第三十五章：Mysql数据库应用部署及Sql操作

**一、数据库的概述；**

**二、数据库的分类；**

**三、Mysql的概述；**

**四、Mysql存储原理；**

**五、案例：centos 7搭建Mysql 5.7；**

**六、Sql语句操作之增删改查；**

**七、Sql语句操作之授权；**

**八、Mysql基本优化操作；**

**九、Mysql存储引擎的应用；**

**十、Mysql配置文件详解：**

**一、数据库的概述；**

**概述：**数据库(Database)是按照数据结构组织、存储和管理数据的仓库，它产生于距今六十多年前，数据库有很多种类型，从最简单的存储有各种数据的表格到能够进行海量数据存储的大型数据库系统，在各个方面得到了广泛的应用。在信息化社会，充分有效地管理和利用各类信息资源，是进行科学研究和决策管理的前提条件。数据库技术是管理信息系统、办公自动化系统、决策支持系统等各类信息系统的核心部分，是进行科学研究和决策管理的重要技术手段。

**二、数据库的分类；**

**关系型数据库：**数据拥有固定的存储结构，通过库--表--行--列的方式存储，存储时会有表的结构化关系，过程如下：解析sql语句--连接层--磁盘存取--结构化成表，；

优势：1.容易理解，二维表的结构非常贴近现实世界，二维表格，容易理解；

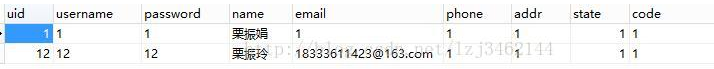
2.使用方便，通用的sql语句使得操作关系型数据库非常方便；

3.易于维护，数据库的ACID属性，大大降低了数据冗余和数据不一致的概率；

瓶颈：1.海量数据的读写效率，对于网站的并发量高，往往达到每秒上万次的请求，对于传统关系型数据库来说，硬盘I/O是一个很大的挑战。

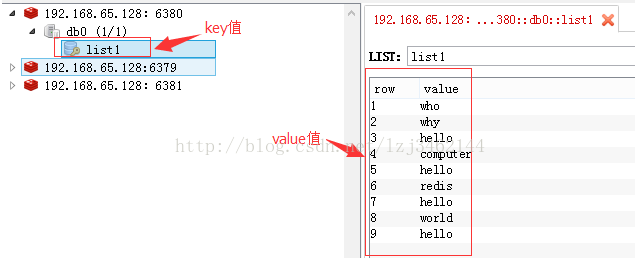
2.高扩展性和可用性，在基于web的结构中，数据库是最难以横向拓展的，当一个应用系统的用户量和访问量与日俱增的时候，数据库没有办法像web Server那样简单的通过添加更多的硬件和服务节点来拓展性能和负载能力。

****



**关系型数据库到非关系型数据库的演变：**关系型数据库的最大优点就是事务的一致性，这个特性，使得关系型数据库中可以适用于一切要求一致性比较高的系统中。比如：银行系统。但是在大部分网页应用中，对这种一致性的要求不是那么的严格，允许有一定的时间间隔，所以关系型数据库这个特点不是那么的重要了。相反，关系型数据库为了维护一致性所付出的巨大代价就是读写性能比较差。而像微博、facebook这类应用，对于并发读写能力要求极高，关系型数据库已经无法应付。所以必须用一种新的数据结构存储来替代关系型数据库。所以非关系型数据库应用而生。

**非关系型数据库：**NoSQL主要指那些非关系型的、分布式的，主要代表MongoDB，Redis、CouchDB。NoSQL提出了另一种理念，以键值来存储，且结构不稳定，每一个元组都可以有不一样的字段，这种就不会局限于固定的结构，可以减少一些时间和空间的开销。使用这种方式，为了获取用户的不同信息，不需要像关系型数据库中，需要进行多表查询。仅仅需要根据key来取出对应的value值即可，所以避免了关系型数据库复杂的查询关系，可以大大增加查询的效率；

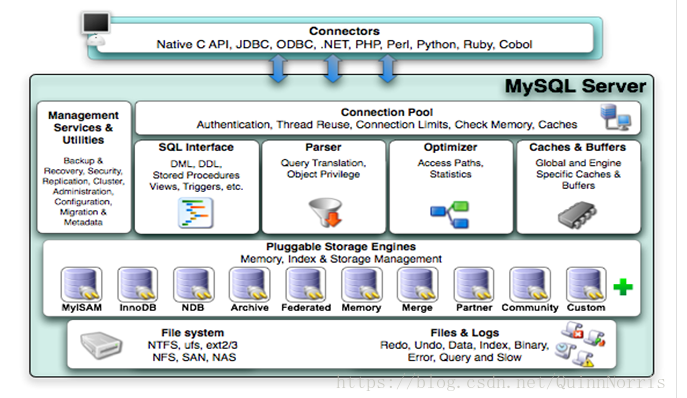


**三、Mysql的概述：**

**概述：**MySQL 是一个关系型数据库管理系统，由瑞典 MySQL AB 公司开发，目前属于 Oracle 旗下公司。是一个真正的多用户、多线程SQL数据库服务器。SQL（结构化查询语言）是世界上最流行的和标准化的数据库语言。MySQL是以一个客户机/服务器结构的实现,它由一个服务器守护程序mysqld和很多不同的客户程序和库组成。   SQL是一种标准化的语言,它使得存储、更新和存取信息更容易；

**四、Mysql存储原理：**

**Mysql整体架构：**

****

1.连接池：最上层负责和客户端进行连接，比如jdbc，odbj这样的数据库连接的API，在这一层有连接池的概念，类似于线程池，连接池可以同时处理很多个数据库请求。同时这一层有SSL的安全概念，可以确保连接是安全的；

2.SQL接口：当SQL语句进入MySQL后，会先到SQL接口中，这一层是封装层，将传过来的SQL语句拆散，将底层的结果封装成SQL的数据格式；

3.解析器：这一层负责将SQL语句进行拆分，验证，如果语句有问题那么就返回错误，如果没问题就继续向下执行；

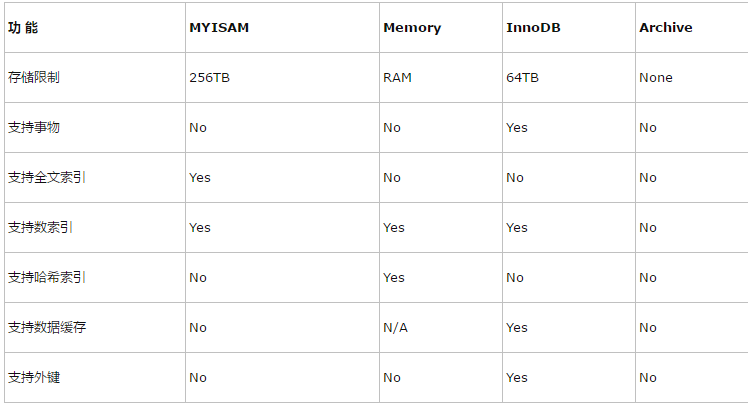
4.优化器：对SQL查询的结果优化处理，产生多种执行计划，最终数据库会选择最优化的方案去执行，尽快返会结果。比如select a,b from c where d。在这里会先查询c表中符合d的数据并将他们的ab项进行投影，返回结果，并不会直接把整张表查出来；

5.缓存：对要查询的SQL语句进行hash后缓存，如果下一次是相同的查询语句，则在SQL接口之后直接返回结果；

6.存储引擎：MySQL有很多种存储引擎，每一种存储引擎有不同的特性，他们负责组织文件的存放形式，位置，访问文件的方法等等。比较常用的有innoDB，MyISAM，MAMORY等；

