GeekBrains

Базы данных. Интерактивный курс

Урок 7

Администрирование MySQL

Параметры запуска сервера

Сервер MySQL — сложный программный продукт. Он должен работать одинаково эффективно в совершенно разных условиях и решать задачи различного характера. Поэтому сервер имеет гибкие настройки, которые позволяют настроить его под свои нужды без перекомпиляции исходного кода.

У сервера MySQL есть два главных способа настройки: это параметры запуска, которые определяют режимы его работы, и переменные состояния, с помощью которых можно его масштабировать.

Часть из переменных состояния могут быть изменены при помощи параметров запуска. В разных операционных системах для просмотра параметров запуска используются разные средства. Например в UNIX-подобных операционных системах для этого используется команда:

```
ps -aux | grep mysqld
```

Мы можем видеть параметр **--basedir=/usr/local/opt/mysql**.

Каталог данных: **--datadir=/usr/local/var/mysql**.

Лог ошибок **--log-error=mac422rih.local.err**.

Pid файл **--pid-file=mac422rih.local.pid**.

В Windows посмотреть начальные параметры можно в свойствах сервиса, при помощи которого осуществляется управление сервером MySQL. Параметров сервера MySQL достаточно много, поэтому их может накапливаться изрядное количество. Просмотреть их полный список можно, передав серверу MySQL параметры **--verbose --help**.

У части параметров есть сокращения: они начинаются с одного дефиса, а не с двух. Например, параметр **--user** имеет сокращение **-u**.

Конфигурационный файл my.cnf

Параметры запуска можно не прописывать непосредственно в строке запуска сервера, а поместить в конфигурационный файл my.cnf, из которого сервер прочитает их при старте:

- /etc/my.cnf
- /etc/mysql/my.cnf
- $MYSQL_HOME/my.cnf
- [datadir]/my.cnf
- ~/.my.cnf

Следует отметить, что внутри конфигурационного файла **my.cnf** параметры указываются без предваряющих дефисов (они не нужны, так как каждый параметр располагается на отдельной строке). Если строка начинается с символа диеза **#** или точки с запятой **;**, то вся строка относится к комментарию. Параметры в конфигурационном файле называют директивами.

В операционной системе Windows конфигурационный файл может иметь как расширение **ini** (такой файл может располагаться в директориях **C:\Windows\my.ini** и в **C:\mysql5\my.ini**), так и расширение **cnf**. В UNIX-подобных операционных системах конфигурационный файл имеет, как правило, расширение **cnf**.

Название секции	Описание
[mysqld]	Сервер MySQL
[server]	Сервер MySQL
[mysqld-5.7]	Сервер MySQL версии 5.7
[mysqld-5.8]	Сервер MySQL версии 5.8
[client]	Любая клиентская утилита
[mysql]	Консольный клиент mysql
[mysqldump]	Утилита создания дампов mysqldump

Конфигурационный файл влияет на работу не только сервера MySQL, но и вспомогательных утилит, таких как консольный клиент mysql, утилита создания SQL-дампов **mysqldump** и т.п. Более того, один конфигурационный файл может управлять работой нескольких серверов MySQL. Поэтому содержимое конфигурационного файла разделено на секции, которые имеют вид [имя_секции].

Имя секции определяют утилиту или сервер, к которой будут относиться перечисленные ниже директивы, до тех пор, пока не встретится новая секция или конец файла.

Наличие секции **[mysqld]** и специальных секций для разных версий обусловлены тем, что с каждой новой версией появляется всё больше и больше параметров запуска, и, если конфигурационный файл управляет несколькими серверами, то ряд директив будут одинаковыми для всех серверов, а ряд — уникальны для каждой из версий.

Точно такая же ситуация с секцией [client] и секциями для каждой из утилит. Дело в том, что каждая из утилит обладает сходными параметрами (например, параметры соединения с серверами у всех одинаковые) и в то же время имеет уникальные параметры, характерные только для её функциональных возможностей.

```
/usr/local/etc/my.cnf

# Порт TCP/IP который прослушивает MySQL сервер, порт 3306 является
# стандартным, однако можно назначить любой другой не занятый порт
port=3307
```

Давайте запустим MySQL на порту, отличном от 3306. Для остановки сервера можно использовать команду:

```
sudo /usr/local/opt/mysql/support-files/mysql.server stop
```

Для повторного запуска:

```
sudo /usr/local/opt/mysql/support-files/mysql.server start
```

для перезапуска можно использовать команду restart

```
sudo /usr/local/opt/mysql/support-files/mysql.server restart
```

В дистрибутиве Linux Ubuntu для остановки сервера можно воспользоваться командой:

```
service mysql stop
```

Для запуска:

```
service mysql start
```

Для перезапуска:

```
service mysql start
```

В Windows проще всего воспользоваться оснасткой сервисов. В сервисах выбрать тот, который был заведен при установке MySQL и в контекстном меню выбрать соответствующий пункт на запуск, остановку или перезапуск сервера.

Параметрам в конфигурационном файле, как правило, выставлены разумные значения. Поэтому большинство из них можно вообще не упоминать в конфигурационном файле.

Переменные сервера

Ознакомиться с текущими значениями параметров можно при помощи команды **SHOW VARIABLES**. Например, формат даты и времени заданы переменным **date_format**, **datetime_format** и **time_format**:

```
SHOW VARIABLES LIKE 'date%_format';
SHOW VARIABLES LIKE 'time%_format';
```

Текущий часовой пояс берется из операционной системы:

```
SHOW VARIABLES LIKE 'system_time_zone';
```

Однако при необходимости может быть изменен в конфигурационном файле **my.cnf**.

На одном из предыдущих уроков мы с вами знакомились с уровнями изоляции. Узнать текущий уровень мы можем при помощи следующей команды:

```
SHOW VARIABLES LIKE 'tx_isolation';
```

А используя директиву **tx_isolation**, можно изменить уровень изоляции в конфигурационном файле **my.cnf**.

