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 ERMaster;
4) HTTP-сервер Apache2.

Вся работа должна выполняться в настроенном рабочем окружении по инструкции с репозитория на GitHub — https://github.com/AVKorotkov/setupenv. Работы из данных методических указаний не следует выполнять 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 VirtualBox. 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

1.1. MySQL monitor: опции и параметры, интерфейс, команды.

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 IP).
- -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.

1.2. Создание и удаление базы данных.

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 (более того, использование данной команды в такой форме с символом «;» в конце приведёт к ошибке, поскольку этот символ будет воспринят как часть имени базы данных):

```
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.

1.3. Создание таблиц. Добавление и выборка записей.

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 сделать это 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 идентичным:

```
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 передав 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.

1.4. Модификация структуры таблицы. Модификация, удаление записей.

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. Или, 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 (как именно будет обработан такой запрос — зависит от настроек сервера MySQL, например, если установлена опция `sql-mode="TRADITIONAL"` в файлах конфигурации, то на такой запрос будет выдана ошибка, и изменения внесены не будут):

```
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)
```

Для самостоятельного решения

Упражнение 1.1. Составить запрос, с помощью которого можно получить пароль пользователя, у которого адрес электронной почты `newuser@yandex.ru`.

Упражнение 1.2. Сменить пароль пользователя с именем `hacker` на `sesamum`.

Упражнение 1.3. 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 1.4. Какие поля ещё можно добавить в таблицу? Какого типа? Какие у них должны быть спецификации? Внести изменения, заполнить данными.

Упражнение 1.5. Добавить ещё несколько записей в таблицу, используя все три выше рассмотренных способа.

Глава 2

Работа с несколькими таблицами

2.1. Внешние ключи и ограничения ссылочной целостности

Продолжим строить базу данных нашего форума.

Любой форум имеет древовидную структуру (в смысле теории графов). На самом верхнем уровне находятся разделы (категории). Каждый раздел может содержать несколько тем, которые, в свою очередь, могут содержать подтемы и т. д. На самом нижнем уровне этой иерархии находятся сообщения (topics). Каждое сообщение имеет название, содержание (текст сообщения), автора и относится к тому или иному разделу форума. Также принято помечать каждое сообщение указанием на дату и время его создания.

Любое сообщение (как правило) может сопровождаться комментариями других пользователей форума. Причём комментироваться могут как исходное сообщение, так и ранее сделанные комментарии.

Добавим в нашу базу данных таблицы, которые будут обеспечивать этот функционал. Конечно, он многого не учитывает: например, общепринятой практикой является разделение всех пользователей на группы (обычные пользователи, пользователи с повышенными правами — например, такие как модераторы или администраторы, пользователи с пониженными правами — например, те, кому запрещено создавать темы или писать комментарии и т. п.). Но в данный момент, чтобы не усложнять модель, эти аспекты учитывать не будем.

При создании таблицы пользователей был создан суррогатный первичный ключ — `id`. Вместо этого можно было в качестве первичного ключа использовать имя пользователя, поскольку оно должно быть уникальным. Используем суррогатные первичные ключи и для других таблиц нашего форума.

Создадим таблицу, хранящую структуру форума. В ней должны быть названия всех разделов и подразделов. Поскольку мы имеем дело здесь с древовидной структурой, то для хранения её мы должны знать для каждого узла его родителя. Следовательно, в каждой записи этого отношения должен быть атрибут, указывающий на запись, которая является для данной записи родительской. Если родитель — корень, то в соответствующую ячейку такой записи будем помещать нуль.

```
mysql> CREATE TABLE category
    -> (
    -> id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    -> name VARCHAR(35) NOT NULL,
    -> parent INT UNSIGNED NOT NULL,
    -> PRIMARY KEY (id)
    -> );
Query OK, 0 rows affected (0.02 sec)
```

Созданная таблица имеет следующую структуру:

```
mysql> DESCRIBE category;
+--------+------------------+------+-----+---------+----------------+
| Field  | Type             | Null | Key | Default | Extra          |
+--------+------------------+------+-----+---------+----------------+
| id     | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| name   | varchar(35)      | NO   |     | NULL    |                |
| parent | int(10) unsigned | NO   |     | NULL    |                |
+--------+------------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)
```

Создадим таблицу сообщений. Поля в ней сделаем следующие: id — идентификатор сообщения (первичный ключ); name — название темы сообщения; content — текст сообщения; dtcreation — дата и время создания сообщения; category_id — идентификатор раздела форума, в котором создаётся сообщение; user_id — идентификатор пользователя, создавшего сообщение. Тип данных поля content — TEXT, это строковый тип, который может содержать от 0 до 65535 символов; тип DATETIME предназначен для одновременного хранения даты и времени, при этом используется следующий формат данных следующего вида: ГГГГ-ММ-ДД ЧЧ:ММ:СС.

```
mysql> CREATE TABLE topic
    -> (
    -> id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    -> name VARCHAR(45) NOT NULL,
    -> content TEXT NOT NULL,
    -> dtcreation DATETIME NOT NULL,
    -> category_id INT UNSIGNED NOT NULL,
    -> user_id INT UNSIGNED NOT NULL,
    -> PRIMARY KEY (id)
    -> );
Query OK, 0 rows affected (0.01 sec)
```

Поле user_id — это внешний ключ, связывающий отношения user и topic (user выступает здесь в роли родителя, а topic — в роли потомка). Аналогично, поле category_id — это внешний ключ, связывающий отношения category и topic (category — родитель, а topic — потомок). Отразим этот факт в таблице topic, воспользовавшись ALTER TABLE:

```
mysql> ALTER TABLE topic
    -> ADD FOREIGN KEY (user_id) REFERENCES user (id)
    -> ON UPDATE RESTRICT
    -> ON DELETE RESTRICT;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE topic
    -> ADD FOREIGN KEY (category_id) REFERENCES category (id)
    -> ON UPDATE RESTRICT
    -> ON DELETE RESTRICT;
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

В этом запросе мы добавили внешний ключ user_id в таблицу topic, ссылающийся на первичный ключ id таблицы user, указав ограничения ссылочной целостности: ON

UPDATE RESTRICT и ON DELETE RESTRICT (запрет на обновление и удаление первичного ключа в родительском отношении, если имеется хотя бы одна ссылка на него в отношении-потомке). Такой же смысл имеет второй запрос, связанный с внешним ключом category_id. Особенностью MySQL является то, что в ограничениях ссылочной целостности условие ON DELETE RESTRICT можно вообще опустить или даже заменить условием ON DELETE NO ACTION, результат будет идентичным (в других СУБД поведение этих двух условий может быть различным — например, в IBM DB2, PostgreSQL и SQLite; в СУБД Oracle условие RESTRICT попросту отсутствует).

Таблица topic имеет теперь следующее описание:

```
mysql> DESCRIBE topic;
+-------------+------------------+------+-----+---------+----------------+
| Field       | Type             | Null | Key | Default | Extra          |
+-------------+------------------+------+-----+---------+----------------+
| id          | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| name        | varchar(45)      | NO   |     | NULL    |                |
| content     | text             | NO   |     | NULL    |                |
| dtcreation  | datetime         | NO   |     | NULL    |                |
| category_id | int(10) unsigned | NO   | MUL | NULL    |                |
| user_id     | int(10) unsigned | NO   | MUL | NULL    |                |
+-------------+------------------+------+-----+---------+----------------+
6 rows in set (0.00 sec)
```

Добавим ещё таблицу, в которой будут храниться комментарии к сообщениям:

```
mysql> CREATE TABLE comment
    -> (
    -> id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    -> name VARCHAR(45) NOT NULL,
    -> content TEXT NOT NULL,
    -> dtcreation DATETIME NOT NULL,
    -> parent INT UNSIGNED NOT NULL,
    -> topic_id INT UNSIGNED NOT NULL,
    -> user_id INT UNSIGNED NOT NULL,
    -> PRIMARY KEY (id),
    -> FOREIGN KEY (topic_id) REFERENCES topic (id)
    -> ON UPDATE RESTRICT
    -> ON DELETE CASCADE,
    -> FOREIGN KEY (user_id) REFERENCES user (id)
    -> ON UPDATE RESTRICT
    -> ON DELETE RESTRICT
    -> );
Query OK, 0 rows affected (0.01 sec)
```

Поля в этой таблице следующие: id — идентификатор комментария (первичный ключ); name — заголовок комментария; content — текст комментария; dtcreation — дата и время создания комментария; parent — идентификатор комментария, на который данный является ответом (если это ответ на комментарий; если комментарий относится к самому исходному сообщению, то в этом поле будем располагать нулевое значение); topic_id — идентификатор сообщения, к которому этот комментарий относится; user_id — идентификатор пользователя, создавшего комментарий. Ограничения ссылочной целостности в этой таблице несколько другие, чем в предыдущей. Объясняется это тем,

что по смыслу удаление некоторого сообщения должно влечь за собой и удаление всех комментариев под ним, поэтому здесь использовано условие ON DELETE CASCADE для внешнего ключа topic_id.

Описание данной таблицы:

```
mysql> DESCRIBE comment;
+------------+------------------+------+-----+---------+----------------+
| Field      | Type             | Null | Key | Default | Extra          |
+------------+------------------+------+-----+---------+----------------+
| id         | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| name       | varchar(45)      | NO   |     | NULL    |                |
| content    | text             | NO   |     | NULL    |                |
| dtcreation | datetime         | NO   |     | NULL    |                |
| parent     | int(10) unsigned | NO   |     | NULL    |                |
| topic_id   | int(10) unsigned | NO   | MUL | NULL    |                |
| user_id    | int(10) unsigned | NO   | MUL | NULL    |                |
+------------+------------------+------+-----+---------+----------------+
7 rows in set (0.00 sec)
```

Заполним некоторыми данными таблицу category. Сделаем это по -другому, чем раньше, без использования команды INSERT. Последняя годится для небольших вставок записей в таблицу (в количестве нескольких штук), для вставки большого количества строк лучше пользоваться другими инструментами. Команда LOAD DATA считывает заранее подготовленные в текстовом файле данные и заносит их в соответствующую таблицу. Эту команду можно использовать для загрузки данных как при использовании файла, расположенного на сервере, так и на стороне клиента (в случае, когда сервер и клиент расположены на одном хосте, то это одно и то же). В случае использования файла с данными на стороне клиента, данные считываются клиентской программой и передаются далее на сервер, при этом дополнительно используется для данной команды ключевое слово LOCAL.

Создаём в любом текстовом редакторе (nano, gedit, kwrite или любом другом) файл в формате CSV — это плоский текстовый файл, каждая строка которого представляет из себя набор полей, разделённых некоторым символом, в качестве которого часто используется запятая (отсюда и название — Comma-Separated Values, т. е., значения, разделённые запятыми, хотя это и не является обязательным — например, с той же целью часто используется символ табуляции). По умолчанию MySQL использует CSV-файлы с разделителем полей в виде табуляции. Вводим следующие данные, разделяя поля символами табуляции и сохраняя результат в файл с именем category1.csv:

```
\N Программирование 0
\N Базы данных 0
\N Языки разметки 0
```

Каждая строка этого файла содержит три поля: в первое поле мы поместили значение NULL (для загрузки из CSV-файла используется при написании пустого значения «\N»); во второе — название темы; в третье — нулевое значение, поскольку это названия разделов на самом верху иерархии. Лишних строк (пустых) в этом файле быть не должно, иначе попытка загрузки данных завершится ошибкой, т. е., должно быть ровно три строки, после которых располагается символ конца файла (визуально он не отображается). Кроме того, между словами «Базы» и «данных» должен быть пробел, а не символ табуляции (это одно значение; если вместо пробела между этими словами поставить знак табуляции, то это будет воспринято как значения из двух разных полей, и вместо загрузки данных в таблицу мы получим сообщение об ошибке; то же самое замечание относится и к последней строке). Загружаем данные из этого файла в таблицу:

```
mysql> LOAD DATA LOCAL INFILE 'category1.csv' INTO TABLE category;
Query OK, 3 rows affected (0.03 sec)
Records: 3  Deleted: 0  Skipped: 0  Warnings: 0
```

В результате выполнения данной команды в таблице `category` мы получаем следующие записи:

```
mysql> SELECT * FROM category;
+----+---------------------------------+--------+
| id | name                            | parent |
+----+---------------------------------+--------+
|  1 | Программирование                |      0 |
|  2 | Базы данных                     |      0 |
|  3 | Языки разметки                  |      0 |
+----+---------------------------------+--------+
3 rows in set (0.00 sec)
```

Подготовим ещё один CSV-файл для загрузки данных в эту же таблицу, `category2.csv`:

```
"","Языки программирования","1"
"","Программы для разработки","1"
"","СУБД","2"
"","Проектирование","2"
"","Создание модели","2"
"","Администрирование","2"
"","Языки описания данных","2"
"","Языки запросов","2"
"","HTML","3"
"","SGML","3"
"","TEX","3"
"","Лёгкие языки разметки","3"
```

Здесь использован несколько иной формат (о чём упоминалось выше): разделителем сделана запятая, значения полей обрамляются символом «"»; в первом поле в качестве значения указывается «""», что равносильно пустому значению, а поскольку поле это должно быть непустым, и при создании таблицы для него была указана спецификация `AUTO_INCREMENT`, то значения для него заполняются автоматически. Для внесения данных в таблицу в таком формате команду надо несколько модифицировать:

```
mysql> LOAD DATA LOCAL INFILE 'category2.csv' INTO TABLE category
    -> FIELDS TERMINATED BY ',' ENCLOSED BY '"';
Query OK, 12 rows affected, 12 warnings (0.01 sec)
Records: 12  Deleted: 0  Skipped: 0  Warnings: 12
```

Здесь дополнительно указывается, что поля разделяются запятыми, а значения заключаются между «"». Теперь в данной таблице имеются следующие записи:

```
mysql> SELECT * FROM category;
+----+-----------------------------------------------+--------+
| id | name                                          | parent |
+----+-----------------------------------------------+--------+
|  1 | Программирование                              |      0 |
|  2 | Базы данных                                   |      0 |
```

```
|  3 | Языки разметки                                 |      0 |
|  4 | Языки программирования                         |      1 |
|  5 | Программы для разработки                       |      1 |
|  6 | СУБД                                           |      2 |
|  7 | Проектирование                                 |      2 |
|  8 | Создание модели                                |      2 |
|  9 | Администрирование                              |      2 |
| 10 | Языки описания данных                          |      2 |
| 11 | Языки запросов                                 |      2 |
| 12 | HTML                                           |      3 |
| 13 | SGML                                           |      3 |
| 14 | TEX                                            |      3 |
| 15 | Лёгкие языки разметки                          |      3 |
+----+------------------------------------------------+--------+
15 rows in set (0.00 sec)
```

Готовим следующий файл, `category3.csv`:

```
"","FORTRAN","4"
"","C","4"
"","C++","4"
"","ALGOL","4"
"","Haskell","4"
"","Python","4"
"","Ruby","4"
"","Java","4"
"","Lisp","4"
"","OCaml","4"
"","Pascal","4"
"","Smalltalk","4"
"","Erlang","4"
"","Prolog","4"
"","Perl","4"
"","Scheme","4"
"","Компиляторы и интерпретаторы","5"
"","Отладчики","5"
"","Системы контроля версий","5"
"","Редакторы и IDE","5"
"","Oracle","6"
"","MySQL","6"
"","SQLite","6"
"","PostgreSQL","6"
"","DB2","6"
"","MongoDB","6"
"","CouchDB","6"
"","Neo4j","6"
"","SQL","11"
"","LogiQL","11"
"","Cypher","11"
"","Markdown","15"
"","reStructuredText","15"
"","Textile","15"
```

```
"","AsciiDoc","15"
```

Загружаем его в таблицу:

```
mysql> LOAD DATA LOCAL INFILE 'category3.csv' INTO TABLE category
    -> FIELDS TERMINATED BY ',' ENCLOSED BY '"';
Query OK, 35 rows affected, 35 warnings (0.03 sec)
Records: 35  Deleted: 0  Skipped: 0  Warnings: 35
```

Теперь в таблице имеются следующие записи:

```
mysql> SELECT * FROM category;
+----+-----------------------------------------------------+--------+
| id | name                                                | parent |
+----+-----------------------------------------------------+--------+
|  1 | Программирование                                    |      0 |
|  2 | Базы данных                                         |      0 |
|  3 | Языки разметки                                      |      0 |
|  4 | Языки программирования                              |      1 |
|  5 | Программы для разработки                            |      1 |
|  6 | СУБД                                                |      2 |
|  7 | Проектирование                                      |      2 |
|  8 | Создание модели                                     |      2 |
|  9 | Администрирование                                   |      2 |
| 10 | Языки описания данных                               |      2 |
| 11 | Языки запросов                                      |      2 |
| 12 | HTML                                                |      3 |
| 13 | SGML                                                |      3 |
| 14 | TEX                                                 |      3 |
| 15 | Лёгкие языки разметки                               |      3 |
| 16 | FORTRAN                                             |      4 |
| 17 | C                                                   |      4 |
| 18 | C++                                                 |      4 |
| 19 | ALGOL                                               |      4 |
| 20 | Haskell                                             |      4 |
| 21 | Python                                              |      4 |
| 22 | Ruby                                                |      4 |
| 23 | Java                                                |      4 |
| 24 | Lisp                                                |      4 |
| 25 | OCaml                                               |      4 |
| 26 | Pascal                                              |      4 |
| 27 | Smalltalk                                           |      4 |
| 28 | Erlang                                              |      4 |
| 29 | Prolog                                              |      4 |
| 30 | Perl                                                |      4 |
| 31 | Scheme                                              |      4 |
| 32 | Компиляторы и интерпретаторы                        |      5 |
| 33 | Отладчики                                           |      5 |
| 34 | Системы контроля версий                             |      5 |
| 35 | Редакторы и IDE                                     |      5 |
| 36 | Oracle                                              |      6 |
| 37 | MySQL                                               |      6 |
| 38 | SQLite                                              |      6 |
```

```
| 39 | PostgreSQL                                                |      6 |
| 40 | DB2                                                       |      6 |
| 41 | MongoDB                                                   |      6 |
| 42 | CouchDB                                                   |      6 |
| 43 | Neo4j                                                     |      6 |
| 44 | SQL                                                       |     11 |
| 45 | LogiQL                                                    |     11 |
| 46 | Cypher                                                    |     11 |
| 47 | Markdown                                                  |     15 |
| 48 | reStructuredText                                          |     15 |
| 49 | Textile                                                   |     15 |
| 50 | AsciiDoc                                                  |     15 |
+----+-----------------------------------------------------------+--------+
50 rows in set (0.00 sec)
```

Наконец, готовим последний файл для загрузки в эту таблицу, `category4.csv`:

```
"","GCC","32"
"","Free Pascal","32"
"","YAP","32"
"","Stalin","32"
"","GHC","32"
"","GDB","33"
"","Git","34"
"","Bazaar","34"
"","SVN","34"
"","Emacs","35"
"","Vim","35"
"","Eclipse","35"
"","Netbeans","35"
```

Загружаем данные:

```
mysql> LOAD DATA LOCAL INFILE 'category4.csv' INTO TABLE category
    -> FIELDS TERMINATED BY ',' ENCLOSED BY '"';
Query OK, 13 rows affected, 13 warnings (0.02 sec)
Records: 13  Deleted: 0  Skipped: 0  Warnings: 13
```

И проверяем результат:

```
mysql> SELECT * FROM category;
+----+-----------------------------------------------------------+--------+
| id | name                                                      | parent |
+----+-----------------------------------------------------------+--------+
|  1 | Программирование                                          |      0 |
|  2 | Базы данных                                               |      0 |
|  3 | Языки разметки                                            |      0 |
|  4 | Языки программирования                                    |      1 |
|  5 | Программы для разработки                                  |      1 |
|  6 | СУБД                                                      |      2 |
|  7 | Проектирование                                            |      2 |
|  8 | Создание модели                                           |      2 |
|  9 | Администрирование                                         |      2 |
```

10	Языки описания данных	2
11	Языки запросов	2
12	HTML	3
13	SGML	3
14	TEX	3
15	Лёгкие языки разметки	3
16	FORTRAN	4
17	C	4
18	C++	4
19	ALGOL	4
20	Haskell	4
21	Python	4
22	Ruby	4
23	Java	4
24	Lisp	4
25	OCaml	4
26	Pascal	4
27	Smalltalk	4
28	Erlang	4
29	Prolog	4
30	Perl	4
31	Scheme	4
32	Компиляторы и интерпретаторы	5
33	Отладчики	5
34	Системы контроля версий	5
35	Редакторы и IDE	5
36	Oracle	6
37	MySQL	6
38	SQLite	6
39	PostgreSQL	6
40	DB2	6
41	MongoDB	6
42	CouchDB	6
43	Neo4j	6
44	SQL	11
45	LogiQL	11
46	Cypher	11
47	Markdown	15
48	reStructuredText	15
49	Textile	15
50	AsciiDoc	15
51	GCC	32
52	Free Pascal	32
53	YAP	32
54	Stalin	32
55	GHC	32
56	GDB	33
57	Git	34
58	Bazaar	34
59	SVN	34
60	Emacs	35

```
| 61 | Vim                                                      |     35 |
| 62 | Eclipse                                                  |     35 |
| 63 | Netbeans                                                 |     35 |
+----+----------------------------------------------------------+--------+
63 rows in set (0.00 sec)
```

2.2. Вложенные запросы

Выбрать из полученной таблицы разделы, находящиеся на самом верху иерархии (в корне форума) просто:

```
mysql> SELECT name AS Раздел
    -> FROM category
    -> WHERE parent=0;
+---------------------------------+
| Раздел                          |
+---------------------------------+
| Программирование                |
| Базы данных                     |
| Языки разметки                  |
+---------------------------------+
3 rows in set (0.00 sec)
```

Как теперь можно сделать выборку подразделов, которые относятся к разделу Программирование? Разумеется, можно использовать тот факт, что значение поля id для раздела Программирование равно единице, и построить запрос аналогично предыдущему, указав это значение поля parent. Нетрудно заметить, что такой способ является неудобным, поскольку каждый раз при формировании запроса нужно смотреть значение поля id, которое является суррогатным ключом и соответственно, имеет искусственно формируемое значение. Но его можно получить с помощью запроса, применив ключевое слово WHERE (что уже рассматривалось ранее):

```
mysql> SELECT id
    -> FROM category
    -> WHERE name='Программирование';
+----+
| id |
+----+
|  1 |
+----+
1 row in set (0.00 sec)
```

В SQL предусмотрена возможность построения вложенных запросов, чем сейчас и воспользуемся:

```
mysql> SELECT name AS Раздел
    -> FROM category
    -> WHERE parent = (
    -> SELECT id
    -> FROM category
    -> WHERE name='Программирование'
    -> );
```

```
+-----------------------------------------------+
| Раздел                                        |
+-----------------------------------------------+
| Языки программирования                        |
| Программы для разработки                      |
+-----------------------------------------------+
2 rows in set (0.01 sec)
```

Внутренний запрос выбирает значение поля id для раздела Программирование, внешний по полученному значению поля parent делает уже выборку подразделов этого раздела.

Аналогичным образом найдём подразделы теперь уже раздела Языки программирования:

```
mysql> SELECT name AS Раздел
    -> FROM category
    -> WHERE parent = (
    -> SELECT id
    -> FROM category
    -> WHERE name='Языки программирования'
    -> );
+--------------+
| Раздел       |
+--------------+
| FORTRAN      |
| C            |
| C++          |
| ALGOL        |
| Haskell      |
| Python       |
| Ruby         |
| Java         |
| Lisp         |
| OCaml        |
| Pascal       |
| Smalltalk    |
| Erlang       |
| Prolog       |
| Perl         |
| Scheme       |
+--------------+
16 rows in set (0.00 sec)
```

Выберем все подразделы двух разделов: Системы контроля версий и Редакторы и IDE. Составим вначале внутренний запрос:

```
mysql> SELECT id
    -> FROM category
    -> WHERE name='Системы контроля версий' OR name='Редакторы и IDE';
+----+
| id |
+----+
```

```
| 34 |
| 35 |
+----+
2 rows in set (0.00 sec)
```

Здесь использовано ключевое слово OR, смысл которого — обычная дизъюнкция. В результате выборки имеем два значения. Теперь нам нужно составить запрос, который выберет для найденных двух значений те разделы, для которых найденные являются родительскими. Для этого используется ключевое слово IN (значение поля должно содержаться среди значений результирующего набора для внутреннего запроса):

```
mysql> SELECT name AS Раздел
    -> FROM category
    -> WHERE parent IN (
    -> SELECT id
    -> FROM category
    -> WHERE name='Системы контроля версий' OR name='Редакторы и IDE'
    -> );
+--------------+
| Раздел       |
+--------------+
| Git          |
| Bazaar       |
| SVN          |
| Emacs        |
| Vim          |
| Eclipse      |
| Netbeans     |
+--------------+
7 rows in set (0.00 sec)
```

Составим запрос, выбирающий названия всех разделов, у которых есть подразделы. Разобьём задачу на два этапа: вначале составим запрос, выбирающий идентификаторы таких разделов. Это ровно те идентификаторы, для записей которых значение совпадает со значением поля parent в некоторой другой записи. Но для многих из таких записей значения поля parent являются одинаковыми, в то время как хотелось бы получить только уникальные значения, без дублирования. Для таких целей существует ключевое слово DISTINCT:

```
mysql> SELECT DISTINCT parent
    -> FROM category;
+--------+
| parent |
+--------+
|      0 |
|      1 |
|      2 |
|      3 |
|      4 |
|      5 |
|      6 |
|     11 |
|     15 |
```

```
|     32 |
|     33 |
|     34 |
|     35 |
+--------+
13 rows in set (0.00 sec)
```

Составим теперь интересующий нас запрос полностью. Идентификаторы разделов, имеющих подразделы, содержатся в наборе всех различных значений поля parent (поскольку значения идентификатора начинаются с единицы, то количество записей в выборке ровно на единицу меньше, чем во внутреннем запросе; нулевое значение поля parent принадлежит корню форума).

