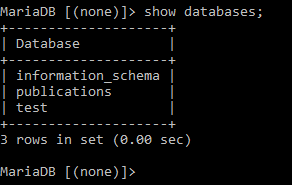
**Начало работы с интерфейсом командной строки**

**D:\xampp\mysql\bin\mysql -u max –p** \\Пример доступа к MySQL в windows **–u**(пользователь) **–p**(пароль)

**mysql -u root –p** \\Пример доступа к MySQL в windows **–u**(пользователь) **–p**(пароль)

Что бы вывести все база данных нужно ввести команду **SHOW databases; (не забаваем о точке с запятой):**



Для выхода из **MySQL** используем команду **quit.**

**Отмена команды**

Если, набрав часть команды, вы решили, что ее вообще не следует выполнять, то ни в коем случае не пользуйтесь сочетанием Ctr + C ! Оно закроет программу. Вместо нее можно ввести символы **\c** и нажать клавишу Enter .

**Поддержка русского языка**

**CREATE DATABASE** sales **CHARACTER SET** utf8**;**

**БЕКАПЫ**

Делаем бекап  
mysqldump -u USER -pPASSWORD DATABASE > /path/to/file/dump.sql

Заливаем бекап в базу данных  
mysql -u USER -pPASSWORD DATABASE < /path/to/dump.sql

**Организация доступа пользователей**

Давайте начнем с создания нового пользователя из консоли MySQL:

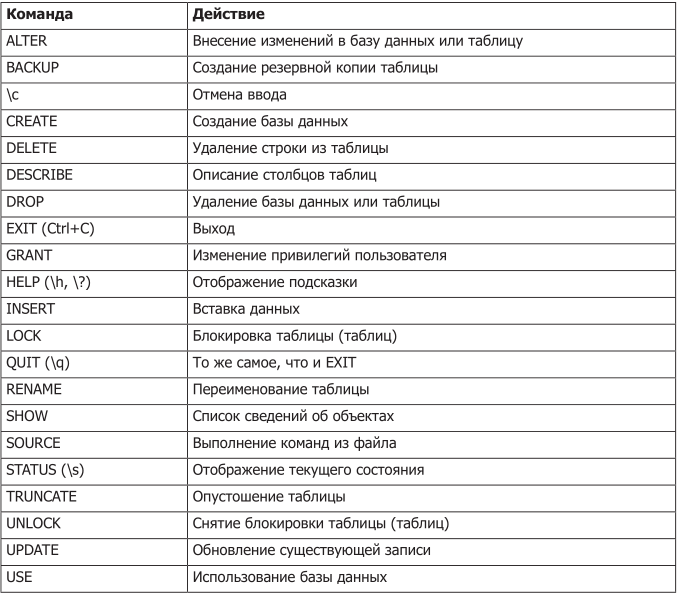
**CREATE USER** 'newuser'@'localhost' **IDENTIFIED BY** 'password';

**GRANT** права **ON** база\_данных.объект **TO** 'имя\_пользователя@имя\_хоста' **IDENTIFIED BY** 'пароль';

**GRANT** ALL PRIVILEGES **ON** publications**.**\* **TO** 'max'@'localhost' **IDENTIFIED BY** 'max';

Обновляем права: **FLUSH PRIVILEGES**;

**Команды MySQL**



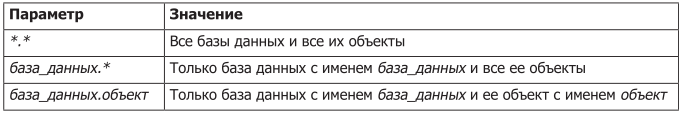
**Создание базы данных**

**CREATE DATABASE publications;** - создание базы данных.

**USE publications;** - использование указанной базы.

**ALTER TABLE** `users` **CONVERT TO CHARACTER SET** 'utf8'; - что бы понимал русский

**Создание таблицы**



CREATE TABLE classics (

author VARCHAR(128),

title VARCHAR(128),

type VARCHAR(16),

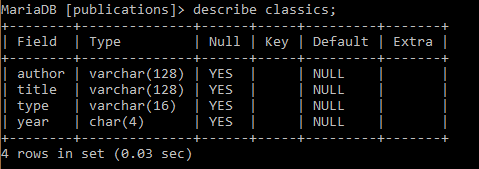
year CHAR(4)) ENGINE MyISAM;

