А. В. Коротков

Методические указания к выполнению лабораторных заданий по курсу «Базы данных».

Оглавление

Введение

Данное пособие предназначено для изучения реляционных баз данных.

Используемое программное обеспечение:

1) СУБД MySQL;
2) MySQL Workbench;
3) Eclipse с плагином ERModeler;
4) HTTP-сервер Apache2.

Вся работа должна выполняться в настроенном рабочем окружении по инструкции с репозитория на GitHub — https://github.com/AVKorotkov/setupenv. Не следует выполнять работы в данных методических указаниях в другой программной среде (например, используя ОС MS Windows), поскольку многие программные компоненты, рассматриваемые в данном пособии, могут работать там по-другому, отсутствуют некоторые возможности (например, нет истории команд в программе MySQL monitor в версии для Windows), другое расположение файлов настроек (что, в общем, понятно, поскольку логическая структура файловой системы в ОС от Microsoft совершенно другая) и/или другое их название, могут иметь специфические особенности работы (или не работать вообще); кроме того, могут возникнуть трудности даже с установкой некоторых программных компонентов. Инструкция из репозитория описывает процесс настройки рабочей среды в дистрибутиве Debian GNU/Linux 7 (Wheezy), который устанавливается в Oracle Virtual Box. Можно, следуя данной инструкции (с небольшими изменениями), установить нужное программное обеспечение и на «реальное железо». В других дистрибутивах GNU/Linux работа всего рассматриваемого программного комплекса тоже возможна, хотя и не гарантируется.

Далее везде предполагается, что пользователю dbuser предоставлены права на работу с базами данных, имена которых начинаются на db (это соответствует тем настройкам, которые присутствуют в скриптах из репозитория по ссылке выше; имя пользователя, его пароль, а также пароль суперпользователя root генерируются автоматически на этапе выполнения скриптов настройки и сообщаются пользователю; также эти данные сохраняются в пользовательском файле настроек MySQL — $HOME/.my.cnf).

Глава 1

Основы работы с базами данных.

В состав пакета MySQL client входит программа MySQL monitor. Это приложение с текстовым интерфейсом, может работать в интерактивном или неинтерактивном режиме. Для начала работы с данной программой в интерактивном режиме следует выполнить в окне любого виртуального терминала (XTerm, Gnome Terminal, XFCE Terminal, Konsole и т. п.) команду:

```
user@debian:~$ mysql -h localhost -P 3306 -u dbuser -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 58
Server version: 5.5.41-0+wheezy1 (Debian)

Copyright (c) 2000, 2014, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Здесь имя программы — mysql, которой могут быть переданы некоторые опции и в качестве параметра имя БД. В примере выше имя БД передано не было, поэтому перед началом работы далее необходимо будет выбрать некоторую БД из имеющихся или создать новую. Опции, переданные в примере выше, имеют следующий смысл:

- -h localhost: имя хоста (или).
- -P 3306: номер порта.
- -u dbuser: имя пользователя.
- -p: пароль пользователя.

Это не все доступные в MySQL monitor опции, их довольно много, познакомиться с имеющимися в распоряжении пользователя опциями данной программы можно в руководстве (при помощи команды man mysql) или другой официальной документации. В данном примере часть или даже все опции могут быть опущены:

- Если соединение происходит с сервером, располагающимся на том же хосте, что и сервер (т. е., по адресу localhost), и если не менять настройки сервера MySQL по умолчанию, то данную опцию можно опустить.
- Если номер порта не изменён в настройках сервера MySQL, то его можно не указывать.
- При настройках по умолчанию MySQL monitor предполагает, что имя пользователя на сервере MySQL совпадает с именем его в системе. Если учётная запись с таким именем действительно есть на сервере MySQL, и требуется инициировать соединение именно от

имени данного пользователя, то в таком случае эту опцию можно опустить. Следует отметить, что:

- на сервере MySQL не обязательно должна присутствовать учётная запись с таким именем;
- даже если она имеется, то не всегда нужно инициировать соединение от имени такого пользователя, а требуется воспользоваться иной учётной записью на сервере MySQL.

Также данная опция может быть опущена, если имя учётной записи, для которой нужно инициировать соединение, указано в настройках.

- Пароль можно либо указать прямо в команде сразу после названия опции -p (без пробела), но так делать не стоит из соображений безопасности. Обычной практикой является просто указание опции -p без пароля, при этом MySQL monitor запрашивает пароль перед началом работы:

```
user@debian:$ mysql -p
Enter password:
```

Его следует ввести с клавиатуры (вводимые символы при этом не отображаются на экране опять-таки по соображениям безопасности). Пароль может быть опущен в двух случаях:

1) Учётная запись пользователя сервера MySQL создана без пароля (иначе говоря, он пустой). Такая практика по соображениям безопасности является плохой.
2) Пароль указан в настройках пользователя.

Таким образом, в простейшем случае начать работу с MySQL Monitor можно вызовом программы без указания опций и параметров:

```
user@debian:~$ mysql
```

Традиционно в UNIX-подобных системах используются два варианта написания опций: полный и сокращённый. Для первого варианта характерно следующее написание их:

```
$ имя_программы --опция1[=значение1] --опция2[=значение2]...
```

Во втором варианте используется написание вида:

```
$ имя_программы -оп1 [значение1] -оп2 [значение2]...
```

В обоих вариантах наличие или отсутствие значений для тех или иных опций зависит от конкретной программы.

Сказанное справедливо и в отношении MySQL monitor. Для запуска этой программы можно использовать длинную форму записи опций:

```
user@debian:~$ mysql --host=localhost --port=3306 --user=dbuser --password
```

История введённых ранее в интерактивном режиме в MySQL monitor команд сохраняется в файле $HOME/.mysql_history (имя и расположение этого файла можно изменить, задав нужное значение переменной окружения MYSQL_HISTFILE).

При работе в программе в интерактивном режиме ввод всех возможных команд осуществляется после получения приглашения, которое по умолчанию имеет вид mysql> (что, при желании, можно изменить; кроме того, при многострочном вводе команд, приглашение, начиная со второй строки, принимает вид ->). В MySQL monitor можно вводить команды, осуществляющие SQL-запросы, а также специальные (собственные) команды программы. К числу последних относится и команда выхода из программы, которая может быть вызвана любым из следующих способов:

1. mysql> exit
   Bye
2. mysql> quit
   Bye
3. mysql> \q
   Bye
4. mysql> Bye

Последний вариант требует небольшого комментария: здесь использовано клавиатурное сочетание <Ctrl>+<d>, которое на экране никак не отображается.

Для получения справки по работе в приложении следует ввести команду получения помощи любым из следующих способов:

1. mysql> help
2. mysql> \h
3. mysql> \?
4. mysql> ?

Для получения справки не о самой программе MySQL monitor, а о сервере MySQL, можно ввести команду help contents, а затем выбрать последовательно нужную категорию, подкатегорию и т. д.:

```
mysql> help contents
You asked for help about help category: "Contents"
For more information, type 'help <item>', where <item> is one of the following
categories:
   Account Management
   Administration
   Compound Statements
   Data Definition
   Data Manipulation
   Data Types
   Functions
   Functions and Modifiers for Use with GROUP BY
   Geographic Features
   Help Metadata
   Language Structure
   Plugins
   Procedures
   Storage Engines
```

```
   Table Maintenance
   Transactions
   User-Defined Functions
   Utility

mysql> help Account Management
You asked for help about help category: "Account Management"
For more information, type 'help <item>', where <item> is one of the following
topics:
   CREATE USER
   DROP USER
   GRANT
   RENAME USER
   REVOKE
   SET PASSWORD

mysql> help DROP USER
Name: 'DROP USER'
Description:
Syntax:
DROP USER user [, user] ...

The DROP USER statement removes one or more MySQL accounts and their
privileges. It removes privilege rows for the account from all grant
tables. An error occurs for accounts that do not exist. To use this
statement, you must have the global CREATE USER privilege or the DELETE
privilege for the mysql database.

Each account name uses the format described in
http://dev.mysql.com/doc/refman/5.5/en/account-names.html. For example:

DROP USER 'jeffrey'@'localhost';

If you specify only the user name part of the account name, a host name
part of '%' is used.

URL: http://dev.mysql.com/doc/refman/5.5/en/drop-user.html
```

Запросы, передаваемые MySQL monitor на выполнение серверу, должны завершаться символом «;». В качестве альтернативы возможно использование вместо «;» сочетаний «\g» либо «\G» (последнее отличается от первых двух видом вывода, о чём будет сказано ниже). Ввод собственных команд MySQL monitor не требует завершения их указанными символами, нужно просто нажимать клавишу <Enter>.

Каким образом можно посмотреть список баз данных, имеющихся на даном сервере? Для этого можно воспользоваться запросом SHOW DATABASES:

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| moodle             |
```

```
| mysql              |
| performance_schema |
+--------------------+
4 rows in set (0.08 sec)
```

Под результатом запроса, который выводится MySQL monitor, можно увидеть также информационное сообщение: количество полученных строк и время обработки запроса.

Создадим новую базу данных, используя запрос CREATE DATABASE:

```
mysql> CREATE DATABASE dbtest;
Query OK, 1 row affected (0.00 sec)
```

В данном примере запроса dbtest — это имя базы данных, которая должна быть создана. Получаем сообщение, что запрос обработан успешно, изменения коснулись одной записи (Query OK, 1 row affected).

