目录

[MySQL优化 4](#_Toc47975183)

[1优化范围 4](#_Toc47975184)

[A.存储,主机和操作系统： 4](#_Toc47975185)

[B.应用:(Index,Lock,session) 4](#_Toc47975186)

[C.数据库优化：(内存,数据库设计,参数） 4](#_Toc47975187)

[2优化工具 5](#_Toc47975188)

[A.top命令 5](#_Toc47975189)

[B. iostat(安装systat),vmstat,dd,iotop,glances,htop 5](#_Toc47975190)

[C.数据库层工具 5](#_Toc47975191)

[3 优化过程(思路) 5](#_Toc47975192)

[A.主机,存储,网络 6](#_Toc47975193)

[B.系统 6](#_Toc47975194)

[C数据库实例 6](#_Toc47975195)

[4 锁 10](#_Toc47975196)

[5 主从优化 10](#_Toc47975197)

[DBA要求 11](#_Toc47975198)

[1需要掌握的内容 11](#_Toc47975199)

[2额外需要掌握 12](#_Toc47975200)

[MySQL(mysql-8.0.16-linux-glibc2.12-x86\_64.tar.xz)安装 12](#_Toc47975201)

[1.二进制软件安装在 /app/mysql 目录下 12](#_Toc47975202)

[2.配置环境 12](#_Toc47975203)

[3.创建数据路径并授权 12](#_Toc47975204)

[4.初始化数据 12](#_Toc47975205)

[5.配置文件 12](#_Toc47975206)

[6.启动数据库 13](#_Toc47975207)

[7.管理员密码设置 14](#_Toc47975208)

[8.管理员密码忘记 14](#_Toc47975209)

[MySQL体系结构和管理 14](#_Toc47975210)

[1MySQL C/S结构介绍 14](#_Toc47975211)

[2MySQL实例构成 14](#_Toc47975212)

[3MySQL中mysqld服务器进程结构 14](#_Toc47975213)

[3.1结构化的查询语句 14](#_Toc47975214)

[3.2 连接层 15](#_Toc47975215)

[3.3 SQL层 15](#_Toc47975216)

[3.4存储引擎层 15](#_Toc47975217)

[3.5 MySQL逻辑存储结构 15](#_Toc47975218)

[3.6 MySQL物理存储结构 15](#_Toc47975219)

[3.7 InnoDB段 区 页 16](#_Toc47975220)

[4用户和权限管理 16](#_Toc47975221)

[4.1用户定义 16](#_Toc47975222)

[4.2 用户操作 16](#_Toc47975223)

[4.3权限管理 16](#_Toc47975224)

[MySQL基础管理 16](#_Toc47975225)

[1配置文件的读取顺序 16](#_Toc47975226)

[2强制使用自定义配置文件 16](#_Toc47975227)

[3MySQL连接管理 16](#_Toc47975228)

[4多实例管理 17](#_Toc47975229)

[4.1准备多个目录 17](#_Toc47975230)

[4.2初始化三套数据 17](#_Toc47975231)

[4.3systemd管理多实列 18](#_Toc47975232)

[4.4授权 18](#_Toc47975233)

[4.5启动 18](#_Toc47975234)

[4.6验证多实例 18](#_Toc47975235)

[5 字符集：charset,排序规则:collation 18](#_Toc47975236)

[6 DDL的应用 18](#_Toc47975237)

[6.1DDL库定义 18](#_Toc47975238)

[6.2库定义规范 18](#_Toc47975239)

[6.3DDL表定义 19](#_Toc47975240)

[6.4建表规范 19](#_Toc47975241)

[6.5 DCL 19](#_Toc47975242)

[6.6 help 19](#_Toc47975243)

[6.7 DML 19](#_Toc47975244)

[6.8 DQL 20](#_Toc47975245)

[7元数据 20](#_Toc47975246)

[8索引 20](#_Toc47975247)

[9索引命令操作 21](#_Toc47975248)

[10.执行计划 21](#_Toc47975249)

[11 慢查询SQL 22](#_Toc47975250)

[12 锁类型 23](#_Toc47975251)

[13 存储引擎 25](#_Toc47975252)

[14 InnoDB存储引擎的物理结构 26](#_Toc47975253)

[15 Innodb核心特性 27](#_Toc47975254)

[16 Innodb核心参数 29](#_Toc47975255)

[MySQL日志管理 30](#_Toc47975256)

[1错误日志 30](#_Toc47975257)

[2二进制日志(binlog) 30](#_Toc47975258)

[3 慢日志(slow log) 33](#_Toc47975259)

[MySQL备份恢复 34](#_Toc47975260)

[1备份策略的设计 34](#_Toc47975261)

[2检查备份可用性 34](#_Toc47975262)

[3备份类型 35](#_Toc47975263)

[4逻辑备份mysqldump 35](#_Toc47975264)

[5物理备份XBK(PKB) 35](#_Toc47975265)

[MySQL主从复制 38](#_Toc47975266)

[1基于GTID主从搭建 39](#_Toc47975267)

[2传统主从搭建 42](#_Toc47975268)

[3主从复制原理 44](#_Toc47975269)

MySQL优化

1优化范围

A.存储,主机和操作系统：

主机架构的稳定性

I/O规划和配置

Swap情况(mysql主机建议关闭)

OS内核参数配置

网络

B.应用:(Index,Lock,session)