Чаще всего изменяются параметры различных буферов и кэшей. Например, переменная **tmp_table_size** позволяет узнать максимальный размер временной таблицы. Если промежуточная

или временная таблица не помещается в этот размер, вместо оперативной памяти она будет размещена на жестком диске.

```
SHOW VARIABLES LIKE 'tmp_table_size';
```

Давайте изменим этот размер. В конфигурационном файле не обязательно прописывать размер вплоть до байта. Мы можем использовать сокращения, килобайты, мегабайты и гигабайты.

```
tmp_table_size=32M
```

Следует следить, чтобы суффиксы набирались в английской раскладке, иначе сервер не сможет перезапуститься. Итак давайте перезагрузим сервер

```
sudo /usr/local/opt/mysql/support-files/mysql.server restart
```

Запрашиваем размер временной таблицы:

```
SHOW VARIABLES LIKE 'tmp_table_size';
```

Изменить размер можно так же при помощи команды SET

```
SET SESSION tmp_table_size = 16777216;
```

Здесь мы устанавливаем размер оперативной памяти, отводимой под временные таблицы, равный 16M, только в рамках текущей сессии. Если мы выйдем из клиента, снова зайдем и запросим результат, сессионные данные обнулятся:

```
SHOW VARIABLES LIKE 'tmp_table_size';
```

Мы можем установить значение для всего сервера, используя ключевое слово GLOBAL

```
SET GLOBAL tmp_table_size = 16777216;
```

Если мы после этого выйдем из клиента и снова зайдем, значение уже не изменится, оно будет сброшено только после перезапуска сервера.

Давайте посмотрим еще на одну системную переменную — **auto_increment_increment**. Системная переменная предназначена для установки приращения в механизме **AUTO_INCRENENT** в таблицах.

```
SHOW VARIABLES LIKE 'auto_increment_increment';
```

Переменная может принимать значения от 1 до 65535 и по умолчанию получает значение 1. Установка значение переменной, например, в 10 приводит к тому, что значение автоинкрементного счётчика увеличивается не на единицу, а сразу на 10.

Не все переменные можно устанавливать глобально или сессионно. Какие из них в каком режиме можно устанавливать, следует уточнять в документации.

Журнальные файлы

MySQL поддерживает несколько видов журнальных файлов, в которых регистрируются различные события, происходящие на MySQL сервере. В журнал ошибок помещаются сообщения обо всех ошибках, происходящих во время запуска, работы или остановки MySQL сервера.

Общий журнал запросов позволяет регистрировать все выполненные запросы.

Бинарный журнал регистрирует все операторы, которые приводят к изменению данных. Данный журнал используется для репликации и восстановления данных.

В журнал медленных запросов заносятся все запросы, выполнение которых потребовало больше времени, чем указано в системной переменной **long_query_time**. По умолчанию это 10 секунд.

Среди этих файлов только бинарный журнал не является текстовым, для его просмотра необходима специальная утилита **mysqlbinlog**. Все остальные файлы текстовые и могут быть просмотрены обычными средствами.

Журнал ошибок

Давайте начнем изучение с журнала ошибок. Получить путь к нему можно, запросив переменную **log_error**:

```
SHOW VARIABLES LIKE 'log_error';
```

Как видим, журнальный файл ошибок находится в каталоге данных. Давайте выйдем из диалогового режима MySQL в консоль командной строки и перейдем в каталог данных:

```
cd /usr/local/var/mysql/
ls -la
```

Здесь мы можем обнаружить err-файл **mac422rih.local.err**. У вас название файла может отличаться, так как по умолчанию его имя совпадает с именем компьютера. Посмотрим последние 20 строк этого файла при помощи команды **tail** (можно использовать любое другое средство просмотра):

```
tail -20 mac422rih.local.err
```

Здесь мы можем видеть последние сообщения от сервера. В штатном режиме тут будут информационные сообщения, сопровождающие перезапуск сервера. Однако, если сервер не стартует именно отсюда, следует начинать расследование причин неполадок.

Общий журнал запросов позволяет регистрировать все соединения клиентов и их SQL-запросы в текстовом формате.

```
SHOW VARIABLES LIKE 'general_log%';
```

По соображениям производительности журнал запросов отключен. Для того, чтобы его включить его ведение в конфигурационном файле my.cnf

```
general_log=ON
```

Теперь нужно перезагрузить сервер:

```
sudo /usr/local/opt/mysql/support-files/mysql.server restart
```

Журнал медленных запросов

Журнал медленных запросов по умолчанию тоже отключен:

```
SHOW VARIABLES LIKE 'slow_query_log%';
SHOW VARIABLES LIKE 'long_query_time';
```

Запрос считается медленным, если время его выполнения превысило значение, заданное параметром **long_query_time**. Значение этого параметра можно при желании изменить в конфигурационном файле или при помощи команды **SET**.

```
slow_query_log=ON
```

Команда записывается в журнал медленных запросов только после того, как она была выполнена, и после того, как сняты все блокировки, поэтому порядок размещения операторов в журнале может отличаться от порядка, в котором они выполнялись.

Журнал медленных запросов может использоваться с целью выявления запросов, на выполнение которых ушло слишком много времени, а значит требующих оптимизации.

Давайте воспользуемся функцией BENCHMARK(), чтобы организовать медленный запрос. Функция принимает в качестве первого аргумента число, в качестве второго — запрос. Этот запрос выполняется столько раз, сколько указано в первом аргументе:

```
SELECT BENCHMARK(1000000000, (SELECT COUNT(*) FROM products));
```

Итак, запрос выполняется 25 секунд, давайте посмотрим содержимое журнала медленных запросов.

```
cat mac422rih-slow.log
```

Здесь у нас только один запрос, когда их много, разобраться в журнале медленных запросов не так просто. Поэтому предварительно лучше прогнать журнал через команду **mysqldumpslow**, которая отобразит список всех встречающихся в журнале запросов.

```
mysqldumpslow mac422rih-slow.log
```

По умолчанию сервер размещает журналы общих и медленных запросов в файлах. Однако журналы могут быть записаны и в специальные таблицы **general_log** и **slow_log** системной базы данных **mysql**.

```
SELECT * FROM mysql.general_log;
SELECT * FROM mysql.slow_log;
```

Сейчас таблицы пусты. MySQL принимает решение о том размещать журналы в файлах или в базе данных, ориентируясь на значение системной переменной **log_output.**

```
SHOW VARIABLES LIKE 'log_output';
```

По умолчанию переменная принимаете значение **FILE:**

- TABLE
- FILE
- NONE

Однако журналы могут быть записаны как в файл, так и в специальные таблицы **general_log** и **slow_log** системной базы данных **mysql**. Если присвоить переменной значение **TABLE**, записи журналов общих и медленных запросов будут помещаться в таблицах **general_log** и **slow_log** базы данных **mysql**. Если присвоить значение **FILE**, как сейчас, записи будут вестись в текстовые файлы. Если присвоить значение **NONE**, записи не сохраняются.