Посмотрим на получившийся результат:

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| dbtest             |
| moodle             |
| mysql              |
| performance_schema |
+--------------------+
5 rows in set (0.00 sec)
```

В списке баз данных обнаруживается только что созданная (нетрудно заметить, что количество полученных строк увеличилось на единицу).

Удаляется база данных с помощью запроса DROP DATABASE:

```
mysql> DROP DATABASE dbtest;
Query OK, 0 rows affected (0.01 sec)
```

Вновь проверим список имеющихся баз данных:

```
mysql> Show daTabases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| moodle             |
| mysql              |
| performance_schema |
+--------------------+
4 rows in set (0.00 sec)
```

Из этого примера нетрудно заметить, что ключевые слова в запросе являются регистронезависимыми. Это верно не только для данного запроса, но и для любых других SQL-запросов в MySQL monitor.

Попробуем создать простую базу данных и заполнить её некоторыми данными. Предположим, мы создаём форум.

```
mysql> CREATE DATABASE dbforum;
Query OK, 1 row affected (0.00 sec)
```

Как уже ранее было отмечено, выбрать базу данных, с которой предполагается работать, можно ещё на этапе вызова MySQL monitor, передав в качестве параметра имя базы данных. Но в любой момент можно сменить текущую базу данных на любую из имеющихся. Для этого служит специальная внутренняя команда MySQL monitor:

```
mysql> USE dbforum;
Database changed
```

Можно воспользоваться другой формой данной команды. Как и для любой другой внутренней команды MySQL monitor, наличие символа «;» в конце команды не является обязательным:

```
mysql> \u dbforum
Database changed
```

Имена баз данных (как и имена и псевдонимы таблиц) являются регистрозависимыми, что показывает следующий пример:

```
mysql> \u dbForum
ERROR 1049 (42000): Unknown database 'dbForum'
```

Попытка ввыполнить команду заканчивается ошибкой, поскольку базы данных с таким именем не существует. Причина этого проста: в UNIX-подобных системах имена файлов являются регистрозависимыми.

Посмотрим, какие таблицы имеются в нашей базе (очевидно, никаких нет, поскольку база данных только что создана):

```
mysql> SHOW TABLES;
Empty set (0.00 sec)
```

Если есть необходимость посмотреть, какие таблицы есть не в текущей базе данных, а в какой-то другой, можно с помощью указания в явном виде имени базы данных:

```
mysql> SHOW TABLES IN mysql;
+---------------------------+
| Tables_in_mysql           |
+---------------------------+
| columns_priv              |
| db                        |
| event                     |
| func                      |
| general_log               |
| help_category             |
| help_keyword              |
| help_relation             |
| help_topic                |
| host                      |
| ndb_binlog_index          |
| plugin                    |
| proc                      |
```

```
| procs_priv                |
| proxies_priv              |
| servers                   |
| slow_log                  |
| tables_priv               |
| time_zone                 |
| time_zone_leap_second     |
| time_zone_name            |
| time_zone_transition      |
| time_zone_transition_type |
| user                      |
+---------------------------+
24 rows in set (0.00 sec)
```

На любом форуме имеются пользователи, которые при регистрации обычно заполняют некоторые данные. Создадим таблицу, которая будет содержать данные о пользователях форума. У каждого пользователя есть имя (логин) и пароль. В качестве ключа можно выбрать либо имя (обычной практикой является требование уникальности имени пользователя форума) либо ввести специальный атрибут — `id` (идентификатор):

```
mysql> CREATE TABLE user
    -> (
    -> id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    -> name VARCHAR(30) NOT NULL,
    -> password VARCHAR(30) NOT NULL,
    -> PRIMARY KEY(id)
    -> );
Query OK, 0 rows affected (0.02 sec)
```

Некоторые пояснения к данному запросу. У каждого поля должен быть определённый тип данных. В рассматриваемой таблице поле `id` имеет тип `INT UNSIGNED`, т. е., неотрицательное целое число, поле `name` (имя) — `VARCHAR(30)`, т. е., строка переменной длины до 30 символов, такой же тип имеет поле `password` (пароль). Помимо типа данных, для полей указаны спецификации: `NOT NULL` означает, что поле не может быть пустым (значение `NULL` имеет специальный смысл: пустое значение или, иными словами, его отсутствие; соответственно, `NOT NULL` означает, что пустым оно быть не может). `AUTO_INCREMENT` указывает на то, что значение этого атрибута автоматически увеличивается (на единицу) при каждом последующем добавлении записей в таблицу, что позволяет добиться уникальности значений такого идентификатора, `PRIMARY KEY` — что соответствующий атрибут является первичным ключом.

Этот запрос можно было сформировать немного по-другому, указав `PRIMARY KEY` в совокупности спецификаций для атрибута `id` (а не отдельно, как было сделано в примере выше), результат был бы аналогичным:

```
mysql> CREATE TABLE user
    -> (
    -> id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    -> name VARCHAR(30) NOT NULL,
    -> password VARCHAR(30) NOT NULL
    -> );
```

Теперь в нашей базе имеется одна таблица:

```
mysql> SHOW TABLES;
+------------------+
| Tables_in_dbforum |
+------------------+
| user             |
+------------------+
1 row in set (0.00 sec)
```

Она только что была создана, в ней отсутствуют записи. Рассмотрим, с помощью какого запроса можно их добавить:

```
mysql> INSERT INTO user VALUES
    -> (NULL,'admin','admpass');
Query OK, 1 row affected (0.02 sec)
```

Здесь была добавлена одна запись, запрос успешно обработан (`Query OK, 1 row affected`). Формат запроса:

`INSERT INTO user VALUES (val11,val12,...),(val21,val22,...),...`

Есть альтернативы данному варианту, они будут ниже рассмотрены.

Каким образом можно теперь выбрать из таблицы уже добавленные в неё данные? Для этого используется запрос `SELECT`. Предположим, мы хотим получить данные всех столбцов во всех записях этой таблицы. Тогда формат запроса совсем простой:

```
mysql> SELECT * FROM user;
+----+-------+----------+
| id | name  | password |
+----+-------+----------+
|  1 | admin | admpass  |
+----+-------+----------+
1 row in set (0.00 sec)
```

Шаблон `*` означает, что мы хотим получить значения всех столбцов, после ключевого слова `FROM` указывается имя таблицы.

Добавим сразу пару записей в нашу таблицу:

```
mysql> INSERT INTO user VALUES
    -> (NULL,'dummy','topsecret'),
    -> (NULL,'hacker','sesam');
Query OK, 2 rows affected (0.01 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

Посмотрим вновь результат выборки из таблицы (здесь использовано альтернативное завершение строки запроса символом «\g»):

```
mysql> SELECT * FROM user\g
+----+--------+-----------+
| id | name   | password  |
+----+--------+-----------+
|  1 | admin  | admpass   |
|  2 | dummy  | topsecret |
|  3 | hacker | sesam     |
+----+--------+-----------+
3 rows in set (0.00 sec)
```

В любое время можно посмотреть структуру таблицы (какие в ней есть поля, каких типов, какие у них имеются дополнительные спецификации):

```
mysql> DESCRIBE user;
+----------+------------------+------+-----+---------+----------------+
| Field    | Type             | Null | Key | Default | Extra          |
+----------+------------------+------+-----+---------+----------------+
| id       | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| name     | varchar(30)      | NO   |     | NULL    |                |
| password | varchar(30)      | NO   |     | NULL    |                |
+----------+------------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)
```

Здесь `Field` — имя поля; `Type` — его тип; `NULL` — указание на то, может ли быть данное поле пустым (если стоит `YES` — то да, если `NO` — нет); `Key` — является ли поле ключом; `Default` — значение по умолчанию (можно указывать в спецификациях — в таких случаях при добавлении записи в таблицу, если значение поля не указывается, ему автоматически присваивается значение по умолчанию; при создании таблицы мы не стали ни для одного поля указывать значение по умолчанию, в таких случаях им становится `NULL`); `Extra` — дополнительные спецификации.

Вернёмся немного назад, к запросам, с помощью которых в таблицу были добавлены записи. В спецификациях поля `id` присутствует `NOT NULL`, а при написании запроса в качестве значения указано `NULL`. Почему же такой запрос был успешно обработан, и, более того, значение поля `id` не пустое (пустое значение и не могло быть добавлено в силу спецификации `NOT NULL`), а равно единице? В спецификациях этого поля имеется `AUTO_INCREMENT`, что обеспечивает автоматическое присваивание очередного значения (начиная с единицы; на каждом последующем шаге значение увеличивается на единицу).

Можно получить описание отдельного поля в таблице:

```
mysql> DESCRIBE user 'name';
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| name  | varchar(30) | NO   |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
1 row in set (0.00 sec)
```

Каким образом можно сделать выборку данных не из всех полей в таблице, а только из интересующих? Например, нас интересуют только имена пользователей. Для составления такого запроса следует указать после ключевого слова `SELECT` имена нужных полей:

```
mysql> SELECT name FROM user;
+--------+
| name   |
+--------+
| admin  |
| dummy  |
| hacker |
+--------+
3 rows in set (0.00 sec)
```

Можно изменить форматирование результата запроса (его заголовок):

```
mysql> SELECT name AS Имя FROM user;
+--------+
| Имя    |
+--------+
| admin  |
| dummy  |
| hacker |
+--------+
3 rows in set (0.00 sec)
```

Применим указанный в последнем запросе подход ко всем полям в таблице user:

```
mysql> SELECT id AS Номер, name AS Имя, password AS Пароль FROM user;
+------------+--------+--------------+
| Номер      | Имя    | Пароль       |
+------------+--------+--------------+
|          1 | admin  | admpass      |
|          2 | dummy  | topsecret    |
|          3 | hacker | sesam        |
+------------+--------+--------------+
3 rows in set (0.00 sec)
```

Как уже было сказано ранее, можно изменить формат вывода результата запроса, завершив строку запроса сочетанием «\G»:

```
mysql> SELECT * FROM user\G
*************************** 1. row ***************************
      id: 1
    name: admin
password: admpass
*************************** 2. row ***************************
      id: 2
    name: dummy
password: topsecret
*************************** 3. row ***************************
      id: 3
    name: hacker
password: sesam
3 rows in set (0.00 sec)
```

Наиболее полезным такой формат может оказаться в тех случаях, когда полей в таблице достаточно много, и в табличном выводе результат не умещается по ширине экрана.