7.文件系统：真正存放物理文件的单位；

**存储引擎：**



1.MyISAM：是mysql 5.5版本之前的默认的存储引擎，它的前身是ISAM，MyISAM管理非事务表，提供告诉存储和检索，以及全文搜索能力，从而受到web开发者的喜爱；MyISAM引擎创建数据库，将产生3个文件。文件的名字以表名字开始，扩展名之处文件类型：frm文件存储表定义、数据文件的扩展名为.MYD（MYData）、索引文件的扩展名时.MYI（MYIndex）；

优点：查询访问较快，因为读的时候会加表锁；

支持多种存储方式：静态表，动态表，压缩表等；

存储总行数；

通过key\_buffer\_size设置缓存索引，调高访问的性能，减少磁盘I/O的压力；

缺点：写入效率慢，因为写的时候会给整张表加锁；

没有事务的概念；

数据恢复起来比较困难；

数据写入过程中阻塞用户对表内数据的读取；

数据读取过程中阻塞用户对表内数据的写入；

生产环境场景：

公司业务不需要事务的支持；

一般单方面的读取数据的需求较多，或者单方面写入数据的需求较多，不适合双方面；

服务器硬件资源相对比较差；

2.innoDB：支持事务，具有事务提交、回滚、崩溃恢复等机制，它的设计是为处理巨大数据量时的最大性能设置的，它的CPU效率是任何磁盘的关系数据库引擎所不能匹敌的，InnoDB存储引擎完全与Mysql服务器整合，为在内存中缓存数据和索引而维持它自己的缓冲池，InnoDB表可以是任何尺寸，即使在文件尺寸被限制为2GB的操作系统上也无影响；InnoDB不创建目录，使用InnoDB时，MySQL将在MySQL数据目录下创建一个名为ibdata1的10MB大小的自动扩展数据文件，以及两个名为ib\_logfile0和ib\_logfile1的5MB大小的日志文件；

优点：支持事务，事务隔离级别；

支持外键，具有安全的约束性；

主键是唯一的聚集索引，和数据存放在一起，效率高；

可以有非聚集索引，非聚集索引单独存放，可以通过其查到聚集索引；

支持分区、表空间、类似于Oracle；

缺点：占用磁盘较多，占用内存较多（缓存）；

读效率慢于MyISAM；

不存储总行数；

生产环境场景：

业务需要事务的支持；

行级锁定对高并发有很好的适应能力；

业务数据更新较为频繁的场景，如微博，论坛等；

业务数据一致性要求较高，比如银行业务；

硬件设备内存较大，利用InnoDB较好的缓存能力来提高内存利用率；

总结：

* 如果要提供读少写多，并且需要提交、回滚、崩溃恢复能力的事物安全能力，并要求实现并发控制，InnoDB是一个好的选择；
* 如果数据表主要用来读多写少，对原子性要求不高，则MyISAM引擎能提供较高的处理效率；
* 如果只是临时存放数据，数据量不大，并且不需要较高的数据安全性，可以选择将数据保存在内存中的Memory引擎，MySQL中使用该引擎作为临时表，存放查询的中间结果；
* 如果只有INSERT和SELECT操作，可以选择Archive，Archive支持高并发的插入操作，但是本身不是事务安全的。Archive非常适合存储归档数据，如记录日志信息可以使用Archive；

使用哪一种引擎需要灵活选择，一个数据库中多个表可以使用不同引擎以满足各种性能和实际需求，使用合适的存储引擎，将会提高整个数据库的性能，在数据库中可以使用命令show engines查看存储引擎；

**五、案例：centos 7搭建Mysql 5.7；**

**案例环境：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 系统类型 | IP地址 | 主机名 | 所需软件 | 硬件 |
| Centos 7.4 1708 64bit | 192.168.100.101 | my.linuxfan.cn | mysql-5.7.12.tar.gz  boost\_1\_59\_0.tar.gz | 内存：4G  CPU核心：2 |

**案例步骤：**

* 下载安装Mysql软件程序；
* 优化调整Mysql程序；
* 初始化Mysql数据库服务；
* 测试连接访问数据库；
* **下载安装Mysql软件程序；**

[root@my ~]# wget http://dev.mysql.com/Downloads/MySQL-5.7/mysql-5.7.12.tar.gz

[root@my ~]# wget https://sourceforge.net/projects/boost/files/boost/1.59.0/boost\_1\_59\_0.tar.gz/download ##需要使用windows下载，不能使用linux下载

[root@my ~]# yum -y remove mysql-\* boost-\*

[root@my ~]# rpm -qa mysql

[root@my ~]# rpm -qa boost

[root@my ~]# yum -y install gcc gcc-c++ ncurses bison libgcrypt perl cmake

[root@my ~]# mount /dev/cdrom /mnt

[root@my ~]# rpm -ivh /mnt/Packages/ncurses-devel-5.9-13.20130511.el7.x86\_64.rpm --nodeps

[root@my ~]# tar -zxvf boost\_1\_59\_0.tar.gz

[root@my ~]# mv boost\_1\_59\_0 /usr/local/boost/

[root@my ~]# groupadd mysql

[root@my ~]# useradd -r -g mysql mysql

[root@my ~]# tar zxvf mysql-5.7.12.tar.gz -C /usr/src/

[root@my ~]# cd /usr/src/mysql-5.7.12/ ##编译多次时，需要删除CMAKE的缓存，rm -rf /usr/src/mysql-5.7.12/CMakeCache.txt

[root@my mysql-5.7.12]# cmake . -DCMAKE\_INSTALL\_PREFIX=/usr/local/mysql -DMYSQL\_DATADIR=/usr/local/mysql/data \

-DDEFAULT\_CHARSET=utf8 -DDEFAULT\_COLLATION=utf8\_general\_ci \

-DMYSQL\_TCP\_PORT=3306 -DMYSQL\_USER=mysql -DWITH\_MYISAM\_STORAGE\_ENGINE=1 -DWITH\_INNOBASE\_STORAGE\_ENGINE=1 \

-DWITH\_ARCHIVE\_STORAGE\_ENGINE=1 -DWITH\_BLACKHOLE\_STORAGE\_ENGINE=1 \

-DWITH\_MEMORY\_STORAGE\_ENGINE=1 -DENABLE\_DOWNLOADS=1 -DDOWNLOAD\_BOOST=1 -DWITH\_BOOST=/usr/local/boost -DSYSCONFDIR=/etc

注释：

-DCMAKE\_INSTALL\_PREFIX=/usr/local/mysql [MySQL安装的根目录]

-DMYSQL\_DATADIR=/mydata/mysql/data [MySQL数据库文件存放目录]

-DDEFAULT\_CHARSET=utf8 [设置默认字符集为utf8]

-DDEFAULT\_COLLATION=utf8\_general\_ci [设置默认字符校对]

-DMYSQL\_TCP\_PORT=3306 [MySQL的监听端口]

-DMYSQL\_USER=mysql [MySQL用户名]

-DWITH\_MYISAM\_STORAGE\_ENGINE=1 [安装MySQL的myisam数据库引擎]

-DWITH\_INNOBASE\_STORAGE\_ENGINE=1 [安装MySQL的innodb数据库引擎]

-DWITH\_ARCHIVE\_STORAGE\_ENGINE=1 [安装MySQL的archive数据库引擎]

-DWITH\_BLACKHOLE\_STORAGE\_ENGINE=1 [安装MySQL的blackhole数据库引擎]

-DWITH\_MEMORY\_STORAGE\_ENGINE=1 [安装MySQL的memory数据库引擎]