или в одну строку

create table classics (author VARCHAR(128), title VARCHAR(128), type VARCHAR(16), year CHAR(4)) ENGINE MyISAM;

Команда **ENGINE MyISAM** , которая указывает **MySQL** тип механизма управления базой данных, применяемого к этой таблице. Чтобы проверить факт создания новой таблицы, наберите команду:

DESCRIBE classics;



Field — имя каждого из полей или столбцов таблицы;

Type — тип данных, сохраняемых в поле;

Null — заголовок, который показывает, может ли поле содержать значение **NULL** ;

Key — **MySQL** поддерживает ключи, или индексы, позволяющие ускорить просмотр и поиск данных. Под заголовком **Key** показан тип применяемого ключа (если таковой имеется);

Default — исходное значение, присваиваемое полю, если при создании новой строки не указано никакого значения;

Extra — дополнительная информация, например, о настройке поля на автоматическое приращение его значения.

**Типы данных**

**VARCHAR** - означает VARiable length CHARacter string — строка символов переменной длины, а команда воспринимает числовое значение, указывающее **MySQL** максимальную длину, разрешенную для строки, хранящейся в этом поле. Этот тип данных очень удобен, поскольку позволяет **MySQL** планировать размер базы данных и эффективнее осуществлять просмотр и поиск данных. Но есть у него и недостаток: ***если присвоить строковое значение длиннее позволенного, оно будет усечено до максимальной длины, объявленной в определении таблицы***.

**CHAR –** в общем практически то же что и **varchar** но не может менять длину в процессе использования

**YEAR -** хранения года в таблице **classics** потому, что он поддерживает только **0000** год и диапазон лет с **1901** по 2155. **MySQL** из соображений эффективности хранит значение года в одном байте, а это значит, что храниться могут только **256** значений, в то время как книги в таблице **classics** изданы задолго до сохраняемого диапазона лет.

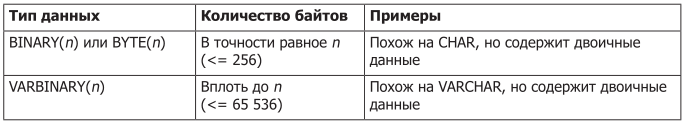
**Тип данных CHAR**

Применяется для хранения строк, заполненных байтами, имеющими связь с таблицей символов



**Тип данных BINARY**

Тип данных **BINARY** применяется для хранения строк, заполненных байтами, не имеющими никакой связи с таблицей символов. Например, тип данных BINARY можно использовать для хранения изображения в формате **GIF**.

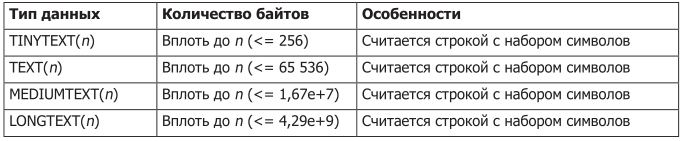


**Типы данных TEXT и VARCHAR**

Типы данных **TEXT** и **VARCHAR** имеют незначительные отличия друг от друга.

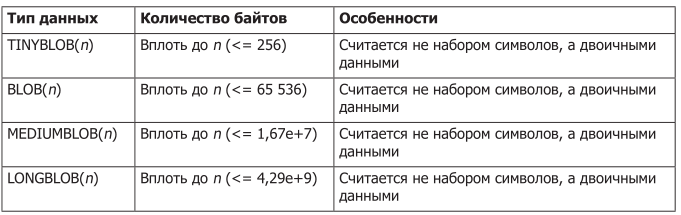
1. До выхода версии 5.0.3 MySQL удалял из полей **VARCHAR** все начальные и замыкающие пробелы.
2. В полях типа **TEXT** не может быть исходных значений.
3. В столбце **TEXT** MySQL индексирует только первые n символов (n задается при создании индекса).