应用程序的稳定性和性能

SQL语句性能

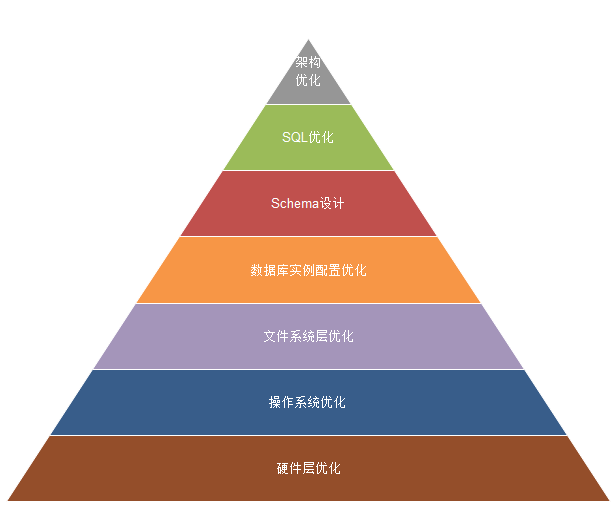
性能欠佳会话管理

C.数据库优化：(内存,数据库设计,参数）

内存

数据库结构(物理&逻辑)

实例配置



2优化工具

A.top命令

%Cpu(s): 17.3 us, 0.8 sy, 0.0 ni, 81.9 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st

id:空闲CPU时间片占比

wa:cpu用来等待的时间占比(等待IO:全表扫描,等待大处理事件,锁)

us:用户程序工作的时间片占比

sy:内核工作花费时间片占比(各种资源分配和调度,高并发连接,锁)

hi:硬件中断时间片占比

si:软件中断时间片占比

st:虚拟时间片占比[物理机可以虚拟化多台机，也会占用cpu]

KiB Mem: 8177036 total, 1281480 used, 6895556 free, 104096 buffers

KiB Swap: 8386556 total, 0 used, 8386556 free. 360988 cached Mem

Linux 6 操作系统,默认回收策略(buffer cache),不立即回收策略。内存使用

达100%-60%时，40%会使用swap;Linux 7 内存使用达到100%-70%时，才会使

swap;

B. iostat(安装systat),vmstat,dd,iotop,glances,htop

iostat -dk 3

dd if=/dev/zero of=/data/bigfiles bs=1m count=1024 (检查磁盘IO性能)

C.数据库层工具

show status

show variables

show index

show processlist

show slave status

show engine innodb status

desc / explain

slowlog

扩展深度优化

pt系列(pt-query-digest pt-osc pt-index)

mysqlslap

sysbench

information\_schema

performance\_schema

sys

3 优化过程(思路)

A.主机,存储,网络

主机：

硬件(PC Server)：Dell R 系列，华为,浪潮，HP，曙光，联想

云产品:ECS,数据库RDS，DRDS，polarDB

IBM 小型机 P6 570 595 P7 720 750 780 P8

CPU根据数据库类型：

OLTP

OLAP

IO密集型：线上系统，OLTP主要是IO密集型业务，高并发。

E系列(至强)，主频相对低,核心数量多.

CPU密集型: 数据分析处理，OLAP，cpu密集型的，需要CPU高计算能力.

(I系列,IBM Power系列),主频高,核心少。

内存：建议是2-3cpu核心数量(ECC)

硬盘：SATA-III SAS Fc SSD(sata) (pci-e)ssd Flash

主机RAID卡的BBU(Battery Backup Unit)关闭

存储：

根据存储数据的种类不同，选择不同的存储设备，配置合理的

RAID级别(raid5，raid10,热备盘)

R0：条带化，性能高

R1: 镜像，安全

R5：校验+条带化，安全和读性能高，写性能低(适合读多写少)

R10: 安全+性能都高,至少四块盘，浪费一半空间(高IO要求)

网络：

1 单卡单口,买好的

2 网卡绑定(binding),交换机堆叠.

B.系统

swap调整(mysql服务器建议关闭swap功能)

查看内存还剩百分多少时，开始使用swap

cat /proc/sys/vm/swappiness

echo 0 > /proc/sys/vm/swappiness (临时关闭)

在/etc/sysctl.conf

上添加vm.swappiness=0 (永久关闭)

sysc tl –p

IO调度策略(deadline)

Centos7 默认deadline

cat /sys/block/sda/queue/schduler

deadline 降低系统wa时间

raid, ssd

数据存储: no lvm 逻辑卷

系统文件格式:exf4 或 xfs

C数据库实例

1.max\_connections最大连接数 判断依据,监控max\_used\_connections数据变化

show VARIABLES LIKE '%max\_connections%'

show STATUS LIKE '%max\_used\_connections%'

2. back\_log 暂存的连接数 判断依据Show full processlist

发现大量待连接进程加大back\_log或 max\_connections

3.wait\_timeout 和interactive\_timeout

wait\_timeout: mysql关闭非交互式连接之前需要等待的秒数

interactive\_timeout:mysql关闭一个交互式连接之前需要等待的秒数

4.key\_buffer\_size

a.myisam表的索引缓冲区

b.临时表的缓冲区

临时表的创建(多表连接,子查询,union),在以上查询语句出现时，会创建临时表，用完后丢弃。临时表的2种创建方式

内存中🡪 key\_buffer\_size

磁盘中🡪ibdata1(5.6)

Ibtmp1(5.7)

通过 SHOW STATUS LIKE 'created\_tmp%'

Created\_tmp\_tables/( Created\_tmp\_tables+ Created\_tmp\_disk\_tables) 比例越高越好

mysqldump 会使用临时表，所以监控要规避监控该操作。

5.query\_cache\_size,max\_connect\_errors,innodb\_thread\_concurrency

query\_cache\_size：查询缓存数

max\_connect\_errors：连接错误数

innodb\_thread\_concurrency：并发用户数小于64,设置为0

工作负载较为严重 设置innodb\_thread\_concurrency=128,并不断调低

这个参数值,达到最佳线程数。也可以从小较大调整

通过show processlist查看数据库连接数情况

通过top命令查看当前cpu的使用情况，是否均匀。

6.会话(connection)级别参数,配置时,并不是越大越好.

sort\_buffer\_size：排序缓冲区

join\_buffer\_size：SQL连接缓冲区

read\_buffer\_size：顺序读入缓冲区

read\_rnd\_buffer\_size：随机读入缓冲区

order by,group by ,distinct,union 每个会话连执行的sql会使用到sort buffer size.