-DENABLE\_DOWNLOADS=1 [编译时允许自主下载相关文件]

-DDOWNLOAD\_BOOST=1 [允许下载BOOST]

-DWITH\_BOOST=/usr/local/boost [指定系统中存在的BOOST]

-DSYSCONFDIR=/etc [MySQL配置文件所在目录]

-DWITH\_READLINE=1 [MySQL的readline library]

-DMYSQL\_UNIX\_ADDR=/var/run/mysql/mysql.sock [MySQL的通讯目录]

-DENABLED\_LOCAL\_INFILE=1 [启用加载本地数据]

-DWITH\_PARTITION\_STORAGE\_ENGINE=1 [启动mysql的分区存储结构]

-DEXTRA\_CHARSETS=all [使MySQL支持所有的扩展字符]

-DWITH\_DEBUG=0 [禁用调试模式]

-DMYSQL\_MAINTAINER\_MODE=0

-DWITH\_SSL:STRING=bundled [通讯时支持ssl协议]

-DWITH\_ZLIB:STRING=bundled [允许使用zlib library]

[root@my ~]# make -j `cat /proc/cpuinfo | grep processor| wc -l`

[root@my ~]# make install

[root@my ~]# ls /usr/local/mysql

bin COPYING data docs include lib logs man mysql-test README share support-files

* **优化调整Mysql程序；**

[root@my ~]# chown -R mysql:mysql /usr/local/mysql

[root@my ~]# cp /usr/src/mysql-5.7.12/support-files/my-default.cnf /etc/my.cnf

[root@my ~]# cp /usr/src/mysql-5.7.12/support-files/mysql.server /etc/init.d/

[root@my ~]# chmod +x /etc/init.d/mysql.server

[root@my ~]# cat <<END >>/usr/lib/systemd/system/mysqld.service ##编写服务控制脚本

[Unit]

Description=mysqldapi

After=network.target

[Service]

Type=forking

PIDFile=/usr/local/mysql/logs/mysqld.pid

ExecStart=/etc/init.d/mysql.server start

ExecReload=/etc/init.d/mysql.server restart

ExecStop=/etc/init.d/mysql.server stop

PrivateTmp=Flase

[Install]

WantedBy=multi-user.target

END

注解：

PrivateTmp=Flase ##此配置必须关闭，不然mysql连接文件mysql.sock文件会默认生成在以下位置/tmp/systemd-private-83bba738e8ff4837b5ae657eff983821-mysqld.service-BPxWpJ/tmp/mysql.sock，导致数据库无法连接，将此配置项关闭后，则文件正常生成在/tmp/mysql.sock

[root@my ~]# echo "export PATH=$PATH:/usr/local/mysql/bin/" >>/etc/profile

[root@my ~]# source /etc/profile

* **初始化Mysql数据库服务；**

[root@my ~]# mysqld --initialize-insecure --user=mysql --basedir=/usr/local/mysql --datadir=/usr/local/mysql/data

注解：

--initialize-insecure ##禁用mysql的密码策略（密码复杂性等），--initializeaize是开启密码策略，自动生成密码在mysqld.log文件中

--user=mysql ##运行的账户

--basedir=/usr/local/mysql ##mysql的安装位置

--datadir=/usr/local/mysql/data ##mysql数据库服务数据的物理存放路径

[root@my ~]# cat <<END >/etc/my.cnf

[mysqld]

basedir = /usr/local/mysql

datadir = /usr/local/mysql/data

port = 3306

sql\_mode=NO\_ENGINE\_SUBSTITUTION,STRICT\_TRANS\_TABLES

character\_set\_server=utf8

init\_connect='SET NAMES utf8'

log-error=/usr/local/mysql/logs/mysqld.log

pid-file=/usr/local/mysql/logs/mysqld.pid

skip-name-resolve

END

注解：

[mysqld] ##声明区域

basedir = /usr/local/mysql ##mysql的安装位置

datadir = /usr/local/mysql/data ##mysql的物理文件存放位置

port = 3306 ##mysql服务监听的端口

sql\_mode=NO\_ENGINE\_SUBSTITUTION,STRICT\_TRANS\_TABLES ##mysql的模式

character\_set\_server=utf8 ##字符集

init\_connect='SET NAMES utf8'

log-error=/usr/local/mysql/logs/mysqld.log ##指定日志文件位置

pid-file=/usr/local/mysql/logs/mysqld.pid ##指定运行服务所产生的pid文件位置

skip-name-resolve ##跳过mysql的域名反向解析

[root@my ~]# mkdir /usr/local/mysql/logs

[root@my ~]# chown mysql:mysql /usr/local/mysql/logs/

[root@my ~]# systemctl start mysqld

[root@my ~]# systemctl enable mysqld

Created symlink from /etc/systemd/system/multi-user.target.wants/mysqld.service to /usr/lib/systemd/system/mysqld.service.

[root@my ~]# netstat -utpln |grep mysqld

tcp 0 0 0.0.0.0:3306 0.0.0.0:\* LISTEN 2885/mysqld

* **测试连接访问数据库；**

[root@my ~]# mysql ##登录mysql设置密码

mysql> alter user 'root'@'localhost' identified by '123123';

mysql> exit

[root@my ~]# mysql -uroot -p123123

mysql> exit

**六、Sql语句操作之增删改查；**

**Sql语句分类：**

* DDL：数据定义语言，用来建立数据库，数据对象和定义其列，如create、alter、drop；
* DML：数据操纵语言，用来查询、插入、删除、修改数据库中的数据，如select、insert、update、delete；
* DCL：数据控制语言，用来控制数据库组件的存取许可，存取权限等，如commit、rollback、grant、revoke；

**数据记录类型：**

日期：



小数类型：



字符串：

****

**使用navicat\_mysql工具进行连接数据库，介绍工具使用：**

**查看数据库：**

mysql> show databases; ##查看数据库

+--------------------+

| Database |

+--------------------+

| information\_schema |

| haha |

| mysql |

| performance\_schema |

| sys |

+--------------------+

mysql> use mysql; ##进入数据库mysql

Database changed

mysql> show tables; ##查看表

...

mysql> desc user; ##查看表的结构，表头

**查看数据记录：**

mysql> select \* from user \G; ##查询user表中的所有数据记录，按行显示

mysql> select host,user from user; ##指定user表的字段进行查询

+-----------+-----------+

| host | user |

+-----------+-----------+

| localhost | mysql.sys |

| localhost | root |

+-----------+-----------+

mysql> delete from mysql.user where user=""; ##删除用户为空的数据记录

**创建数据库：**

mysql> create database auth; ##创建数据库auth

Query OK, 1 row affected (0.00 sec)

mysql> show databases;

+--------------------+

| Database |

+--------------------+

| information\_schema |

| auth |

| haha |

| mysql |

| performance\_schema |

| sys |

+--------------------+

6 rows in set (0.00 sec)

**创建表：**

mysql> create table auth.users(user\_name char(16) not null, user\_passwd char(48) default '',primary key (user\_name)); ##创建表auth.users

Query OK, 0 rows affected (0.00 sec)

mysql> desc users; ##查看表的结构

+-------------+----------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+-------------+----------+------+-----+---------+-------+

| user\_name | char(16) | NO | PRI | NULL | |

| user\_passwd | char(48) | YES | | | |

+-------------+----------+------+-----+---------+-------+

**创建数据记录：**

mysql> insert into auth.users values('hehe','pwd@123'); ##新增数据记录

Query OK, 1 row affected (0.00 sec)

mysql> select \* from auth.users; ##查看表的数据记录

+-----------+-------------+

| user\_name | user\_passwd |

+-----------+-------------+

| hehe | pwd@123 |

+-----------+-------------+

1 row in set (0.00 sec)

