*Лекция 10: Базы данных и СУБД. Введение в SQL*

### **Базы данных: основные понятия**

В жизни мы часто сталкиваемся с необходимостью хранить какую-либо информацию, а потому часто имеем дело и с *базами данных*. Например, мы используем записную книжку для хранения номеров телефонов своих друзей и планирования своего времени. Телефонная книга содержит информацию о людях, живущих в одном городе. Все это своего рода *базы данных*. Ну а раз это *базы данных*, то посмотрим, как в них хранятся данные. Например, телефонная книга представляет собой таблицу ([табл. 10.1](https://www.intuit.ru/studies/courses/42/42/lecture/27193?page=1#table.10.1)).

В этой таблице данные – это собственно номера телефонов, адреса и ФИО., т.е. строки "Иванов Иван Иванович", "32-43-12" и т.п., а названия столбцов этой таблицы, т.е. строки "ФИО", "Номер телефона" и "*Адрес*" задают смысл этих данных, их семантику.

Таблица 10.1. *Пример базы данных: телефонная книга*

|  |  |  |
| --- | --- | --- |
| **ФИО** | **Номер телефона** | **Адрес** |
| Иванов Иван Иванович | 32-43-12 | ул. Ленина, 12, 43 |
| Ильин Федор Иванович | 32-32-34 | пр. Маркса, 32, 45 |

Теперь представьте, что записей в этой таблице не две, а две тысячи, вы занимаетесь созданием этого справочника и где-то произошла ошибка (например, опечатка в адресе). Видимо, тяжеловато будет найти и исправить эту ошибку вручную. Нужно воспользоваться какими-то *средствами автоматизации*. Для управления большим количеством данных программисты (не без помощи математиков) придумали системы управления *базами данных* ( *СУБД* ). *По* сравнению с текстовыми *базами данных* электронные *СУБД*имеют огромное число преимуществ, от возможности быстрого поиска информации, взаимосвязи данных между собой до использования этих данных в различных прикладных программах и одновременного доступа к данным нескольких пользователей.

Для точности дадим *определение* *базы данных*, предлагаемое *Глоссарий*.ру

**База данных** – это совокупность связанных данных, организованных по определенным правилам, предусматривающим общие принципы описания, хранения и манипулирования, независимая от прикладных программ. *База данных* является *информационной моделью предметной области*. Обращение к *базам данных* осуществляется с помощью системы управления *базами данных* ( **СУБД** ). *СУБД* обеспечивает поддержку создания *баз данных*, *централизованного управления* и организации доступа к ним различных пользователей.

Итак, мы пришли к выводу, что хранить данные независимо от программ, так, что они связаны между собой и организованы *по*определенным правилам, целесообразно. Но вопрос, как хранить данные, *по* каким правилам они должны быть организованы, остался открытым. Способов существует множество (кстати, называются они моделями представления или хранения данных). Наиболее популярные – объектная и *реляционная* модели данных.

Автором **реляционной модели** считается Э. Кодд, который первым предложил использовать для обработки данных аппарат теории множеств (*объединение*, *пересечение*, *разность*, *декартово произведение*) и показал, что любое *представление* данных сводится к совокупности двумерных таблиц особого вида, известного в математике как *отношение*.

Таким образом, реляционная *база данных* представляет собой набор таблиц (точно таких же, как приведенная выше), связанных между собой. Строка в таблице соответствует сущности реального мира (в приведенном выше примере это *информация* о человеке).

**Примеры реляционных*****СУБД*:*****MySql*, PostgreSql.**

В основу **объектной модели** положена концепция объектно-ориентированного программирования, в которой данные представляются в виде набора объектов и классов, связанных между собой родственными отношениями, а работа с объектами осуществляется с помощью скрытых (инкапсулированных) в них методов.

**Примеры объектных*****СУБД*:*****Cache*, GemStone (от Servio Corporation), ONTOS (ONTOS).**

В последнее время производители *СУБД* стремятся соединить два этих подхода и проповедуют объектно-реляционную модель представления данных. Примеры таких *СУБД* – *IBM* *DB2* for Common Servers, Oracle8.

Поскольку мы собираемся работать с *Mysql*, то будем обсуждать аспекты работы только с реляционными *базами данных*. Нам осталось рассмотреть еще два важных понятия из этой области: *ключи* и *индексирование*, после чего мы сможем приступить к изучению языка запросов *SQL*.

#### **Ключи**

Для начала давайте подумаем над таким вопросом: какую информацию нужно дать о человеке, чтобы собеседник точно сказал, что это именно тот человек, сомнений быть не может, второго такого нет? Сообщить фамилию, очевидно, недостаточно, поскольку существуют однофамильцы. Если собеседник человек, то мы можем приблизительно объяснить, о ком речь, например вспомнить поступок, который совершил тот человек, или еще как-то. Компьютер же такого объяснения не поймет, ему нужны четкие правила, как определить, о ком идет речь. В системах управления *базами данных* для решения такой задачи ввели понятие *первичного ключа*.