7. max\_allowed\_packet

server接受数据包大小

8.thread\_cache\_size

服务器线程缓存,通过比较connections和Threads\_created状态变量，可以看到

这个变量的作用。设置规则：1GB内存配置 4个 2GB内存配置8个

show status like 'threads\_%'

**9.innodb\_buffer\_pool\_size**

缓冲数据和索引 达到物理内存的70%-80%，建议70%

判断依据:show engine innodb status\G 查看数据页使用情况

**10.innodb\_flush\_log\_at\_trx\_commit**

控制innodb将log buffer的数据写入日志文件并flush磁盘时间点,取值:0/1/2

0:表示事务提交时，不做日志写入操作。而是每秒将log buffer里数据写入

日志文件并flush磁盘

1:每次事务提交时会引起redo日志写入,flush磁盘操作，确保事物ACID

2:每次事务提交时会引起redo日志写入动作,但每秒完成一次flush磁盘操作.

**11.innodb\_log\_buffer\_size, innodb\_log\_files\_in\_group**

redo日志文件大小,缓冲区更大能提高性能。

show status like '%commit%'

查看com\_commit大小 VS查看磁盘IO最大性能[使用dd命令]

**12. binary\_log**

为每个session分配内存，在事务过程中存储二进制日志缓存，

提高bin-log的效率。事务小不频繁设置1M，事务多且大,设置2-4M

binglog\_cache\_size=2M

表示binglog能够使用的最大缓存

max\_binglog\_cache\_size=8M

默认是1G，该变量不能大于1G或小于4k, 导入大容量SQL文件时,

建议关闭sql\_log\_bin

max\_binglog\_size=512M

定义mysql清除过期日志时间，二进制自动删除天数

expire\_logs\_days=7

每次事务提交时,刷新binlog到磁盘

**sync\_binlog=1**

**binlog\_format=row**

**13.innodb\_flush\_method=(O\_DIRECT,fsync)**

1.fsync：

a数据页持久化时,先写入OS buffer,然后OS决定什么时候写入磁盘

b 在 redo buffer需要持久化时, 先写入OS buffer,然后OS决定什么

时候写入磁盘,但 innodb\_flush\_log\_at\_trx\_commit=1时,日志在每commit时 直接写入磁盘

2.O\_DIRECT：

a 数据页持久化时.直接写入磁盘

b在 redo buffer需要持久化时, 先写入OS buffer,然后OS决定什么时

候写入磁盘但 innodb\_flush\_log\_at\_trx\_commit=1时,日志在每次

commit时直接写入磁盘

**最安全模式：**

**innodb\_flush\_log\_at\_trx\_commit=1**

**sync\_binlog=1**

**innodb\_flush\_method= O\_DIRECT**

**最高性能模式**

**innodb\_flush\_log\_at\_trx\_commit=0**

**innodb\_flush\_method= fsync**

**参数优化结果**

**[mysqld]**

**basedir=/data/mysql**

**datadir=/data/mysql/data**

**socket=/tmp/mysql.sock**

**log-error=/var/log/mysql\_error.log**

**log\_bin=/data/binlog/mysql-bin**

**binlog\_format=row**

**skip-name-resolve**

**server-id=10**

**gtid\_mode=on**

**enforce-gtid-consistency=true**

**log-slave-upadates=1**

**relay\_log\_purge=0**

**max\_connections=1024**

**back\_log=128**

**wait\_timeout=3600**

**interactive\_timeout=7200**

**key\_buffer\_size=32M**

**query\_cache\_size=64M**

**query\_cache\_type=1**

**query\_cache\_limit=50M**

**max\_connect\_errors=20**

**sort\_buffer\_size=2M**

**max\_allowed\_packet=32M**

**join\_buffer\_size=2M**

**thread\_cache\_size=128**

**innodb\_buffer\_pool\_size=1024M**

**innodb\_flush\_log\_at\_trx\_commit=1**

**innodb\_log\_buffer\_size=32M**

**innodb\_log\_file\_size=256M**

**innodb\_log\_files\_in\_group=3**

**binlog\_cache\_size=2M**

**max\_binlog\_cache\_size=8M**

**max\_binlog\_size=512M**

**expire\_logs\_days=7**

**read\_buffer\_size=2M**

**read\_rnd\_buffer\_size=2M**

**bluk\_insert\_buffer\_size=8M**

4 锁

1. 监控是否有锁等待

SHOW STATUS LIKE 'innodb\_row\_lock%';