Давайте отредактируем конфигурационный файл **my.cnf**:

```
log_output=TABLE
```

Снова выполним наш медленный запрос:

```
SELECT BENCHMARK(1000000000, (SELECT COUNT(*) FROM products));
```

Обратимся к таблице slow_log базы данных mysql:

```
SELECT * FROM mysql.slow_log\G
```

Теперь все запросы будут фиксироваться в системной базы данных **mysql**, а не в файлах каталога данных.

Бинарный журнал

Бинарный журнал регистрирует все запросы, которые приводят к изменению состояния базы данных. Однако в журнал заносятся только те запросы, которые действительно обновляют данные.

Так, операторы **UPDATE** и **DELETE**, конструкция **WHERE** которых не соответствует ни одной строке, в журнале не регистрируются. Также пропускаются даже те операторы **UPDATE**, которые присваивают столбцу уже имеющиеся у него значение.

Основное назначение бинарного журнала — предоставить возможность обновить базу данных во время операции восстановления настолько подробно, насколько это возможно. Так как именно в

бинарном журнале будут содержаться все изменения, произошедшие после того, как была сделана резервная копия всей системы.

Обычно резервную копию с базы данных снимают раз в сутки. Если происходит сбой, база данных восстанавливается из резервной копии, а изменения, которые были сделаны с момента возникновения сбоя беруться из бинарного журнала.

Узнать, включены ли бинарные логи, можно при помощи переменной **log_bin**:

```
SHOW VARIABLES LIKE 'log_bin';
```

Давайте их включим в конфигурационном файле my.cnf

```
log_bin=ON
server-id=1
```

Кроме того, нам потребуется присвоить серверу уникальный идентификатор при помощи директивы **server-id**. Этот идентификатор используется, чтобы отличать один сервер от другого, когда они связаны репликацией. Репликацию мы более подробно рассмотрим далее на уроке.

В каталоге данных у нас появилось два файла:

- ON.index
- ON.000001

Посмотрим содержимое индексного файла.

```
cat ON.index
```

Индексный файл содержит список бинарных файлов, которых может быть несколько. Сервер MySQL присоединяет в конец имени файла бинарного журнала расширение в виде номера. Этот номер увеличивается каждый раз при запуске сервера или выполнении команд по очистке журналов.

Новый бинарный журнал создается автоматически, когда размер текущего журнала достигает максимума, заданного в системной переменной **max_binlog_size**. В бинарный журнал записываются все изменения базы данных.

Давайте запустим на выполнение SQL-дамп **shop.sql**:

```
ls -la | grep ON
```

Размер файла **ON.000001** вырос. Файл бинарный, просто так посмотреть его содержимое мы не можем. Однако мы можем преобразовать его в текстовую форму при помощи утилиты **mysqlbinlog**:

```
mysqlbinlog ON.000001
```

Мы получаем тестовое содержимое, готовое к выполнению в клиенте **mysql**. Вся вспомогательная информация оформлена в виде SQL-комментариев.

Соединения

Количество соединений, которые может одновременно обслуживать MySQL, ограничено системной переменной **max_connections**:

```
SHOW VARIABLES LIKE 'max_connections';
```

При желании ее можно изменить в конфигурационном файле **my.cnf**. Однако делать это следует осмотрительно, так как как каждое соединение MySQL потребляет довольно много оперативной памяти. Иногда это мегабайты или даже десятки мегабайт на одно соединение. Если выставить большое значение и количество соединений будет слишком велико, можно исчерпать всю оперативную память сервера. Отслеживать текущие соединения можно при помощи команды:

```
SHOW PROCESSLIST;
```

Каждая строка результирующей таблицы, возвращаемой оператором **SHOW PROCESSLIST**, соответствует одному соединению с сервером.

Столбец **Time** указывает время соединения. Большое время, как тут, обычно указывает на постоянные соединения от консольных и графических клиентов, либо на зависшие или заблокированные запросы, которые ожидают освобождения ресурсов.

В столбце **Info** указывается статус соединения. Если мы хотим снять какое-то соединение, мы можем воспользоваться командой **KILL**, передав ей идентификатор соединения из первого столбца:

```
KILL 1
```

Права пользователей

СУБД MySQL — многопользовательская среда. Это означает, что для доступа к таблицам базы данных могут быть созданы различные учетные записи с разным уровнем привилегий. Создать новую учетную запись можно при помощи оператора **CREATE USER**:

```
CREATE USER foo;
```

Если пароль не указывается, по умолчанию в его качестве выступает пустая строка. После того, как новая учетная запись создана, в любом клиенте можно авторизоваться с именем пользователя.

```
$ mysql -u foo
mysql> SELECT USER();
```

Чтобы задать пароль, нужно использовать ключевое слово **IDENTIFIED BY**, за которым в одиночных кавычках следует пароль.

```
CREATE USER shop IDENTIFIED WITH sha256_password BY 'pass';
$ mysql -u shop -p
mysql> SELECT USER();
```

Однако созданные при помощи оператора **CREATE USER** учетные записи не обладают привилегиями. С использованием такой учетной записи невозможно просматривать таблицы и осуществлять запросы. Для наделения учетной записи привилегиями необходимо воспользоваться оператором **GRANT**. Получить список пользователей можно, обратившись к таблице users системной базы данных shop:

```
SELECT Host, User FROM mysql.user;
```

Оператор **DROP USER** позволяет удалить учетную запись:

```
DROP USER foo;
SELECT Host, User FROM mysql.user;
```