```
mysql> SELECT name
    -> FROM category
    -> WHERE id IN (
    -> SELECT DISTINCT parent
    -> FROM category
    -> );
+-------------------------------------------------------+
| name                                                  |
+-------------------------------------------------------+
| Программирование                                      |
| Базы данных                                           |
| Языки разметки                                        |
| Языки программирования                                |
| Программы для разработки                              |
| СУБД                                                  |
| Языки запросов                                        |
| Лёгкие языки разметки                                 |
| Компиляторы и интерпретаторы                          |
| Отладчики                                             |
| Системы контроля версий                               |
| Редакторы и IDE                                       |
+-------------------------------------------------------+
12 rows in set (0.00 sec)
```

2.3. Группировка и псевдонимы

Усложним задачу. Выведем помимо тех разделов форума, у которых есть подразделы, ещё и количество подразделов для каждого такого раздела. Это можно сделать с помощью следующего запроса:

```
mysql> SELECT c1.name AS Раздел, COUNT(c2.id) AS Число
    -> FROM category c1, category c2
    -> WHERE c1.id IN (
    -> SELECT DISTINCT parent
    -> FROM category
    -> ) AND
    -> c2.parent=c1.id
    -> GROUP BY c1.id;
```

```
+-----------------------------------------------------+------------+
| Раздел                                              | Число      |
+-----------------------------------------------------+------------+
| Программирование                                    |          2 |
| Базы данных                                         |          6 |
| Языки разметки                                      |          4 |
| Языки программирования                              |         16 |
| Программы для разработки                            |          4 |
| СУБД                                                |          8 |
| Языки запросов                                      |          3 |
| Лёгкие языки разметки                               |          4 |
| Компиляторы и интерпретаторы                        |          5 |
| Отладчики                                           |          1 |
| Системы контроля версий                             |          3 |
| Редакторы и IDE                                     |          4 |
+-----------------------------------------------------+------------+
12 rows in set (0.00 sec)
```

Здесь использовано несколько ранее не рассматривавшихся конструкций. После ключевого слова FROM в запросе перечислены (через запятую) `category c1` и `category c2`. Выборка данных идёт из одной таблицы, но для решения задачи её нужно делать так, как будто данные извлекаются из двух разных таблиц. Для этого используются псевдонимы (имя, которое указывается после имени таблицы через пробел). В данном примере для таблицы `category` использовано два псевдонима: `c1` и `c2`. Можно теперь в запросе использовать их в качестве заменителя имени таблицы. В тех случаях, когда в запросе присутствует более одной таблицы (или таблица одна, но более одного псевдонима), а имена столбцов не являются уникальными (для одной таблицы с псевдонимами это неизбежно), то для того, чтобы различать, из какой именно таблицы (или для какого псевдонима) используется в запросе имя столбца, его следует указывать в формате `имя_таблицы.имя_столбца` (или, при использовании псевдонимов, в формате `псевдоним.имя_столбца`). Условие выборки представляет в данном случае конъюнкцию двух условий (AND). GROUP BY означает группировку результатов выборки по указанному полю (в данном случае — по имени раздела).

2.4. Ещё некоторые способы вставки записей

Заполним некоторыми данными остальные две таблицы.

Вначале вставляем записи в таблицу `topic`, поскольку комментариев к несуществующим темам не бывает:

```
mysql> INSERT INTO topic SET
    -> name = 'Как написать функцию?',
    -> content = 'Помогите.',
    -> dtcreation = NOW(),
    -> category_id = (
    -> SELECT id FROM category
    -> WHERE name = 'Pascal'
    -> ),
    -> user_id = (
    -> SELECT id FROM user
    -> WHERE name = 'dummy'
```

```
    -> );
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO topic SET
    -> name = 'Как написать запрос?',
    -> content = 'Помогите!',
    -> dtcreation = NOW(),
    -> category_id = (
    -> SELECT id FROM category
    -> WHERE name = 'SQL'
    -> ),
    -> user_id = (
    -> SELECT id FROM user
    -> WHERE name = 'dummy'
    -> );
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO topic SET
    -> name = 'Как отладить?',
    -> content = 'Помогите!!',
    -> dtcreation = NOW(),
    -> category_id = (
    -> SELECT id FROM category
    -> WHERE name = 'GDB'
    -> ),
    -> user_id = (
    -> SELECT id FROM user
    -> WHERE name = 'newuser'
    -> );
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO topic SET
    -> name = 'Как обновить?',
    -> content = 'Не могу понять.',
    -> dtcreation = NOW(),
    -> category_id = (
    -> SELECT id FROM category
    -> WHERE name = 'Git'
    -> ),
    -> user_id = (
    -> SELECT id FROM user
    -> WHERE name = 'dummy'
    -> );
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO topic SET
    -> name = 'Подойдёт ли SQLite?',
    -> content = 'Не знаю.',
    -> dtcreation = NOW(),
    -> category_id = (
    -> SELECT id FROM category
```

```
    -> WHERE name = 'SQLite'
    -> ),
    -> user_id = (
    -> SELECT id FROM user
    -> WHERE name = 'dummy'
    -> );
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO topic SET
    -> name = 'Предлагаю обсудить.',
    -> content = 'Мне нравится.',
    -> dtcreation = NOW(),
    -> category_id = (
    -> SELECT id FROM category
    -> WHERE name = 'Vim'
    -> ),
    -> user_id = (
    -> SELECT id FROM user
    -> WHERE name = 'admin'
    -> );
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO topic SET
    -> name = 'Отличный редактор.',
    -> content = 'Пользуюсь.',
    -> dtcreation = NOW(),
    -> category_id = (
    -> SELECT id FROM category
    -> WHERE name = 'Emacs'
    -> ),
    -> user_id = (
    -> SELECT id FROM user
    -> WHERE name = 'hacker'
    -> );
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO topic SET
    -> name = 'Как написать?',
    -> content = 'Не получается.',
    -> dtcreation = NOW(),
    -> category_id = (
    -> SELECT id FROM category
    -> WHERE name = 'SQL'
    -> ),
    -> user_id = (
    -> SELECT id FROM user
    -> WHERE name = 'newuser'
    -> );
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO topic SET
```

```
    -> name = 'Это хороший язык?',
    -> content = 'Подскажите.',
    -> dtcreation = NOW(),
    -> category_id = (
    -> SELECT id FROM category
    -> WHERE name = 'Ruby'
    -> ),
    -> user_id = (
    -> SELECT id FROM user
    -> WHERE name = 'smartuser'
    -> );
Query OK, 1 row affected (0.01 sec)
```

Здесь была использована ещё одна форма запроса на добавление записи в таблицу:

`INSERT INTO имя_таблицы SET имя_ст1 = зн1, имя_ст2 = зн2,...`

При вставке записей, значения `id` были просто опущены ввиду их автоматического присваивания путём наращивания; для ввода значений полей `category_id` и `user_id` были использованы внутренние запросы, извлекающие идентификаторы раздела и пользователя по их названиям и именам — это удобнее, чем, вставляя запись о теме, которая относится к некоторому разделу (например, `GDB`) и создаётся некоторым пользователем (например, `newuser`), отыскивать идентификаторы раздела и пользователя вручную в соответствующих таблицах. При вставке значений даты и времени создания темы была использована встроенная функция `NOW()`, которая даёт текущие значения даты и времени.

После этого мы имеем в таблице `topic` следующие записи:

```
mysql> SELECT * FROM topic\G
*************************** 1. row ***************************
         id: 1
       name: Как написать функцию?
    content: Помогите.
 dtcreation: 2015-05-18 13:23:01
category_id: 26
    user_id: 2
*************************** 2. row ***************************
         id: 2
       name: Как написать запрос?
    content: Помогите!
 dtcreation: 2015-05-18 13:24:11
category_id: 44
    user_id: 2
*************************** 3. row ***************************
         id: 3
       name: Как отладить?
    content: Помогите!!
 dtcreation: 2015-05-18 13:26:19
category_id: 56
    user_id: 6
*************************** 4. row ***************************
         id: 4
       name: Как обновить?
    content: Не могу понять.
```

```
 dtcreation: 2015-05-18 13:29:44
category_id: 57
    user_id: 2
*************************** 5. row ***************************
         id: 5
       name: Подойдёт ли SQLite?
    content: Не знаю.
 dtcreation: 2015-05-18 13:37:40
category_id: 38
    user_id: 2
*************************** 6. row ***************************
         id: 6
       name: Предлагаю обсудить.
    content: Мне нравится.
 dtcreation: 2015-05-18 13:39:58
category_id: 61
    user_id: 1
*************************** 7. row ***************************
         id: 7
       name: Отличный редактор.
    content: Пользуюсь.
 dtcreation: 2015-05-18 13:49:15
category_id: 60
    user_id: 3
*************************** 8. row ***************************
         id: 8
       name: Как написать?
    content: Не получается.
 dtcreation: 2015-05-18 13:50:49
category_id: 44
    user_id: 6
*************************** 9. row ***************************
         id: 9
       name: Это хороший язык?
    content: Подскажите.
 dtcreation: 2015-05-18 13:56:59
category_id: 22
    user_id: 7
9 rows in set (0.00 sec)
```

Заполним данными таблицу comment. Предположим, пользователь admin отвечает на первую тему:

```
mysql> INSERT INTO comment
    -> (name, content, dtcreation, parent, topic_id, user_id)
    -> VALUES ('Это просто:','function имя (переменные): тип;',NOW(),0,1,1);
Query OK, 1 row affected (0.02 sec)
```

Ответ пользователя dummy на первый комментарий:

```
mysql> INSERT INTO comment
    -> (name, content, dtcreation, parent, topic_id, user_id)
    -> VALUES ('Спасибо.','Благодарю за подсказку.',NOW(),1,1,2);
```

```
Query OK, 1 row affected (0.01 sec)
```

Наконец, ответ пользователя admin на предыдущий комментарий:

```
mysql> INSERT INTO comment
    -> (name, content, dtcreation, parent, topic_id, user_id)
    -> VALUES ('Пожалуйста.','Обращайтесь ещё.',NOW(),2,1,1);
Query OK, 1 row affected (0.01 sec)
```

В таблице comment имеются теперь следующие записи (все записи являются здесь комментариями под одной и той же темой):

```
mysql> SELECT * FROM comment\G
*************************** 1. row ***************************
        id: 1
      name: Это просто:
   content: function имя (переменные): тип;
dtcreation: 2015-05-18 15:11:41
    parent: 0
  topic_id: 1
   user_id: 1
*************************** 2. row ***************************
        id: 2
      name: Спасибо.
   content: Благодарю за подсказку.
dtcreation: 2015-05-18 15:14:17
    parent: 1
  topic_id: 1
   user_id: 2
*************************** 3. row ***************************
        id: 3
      name: Пожалуйста.
   content: Обращайтесь ещё.
dtcreation: 2015-05-18 15:16:38
    parent: 2
  topic_id: 1
   user_id: 1
3 rows in set (0.00 sec)
```

Для некоторых полей при создании таблицы comment стоило бы задать значения по умолчанию, например, так:

```
name       = 'Re:'
parent     = 0
dtcreation = NOW()
```

Здесь имеется одна тонкость: в текущей версии MySQL из пакета для Debian GNU/Linux значение NOW() по умолчанию не может быть установлено для столбцов, которые имеют тип данных DATETIME (такая возможность появилась в более поздней версии MySQL). Но имеется другой тип данных такого же формата, как и DATETIME — TIMESTAMP. Имеется ряд принципиальных различий в работе MySQL с этими типами данных, например, допустимый диапазон значений у них кардинально различается. Но этими особенностями мы

можем в данном случае пока пренебречь и заменить тип данного столбца, попутно установив все нужные нам значения по умолчанию.

Внесём соответствующие изменения:

```
mysql> ALTER TABLE comment
    -> MODIFY name VARCHAR(45) NOT NULL DEFAULT 'Re:';
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE comment
    -> MODIFY parent INT UNSIGNED NOT NULL DEFAULT 0;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE comment
    -> MODIFY dtcreation TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP;
Query OK, 3 rows affected (0.09 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

CURRENT_TIMESTAMP имеет тот же смысл, что и значение, возвращаемое встроенной функцией NOW().

Текущее описание таблицы comment:

```
mysql> DESCRIBE comment\G
*************************** 1. row ***************************
  Field: id
   Type: int(10) unsigned
   Null: NO
    Key: PRI
Default: NULL
  Extra: auto_increment
*************************** 2. row ***************************
  Field: name
   Type: varchar(45)
   Null: NO
    Key:
Default: Re:
  Extra:
*************************** 3. row ***************************
  Field: content
   Type: text
   Null: NO
    Key:
Default: NULL
  Extra:
*************************** 4. row ***************************
  Field: dtcreation
   Type: timestamp
   Null: NO
    Key:
Default: CURRENT_TIMESTAMP
  Extra:
```

```
*************************** 5. row ***************************
  Field: parent
   Type: int(10) unsigned
   Null: NO
    Key:
Default: 0
  Extra:
*************************** 6. row ***************************
  Field: topic_id
   Type: int(10) unsigned
   Null: NO
    Key: MUL
Default: NULL
  Extra:
*************************** 7. row ***************************
  Field: user_id
   Type: int(10) unsigned
   Null: NO
    Key: MUL
Default: NULL
  Extra:
7 rows in set (0.00 sec)
```

Добавим ещё несколько записей в таблицу comment:

```
mysql> INSERT INTO comment
    -> (content, topic_id, user_id)
    -> VALUES ('А что именно не получается?',2,1);
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO comment
    -> (content, parent, topic_id, user_id)
    -> VALUES ('Ничего не выходит!!!',4,2,2);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO comment
    -> (content, parent, topic_id, user_id)
    -> VALUES ('Конкретизируйте вопрос.',5,2,1);
Query OK, 1 row affected (0.02 sec)

mysql> INSERT INTO comment
    -> (content, parent, topic_id, user_id)
    -> VALUES ('Пишите более конкретно.',5,2,3);
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO comment
    -> (content, parent, topic_id, user_id)
    -> VALUES ('Опишите проблему.',5,2,7);
Query OK, 1 row affected (0.00 sec)
```