Каким образом можно сделать добавление записей в базу данных более эффективным? Можно заранее подготовить в любом текстовом редакторе SQL-скрипт, в котором разместить любое количество запросов и вызвать его на выполнение с помощью MySQL monitor либо прямо во время работы с этой программой в интерактивном режиме, либо пережав такой скрипт программе с помощью перенаправления ввода — в последнем случае после обработки скрипта программа завершит работу.

Вначале рассмотрим первый способ. Воспользуемся двумя внутренними командами: «\!» и «\.». Первая команда позволяет выполнять любые программы, не покидая MySQL monitor. Вызовем редактор nano, в котором напишем SQL-скрипт, а потом вызовем его на исполнение.

```
mysql> \! nano insert1.sql
```

В редакторе набираем текст скрипта:

```
INSERT INTO user VALUES (NULL,'badguy','bgpass');
```

Сохраняем файл (ему присваивается имя `insert1.sql` и сохраняется он в текущем каталоге, т. е., том, откуда был вызван `MySQL monitor`; каталог можно выбрать для сохранения и другой, указав нужный путь к файлу перед выполнением данной команды).

При помощи второй команды передаём созданный скрипт на выполнение `MySQL monitor`:

```
mysql> \. insert1.sql
Query OK, 1 row affected (0.01 sec)
```

Вновь сделаем выборку всех записей из нашей таблицы:

```
mysql> SELECT * FROM user;
+----+--------+-----------+
| id | name   | password  |
+----+--------+-----------+
|  1 | admin  | admpass   |
|  2 | dummy  | topsecret |
|  3 | hacker | sesam     |
|  4 | badguy | bgpass    |
+----+--------+-----------+
4 rows in set (0.00 sec)
```

Количество записей увеличилось на одну — ту, что мы добавили только что.

Завершим работу с `MySQL monitor`, чтобы рассмотреть второй способ передачи SQL-скрипта на выполнение:

```
mysql> \q
Bye
```

Вызовем снова редактор `nano`:

```
desktop:$ nano insert2.sql
```

и создадим в нём скрипт следующего содержания:

```
USE dbforum
INSERT INTO user VALUES (NULL,'goodguy','ggpass');
```

Сохраняем файл и выходим из редактора.

Запустим `MySQL monitor` в неинтерактивном режиме, воспользовавшись стандартным для UNIX-подобных систем перенаправлением ввода для передачи на исполнение только что написанного скрипта. Поскольку в настройках MySQL, как было отмечено во введении, имя пользователя и его пароль сохранены, то можно никаких опций при вызове на выполнение не передавать:

```
desktop:$ mysql < insert2.sql
desktop:$
```

`MySQL monitor` обрабатывает скрипт и завершает работу.

Вновь запустим его на выполнение в интерактивном режиме, указав имя базы данных при вызове, тогда она после запуска сразу станет текущей:

```
desktop:$ mysql dbforum
```

Снова делаем выборку всех записей из нашей таблицы:

```
mysql> SELECT * FROM user;
+----+---------+-----------+
| id | name    | password  |
+----+---------+-----------+
|  1 | admin   | admpass   |
|  2 | dummy   | topsecret |
|  3 | hacker  | sesam     |
|  4 | badguy  | bgpass    |
|  5 | goodguy | ggpass    |
+----+---------+-----------+
5 rows in set (0.00 sec)
```

И вновь количество записей увеличилось на одну — вставленную из скрипта в неинтерактивном режиме.