**更新数据记录：**

mysql> update auth.users set user\_name="haha" where user\_name="hehe"; ##更新数据记录

Query OK, 1 row affected (0.00 sec)

mysql> select \* from auth.users;

+-----------+-------------+

| user\_name | user\_passwd |

+-----------+-------------+

| haha | pwd@123 |

+-----------+-------------+

1 row in set (0.00 sec)

**新增数据结构（列）：**

mysql> alter table auth.users add (user\_class char(16)); ##新增数据结构（列）

Query OK, 1 row affected (0.01 sec)

mysql> select \* from auth.users;

+-----------+-------------+------------+

| user\_name | user\_passwd | user\_class |

+-----------+-------------+------------+

| haha | pwd@123 | NULL |

+-----------+-------------+------------+

1 row in set (0.00 sec)

**更新数据结构的名称：**

mysql> alter table auth.users change user\_class user\_db integer; ##更新数据结构名

Query OK, 1 row affected (0.01 sec)

mysql> select \* from auth.users;

+-----------+-------------+---------+

| user\_name | user\_passwd | user\_db |

+-----------+-------------+---------+

| haha | pwd@123 | NULL |

+-----------+-------------+---------+

1 row in set (0.00 sec)

**更新数据记录：**

mysql> update auth.users set user\_db="1" where user\_name="haha"; ##更新数据记录

Query OK, 1 row affected (0.00 sec)

mysql> select \* from auth.users;

+-----------+-------------+------------+

| user\_name | user\_passwd | user\_db |

+-----------+-------------+------------+

| haha | pwd@123 | 1 |

+-----------+-------------+------------+

1 row in set (0.00 sec)

**删除数据记录：**

mysql> delete from auth.users where user\_name=haha; ##删除数据记录

Query OK, 1 row affected (0.00 sec)

mysql> select \* from auth.users;

**删除数据结构（列）：**

mysql> alter table auth.users drop user\_db; ##删除数据结构（列）

Query OK, 1 row affected (0.00 sec)

mysql> select \* from auth.users;

+-----------+-------------+

| user\_name | user\_passwd |

+-----------+-------------+

| haha | pwd@123 |

+-----------+-------------+

1 row in set (0.00 sec)

**更新数据结构的名称：**

mysql> alter table users change user\_name user\_db char(16); ##更新数据结构名

Query OK, 0 rows affected (0.00 sec)

mysql> select \* from users;

+-----------+-------------+

| user\_db | user\_passwd |

+-----------+-------------+

| haha | pwd@123 |

+-----------+-------------+

1 row in set (0.00 sec)

**更新数据结构（列）的数据类型：**

mysql> desc auth.users; ##查看更改前的数据结构

+-------------+----------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+-------------+----------+------+-----+---------+-------+

| user\_name | char(16) | NO | PRI | NULL | |

| user\_passwd | char(48) | YES | | | |

| user\_db | int(11) | YES | | NULL | |

+-------------+----------+------+-----+---------+-------+

mysql> alter table auth.users modify user\_name char(48); ##更改第一列的数据库结构为char（48）

Query OK, 1 row affected (0.00 sec)

mysql> desc auth.users; ##查看更改后的数据结构

+-------------+----------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+-------------+----------+------+-----+---------+-------+

| user\_name | char(48) | NO | PRI | NULL | |

| user\_passwd | char(48) | YES | | | |

| user\_db | int(11) | YES | | NULL | |

+-------------+----------+------+-----+---------+-------+

**删除表和库：**

mysql> drop table auth.users; ##删除表users

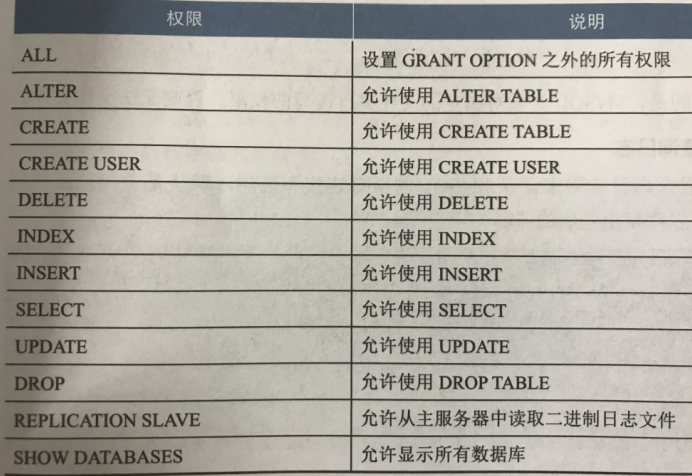
Query OK, 0 rows affected (0.00 sec)

mysql> drop database auth; ##删除库auth

Query OK, 0 rows affected (0.00 sec)

**七、Sql语句操作之授权；**

**权限列表：**

****

**语法：**

grant all on 库.\* to 用户@客户机地址 identified by ‘密码’; ##授权语法

show grants for 用户@客户机地址； ##查看权限

revoke 权限列表 on 库.\* from 用户@客户机地址; ##取消授权

flush privileges; ##刷新权限

drop user '用户名'@'客户机地址'； ##删除账户

**特殊表示：**

%表示所有地址；

192.168.100.%表示整个网段；

**八、Mysql基本优化操作；**

**1.忘记密码：**

[root@my ~]# vi /etc/my.cnf

[mysqld]

skip-grant-tables ##添加该行，跳过密码验证

:wq

[root@my ~]# systemctl restart mysqld

[root@my ~]# mysql ##登录后操作

mysql> use mysql;

Database changed

mysql> update user set authentication\_string=password('123456') where user="root";

mysql> select Host,user,authentication\_string from user;

+-----------+-----------+-------------------------------------------+

| Host | user | authentication\_string（原mysql 5.5的password字段） |

+-----------+-----------+-------------------------------------------+

| localhost | root | \*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9 |

| localhost | mysql.sys | \*THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE |

+-----------+-----------+-------------------------------------------+

2 rows in set (0.01 sec)

[root@my ~]# vi /etc/my.cnf

[mysqld]

#skip-grant-tables ##注释该行

:wq

[root@my ~]# systemctl restart mysqld

**2.解决mysql乱码问题：**

[root@my ~]# vi /et/my.cnf

[client]

default-character-set=utf8

[root@my ~]# systemctl restart mysqld

**3.Mysql的日志文件类型：**

a.错误日志：记录启动、运行或停止mysqld时出现的问题；

log-error=d:/mysql\_log\_err.txt

b.查询日志：记录建立的客户端连接和执行的所有语句（包括错误的）；

log=/usr/local/mysql/log/mysql\_log.txt

c.二进制日志：记录所有更改数据的语句、还用于主从复制；

log-bin=/usr/local/mysql/log/mysql\_log\_bin

d.慢日志：记录所有执行时间超过long\_query\_time秒的所有查询；

long\_query\_time =1 ##时间，单位秒

log-slow-queries= /usr/local/mysql/log/mysql\_log\_slow.txt

**4.Mysql的慢查询：**

**概述：**慢查询主要用于跟踪异常的sql语句，可以分析出当前程序哪些sql语句比较耗费资源，慢查询日志则是用来记录在mysql中响应时间超过阈值的语句；

**导致慢查询的因素：**

1、没有索引或者没有用到索引(这是查询慢最常见的问题，是程序设计的缺陷) ；  
2、磁盘I/O吞吐量小，形成了瓶颈效应；  
3、没有创建计算列导致查询不优化；  
4、内存不足 ；  
5、网络速度慢 ；  
6、查询出的数据量过大（可以采用多次查询，其他的方法降低数据量） ；  
7、锁或者死锁(这也是查询慢最常见的问题，是程序设计的缺陷) ；  
8、sp\_lock,sp\_who,活动的用户查看,原因是读写竞争资源；  
9、返回了不必要的行和列 ；  
10、查询语句不好，没有优化 ；

