



Урок 6

Масштабирование MySQL и NoSQL

Что такое кластер, что такое репликация данных, где использовать NoSQL.

Репликация базы данных

[Шаг 1. Настройка Мастера](#)

[Шаг 2. Права на репликацию](#)

[Шаг 3. Дамп базы](#)

[Шаг 4. Создание базы на Слейве](#)

[Шаг 5. Настройка Слейва](#)

[Шаг 6. Запуск Слейва](#)

[Статус репликации](#)

Кластеризация БД

[Немного истории](#)

[Подготовка](#)

[Настройка кластера](#)

[failover-доступ](#)

NoSQL

[История](#)

MongoDB

[Архитектура](#)

[Создание базы данных](#)

[Выборка данных](#)

[Запрос к вложенным объектам](#)

[Условные операторы в MongoDB](#)

[Оператор \\$ne](#)

[Поиск по массивам и операторы \\$in, \\$nin, \\$all](#)

[Оператор \\$or](#)

[Оператор \\$size](#)

[Оператор \\$exists](#)

[Оператор \\$regex](#)

[Обновление данных](#)

[Обновление отдельного поля](#)

[Удаление поля](#)

[Особенности работы MongoDB](#)

[Практическая работа](#)

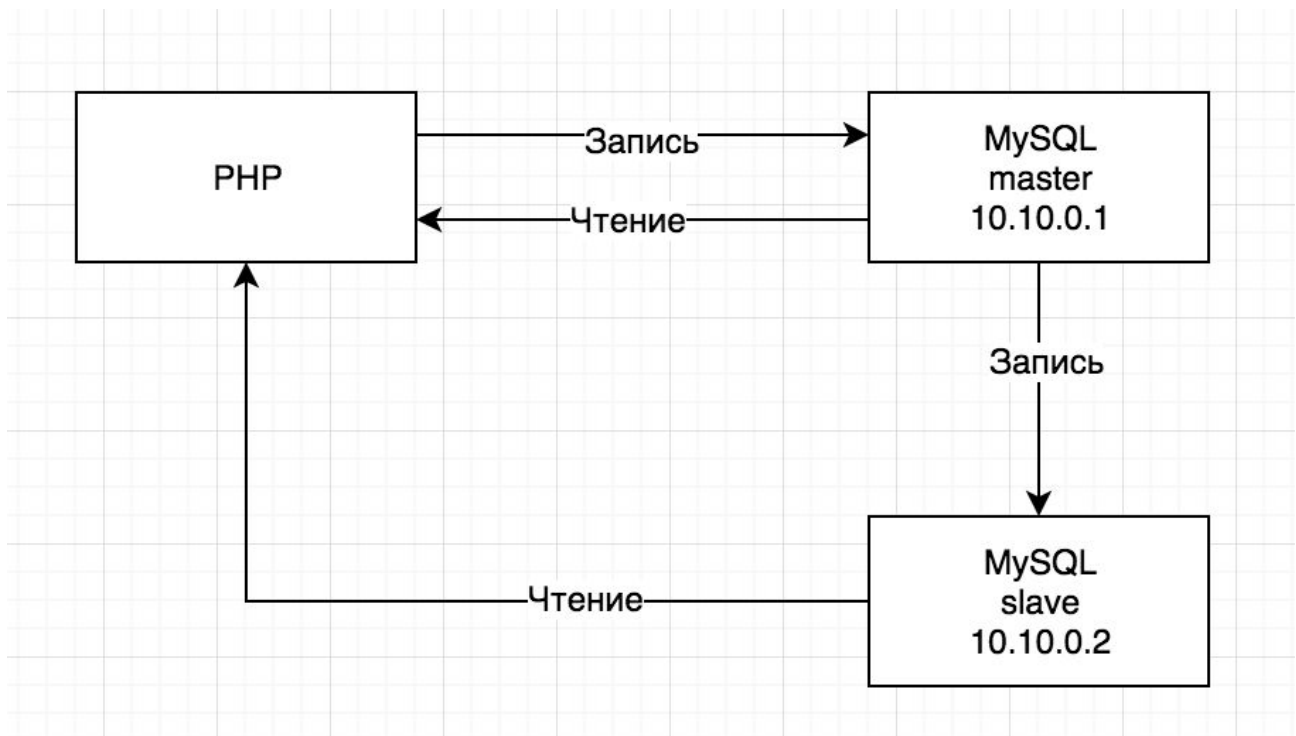
[Практическое задание](#)