Это означает, что VARCHAR является более приемлемым и быстрее обрабатываемым типом данных, если нужно вести поиск по всему содержимому поля. Если поиск никогда не будет вестись более чем в конкретном количестве начальных символов хранящегося в поле значения, то, наверное, нужно остановить свой выбор на типе данных **TEXT**



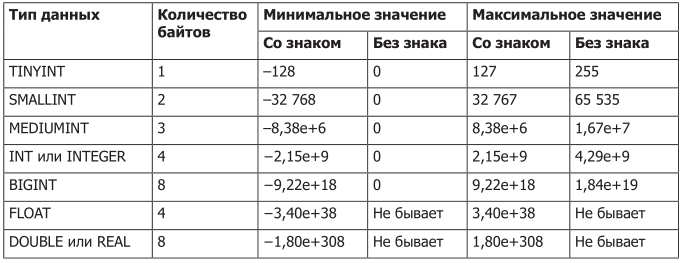
**Тип данных BLOB**

Термин **BLOB** означает **Binary Large Object** — большой двоичный объект, и поэтому, как и можно было предположить, тип данных **BLOB** больше всего подходит для хранения двоичных данных, превышающих по объему **65 536** байт. Другим основным отличием **BLOB** от типа данных BINARY является то, что для столбцов типа BLOB нельзя задавать исходные значения



**Числовые типы данных**

В **MySQL** поддерживаются различные числовые типы данных — от одиночного байта до чисел с плавающей точкой с удвоенной точностью. Хотя для числового поля можно использовать до **8** байт, лучше все же выбрать поле с самым скромным типом данных, в котором способно уместиться наибольшее из ожидаемых вами значений. Тогда ваша база данных будет небольшой по объему и быстрой по доступу.



Т.е. к примеру у нас есть тип данных TINYINT который без знака может иметь значение от 0 до 255, а со знаком от -128 до 127.

Чтобы указать, какой именно тип данных используется, со знаком или без знака, применяется спецификатор **UNSIGNED.** В следующем примере создается таблица по имени **tablename** , содержащая поле **fieldname** с типом данных **UNSIGNED INTEGER**

**CREATE TABLE tablename (fieldname INT UNSIGNED);**

При создании числового поля можно также передать в качестве параметра необязательное число:

**CREATE TABLE tablename (fieldname INT(4));**

Но при этом следует помнить, что, в отличие от типов данных **BINARY** и **CHAR** , этот параметр не показывает количество байтов, выделяемых под хранение. Может быть, это противоречит интуитивному восприятию, но на самом деле это число обозначает отображаемую ширину данных в поле при его извлечении. Оно часто используется вместе со спецификатором **ZEROFILL**:

**CREATE TABLE tablename (fieldname INT(4) ZEROFILL);** ZEROFILL –добавляет 0 если ширина числа меньше 4х знаков( 8 будет записано как 0008), если значение 4ре символа или более ZEROFILL игнорируется.

**Типы данных DATE и TIME**



Значения, имеющие типы данных **DATETIME** и **TIMESTAMP** , отображаются одинаково. Основное различие в том, что у **TIMESTAMP** слишком узкий диапазон (**от 1970 до 2037** года), а в **DATETIME** может храниться практически любая нужная дата, если только вы не интересуетесь античной историей или научной фантастикой.

Но **TIMESTAMP** также полезен, потому что, используя его, можно позволить **MySQL** установить для вас нужное значение. Если при добавлении строки не задавать значение для поля с этим типом данных, то в него автоматически будет вставлено текущее время. Можно также заставить **MySQL** обновлять столбец с типом данных TIMESTAMP при каждом изменении строки.

**Тип данных AUTO\_INCREMENT**

Иногда нужно обеспечить уникальность каждой строки, имеющейся в базе данных. В вашей программе это можно сделать за счет тщательной проверки вводимых данных и обеспечении их различия хотя бы в одном из значений в любых двух строках. Но такой подход не защищен от ошибок и работает только в конкретных обстоятельствах. Например, в таблице один и тот же автор может появляться несколько раз. Точно так же, скорее всего, будет повторяться год издания и т. д. В таком случае гарантировать отсутствие продублированных строк будет довольно трудно.

В соответствии с названием столбца, которому назначен этот тип данных, его содержимому будет устанавливаться значение на единицу большее, чем значение записи в этом же столбце в предыдущей вставленной строке. В примере показано, как нужно добавлять новый столбец по имени **id** к таблице **classics** и придавать ему свойства автоприращения:

**ALTER TABLE classics ADD id INT UNSIGNED NOT NULL AUTO\_INCREMENT KEY;**

Здесь представлена команда **ALTER** , очень похожая на команду **CREATE** . Команда **ALTER** работает с уже существующей таблицей и может добавлять, изменять или удалять столбцы. В нашем примере добавляется столбец по имени **id** , имеющий следующие характеристики.