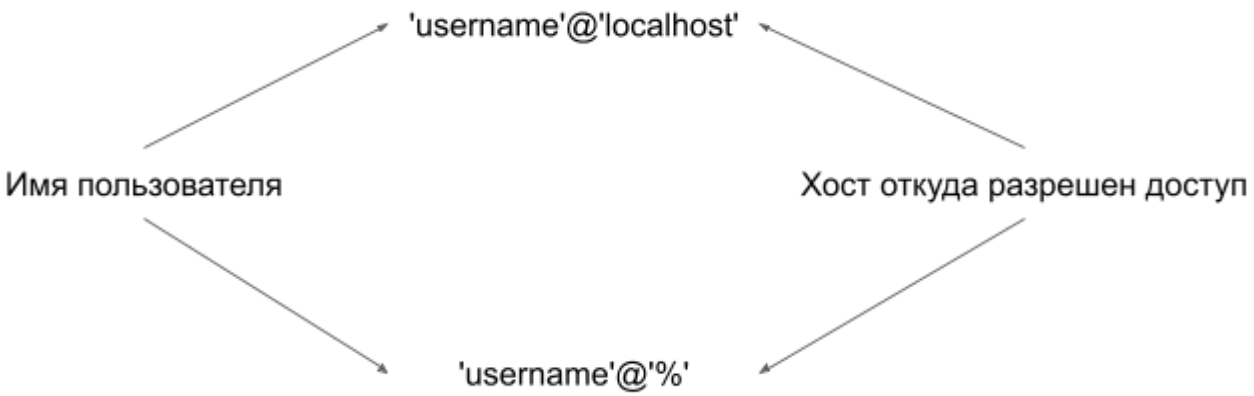
Оператор **DROP USER** не закрывает автоматически соединение удаляемого пользователя, который сможет работать с базой данных, до тех пор, пока не будет закрыто его соединение или связь с сервером не оборвется.

Изменять имя уже существующей учётной записи можно при помощи оператора **RENAME USER**:

```
RENAME USER shop TO foo;
SELECT Host, User FROM mysql.user;
```

Учетная запись

Следует отметить, что учетная запись является составной и принимает форму **'username'@'host'**, где **username** — имя, а **host** — наименование хоста, с которого пользователю **username** разрешено обращаться к серверу MySQL.



Если к IP-адресу хоста привязано доменное имя, его можно использовать вместо этого хоста.

Количество адресов, с которых необходимо обеспечить доступ пользователю, может быть значительным и включать целые диапазоны. Для задания диапазона в имени хоста используется специальный символ **%**, выполняющий ту же функцию, что и в операторе **LIKE**. Так, учетная запись **'username'@'%'** разрешает пользователю **'username'** обращаться к серверу MySQL с любых компьютеров сети.

Управление привилегиями

Рассмотренные команды позволяют создавать, удалять и редактировать учетные записи, однако они не разрешают изменять привилегии пользователя, т. е., сообщать СУБД MySQL, какой пользователь имеет право только на чтение информации, какой — на чтение и редактирование, а кому предоставлены права изменять структуру базы данных и создавать учетные записи для остальных пользователей.

Для решения этих задач предназначены операторы **GRANT** и **REVOKE**: оператор **GRANT** назначает привилегии пользователю, а **REVOKE** — удаляет.

Если учетная запись, которая появляется в команде **GRANT**, не существует, она автоматически создается. Однако удаление всех привилегий с помощью команды **REVOKE** не приводит к автоматическому уничтожению учетной записи. Для полного удаления пользователя необходимо либо воспользоваться оператором **DROP USER**.

Давайте воспользуемся командой **GRANT**:

```
GRANT ALL ON *.* TO 'foo'@'localhost' IDENTIFIED WITH sha256_password BY 'pass';
```

Запрос создает пользователя с именем **foo** и паролем **pass**. Этот пользователь может обращаться к серверу MySQL с локального хоста (**localhost**) и имеет все права (**ALL**) для всех баз данных (***.***). Если такой пользователь уже существует, то его привилегии будут изменены на **ALL**.

Отобрать права у пользователя можно при помощи команды **REVOKE**:

```
REVOKE ALL ON *.* FROM 'foo'@'localhost';
```

Существует большое количество самых разнообразных привилегий.

```
GRANT SELECT, INSERT, DELETE, UPDATE ON *.* TO foo;
```

Например, тут пользователю **foo** разрешается полный доступ на просмотр, заполнение, редактирование и удаление таблиц.

```
GRANT ALL ON *.* TO foo;
GRANT GRANT OPTION ON *.* TO foo;
```

Чтобы назначить все привилегии сразу, следует назначить привилегию **ALL** и **GRANT OPTION**. **ALL** разрешает все операции, кроме назначение прав себе и другим пользователям, за это отвечает специальная привилегия **GRANT OPTION**.

Выполнить операции по наделению пользователя всеми правами в одном запросе не получиться, ключевое слово **ALL** всегда употребляется отдельно и не должно использоваться совместно с другими ключевыми словами:

```
GRANT ALL, GRANT OPTION ON *.* TO foo;
```

Однако использование ключевого слова **USEAGE**, означающего полное отсутствие привилегий, совместимо с другими ключевыми словами:

```
GRANT USAGE, SELECT ON *.* TO foo;
```

Ключевое слово ON в операторе GRANT задает уровень привилегий:

- GRANT USAGE ON *.* TO foo;
- GRANT USAGE ON * TO foo;
- GRANT USAGE ON shop.* TO foo;
- GRANT USAGE ON shop.catalogs TO foo;
- GRANT SELECT (id, name), UPDATE (name) ON shop.catalogs TO foo.

Глобальный уровень — касается всех баз данных и таблиц, входящих в их состав. Таким образом, пользователь с полномочиями на глобальном уровне может обращаться ко всем базам данных.

Если текущая база данных не была выбрана при помощи оператора **USE**, данное предложение эквивалентно **ON *.***, если произведен выбор текущей базы данных, то устанавливаемые при помощи оператора **GRANT** привилегии относятся ко всем таблицам текущей базы данных

Уровень базы данных — данное предложение означает, что привилегии распространяются на таблицы базы данных **db.**

Уровень таблицы — данное предложение означает, что привилегии распространяются на таблицу **tbl** базы данных **db**.

Уровень столбца — привилегии уровня столбца касаются отдельных столбцов в таблице **tbl** базы данных **db**.

Список столбцов указывается в скобках через запятую после ключевых слов **SELECT**, **INSERT, UPDATE**.