1. 检查锁等待的事务

SELECT trx\_id,trx\_state,trx\_mysql\_thread\_id,trx\_query FROM

`information\_schema`.`INNODB\_TRX` WHERE trx\_state='LOCK WAIT';

trx\_mysql\_thread\_id 对应 SHOW processlist里的id

c 检查锁源

SELECT locked\_table,locked\_index,locked\_type,

waiting\_trx\_id ,waiting\_pid,waiting\_query,waiting\_lock\_mode,

sql\_kill\_blocking\_query,sql\_kill\_blocking\_connection,

blocking\_lock\_id,blocking\_pid FROM sys.`innodb\_lock\_waits`;

locked\_type:锁类型(record,gap,next lock)

waiting\_trx\_id:等待的事务id

waiting\_pid:等待事务的连接线程id

waiting\_query:等待事务语句

waiting\_lock\_mode:等待锁类型(X,S)

blocking\_lock\_id:锁源的事务id

blocking\_pid:锁源的线程号

d 根据锁源pid,找到锁源SQL的线程ID

SELECT thread\_id,NAME,processlist\_id FROM **`performance\_schema`.threads**

WHERE processlist\_id= sys.`innodb\_lock\_waits`. blocking\_pid

e 根据锁源SQL线程的ID,找到锁源SQL语句

SELECT thread\_id,event\_name,sql\_text FROM

performance\_schema.`events\_statements\_current`

WHERE thread\_id=**performance\_schema.threads. thread\_id**

f 检测/打印死锁情况

Show engine innodb status\G;(仅显示最后一条死锁情况)

SHOW VARIABLES LIKE '%dead%'

innodb\_print\_all\_deadlocks=ON

5 主从优化

5.7以上版本,必须开启git,binlog必须是row模式

gtid\_mode=on

enforce\_gtid\_consistency=on

log\_slave\_updates=on

slave\_parallel\_type=LOGICAL\_CLOCK

slave\_parallel\_workers=16

master\_info\_repository=table

relay\_log\_info\_repository=table

relay\_log\_recovery=on

gtid模式

CHANGE MASTER TO

MASTER\_HOST='master2.mycompany.com',

MASTER\_USER='replication',

MASTER\_PASSWORD='bigs3cret',

MASTER\_PORT=3306,

MASTER\_AUTO\_POSITION=1

传统模式

CHANGE MASTER TO

MASTER\_HOST='master2.mycompany.com',

MASTER\_USER='replication',

MASTER\_PASSWORD='bigs3cret',

MASTER\_PORT=3306,

MASTER\_LOG\_FILE='master2-bin.001',

MASTER\_LOG\_POS=4,

MASTER\_CONNECT\_RETRY=10;

DBA要求

1需要掌握的内容

MySQL5.7安装部署(二进制),编译扩展

MySQL升级步骤扩展

MySQL体系结构原理

MySQL基础管理

基础SQL语句使用

SQL高级应用

Information\_schema获取元数据

索引,执行计划管理(基础优化)

存储引擎

日志管理

备份和恢复

主从复制及架构演变

传统的高可用及读写分离(MHA&Atlas)

传统分布式架构设计与实现-扩展(MGR,InnoDB Cluster)

MySQL优化(安全,性能)

MySQL监控(zabbix,open-falcon)

RDS(阿里云)

2额外需要掌握

Redis,Mongodb,ES

MySQL(mysql-8.0.16-linux-glibc2.12-x86\_64.tar.xz)安装

1.二进制软件安装在 /app/mysql 目录下

a.删除linux原有的mysql,查询

rpm –qa|grep mariab

dpkg –l|grep mariab

b.若有老版本mysql,先删除。

2.配置环境

a.group add mysql

b.useradd add mysql -g mysql

c.chown mysql:mysql -R /app/mysql

d.usermod -s /sbin/nologin mysql

e.vim /etc/profile

添加export PATH=/app/mysql/bin:$PATH

source /etc/profile

f.检查mysql版本 mysql –V

3.创建数据路径并授权

a.添新磁盘模拟数据盘

如:/dev/sdb 挂载到 /data/mysql/data目录上

b.查看磁盘情况fdisk –l

c.格式化并挂载磁盘

mkfs.xfs /dev/sdb 格式化

mkdir /data

blkid 查询/dev/sdb的uuid信息

vim **/etc/fstab** 添加如下信息

/dev/disk/by-uuid/72d56e6f-cdc1-4359-994e-f8cc1323f695 /data xfs defaults 0 0

mount –a 或者 mount /dev/sdb /data

df –h 检查磁盘是否挂载成功

chown mysql:mysql -R /data

4.初始化数据

mysqld –-initialize --user=mysql --user=mysql --datadir=/data/mysql

会生成临时密码

A temporary password is generated for root@localhost: GMdkadzaT0!x

无限制，无临时密码

mysqld --initialize-insecure --user=mysql --basedir=/app/mysql

--datadir=/data/mysql

--**innodb\_data\_file\_path=ibdata1:512M:ibdata2:512M:autoextend**

**--innodb\_autoextend\_increment=64**

5.配置文件

root@kenel-server:/app/mysql# cat > /etc/my.cnf <<eof

> [mysqld]

> user=mysql

> basedir=/app/mysql

> datadir=/data/mysql

> socket=/tmp/mysql.sock

> server\_id=1

> port=3306

>mysqlx\_port=33060

>log\_error=/data/mysql/mysql.log

> [mysql]

> socket=/tmp/mysql.sock

> eof

6.启动数据库

cp /app/mysql/support-files/mysql.server /etc/init.d/mysqld

启动

/etc/init.d/mysqld start

service mysqld start

配置自启动(Ubuntu)

cat >/etc/systemd/system/mysqld.service << EOF

[Unit]

Description=MySQL Community Server

Documentation=man:mysqld(8)

Documentation=http://dev.mysql.com/doc/refman/en/using-systemd.html

After=network.target

[Install]

WantedBy=multi-user.target

[Service]