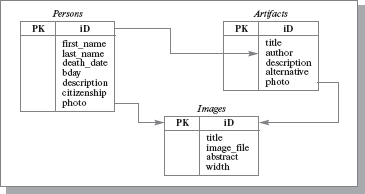
***Первичный ключ*** (primary key, PK) – минимальный набор полей, уникально идентифицирующий запись в таблице. Значит, *первичный ключ* – это в первую очередь набор полей таблицы, во-вторых, каждый набор значений этих полей должен определять единственную запись (строку) в таблице и, в-третьих, этот набор полей должен быть минимальным из всех обладающих таким же свойством. Поскольку *первичный ключ* определяет только одну уникальную запись, то никакие две записи таблицы не могут иметь одинаковых значений *первичного ключа* .

Например, в нашей таблице ([см. выше](https://www.intuit.ru/studies/courses/42/42/lecture/27193?page=1#table.10.1)) ФИО и адрес позволяют однозначно выделить запись о человеке. Если же говорить в общем, без связи с решаемой задачей, то такие знания не позволяют точно указать на единственного человека, поскольку существуют однофамильцы, живущие в разных городах по одному адресу. Все дело в границах, которые мы сами себе задаем. Если считаем, что знания ФИО, телефона и адреса без указания города для наших целей достаточно, то все замечательно, тогда поля ФИО и адрес могут образовывать *первичный ключ*. В любом случае проблема создания *первичного ключа* ложится на плечи того, кто проектирует *базу данных* (разрабатывает *структуру хранения* данных). Решением этой проблемы может стать либо выделение характеристик, которые естественным образом определяют запись в таблице (задание так называемого логического, или естественного, PK), либо создание дополнительного поля, предназначенного именно для однозначной идентификации записей в таблице (задание так называемого суррогатного, или искусственного, PK). Примером логического *первичного ключа* является номер паспорта в *базе данных* о паспортных данных жителей или ФИО и адрес в телефонной книге (таблица выше). Для задания суррогатного *первичного ключа* в нашу таблицу можно добавить поле id (идентификатор), значением которого будет целое число, уникальное для каждой строки таблицы. Использование таких суррогатных *ключей* имеет смысл, если естественный *первичный ключ* представляет собой большой набор полей или его выделение нетривиально.

Кроме однозначной идентификации записи, *первичные ключи* используются для *организации связей* с другими таблицами.

Например, у нас есть три таблицы: содержащая информацию об исторических личностях (Persons), содержащая информацию об их изобретениях (*Artifacts*) и содержащая изображения как личностей, так и артефактов (Images) ([рис 10.1](https://www.intuit.ru/studies/courses/42/42/lecture/27193?page=1#image.10.1)).

*Первичным ключом* во всех этих таблицах является поле id (идентификатор). В таблице *Artifacts* есть поле author, в котором записан идентификатор, присвоенный автору изобретения в таблице Persons. Каждое значение этого поля является **внешним ключом** для *первичного ключа* таблицы Persons. Кроме того, в таблицах Persons и *Artifacts* есть поле photo, которое ссылается на изображение в таблице Images. Эти поля также являются *внешними ключами* для *первичного ключа* таблицы Images и устанавливают однозначную логическую связь Persons-Images и *Artifacts*-Images. То есть если значение *внешнего ключа* photo в таблице личности равно 10, то это значит, что фотография этой личности имеет id=10 в таблице изображений. Таким образом, **внешние ключи** используются для *организации связей* между таблицами *базы данных* (родительскими и дочерними) и для поддержания ограничений *ссылочной целостности* данных.



**Рис. 10.1.**Пример использования первичных ключей для организации связей с другими таблицами

#### **Индексирование**

Одна из основных задач, возникающих при работе с *базами данных*, – это задача поиска. При этом, поскольку информации в *базе данных*, как правило, содержится много, перед программистами встает задача не просто поиска, а эффективного поиска, т.е. поиска за сравнительно небольшое время и с достаточной точностью. Для этого (для оптимизации производительности запросов) производят **индексирование** некоторых полей таблицы. Использовать индексы полезно для быстрого поиска строк с указанным значением одного столбца. Без индекса чтение таблицы осуществляется по всей таблице, начиная с первой записи, пока не будут найдены соответствующие строки. Чем больше таблица, тем больше накладные расходы. Если же таблица содержит индекс по рассматриваемым столбцам, то *база данных* может быстро определить позицию для поиска в середине файла данных без просмотра всех данных. Это происходит потому, что *база данных* помещает проиндексированные поля поближе в памяти, так, чтобы можно было побыстрее найти их значения. Для таблицы, содержащей 1000 строк, это будет как минимум в 100 раз быстрее по сравнению с *последовательным перебором* всех записей. Однако в случае, когда необходим доступ почти ко всем 1000 строкам, быстрее будет последовательное чтение, так как при этом не требуется операций поиска по диску. Так что иногда индексы бывают только помехой. Например, если копируется большой объем данных в таблицу, то лучше не иметь никаких индексов. Однако в некоторых случаях требуется задействовать сразу несколько индексов (например, для обработки запросов к часто используемым таблицам).