Все записи в этой таблице:

```
mysql> SELECT * FROM comment\G
```

```
*************************** 1. row ***************************
        id: 1
      name: Это просто:
   content: function имя (переменные): тип;
dtcreation: 2015-05-18 15:11:41
    parent: 0
  topic_id: 1
   user_id: 1
*************************** 2. row ***************************
        id: 2
      name: Спасибо.
   content: Благодарю за подсказку.
dtcreation: 2015-05-18 15:14:17
    parent: 1
  topic_id: 1
   user_id: 2
*************************** 3. row ***************************
        id: 3
      name: Пожалуйста.
   content: Обращайтесь ещё.
dtcreation: 2015-05-18 15:16:38
    parent: 2
  topic_id: 1
   user_id: 1
*************************** 4. row ***************************
        id: 4
      name: Re:
   content: А что именно не получается?
dtcreation: 2015-05-18 19:33:27
    parent: 0
  topic_id: 2
   user_id: 1
*************************** 5. row ***************************
        id: 5
      name: Re:
   content: Ничего не выходит!!!
dtcreation: 2015-05-18 19:35:37
    parent: 4
  topic_id: 2
   user_id: 2
*************************** 6. row ***************************
        id: 6
      name: Re:
   content: Конкретизируйте вопрос.
dtcreation: 2015-05-18 19:37:13
    parent: 5
  topic_id: 2
   user_id: 1
*************************** 7. row ***************************
        id: 7
      name: Re:
```

```
   content: Пишите более конкретно.
dtcreation: 2015-05-18 19:38:46
    parent: 5
  topic_id: 2
   user_id: 3
*************************** 8. row ***************************
        id: 8
      name: Re:
   content: Опишите проблему.
dtcreation: 2015-05-18 19:39:49
    parent: 5
  topic_id: 2
   user_id: 7
8 rows in set (0.00 sec)
```

2.5. Выборка из нескольких таблиц

Выглядит последняя выборка недостаточно информативно. Вместо идентификаторов темы и пользователя желательно получить их названия и имена соответственно. Этого можно добиться с помощью выборки из трёх таблиц: `comment`, `topic` и `user`. Попутно здесь было произведено некоторое «украшательство» вывода, в частности, с использованием встроенной функции `CONCAT`, которая конкатенирует (соединяет) строки, передаваемые ей в качестве аргументов.

```
mysql> SELECT
    -> comment.id AS 'Comment #',
    -> comment.name AS Header,
    -> comment.content AS Text,
    -> comment.dtcreation AS 'Date & Time',
    -> CONCAT('На комментарий № ', comment.parent) AS 'Response',
    -> topic.name AS Theme,
    -> user.name AS User
    -> FROM comment, topic, user
    -> WHERE
    -> comment.user_id = user.id
    -> AND
    -> comment.topic_id = topic.id
    -> \G
*************************** 1. row ***************************
  Comment #: 1
     Header: Это просто:
       Text: function имя (переменные): тип;
Date & Time: 2015-05-18 15:11:41
   Response: На комментарий № 0
      Theme: Как написать функцию?
       User: admin
*************************** 2. row ***************************
  Comment #: 2
     Header: Спасибо.
       Text: Благодарю за подсказку.
Date & Time: 2015-05-18 15:14:17
```

```
   Response: На комментарий № 1
      Theme: Как написать функцию?
       User: dummy
*************************** 3. row ***************************
  Comment #: 3
     Header: Пожалуйста.
       Text: Обращайтесь ещё.
Date & Time: 2015-05-18 15:16:38
   Response: На комментарий № 2
      Theme: Как написать функцию?
       User: admin
*************************** 4. row ***************************
  Comment #: 4
     Header: Re:
       Text: А что именно не получается?
Date & Time: 2015-05-18 19:33:27
   Response: На комментарий № 0
      Theme: Как написать запрос?
       User: admin
*************************** 5. row ***************************
  Comment #: 5
     Header: Re:
       Text: Ничего не выходит!!!
Date & Time: 2015-05-18 19:35:37
   Response: На комментарий № 4
      Theme: Как написать запрос?
       User: dummy
*************************** 6. row ***************************
  Comment #: 6
     Header: Re:
       Text: Конкретизируйте вопрос.
Date & Time: 2015-05-18 19:37:13
   Response: На комментарий № 5
      Theme: Как написать запрос?
       User: admin
*************************** 7. row ***************************
  Comment #: 7
     Header: Re:
       Text: Пишите более конкретно.
Date & Time: 2015-05-18 19:38:46
   Response: На комментарий № 5
      Theme: Как написать запрос?
       User: hacker
*************************** 8. row ***************************
  Comment #: 8
     Header: Re:
       Text: Опишите проблему.
Date & Time: 2015-05-18 19:39:49
   Response: На комментарий № 5
      Theme: Как написать запрос?
       User: smartuser
```

```
8 rows in set (0.00 sec)
```

Полученный результат гораздо лучше того, что был выше. Тем не менее, он обладает тем недостатком, что если запись в таблице comment имеет нулевое значение parent, т. е., является ответом на исходное сообщение, а не на другой комментарий, то в выводе результатов запроса мы получаем бессмысленное «На комментарий № 0», в то время как желательно получить в этом случае «На тему Название_темы». Для этого можно воспользоваться CASE-выражениями. Это конструкция вида:

```
CASE
  WHEN условие1
    THEN выражение1
  WHEN условие2
    THEN выражение2
  ...
  WHEN условиеN
    THEN выражениеN
  [ELSE выражение]
END
```

Если ни одно из условий условие1, условие2,...,условиеN не является истинным, то возвращается то выражение, которое присутствует в секции ELSE (если такой секции нет, то в таком случае возвращается значение NULL). В противном случае возвращается то выражение, которое соответствует условию, принимающему истинное значение.

Используем CASE-выражение для построения нашего запроса:

```
mysql> SELECT
    -> comment.id AS 'Comment #',
    -> comment.name AS Header,
    -> comment.content AS Text,
    -> comment.dtcreation AS 'Date & Time',
    -> CASE
    -> WHEN comment.parent = 0
    -> THEN CONCAT('На тему ', '«', topic.name, '»')
    -> ELSE CONCAT('На комментарий № ', comment.parent)
    -> END Response,
    -> topic.name AS Theme,
    -> user.name AS User
    -> FROM comment, topic, user
    -> WHERE
    -> comment.user_id = user.id
    -> AND
    -> comment.topic_id = topic.id
    -> \G
*************************** 1. row ***************************
  Comment #: 1
     Header: Это просто:
       Text: function имя (переменные): тип;
Date & Time: 2015-05-18 15:11:41
   Response: На тему «Как написать функцию?»
      Theme: Как написать функцию?
       User: admin
```

```
*************************** 2. row ***************************
  Comment #: 2
     Header: Спасибо.
       Text: Благодарю за подсказку.
Date & Time: 2015-05-18 15:14:17
   Response: На комментарий № 1
      Theme: Как написать функцию?
       User: dummy
*************************** 3. row ***************************
  Comment #: 3
     Header: Пожалуйста.
       Text: Обращайтесь ещё.
Date & Time: 2015-05-18 15:16:38
   Response: На комментарий № 2
      Theme: Как написать функцию?
       User: admin
*************************** 4. row ***************************
  Comment #: 4
     Header: Re:
       Text: А что именно не получается?
Date & Time: 2015-05-18 19:33:27
   Response: На тему «Как написать запрос?»
      Theme: Как написать запрос?
       User: admin
*************************** 5. row ***************************
  Comment #: 5
     Header: Re:
       Text: Ничего не выходит!!!
Date & Time: 2015-05-18 19:35:37
   Response: На комментарий № 4
      Theme: Как написать запрос?
       User: dummy
*************************** 6. row ***************************
  Comment #: 6
     Header: Re:
       Text: Конкретизируйте вопрос.
Date & Time: 2015-05-18 19:37:13
   Response: На комментарий № 5
      Theme: Как написать запрос?
       User: admin
*************************** 7. row ***************************
  Comment #: 7
     Header: Re:
       Text: Пишите более конкретно.
Date & Time: 2015-05-18 19:38:46
   Response: На комментарий № 5
      Theme: Как написать запрос?
       User: hacker
*************************** 8. row ***************************
  Comment #: 8
     Header: Re:
```

```
       Text: Опишите проблему.
Date & Time: 2015-05-18 19:39:49
   Response: На комментарий № 5
      Theme: Как написать запрос?
       User: smartuser
8 rows in set (0.00 sec)
```

Помимо рассмотренного выше варианта CASE-выражения, имеется более простой:

```
CASE значение
  WHEN значение1
    THEN выражение1
  WHEN значение2
    THEN выражение2
  ...
  WHEN значениеN
    THEN выражениеN
  [ELSE выражение]
END
```

Работает он следующим образом: значение сравнивается со значениями значение1, значение2,...,значениеN и возвращается соответствующее выражение, для которого происходит совпадение значений. Если совпадений не произошло, то возвращается выражение из секции ELSE (которая не является обязательной, поэтому в последнем случае, когда отсутствуют совпадения значений из секций WHEN), возвращается NULL. Используем его для получения того же результата, что и в ранее рассмотренном примере:

```
mysql> SELECT
    -> comment.id AS 'Comment #',
    -> comment.name AS Header,
    -> comment.content AS Text,
    -> comment.dtcreation AS 'Date & Time',
    -> CASE comment.parent
    -> WHEN 0
    -> THEN CONCAT('На тему ', '«', topic.name, '»')
    -> ELSE CONCAT('На комментарий № ', comment.parent)
    -> END Response,
    -> topic.name AS Theme,
    -> user.name AS User
    -> FROM comment, topic, user
    -> WHERE comment.user_id = user.id
    -> AND
    -> comment.topic_id = topic.id
    -> \G
*************************** 1. row ***************************
  Comment #: 1
     Header: Это просто:
       Text: function имя (переменные): тип;
Date & Time: 2015-05-18 15:11:41
   Response: На тему «Как написать функцию?»
      Theme: Как написать функцию?
       User: admin
```

```
*************************** 2. row ***************************
  Comment #: 2
     Header: Спасибо.
       Text: Благодарю за подсказку.
Date & Time: 2015-05-18 15:14:17
   Response: На комментарий № 1
      Theme: Как написать функцию?
       User: dummy
*************************** 3. row ***************************
  Comment #: 3
     Header: Пожалуйста.
       Text: Обращайтесь ещё.
Date & Time: 2015-05-18 15:16:38
   Response: На комментарий № 2
      Theme: Как написать функцию?
       User: admin
*************************** 4. row ***************************
  Comment #: 4
     Header: Re:
       Text: А что именно не получается?
Date & Time: 2015-05-18 19:33:27
   Response: На тему «Как написать запрос?»
      Theme: Как написать запрос?
       User: admin
*************************** 5. row ***************************
  Comment #: 5
     Header: Re:
       Text: Ничего не выходит!!!
Date & Time: 2015-05-18 19:35:37
   Response: На комментарий № 4
      Theme: Как написать запрос?
       User: dummy
*************************** 6. row ***************************
  Comment #: 6
     Header: Re:
       Text: Конкретизируйте вопрос.
Date & Time: 2015-05-18 19:37:13
   Response: На комментарий № 5
      Theme: Как написать запрос?
       User: admin
*************************** 7. row ***************************
  Comment #: 7
     Header: Re:
       Text: Пишите более конкретно.
Date & Time: 2015-05-18 19:38:46
   Response: На комментарий № 5
      Theme: Как написать запрос?
       User: hacker
*************************** 8. row ***************************
  Comment #: 8
     Header: Re:
```

```
       Text: Опишите проблему.
Date & Time: 2015-05-18 19:39:49
   Response: На комментарий № 5
      Theme: Как написать запрос?
       User: smartuser
8 rows in set (0.00 sec)
```

2.6. Сортировка результатов выборки

Вывод результатов запроса можно сортировать по одному или нескольким критериям, например, по теме. Для этого используется ключевое слово ORDER BY:

```
mysql> SELECT
    -> comment.id AS 'Comment #',
    -> comment.name AS Header,
    -> comment.content AS Text,
    -> comment.dtcreation AS 'Date & Time',
    -> CASE
    -> WHEN comment.parent = 0
    -> THEN CONCAT('На тему ', '«', topic.name, '»')
    -> ELSE CONCAT('На комментарий № ', comment.parent)
    -> END Response,
    -> topic.name AS Theme,
    -> user.name AS User
    -> FROM comment, topic, user
    -> WHERE
    -> comment.user_id = user.id
    -> AND
    -> comment.topic_id = topic.id
    -> ORDER BY Theme
    -> \G
*************************** 1. row ***************************
  Comment #: 4
     Header: Re:
       Text: А что именно не получается?
Date & Time: 2015-05-18 19:33:27
   Response: На тему «Как написать запрос?»
      Theme: Как написать запрос?
       User: admin
*************************** 2. row ***************************
  Comment #: 5
     Header: Re:
       Text: Ничего не выходит!!!
Date & Time: 2015-05-18 19:35:37
   Response: На комментарий № 4
      Theme: Как написать запрос?
       User: dummy
*************************** 3. row ***************************
  Comment #: 6
     Header: Re:
       Text: Конкретизируйте вопрос.
```

```
Date & Time: 2015-05-18 19:37:13
   Response: На комментарий № 5
      Theme: Как написать запрос?
       User: admin
*************************** 4. row ***************************
  Comment #: 7
     Header: Re:
       Text: Пишите более конкретно.
Date & Time: 2015-05-18 19:38:46
   Response: На комментарий № 5
      Theme: Как написать запрос?
       User: hacker
*************************** 5. row ***************************
  Comment #: 8
     Header: Re:
       Text: Опишите проблему.
Date & Time: 2015-05-18 19:39:49
   Response: На комментарий № 5
      Theme: Как написать запрос?
       User: smartuser
*************************** 6. row ***************************
  Comment #: 1
     Header: Это просто:
       Text: function имя (переменные): тип;
Date & Time: 2015-05-18 15:11:41
   Response: На тему «Как написать функцию?»
      Theme: Как написать функцию?
       User: admin
*************************** 7. row ***************************
  Comment #: 2
     Header: Спасибо.
       Text: Благодарю за подсказку.
Date & Time: 2015-05-18 15:14:17
   Response: На комментарий № 1
      Theme: Как написать функцию?
       User: dummy
*************************** 8. row ***************************
  Comment #: 3
     Header: Пожалуйста.
       Text: Обращайтесь ещё.
Date & Time: 2015-05-18 15:16:38
   Response: На комментарий № 2
      Theme: Как написать функцию?
       User: admin
8 rows in set (0.00 sec)
```