User=mysql

Group=mysql

Type=notify

PermissionsStartOnly=true

#ExecStartPre=/usr/share/mysql-8.0/mysql-systemd-start pre

ExecStart=/app/mysql/bin/mysqld --defaults-file=/etc/my.cnf

TimeoutSec=0

LimitNOFILE = 10000

Restart=on-failure

RestartPreventExitStatus=1

# Always restart when mysqld exits with exit code of 16. This special exit code

# is used by mysqld for RESTART SQL.

RestartForceExitStatus=16

# Set enviroment variable MYSQLD\_PARENT\_PID. This is required for restart.

Environment=MYSQLD\_PARENT\_PID=1

RuntimeDirectory=mysqld

RuntimeDirectoryMode=755

EOF

sudo systemctl enable mysqld

sudo systemctl status mysqld 检查

sudo systemctl start mysqld 启动服务

7.管理员密码设置

mysqladmin -uroot -p password kenel

8.管理员密码忘记

/etc/init.d/mysqld stop

mysqld\_safe --skip-grant-tables --skip-networking &

mysqladmin –uroot -p shutdown

登录并修改密码

**flush privileges;**(加载权限表)

alter user root@’localhost’ identified by ‘abc123’

**/etc/init.d/mysqld restart**

MySQL体系结构和管理

1MySQL C/S结构介绍

两种连接方式:

TCP/IP: mysql –uroot –p –h10.27.122.3 –P3306

Socket: mysql –uroot –p –S /tmp/mysql.sock

2MySQL实例构成

实例: mysqld+master thread+处理任务Thread+预分配内存

3MySQL中mysqld服务器进程结构

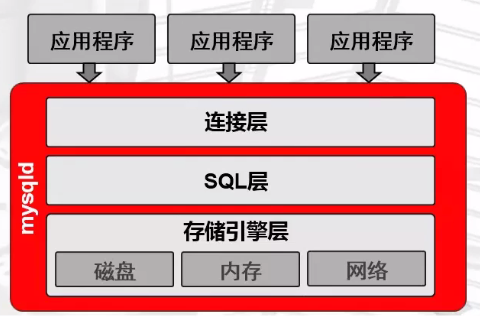
3.1结构化的查询语句

DQL数据查询

DDL 数据定义

DML数据操作

DCL 数据控制



3.2 连接层

a.提供连接协议(TCP/IP,socket)

b.用户名/密码/IP等合法性验证

c.开启专用连接线程(接受语句,返回结构)

show processlist;

d.将语句交给下一层

3.3 SQL层

a.接受语句

b.语法检查和SQL MODE

c.语义检查和权限检查

d.解析语句，生成多种执行计划树

e.通过优化器算法(执行代价cpu,io,mem)

f.优化器选择最优的执行方法

g.语句执行器,真正运行SQL

h.提供查询缓存(默认不开启)，一般不开,使用redis

i.记录操作日志(binlog)

3.4存储引擎层

根据SQL层提供取数据的方法,拿到磁盘上的数据，返回给SQL，结构化成表，再

通过连接层的连接线程返回给用户.

3.5 MySQL逻辑存储结构

库 create database wordpress charset UTF8;

show databases;

user wordpress;

表 列[字段] 列属性 数据行[记录] 表属性(元数据)

3.6 MySQL物理存储结构

库 使用FS上的目录来表示.

表 MyISAM：

User.frm 存储的表结构(列，列属性)

User.MYD 存储数据记录

User.MYI 存储索引

InnoDB：

Time\_zone.frm 存储的表结构(列，列属性)

Time\_zone.ibd 存储数据记录和索引

ibdata1(表属性存储在)数据字典表里

3.7 InnoDB段 区 页

在非分区情况下

一个表就是一个段

一个段由多个区构成

一个区在(16k),64个连续的页，总共1M大小构成.

4用户和权限管理

4.1用户定义

用户名@’白名单’

wordpress@’%’

[wordpress@’10.27.122.%](mailto:wordpress@'10.27.122.%25)’

4.2 用户操作

create user kenel@’10.27.89.%’ identified by ‘abc123’

alter user [kenel@’10.27.89.%](mailto:kenel@'10.27.89.%25)’ identified by ‘abc321’

drop user [kenel@’10.27.89.%](mailto:kenel@'10.27.89.%25)’

4.3权限管理

权限列表 ALL,SELECT,INSERT,UPDATE,DELETE,CREATE,DROP,RELOAD,SHUTDOWN,PROCESS,FILE,

REFERENCES,INDEX,ALTER,SHOW DATABASES,SUPER,CREATE TEMPORARY TABLE,LOCK TABLES,

EXECUTE,REPLICATION SLAVE,REPLICATION CLIENT,CREATE VIEW,SHOW VIEW,CREATE ROUTINE,

ALTER ROUTINE,CREATE USER,EVENT,TRIGGER,CREATE TABLESPACE

**with grant option  
 grant 权限 on 数据库.数据表 to 用户/角色 [ with grant option]**

MySQL基础管理

1配置文件的读取顺序

mysqld –help –verbose|grep my.cnf

/etc/my.cnf /etc/mysql/my.cnf /usr/local/mysql/etc/my.cnf ~/my.cnf

2强制使用自定义配置文件

--defaults-file

3MySQL连接管理

TCP/IP：mysql -uroot -p -h10.27.122.38 -P3306

Socket：mysql -uroot -p -S /tmp/mysql.sock

**mysql -uroot -p -S /tmp/mysql.sock -h10.27.122.38**

**优先以远程 [TCP/IP]登陆**

**Mysql客户端工具**

4多实例管理

4.1准备多个目录

mkdir -p /data/330{7,8,9}/data

cat>/data/3307/my.cnf<<EOF

[mysqld]

basedir=/app/mysql

datadir=/data/3307/data

socket=/data/3307/mysql.sock

log\_error=/data/3307/mysql.log

pid-file=/data/3307/mysql.pid

port=3307

mysqlx\_port=33070

service\_id=7

log\_bin=/data/3307/mysql-bin

EOF

cat>/data/3308/my.cnf<<EOF

[mysqld]