1. **INT UNSIGNED** — делает столбец способным принять целое число, достаточно большое для того, чтобы в таблице могло храниться более 4 млрд записей.
2. **NOT NULL** — обеспечивает наличие значения в каждой записи столбца. Многие программисты используют его в поле **NULL** , чтобы показать отсутствие в нем какого-либо значения. Но тогда могут появляться дубликаты, противоречащие самому смыслу существования этого столбца. Поэтому появление в нем значения **NULL** запрещено.
3. **AUTO\_INCREMENT** — заставляет **MySQL** установить для этого столбца уникальное значение в каждой строке, как было описано ранее. Фактически мы не управляем значением, которое будет появляться в каждой строке этого столбца, но это и не нужно: все, о чем мы беспокоимся, — гарантия уникальности этого значения.
4. **KEY** — столбец с автоприращением полезно использовать в качестве ключа, поскольку вы будете стремиться искать строки на основе значений этого столбца.

Теперь каждая запись будет иметь уникальное число в столбце **id** , для первой записи это будет начальное число **1**, а счет других записей будет вестись по нарастающей. Как только будет вставлена новая строка, в ее столбец **id** будет автоматически записано следующее по порядку число.

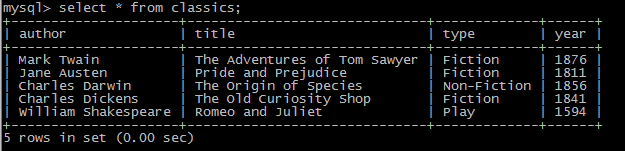
**ALTER TABLE classics DROP id; -** удалит столбец **id**

**Добавление данных в таблицу**

INSERT INTO classics(author, title, type, year) VALUES('Mark Twain','The Adventures of Tom Sawyer','Fiction','1876');

Добавляем таким способом наши значения и проверяем:

SELECT \* FROM classics;



**Команда INSERT** . Ее первая часть, **INSERT INTO classics** , сообщает **MySQL**, куда нужно вставлять следующие за ней данные. Затем в круглых скобках перечисляются четыре имени столбцов: **author** , **title** , **type** и **year** , которые отделяются друг от друга запятыми. Таким образом **MySQL** сообщается, что именно в эти четыре поля будут вставляться данные.

Во второй строке каждой команды **INSERT** содержится ключевое слово **VALUES** , за которым следуют четыре строковых значения, взятых в кавычки и отделенных друг от друга запятыми. Они обеспечивают **MySQL** теми четырьмя значениями, которые будут вставлены в четыре ранее указанных столбца.

**Переименование таблиц**

**ALTER TABLE classics RENAME pre1900;** - соответственно переименует таблицу **classics** в **pre1900**.

**Изменение типа данных столбца**

Для изменения типа данных столбца также используется команда **ALTER** , но в этом случае вместе с ней применяется ключевое слово **MODIFY** .

**ALTER TABLE classics MODIFY year SMALLINT;**

Изменит тип данных столбца **year** на **SMALLINT**(был **CHAR**).

**Добавление нового столбца**

**ALTER TABLE classics ADD pages SMALLINT UNSIGNED; -** добавит столбец pages с типом данных **SMALLINT** без знака.

**Переименование столбца**

**ALTER TABLE classics CHANGE type category VARCHAR(16);** - изменит имя столбца с **type** на **category**

Обратите внимание на добавление **VARCHAR(16)** в конце этой команды. Это связано с тем, что ключевое слово **CHANGE** требует указания типа данных даже в том случае, если вы не собираетесь его изменять, и **VARCHAR(16)** — тот самый тип данных, который был указан при создании столбца **type**

**Удаление столбца**

**ALTER TABLE classics DROP pages; -** удалит столбец **pages.**

**Нужно быть осторожным, т.к. последствия удаления необратимы , этой командой можно удалять таблицы и даже БД**

**drop table checking;** удалит таблицу **checking**

**Индексы**

**ALTER TABLE** classics **ADD INDEX**(author(20));

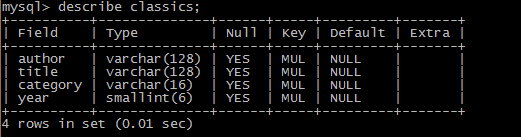
**ALTER TABLE** classics **ADD INDEX**(title(20));