Сортируем по теме и имени пользователя:

```
mysql> SELECT
    -> comment.id AS 'Comment #',
    -> comment.name AS Header,
    -> comment.content AS Text,
```

```
    -> comment.dtcreation AS 'Date & Time',
    -> CASE
    -> WHEN comment.parent = 0
    -> THEN CONCAT('На тему ', '«', topic.name, '»')
    -> ELSE CONCAT('На комментарий № ', comment.parent)
    -> END Response,
    -> topic.name AS Theme,
    -> user.name AS User
    -> FROM comment, topic, user
    -> WHERE
    -> comment.user_id = user.id
    -> AND
    -> comment.topic_id = topic.id
    -> ORDER BY Theme, User
    -> \G
*************************** 1. row ***************************
  Comment #: 4
     Header: Re:
       Text: А что именно не получается?
Date & Time: 2015-05-18 19:33:27
   Response: На тему «Как написать запрос?»
      Theme: Как написать запрос?
       User: admin
*************************** 2. row ***************************
  Comment #: 6
     Header: Re:
       Text: Конкретизируйте вопрос.
Date & Time: 2015-05-18 19:37:13
   Response: На комментарий № 5
      Theme: Как написать запрос?
       User: admin
*************************** 3. row ***************************
  Comment #: 5
     Header: Re:
       Text: Ничего не выходит!!!
Date & Time: 2015-05-18 19:35:37
   Response: На комментарий № 4
      Theme: Как написать запрос?
       User: dummy
*************************** 4. row ***************************
  Comment #: 7
     Header: Re:
       Text: Пишите более конкретно.
Date & Time: 2015-05-18 19:38:46
   Response: На комментарий № 5
      Theme: Как написать запрос?
       User: hacker
*************************** 5. row ***************************
  Comment #: 8
     Header: Re:
       Text: Опишите проблему.
```

```
Date & Time: 2015-05-18 19:39:49
   Response: На комментарий № 5
      Theme: Как написать запрос?
       User: smartuser
*************************** 6. row ***************************
  Comment #: 1
     Header: Это просто:
       Text: function имя (переменные): тип;
Date & Time: 2015-05-18 15:11:41
   Response: На тему «Как написать функцию?»
      Theme: Как написать функцию?
       User: admin
*************************** 7. row ***************************
  Comment #: 3
     Header: Пожалуйста.
       Text: Обращайтесь ещё.
Date & Time: 2015-05-18 15:16:38
   Response: На комментарий № 2
      Theme: Как написать функцию?
       User: admin
*************************** 8. row ***************************
  Comment #: 2
     Header: Спасибо.
       Text: Благодарю за подсказку.
Date & Time: 2015-05-18 15:14:17
   Response: На комментарий № 1
      Theme: Как написать функцию?
       User: dummy
8 rows in set (0.00 sec)
```

Сортируем по теме, ответу и имени пользователя:

```
mysql> SELECT
    -> comment.id AS 'Comment #',
    -> comment.name AS Header,
    -> comment.content AS Text,
    -> comment.dtcreation AS 'Date & Time',
    -> CASE
    -> WHEN comment.parent = 0
    -> THEN CONCAT('На тему ', '«', topic.name, '»')
    -> ELSE CONCAT('На комментарий № ', comment.parent)
    -> END Response,
    -> topic.name AS Theme,
    -> user.name AS User
    -> FROM comment, topic, user
    -> WHERE
    -> comment.user_id = user.id
    -> AND
    -> comment.topic_id = topic.id
    -> ORDER BY Theme, Response, User
    -> \G
*************************** 1. row ***************************
```

```
  Comment #: 5
     Header: Re:
       Text: Ничего не выходит!!!
Date & Time: 2015-05-18 19:35:37
   Response: На комментарий № 4
      Theme: Как написать запрос?
       User: dummy
*************************** 2. row ***************************
  Comment #: 6
     Header: Re:
       Text: Конкретизируйте вопрос.
Date & Time: 2015-05-18 19:37:13
   Response: На комментарий № 5
      Theme: Как написать запрос?
       User: admin
*************************** 3. row ***************************
  Comment #: 7
     Header: Re:
       Text: Пишите более конкретно.
Date & Time: 2015-05-18 19:38:46
   Response: На комментарий № 5
      Theme: Как написать запрос?
       User: hacker
*************************** 4. row ***************************
  Comment #: 8
     Header: Re:
       Text: Опишите проблему.
Date & Time: 2015-05-18 19:39:49
   Response: На комментарий № 5
      Theme: Как написать запрос?
       User: smartuser
*************************** 5. row ***************************
  Comment #: 4
     Header: Re:
       Text: А что именно не получается?
Date & Time: 2015-05-18 19:33:27
   Response: На тему «Как написать запрос?»
      Theme: Как написать запрос?
       User: admin
*************************** 6. row ***************************
  Comment #: 2
     Header: Спасибо.
       Text: Благодарю за подсказку.
Date & Time: 2015-05-18 15:14:17
   Response: На комментарий № 1
      Theme: Как написать функцию?
       User: dummy
*************************** 7. row ***************************
  Comment #: 3
     Header: Пожалуйста.
       Text: Обращайтесь ещё.
```

```
Date & Time: 2015-05-18 15:16:38
   Response: На комментарий № 2
      Theme: Как написать функцию?
       User: admin
*************************** 8. row ***************************
  Comment #: 1
     Header: Это просто:
       Text: function имя (переменные): тип;
Date & Time: 2015-05-18 15:11:41
   Response: На тему «Как написать функцию?»
      Theme: Как написать функцию?
       User: admin
8 rows in set (0.00 sec)
```

Можно сортировать в обратном порядке. Отсортируем таким образом по дате создания комментария (при этом, чем раньше сделан комментарий, тем он ниже в списке появляется):

```
mysql> SELECT
    -> comment.id AS 'Comment #',
    -> comment.name AS Header,
    -> comment.content AS Text,
    -> comment.dtcreation AS 'Date & Time',
    -> CASE
    -> WHEN comment.parent = 0
    -> THEN CONCAT('На тему ', '«', topic.name, '»')
    -> ELSE CONCAT('На комментарий № ', comment.parent)
    -> END Response,
    -> topic.name AS Theme,
    -> user.name AS User
    -> FROM comment, topic, user
    -> WHERE
    -> comment.user_id = user.id
    -> AND
    -> comment.topic_id = topic.id
    -> ORDER BY comment.dtcreation DESC
    -> \G
*************************** 1. row ***************************
  Comment #: 8
     Header: Re:
       Text: Опишите проблему.
Date & Time: 2015-05-18 19:39:49
   Response: На комментарий № 5
      Theme: Как написать запрос?
       User: smartuser
*************************** 2. row ***************************
  Comment #: 7
     Header: Re:
       Text: Пишите более конкретно.
Date & Time: 2015-05-18 19:38:46
   Response: На комментарий № 5
      Theme: Как написать запрос?
```

```
       User: hacker
*************************** 3. row ***************************
  Comment #: 6
     Header: Re:
       Text: Конкретизируйте вопрос.
Date & Time: 2015-05-18 19:37:13
   Response: На комментарий № 5
      Theme: Как написать запрос?
       User: admin
*************************** 4. row ***************************
  Comment #: 5
     Header: Re:
       Text: Ничего не выходит!!!
Date & Time: 2015-05-18 19:35:37
   Response: На комментарий № 4
      Theme: Как написать запрос?
       User: dummy
*************************** 5. row ***************************
  Comment #: 4
     Header: Re:
       Text: А что именно не получается?
Date & Time: 2015-05-18 19:33:27
   Response: На тему «Как написать запрос?»
      Theme: Как написать запрос?
       User: admin
*************************** 6. row ***************************
  Comment #: 3
     Header: Пожалуйста.
       Text: Обращайтесь ещё.
Date & Time: 2015-05-18 15:16:38
   Response: На комментарий № 2
      Theme: Как написать функцию?
       User: admin
*************************** 7. row ***************************
  Comment #: 2
     Header: Спасибо.
       Text: Благодарю за подсказку.
Date & Time: 2015-05-18 15:14:17
   Response: На комментарий № 1
      Theme: Как написать функцию?
       User: dummy
*************************** 8. row ***************************
  Comment #: 1
     Header: Это просто:
       Text: function имя (переменные): тип;
Date & Time: 2015-05-18 15:11:41
   Response: На тему «Как написать функцию?»
      Theme: Как написать функцию?
       User: admin
8 rows in set (0.00 sec)
```

При сортировке по нескольким полям возможно использование для одних полей сорти-

ровки по возрастанию значений, а для других — по убыванию.

2.7. Группировка результатов выборки из нескольких таблиц

Выведем имена пользователей, которые создавали темы, а также количество созданных ими тем, сгруппировав вывод по имени пользователя:

```
mysql> SELECT
    -> user.name AS Имя,
    -> COUNT(topic.user_id) AS 'Число тем'
    -> FROM user, topic
    -> WHERE topic.user_id = user.id
    -> GROUP BY user.name;
+-----------+-------------------+
| Имя       | Число тем         |
+-----------+-------------------+
| admin     |                 1 |
| dummy     |                 4 |
| hacker    |                 1 |
| newuser   |                 2 |
| smartuser |                 1 |
+-----------+-------------------+
5 rows in set (0.00 sec)
```

То же самое сделаем для пользователей и их комментариев:

```
mysql> SELECT
    -> user.name AS Имя,
    -> COUNT(comment.user_id) AS 'Число комментариев'
    -> FROM user, comment
    -> WHERE comment.user_id = user.id
    -> GROUP BY user.name;
+-----------+-------------------------------------+
| Имя       | Число комментариев                  |
+-----------+-------------------------------------+
| admin     |                                   4 |
| dummy     |                                   2 |
| hacker    |                                   1 |
| smartuser |                                   1 |
+-----------+-------------------------------------+
4 rows in set (0.00 sec)
```

Для самостоятельного решения

Упражнение 2.1. Добавить записи в таблицу `comment`, относящиеся к остальным темам из таблицы `topic`, используя рассмотренные выше способы.

Упражнение 2.2. Модифицировать ранее рассмотренный запрос с конструкцией `CASE`, так чтобы значение `Response` в тех случаях, когда заголовок комментария не заполняется

(т. е., поле name в таблице comment получает значение по умолчанию — Re:), приобретало вид: «Re: комментарий № номер_комментария», когда ответ делается на комментарий, либо «Re: название_темы» в противном случае, т. е., когда ответ делается на на саму тему.

Упражнение 2.3. Вывести названия тем, а также количество комментариев к этим темам, сгруппировав вывод по названию темы.

Глава 3

Операции соединения и объединения результатов запросов

3.1. Операция соединения JOIN, разновидности и применение

Составим запрос, в котором выберем все темы из таблицы topic и пользователей из таблицы user, которые эти темы создали, отсортировав результат по имени пользователя:

```
mysql> SELECT
    -> user.name AS Имя,
    -> topic.name AS Название
    -> FROM user, topic
    -> WHERE topic.user_id = user.id
    -> ORDER BY user.name;
+-----------+----------------------------------------+
| Имя       | Название                               |
+-----------+----------------------------------------+
| admin     | Предлагаю обсудить.                    |
| dummy     | Как написать запрос?                   |
| dummy     | Как написать функцию?                  |
| dummy     | Подойдёт ли SQLite?                    |
| dummy     | Как обновить?                          |
| hacker    | Отличный редактор.                     |
| newuser   | Как отладить?                          |
| newuser   | Как написать?                          |
| smartuser | Это хороший язык?                      |
+-----------+----------------------------------------+
9 rows in set (0.00 sec)
```

Стоит отметить, что в результат не попадает пользователь goodguy, поскольку он не создал ни одной темы.

Того же результата можно добиться использованием операции соединения таблиц. Имеется несколько разновидностей этой операции. Рассмотрим первый вариант — INNER JOIN:

```
mysql> SELECT
    -> user.name AS Имя,
    -> topic.name AS Название
    -> FROM user
    -> INNER JOIN topic
    -> ON topic.user_id = user.id
    -> ORDER BY user.name;
+-----------+----------------------------------------+
```

```
| Имя       | Название                                |
+-----------+-----------------------------------------+
| admin     | Предлагаю обсудить.                     |
| dummy     | Как написать запрос?                    |
| dummy     | Как написать функцию?                   |
| dummy     | Подойдёт ли SQLite?                     |
| dummy     | Как обновить?                           |
| hacker    | Отличный редактор.                      |
| newuser   | Как отладить?                           |
| newuser   | Как написать?                           |
| smartuser | Это хороший язык?                       |
+-----------+-----------------------------------------+
9 rows in set (0.00 sec)
```

Имена таблиц разделяются данным ключевым словом, после чего идёт условие (ON условие), по которому производится выборка. Выбраны будут только те данные из таблиц, для которых произошло выполнение указанного условия. Результат получен тот же, что и в запросе выше. Т. е., в результат попадают только те значения поля name из таблицы user, для которых нашлось значение поля user_id в таблице topic, равное значению поля id таблицы user из той же строки. Аналогично, в результат попадают только те значения поля name из таблицы topic, для которых нашлось значение поля id в таблице user, равное значению поля user_id таблицы topic из той же строки.