**解决慢查询的方法：**

1、把数据、日志、索引放到不同的I/O设备上，增加读取速度；  
2、纵向、横向分割表，减少表的尺寸；  
3、升级硬件（内存、cpu核心数）；  
4、根据查询条件,建立索引,优化索引、优化访问方式，限制结果集的数据量。注意填充因子要适当（最好是使用默认值0）。索引应该尽量小，使用字节数小的列建索引好（参照索引的创建）,不要对有限的几个值的字段建单一索引如性别字段 ；  
5、在数据库前段简历内存缓存机制；

**配置：**

[root@my ~]# mysql -uroot -p123123

mysql> show variables like "%slow%";

+---------------------------+-----------------------------------+

| Variable\_name | Value |

+---------------------------+-----------------------------------+

| log\_slow\_admin\_statements | OFF |

| log\_slow\_slave\_statements | OFF |

| slow\_launch\_time | 2 |

| slow\_query\_log | OFF |

| slow\_query\_log\_file | /usr/local/mysql/data/my-slow.log |

+---------------------------+-----------------------------------+

5 rows in set (0.01 sec)

mysql> exit

[root@my ~]# cat <<END >>/etc/my.cnf

slow\_query\_log=ON

long\_query\_time=0.01

slow\_query\_log\_file=/usr/local/mysql/data/localhost-slow.log

END

[root@my ~]# systemctl restart mysqld

[root@my ~]# mysql -uroot -p123123

mysql> show variables like "%slow%";

+---------------------------+------------------------------------------+

| Variable\_name | Value |

+---------------------------+------------------------------------------+

| log\_slow\_admin\_statements | OFF |

| log\_slow\_slave\_statements | OFF |

| slow\_launch\_time | 2 |

| slow\_query\_log | ON |

| slow\_query\_log\_file | /usr/local/mysql/data/localhost-slow.log |

+---------------------------+------------------------------------------+

5 rows in set (0.00 sec)

mysql> show variables like 'long\_query\_time';

+-----------------+----------+

| Variable\_name | Value |

+-----------------+----------+

| long\_query\_time | 0.010000 |

+-----------------+----------+

1 row in set (0.00 sec)

mysql> select sleep(2);

+----------+

| sleep(2) |

+----------+

| 0 |

+----------+

1 row in set (2.00 sec)

mysql> exit

[root@my ~]# cat /usr/local/mysql/data/localhost-slow.log

/usr/local/mysql/bin/mysqld, Version: 5.7.12-log (Source distribution). started with:

Tcp port: 3306 Unix socket: (null)

Time Id Command Argument

# Time: 2018-10-31T20:02:42.719632Z

# User@Host: root[root] @ localhost [] Id: 3

# Query\_time: 2.000750 Lock\_time: 0.000000 Rows\_sent: 1 Rows\_examined: 0

SET timestamp=1541016162;

select sleep(2);

**5.Mysqldumpslow命令用法：**

**参数：**  
-s：排序方式，按锁的时间l、返回的记录数r、查询的时间t、记录的次数c、倒序的话可以加r  
-t：查询前多少条记录  
-g：支持正则表达式，以及忽略大小写

**案例：**  
a.查询按时间返回前5条日志信息，格式如下：  
mysqldumpslow -s t -t 5 /usr/local/mysql/data/localhost-slow.log

b.查看有关select语句按时间返回前5条日志信息，格式如下：

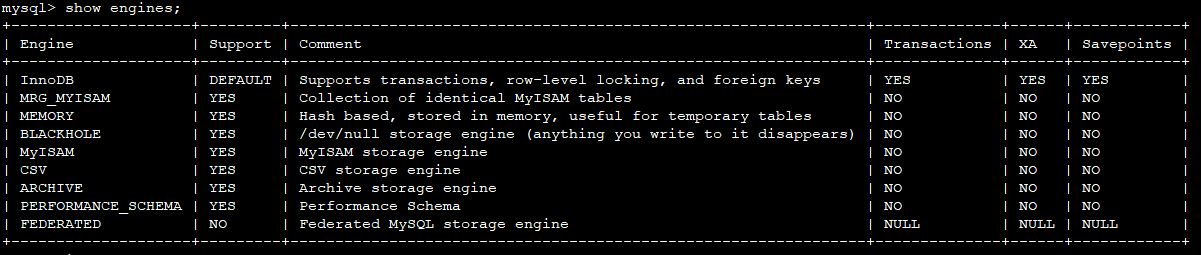
mysqldumpslow -s t -t 5 -g "select" /usr/local/mysql/data/localhost-slow.log

c.查看按返回行数从大到小，查看前两行，格式如下：

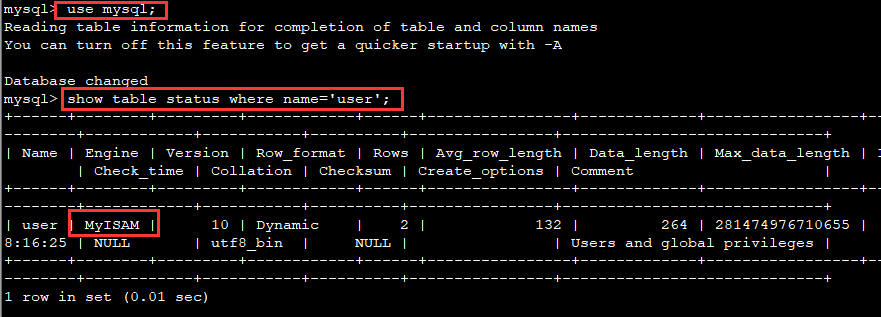
mysqldumpslow -s r -t 2 /usr/local/mysql/data/localhost-slow.log