Если говорить о *MySQL*, то там существует три вида индексов: *PRIMARY* , *UNIQUE* , и *INDEX* , а слово ключ ( KEY ) используется как синоним слова индекс ( *INDEX* ). Все индексы хранятся в памяти в виде *B-деревьев*.

***PRIMARY*** – *уникальный индекс* (ключ) с ограничением, что все индексированные им поля не могут иметь пустого значения (т.е. они NOT NULL ). Таблица может иметь только один первичный индекс, но он может состоять из нескольких полей.

***UNIQUE*** – *ключ (индекс*), задающий поля, которые могут иметь только уникальные значения.

***INDEX*** – обычный индекс (как мы описали выше). В *MySqL*, кроме того, можно индексировать строковые поля по заданному числу символов от начала строки.

### **СУБД MySQL**

Продолжим разговор о *СУБД* *MySQL*. ***MySQL*** – это реляционная система управления *базами данных*. То есть данные в ее базах хранятся в виде логически связанных между собой таблиц, доступ к которым осуществляется с помощью языка запросов *SQL* . *MySQL* – свободно распространяемая система, т.е. платить за ее применение не нужно. Кроме того, это достаточно быстрая, надежная и, главное, простая в использовании *СУБД*, вполне подходящая для не слишком глобальных проектов.

Работать с *MySQL* можно не только в текстовом режиме, но и в графическом. Существует очень популярный визуальный интерфейс (кстати, написанный на PHP) для работы с этой *СУБД*. Называется он ***PhpMyAdmin***. Этот интерфейс позволяет значительно упростить работу с *базами данных* в *MySQL* .

В текстовом режиме работа с *базой данных* выглядит просто как ввод команд в командную строку ([рис 10.2](https://www.intuit.ru/studies/courses/42/42/lecture/27193?page=2#image.10.2)), а результаты выборок возвращаются в виде своеобразных таблиц, поля в которых налезают друг на друга, если данные не помещаются на экран ([рис 10.3](https://www.intuit.ru/studies/courses/42/42/lecture/27193?page=2#image.10.3)).

Работа с MySQL в командной строке. Команда show databases — вывести все имеющиеся базы данных

**Рис. 10.2.**Работа с MySQL в командной строке. Команда show databases — вывести все имеющиеся базы данных

*PhpMyAdmin* позволяет пользоваться всеми достоинствами браузера, включая прокрутку изображения, если оно не умещается на экран. Многие из базовых *SQL* -функций работы с данными в *PhpMyAdmin* сведены к интуитивно понятным интерфейсам и действиям, напоминающим переход *по* ссылкам в *Internet*. Но, тем не менее, стоит все же поработать и в текстовом режиме.



**Рис. 10.3.**Работа с MySQL в командной строке. Результат обработки команды show databases

Перед тем как переходить к детальному изучению языка *SQL*, несколько слов об установке *MySQL* и подготовке к работе. Если вы не собираетесь заниматься администрированием сервера, то *информация*, приведенная ниже, пригодится вам только для общего развития. Итак, устанавливается *MySQL* очень просто – автоматически, пару раз нажмите OK, и все. После этого вы можете зайти в директорию, где лежат файлы типа *mysql*.exe, mysqld.exe и т.п. (у нас под *Windows* XP это c:\*mysql*\bin ) Последний *файл*запускает *Mysql*-*сервер*. В некоторых системах *сервер* запускается в виде сервиса. После запуска сервера следует запустить *mysql*-клиент, запустив программу *mysql*.exe. Здесь даже пароля не спросят. Более того, если вы наберете

shell> mysql.exe -u root

или

shell>mysql -u root mysql

то получите все *права* администратора *mysql* сервера. Кстати, выполнять эти команды надо, находясь в той директории, где лежат файлы *mysql*.exe.

Для начала, не вдаваясь в подробности команд, исправим эти два недочета (отсутствие пароля у администратора и возможность входа анонимным пользователям):

shell> mysql -u root mysql

mysql> UPDATE user SET Password=PASSWORD('new\_password')

WHERE user='root';

mysql> DELETE FROM user WHERE user='';

mysql> FLUSH PRIVILEGES;

Все данные о пользователях *MySQL* хранит в таблице user в специальной *базе данных* *mysql*, *доступ* к которой имеет только *администратор* сервера. Поэтому, чтобы изменить какой-либо *пароль*, нужно изменить эту таблицу. *Пароль* задается с помощью функции PASSWORD, которая кодирует введенные данные. Кроме изменения пароля администратора, нужно еще удалить всех пользователей, не имеющих логина (*команда* *DELETE* ). *Команда* *Flush* *Privileges* заставляет вступить в действие изменения, произошедшие в системной *базе данных* (*mysql*).