[Дополнительные материалы](#)

[Используемая литература](#)

Репликация базы данных

Пассивная MySQL-репликация, построенная по виду Ведущий-Ведомый или Master-Slave, является одним из классических способов построения отказоустойчивой архитектуры приложения. Такой подход позволяет не только зарезервировать данные на зеркальном сервере, но и распределить нагрузку чтения и записи между серверами подобного кластера.



Построение системы реплицирования осуществляется по шагам, которые мы рассмотрим ниже. Для примера будем строить систему из 2 серверов со следующими параметрами:

- Master (ведущий) сервер, 10.10.0.1;
- Slave (ведомый) сервер, 10.10.0.2.

Шаг 1. Настройка Мастера

Ведущий сервер должен быть обозначен как master. Для этого в файле конфигурации my.cnf вносятся следующие настройки:

```
# выбираем ID сервера, произвольное число, лучше начинать с 1
server-id = 1
# путь к бинарному логу
log_bin = /var/log/mysql/mysql-bin.log
# название базы данных, которая будет реплицироваться
binlog_do_db = newdatabase
```

После изменения настроек потребуется рестарт ведущего сервера Mysql:

```
/etc/init.d/mysql restart
```

Шаг 2. Права на репликацию

Теперь нужно обеспечить доступ к ведущему серверу, чтобы ведомые сервера могли получать обновления:

```
mysql -u root -p
```

```
mysql> GRANT REPLICATION SLAVE ON *.* TO 'slave_user'@'%' ...
... IDENTIFIED BY 'password'; FLUSH PRIVILEGES;
```

При первом создании репликации нужно заблокировать на изменение все таблицы на ведущем сервере:

```
USE newdatabase;
FLUSH TABLES WITH READ LOCK;
```

После этого проверим статус ведущего сервера:

```
SHOW MASTER STATUS;
```

Результат будет выглядеть примерно так:

```
mysql> SHOW MASTER STATUS;
```

File	Position	Binlog_Do_DB	Binlog_Ignore_DB
mysql-bin.000001	107	newdatabase	

1 row in set (0.00 sec)

Шаг 3. Дамп базы

Чтобы система реплицирования данных начала работать, нужно обеспечить ей актуальные данные для старта, сняв дамп данных с заблокированного ведущего сервера:

```
mysqldump -u root -p newdatabase > newdatabase.sql
```

По завершении снятия дампа можно разблокировать сервер:

```
UNLOCK TABLES;
```

Шаг 4. Создание Слейв-сервера

Существует несколько вариантов его создания:

- запустить на выделенном компьютере с IP-адресом, отличным от IP-адреса ведущего сервера;
- воспользоваться виртуализацией и запустить на одном компьютере два контейнера с MySQL-серверами;
- Воспользоваться утилитой `mysqld_multi` для запуска на одном компьютере двух mysql-серверов на разных портах, например, 3306 и 3307.

При знакомстве с репликацией, проще всего воспользоваться последним вариантом. Для этого необходимо в конфигурационном файле `my.cnf` добавить 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
```

Каждый из серверов будет смотреть в разный каталог данных `datadir`. 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. Дополнительно указываем разные порты и сокеты.

Запустить сервера можно при помощи утилиты `mysqld_multi`

```
mysqld_multi start
```

Для остановки сервера, можно передать утилите `mysqld_multi`

```
mysqld_multi stop
```

Теперь один сервер будет работать на 3306 порту, другой — на 3307.

Шаг 5. Создание базы на Слейве

Теперь поработаем с ведомым сервером. Здесь нам нужно создать копию БД с ведущего сервера:

```
CREATE DATABASE newdatabase;  
  
mysql -u root -p newdatabase < newdatabase.sql
```

Шаг 6. Настройка Слейва

Теперь в файле `my.cnf` на ведомом сервере вам нужно сконфигурировать свойства реплицирования данных:

```
# ID Слейва, удобно выбирать следующим числом после Мастера  
server-id = 2  
# Путь к relay-логу  
relay-log = /var/log/mysql/mysql-relay-bin.log  
# Путь к bin-логу на Мастере  
log_bin = /var/log/mysql/mysql-bin.log  
# База данных для репликации  
binlog_do_db = newdatabase
```

Шаг 7. Запуск Слейва

Подготовительные этапы завершены, и мы можем запускать процесс репликации. Для этого осталось только указать, откуда и с какой позиции читать данные ведомому серверу:

```
CHANGE MASTER TO MASTER_HOST='10.10.0.1', MASTER_USER='slave_user',  
MASTER_PASSWORD='password',  
MASTER_LOG_FILE = 'mysql-bin.000001', MASTER_LOG_POS = 107;  
START SLAVE;
```

Информацию для заполнения полей `MASTER_LOG_FILE` и `MASTER_LOG_POS` извлекается из результата запроса `SHOW MASTER STATUS`, который был выполнен на втором шаге.

Статус репликации

Для того, чтобы проверить состояние ведомого сервера, вам надо выполнить на нём вот такой запрос:

```
mysql> show slave status \G
***** 1. row *****
Slave_IO_State: Waiting for master to send event
Master_Host:
Master_User:
Master_Port: 3306
Connect_Retry: 60
Master_Log_File: mysql-bin.003572
Read_Master_Log_Pos: 18012440
Relay_Log_File: relay.000376
Relay_Log_Pos: 18012603
Relay_Master_Log_File: mysql-bin.003572
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
Replicate_Do_DB:
Replicate_Ignore_DB:
Replicate_Do_Table:
Replicate_Ignore_Table:
Replicate_Wild_Do_Table:
Replicate_Wild_Ignore_Table:
Last_Errno: 0
Last_Error:
Skip_Counter: 0
Exec_Master_Log_Pos: 18012440
Relay_Log_Space: 18012813
Until_Condition: None
Until_Log_File:
Until_Log_Pos: 0
Master_SSL_Allowed: No
Master_SSL_CA_File:
Master_SSL_CA_Path:
Master_SSL_Cert:
Master_SSL_Cipher:
Master_SSL_Key:
Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
Last_IO_Errno: 0
Last_IO_Error:
Last_SQL_Errno: 0
Last_SQL_Error:
Replicate_Ignore_Server_Ids:
Master_Server_Id: 17224121
Master_UUID:
Master_Info_File: /data/mysql/master.info
SQL_Delay: 0
SQL_Remaining_Delay: NULL
Slave_SQL_Running_State: Slave has read all relay log; waiting for the slave I/O thread to update it
Master_Retry_Count: 86400
Master_Bind:
Last_IO_Error_Timestamp:
Last_SQL_Error_Timestamp:
Master_SSL_Crl:
Master_SSL_Crlpath:
Retrieved_Gtid_Set:
Executed_Gtid_Set:
Auto_Position: 0
1 row in set (0.00 sec)
```

Кластеризация БД

Если возможностей одной ноды MySQL уже недостаточно, на помощь приходит кластеризация.

Технология MySQL Cluster вводит в работу совершенно новый для MySQL способ хранения данных – NDB Cluster. Он позволяет реализовывать выполнение команд по изменению данных на нескольких ведущих серверах, объединенных в кластер.

СУБД MySQL имеет довольно высокие рейтинги использования в веб-проектах, оставаясь при этом простой в использовании и запуске. Все популярные языки программирования для веб поддерживают эту СУБД без дополнительных модулей. Однако это не делает MySQL совершенным решением. MySQL имеет весьма большие проблемы, связанные с быстродействием и надежностью. Тонкая

настройка MySQL – это отдельная глава для изучения.

Для создания отказоустойчивого кластера требуется минимум три сервера. Это делается, чтобы два сервера могли проверять корректность своих данных на третьем:

- db1 (IP 10.10.171.2)
- db2 (IP 10.10.171.3)
- db3 (IP 10.10.171.4)

В данной терминологии сервер DB1 – это уже имеющийся у нас сервер со стандартным MySQL-сервером на борту. Данные с него будут переноситься в наш кластер. Сервера DB2 и DB3 для нас являются новыми.

Немного истории

Для организации кластера мы будем использовать СУБД Percona. На самом деле эта СУБД является тем же MySQL, но при этом – его ответвлением (fork`ом). Для повышения быстродействия MySQL множество системных администраторов и разработчиков постоянно устанавливали пакет исправлений от Google и от Петра Зайцева. Именно он и стал основателем компании Percona. Затем он же создал пакет XtraDB Cluster, включающий в себя библиотеки Gallera Cluster – производительной системы для организации кластеров на базе MySQL.

Подготовка

Сам по себе XtraDB cluster не содержится в штатных репозиториях Linux ОС. Значит, нам будет необходимо расширить доступные дистрибутивы.

Инструкции по установке для различных систем можно найти здесь

<https://www.percona.com/doc/percona-server/LATEST/installation.html#installing-from-binaries>

Для того, чтобы узлы (ноды) нашего будущего кластера умели обмениваться данными, каждый сервер должен быть доступен для остальных через порты 4444 и 4567, а также 3306 (штатный порт mysql) и 9199.

Настройка кластера

В штатном файле настроек нужно ввести строки:

```
[mysqld]
datadir=/var/lib/mysql
user=mysql
wsrep_provider=/usr/lib64/libgalera_smm.so
wsrep_cluster_address=gcomm://10.10.171.2,10.10.171.3,10.10.171.4
binlog_format=ROW
default_storage_engine=InnoDB
innodb_autoinc_lock_mode=2
wsrep_node_address=10.10
wsrep_sst_method=xtrabackup-v2
wsrep_cluster_name=axsystems_xtradb_test
wsrep_sst_auth="syncuser:some_pass"
```

После этих манипуляций вы можете запускать первую ноду нашего кластера в работу.

```
/etc/init.d/mysql bootstrap-pxc
```

Как и в случае с репликацией, нам надо создать пользователя, который будет осуществлять синхронизацию:

```
mysql@db1> CREATE USER 'syncuser'@'localhost' IDENTIFIED BY 'some_pass';  
mysql@db1> GRANT RELOAD, LOCK TABLES, REPLICATION CLIENT ON *.* TO  
'syncuser'@'localhost';  
mysql@db1> FLUSH PRIVILEGES;
```

Теперь полностью копируем конфигурацию на сервер DB2, внося соответствующие коррективы:

```
[mysqld]  
datadir=/var/lib/mysql  
user=mysql  
wsrep_provider=/usr/lib64/libgalera_smm.so  
wsrep_cluster_address=gcomm://10.10.171.2,10.10.171.3,10.10.171.4  
binlog_format=ROW  
default_storage_engine=InnoDB  
innodb_autoinc_lock_mode=2  
wsrep_node_address=10.10  
wsrep_sst_method=xtrabackup-v2  
wsrep_cluster_name=axsystems_xtradb_test
```

После этого подключаем новый сервер к кластеру:

```
/etc/init.d/mysql start
```

Процесс запуска довольно долгий в силу того, что второй сервер начинает копировать данные к себе. Ведь вы обратили внимание на то, что мы не снимали бэкап? При этом оба сервера полностью работоспособны и доступны для создания новых данных.

После того, как сервер запустится, вы можете проверять состояние всего кластера, запуская с любого сервера команду:

```
mysql> show status like 'wsrep%';
```

Variable_name	Value
wsrep_local_state_uuid	c2883338-834d-11e2-0800-03c9c68e41ec
wsrep_local_state	4
wsrep_local_state_comment	Synced
wsrep_cluster_size	2
wsrep_cluster_status	Primary
wsrep_connected	ON
wsrep_ready	ON

40 rows in set (0.01 sec)

Обратите внимание на следующие параметры:

- status – synced;
- cluster_size – 2 (узла).

Они говорят, что кластер начал работать в штатном режиме. Теперь мы можем добавлять третий и последующие узлы по образу и подобию.

failover-доступ

Работа высоконагруженного приложения может быть организована таким образом, чтобы запросы поступали не к одному серверу MySQL, а распределялись между несколькими серверами. Для этого часто применяется балансировщик нагрузки haproxy. Давайте попробуем создать архитектуру, в которой наш балансировщик будет установлен на базе каждой ноды MySQL.