Что делать в тех случаях, когда по тем или иным причинам оказывается, что ранее сделанная таблица нас перестаёт устраивать? Например, нужно добавить какой-то новый столбец? Добавим в таблицу `user` столбец `sex`, в котором будем хранить пол пользователей форума. Какой тип данных можно для этого использовать? В отличие от, например, имени, пол может принимать только два значения: «мужской» или «женский». Для хранения данных, значения которых могут быть только из некоторого конечного списка, можно использовать тип «перечислимый». Структуру таблицы можно изменить при помощи запроса `ALTER TABLE`:

```
mysql> ALTER TABLE user ADD sex ENUM('M','F') NOT NULL;
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

Синтаксис вполне прозрачный, в каких-то комментариях не нуждается. Новый столбец в данном случае становится последним столбцом таблицы.

Проверим новую структуру нашей таблицы:

```
mysql> DESCRIBE user;
+----------+------------------+------+-----+---------+----------------+
| Field    | Type             | Null | Key | Default | Extra          |
+----------+------------------+------+-----+---------+----------------+
| id       | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| name     | varchar(30)      | NO   |     | NULL    |                |
| password | varchar(30)      | NO   |     | NULL    |                |
| sex      | enum('M','F')    | NO   |     | NULL    |                |
+----------+------------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

Что стало с записями в таблице после добавления нового столбца? Сделаем выборку всех записей из таблицы:

```
mysql> SELECT * FROM user;
+----+---------+-----------+-----+
| id | name    | password  | sex |
+----+---------+-----------+-----+
```

```
|  1 | admin   | admpass   | M   |
|  2 | dummy   | topsecret | M   |
|  3 | hacker  | sesam     | M   |
|  4 | badguy  | bgpass    | M   |
|  5 | goodguy | ggpass    | M   |
+----+---------+-----------+-----+
5 rows in set (0.00 sec)
```

Значения этого столбца для всех записей получилось равным 'M' (первый элемент в списке значений).

Вставим ещё одну запись в таблицу:

```
mysql> INSERT INTO user VALUES
    -> (NULL,'newuser','newpass','F');
Query OK, 1 row affected (0.01 sec)
```

И вновь сделаем выборку:

```
mysql> SELECT * FROM user;
+----+---------+-----------+-----+
| id | name    | password  | sex |
+----+---------+-----------+-----+
|  1 | admin   | admpass   | M   |
|  2 | dummy   | topsecret | M   |
|  3 | hacker  | sesam     | M   |
|  4 | badguy  | bgpass    | M   |
|  5 | goodguy | ggpass    | M   |
|  6 | newuser | newpass   | F   |
+----+---------+-----------+-----+
6 rows in set (0.00 sec)
```

Весьма вероятно, что после добавления нового столбца значения его в части записей неверны, и их необходимо исправить. Ставя вопрос шире: как модифицировать значение в некоторой ячейке (или наборе ячеек) таблицы? Следующим запросом можно изменить (UPDATE) в таблице user значение поля sex на 'F' (используется ключевое слово SET) в той записи (WHERE), в которой имя пользователя — dummy:

```
mysql> UPDATE user
    -> SET sex='F'
    -> WHERE name='dummy';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Проверим, что получилось в итоге. Поскольку нас интересует только пользователь dummy, то сделаем выборку данных только для этого пользователя, сформировав условие использованием ключевого слова WHERE:

```
mysql> SELECT * FROM user WHERE name='dummy';
+----+-------+-----------+-----+
| id | name  | password  | sex |
+----+-------+-----------+-----+
|  2 | dummy | topsecret | F   |
+----+-------+-----------+-----+
1 row in set (0.00 sec)
```

На самом деле, нас интересует только значение поля sex, поэтому можно выбрать значение только этого поля для данного пользователя:

```
mysql> SELECT sex FROM user WHERE name='dummy';
+-----+
| sex |
+-----+
| F   |
+-----+
1 row in set (0.00 sec)
```

Рассмотрим ещё несколько примеров на выборку записей по условиям.

Выбираем всех пользователей мужского пола:

```
mysql> SELECT * FROM user WHERE sex='M';
+----+---------+----------+-----+
| id | name    | password | sex |
+----+---------+----------+-----+
|  1 | admin   | admpass  | M   |
|  3 | hacker  | sesam    | M   |
|  4 | badguy  | bgpass   | M   |
|  5 | goodguy | ggpass   | M   |
+----+---------+----------+-----+
4 rows in set (0.00 sec)
```

Аналогично для женского:

```
mysql> SELECT * FROM user WHERE sex='F';
+----+---------+-----------+-----+
| id | name    | password  | sex |
+----+---------+-----------+-----+
|  2 | dummy   | topsecret | F   |
|  6 | newuser | newpass   | F   |
+----+---------+-----------+-----+
2 rows in set (0.00 sec)
```

Можно ли получить не сами записи, а только их количество? Для этого есть возможность воспользоваться ещё одним имеющимся в нашем распоряжении инструментом — встроенными функциями, которые предназначены для вычисление различных значений. Функция COUNT() возвращает число строк в результирующем наборе, если в качестве аргумента ей передаётся *. Например, следующий запрос возвращает количество записей в таблице user, для которых значение поля sex равно 'M':

```
mysql> SELECT COUNT(*) FROM user WHERE sex='M';
+----------+
| COUNT(*) |
+----------+
|        4 |
+----------+
1 row in set (0.00 sec)
```

Добавим ещё одно поле в таблицу — адрес электронной почты (ключевое слово AFTER использовано для того, чтобы добавить новый столбец непосредственно после столбца password, а не в конец таблицы):