**ALTER TABLE** classics ADD **INDEX**(category(4));

**ALTER TABLE** classics **ADD INDEX**(year);

Add index добавляет индексирование к каждому столбцу, в скобках указана длинна индекса(первые 20 символов, первые 4 символа, без цифры все символы)

describe classic ; выведет:



В поле Key мы видим значание MUL что значит, что в данном столбще могут встречаться одинаковые значения поскольку имена авторов могут встречаться многократно, одни и те же названия книг могут использоваться множеством авторов и т. д.

**Использование команды CREATE INDEX**

Индекс можно добавить не только командой ALTER TABLE , но и командой CREATE INDEX . Эти две команды являются равнозначными, за исключением того, что CREATE INDEX не может использоваться для создания индекса типа первичного ключа — PRIMARY KEY. Формат этой команды показан во второй строке примера :

**ALTER** **TABLE** classics **ADD** **INDEX**(author(20));

**CREATE INDEX** author **ON** classics (author(20));

**Добавление индексов при создании таблиц**

Чтобы добавить индекс, не нужно выжидать какое-то время после создания таблицы. Это может отнять много времени, поскольку добавление индекса к большой таблице — длительный процесс. Поэтому рассмотрим команду, создающую таблицу **classics** с уже имеющимися индексами.

CREATE TABLE classics (

author VARCHAR(128),

title VARCHAR(128),

category VARCHAR(16),

year SMALLINT,

**INDEX(author(20)),**

**INDEX(title(20)),**

**INDEX(category(4))**,

INDEX(year)) ENGINE MyISAM;

**Первичные ключи**

Можно вести поиск по всем имеющимся в таблице данным, но нет единого уникального ключа для каждого издания, обеспечивающего мгновенный доступ к строке. Например, что бы добавить уникальный **ISBN** для книг, можно воспользоваться **PRIMARY KEY**

Поэтому продолжим работу с таблицей и создадим новый столбец для этого ключа. Теперь, помня о том, что номер ISBN состоит из 13 символов, можно решить, что с этой задачей справится следующая команда:

**ALTER TABLE** classics **ADD** isbn **CHAR**(13) **PRIMARY KEY**;

Если запустить эту команду на выполнение, будет получено сообщение об ошибке, связанной с дубликатом записи для ключа 1: **Duplicate entry** .



Причина в том, что таблица уже заполнена данными, а эта команда пытается добавить столбец со значением NULL к каждой строке, что запрещено, поскольку все столбцы, использующие первичный ключ, должны иметь уникальное значение. Но если бы таблица была пуста, то эта команда была бы выполнена без проблем, как и при добавлении первичного ключа сразу же после создания таблицы.

То есть мы добавляем столбец isbn к таблице, если быть точнее каждая строка будет содержать столбец с автором, типом, названием, годом выпуска и соответственно isbn книги. Команда добавляет primary key для столбца isbn, но т.к. есть уже записи(строки в таблице) со значениями а мы добавляем ещё столбец для всех строк и в этом столбце нет ни одного значения(т.е. значение NULL) а ключ праймари запрещает повторяющиеся записи в столбце мы и получаем ошибку.

В сложившейся ситуации нужно немного схитрить: создать новый столбец без индекса, заполнить его данными, а затем добавить индекс ретроспективно, воспользоваться командой из примера.

Учтите, что в этом примере применяются ключевые слова **UPDATE** и **WHERE** , которые более подробно будут рассмотрены в подразделе «Создание запросов к базе данных MySQL» далее.

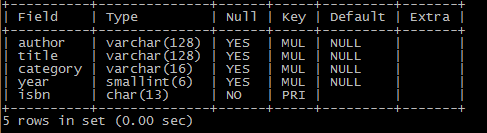
**ALTER TABLE** classics **ADD** isbn **CHAR**(13);

**UPDATE** classics **SET** isbn='9781598184891' **WHERE** year='1876';

… и т.д. …

**ALTER TABLE** classics **ADD** **PRIMARY KEY**(isbn);

**DESCRIBE** classics;



В строке **isbn** I столбце Key стоит PRI, что означает, что здесь не может быть одинаковых значений.