Для просмотра существующих привилегий необходимо было выполнить оператор **SHOW GRANTS**:

```
SHOW GRANTS;
```

Ключевое слово **WITH** оператора **GRANT**, помимо привилегии **GRANT OPTION**, позволяет наложить ограничения на число подключений, запросов, обновлений и параллельных запросов в час:

```
GRANT ALL ON shop.* TO 'foo'@'localhost' IDENTIFIED WITH sha256_password BY
'pass'
WITH MAX_CONNECTIONS_PER_HOUR 10
      MAX_QUERIES_PER_HOUR 1000
      MAX_UPDATES_PER_HOUR 200
      MAX_USER_CONNECTIONS 3;
```

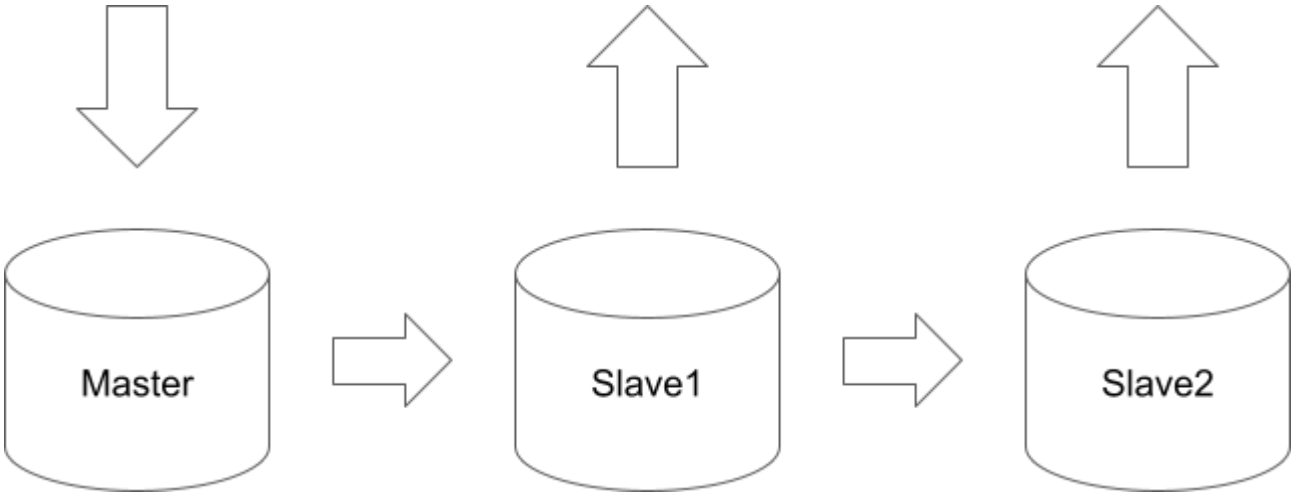
Запрос устанавливает для пользователя **foo** полный доступ к базе данных **shop**, но с ограничением — не более 10 подключений к серверу MySQL в час и не более 1000 запросов, из которых только 200 могут быть операциями обновления **UPDATE**, а 3 запроса — протекать одновременно.

Если значение каждой из опции равно 0 (по умолчанию) — это означает, что у пользователя нет ограничений по этому параметру.

Репликация

Репликация (replication) означает хранение копий одних и тех же данных на нескольких машинах, соединенных с помощью сети. Есть несколько возможных причин репликации данных:

- ради хранения данных географически близко к пользователям (и сокращения, таким образом, задержек);
- чтобы система могла продолжать работать при отказе некоторых ее частей (и повышения, таким образом, доступности);
- для горизонтального масштабирования количества машин, обслуживающих запросы на чтение (и повышения, таким образом, пропускной способности по чтению).



Данные записываются в один из серверов, который называется мастером, и передаются за счет копирования бинарного журнала на связанные с ним, которые называют слейв-серверами. При классической репликации набор данных, которые хранит база данных, должен убираться на одной машине. Каждая из машин, участвующих в репликации, должна хранить у себя копию всех данных.

Репликация по определению может масштабировать только операцию чтения, операция записи не масштабируется. Все сервера, сколько бы их ни было, должны выполнить запись. Выгоду мы приобретаем, только если можем разделить запросы на чтение среди нескольких серверов.

Чтобы продемонстрировать репликацию, нам потребуется запустить два MySQL-сервера. Один будет выступать master-сервером, принимающим запросы на вставку и обновление данных, другой — слейв-сервером, который будет получать изменения с master-сервера.

Для этого предлагаю воспользоваться утилитой **mysqld_multi**, которая позволяет запускать несколько экземпляров mysql-сервера на одном хосте.

Предварительно останавливаем сервер и инициируем дополнительный каталог данных

```
bin/mysqld --initialize --user=mysql --basedir=/usr/local/opt/mysql
--datadir=/usr/local/var/mysql1
```

Далее редактируем конфигурационный файла **my.cnf**. Давайте уберем все предыдущие изменения и добавим две секции: **mysql1** и **mysql2**.

```
[mysqld]
```

```
bind-address = 127.0.0.1
skip-grant-tables

[mysqld1]
socket      = /tmp/mysql.sock1
port        = 3306
pid-file    = /usr/local/var/mysql1/mysqld1.pid
datadir     = /usr/local/var/mysql1

[mysqld2]
socket      = /tmp/mysql.sock2
port        = 3307
pid-file    = /usr/local/var/mysql2/mysqld2.pid
datadir     = /usr/local/var/mysql2
```

Мы изменяем каталог данных, каждый из серверов будет смотреть в разный каталог данных, кроме того, мы указываем директиву **skip-grant-tables**, которая требует, чтобы настройки привилегий пользователей игнорировались и мы могли бы обратиться к серверам, не указывая пароль.

Кроме того, мы явно указываем разные порты и сокеты. По умолчанию, если вы обращаетесь к mysql-серверу с текущей машины, клиент **mysql** будет пытаться использовать сокет. Если обращение идет с внешней машины, вам волей-неволей придется использовать IP-адрес и порт.

Запустим сервер при помощи команды **mysqld_multi**:

```
mysqld_multi start
```

При старте MySQL инициализирует новые каталоги данных, для каждого из серверов указывается временный пароль root-пользователя. Однако он нам не потребуется, так как при помощи директивы **skip-grant-tables** мы явно отключили проверки паролей и привилегий.

```
mysqld_multi stop
mysqld_multi start 1
mysqld_multi start 2

mysql --socket=/tmp/mysql.sock1 -u root
mysql --socket=/tmp/mysql.sock2 -u root

mysql> FLUSH PRIVILEGES;
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY '';

mysqld_multi stop
ps aux | grep mysqld

mysql --socket=/tmp/mysql.sock2 -u root
mysql --socket=/tmp/mysql.sock2 -u root
```

Давайте сервер **mysql1** у нас будет master-сервером, а **mysql2** — slave-сервером. Для настройки сервера в качестве главного необходимо включить бинарный журнал (**binary log**) и назначить уникальный идентификатор сервера.

```
[mysqld1]
socket      = /tmp/mysql.sock1
port        = 3306
pid-file    = /usr/local/var/mysql1/mysqld1.pid
datadir     = /usr/local/var/mysql1
log-bin     = master-bin
log-bin-index = master-bin.index
server-id   = 1
```

Идентификатор сервера нужен, чтобы отличить один от другого. В параметре **log-bin** задано базовое имя для всех файлов, созданных в двоичном журнале. В параметре **log-bin-index** задано базовое имя индексного файла двоичного журнала, в котором содержится список всех файлов двоичного журнала.