**九、Mysql存储引擎的应用；**

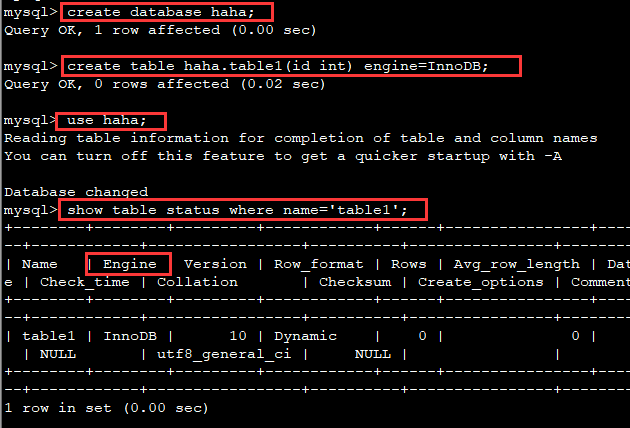
1.查看数据库可以配置的存储引擎类型：



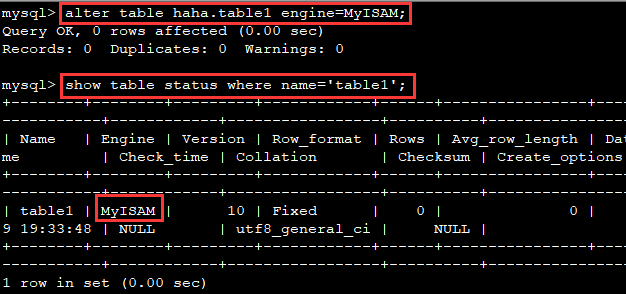
2.查看表正在使用的存储引擎：



3. 创建表并且指定存储引擎；



4.修改已经存在表的存储引擎：



5.修改Mysql服务的默认存储引擎：

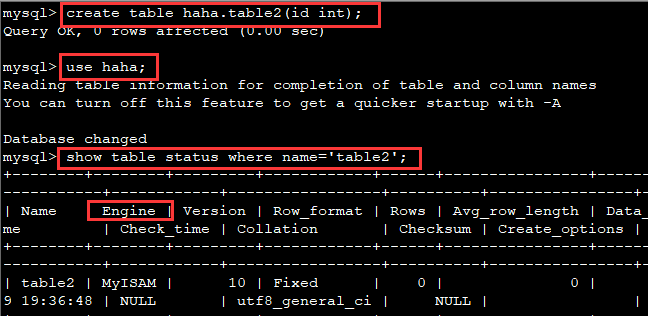
[root@my ~]# vi /etc/my.cnf ##新添配置项

[mysqld]

default-storage-engine=MyISAM

[root@my ~]# systemctl restart mysqld

[root@my ~]# mysql -uroot -p123123



**十、Mysql配置文件详解：**

[root@my ~]# vi /etc/my.cnf

[mysqld]

########basic settings########

basedir = /usr/local/mysql

datadir = /usr/local/mysql/data

port = 3306

sql\_mode=NO\_ENGINE\_SUBSTITUTION,STRICT\_TRANS\_TABLES

character\_set\_server=utf8

init\_connect='SET NAMES utf8'

log-error=/usr/local/mysql/logs/mysqld.log

pid-file=/usr/local/mysql/logs/mysqld.pid

user = mysql

symbolic-links=0

server-id = 11

#bind\_address = 192.168.100.101

autocommit = 1

skip\_name\_resolve = 1

max\_connections = 800

max\_connect\_errors = 100

transaction\_isolation = READ-COMMITTED

explicit\_defaults\_for\_timestamp = 1

join\_buffer\_size = 128M

tmp\_table\_size = 128M

tmpdir = /dev/shm

max\_allowed\_packet = 16M

interactive\_timeout = 60

wait\_timeout = 60

read\_buffer\_size = 16M

read\_rnd\_buffer\_size = 32M

sort\_buffer\_size = 32M

########log settings########

slow\_query\_log = 1

slow\_query\_log\_file = /usr/local/mysql/logs/mysql-slow.log

log\_queries\_not\_using\_indexes = 1

log\_slow\_admin\_statements = 1

log\_slow\_slave\_statements = 1

log\_throttle\_queries\_not\_using\_indexes = 10

expire\_logs\_days = 90

long\_query\_time = 1

min\_examined\_row\_limit = 100

########replication settings########

#master\_info\_repository = TABLE

#relay\_log\_info\_repository = TABLE

log\_bin = /usr/local/mysql/logs/mysql-bin

#sync\_binlog = 4

gtid\_mode = on

enforce\_gtid\_consistency = 1

#log\_slave\_updates

binlog\_format = row

#relay\_log = /usr/local/mysql/logs/mysql-relay.log

#relay\_log\_recovery = 1

#binlog\_gtid\_simple\_recovery = 1

#slave\_skip\_errors = ddl\_exist\_errors

########innodb settings########

innodb\_page\_size = 16K

innodb\_buffer\_pool\_size = 4G

#innodb\_buffer\_pool\_instances = 8

#innodb\_buffer\_pool\_load\_at\_startup = 1

#innodb\_buffer\_pool\_dump\_at\_shutdown = 1

#innodb\_lru\_scan\_depth = 2000

innodb\_lock\_wait\_timeout = 5

#innodb\_io\_capacity = 4000

#innodb\_io\_capacity\_max = 8000

#innodb\_flush\_method = O\_DIRECT

#innodb\_log\_group\_home\_dir = /usr/local/mysql/logs/redolog/

#innodb\_undo\_directory = /usr/local/mysql/logs/undolog/

#innodb\_undo\_logs = 128

#innodb\_undo\_tablespaces = 0

#innodb\_flush\_neighbors = 1

#innodb\_log\_file\_size = 4G

#innodb\_log\_buffer\_size = 16M

#innodb\_purge\_threads = 4

innodb\_large\_prefix = 1

innodb\_thread\_concurrency = 64

#innodb\_print\_all\_deadlocks = 1

#innodb\_strict\_mode = 1

innodb\_sort\_buffer\_size = 64M

########semi sync replication settings########

#plugin\_dir=/usr/local/mysql/lib/plugin

#plugin\_load = "rpl\_semi\_sync\_master=semisync\_master.so;rpl\_semi\_sync\_slave=semisync\_slave.so"

#loose\_rpl\_semi\_sync\_master\_enabled = 1

#loose\_rpl\_semi\_sync\_slave\_enabled = 1

#loose\_rpl\_semi\_sync\_master\_timeout = 5000

[mysqld-5.7]

#innodb\_buffer\_pool\_dump\_pct = 40

innodb\_page\_cleaners = 4

#innodb\_undo\_log\_truncate = 1

#innodb\_max\_undo\_log\_size = 2G

#innodb\_purge\_rseg\_truncate\_frequency = 128

#binlog\_gtid\_simple\_recovery=1

log\_timestamps=system

#transaction\_write\_set\_extraction=MURMUR32

#show\_compatibility\_56=on

注解：

[mysqld]

user = mysql

mysql以什么用户运行

port = 31306

mysql运行在哪个端口

datadir = /usr/loca/mysql/data/

mysql的数据目录

socket=/tmp/mysql.sock

mysql以socket方式运行的sock文件位置

symbolic-links=0

是否支持符号链接，即数据库或表可以存储在my.cnf中指定datadir之外的分区或目录，为0不开启

########basic settings########

server-id = 11

mysql的服务器分配id，在启用主从和集群的时候必须指定，每个节点必须不同

#bind\_address = 10.166.224.32

mysql监听的ip地址，如果是127.0.0.1，表示仅本机访问

autocommit = 1

数据修改是否自动提交，为0不自动提交

character\_set\_server=utf8mb4

服务器使用的字符集

skip\_name\_resolve = 1

禁用DNS主机名查找，启用以后用内网地址向mysqlslap请求响应快了一半

max\_connections = 800

mysql最大连接数

max\_connect\_errors = 1000

某台host连接错误次数等于max\_connect\_errors（默认10） ，主机'host\_name'再次尝试时被屏蔽。可有效反的防止dos攻击

transaction\_isolation = READ-COMMITTED

数据库事务隔离级别

1.READ-UNCOMMITTED(读取未提交内容)级别  
2. READ-COMMITTED（读取提交内容）  
3. REPEATABLE-READ(可重读)  
4.SERIERLIZED(可串行化)  
默认级别REPEATABLE-READ

explicit\_defaults\_for\_timestamp = 1

mysql中TIMESTAMP类型和其他的类型有点不一样(在没有设置explicit\_defaults\_for\_timestamp=1的情况下）

join\_buffer\_size = 128M

当我们的join是ALL,index,rang或者Index\_merge的时候使用的buffer。 实际上这种join被称为FULL JOIN

tmp\_table\_size = 128M

规定了内部内存临时表的最大值，每个线程都要分配。（实际起限制作用的是tmp\_table\_size和max\_heap\_table\_size的最小值。）如果内存临时表超出了限制，MySQL就会自动地把它转化为基于磁盘的MyISAM表，存储在指定的tmpdir目录下