Если вместо INNER JOIN использовать LEFT JOIN, то в результате будут данные из всех записей первой таблицы, а также те данные для второй таблицы, для которых оказалось выполнено условие:

```
mysql> SELECT
    -> user.name AS Имя,
    -> topic.name AS Название
    -> FROM user
    -> LEFT JOIN topic
    -> ON topic.user_id = user.id
    -> ORDER BY user.name;
+-----------+-----------------------------------------+
| Имя       | Название                                |
+-----------+-----------------------------------------+
| admin     | Предлагаю обсудить.                     |
| dummy     | Как написать запрос?                    |
| dummy     | Как написать функцию?                   |
| dummy     | Подойдёт ли SQLite?                     |
| dummy     | Как обновить?                           |
| goodguy   | NULL                                    |
| hacker    | Отличный редактор.                      |
| newuser   | Как написать?                           |
| newuser   | Как отладить?                           |
| smartuser | Это хороший язык?                       |
+-----------+-----------------------------------------+
10 rows in set (0.00 sec)
```

Поскольку у пользователя goodguy нет созданных им тем, то во второй таблице нет совпадений по стоящему в запросе условию, в результате поэтому мы получаем для данного пользователя значение NULL в колонке Название.

Аналогично работает запрос с RIGHT JOIN, только таблицы меняются ролями:

```
mysql> SELECT
    -> user.name AS Имя,
    -> topic.name AS Название
    -> FROM user
    -> RIGHT JOIN topic
    -> ON topic.user_id = user.id
    -> ORDER BY user.name;
+-----------+----------------------------------------+
| Имя       | Название                               |
+-----------+----------------------------------------+
| admin     | Предлагаю обсудить.                    |
| dummy     | Как написать запрос?                   |
| dummy     | Как написать функцию?                  |
| dummy     | Подойдёт ли SQLite?                    |
| dummy     | Как обновить?                          |
| hacker    | Отличный редактор.                     |
| newuser   | Как отладить?                          |
| newuser   | Как написать?                          |
| smartuser | Это хороший язык?                      |
+-----------+----------------------------------------+
9 rows in set (0.06 sec)
```

Нетрудно заметить, что количество полученных результатов в последних двух запросах отличается на единицу, это объясняется тем, что у каждой темы есть автор, но не у каждого пользователя имеются созданные им темы. Поэтому в запросе с LEFT JOIN присутствует строка с goodguy, а в запросе с RIGHT JOIN — нет.

Поменяем местами таблицы в последнем запросе, оставив сортировку по имени пользователя:

```
mysql> SELECT
    -> topic.name AS Название,
    -> user.name AS Имя
    -> FROM topic
    -> RIGHT JOIN user
    -> ON topic.user_id = user.id
    -> ORDER BY user.name;
+----------------------------------------+-----------+
| Название                               | Имя       |
+----------------------------------------+-----------+
| Предлагаю обсудить.                    | admin     |
| Подойдёт ли SQLite?                    | dummy     |
| Как обновить?                          | dummy     |
| Как написать запрос?                   | dummy     |
| Как написать функцию?                  | dummy     |
| NULL                                   | goodguy   |
| Отличный редактор.                     | hacker    |
| Как написать?                          | newuser   |
| Как отладить?                          | newuser   |
| Это хороший язык?                      | smartuser |
+----------------------------------------+-----------+
10 rows in set (0.00 sec)
```

То же самое, но с сортировкой по названию темы:

```
mysql> SELECT
    -> topic.name AS Название,
    -> user.name AS Имя
    -> FROM topic
    -> RIGHT JOIN user
    -> ON topic.user_id = user.id
    -> ORDER BY topic.name;
+----------------------------------------+-----------+
| Название                               | Имя       |
+----------------------------------------+-----------+
| NULL                                   | goodguy   |
| Как написать запрос?                   | dummy     |
| Как написать функцию?                  | dummy     |
| Как написать?                          | newuser   |
| Как обновить?                          | dummy     |
| Как отладить?                          | newuser   |
| Отличный редактор.                     | hacker    |
| Подойдёт ли SQLite?                    | dummy     |
| Предлагаю обсудить.                    | admin     |
| Это хороший язык?                      | smartuser |
+----------------------------------------+-----------+
10 rows in set (0.00 sec)
```

Выведем имена всех пользователей, которые не создали ни одной темы на форуме, отсортировав по имени пользователя:

```
mysql> SELECT
    -> user.name AS Имя,
    -> topic.name AS Название
    -> FROM user
    -> LEFT JOIN topic
    -> ON topic.user_id = user.id
    -> WHERE topic.user_id IS NULL
    -> ORDER BY user.name;
+---------+------------------+
| Имя     | Название         |
+---------+------------------+
| goodguy | NULL             |
+---------+------------------+
1 row in set (0.00 sec)
```

IS NULL означает, что значение соответствующего поля равно NULL.

Найдём количество тем, созданных пользователем dummy:

```
mysql> SELECT COUNT(user.id) AS 'Число тем dummy'
    -> FROM user
    -> INNER JOIN topic
    -> ON topic.user_id = user.id
    -> WHERE user.name = 'dummy';
+-----------------------+
```

```
| Число тем dummy         |
+-------------------------+
|                       4 |
+-------------------------+
1 row in set (0.00 sec)
```

Есть ещё один вид соединения — CROSS JOIN, который представляет из себя, фактически, декартово произведение таблиц, которые запрос соединяет, т. е., каждой строке первой таблицы сопоставляется каждая строка второй:

```
mysql> SELECT
    -> user.name, topic.name
    -> FROM user
    -> CROSS JOIN topic;
+-----------+-----------------------------------------+
| name      | name                                    |
+-----------+-----------------------------------------+
| admin     | Как написать функцию?                   |
| dummy     | Как написать функцию?                   |
| goodguy   | Как написать функцию?                   |
| hacker    | Как написать функцию?                   |
| newuser   | Как написать функцию?                   |
| smartuser | Как написать функцию?                   |
| admin     | Как написать запрос?                    |
| dummy     | Как написать запрос?                    |
| goodguy   | Как написать запрос?                    |
| hacker    | Как написать запрос?                    |
| newuser   | Как написать запрос?                    |
| smartuser | Как написать запрос?                    |
| admin     | Как отладить?                           |
| dummy     | Как отладить?                           |
| goodguy   | Как отладить?                           |
| hacker    | Как отладить?                           |
| newuser   | Как отладить?                           |
| smartuser | Как отладить?                           |
| admin     | Как обновить?                           |
| dummy     | Как обновить?                           |
| goodguy   | Как обновить?                           |
| hacker    | Как обновить?                           |
| newuser   | Как обновить?                           |
| smartuser | Как обновить?                           |
| admin     | Подойдёт ли SQLite?                     |
| dummy     | Подойдёт ли SQLite?                     |
| goodguy   | Подойдёт ли SQLite?                     |
| hacker    | Подойдёт ли SQLite?                     |
| newuser   | Подойдёт ли SQLite?                     |
| smartuser | Подойдёт ли SQLite?                     |
| admin     | Предлагаю обсудить.                     |
| dummy     | Предлагаю обсудить.                     |
| goodguy   | Предлагаю обсудить.                     |
| hacker    | Предлагаю обсудить.                     |
| newuser   | Предлагаю обсудить.                     |
```

```
| smartuser | Предлагаю обсудить.                    |
| admin     | Отличный редактор.                     |
| dummy     | Отличный редактор.                     |
| goodguy   | Отличный редактор.                     |
| hacker    | Отличный редактор.                     |
| newuser   | Отличный редактор.                     |
| smartuser | Отличный редактор.                     |
| admin     | Как написать?                          |
| dummy     | Как написать?                          |
| goodguy   | Как написать?                          |
| hacker    | Как написать?                          |
| newuser   | Как написать?                          |
| smartuser | Как написать?                          |
| admin     | Это хороший язык?                      |
| dummy     | Это хороший язык?                      |
| goodguy   | Это хороший язык?                      |
| hacker    | Это хороший язык?                      |
| newuser   | Это хороший язык?                      |
| smartuser | Это хороший язык?                      |
+-----------+----------------------------------------+
54 rows in set (0.00 sec)
```

CROSS в запросе можно опустить, JOIN — тоже, поставив вместо последнего запятую. Результат будет идентичным. Данная операция обычно смысла не имеет, поскольку её результаты нельзя как-либо осмысленно интерпретировать. Использовать её на больших (по числу записей) таблицах надо с осторожностью, поскольку количество результатов выборки равно произведению числа записей в одной таблице на число записей в другой. Вообще в таких случаях стоит ограничивать количество результатов в выборке с помощью ключевого слова LIMIT:

```
mysql> SELECT
    -> user.name, topic.name
    -> FROM user
    -> CROSS JOIN topic
    -> LIMIT 10;
+-----------+----------------------------------------+
| name      | name                                   |
+-----------+----------------------------------------+
| admin     | Как написать функцию?                  |
| dummy     | Как написать функцию?                  |
| goodguy   | Как написать функцию?                  |
| hacker    | Как написать функцию?                  |
| newuser   | Как написать функцию?                  |
| smartuser | Как написать функцию?                  |
| admin     | Как написать запрос?                   |
| dummy     | Как написать запрос?                   |
| goodguy   | Как написать запрос?                   |
| hacker    | Как написать запрос?                   |
+-----------+----------------------------------------+
10 rows in set (0.00 sec)
```

3.2. Операция объединения UNION

Данная операция позволяет скомбинировать несколько запросов вместе и вывести один результат.

Предположим, нам нужно вывести заголовки всех комментариев вместе с заголовками тех комментариев, на которые они являются ответом, если комментарий написан в ответ на другой комментарий или, в противном случае, название соответствующей темы. Вначале будем выводить колонку с исходным сообщением, а потом ответ на него. Кроме того, в результат добавим: идентификатор родительского комментария (эту колонку сделаем самой первой) и идентификатор комментария (третья колонка). Такой запрос естественным образом разбивается на две части: для тех комментариев, которые являются ответами на саму тему (первый запрос) и для тех комментариев, которые являются ответами на другой комментарий (второй запрос). Составим вначале отдельно первый:

```
mysql> SELECT
    -> comment.parent AS P,
    -> topic.name AS Тема,
    -> comment.id AS C,
    -> comment.name AS Заголовок
    -> FROM topic, comment
    -> WHERE
    -> comment.topic_id = topic.id
    -> AND
    -> comment.parent = 0;
+---+----------------------------------------+---+----------------------+
| P | Тема                                   | C | Заголовок            |
+---+----------------------------------------+---+----------------------+
| 0 | Как написать функцию?                  | 1 | Это просто:          |
| 0 | Как написать запрос?                   | 4 | Re:                  |
+---+----------------------------------------+---+----------------------+
2 rows in set (0.00 sec)
```

Теперь второй:

```
mysql> SELECT
    -> p.id,
    -> p.name,
    -> c.id,
    -> c.name
    -> FROM comment p, comment c
    -> WHERE c.parent = p.id;
+----+----------------------+----+----------------------+
| id | name                 | id | name                 |
+----+----------------------+----+----------------------+
|  1 | Это просто:          |  2 | Спасибо.             |
|  2 | Спасибо.             |  3 | Пожалуйста.          |
|  4 | Re:                  |  5 | Re:                  |
|  5 | Re:                  |  6 | Re:                  |
|  5 | Re:                  |  7 | Re:                  |
|  5 | Re:                  |  8 | Re:                  |
+----+----------------------+----+----------------------+
6 rows in set (0.00 sec)
```

Наконец, консолидируем результаты выборки двух запросов в один:

```
mysql> (SELECT
    -> comment.parent AS P,
    -> topic.name AS Тема,
    -> comment.id AS C,
    -> comment.name AS Заголовок
    -> FROM topic, comment
    -> WHERE
    -> comment.topic_id = topic.id
    -> AND
    -> comment.parent = 0)
    -> UNION
    -> (SELECT
    -> p.id,
    -> p.name,
    -> c.id,
    -> c.name
    -> FROM comment p, comment c
    -> WHERE
    -> c.parent = p.id);
+---+----------------------------------------+---+----------------------+
| P | Тема                                   | C | Заголовок            |
+---+----------------------------------------+---+----------------------+
| 0 | Как написать функцию?                  | 1 | Это просто:          |
| 0 | Как написать запрос?                   | 4 | Re:                  |
| 1 | Это просто:                            | 2 | Спасибо.             |
| 2 | Спасибо.                               | 3 | Пожалуйста.          |
| 4 | Re:                                    | 5 | Re:                  |
| 5 | Re:                                    | 6 | Re:                  |
| 5 | Re:                                    | 7 | Re:                  |
| 5 | Re:                                    | 8 | Re:                  |
+---+----------------------------------------+---+----------------------+
8 rows in set (0.00 sec)
```

Отметим, что таблица с консолидированным результатом имеет те заголовки, что мы определили для первого запроса.