Можно не указывать имя файла в параметре **log-bin**. В этом случае значение будет назначено по умолчанию. Однако если имя хоста будет изменено, то имена файлов бинарного журнала тоже обновятся. Поэтому лучше их назначить сразу и явно. Перезапускаем сервер.

Перейдем в каталог данных первого сервера:

```
cd /usr/local/var/mysql1
ls -la
```

Видим индексный и первый файлы бинарного журнала. Подчиненный сервер посылает запрос на главный на получение всех изменений. Чтобы подключить подчиненный сервер, на главном должен быть пользователь с особыми полномочиями репликации.

```
mysql --socket=/tmp/mysql.sock1 -u root
mysql> CREATE USER repl_user;
mysql> GRANT REPLICATION SLAVE ON *.* TO repl_user IDENTIFIED BY '321321';
```

Привилегия **REPLICATION SLAVE** позволяет пользователю получать доступ к двоичному журналу. Теперь давайте настроим **slave**, или подчиненный сервер.

Как и главному, подчиненному серверу нужно присвоить уникальный идентификатор. Кроме того, в файл **my.cnf** можно добавить имена файлов журнала ретрансляции и индекса журнала ретрансляции:

```
[mysqld2]
socket      = /tmp/mysql.sock2
port        = 3307
pid-file    = /usr/local/var/mysql2/mysqld2.pid
datadir     = /usr/local/var/mysql2
server-id   = 2
relay-log-index = slave-relay-bin.index
relay-log   = slave-relay-bin
```

По умолчанию значения параметров **relay-log** и **relay-log-index**, как и для параметров **log-bin** и **log-bin-index**, зависят от имени хоста.

Использование умолчаний чревато проблемой, так как в случае изменения имени узла сервера индексный файл журнала ретрансляции не будет найден, и сервер будет полагать, что файлы журнала ретрансляции пустые.

Осталось сделать заключительный шаг: указать slave-серверу главный, чтобы он знал, откуда выполнять репликацию.

Для этого нужно ответить на четыре вопроса о главном сервере:

- Имя хоста.
- Порт.
- Учетная запись для репликации.
- Пароль к учетной записи.

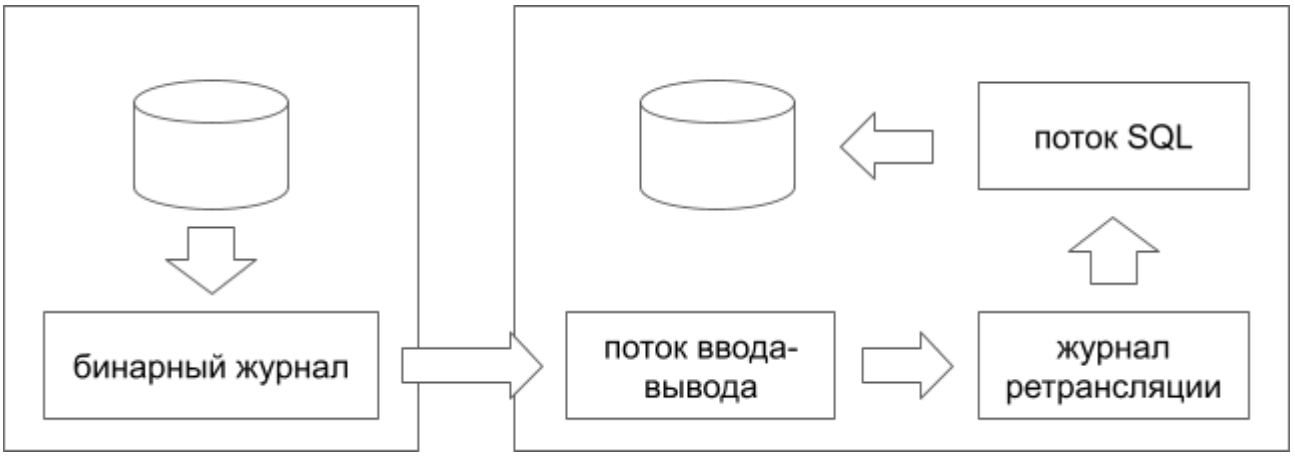
```
CHANGE MASTER TO
MASTER_HOST = 'localhost',
MASTER_PORT = 3306,
MASTER_USER = 'repl_user',
MASTER_PASSWORD = '321321';

START SLAVE;
```

Простейшая репликация между главным и подчиненным серверами настроена. Проверить успешность запуска процесса репликации на подчиненном сервере можно при помощи оператора **SHOW SLAVE STATUS**:

```
SHOW SLAVE STATUS\G
```

Внутреннее устройство репликации

Главный сервер записывает изменения данных в бинарный журнал. Подчиненный сервер копирует содержимое бинарного журнала в свой журнал ретрансляции. За это отвечает специальный поток ввода данных. Далее подчиненный сервер воспроизводит события из журнала ретрансляции, применяя изменения к собственным данным. Этим занимается еще один поток, который называется SQL.



Возвращаясь к отчету команды **SHOW SLAVE STATUS**, мы можем увидеть две строки: **Slave_IO_Running** и **Slave_SQL_Running**, именно они показывают статус потоков SQL и ввода-вывода. Если один из потоков не работает, вместо значений **Yes** будет выведено значение **No**.

По значению SQL_Delay можно судить о том, насколько slave-сервер отстает от мастера: чем меньше это значение, тем лучше.

