого, используйте нечто вроде следующего (это хороший пример использования псевдонимов столбцов):
  `SELECT user,MAX(salary) AS sum FROM users GROUP BY users HAVING sum > 10`

- LIMIT принимает один или два аргумента. Один аргумент задает максимальное число строк в результате. В случае двух аргументов этот максимум задает второй аргумент, а первый указывает смещение первой строки.
- INTO OUTFILE 'filename' пишет результаты в файл. Файл не должен существовать на момент выполнения этой команды. См. раздел LOAD DATA INFILE для более подробной информации. Это может быть весьма опасной командой, если daemon запущен от имени root. Самое лучшее предоставить право доступа *file* только когда это абсолютно необходимо.
- Вы можете использовать числовое значение в предложении ORDER BY для определения столбца, который Вас интересует. ТО ЕСТЬ, если Вы желаете провести сортировку второго столбца, определенного в вашем запросе SELECT, следует написать "ORDER BY 2;". Это также полезно, когда Вы использовали функцию в вашем SELECT.

  ПРИМЕР:

```
  SELECT Widget_Table.widget_id, Widget_Table.widget_name,
Purchase_Order_Item.widget_id, sum(Purchase_Order_Item.quantity)
FROM Widget_Table, Purchase_Order_Item          WHERE
Widget_Table.widget_id = Purchase_Order_Item.widget_id          GROUP
BY Widget_Table.widge t_name ORDER BY 4;
```

Присоединения

Свойство объединения SQL дает способность определить связи между таблицами и отыскивать) информацию, основанную на этих связях.

Связи перечисляются в предложении FROM запроса SELECT. Каждая связь отделяется запятой.

ПРИМЕР:

```
$ mysql mysql  Welcome to the mysql monitor. Commands ends with ; or \g. Type
'help' for help.   mysql> SELECT db.user, db.delete_priv, user.user,
user.delete_priv     -> FROM db,user WHERE db.user = user.user;
```

Этот запрос соединит таблицы `db` и `user` посредством поля `user`. Это распечатает что-то вроде следующего:

```
 +------+-------------+------+-------------+ | user | delete_priv | user |
delete_priv | +------+-------------+------+-------------+ |mke   | N
| mke  | N           | +------+-------------+------+-------------+
```

Первые два поля фактические `db.user` и `db.delete_priv` , последние два `user.user` и `user.delete_priv`.

Обратите внимание, что мы используем имена таблицы в нашем запросе, чтобы определить точно, с какими полями мы работаем.

Вы можете объединить до пятнадцати таблиц в одном объединении.

MySQL не будет использовать ключи, чтобы соединить таблицы посредством полей, которые не имеют идентичный тип. Это означает, что Вы должны всегда использовать те же самые типы для полей, которые предназначены, для использования в объединениях.

Псевдонимы могут также использоваться для имен столбца. См. детали в следующем разделе.

Псевдонимы

СУБД MySQL поддерживает концепцию псевдонимов для таблиц и полей.

Псевдонимы для таблиц являются стандартной частью языка SQL.

ПРИМЕР:

```
SELECT A.user,A.select_priv,A.insert_priv,A.update_priv FROM user A
```

В этом примере использован псевдоним таблицы, чтобы сократить ваш запрос, объявляя псевдоним, который короче имени таблицы. Вы используете псевдоним в первой части выбора, и определяете это в FROM, определяя реальное имя таблицы, пробел и псевдоним. Если Вы имеете больше чем одну таблицу, для которой Вы

желаете создать псевдоним, просто добавьте запятую после каждой пары имя/псевдоним таблицы.

Если Вы используете псевдонимы с запросом, который будет иметь предложение WHERE, Вы должны использовать псевдоним в предложении WHERE вместо реального имени таблицы.

Псевдонимы для полей таблицы - специфическое для MySQL расширение.