```
mysql> ALTER TABLE user ADD email VARCHAR(30) NOT NULL AFTER password;
Query OK, 6 rows affected (0.02 sec)
Records: 6  Duplicates: 0  Warnings: 0
```

Вновь посмотрим описание нашей таблицы:

```
mysql> DESCRIBE user;
+----------+------------------+------+-----+---------+----------------+
| Field    | Type             | Null | Key | Default | Extra          |
+----------+------------------+------+-----+---------+----------------+
| id       | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| name     | varchar(30)      | NO   |     | NULL    |                |
| password | varchar(30)      | NO   |     | NULL    |                |
| email    | varchar(30)      | NO   |     | NULL    |                |
| sex      | enum('M','F')    | NO   |     | NULL    |                |
+----------+------------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

Сделаем выборку всех записей:

```
mysql> SELECT * FROM user;
+----+---------+-----------+-------+-----+
| id | name    | password  | email | sex |
+----+---------+-----------+-------+-----+
|  1 | admin   | admpass   |       | M   |
|  2 | dummy   | topsecret |       | F   |
|  3 | hacker  | sesam     |       | M   |
|  4 | badguy  | bgpass    |       | M   |
|  5 | goodguy | ggpass    |       | M   |
|  6 | newuser | newpass   |       | F   |
+----+---------+-----------+-------+-----+
6 rows in set (0.00 sec)
```

Установим значения данного поля для всех записей в таблице:

```
mysql> UPDATE user SET email='admin@uni.udm.ru' WHERE name='admin';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE user SET email='dummy@mail.ru' WHERE name='dummy';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE user SET email='hacker@mail.ru' WHERE name='hacker';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE user SET email='badguy@mail.ru' WHERE name='badguy';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE user SET email='goodguy@yandex.ru' WHERE name='goodguy';
Query OK, 1 row affected (0.00 sec)
```

```
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE user SET email='newuser@yandex.ru' WHERE name='newuser';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

И вновь сделаем выборку всех записей:

```
mysql> SELECT * FROM user;
+----+---------+-----------+-------------------+-----+
| id | name    | password  | email             | sex |
+----+---------+-----------+-------------------+-----+
|  1 | admin   | admpass   | admin@uni.udm.ru  | M   |
|  2 | dummy   | topsecret | dummy@mail.ru     | F   |
|  3 | hacker  | sesam     | hacker@mail.ru    | M   |
|  4 | badguy  | bgpass    | badguy@mail.ru    | M   |
|  5 | goodguy | ggpass    | goodguy@yandex.ru | M   |
|  6 | newuser | newpass   | newuser@yandex.ru | F   |
+----+---------+-----------+-------------------+-----+
6 rows in set (0.00 sec)
```

Помимо добавления в таблицу новых записей, можно и удалять имеющиеся. Например, удалить пользователя с именем badguy можно следующим запросом:

```
mysql> DELETE FROM user
    -> WHERE name='badguy';
Query OK, 1 row affected (0.01 sec)
```

Если сейчас сделать выборку всех записей в таблице, то можно обнаружить, что удалённой записи действительно больше нет:

```
mysql> SELECT * FROM user;
+----+---------+-----------+-------------------+-----+
| id | name    | password  | email             | sex |
+----+---------+-----------+-------------------+-----+
|  1 | admin   | admpass   | admin@uni.udm.ru  | M   |
|  2 | dummy   | topsecret | dummy@mail.ru     | F   |
|  3 | hacker  | sesam     | hacker@mail.ru    | M   |
|  5 | goodguy | ggpass    | goodguy@yandex.ru | M   |
|  6 | newuser | newpass   | newuser@yandex.ru | F   |
+----+---------+-----------+-------------------+-----+
5 rows in set (0.00 sec)
```

Добавим ещё одного пользователя в таблицу:

```
mysql> INSERT INTO user VALUES (NULL,'smartuser','smartpass','smart@mail.ru','F');
Query OK, 1 row affected (0.01 sec)
```

Делаем выборку всех записей:

```
mysql> SELECT * FROM user;
+----+-----------+-----------+-------------------+-----+
| id | name      | password  | email             | sex |
```

```
+----+-----------+-----------+-------------------+-----+
|  1 | admin     | admpass   | admin@uni.udm.ru  | M   |
|  2 | dummy     | topsecret | dummy@mail.ru     | F   |
|  3 | hacker    | sesam     | hacker@mail.ru    | M   |
|  5 | goodguy   | ggpass    | goodguy@yandex.ru | M   |
|  6 | newuser   | newpass   | newuser@yandex.ru | F   |
|  7 | smartuser | smartpass | smart@mail.ru     | F   |
+----+-----------+-----------+-------------------+-----+
6 rows in set (0.00 sec)
```

Удалённая ранее запись имела значение id, равное 4. Нетрудно заметить, что после добавления новой записи, её id получил значение 7, а не 4.

Может возникнуть потребность внести изменения в структуру таблицы иного плана, например, изменить спецификации некоторого поля. Скажем, поле name в нашей таблице явно должно быть уникальным, но при создании таблицы это не было указано. Можно это исправить, воспользовавшись вновь ALTER TABLE:

```
mysql> ALTER TABLE user MODIFY name VARCHAR(20) NOT NULL UNIQUE;
Query OK, 6 rows affected (0.02 sec)
Records: 6  Duplicates: 0  Warnings: 0
```

Ключевое слово MODIFY указывает на то, что мы модифицируем поле name, добавляя спецификацию UNIQUE (заодно здесь изменяется тип данных: вместо строки переменной длины до 30 символов будет строка до 20 символов длиной — предположим, было решено, что такой длины достаточно).

После этого таблица имеет следующе описание:

```
mysql> DESCRIBE user;
+----------+------------------+------+-----+---------+----------------+
| Field    | Type             | Null | Key | Default | Extra          |
+----------+------------------+------+-----+---------+----------------+
| id       | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| name     | varchar(20)      | NO   | UNI | NULL    |                |
| password | varchar(30)      | NO   |     | NULL    |                |
| email    | varchar(30)      | NO   |     | NULL    |                |
| sex      | enum('M','F')    | NO   |     | NULL    |                |
+----------+------------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

Следует с осторожностью пользоваться данной возможностью для таблиц, в которых уже есть записи, иначе можно получить нежелательный эффект:

```
mysql> ALTER TABLE user MODIFY name VARCHAR(2) NOT NULL UNIQUE;
Query OK, 6 rows affected, 6 warnings (0.03 sec)
Records: 6  Duplicates: 0  Warnings: 6
```

В выборке записей после этого получаем следующее:

```
mysql> SELECT * FROM user;
+----+------+-----------+-------------------+-----+
| id | name | password  | email             | sex |
+----+------+-----------+-------------------+-----+
|  1 | ad   | admpass   | admin@uni.udm.ru  | M   |
```

```
|  2 | du   | topsecret | dummy@mail.ru     | F   |
|  3 | ha   | sesam     | hacker@mail.ru    | M   |
|  5 | go   | ggpass    | goodguy@yandex.ru | M   |
|  6 | ne   | newpass   | newuser@yandex.ru | F   |
|  7 | sm   | smartpass | smart@mail.ru     | F   |
+----+------+-----------+-------------------+-----+
6 rows in set (0.00 sec)
```

Упражнение 1. Составить запрос, с помощью которого можно получить пароль пользователя, у которого адрес электронной почты `newuser@yandex.ru`.

Упражнение 2. Сменить пароль пользователя с именем `hacker` на `sesamum`.

Упражнение 3. Есть ли в таблице ещё какое-нибудь поле, которое претендует на то, чтобы его значения были уникальными? Если да, то какое? Внести изменения в таблицу.

Упражнение 4. Какие поля ещё можно добавить в таблицу? Какого типа? Какие у них должны быть спецификации? Внести изменения, заполнить данными.

Упражнение 5. Добавить ещё несколько записей в таблицу, используя все три выше рассмотренных способа.

Литература

1) Бойко В. В., Савинков В. М. Проектирование баз данных информационных систем. М.: Финансы и статистика, 1989.

2) Бхамидипати К. SQL. Справочник программиста. М.: Эком, 2003.

3) Грофф Дж. Р. SQL: Полное руководство. Киев: Изд. гр. BHV, 2001.

4) Дейт К. Дж. Введение в системы баз данных. М.: Вильямс, 2006.

5) Диго С. М. Базы данных: проектирование и использование: учебник для вузов по специальности «Прикладная информатика (по областям)». М.: Финансы и статистика, 2005.

6) Дюбуа П. MySQL. Сборник рецептов. СПб.—М.: Символ-Плюс, 2005.

7) Коннолли Т., Бегг К. Базы данных. Проектирование, реализация и сопровождение: Теория и практика. М.; СПб.; Киев: Изд. Дом «Вильямс», 2003.

8) Крёнке Д. М. Теория и практика построения баз данных. СПб.: Питер, 2005.

9) Кузин А. В. Базы данных: учебное пособие. М.: Академия, 2008.

10) Кузнецов С. Д. Основы баз данных: курс лекций. М.: Интернет-Университет Информационных Технологий, 2005.

11) Хансен Г. Базы данных: разработка и управление. М.: БИНОМ, 2000.

12) Харрингтон Д. Проектирование реляционных баз данных. М.: Лори, 2006.

13) Хомоненко А. Д., Цыганков В. М., Мальцев М. Г. Базы данных. М.: Бином-Пресс; СПб.: КОРОНА принт, 2006.