tmpdir = /dev/shm/mysql-tmp/

保存临时文件的目录

max\_allowed\_packet = 16M

mysql最大接受的数据包大小

sql\_mode = "STRICT\_TRANS\_TABLES,NO\_ENGINE\_SUBSTITUTION,NO\_ZERO\_DATE,NO\_ZERO\_IN\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_AUTO\_CREATE\_USER"

sql\_mode 模式，定义了你MySQL应该支持的sql语法，对数据的校验等等，限制一些所谓的‘不合法’的操作

interactive\_timeout = 60

服务器关闭交互式连接前等待活动的秒数。交互式客户端定义为在mysql\_real\_connect()中使用CLIENT\_INTERACTIVE选项的客户端

wait\_timeout = 60

服务器关闭非交互连接之前等待活动的秒数，在线程启动时，根据全局wait\_timeout值或全局interactive\_timeout值初始化会话wait\_timeout值，取决于客户端类型(由mysql\_real\_connect()的连接选项CLIENT\_INTERACTIVE定义)

read\_buffer\_size = 16M

读入缓冲区的大小，将对表进行顺序扫描的请求将分配一个读入缓冲区，MySQL会为它分配一段内存缓冲区

read\_rnd\_buffer\_size = 32M

随机读缓冲区大小，当按任意顺序读取行时（列如按照排序顺序）将分配一个随机读取缓冲区，进行排序查询时，MySQL会首先扫描一遍该缓冲，以避免磁盘搜索，提高查询速度

sort\_buffer\_size = 32M

是一个connection级参数，在每个connection第一次需要使用这个buffer的时候，一次性分配设置的内存

########log settings########

#log\_error = /data/local/mysql-5.7.19/log/mysql-error.log

错误日志位置

slow\_query\_log = 1

是否开启慢查询日志收集

slow\_query\_log\_file = /data/local/mysql-5.7.19/log/mysql-slow.log

慢查询日志位置

log\_queries\_not\_using\_indexes = 1

是否记录未使用索引的语句

log\_slow\_admin\_statements = 1

慢查询也记录那些慢的optimize table，analyze table和alter table语句

log\_slow\_slave\_statements = 1

记录由Slave所产生的慢查询

log\_throttle\_queries\_not\_using\_indexes = 10

设定每分钟记录到日志的未使用索引的语句数目，超过这个数目后只记录语句数量和花费的总时间

expire\_logs\_days = 90

日志自动过期清理天数

long\_query\_time = 1

设置记录慢查询超时时间

min\_examined\_row\_limit = 100

查询检查返回少于该参数指定行的SQL不被记录到慢查询日志

########replication settings########

#master\_info\_repository = TABLE

从机保存主节点信息方式，设成file时 会生成master.info 和 relay-log.info2个文件，设成table，信息就会存在mysql.master\_slave\_info表中。不管是设置的哪种值，都不要移动或者编辑相关的文件和表

#relay\_log\_info\_repository = TABLE

用于保存slave读取relay log的位置信息，可选值为“FILE”、“TABLE”，以便crash重启后继续恢复

log\_bin = /data/local/mysql-5.7.19/log/mysql-bin

binlog的保存位置，不能指定确定的文件名如mysql-bin.log，只能指定位置和前缀，会生成以前缀为开头的一系列文件

#sync\_binlog = 4

这个参数是对于MySQL系统来说是至关重要的，他不仅影响到Binlog对MySQL所带来的性能损耗，而且还影响到MySQL中数据的完整性。对于“sync\_binlog”参数的各种设置的说明如下：

sync\_binlog=0，当事务提交之后，MySQL不做fsync之类的磁盘同步指令刷新binlog\_cache中的信息到磁盘，而让Filesystem自行决定什么时候来做同步，或者cache满了之后才同步到磁盘。

sync\_binlog=n，当每进行n次事务提交之后，MySQL将进行一次fsync之类的磁盘同步指令来将binlog\_cache中的数据强制写入磁盘。

在MySQL中系统默认的设置是sync\_binlog=0，也就是不做任何强制性的磁盘刷新指令，这时候的性能是最好的，但是风险也是最大的。因为一旦系统Crash，在binlog\_cache中的所有binlog信息都会被丢失。而当设置为“1”的时候，是最安全但是性能损耗最大的设置。因为当设置为1的时候，即使系统Crash，也最多丢失binlog\_cache中未完成的一个事务，对实际数据没有任何实质性影响。从以往经验和相关测试来看，对于高并发事务的系统来说，“sync\_binlog”设置为0和设置为1的系统写入性能差距可能高达5倍甚至更多。

gtid\_mode = on

启用gtid类型，否则就是普通的复制架构

enforce\_gtid\_consistency = 1

强制GTID的一致性

#log\_slave\_updates

slave更新是否记入日志，在做双主架构时异常重要，影响到双主架构是否能互相同步

binlog\_format = row

binlog日志格式，可选值“MIXED”、“ROW”、“STATEMENT”，在5.6版本之前默认为“STATEMENT”，5.6之后默认为“MIXED”；因为“STATEMENT”方式在处理一些“不确定”性的方法时会造成数据不一致问题，我们建议使用“MIXED”或者“ROW”

#relay\_log = /data/local/mysql-5.7.19/log/mysql-relay.log

从机保存同步中继日志的位置

#relay\_log\_recovery = 1

当slave从库宕机后，假如relay-log损坏了，导致一部分中继日志没有处理，则自动放弃所有未执行的relay-log，并且重新从master上获取日志，这样就保证了relay-log的完整性

#binlog\_gtid\_simple\_recovery = 1

这个参数控制了当mysql启动或重启时，mysql在搜寻GTIDs时是如何迭代使用binlog文件的。 这个选项设置为真，会提升mysql执行恢复的性能。因为这样mysql-server启动和binlog日志清理更快

#slave\_skip\_errors = ddl\_exist\_errors

跳过指定error no类型的错误，设成all 跳过所有错误

########innodb settings########

innodb\_page\_size = 16K

innodb每个数据页大小，这个参数在一开始初始化时就要加入my.cnf里，如果已经创建了表，再修改，启动MySQL会报错

innodb\_buffer\_pool\_size = 4G

缓存innodb表的索引，数据，插入数据时的缓冲，专用mysql服务器设置的大小： 操作系统内存的70%-80%最佳

#innodb\_buffer\_pool\_instances = 8

可以开启多个内存缓冲池，把需要缓冲的数据hash到不同的缓冲池中，这样可以并行的内存读写

#innodb\_buffer\_pool\_load\_at\_startup = 1

默认为关闭OFF。如果开启该参数，启动MySQL服务时，MySQL将本地热数据加载到InnoDB缓冲池中

#innodb\_buffer\_pool\_dump\_at\_shutdown = 1

默认为关闭OFF。如果开启该参数，停止MySQL服务时，InnoDB将InnoDB缓冲池中的热数据保存到本地硬盘

#innodb\_lru\_scan\_depth = 2000

根据 官方文档 描述，它会影响page cleaner线程每次刷脏页的数量， 这是一个每1秒 loop一次的线程

innodb\_lock\_wait\_timeout = 5

事务等待获取资源等待的最长时间，超过这个时间还未分配到资源则会返回应用失败；参数的时间单位是秒

#innodb\_io\_capacity = 4000  
#innodb\_io\_capacity\_max = 8000

这两个设置会影响InnoDB每秒在后台执行多少操作. 大多数写IO(除了写InnoDB日志)是后台操作的. 如果你深度了解硬件性能(如每秒可以执行多少次IO操作),则使用这些功能是很可取的,而不是让它闲着