```
#debian
apt-get install -y haproxy

#centos
yum install -y haproxy
```

После установки нужно настроить балансировщик, изменив файл /etc/haproxy/haproxy.cfg:

```

global
    log            127.0.0.1 local2
    chroot         /var/lib/haproxy
    pidfile        /var/run/haproxy.pid
    maxconn        4000
    user           haproxy
    group          haproxy
    daemon
    #stats socket /var/lib/haproxy/stats

defaults
    log            global
    mode           http
    option         tcplog
    option         dontlognull
    retries        3
    redispatch
    maxconn        2000
    contimeout     5000
    clitimeout     50000
    srvtimeout     50000

listen stats 0.0.0.0:8086
    mode http
    stats enable
    stats uri /
    stats realm "balancer"
    stats auth logan:q6SJYy5cB3KKy34z

listen mysql-cluster 0.0.0.0:3306
    mode tcp
    balance roundrobin
    option httpchk
    server db1 10.10.171.2:3306 check port 9199 inter 12000 rise 3 fall 3
    server db2 10.10.171.3:3306 check port 9199 inter 12000 rise 3 fall 3
    server db3 10.10.171.4:3306 check port 9199 inter 12000 rise 3 fall 3

```

Указанный в настройках сетевой порт 9199 будет применяться балансировщиком для контроля исправности и работоспособности ноды. При этом, разумеется, XtraDB по этому порту не принимает соединения.

Для контроля работы кластера xtradb необходимо создать сервис, который будет управляться через утилиту xinetd.

```
#centos
yum install -y xinetd

#debian
apt-get install -y xinetd
```

Теперь напишем скрипт /etc/xinetd.d/mysqlchk для проверки:

```
service mysqlchk
{
    disable = no
    flags    = REUSE
    socket_type = stream
    port     = 9199
    wait     = no
    user     = nobody
    server    = /usr/bin/clustercheck
    server_args = syncuser Ttm3wsbPE72Km96R 1 /var/tmp/mss.log 0 /etc/my.cnf
    log_on_failure += USERID
    only_from = 0.0.0.0/0
    per_source = UNLIMITED
}
```

Наиболее важными настройками являются server_args. Очередность в них строго определена: вы указываете имя пользователя, из-под которого производится контроль реплицирования, пароль, файл журналов и файл конфигурации текущей ноды MySQL.

Теперь добавим настройку в /etc/services:

```
mysqlchk          9199/tcp #mysqlcheck
```

Итак, подготовительная стадия завершена, мы можем перезапустить xinetd и проверить его работу.

```
db1> telnet 127.0.0.1 9199

Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
HTTP/1.1 200 OK
Content-Type: text/plain
Connection: close
Content-Length: 40

Percona XtraDB Cluster Node is synced.
Connection closed by foreign host.
```

Вышеперечисленные действия необходимо произвести со всеми нодами MySQL. По завершении вы

перезапускаете балансировщик HAProxy. Вам станет доступен веб-интерфейс балансировки: [http://\[IP СЕРВЕРА\]:8086/](http://[IP СЕРВЕРА]:8086/)). В нём вы можете проверить, что балансировщику доступны все ноды. Отказоустойчивый кластер готов к использованию по адресу localhost!

NoSQL

Вы наверняка заметили, что технологии NoSQL по популярности обсуждений на технических ресурсах зачастую обходят классические реляционные базы данных. Идёт бурное развитие модульных решений на базе различных языков программирования. Решения NoSQL идут бок о бок с архитектурами, обрабатывающими большие данные (Big Data), легко масштабируемыми и отказоустойчивыми.

История

Несмотря на то, что понятие NoSQL появилось ещё в девяностых, современный вид подобные решения обрели только в 2009 году. Тогда словом NoSQL именовалось opensource-решение БД, разработанное Карло Строззи, хранившее данные в виде ASCII файлов. Вместо SQL-запросов эта СУБД применяла shell-скрипты. Это решение не имело совершенно ничего общего с тем NoSQL, который развивается в наши дни.

В июне 2009 в городе Сан-Франциско Йохан Оскарссон собрал встречу IT-специалистов, посвященную обзору тенденций среди технологий по хранению и обработке данных. Тому послужили фундаментом такие решения, как BigTable и Dynamo. Для привлечения внимания Эрик Эванс из компании RackSpace предложил простой и ёмкий хэштег – «NoSQL».

Поскольку NoSQL является огромным пластом технологий, охватить его весь нереально. В качестве практической основы мы будем рассматривать MongoDB.

MongoDB

MongoDB (от англ. humongous – огромный) – open source документоориентированная СУБД, не имеющая описаний схем своих таблиц.

Архитектура

MongoDB использует для хранения документы в JSON-подобном формате. Сами документы сохраняются в двоичном виде, имея в итоге формат BSON. В Mongo нет транзакций. При этом атомарность гарантируется исключительно на уровне целого документа. Таким образом, понятно, что в СУБД отсутствует частичное обновление документов.

Также в СУБД нет изоляции. Данные, читаемые одним соединением, вполне доступны для изменений другим.

MongoDB имеет асинхронную репликацию master – slave. Она базируется на обмене журналом изменений от мастера к слэйвам. В случае выхода из строя ведущего узла возможно автовосстановление, но для этого серверы mongodb должны быть объединены в т. н. кворум, на основании данных в котором можно гарантированно качественно восстановить данные на мастере.

Создание базы данных

При установке из коробки БД создаётся в c:/data/db

Давайте запустим mongo.exe и создадим новую БД:

```
use users
```

После этого нужно установить логин и пароль для администратора:

```
db.createUser({user:"admin", pwd:"1234",roles:["readWrite","dbAdmin"]})
```

Для управления используется Mongo-shell.

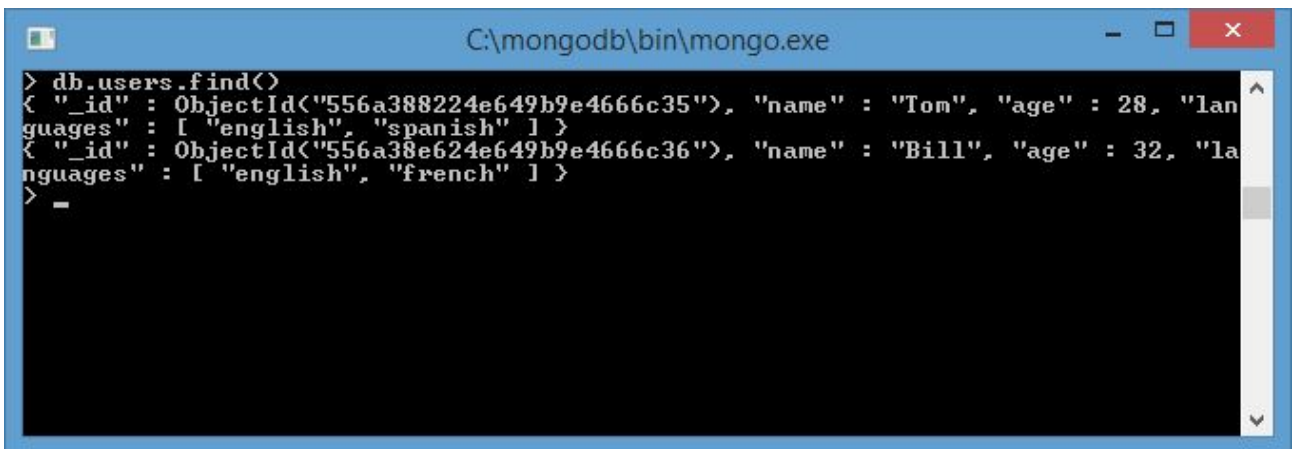
Чтобы добавить данные в коллекцию, нужно вызвать функцию insert.

```
db.users.insert({"name": "Tom", "age": 28, languages: ["english", "spanish"]})
```

Выборка данных

Для выборки данных в отличие от MySQL нужна команда find()

```
db.users.find()
```



```
C:\mongodb\bin\mongo.exe
> db.users.find()
{ "_id" : ObjectId("556a388224e649b9e4666c35"), "name" : "Tom", "age" : 28, "languages" : [ "english", "spanish" ] }
{ "_id" : ObjectId("556a38e624e649b9e4666c36"), "name" : "Bill", "age" : 32, "languages" : [ "english", "french" ] }
>
```

Запрос к вложенным объектам

Создадим вложенный объект с ключом company.