ПРИМЕР:

```
SELECT user.user AS "User Name", user.delete_priv AS "Delete" FROM user;
```

Одно хорошее дело, которое делают псевдонимы поля - это то, что они позволяют Вам определять более дружественные метки для вашего вывода. Результат вышеупомянутого запроса мог бы окончательно выглядеть примерно так:

```
 +-----------+--------+ | User Name | Delete | +-----------+--------+ | root
| Y       | | mke        | N       | | dummy     | N       | | admin     | N
| +-----------+--------+
```

Хороший совет - брать псевдонимы в кавычки, в данном примере "Delete" вызвало бы ошибку синтаксического анализа при применение без кавычек. (Это потому, что DELETE является ключевым словом SQL.

INSERT INTO

СИНТАКСИС:

```
INSERT INTO table [(column_name, ...)] VALUES (expression,...) ||
INSERT INTO table [(column_name, ...)] SELECT ...
```

ОПИСАНИЕ:

Вставляет данные в таблицу.

- В выражении можно использовать любое предыдущее поле в списке column_name (или таблицу, если список имен столбцов не задан).
- При использовании SELECT вы не можете указать ORDER BY.
- Вы можете использовать функцию C API mysql_info для получения строки:

  @result{Records: 220 Duplicates: 1 Warnings: 1}

  - Records показывает число записей, возвращенных SELECT'ом.
  - Duplicates = число строк, которые не могли быть вставлены из-за дублирования ключей.
  - Warnings = счетчик числа столбцов в запросе SELECT, которые равны NULL, но были объявлены как NOT NULL для таблицы, в которую Вы вставляете результаты. Столбцы получат значение по умолчанию (помните: в MySQL все NOT NULL столбцы имеют значение по умолчанию!). Если Вы не объявляли при создании таблицы это самое значение, оно будет автоматически назначено, основанным на типе поля.

- Если Вы желаете вставить NULL в данное значение, Вы должны сделать это, не определяя значение для поля, в которое Вы желаете вставить NULL.

ПРИМЕР:

```
INSERT INTO Customer(customer_name,customer_contact) VALUES("Joes
Wholesale","Joe Smith")
```

Этот запрос создаст новую запись в таблице Customer, которая будет содержать автоматически сгенерированный customer_id, и значения, определенные в запросе. Все другие поля будут пустыми (NULL).

Вы также можете использовать SELECT для копирования элементов из одной таблицы в другую. MySQL поддерживает ограниченную форму запросов sub, для выполнения этой возможности.

Вы должны иметь права доступа *insert* для использования этой команды.

LOAD DATA INFILE

СИНТАКСИС:

```
LOAD DATA INFILE syntax
```

ОПИСАНИЕ:

Команды, для чтения данных из текстового файла.

ПРИМЕР:

```
LOAD DATA INFILE 'customer.tab' [REPLACE | IGNORE] INTO TABLE Customer
[fields [terminated by ',' [optionally] enclosed by '"' escaped by '\\' ]]
[lines terminated by '\n'] [(field list)]
```

Для записи в текстовый файл используйте:
```
SELECT ... INTO OUTFILE 'customer.tab' fields terminated by ',' enclosed by
'"' escaped by '\\' lines terminated by '\n' .
```

"fields terminated by"	Имеет значение по умолчанию \t.
"fields [optionally] enclosed by"	Имеет значение по умолчанию ".
"fields escaped by"	Имеет значение по умолчанию '\\'.
"lines terminated by"	Имеет значение по умолчанию '\n'.

"fields terminated by" и "lines terminated by" могут быть больше, чем 1 символом.

Если "fields terminated by" и "fields enclosed by" являются пустыми строками, то размер строки будет фиксированным. То есть, будет производиться чтение полей одной длины.

С фиксированными значениями NULL для размера строки будут выводиться пустые строки.

Если указаны "optionally" в "enclosed by" и Вы не используете фиксированный размер строк, только строки с этим символом будут включены в команду SELECT ... INTO.

Если "escaped by" не пусто, то следующие символы будут снабжены префиксом: "escaped by", ASCII 0, и первый символ из "fields terminated by", "fields enclosed by" и "lines terminated by".