При использования данной операции, объединяемые запросы должны удовлетворять следующим условиям:

1) Запросы должны возвращать одно и то же количество столбцов.
2) Соответствующие столбцы должны иметь один и тот же тип данных (т. е., если, например, второй столбец из первого запроса имеет тип `VARCHAR(45)`, то такой же тип должен иметь второй столбец во втором запросе).
3) В результирующем наборе все строки уникальны (даже если при выполнении условий могут быть дублирующие). Для отмены такого поведения следует вместо `UNION` использовать `UNION ALL`.
4) Заключение в скобки объединяемых запросов не является обязательным помимо тех случаев, когда используется сортировка (`ORDER BY`) или ограничение по числу результатов выборки (`LIMIT`) в подзапросе.

5) Использование ORDER BY в подзапросах обычно не имеет смысла, поскольку операция объединения выдаёт результат, не обращая внимания на порядок: MySQL игнорирует все ORDER BY в подзапросах, если они используются без LIMIT.

Для самостоятельного решения

Упражнение 3.1. Вывести имена всех пользователей, которые оставили хотя бы один комментарий на форуме и заголовки соответствующих комментариев, отсортировав по имени пользователя и заголовку комментария.

Упражнение 3.2. Вывести названия всех тем, под которыми был оставлен хотя бы один комментарий и заголовки соответствующих комментариев, отсортировав по названию темы и заголовку комментария.

Упражнение 3.3. Вывести имена всех пользователей, которые не оставляли ни одного комментария на форуме, отсортировав по имени пользователя.

Упражнение 3.4. Вывести названия всех тем, под которыми не было оставлено ни одного комментария на форуме, отсортировав по названию темы.

Упражнение 3.5. Составить запрос для получения заголовков всех комментариев, на которые был дан хотя бы один ответ и заголовков ответов на них.

Упражнение 3.6. Вывести количество комментариев, которые были написаны под темой «Как написать функцию?».

Упражнение 3.7. Вывести количество пользователей, написавших хотя бы один комментарий под темой «Как написать запрос?».

Упражнение 3.8. Вывести имена пользователей, написавших хотя бы один комментарий или создавших хотя бы одну тему, а также название соответствующей темы или заголовок комментария.

Глава 4

Ссылочная целостность

Добавим ещё одну запись в таблицу user:

```
mysql> INSERT INTO user VALUES
    -> (NULL,'verybadguy','vbgpass','vbg@mail.ru','M');
Query OK, 1 row affected (0.01 sec)
```

Результат добавления:

```
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   |
|  8 | verybadguy | vbgpass   | vbg@mail.ru       | M   |
+----+------------+-----------+-------------------+-----+
7 rows in set (0.00 sec)
```

Изменим спецификации столбца dtcreation таблицы topic подобно тому, как было ранее сделано для таблицы comment:

```
mysql> ALTER TABLE topic
    -> MODIFY dtcreation TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP;
Query OK, 9 rows affected (0.04 sec)
Records: 9  Duplicates: 0  Warnings: 0
```

Теперь эта таблица имеет такой вид:

```
mysql> DESCRIBE topic\G
*************************** 1. row ***************************
  Field: id
   Type: int(10) unsigned
   Null: NO
    Key: PRI
Default: NULL
  Extra: auto_increment
*************************** 2. row ***************************
  Field: name
   Type: varchar(45)
   Null: NO
    Key:
```

```
Default: NULL
  Extra:
*************************** 3. row ***************************
  Field: content
   Type: text
   Null: NO
    Key:
Default: NULL
  Extra:
*************************** 4. row ***************************
  Field: dtcreation
   Type: timestamp
   Null: NO
    Key:
Default: CURRENT_TIMESTAMP
  Extra:
*************************** 5. row ***************************
  Field: category_id
   Type: int(10) unsigned
   Null: NO
    Key: MUL
Default: NULL
  Extra:
*************************** 6. row ***************************
  Field: user_id
   Type: int(10) unsigned
   Null: NO
    Key: MUL
Default: NULL
  Extra:
6 rows in set (0.00 sec)
```

Добавим тему от пользователя verybadguy:

```
mysql> INSERT INTO topic SET
    -> name = 'Помогите!!!',
    -> content = 'Помогите!!!',
    -> category_id = (
    -> SELECT id FROM category
    -> WHERE name = 'MySQL'
    -> ),
    -> user_id = (
    -> SELECT id FROM user
    -> WHERE name = 'verybadguy'
    -> );
Query OK, 1 row affected (0.01 sec)
```

Проверим результат добавления:

```
mysql> SELECT
    -> id, name
    -> FROM topic
```

```
    -> WHERE
    -> user_id = (
    -> SELECT id FROM user
    -> WHERE name = 'verybadguy'
    -> );
+----+--------------------+
| id | name               |
+----+--------------------+
| 10 | Помогите!!!        |
+----+--------------------+
1 row in set (0.00 sec)
```

Добавим от этого пользователя несколько комментариев к им же созданной теме. Первый комментарий:

```
mysql> INSERT INTO comment SET
    -> content = 'по',
    -> topic_id = (
    -> SELECT id FROM topic
    -> WHERE name = 'Помогите!!!'
    -> ),
    -> user_id = (
    -> SELECT id FROM user
    -> WHERE name = 'verybadguy'
    -> );
Query OK, 1 row affected (0.01 sec)
```

Результат добавления:

```
mysql> SELECT
    -> c.id,
    -> c.name AS Header,
    -> c.content AS Content,
    -> c.parent AS Parent,
    -> t.name AS Topic,
    -> u.name AS User
    -> FROM comment c
    -> INNER JOIN topic t
    -> ON topic_id = t.id
    -> INNER JOIN user u
    -> ON c.user_id = u.id
    -> WHERE u.id = (
    -> SELECT id
    -> FROM user
    -> WHERE name = 'verybadguy'
    -> )\G
*************************** 1. row ***************************
     id: 9
 Header: Re:
Content: по
 Parent: 0
  Topic: Помогите!!!
   User: verybadguy
```

```
1 row in set (0.00 sec)
```

Добавляем ещё один комментарий:

```
mysql> INSERT INTO comment SET
    -> content = 'мо',
    -> topic_id = (
    -> SELECT id FROM topic
    -> WHERE name = 'Помогите!!!'
    -> ),
    -> parent = 9,
    -> user_id = (
    -> SELECT id FROM user
    -> WHERE name = 'verybadguy'
    -> );
Query OK, 1 row affected (0.00 sec)
```

Результат добавления:

```
mysql> SELECT
    -> c.id,
    -> c.name AS Header,
    -> c.content AS Content,
    -> c.parent AS Parent,
    -> t.name AS Topic,
    -> u.name AS User
    -> FROM comment c
    -> INNER JOIN topic t
    -> ON topic_id = t.id
    -> INNER JOIN user u
    -> ON c.user_id = u.id
    -> WHERE u.id = (
    -> SELECT id
    -> FROM user
    -> WHERE name = 'verybadguy'
    -> )\G
*************************** 1. row ***************************
     id: 9
 Header: Re:
Content: по
 Parent: 0
  Topic: Помогите!!!
   User: verybadguy
*************************** 2. row ***************************
     id: 10
 Header: Re:
Content: мо
 Parent: 9
  Topic: Помогите!!!
   User: verybadguy
2 rows in set (0.00 sec)
```

Ещё добавляем:

```
mysql> INSERT INTO comment SET
    -> content = 'ги',
    -> topic_id = (
    -> SELECT id FROM topic
    -> WHERE name = 'Помогите!!!'
    -> ),
    -> parent = 10,
    -> user_id = (
    -> SELECT id FROM user
    -> WHERE name = 'verybadguy'
    -> );
Query OK, 1 row affected (0.01 sec)
```

Смотрим результат:

```
mysql> SELECT
    -> c.id,
    -> c.name AS Header,
    -> c.content AS Content,
    -> c.parent AS Parent,
    -> t.name AS Topic,
    -> u.name AS User
    -> FROM comment c
    -> INNER JOIN topic t
    -> ON topic_id = t.id
    -> INNER JOIN user u
    -> ON c.user_id = u.id
    -> WHERE u.id = (
    -> SELECT id
    -> FROM user
    -> WHERE name = 'verybadguy'
    -> )\G
*************************** 1. row ***************************
     id: 9
 Header: Re:
Content: по
 Parent: 0
  Topic: Помогите!!!
   User: verybadguy
*************************** 2. row ***************************
     id: 10
 Header: Re:
Content: мо
 Parent: 9
  Topic: Помогите!!!
   User: verybadguy
*************************** 3. row ***************************
     id: 11
 Header: Re:
Content: ги
 Parent: 10
  Topic: Помогите!!!
```

```
   User: verybadguy
3 rows in set (0.00 sec)
```

Последний комментарий добавляем:

```
mysql> INSERT INTO comment SET
    -> content = 'те!',
    -> topic_id = (
    -> SELECT id FROM topic
    -> WHERE name = 'Помогите!!!'
    -> ),
    -> parent = 11,
    -> user_id = (
    -> SELECT id FROM user
    -> WHERE name = 'verybadguy'
    -> );
Query OK, 1 row affected (0.00 sec)
```

Результат:

```
mysql> SELECT c.id,
    -> c.name AS Header,
    -> c.content AS Content,
    -> c.parent AS Parent,
    -> t.name AS Topic,
    -> u.name AS User
    -> FROM comment c
    -> INNER JOIN topic t
    -> ON topic_id = t.id
    -> INNER JOIN user u
    -> ON c.user_id = u.id
    -> WHERE u.id = (
    -> SELECT id
    -> FROM user
    -> WHERE name = 'verybadguy'
    -> )\G
*************************** 1. row ***************************
     id: 9
 Header: Re:
Content: по
 Parent: 0
  Topic: Помогите!!!
   User: verybadguy
*************************** 2. row ***************************
     id: 10
 Header: Re:
Content: мо
 Parent: 9
  Topic: Помогите!!!
   User: verybadguy
*************************** 3. row ***************************
     id: 11
 Header: Re:
```

```
Content: ги
 Parent: 10
  Topic: Помогите!!!
   User: verybadguy
*************************** 4. row ***************************
     id: 12
 Header: Re:
Content: те!
 Parent: 11
  Topic: Помогите!!!
   User: verybadguy
4 rows in set (0.00 sec)
```

Теперь число тем, созданных пользователем verybadguy, равно:

```
mysql> SELECT COUNT(*)
    -> FROM topic
    -> WHERE user_id = (
    -> SELECT id
    -> FROM user
    -> WHERE name = 'verybadguy'
    -> );
+----------+
| COUNT(*) |
+----------+
|        1 |
+----------+
1 row in set (0.00 sec)
```

А число созданных им комментариев:

```
mysql> SELECT COUNT(*)
    -> FROM comment
    -> WHERE user_id = (
    -> SELECT id
    -> FROM user
    -> WHERE name = 'verybadguy'
    -> );
+----------+
| COUNT(*) |
+----------+
|        4 |
+----------+
1 row in set (0.00 sec)
```

Если теперь попытаться удалить пользователя verybadguy:

```
mysql> DELETE FROM user
    -> WHERE name = 'verybadguy';
```

то получим ошибку выполнения такого запроса ввиду того, что в таблице topic имеется запись со значением внешнего ключа, равным идентификатору данного пользователя

(ранее нами были сделаны в этой таблице ограничения ссылочной целостности для этого внешнего ключа ON DELETE RESTRICT).

Но удалить тему, созданную пользователем verybadguy, можно, хотя к ней и имеются комментарии и, соответственно, в таблице comment имеются значения внешних ключей, равные идентификатору данной темы, поскольку правило ограничения ссылочной целостности мы ранее задали как ON DELETE CASCADE, поэтому вместе с темой будут удалены и все комментарии к ней:

```
mysql> DELETE FROM topic
    -> WHERE name = 'Помогите!!!';
Query OK, 1 row affected (0.00 sec)
```

Если теперь вновь проверить количество тем, созданных пользователем verybadguy, то получим:

```
mysql> SELECT COUNT(*)
    -> FROM topic
    -> WHERE user_id = (
    -> SELECT id
    -> FROM user
    -> WHERE name = 'verybadguy'
    -> );
+----------+
| COUNT(*) |
+----------+
|        0 |
+----------+
1 row in set (0.00 sec)
```

Аналогично для количества комментариев, созданных этим пользователем:

```
mysql> SELECT COUNT(*)
    -> FROM comment
    -> WHERE user_id = (
    -> SELECT id
    -> FROM user
    -> WHERE name = 'verybadguy'
    -> );
+----------+
| COUNT(*) |
+----------+
|        0 |
+----------+
1 row in set (0.00 sec)
```

Как нетрудно заметить, у рассматриваемого пользователя теперь нет ни одной темы и ни одного комментария.

Для самостоятельного решения

Упражнение 4.1. Предположим, на нашем форуме создана некая тема. К ней затем был написан некий комментарий, к первому комментарию был написан второй (как ответ на

первый), а в ответ на второй комментарий кто-то написал третий. Т. е., первый комментарий ссылается на саму тему, второй — на первый, а третий — на второй. Предположим, по каким-либо причинам второй комментарий был удалён. Тогда ссылка третьего комментария «повисает в воздухе» (значение поля `parent` равно несуществующему значению поля `id`). Как такую ситуацию можно исправить? Внести соответствующие изменения.

Упражнение 4.2. Создать тему, три комментария к ней, удалить второй комментарий и проверить, как работает решение, сделанное в предыдущем упражнении.

Глава 5

Представления

Представления — хранимые запросы, которые при обращении к ним производят некоторый результирующий набор данных. Представление можно считать виртуальной таблицей.