**Создание индекса FULLTEXT**

В отличие от обычного индекса имеющийся в **MySQL** индекс **FULLTEXT** позволяет осуществлять сверхбыстрый поиск целых столбцов текста. Он сохраняет каждое слово каждой строки данных в специальном индексе, в котором можно вести поиск, используя «естественный язык» наподобие того, что применяется в поисковом механизме.

Рассмотрим некоторые особенности индексов **FULLTEXT** , о которых нужно знать.

1. До версии MySql 5.6 работал только с таблицами типа MyISAM, Если нужно привести таблицу к типу MyISAM , можно применить команду MySQL: **ALTER TABLE** tablename **ENGINE = MyISAM**;
2. Индексы **FULLTEXT** могут создаваться только для столбцов с типами данных **CHAR** , **VARCHAR** и **TEXT**
3. Определение индекса **FULLTEXT** может быть дано в инструкции **CREATE TABLE** при создании таблицы или добавлено позже с использованием инструкции **ALTER TABLE** (или **CREATE INDEX** ).
4. ***Намного*** быстрее будет загрузить большие наборы данных в таблицу, не имеющую индекса **FULLTEXT** , а затем создать индекс, чем загружать их в таблицу, у которой уже имеется индекс **FULLTEXT**

**ALTER TABLE** classics **ADD FULLTEXT**(author,title);

**Создание запросов к базе данных MySQL**

Итак, мы создали базу данных MySQL и таблицы, заполнили их данными и добавили к ним индексы, чтобы ускорить поиск.

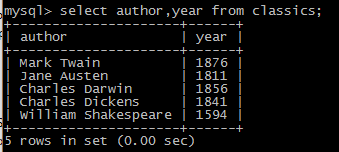
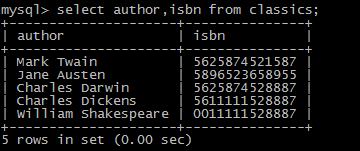
**SELECT**

**SELECT** что-нибудь **FROM** имя\_таблицы;

Этим что-нибудь, как вы уже видели, может быть символ звездочки ( \* ), означающий «каждый столбец», вместо него можно указать какие-нибудь конкретные столбцы.

**SELECT** author,year **FROM** classics;

**SELECT** title,isbn **FROM** classics;



**SELECT COUNT**

Другой заменой параметра что-нибудь является функция COUNT , которая может быть использована множеством способов. В примере она отображает количество строк в таблице за счет передачи ей в качестве параметра символа звездочки ( **\*** ), означающего «все строки». В соответствии с вашими ожиданиями будет возвращено число 5, поскольку в таблицу внесены сведения о пяти книгах.

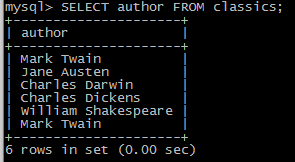
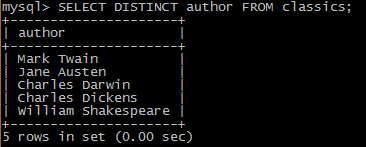
**SELECT COUNT**(\*) **FROM** classics;

**SELECT DISTINCT**

Этот спецификатор (и его синоним **DISTINCTROW** ) позволяет исключать множество записей, имеющих одинаковые данные. Предположим, к примеру, что вам нужно получить список всех авторов, фигурирующих в таблице. Если просто выбрать столбец **author** из таблицы, содержащей несколько книг одного и того же автора, то будет отображен длинный список с одинаковыми именами авторов, повторяющимися снова и снова. Но за счет добавления ключевого слова **DISTINCT** можно показать каждого автора всего лишь один раз.

**SELECT** author **FROM** classics;

**SELECT DISTINCT** author **FROM** classics;



**DELETE**

Когда нужно удалить строку из таблицы, применяется команда **DELETE** . Ее синтаксис похож на синтаксис команды **SELECT** , он позволяет сузить диапазон удаляемой информации до конкретной строки или строк путем использования таких спецификаторов, как **WHERE** и **LIMIT**

**DELETE FROM** classics **WHERE** title='About Me'; - удалит все строки в которых столбец title содержит запись 'About Me'

**WHERE**

Ключевое слово **WHERE** позволяет сузить диапазон действия запроса, возвращая только те данные, в отношении которых конкретное выражение возвращает истинное значение.