#innodb\_flush\_method = O\_DIRECT

默认值为 fdatasync. 如果使用 硬件RAID磁盘控制器, 可能需要设置为 O\_DIRECT. 这在读取InnoDB缓冲池时可防止“双缓冲(double buffering)”效应,否则会在文件系统缓存与InnoDB缓存间形成2个副本(copy). 如果不使用硬件RAID控制器,或者使用SAN存储时, O\_DIRECT 可能会导致性能下降

#innodb\_log\_group\_home\_dir = /data/local/mysql-5.7.19/log/redolog/

innodb重做日志保存目录

#innodb\_undo\_directory = /data/local/mysql-5.7.19/log/undolog/

innodb回滚日志保存目录

#innodb\_undo\_logs = 128

undo回滚段的数量， 至少大于等于35，默认128

#innodb\_undo\_tablespaces = 0

用于设定创建的undo表空间的个数，在mysql\_install\_db时初始化后，就再也不能被改动了；默认值为0，表示不独立设置undo的tablespace，默认记录到ibdata中；否则，则在undo目录下创建这么多个undo文件，例如假定设置该值为4，那么就会创建命名为undo001~undo004的undo tablespace文件，每个文件的默认大小为10M。修改该值会导致Innodb无法完成初始化，数据库无法启动，但是另两个参数可以修改

#innodb\_flush\_neighbors = 1

InnoDB存储引擎在刷新一个脏页时，会检测该页所在区(extent)的所有页，如果是脏页，那么一起刷新。这样做的好处是通过AIO可以将多个IO写操作合并为一个IO操作。对于传统机械硬盘建议使用，而对于固态硬盘可以关闭。

#innodb\_log\_file\_size = 4G

这个值定义了日志文件的大小，innodb日志文件的作用是用来保存redo日志。一个事务对于数据或索引的修改往往对应到表空间中的随机的位置，因此当刷新这些修改到磁盘中就会引起随机的I/O，而随机的I/O往往比顺序的I/O更加昂贵的开销，因为随机的I/O需要更多的开销来定位到指定的位置。innodb使用日志来将随机的I/O转为顺序的I/O，只要日志文件是安全的，那么事务就是永久的，尽管这些改变还没有写到数据文件中，如果出现了当机或服务器断电的情况，那么innodb也可以通过日志文件来恢复以及提交的事务。但是日志文件是有一定的大小的，所以必须要把日志文件记录的改变写到数据文件中，innodb对于日志文件的操作是循环的，即当日志文件写满后，会将指针重新移动到文件开始的地方重新写，但是它不会覆盖那些还没有写到数据文件中的日志，因为这是唯一记录了事务持久化的记录

如果对 Innodb 数据表有大量的写入操作，那么选择合适的 innodb\_log\_file\_size 值对提升MySQL性能很重要。然而设置太大了，就会增加恢复的时间，因此在MySQL崩溃或者突然断电等情况会令MySQL服务器花很长时间来恢复

#innodb\_log\_buffer\_size = 16M

事务在内存中的缓冲。 分配原 则：控制在2-8M.这个值不用太多的。他里面的内存一般一秒钟写到磁盘一次

#innodb\_purge\_threads = 4

控制是否使用，使用几个独立purge线程（清除二进制日志）

innodb\_large\_prefix = 1

mysql在5.6之前一直都是单列索引限制767，起因是256×3-1。这个3是字符最大占用空间（utf8）。但是在5.6以后，开始支持4个字节的uutf8。255×4>767, 于是增加了这个参数。这个参数默认值是OFF。当改为ON时，允许列索引最大达到3072

innodb\_thread\_concurrency = 64

InnoDB kernel并发最大的线程数。 1) 最少设置为(num\_disks+num\_cpus)\*2。 2) 可以通过设置成1000来禁止这个限制

#innodb\_print\_all\_deadlocks = 1

是否将死锁相关信息保存到MySQL 错误日志中

#innodb\_strict\_mode = 1

开启InnoDB严格检查模式，尤其采用了页数据压缩功能后，最好是开启该功能。开启此功能后，当创建表（CREATE TABLE）、更改表（ALTER TABLE）和创建索引（CREATE INDEX）语句时，如果写法有错误，不会有警告信息，而是直接抛出错误，这样就可直接将问题扼杀在摇篮里

innodb\_sort\_buffer\_size = 64M

ORDER BY 或者GROUP BY 操作的buffer缓存大小

########semi sync replication settings########

#plugin\_dir=/data/local/mysql-5.7.19/lib/plugin

指定mysql的插件目录

#plugin\_load = "rpl\_semi\_sync\_master=semisync\_master.so;rpl\_semi\_sync\_slave=semisync\_slave.so"

指定载入哪些插件

#loose\_rpl\_semi\_sync\_master\_enabled = 1

控制主库上是否开启semisync

#loose\_rpl\_semi\_sync\_slave\_enabled = 1

控制备库是否开启semisync

#loose\_rpl\_semi\_sync\_master\_timeout = 5000

单位毫秒，防止半同步复制在没有收到确认的情况下，发送堵塞。master在超时之前没有收到确认，将恢复到异步复制，继续执行半同步没有进行的操作

[mysqld-5.7]

#innodb\_buffer\_pool\_dump\_pct = 40

表示转储每个bp instance LRU上最热的page的百分比。通过设置该参数可以减少转储的page数

innodb\_page\_cleaners = 4

为了提升扩展性和刷脏效率，在5.7.4版本里引入了多个page cleaner线程。从而达到并行刷脏的效果

在该版本中，Page cleaner并未和buffer pool绑定，其模型为一个协调线程 + 多个工作线程，协调线程本身也是工作线程。因此如果innodb\_page\_cleaners设置为8，那么就是一个协调线程，加7个工作线程

#innodb\_undo\_log\_truncate = 1

是否开启在线回收（收缩）undo log日志文件，支持动态设置

#innodb\_max\_undo\_log\_size = 2G

当超过这个阀值（默认是1G），会触发truncate回收（收缩）动作，truncate后空间缩小到10M

#innodb\_purge\_rseg\_truncate\_frequency = 128

控制回收（收缩）undo log的频率。undo log空间在它的回滚段没有得到释放之前不会收缩， 想要增加释放回滚区间的频率，就得降低设定值

#binlog\_gtid\_simple\_recovery=1

这个参数控制了当mysql启动或重启时，mysql在搜寻GTIDs时是如何迭代使用binlog文件的。 这个选项设置为真，会提升mysql执行恢复的性能。因为这样mysql-server启动和binlog日志清理更快。该参数为真时，mysql-server只需打开最老的和最新的这2个binlog文件

log\_timestamps=system

在MySQL 5.7.2 新增了 log\_timestamps 这个参数，该参数主要是控制 error log、genera log，等等记录日志的显示时间参数。 在 5.7.2 之后改参数为默认 UTC 这样会导致日志中记录的时间比中国这边的慢，导致查看日志不方便。修改为 SYSTEM 就能解决问题

#transaction\_write\_set\_extraction=MURMUR32

这个神奇的参数5.7.6版本引入，用于定义一个记录事务的算法，这个算法使用hash标识来记录事务。如果使用MGR，那么这个hash值需要用于分布式冲突检测何处理，在64位的系统，官网建议设置该参数使用 XXHASH64 算法。如果线上并没有使用该功能，应该设为off

#show\_compatibility\_56=on

从mysql5.7.6开始information\_schema.global\_status已经开始被舍弃，为了兼容性，此时需要打开 show\_compatibility\_56