basedir=/app/mysql

datadir=/data/3308/data

socket=/data/3308/mysql.sock

log\_error=/data/3308/mysql.log

pid-file=/data/3308/mysql.pid

port=3308

mysqlx\_port=33080

service\_id=8

log\_bin=/data/3308/mysql-bin

EOF

cat>/data/3309/my.cnf<<EOF

[mysqld]

basedir=/app/mysql

datadir=/data/3309/data

socket=/data/3309/mysql.sock

log\_error=/data/3309/mysql.log

pid-file=/data/3307/mysql.pid

port=3309

mysqlx\_port=33090

service\_id=9

log\_bin=/data/3309/mysql-bin

EOF

4.2初始化三套数据

mv /etc/my.cnf /etc/my.cnf.bak

mysqld --initialize-insecure --user=mysql --data-dir=/data/3307/data --basedir=/app/mysql

mysqld --initialize-insecure --user=mysql --data-dir=/data/3308/data --basedir=/app/mysql

mysqld --initialize-insecure --user=mysql --data-dir=/data/3308/data --basedir=/app/mysql

4.3systemd管理多实列

cd /etc/systemd/system

cp mysqld.service mysqld3307.service

cp mysqld.service mysqld3308.service

cp mysqld.service mysqld3309.service

vim mysql3307.service

ExecStart=/app/mysql/bin/mysqld --defaults-file=/data/3307/my.cnf

vim mysql3308.service

ExecStart=/app/mysql/bin/mysqld --defaults-file=/data/3308/my.cnf

vim mysql3309.service

ExecStart=/app/mysql/bin/mysqld --defaults-file=/data/3309/my.cnf

4.4授权

chown -R mysql:mysql /data/\*

4.5启动

systemctl start mysqld3307.service

systemctl start mysqld3308.service

systemctl start mysqld3309.service

4.6验证多实例

netstat –lnp|grep 330

mysql -S /data/3307/mysql.sock -e “select @@server\_id”

mysql -S /data/3308/mysql.sock -e “select @@server\_id”

mysql -S /data/3309/mysql.sock -e “select @@server\_id”

5 字符集：charset,排序规则:collation

show charset

show collation

字符集\_general\_ci 英文大小写不敏感

字符集\_bin 大小写不敏感

6 DDL的应用

6.1DDL库定义

CREATE DATABASE wordpress CHARSET utf8mb4 COLLATE utf8mb4\_general\_ci;

Show create database wordpress;

DROP DATABASE wordpress;

Alter database wordpress CHARSET utf8mb4 COLLATE utf8mb4\_general\_ci;

6.2库定义规范

库名使用小写

库名不使用以数字开头

不使用数据库内部的关键字

设置字符集

6.3DDL表定义

表名,列名，列属性，表属性

列属性:

Primary Key:主键

NOT NULL:非空

UNIQUE Key:唯一

DEFAULT:默认值

UNSIGNED：无符号,配合数子列，非负数

COMMENT:备注

AUTO\_INCREMENT:自增长

CREATE TABLE stu(

id BIGINT NOT NULL AUTO\_INCREMENT COMMENT '学号',

sname VARCHAR(255) NOT NULL COMMENT '姓名',

age TINYINT UNSIGNED NOT NULL DEFAULT '20' COMMENT '年龄',

gender ENUM('m','f','n') NOT NULL DEFAULT 'n' COMMENT '性别',

inTime DATETIME NOT NULL DEFAULT NOW() COMMENT '入学时间',

PRIMARY KEY(id)

)ENGINE INNODB CHARSET utf8mb4;

DROP TABLE stu;

Create table stu1 like stu;

ALTER TABLE stu MODIFY age TINYINT NOT NULL DEFAULT '20' COMMENT '年龄';

ALTER TABLE stu CHANGE age age1 TINYINT NOT NULL DEFAULT '20' COMMENT '年龄';

ALTER TABLE stu add qq varchar(20) not null unique comment 'QQ号' [FIRST/AFTER age];

ALTER TABLE stu drop qq;

Desc stu

6.4建表规范

表名小写字母,不能数字开头

不能保留字符,使用和业务有关的表名

选择合适数据类型和长度

每个列设置NOT NULL+DEFAULT

每个列设置注释

表必须设置存储引擎和字符集

主键列尽量使用无关列，自增长

enum 保存不用数字，只能是字符串类型.

6.5 DCL

**grant 权限 on 数据库.数据表 to 用户/角色 [ with grant option]**

**revoke 权限 on 数据库.数据表 from 用户/角色**

**revoke 权限, grant option [on 数据库.数据表 ] from 用户/角色**

6.6 help

**Help show** ;Help create database;help create table

6.7 DML

INSERT,UPDATE,DELETE

6.8 DQL

**SHOW** VARIABLES LIKE 'innodb\_flush%'

**Select** database()

常用的聚合函数 SUM(),avg(),MAX(),MIN(),COUNT(),**GROUP\_CONCAT()**

7元数据

information\_schema 保存了大量元数据查询的视图

SELECT TABLE\_SCHEMA,TABLE\_NAME,TABLE\_ROWS,AVG\_ROW\_LENGTH,INDEX\_LENGTH FROM TABLES

WHERE TABLE\_SCHEMA='OAM\_DEV'

show 部分元数据的查询

show databases;show tables; show create database xx; show create table xx;

show processlist;show charset;show collation;show grants for 用户;

show variables like ‘%%’;show engines;show index from 表;

show engine innodb status\G;show binary logs;show binlog events in ‘xx’;

show master status;show slave status;show relaylog events in ‘xx’

show status;