**DELETE FROM** classics **WHERE** title='About Me'; - точный запрос

**SELECT** author**,**title **FROM** classics **WHERE** title **LIKE** "%harle%"; - поиск по маске использую **LIKE**

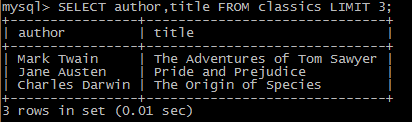
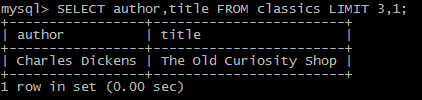
**LIMIT**

Спецификатор **LIMIT** позволяет выбрать количество выводимых в запросе строк и место, с которого таблица начнет их возвращать. Когда передается один параметр, он указывает **MySQL** начать действие спецификатора с верхней части результатов и вернуть только то количество строк, которое задано этим параметром. Если передать спецификатору два параметра, то первый укажет смещение относительно начала результатов, которое **MySQL** должна учесть при их отображении, а второй укажет, сколько строк нужно вывести.

**ПРИМЕР**. Первая возвращает первые три строки из таблицы. Вторая возвращает две строки, начиная с позиции 1 (пропуская первую строку т.к. счет идет с 0). А последняя возвращает одну строку, начинающуюся с позиции 3:

**SELECT** author,title **FROM** classics **LIMIT** 3;

**SELECT** author,title **FROM** classics **LIMIT** 3,1;



**MATCH...AGAINST**

Конструкция ***MATCH...AGAINST*** может быть применена к столбцу, для которого был создан индекс **FULLTEXT**. Используя эту конструкцию, можно вести поиск, применяя в качестве критерия элементы обычного языка, как при работе с поисковыми механизмами Интернета. В отличие от конструкций **WHERE...=** или **WHERE...LIKE** конструкция **MATCH...AGAINST** позволяет вводить в поисковый запрос несколько слов и проверять на их наличие все слова в столбцах, имеющих индекс **FULLTEXT** . Индексы **FULLTEXT** нечувствительны к регистру букв, поэтому неважно, какой именно регистр используется в ваших запросах.

**SELECT** author,title **FROM** classics **WHERE** **MATCH**(author,title) **AGAINST**('curiosity shop');

запрос требует вернуть любые строки, содержащие в любом месте(author,title) и в любом порядке оба слова: **curiosity** и **shop** . НЕ ЗАБЫВАЕМ ПРО СТОП СЛОВА ТИПА The , and и т.д.

**MATCH...AGAINST...IN BOOLEAN MODE**

Булев режим позволяет также ставить впереди искомых слов знак + или – , чтобы показать, что они должны быть включены или исключены. Если обычный булев режим требует «искать присутствие любого из этих слов», то знак «плюс» означает, что «это слово обязательно должно присутствовать, иначе строку возвращать не нужно». Знак «минус» означает, что «этого слова быть не должно, а если оно присутствует, то строку возвращать не нужно».

**SELECT** author,title **FROM** classics **WHERE** **MATCH**(author,title) **AGAINST**('+charles -species' **IN BOOLEAN MODE**);

**SELECT** author,title **FROM** classics **WHERE** **MATCH**(author,title) **AGAINST**('"origin of"' **IN BOOLEAN MODE**);

Первый запрос требует вернуть все строки, в которых содержится слово **charles** и нет слова **species** . Во втором запросе используются двойные кавычки, чтобы потребовать вернуть все строки, включающие в себя фразу **origin of**

двойные кавычки отменяют учет стоповых слов.

**UPDATE...SET**

Эта конструкция позволяет обновлять содержимое поля. Если нужно изменить содержимое одного или нескольких полей, сначала следует сузить область действия запроса до того поля или полей, которые будут подвергаться изменениям, практически тем же способом, который применялся в команде **SELECT** .

**UPDATE** classics **SET** author='Mark Twain (Samuel Langhorne Clemens)' **WHERE** author='Mark Twain';

**UPDATE** classics **SET** category='Classic Fiction' **WHERE** category='Fiction';

В первом запросе, действие которого затрагивает только одну строку, к литературному псевдониму Mark Twain добавляется настоящее имя писателя — Samuel Langhorne Clemens , заключенное в скобки. А вот второй запрос воздействует на три столбца, поскольку он заменяет все появления слова Fiction в столбце **category** термином Classic Fiction

**ORDER BY**

Спецификатор **ORDER BY** позволяет отсортировать возвращаемые результаты по одному или нескольким столбцам в возрастающем или в убывающем порядке.