```
db.users.insert({"company":{"name":"Microsoft"},"age": 25, languages: ["english", "spanish"]})
```

Теперь для поиска всех документов, имеющих в качестве значения данного ключа name=microsoft,

нам необходимо производить вызов через оператор точку:

```
db.users.find({"company.name": "Microsoft"})
```

Условные операторы в MongoDB

Для фильтрации данных в выборках есть возможность указывать условные конструкции при помощи сравнения:

- \$gt (больше, чем);
- \$lt (меньше, чем);
- \$gte (больше или равно);
- \$lte (меньше или равно).

Например, найдем все документы, у которых значение ключа age меньше 30:

```
db.users.find ({age: {$lt : 30}})
```

В данном случае сравнение производится с типом integer. Когда нужно сравнить строки, конструкция изменится следующим образом:

```
db.users.find ({age: {$gt : "30"}}),
```

при этом результат тот же. Допускается также комбинация операторов:

```
db.users.find ({age: {$gt : 30, $lt: 50}})
```

Оператор \$ne

Также есть оператор отрицания \$ne:

```
db.users.find ({age: {$ne : 22}})
```

Поиск по массивам и операторы \$in, \$nin, \$all

Оператор \$in по заданному массиву значений выбирает те ключи, значения которых попадают в этот самый заданный массив:

```
db.users.find ({age: {$in : [22, 32]}})
db.users.find ({age: {$nin : [22, 32]}})
```

Есть также \$all, который очень похож на \$in, но он требует, чтобы искомые документы имели весь

заданный массив выражений.

```
db.users.find ({languages: {$all : [ "english" ,  "spanish" ]}})
```

Оператор \$or

Оператор задает набор пар ключ-значение для поиска в документе.

```
db.users.find ({ $or : [{name: "Tom"}, {age: 22}]})
```

Такое выражение найдет все документы, в которых либо name=Tom, либо age=22.

Оператор \$size

Оператор применяется для отыскания документов, в которых массивы данных имеют размерность, равную \$size.

```
db.users.find ({languages: {$size:2}})
```

Оператор \$exists

Оператор извлекает документы, имеющих заданный ключ в принципе:

```
db.users.find ({company: {$exists:true}})
```

Оператор \$regex

Оператор указывает регулярное выражение, которому соответствует искомое поле:

```
db.users.find ({name: {$regex:"b"}})
```

Обновление данных

Наравне с MySQL СУБД MongoDB может менять данные. Самый простой метод – save. Ему передаётся id документа, данные которого надо обновить. В случае, если по данному id документ не найден, СУБД производит вставку новой записи.

```
db.users.save({name: "Eugene", age : 29, languages: ["english", "german", "spanish"]})
```

Данная функция по результатам работы вернёт объект WriteResult:

```
WriteResult({"nInserted" : 1 })
```

Но save работает довольно просто. Усложним подход и станем использовать update. Эта функция на вход принимает три параметра:

- query: запрос на поиск;
- objNew: новый документ после обновления;
- options: принимает два аргумента: upsert и multi.

Обновление отдельного поля

В некоторых ситуациях вам нужно обновить не весь документ, а только один ключ в нём. Специально для этих целей в MongoDB есть функция \$set. Если указанный документ не содержит указанное поле, оно будет создано.

```
db.users.update({name : "Eugene", age: 29}, {$set: {age : 30}})
```

Удаление поля

Чтобы удалить поле, используйте \$unset:

```
db.users.update({name : "Tom"}, {$unset: {salary: 1}})
```

Не волнуйтесь – если ключа в документе нет, оператор просто завершит работу. Помимо этого у вас есть возможность обновить несколько полей сразу:

```
db.users.update({name : "Tom"}, {$unset: {salary: 1, age: 1}})
```

Наиболее полный список функций вы всегда сможете найти в [официальной документации MongoDB](#).

Особенности работы MongoDB

При работе с MongoDB непременно нужно знать технические особенности реализации данного решения.

Во-первых, данное хранилище использует достаточно строгие блокировки. И если в базу идёт активная запись, остальные пользователи будут становиться в очередь на чтение. Это нужно иметь в виду при построении приложения.

Во-вторых, MongoDB плохо чистит занимаемое пространство на HDD. Удаляемые логически записи по факту продолжают занимать место, пока вы не запустите сжатие, которое требует, чтобы на диске было доступно как минимум столько же места, сколько уже весит ваша база.

В-третьих, документы в Mongo имеют лимит объёма в 16 мегабайт. Таким образом, для больших агрегаций пригодится механизм map/reduce.

В-четвертых, MongoDB неустойчива по отношению к аварийному выключению. Дело в том, что все используемые данные при работе хранятся в RAM, и лишь спустя какое-то время фиксируются на более медленном HDD. При нештатном отключении данные просто теряются. В последних версиях появилось журналирование, которое позволяет восстанавливать аварийно потерянные данные.

Бэкап в Mongo может быть произведен двумя способами. Можно просто скопировать файлы хранилища. Но, как уже было сказано ранее, есть риск того, что в них не будет храниться вся актуальная информация. Поэтому предпочтительнее применять команду mongodump. Она работает как и mysqldump, собирая дампы данных. Этот дампы можно развернуть при помощи команды mongorestore. Но лучше всего при запуске mongodbg настроить репликацию master-slave, снимая резервные копии со slave, чтобы не создавать нагрузку на сервер оригинальных данных. Также slave будет полезен, если умрет master.

Практическая работа

Практическая часть будет состоять из работы с MongoDB. Для начала нужно познакомиться с утилитой `mongodump`. Чтобы сделать полный экспорт содержимого и данных схемы, можно воспользоваться командой:

```
mongodump
```

Данная команда должна быть запущена из новой сессии терминала (при наличии уже запущенного `Mongo shell`). Данная команда будет использовать стандартный хост и порт, то есть выглядеть это будет вот так:

```
mongodump --host localhost --port 27017
```

Вы можете поменять флаги, если вам нужно подключиться к другому серверу.

При запуске данные будут извлекаться в ту же директорию, где и была запущена команда. Чтобы изменить директорию, можно использовать следующий флаг:

```
mongodump --out /data/backup/
```

Импорт определенной коллекции:

```
mongodump --collection myCollection --db test
```

Чтобы сделать экспорт нелокальной базы данных, можно воспользоваться такой командой:

```
mongodump --host mongodb1.example.net --port 3017 --username user --password pass --out /opt/backup/mongodump-2013-10-24
```

Чтобы восстановить данные из дампа, нужно воспользоваться командой:

```
mongorestore --port <port number> <path to the backup>
```

Восстановить данные из дампа:

```
mongorestore dump-2013-10-25/
```

Также можно проделать импорт из файла json типа:

```
mongoimport --db test --collection docs --file primer-dataset.json
```

Давайте сделаем эти действия на примере нашей схемы (схема будет приложена к уроку).

Давайте попрактикуемся с выборками данных. Для начала посмотрим список баз данных и коллекций, которые в них есть.

Показать список схем:

```
show dbs;
```

Далее используем созданную схему:

```
use test;
```

Какие коллекции есть в используемой схеме:

```
show collections;
```

Делаем пробную выборку данных:

```
db.docs.find();
```

Выборка с условием:

```
db.docs.find({restaurant_id: "40356731"});
```

Любые запросы далее на выбор учителя.

Практическое задание

1. Настроить и запустить master-сервер.
2. Установить MongoDB и повторить запросы из методички.

Дополнительные материалы

1. «MySQL 5» – Игорь Симдянов, Максим Кузнецов.
2. «Обеспечение высокой доступности систем на основе MySQL» – Чарльз Белл, Мэтс Киндал.
3. «MongoDB в действии» – Кайл Бэнкер.

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. <http://ruhighload.com/post/%D0%9A%D0%B0%D0%BA+%D0%BD%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B8%D1%82%D1%8C+MySQL+Master-Slave+%D1%80%D0%B5%D0%BF%D0%BB%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8E>
2. <https://prudnitskiy.pro/2015/5/1/xtradb-quickstart/>
3. <https://habrahabr.ru/post/152477/>
4. <https://habrahabr.ru/post/127290/>
5. <http://metanit.com/nosql/mongodb/2.1.php>
6. <https://docs.mongodb.com/getting-started/shell/import-data/>
7. <https://docs.mongodb.com/v3.2/tutorial/install-mongodb-on-os-x/#run-mongodb>