Теперь создадим *базу данных*, с которой будем работать (мы все еще работаем как *администратор* сервера):

mysql>create database book;

Как можно заметить, все команды в *MySQL* заканчиваются точкой с запятой. Если вы забыли поставить этот знак, то выдается приглашение его поставить до тех пор, пока это не будет сделано:

mysql> show tables

->

->

Теперь последнее действие – создадим простого пользователя, предоставим ему *доступ* к созданной *базе данных*, и начнем работать.

mysql> GRANT ALL PRIVILEGES ON book.\* TO nina@localhost

IDENTIFIED BY '123';

*Команда* *GRANT* наделяет пользователя nina, зашедшего на *сервер* с этой же машины (c *localhost*) и идентифицируемого паролем "123", определенными правами (в данном случае всеми) на все таблицы *базы данных* book. Теперь мы можем выйти и зайти как *пользователь* nina с соответствующим паролем:

shell>mysql -u nina -p

Enter password: \*\*\*

Welcome to the MySQL monitor!...

mysql>

Если вы собираетесь пользоваться *базой данных* на чужом сервере, то его *администратор* проделает все описанные выше действия за вас, т.е. все настроит и создаст пользователя и *базу данных*. В следующей главе описаны команды языка *SQL*, которые пригодятся для работы с данными, хранящимися в *СУБД* *MySQL*.

### **Язык SQL**

Итак, мы в общих чертах познакомились с основными понятиями теории *баз данных*, установили и настроили для работы *MySQL*. Теперь самое время научиться манипулировать данными, хранящимися в *базах данных*. Для этого нам понадобится ***SQL*** – *структурированный язык* запросов. Этот язык дает возможность создавать, редактировать и удалять информацию, хранящуюся в *базах данных*, создавать новые *базы данных* и многое другое. *SQL* является стандартом ANSI (Американский национальный институт стандартов) и ISO (Международная организация по стандартизации).

#### **Немного истории**

Первый международный стандарт языка *SQL* был принят в 1989 г., его часто называют *SQL/89*. Среди недостатков этого стандарта выделяют в первую очередь то, что многие важные свойства он устанавливал как определяемые в реализации. Отсюда произошло множество расхождений в *реализациях языка* разными производителями. Кроме того, высказывались претензии по поводу отсутствия в этом стандарте упоминаний о практических аспектах языка, таких как его встраивание в язык программирования Си.

Следующий международный стандарт языка *SQL* был принят в конце 1992 г. И стал называться *SQL/92*. Он получился гораздо более точным и полным, чем *SQL/89*, хотя и не был лишен недостатков. В настоящее время большинство систем почти полностью реализуют этот стандарт. Однако, как известно, прогресс не остановишь, и в 1999 году появился новый стандарт *SQL*:1999, также известный как *SQL3*. *SQL3* характеризуется как "объектно-ориентированный *SQL* " и является основой нескольких объектно-реляционных систем управления *базами данных* (например, ORACLE8 *компании Oracle*, Universal Server *компании Informix* и *DB2* Universal Database *компании IBM*). Этот стандарт является не просто слиянием *SQL-92* и объектной технологии. Он содержит ряд расширений традиционного *SQL*, а сам документ составлен таким образом, чтобы добиться более эффективной работы в области стандартизации в будущем.

Если говорить о *MySQL*, то она соответствует начальному уровню SQL92, содержит несколько расширений этого стандарта и стремится к полной поддержке стандарта ANSI SQL99, но без ущерба для скорости и качества кода.

Далее, говоря об основах языка *SQL*, будем придерживаться его реализации в *СУБД* *MySQL*.

#### **Основные операторы языка SQL**

Функции любой *СУБД* включают:

1. создание, удаление, изменение *базы данных* (БД);
2. добавление, изменение, удаление, назначение прав пользователя;
3. внесение, удаление и изменение данных в БД (таблиц и записей);
4. выборку данных из БД.

К первым двум функциям имеют доступ только администраторы *СУБД* или привилегированные пользователи. Рассмотрим, как решаются последние две задачи (на самом деле это семь задач).

Прежде чем что-либо делать с данными, нужно создать таблицы, в которых эти данные будут храниться, научиться изменять структуру этих таблиц и удалять их, если потребуется. Для этого в языке *SQL* существуют операторы *CREATE TABLE* , *ALTER TABLE* и *DROP TABLE* .

##### **Оператор CREATE TABLE**

Оператор ***CREATE TABLE*** создает таблицу с заданным именем в текущей *базе данных* . Правила для допустимых имен таблицы приведены в документации. Если нет активной текущей *базы данных* или указанная таблица уже существует, то возникает ошибка выполнения команды.

В версии *MySQL* 3.22 и более поздних *имя таблицы* может быть указано как имя\_базы\_данных.имя\_таблицы. Эта форма записи работает независимо от того, является ли указанная *база данных* текущей.