**SELECT** author,title **FROM** classics **ORDER BY** author;

**SELECT** author,title **FROM** classics **ORDER BY** title DESC;

Первый запрос возвращает издания, отсортированные по авторам в возрастающем алфавитном порядке (этот режим используется по умолчанию), а второй возвращает их отсортированными по названию в убывающем порядке.

**SELECT** author,title,year **FROM** classics **ORDER** **BY** author,year **DESC**;

Ключевое слово **DESC** применяется только к столбцу, который указан перед ним, — year .

**GROUP BY**

Точно так же, как и при использовании **ORDER BY** , можно сгруппировать результаты, возвращаемые запросом, с помощью спецификатора **GROUP BY** , который больше всего подходит для извлечения информации о группе данных. Например, если нужно узнать, сколько изданий каждой категории присутствует в таблице **classics** , можно ввести запрос:

SELECT category,COUNT(author) FROM classics GROUP BY category;

**Объединение таблиц**

Управление несколькими таблицами, содержащими различные виды информации в одной базе данных, считается вполне обычным делом. Рассмотрим, к примеру, таблицу клиентов — customers , для которой нужно обеспечить возможность использования перекрестных ссылок с приобретенными ими книгами из таблицы **classics**

CREATE TABLE customers (name VARCHAR(128), isbn char(4)) ENGINE MyISAM; - создаем таблицу customers;

Существует также быстрый способ для вставки сразу нескольких строк данных, как в примере, позволяющий заменить два(или более) отдельных запроса INSERT INTO одним

**INSERT** **INTO** customers(name,isbn) **VALUES** ('Joe Bloggs','9780099533474'), ('Mary Smith','9780582506206');

При создании новой таблицы следует обратить внимание на то, что у нее есть кое-что общее с таблицей **classics**: столбец под названием **isbn** . Поскольку его предназначение в обеих таблицах совпадает (**ISBN** всегда является ссылкой на одну и ту же книгу), этот столбец можно использовать для связывания двух таблиц вместе в едином запросе

**SELECT** name,author,title **FROM** customers,classics **WHERE** customers.isbn=classics.isbn;

**NATURAL JOIN**

Используя **NATURAL JOIN** , можно сократить количество вводимого текста и сделать запросы немного более понятными. В этом виде объединения участвуют две таблицы, в которых ***автоматически объединяются столбцы с одинаковыми именами***. Для получения тех же результатов, что и в примере , можно ввести следующий запрос:

**SELECT** name,author,title **FROM** customers **NATURAL JOIN** classics;

**Использование ключевого слова AS**

Можно сократить количество вводимого текста и улучшить читаемость запроса за счет создания псевдонимов с помощью ключевого слова **AS** . После имени таблицы нужно поставить **AS** , а затем используемый псевдоним. Следующий код идентичен по своей работе коду **NATURAL JOIN**

SELECT name,author,title FROM customers AS cust, classics AS class WHERE cust.isbn=class.isbn;

**Использование логических операторов**

Для дальнейшего сужения пространства выбора в запросах **MySQL**, использующих ключевое слово **WHERE** , можно также задействовать логические операторы **AND** , **OR** и **NOT** . В примере показаны варианты применения каждого из них, но их можно использовать в любых сочетаниях.

**SELECT** author,title **FROM** classics **WHERE** author **LIKE** "Charles%" **AND** author **LIKE** "%Darwin";

**SELECT** author,title **FROM** classics **WHERE** author **LIKE** "%Mark Twain%" **OR** author **LIKE** "%Samuel Langhorne%";

**SELECT** author,title **FROM** classics **WHERE** author **LIKE** "Charles%" **AND** author **NOT** **LIKE** "%Darwin";

Первый возвращает сведения о книгах, для которых значение столбца author начинается с Charles и заканчивается Darwin . Второй запрос ищет книги, принадлежащие перу Марка Твена, используя для этого либо литературный псевдоним — Mark Twain , либо настоящее имя писателя — Samuel Langhorne Clemens . Третий запрос возвращает книги с авторами, чье имя Charles , а фамилия не Darwin

**Функции MySQL**