Ограничения масштабирования

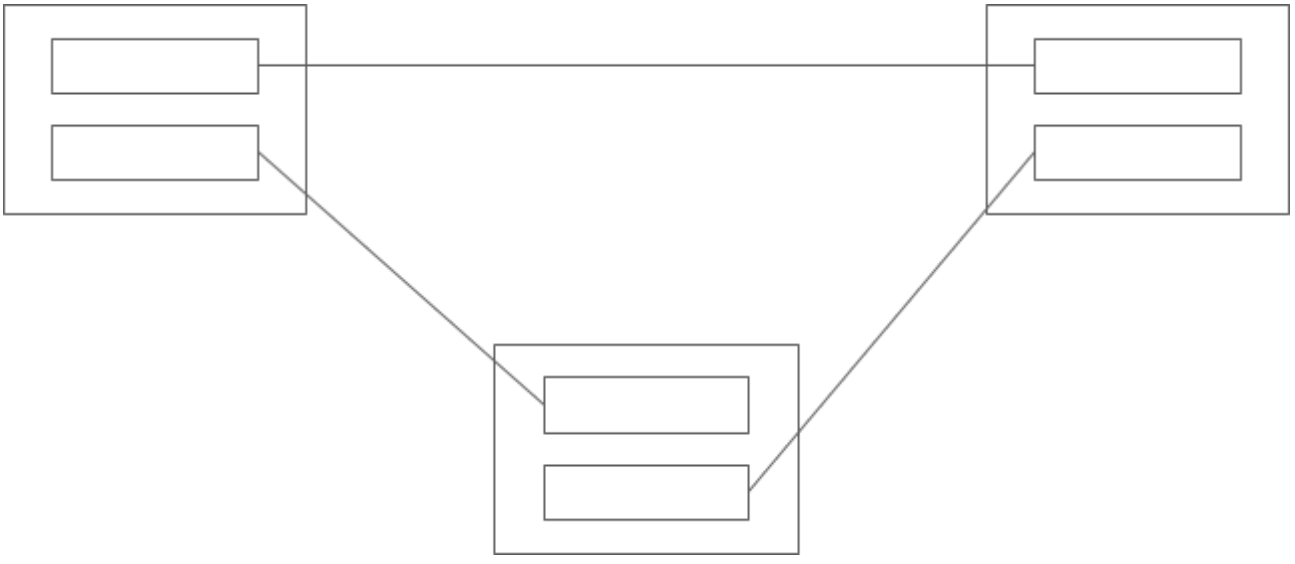
При классической репликации набор данных, которые хранит база, должен убираться на одной машине. Каждая из машин, участвующих в репликации, должна хранить у себя копию всех данных.

Репликация по определению может масштабировать только чтение, запись не масштабируется. Все сервера, сколько бы их ни было, должны выполнить запись. Выгоду мы приобретаем, только если можем разделить запросы на чтение среди нескольких серверов.

сервера	операций	INSERT	SELECT
1	100 000	20 000	80 000
2	200 000	40 000	60 000
3	300 000	60 000	40 000
4	400 000	80 000	20 000
5	500 000	100 000	0

Шардирование

В тех случаях, когда данных не убираются на одной машине, их придется дробить на части и располагать несколько копий одной и той же части на нескольких машинах.



Такой подход называется секционированием или шардированием. Между этими частями, которые располагаются на разных хостах, также устанавливаются отношения репликации.

Только здесь репликация осуществляется не между физическими серверами, а между фрагментами или секциями большой базы данных.

Откровенно говоря, MySQL довольно плохо поддерживает шардирование из коробки. Для него, как правило, выбирают либо базу данных, которая поддерживает шардирование из коробки (PostgreSQL, MongoDB, Cassandra), либо эксплуатируют MySQL в специальном режиме, например режиме кластера.

В любом случае, даже если вы используете шардирование, вы сталкиваетесь со всеми особенностями репликации, которой будут посвящены оставшиеся ролики урока.

Типы репликации

Различают несколько типов репликации:

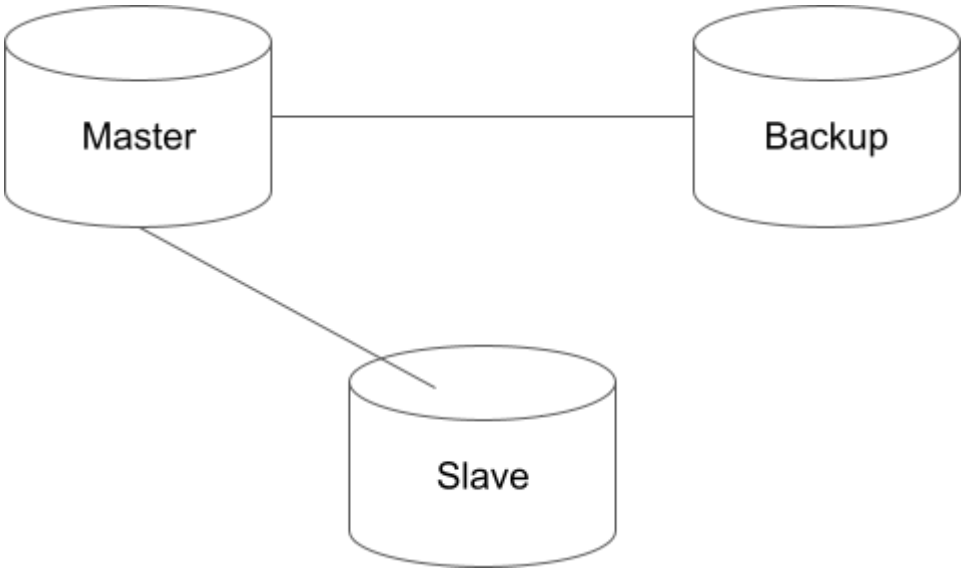
- синхронная,
- асинхронная.

В классической MySQL используется асинхронная репликация, когда изменения применяются на master-сервере, а на подчиненных серверах они доставляются с задержкой. При синхронной репликации мастер-сервер ожидает ответа от всех реплик и считает запрос завершенным только когда данные сохранились на всех хостах, участвующих в репликации.

В случае синхронной репликации резко возрастает время ответа сервера для операций записи, а зачастую и чтения. В синхронной репликации возникают ошибки, связанные с задержками доставки данных, которые могут достигать минуты. Вы публикуете сообщение, хотите его прочитать, но попадаете на slave-сервер, где этого сообщения еще нет, т.е., ваше приложение уверено, что оно должно быть в базе данных, но запись появилась на master-сервере и еще не добралась до slave-сервера.