В версии *MySQL* 3.23 при создании таблицы можно использовать ключевое слово *TEMPORARY*. Временная таблица автоматически удаляется по завершении соединения, а ее имя действительно только в течение данного соединения. Это означает, что в двух разных соединениях могут использоваться временные таблицы с одинаковыми именами без конфликта друг с другом или с существующей таблицей с тем же именем (существующая таблица скрыта, пока не удалена временная таблица). В версии *MySQL* 4.0.2 для создания временных таблиц необходимо иметь привилегии CREATE *TEMPORARY* TABLES.

В версии *MySQL* 3.23 и более поздних можно использовать ключевые слова IF NOT EXISTS для того, чтобы не возникала ошибка, если указанная таблица уже существует. Следует учитывать, что при этом идентичность структур этих таблиц не проверяется.

Каждая таблица представлена набором определенных файлов в директории *базы данных*.

**Синтаксис**

CREATE [TEMPORARY] TABLE [IF NOT EXISTS]

имя\_таблицы [(определение\_столбца,...)]

[опции\_таблицы] [select\_выражение]

В выражении определение\_столбца перечисляют, какие столбцы должны быть созданы в таблице. Каждый столбец таблицы может быть пустым ( NULL ), иметь значение по умолчанию, являться ключом или автоинкрементом. Кроме того, для каждого столбца обязательно указывается тип данных, которые будут в нем храниться. Если не указывается ни NULL, ни NOT NULL, то столбец интерпретируется так, как будто указано NULL. Если поле помечают как автоинкремент ( AUTO\_INCREMENT ), то его значение автоматически увеличивается на единицу каждый раз, когда происходит добавление данных в таблицу и в это поле записывается пустое значение ( NULL, т.е. ничего не записывается) или 0. Автоинкремент в таблице может быть только один, и при этом он обязательно должен быть проиндексирован. Последовательность AUTO\_INCREMENT начинается с 1. Наличие автоинкремента является одной из особенностей *MySQL*. Формально описание столбца (определение\_столбца) выглядит так:

имя\_столбца тип [NOT NULL | NULL]

[DEFAULT значение\_по\_умолчанию]

[AUTO\_INCREMENT][PRIMARY KEY]

[reference\_definition]

Тип столбца (тип в выражении определение\_столбца) может быть одним из следующих:

* целый: INT[(length)] [*UNSIGNED*] [ZEROFILL]
* действительный: REAL[(length,decimals)] [*UNSIGNED*] [ZEROFILL]
* символьный: CHAR(length) [BINARY] и VARCHAR(length) [BINARY]
* дата и время: DATE и TIME
* для работы с большими объектами: *BLOB*
* текстовый: TEXT
* перечислимое множество: *ENUM*(value1,value2,value3,...) и SET(value1,value2,value3,...)

Полный список типов смотрите в документации *MySQL*.

Вместо перечисления столбцов и их свойств в определении\_столбца можно задавать списки ключевых и индексных полей, ограничения и проверки:

PRIMARY KEY (имя\_индексируемого\_столбца, ...)

или

KEY [имя\_индекса] (имя\_индексируемого\_столбца,...)

или

INDEX [имя\_индекса] (имя\_индексируемого\_столбца,...)

или

UNIQUE [INDEX] [имя\_индекса]

(имя\_индексируемого\_столбца,...)

или

FULLTEXT [INDEX] [имя\_индекса]

(имя\_индексируемого\_столбца,...)

или

[CONSTRAINT symbol]

FOREIGN KEY [имя\_индекса]

(имя\_индексируемого\_столбца,...)

[reference\_definition]

или

CHECK (expr)

При задании всех этих элементов указывается список полей (столбцов), которые будут входить в индекс, ключ или ограничение, имя\_индексируемого\_столбца записывается следующим образом:

имя\_столбца [(длина\_индекса)]

*FOREIGN* KEY, CHECK и REFERENCES на самом деле ничего не делают в *MySQL*. Они добавлены только для совместимости с другими SQL-серверами. Поэтому на них мы останавливаться не будем.

Кроме всего перечисленного, при создании таблицы можно указать некоторые ее свойства (опции\_таблицы), например такие:

* тип таблицы: TYPE = {BDB | *HEAP* | *ISAM* | *InnoDB* | MERGE | MRG\_MYISAM | *MYISAM* }
* начальное значение счетчика автоинкремента: AUTO\_INCREMENT = число
* средняя длина строк в таблице: AVG\_ROW\_LENGTH = число
* комментарии к таблице (строка из 60 символов): COMMENT = "строка"
* максимальное и минимальное предполагаемое число строк: MAX\_ROWS = число и MIN\_ROWS = число

И последний (опять же опциональный) элемент команды CREATE – это выражение *SELECT* (select\_выражение). Синтаксис такой:

[IGNORE | REPLACE] SELECT ...