Если использован REPLACE, новая строка заменит все строки, которые имеют тот же самый уникальный ключ. Если использован IGNORE, строки будут пропущен, если там уже существует запись с идентичным уникальным ключом. Если ни один из вышеупомянутых параметров не используется, будет выдана ошибка, и остальная часть textfile будет игнорироваться, если найден дублирующий ключ.

Некоторые ситуации, которые не поддерживаются LOAD DATA INFILE:

- Фиксированные размеры строк ("FIELDS TERMINATED BY" и "FIELDS ENCLOSED BY" являются пустыми) и поля BLOB.
- Разделитель, являющийся префиксом другого разделителя.
- "FIELDS ESCAPED BY" пустое и данные содержат один или несколько разделителей.

Все строки читаются в таблицу. Если строка имеет слишком мало полей, остальная часть полей в таблице устанавливается в значения по умолчанию.

По соображениям безопасности textfile должен находиться в каталоге баз данных или быть читаемым всеми.

Если "FIELDS ENCLOSED BY" не пустое, то NULL читается как значение NULL. Если "FIELDS ESCAPED" не пустое, то \N тоже читается как значение NULL. Note Обратите внимание, что это БОЛЬШАЯ N, верхний регистр!

Когда запрос LOAD DATA выполнен, Вы можете получить следующую строку информации, используя функцию C API mysql_info().

```
@result{Records: 1 Deleted: 0 Skiped: 0 Warnings: 0}
```

Переменная Warnings увеличивается с каждым столбцом, который не может быть сохранен без потери точности, для каждого столбца, который не получал значение из строки текста при чтении (это случается, если строка слишком короткая) и для каждой строки, которая имеет большее количество данных чем может вписываться в данные столбцы.

Вы должны иметь права доступа *select* и *insert* таблице *user* для использования этой команды.

SET OPTION

СИНТАКСИС:

```
SET OPTION SQL_VALUE_OPTON=value, ...
```

ОПИСАНИЕ:

Меняет или устанавливает опции MySQL. Опции действуют только в пределах текущего сеанса.

MySQL поддерживает следующие опции (в этой версии пока одну):

SQL_SELECT_LIMIT=value	Максимальное число записей, которое возвращает SELECT. Если SELECT имеет параметр LIMIT, то используется значение из этой опции.

UPDATE

СИНТАКСИС:

```
UPDATE table SET column=expression,... WHERE where_definition
```

Здесь where_definition:

```
 where_definition:   where_expr or where_expr [AND | OR] where_expr
```

Здесь where where_expr имеет формат:

```
where_expr:   column_name [> | >= | = | <> | <= | < ]
column_name_or_constant or   column_name LIKE column_name_or_constant or
column_name IS NULL or column_name IS NOT NULL or (where_definition)
```

ОПИСАНИЕ:

Обновляет одно или несколько полей в таблице MySQL.

- Все обновления выполняются слева направо.
- Внутри UPDATE на одной таблице все операции атомарные. Например, Вы можете увеличивать счетчик внутри таблицы, просто прибавляя 1 к соответствующей переменной.

ПРИМЕРЫ:

```
UPDATE Widget_Table SET widgets_on_hand=widgets_on_hand - 300 where
widget_id=3;
```

Этот запрос вычтет 300 из значения widgets_on_hand для widget = 3.

```
DELETE FROM Purchase_Order_Item WHERE purchase_order = 456
```

Этот запрос удалит все записи из Purchase_Order_Item, которые имеют значение 456 для purchase_order. Обратите внимание, что вообще Вы НИКОГДА не должны бы удалять данные из этого сорта базы данных. Вы создаете базы данных, чтобы следить за информацией, и даже плохая информация могла бы стать полезной в некотором случае. Гораздо лучше иметь некоторый тип кода состояния, который Вы используете, когда данные стали недопустимыми по каким-либо причинам.

Вы также хотели бы удалять запись в Purchase_Order для purchase_order 456. Важно убедиться, что, когда Вы удаляете информацию, Вы избавляетесь от всех ссылок к этой информации. Иначе Вы закончите с разрушенной базой данных.