Предположим, мы хотим часто использовать запрос, выбирающий из таблицы `user` данные из двух столбцов: `name` и `email`. Это можно сделать следующим образом:

```
mysql> SELECT
    -> name AS Имя,
    -> email AS Почта
    -> FROM user;
+-----------+-------------------+
| Имя       | Почта             |
+-----------+-------------------+
| admin     | admin@uni.udm.ru  |
| dummy     | dummy@mail.ru     |
| hacker    | hacker@mail.ru    |
| goodguy   | goodguy@yandex.ru |
| newuser   | newuser@yandex.ru |
| smartuser | smart@mail.ru     |
+-----------+-------------------+
6 rows in set (0.01 sec)
```

Но можно составить представление и использовать его:

```
mysql> CREATE VIEW namemail AS
    -> SELECT
    -> name AS Имя,
    -> email AS Почта
    -> FROM user;
Query OK, 0 rows affected (0.01 sec)
```

Теперь запрос упрощается:

```
mysql> SELECT * FROM namemail;
+-----------+-------------------+
| Имя       | Почта             |
+-----------+-------------------+
| admin     | admin@uni.udm.ru  |
| dummy     | dummy@mail.ru     |
| hacker    | hacker@mail.ru    |
| goodguy   | goodguy@yandex.ru |
| newuser   | newuser@yandex.ru |
| smartuser | smart@mail.ru     |
+-----------+-------------------+
6 rows in set (0.00 sec)
```

Рассмотрим несколько более сложный пример: выбираем из таблиц `user` и `topic` имена пользователей и количество созданных ими тем, группируя по имени пользователя:

```
mysql> SELECT
    -> user.name AS Имя,
    -> COUNT(topic.id) AS 'Число тем'
    -> FROM user
    -> LEFT JOIN topic
    -> ON topic.user_id = user.id
    -> GROUP BY user.name;
+-----------+-------------------+
| Имя       | Число тем         |
+-----------+-------------------+
| admin     |                 1 |
| dummy     |                 4 |
| goodguy   |                 0 |
| hacker    |                 1 |
| newuser   |                 2 |
| smartuser |                 1 |
+-----------+-------------------+
6 rows in set (0.00 sec)
```

То же самое с созданием представления:

```
mysql> CREATE VIEW usertopic AS
    -> SELECT
    -> user.name AS Имя,
    -> COUNT(topic.id) AS 'Число тем'
    -> FROM user
    -> LEFT JOIN topic
    -> ON topic.user_id = user.id
    -> GROUP BY user.name;
Query OK, 0 rows affected (0.01 sec)
```

Выборка теперь выглядит следующим образом:

```
mysql> SELECT * FROM usertopic;
+-----------+-------------------+
| Имя       | Число тем         |
+-----------+-------------------+
| admin     |                 1 |
| dummy     |                 4 |
| goodguy   |                 0 |
| hacker    |                 1 |
| newuser   |                 2 |
| smartuser |                 1 |
+-----------+-------------------+
6 rows in set (0.00 sec)
```

Для самостоятельного решения

Упражнение 5.1. Написать представление (и сделать выборку с его помощью) для получения названий всех тем вместе с именами пользователей, их создавших.

Упражнение 5.2. Написать представление (и сделать выборку с его помощью) для получения заголовков всех комментариев вместе с именами пользователей, их создавших.

Упражнение 5.3. Написать представление (и сделать выборку с его помощью) для получения имён всех пользователей и количества комментариев, ими оставленных.

Глава 6

Транзакции

Транзакция — это набор инструкций SQL, который выполняется как единое целое, и при необходимости может быть отменён. Если все инструкции в транзакции были выполнены успешно, можно зафиксировать (COMMIT) их результат, и все изменения будут отражены в базе данных. В противном случае, можно «откатить» (ROLLBACK) всё назад для отмены произведённых действий. Любая инструкция, выполненная с начала транзакции (которое отмечается конструкцией START TRANSACTION) и до этого момента, не вносит каких-либо изменений в базу данных.

Рассмотрим пример. Начнём транзакцию:

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
```

Добавим в таблицу user ещё одного пользователя:

```
mysql> INSERT INTO user VALUES
    -> (NULL,'cleverguy','cleverpass','cleverguy@mail.ru','M');
Query OK, 1 row affected (0.00 sec)
```

Проверим наличие новой записи в таблице, запустив ещё один экземпляр MySQL monitor:

```
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   |
|  8 | verybadguy | vbgpass   | vbg@mail.ru       | M   |
+----+------------+-----------+-------------------+-----+
7 rows in set (0.00 sec)
```

Как нетрудно заметить, пользователь не был внесён в базу, поскольку мы не зафиксировали сделанные изменения.

Зафиксируем их:

```
mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)
```

Проверим снова:

```
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   |
|  8 | verybadguy | vbgpass    | vbg@mail.ru       | M   |
|  9 | cleverguy  | cleverpass | cleverguy@mail.ru | M   |
+----+------------+------------+-------------------+-----+
8 rows in set (0.00 sec)
```

Теперь новый пользователь появился в базе данных.

Сделаем теперь транзакцию с «откатом» внесённых изменений:

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
```

Добавляем нового пользователя в таблицу `user`:

```
mysql> INSERT INTO user VALUES
    -> (NULL,'stupidguy','stupidpass','stupid@yandex.ru','M');
Query OK, 1 row affected (0.00 sec)
```

Проверим наличие новой записи в таблице во втором экземпляре `MySQL monitor`:

```
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   |
|  8 | verybadguy | vbgpass    | vbg@mail.ru       | M   |
|  9 | cleverguy  | cleverpass | cleverguy@mail.ru | M   |
+----+------------+------------+-------------------+-----+
8 rows in set (0.00 sec)
```

Добавленной записи не появилось, поскольку транзакцию мы не завершили.

Делаем «откат»:

```
mysql> ROLLBACK;
Query OK, 0 rows affected (0.01 sec)
```

И вновь проверяем содержимое таблицы `user`:

```
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   |
|  8 | verybadguy | vbgpass    | vbg@mail.ru       | M   |
|  9 | cleverguy  | cleverpass | cleverguy@mail.ru | M   |
+----+------------+------------+-------------------+-----+
8 rows in set (0.00 sec)
```

Никаких изменений в ней не произошло ввиду сделанного «отката».

Для самостоятельного решения

Упражнение 6.1. С помощью транзакций создать две темы в таблице `topic`, зафиксировав первую транзакцию и «откатив» вторую. Проверить полученные результаты.

Глава 7

Индексы

7.1. Виды индексов и их создание

Создадим несколько индексов для наших таблиц. В каждой из них имеются первичные ключи, которые являются индексами. Кроме того, в таблице `user` поле `name` объявлено как `UNIQUE`, поэтому оно также проиндексировано.

Можно вывести все имеющиеся индексы во всех таблицах базы данных:

```
mysql> SHOW INDEX FROM user\G
*************************** 1. row ***************************
        Table: user
   Non_unique: 0
     Key_name: PRIMARY
 Seq_in_index: 1
  Column_name: id
    Collation: A
  Cardinality: 8
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
*************************** 2. row ***************************
        Table: user
   Non_unique: 0
     Key_name: name
 Seq_in_index: 1
  Column_name: name
    Collation: A
  Cardinality: 8
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
2 rows in set (0.00 sec)

mysql> SHOW INDEX FROM category\G
*************************** 1. row ***************************
        Table: category
   Non_unique: 0
     Key_name: PRIMARY
```

```
 Seq_in_index: 1
  Column_name: id
    Collation: A
  Cardinality: 63
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
1 row in set (0.01 sec)

mysql> SHOW INDEX FROM topic\G
*************************** 1. row ***************************
        Table: topic
   Non_unique: 0
     Key_name: PRIMARY
 Seq_in_index: 1
  Column_name: id
    Collation: A
  Cardinality: 9
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
*************************** 2. row ***************************
        Table: topic
   Non_unique: 1
     Key_name: user_id
 Seq_in_index: 1
  Column_name: user_id
    Collation: A
  Cardinality: 9
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
*************************** 3. row ***************************
        Table: topic
   Non_unique: 1
     Key_name: category_id
 Seq_in_index: 1
  Column_name: category_id
    Collation: A
  Cardinality: 9
     Sub_part: NULL
       Packed: NULL
```

```
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
3 rows in set (0.01 sec)

mysql> SHOW INDEX FROM comment\G
*************************** 1. row ***************************
        Table: comment
   Non_unique: 0
     Key_name: PRIMARY
 Seq_in_index: 1
  Column_name: id
    Collation: A
  Cardinality: 8
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
*************************** 2. row ***************************
        Table: comment
   Non_unique: 1
     Key_name: topic_id
 Seq_in_index: 1
  Column_name: topic_id
    Collation: A
  Cardinality: 4
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
*************************** 3. row ***************************
        Table: comment
   Non_unique: 1
     Key_name: user_id
 Seq_in_index: 1
  Column_name: user_id
    Collation: A
  Cardinality: 8
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
3 rows in set (0.00 sec)
```

Добавим индекс для поля `email` таблицы `user`. Очевидно, данное поле должно быть

уникальным, поэтому добавление индекса может быть сделано следующим образом (воспользовавшись конструкцией ALTER TABLE):

```
mysql> ALTER TABLE user ADD UNIQUE (email);
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Выведем вновь все индексы для данной таблицы:

```
mysql> SHOW INDEX FROM user\G
*************************** 1. row ***************************
        Table: user
   Non_unique: 0
     Key_name: PRIMARY
 Seq_in_index: 1
  Column_name: id
    Collation: A
  Cardinality: 8
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
*************************** 2. row ***************************
        Table: user
   Non_unique: 0
     Key_name: name
 Seq_in_index: 1
  Column_name: name
    Collation: A
  Cardinality: 8
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
*************************** 3. row ***************************
        Table: user
   Non_unique: 0
     Key_name: email
 Seq_in_index: 1
  Column_name: email
    Collation: A
  Cardinality: 8
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
3 rows in set (0.00 sec)
```

Нетрудно заметить, что на один индекс в таблице стало больше.
Добавим индекс для поля name таблицы topic:

```
mysql> ALTER TABLE topic ADD INDEX (name);
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Проверяем результат:

```
mysql> SHOW INDEX FROM topic\G
*************************** 1. row ***************************
        Table: topic
   Non_unique: 0
     Key_name: PRIMARY
 Seq_in_index: 1
  Column_name: id
    Collation: A
  Cardinality: 9
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
*************************** 2. row ***************************
        Table: topic
   Non_unique: 1
     Key_name: user_id
 Seq_in_index: 1
  Column_name: user_id
    Collation: A
  Cardinality: 9
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
*************************** 3. row ***************************
        Table: topic
   Non_unique: 1
     Key_name: category_id
 Seq_in_index: 1
  Column_name: category_id
    Collation: A
  Cardinality: 9
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
```

```
*************************** 4. row ***************************
        Table: topic
   Non_unique: 1
     Key_name: name
 Seq_in_index: 1
  Column_name: name
    Collation: A
  Cardinality: 9
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
4 rows in set (0.00 sec)
```

Добавим ещё индекс для поля name таблицы comment:

```
mysql> ALTER TABLE comment ADD INDEX (name);
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Сравниваем с тем, что было ранее:

```
mysql> SHOW INDEX FROM comment\G
*************************** 1. row ***************************
        Table: comment
   Non_unique: 0
     Key_name: PRIMARY
 Seq_in_index: 1
  Column_name: id
    Collation: A
  Cardinality: 8
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
*************************** 2. row ***************************
        Table: comment
   Non_unique: 1
     Key_name: topic_id
 Seq_in_index: 1
  Column_name: topic_id
    Collation: A
  Cardinality: 4
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
```

```
*************************** 3. row ***************************
        Table: comment
   Non_unique: 1
     Key_name: user_id
 Seq_in_index: 1
  Column_name: user_id
    Collation: A
  Cardinality: 8
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
*************************** 4. row ***************************
        Table: comment
   Non_unique: 1
     Key_name: name
 Seq_in_index: 1
  Column_name: name
    Collation: A
  Cardinality: 8
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
4 rows in set (0.00 sec)
```

7.2. Альтернативный синтаксис. Удаление индексов

Индексы можно и удалять, если по-какой либо причине от того или иного индекса было решено отказаться. Синтаксис с использованием конструкции ALTER TABLE аналогичен синтаксису для добавления индекса, только вместо ключевого слова ADD используется DROP. Удалим только что созданный индекс из таблицы topic:

```
mysql> ALTER TABLE topic DROP INDEX name;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

С созданным индексом для таблицы комментариев поступим так же, но воспользуемся альтернативным синтаксисом:

```
mysql> DROP INDEX name ON comment;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Первичный ключ с помощью такого синтаксиса удалить нельзя (это можно сделать только с использованием ALTER TABLE).

Теперь вновь создадим только что удалённые индексы, но с использованием альтернативного синтаксиса. Для таблицы topic:

```
mysql> CREATE INDEX name ON topic (name);
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Для таблицы comment:

```
mysql> CREATE INDEX name ON topic (name);
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Так же, как и при удалении, альтернативный синтаксис не может использоваться для создания первичного ключа. Также стоит отметить, что опускать имя индекса при использовании альтернативного синтаксиса нельзя; кроме того, в отличие от синтаксиса с ALTER TABLE, нельзя создать (или удалить) сразу несколько индексов одним запросом. Любой вид индекса может быть создан на стадии создания таблицы (CREATE TABLE), как ранее было сделано при создании первичных ключей для всех таблиц нашей базы данных.

Для самостоятельного решения

Упражнение 7.1. Можно ли было определить уникальными какие-то из тех индексов, что мы добавили выше как неуникальные? Почему?

Упражнение 7.2. Имеет ли смысл какие-то ещё столбцы в базе проиндексировать? Если да, то как именно — как уникальные или нет?

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.