(любое корректное выражение SELECT)

Если при создании таблицы в команде CREATE указывается выражение *SELECT* , то все поля, полученные выборкой, добавляются в создаваемую таблицу.

**Пример 10.1**. Создадим таблицу Persons, структура которой была приведена на [рисунке 10.1](https://www.intuit.ru/studies/courses/42/42/lecture/27193?page=1#image.10.1).

mysql>CREATE TABLE Persons

(id INT PRIMARY KEY AUTO\_INCREMENT,

first\_name VARCHAR(50), last\_name

VARCHAR(100), death\_date INT,

description TEXT, photo INT,

citizenship CHAR(50) DEFAULT 'Russia');

Пример 10.1. Создание таблицы Persons

С помощью специфичной для *MySql* команды SHOW можно просмотреть существующие *базы данных*, таблицы в *базе данных* и поля в таблице.

Показать все *базы данных*:

mysql>SHOW databases;

Сделать текущей *базу данных* book и показать все таблицы в ней:

mysql>use book;

mysql>show tables;

Показать все столбцы в таблице Persons:

mysql> show columns from Persons;

##### **Оператор DROP TABLE**

Оператор ***DROP TABLE*** удаляет одну или несколько таблиц. Все табличные данные и определения удаляются, так что при работе с этой командой следует соблюдать осторожность.

Синтаксис:

DROP TABLE [IF EXISTS] имя\_таблицы

[, имя\_таблицы,...]

[RESTRICT | CASCADE]

В версии *MySQL* 3.22 и более поздних можно использовать ключевые слова IF EXISTS, чтобы предупредить ошибку, если указанные таблицы не существуют.

Опции *RESTRICT* и CASCADE позволяют упростить перенос программы с других *СУБД*. В данный момент они не задействованы.

mysql> DROP TABLE IF EXISTS Persons,

Artifacts, test;

Пример 10.2. Использование оператора DROP TABLE

##### **Оператор ALTER TABLE**

Оператор ***ALTER TABLE*** обеспечивает возможность изменять структуру существующей таблицы. Например, можно добавлять или удалять столбцы, создавать или уничтожать индексы или переименовывать столбцы либо саму таблицу. Можно также изменять комментарий для таблицы и ее тип.

Синтаксис:

ALTER [IGNORE] TABLE имя\_таблицы

alter\_specification

[, alter\_specification ...]

Можно производить следующие изменения в таблице (все они записываются в alter\_specification ):

* добавление поля:
* ADD [COLUMN] определение\_столбца

[FIRST | AFTER имя\_столбца ]

или

ADD [COLUMN] (определение\_столбца,

определение\_столбца,...)

Здесь, как и далее, определение\_столбца записывается так же, как при создании таблицы.

* добавление индексов:

ADD *INDEX* [имя\_индекса] (имя\_индексируемого\_столбца,...) или ADD *PRIMARY* KEY (имя\_индексируемого\_столбца,...)или ADD *UNIQUE* [имя\_индекса] (имя\_индексируемого\_столбца,...) или ADD FULLTEXT [имя\_индекса] (имя\_индексируемого\_столбца,...)

* изменение поля:

*ALTER* [COLUMN] имя\_столбца {*SET DEFAULT* *literal* | DROP DEFAULT} или CHANGE [COLUMN] старое\_имя\_столбца определение\_столбца или MODIFY [COLUMN] определение\_столбца

* удаление поля, индекса, ключа:
* DROP [COLUMN] имя\_столбца
* DROP PRIMARY KEY

DROP INDEX имя\_индекса

* переименование таблицы:

RENAME [TO] новое\_имя\_таблицы

* переупорядочение полей таблицы:

ORDER BY поле

или

опции\_таблицы

Если оператор *ALTER TABLE* используется для изменения определения типа столбца, но *DESCRIBE* имя\_таблицы показывает, что столбец не изменился, то, возможно, *MySQL* игнорирует данную модификацию по одной из причин, описанных в специальном разделе документации. Например, при попытке изменить столбец VARCHAR на CHAR *MySQL* будет продолжать использовать VARCHAR, если данная таблица содержит другие столбцы с переменной длиной.

Оператор *ALTER TABLE* во время работы создает временную копию исходной таблицы. Требуемое изменение выполняется на копии, затем исходная таблица удаляется, а новая переименовывается. Это делается для того, чтобы в новую таблицу автоматически попадали все обновления, кроме неудавшихся. Во время выполнения *ALTER TABLE* исходная таблица доступна для чтения другими клиентами. Операции обновления и записи в этой таблице приостанавливаются, пока не будет готова новая таблица. Следует отметить, что при использовании любой другой опции для *ALTER* TABLE, кроме RENAME, *MySQL* всегда будет создавать временную таблицу, даже если данные, строго говоря, и не нуждаются в копировании (например, при изменении имени столбца).