Вы должны иметь права доступа *update* для использования этой команды.

SHOW

СИНТАКСИС:

```
SHOW DATABASES [LIKE wild]
SHOW KEYS FROM table_name
SHOW TABLES [FROM database] [LIKE wild]
SHOW [COLUMNS|FIELDS] FROM table [FROM database] [LIKE wild]
```

ОПИСАНИЕ:

Отображает информацию о базе данных MySQL. "wild" эквивалент регулярному выражению для SQL LIKE.

ПРИМЕР:

```
$ mysql WidgetDB  Welcome to the mysql monitor.  Commands ends with ; or \g.
Type 'help' for help.  mysql> SHOW fields FROM Widget_Table from WidgetDB;  6
rows in set (0.34 sec) +--------------------+-------------+------+-----+----
-----+----------------+ | Field              | Type        | Null | Key |
Default | Extra          | +--------------------+-------------+------+-----
+---------+----------------+ | widget_id          | mediumint(8) |      | PRI
| 0       | auto_increment | | widget_name        | char(60)     |      | MUL
|         |                | | widget_color_id    | mediumint(8) |      | MUL
| 0       |                | | widget_size_id     | mediumint(8) |      |
| 0       |                | | widgets_on_hand    | smallint(5)  |      |
| 0       |                | | widget_price       | float(8,2)   |      |
| 0.00    |                | | commission_percent | float(4,2)   |      |
| 0.00    |                | +--------------------+-------------+------+----
-+---------+----------------+  mysql>
```

Первые два поля довольно очевидны. `Null` будет содержать YES, если это поле может быть равным NULL, `Key` сообщает имеет ли это поле индекс, `Default`сообщает Вам значение по умолчанию, которое будет назначено этому полю, если там ничего не окажется после выполнения команды INSERT, `Extra` указывает другие атрибуты поля, такие как AUTO_INCREMENT, например.

О строках

- Строка может иметь ' или " в качестве ограничителей.
- \ является управляющим символом. Распознаются следующие управляющие последовательности:

\0	ASCII 0. Примечание: это - 5C 30, а не 5C 00!
\n	Новая строка.
\t	Табуляция.
\r	Возврат каретки.
\b	backspace
\'	'

\"	"
\\	\
\%	% (используется в строках с символами подстановки для поиска '%')
_	_ (используется в строках с символами подстановки для поиска '_')

Примеры правильных строк:

- `'hello'`
- `"hello"`
- `'""hello""'`
- `"'ello"`
- `"'e"l"lo"`
- `'\'hello'`
- `"This\nIs\nFour\nlines"`

' в строке записывается как ''.

" в строке записывается как "".

Пример, чтобы прояснить ситуацию:

```
 mysql> select 'hello',''hello'',""hello"",'''h''e''l''l''o''',"hel""lo";
1 rows in set (0.01 sec)  +-------+---------+-----------+-------------+------
--+ | hello | 'hello' | ""hello"" | 'h'e'l'l'o' | hel"lo | +-------+---------
+-----------+-------------+--------+ | hello | 'hello' | ""hello"" |
'h'e'l'l'o' | hel"lo | +-------+---------+-----------+-------------+--------+
```

О числах

- Целые числа состоят из последовательности цифр.
- Плавающие числа состоят из последовательности цифр с факультативным десятичным разделителем представляемым точкой ".".
- В версии 3.20.X все вычисления выполняются с числами типа doubles, что приводит к тому, что большие значения типа ulonglong усекаются. В версии 3.21.X это исправлено.

Имена таблиц и столбцов

Вы можете использовать только набор символов ISO8859-2 (или набор символов который Вы определили при компиляции начальном выборе конфигурации и компиляции пакета), символ подчеркивания, и 0-9 в именах столбцов.

Дефисы, пробелы и другие специальные символы не могут использоваться, поскольку они сделали бы невозможным использование таблицы или столбца в операторе SELECT.