**help show;**

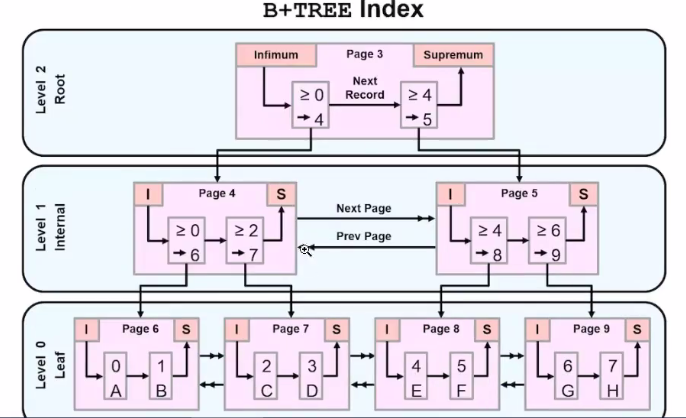
8索引

**B树（B-树）,B+树，B\*树**

**B+树 叶子结点 有双向指针,比B树优化了范围查找。**

**B\*树 枝接点 有双向指针,比B+树优化了枝节点。**

**辅助索引，聚簇索引**

****

辅助索引与聚簇索引区别：

1. 表中任何一个列都可以创建辅助索引,只要名字不同即可。
2. 在一张表中，聚集索引只能有一个，一般为主键。
3. 辅助索引，叶子节点只存储索引列的有序值+聚集索引列值
4. 聚集索引，叶子节点存储有序的整行数据
5. MySQL的表数据存储是聚集索引组织表

辅助索引:

1. 单列辅助索引
2. 联合索引（覆盖索引）
3. 唯一索引

索引高度

索引树的高度，应当越低越好，一般维持在3-4最佳

1. 数据行较多时

分区 parttion,用的较少

分片 分布式架构

1. 字段长度

业务允许，尽量选择字符长度短的列作为索引列。业务不允许，可采用前缀前缀.

9索引命令操作

desc 表;

PRI:主键，聚集索引

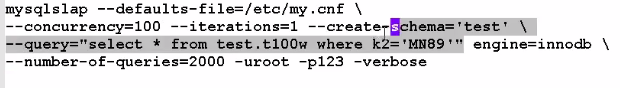
MUL:辅助索引

UNI:唯一索引

show index from 表\G;

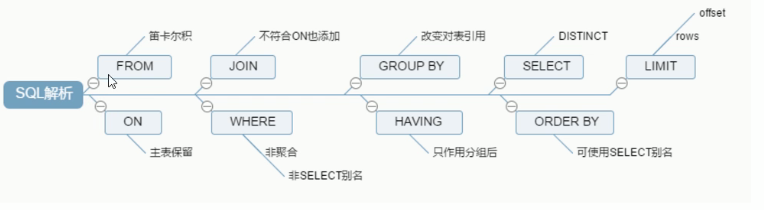
alter table 表 add index idx\_name(name)

压测



10.执行计划

**explain SQL语句,desc SQL语句**



执行计划包含的字段:

C:\Users\KENELL~1.000\AppData\Local\Temp\1594200124(1).png

**id**: 询序列号,包含一组数字,表示执行SQL的顺序.

id相同,执行顺序由上而下。

id不同,id值越大优先级越高，越先被执行。

**select\_type:** Simple 简单查询,不包含子查询或union

Primary 查询中包含了任何复杂的子部分，最外层查询则被标记。

Subquery 在select或where中包含子查询

Derived 在FROM列表中包含子查询被标记为derived(衍生)

Mysql会递归执行这些子查询，把结果放在临时表里。

Union 若第二个SELECT 出现在UNION之后，则被标记为Union;

若union 包含在FROM子句的子查询中，外层的SELECT将被标记derived

Union Result 从union表获取结果的select

**table:**表名, <union*M,N*>,<derived*N*>,<subquery*N*>

**type:system>const>eq\_ref>ref>fulltext>ref\_or\_null**>index\_merge>

unique\_subquery>index\_subquery>**range>index>ALL**

**possible\_keys:**显示可能应用在这张表上的索引，一个或多个。查询的字段若存在索引，则索引被列出

但不一定被查询实际用到。

**key**:实际使用的索引.如果为null，则没有使用索引。查询中若使用了覆盖索引，则该索引仅出现在key

列表中。

**key\_len:**表示索引中使用的字节数，可通过该列计算查询中使用的索引长度。Key\_len显示的值为索引

字段的最大可能长度，并非实际使用长度，即Key\_len 是根据表定义计算而得，不是通过表内检索出的

**ref:**显示索引的哪一列被使用了，如果可能的话，是一个常数.那些列或常量被用于查找索引列上的值。

显示将哪些列或常量与键列中命名的索引进行比较，以便从表中选择行。

**rows:**根据表的统计信息及索引情况，大致估算找到所需记录读取的行数

**extras:**十分重要额外信息,排序字段也要索引。如下说明

**using filesort** 说明数据使用外部排序，而不是按照表内索引顺序进行读取，

mysql无法利用索引完成的排序操作称为 “文件排序”

**using temporary** 使用了临时表保存中间结果 ,常见于排序order by 和分组查询 group by

**using index** 表示相应的select操作中使用了覆盖索引，避免访问表的数据行。如果同时出现

using where,表明索引被用来执行索引键值的查找。如果没有同时出现using where

表明索引来读取数据而非执行查找动作。

using where 表明使用where过滤

using join buffer 表明使用连接缓存

impossible where ：where子句的值总是false,不能用来获取任何元组

select tables optimized away 在没有group by 子句的情况下，基于索引优化MIN/MAX操作

distinct 优化distinct操作,在找到第一匹配的元组后即停止找同样值得操作

11 慢查询SQL

slow\_query\_log=1

slow\_query\_log\_file=/var/logs/mysql\_slow.log

long\_query\_time=3

a.分析工具mysqldumpslow

mysqldumpslow --help

获取到慢sql

b.开启profile

SHOW VARIABLES LIKE '%profiling%'

set profiling=ON

c.运行慢SQL

d.查看结果 show profiles

e. **SHOW PROFILE** [type [, type] ... ]