**Пример10.3**. Добавим в созданную таблицу Persons поле для записи года рождения человека:

mysql> ALTER TABLE Persons

ADD bday INTEGER AFTER last\_name;

Пример 10.3. Добавление в таблицу Persons поля для записи года рождения человека

Итак, мы научились работать с таблицами: создавать, удалять и изменять их. Теперь разберемся, как делать то же самое с данными, которые в этих таблицах хранятся.

##### **Оператор SELECT**

Оператор ***SELECT*** применяется для извлечения строк, выбранных из одной или нескольких таблиц. То есть с его помощью мы задаем столбцы или выражения, которые надо извлечь ( select\_выражения ), таблицы ( table\_references ), из которых должна производиться выборка, и, возможно, условие ( where\_definition ), которому должны соответствовать данные в этих столбцах, и порядок, в котором эти данные нужно выдать.

Кроме того, оператор *SELECT* можно использовать для извлечения строк, вычисленных без ссылки на какую-либо таблицу. Например, чтобы вычислить, чему равно 2\*2, нужно просто написать

mysql> SELECT 2\*2;

Упрощенно структуру оператора *SELECT* можно представить следующим образом:

SELECT select\_выражение1, select\_выражение2,

...

[FROM table\_references

[WHERE where\_definition]

[ORDER BY {число | имя\_столбца |

формула}

[ASC | DESC], ...]]

Квадратные скобки [ ] означают, что использование находящегося в них оператора необязательно, вертикальная черта | означает перечисление возможных вариантов. После ключевого слова ORDER BY указывают имя столбца, число (целое беззнаковое) или формулу и способ упорядочения (по возрастанию – *ASC*, или по убыванию – *DESC* ). По умолчанию используется упорядочение по возрастанию.

Когда в select\_выражении мы пишем " \* ", это значит выбрать все столбцы. Кроме " \* " в select\_выражения могут использоваться функции типа max, min и *avg*.

**Пример 10.4**. Выбрать из таблицы Persons все данные, для которых поле first\_name имеет значение 'Александр':

mysql> SELECT \* FROM Persons

WHERE first\_name='Александр';

Пример 10.4. Использование оператора SELECT

Выбрать название и описание ( title, description ) артефакта под номером 10:

mysql> SELECT title,description

FROM Artifacts WHERE id=10;

##### 

##### **Оператор INSERT**

Оператор ***INSERT*** вставляет новые строки в существующую таблицу. Оператор имеет несколько форм. Параметр имя\_таблицы во всех этих формах задает таблицу, в которую должны быть внесены строки. Столбцы, для которых задаются значения, указываются в списке имен столбцов ( имя\_столбца ) или в части SET.

Синтаксис:

1. INSERT [LOW\_PRIORITY | DELAYED] [IGNORE]
2. [INTO] имя\_таблицы [(имя\_столбца,...)]
3. VALUES (выражение,...),(...),...

Эта форма команды *INSERT* вставляет строки в соответствии с точно указанными в команде значениями. В скобках после имени таблицы перечисляются столбцы, а после ключевого слова VALUES – их значения.

Например:

mysql> INSERT INTO Persons

(last\_name, bday) VALUES

('Иванов', '1934');

вставит в таблицу Persons строку, в которой значения фамилии ( last\_name ) и даты рождения ( bday ) будут заданы соответственно как "Иванов" и "1934".

1. INSERT [LOW\_PRIORITY | DELAYED] [IGNORE]
2. [INTO] имя\_таблицы [(имя\_столбца,...)]
3. SELECT ...

Эта форма команды *INSERT* вставляет строки, выбранные из другой таблицы или таблиц.

Например:

mysql> INSERT INTO Artifacts (author)

SELECT id FROM Persons

WHERE last\_name='Иванов'

AND bday='1934';

вставит в таблицу *Artifacts* в поле "автор" ( author ) значение идентификатора, выбранного из таблицы Persons по условию, что фамилия человека Иванов и год рождения 1934.

1. INSERT [LOW\_PRIORITY | DELAYED] [IGNORE]
2. [INTO] имя\_таблицы
3. SET имя\_столбца=выражение,
4. имя\_столбца=выражение, ...

Например:

mysql> INSERT INTO Persons

SET last\_name='Петров',

first\_name='Иван';

Эта команда вставит в таблицу Persons в поле last\_name значение "Петров", а в поле first\_name – строку "Иван".

Форма *INSERT.*.. VALUES со списком из нескольких значений поддерживается в версии *MySQL* 3.22.5 и более поздних. Синтаксис выражения имя\_столбца=выражение поддерживается в версии *MySQL* 3.22.10 и более поздних.

Действуют следующие соглашения.