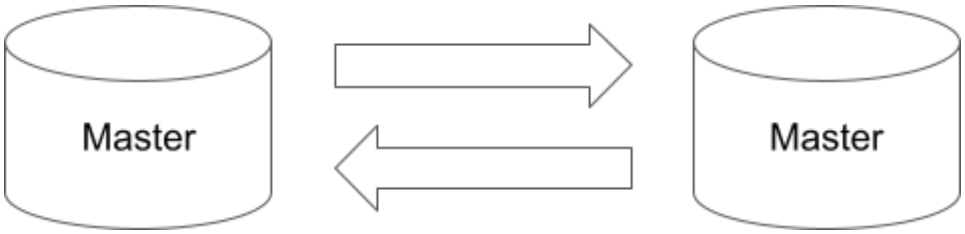
Если вы разрабатываете приложение, базу данных которого планируете в дальнейшем масштабировать при помощи асинхронной репликации, используйте репликацию сразу на этапе разработки. Заменить монолитную базу данных реплицированной потом крайне сложно. Появляется целый класс очень сложно улавливаемых ошибок, когда вы вставляете данные на master-сервере, читаете их со slave-сервера, а их еще нет, они не доставлены.

Когда серверов более чем один, появляются различные варианты их использования.



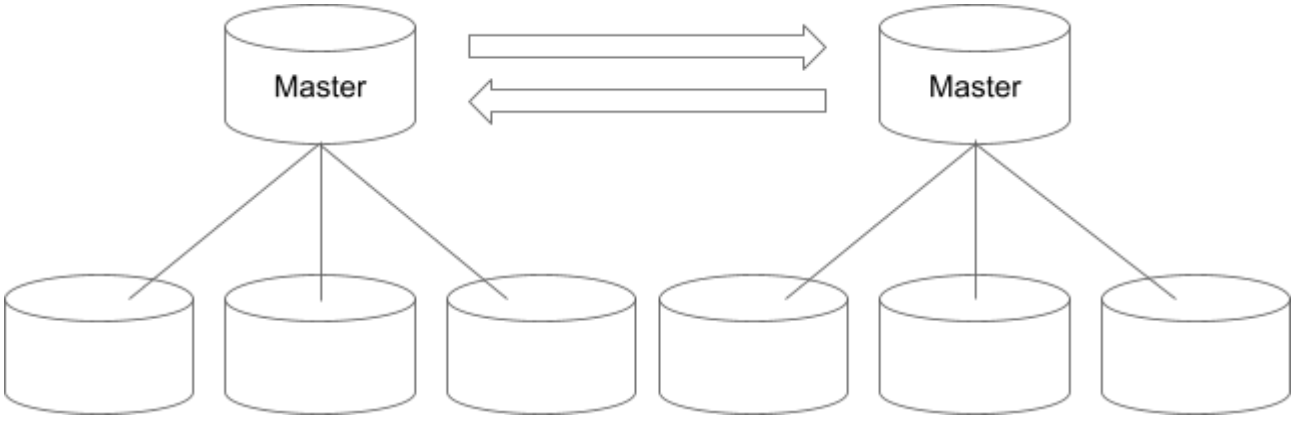
Наиболее простая топология дублирования серверов — это топология горячего резерва. Она состоит из главного сервера и выделенного сервера, называемого «горячим резервом» и дублирующего главный сервер. Сервер горячего резерва подключен к главному как подчиненный, читая и внося все изменения.

Идея состоит в том, что в случае отказа главного сервера горячий резерв — его точная копия — заменит его. Это даст возможность переключить всех клиентов и подчиненные серверы на резервный и продолжить работу.



Довольно популярной топологией является **master-master**. В ней используются два главных сервера, которые реплицируют изменения друг друга. Эта система проста в использовании, поскольку симметрична:

- для перехода на резервный главный сервер не требуется менять конфигурацию главного сервера,
- вернуться на исходный главный сервер при отказе резервного очень просто.



Каждый из master-серверов может выступать в качестве главного для целой группы mysql-серверов. Если на обоих главных серверах будет обновлена одна и та же информация — между обновлениями возникнет конфликт, который может привести к останову репликации.

Частично исключить проблему конфликтов изменений можно, разрешив запросы на запись только на одном из серверов, т. е., сделав пассивным один из серверов. Решить такие конфликты можно, либо записывая информацию в разные таблицы, либо организовать запись таким образом, чтобы исключить конфликты.

Большинство конфликтов связаны со вставкой новых значений в таблицу, так как два master-сервера ведут свой отсчет счетчика **auto_increment** для первичного ключа.

Сервер MySQL предоставляет две серверные переменные для управления механизмом **AUTO_INCREMENT**:

- **auto_increment_offset** — назначает начальное значение всех столбцов **AUTO_INCREMENT**;
- **auto_increment_increment** — устанавливает приращение, которое используется для вычисления следующего значения в столбце **AUTO_INCREMENT**.

У любого подчиненного сервера может быть только один главный, поэтому единственный способ этого достичь — настроить репликацию по кругу. Это не рекомендуется, но вполне возможно. Не рекомендуют этот вариант потому, что очень сложно добиться его правильной работы при сбое.

Используемые источники

1. https://dev.mysql.com/doc/refman/5.7/en/server-administration.html
2. https://dev.mysql.com/doc/refman/5.7/en/server-logs.html
3. https://dev.mysql.com/doc/refman/5.7/en/account-management-sql.html
4. https://dev.mysql.com/doc/refman/5.7/en/sql-syntax-replication.html
5. Линн Бейли. Head First. Изучаем SQL. — СПб.: Питер, 2012. — 592 с.
6. Грофф, Джеймс Р., Вайнберг, Пол Н., Оппель, Эндрю Дж. SQL: полное руководство, 3-е изд. : Пер. с англ. — М.: ООО "И.Д. Вильямс", 2015. — 960 с.

7. Дейт К. Дж. SQL и реляционная теория. Как грамотно писать код на SQL. — Пер. с англ. — СПб.: Символ-Плюс, 2010. — 480 с.

8. Кузнецов М.В., Симдянов И.В. MySQL на примерах. — СПб.: БХВ-Петербург, 2007. — 592с.

9. Кузнецов М.В., Симдянов И.В. MySQL 5. — СПб.: БХВ-Петербург, 2006. — 1024с.

10. Дейт К. Дж. Введение в системы баз данных, 8-е издание.: Пер. с англ. — М.: Издательский дом "Вильямс", 2005. — 1328 с.

11. Карвин Б. Программирование баз данных SQL. Типичные ошибки и их устранение. — Рид Групп, 2011. — 336 с.