* Если не указан список столбцов для *INSERT.*.. VALUES или *INSERT.*.. *SELECT* , то величины для всех столбцов должны быть определены в списке VALUES() или в результате работы *SELECT* . Если порядок столбцов в таблице неизвестен, для его получения можно использовать *DESCRIBE* имя\_таблицы.
* Любой столбец, для которого явно не указано значение, будет установлен в свое значение по умолчанию. Например, если в заданном списке столбцов не указаны все столбцы в данной таблице, то не упомянутые столбцы устанавливаются в свои значения по умолчанию.
* Выражение expression может относиться к любому столбцу, который ранее был внесен в список значений. Например, можно указать следующее:
* mysql> INSERT INTO имя\_таблицы (col1,col2)

VALUES(15,col1\*2);

Но нельзя указать:

mysql> INSERT INTO имя\_таблицы (col1,col2)

VALUES(col2\*2,15);

Мы еще не обсудили три необязательных параметра, присутствующих во всех трех формах команды: LOW\_PRIORITY, *DELAYED* и IGNORE.

Параметры LOW\_PRIORITY и *DELAYED* используются, когда с таблицей работает большое число пользователей. Они предписывают устанавливать приоритет данной операции перед операциями других пользователей. Если указывается ключевое слово LOW\_PRIORITY, то выполнение данной команды *INSERT* будет задержано до тех пор, пока другие клиенты не завершат чтение этой таблицы. В этом случае клиент должен ожидать, пока данная команда вставки не будет завершена, что в случае интенсивного использования таблицы может потребовать значительного времени. В противоположность этому команда *INSERT* *DELAYED* позволяет данному клиенту продолжать операцию сразу же, независимо от других пользователей.

Если в команде *INSERT* указывается ключевое слово IGNORE, то все строки, имеющие дублирующиеся ключи *PRIMARY* или *UNIQUE* в этой таблице, будут проигнорированы и не внесены в таблицу. Если не указывать IGNORE, то данная операция вставки прекращается при обнаружении строки, имеющей дублирующееся значение существующего ключа.

##### **Оператор UPDATE**

Синтаксис:

UPDATE [LOW\_PRIORITY] [IGNORE] имя\_таблицы

SET имя\_столбца1=выражение1

[, имя\_столбца2=выражение2, ...]

[WHERE where\_definition]

[LIMIT число]

Оператор ***UPDATE*** обновляет значения существующих столбцов таблицы в соответствии с введенными значениями. В выражении SET указывается, какие именно столбцы следует модифицировать и какие величины должны быть в них установлены. В выражении WHERE, если оно присутствует, задается, какие строки подлежат обновлению. В остальных случаях обновляются все строки. Если задано выражение ORDER BY, то строки будут обновляться в указанном в нем порядке.

Если указывается ключевое слово LOW\_PRIORITY, то выполнение данной команды *UPDATE* задерживается до тех пор, пока другие клиенты не завершат чтение этой таблицы.

Если указывается ключевое слово IGNORE, то команда обновления не будет прервана, даже если возникнет ошибка дублирования ключей. Строки, из-за которых возникают конфликтные ситуации, обновлены не будут.

Если в выражении, которое задает новое значение столбца, используется имя этого поля, то команда *UPDATE* использует для этого столбца его текущее значение. Например, следующая команда устанавливает столбец death\_date в значение, на единицу большее его текущей величины:

mysql> UPDATE Persons

SET death\_date=death\_date+1;

В версии *MySQL* 3.23 можно использовать параметр LIMIT #, чтобы убедиться, что было изменено только заданное количество строк.

Например, такая операция заменит в первой строке нашей таблицы экспонатов название title на строку "Ламповая ЭВМ":

mysql> UPDATE Artifacts

SET title='Ламповая ЭВМ' Limit 1;

##### 

##### **Оператор DELETE**

Оператор ***DELETE*** удаляет из таблицы имя\_таблицы строки, удовлетворяющие заданным в where\_definition условиям, и возвращает число удаленных записей.

Если оператор *DELETE* запускается без определения WHERE, то удаляются все строки.

Синтаксис:

DELETE [LOW\_PRIORITY] FROM имя\_таблицы

[WHERE where\_definition]

[LIMIT rows]

Например, следующая команда удалит из таблицы Persons все записи, у которых поле "год рождения" ( bday ) больше 2003:

mysql> DELETE FROM Persons WHERE bday>2003;

Удалить все записи в таблице можно еще и с помощью такой команды:

mysql> DELETE FROM Persons WHERE 1>0;

Но этот метод работает гораздо медленнее, чем использование той же команды без условия:

mysql> DELETE FROM Persons;

Специфическая для *MySQL* опция LIMIT для команды *DELETE* указывает серверу максимальное количество строк, которые следует удалить до возврата управления клиенту. Эта опция может использоваться для гарантии того, что данная команда *DELETE* не потребует слишком много времени для выполнения.

### ***Заключение***

Итак, мы разобрались с основами реляционных *баз данных*, научились создавать простые и не очень SQL-запросы. Надеюсь, что большое количество технических деталей не помешало читателям получить *представление* о базовых элементах языка, поскольку все это наверняка пригодится для решения практических задач.