[FOR QUERY **n**]

[LIMIT row\_count [OFFSET offset]]

type: {

ALL

| BLOCK IO| CONTEXT SWITCHES

| CPU | IPC | MEMORY

| PAGE FAULTS | SOURCE | SWAPS

}

或者

SELECT \*

FROM INFORMATION\_SCHEMA.PROFILING

WHERE QUERY\_ID =**n**

建立索引原则:

1. 建表必须有主键
2. 经常做为where 条件列,order by,group by,join on,distinct的条件
3. 尽量使用唯一值多的列作为联合索引的前导列，其他按照联合索引优化细节处理
4. 列值长度较长的索引列，建议使用前缀索引
5. 降低索引条目，不创建没用的索引,不常使用索引需要清理，percona tookit

Pt-duplicate-key-checker

1. 索引维护避开业务繁忙值
2. 小表不建索引

不走索引的情况

1. 没有查询条件,或者查询条件没有建立索引
2. 查询结果集是原表中的大部分数据,应该是25%以上
3. 索引本身失效,统计数据不真实
4. 查询条件使用函数在索引上,或者对索引列进行运算(+,-,\*,/,!等)
5. 隐式转换导致索引失效
6. <>,not in 不走索引
7. like ‘%aa’不走索引
8. 联合索引

12 锁类型

那些表上有锁:SHOW OPEN TABLES FROM oceanv5;

查看表锁状态:SHOW STATUS LIKE 'table%'

查看行锁：SHOW STATUS LIKE 'innodb\_row\_lock%'

表锁：

session1

**lock table t1 read**

select \* from t1;//ok

update t1 set name=’zz’ where id=1;

//error, Table 't1' was locked with a READ lock and can't be updated

select \* from t2;

//error, Table 't2' was not locked with LOCK TABLES

session2

select \* from t1;//ok

select \* from t2;//ok

update t1 set name=’zz’ where id=1;// 处于被阻塞状态.

只有session1 unlock后，才执行

session1

**lock table t1 write**

select \* from t1;//ok

update t1 set name=’zz’ where id=1;//ok

select \* from t2;

// error, Table 't2' was not locked with LOCK TABLES

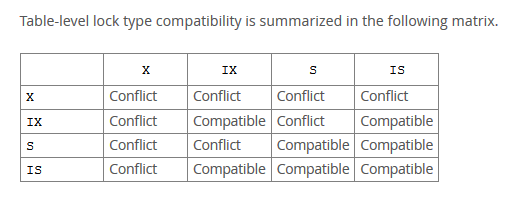
session2

select \* from t2;//ok

select \* from t1;// 处于被阻塞状态.只有session1 unlock后，才执行

update t1 set name=’zz’ where id=1;// 处于被阻塞状态

只有session1 unlock后，才执行



**Shared and Exclusive Locks**

InnoDB implements standard row-level locking where there are two types of locks, shared (S) locks and exclusive (X) locks.

**Intention Locks**

InnoDB supports multiple granularity locking which permits coexistence of row locks and table locks. For example, a statement such as *LOCK TABLES ... WRITE takes an exclusive lock* (an X lock) on the specified table. To make locking at multiple granularity levels practical, InnoDB uses intention locks. Intention locks are table-level locks that indicate which type of lock (shared or exclusive) a transaction requires later for a row in a table. There are two types of intention locks:

An intention shared lock (IS) indicates that a transaction intends to set a shared lock on individual rows in a table.

An intention exclusive lock (IX) indicates that a transaction intends to set an exclusive lock on individual rows in a table.

For example, SELECT ... FOR SHARE sets an IS lock, and SELECT ... FOR UPDATE sets an IX lock.

The intention locking protocol is as follows:

Before a transaction can acquire a shared lock on a row in a table, it must first acquire an IS lock or stronger on the table.

Before a transaction can acquire an exclusive lock on a row in a table, it must first acquire an IX lock on the table.

**Record Locks(**行锁**)**

在创建表t1(id,name) 并分别在id,name创建2个单列索引.

索引失效，导致行锁升级为表锁。

A record lock is a lock on an index record.

For example, SELECT c1 FROM t WHERE c1 = 10 FOR UPDATE; prevents any other transaction from inserting, updating, or deleting rows where the value of t.c1 is 10

Record locks always lock index records, even if a table is defined with no indexes.

SHOW ENGINE INNODB STATUS

**gap locks** (间隙锁) 索引锁

范围条件，请求共享或排他锁时，innodb会给符合条件的已有数据记录索引项加锁；对键

值范围内但不存在的记录，叫做”间隙锁”，

A gap lock is a lock on a gap between index records, or a lock on the gap before the first or after the last index record. For example, SELECT c1 FROM t WHERE c1 BETWEEN 10 and 20 FOR UPDATE;

**Next-Key Locks**索引锁

A next-key lock is a combination of a record lock on the index record and a gap lock on the gap before

the index record. By default, InnoDB operates in REPEATABLE READ transaction isolation level. In this

case, InnoDB uses next-key locks for searches and index scans, which prevents phantom rows

**Insert Intention Locks**

An insert intention lock is a type of gap lock set by INSERT operations prior to row insertion.

**AUTO-INC Locks**

An AUTO-INC lock is a special table-level lock taken by transactions inserting into tables with AUTO\_INCREMENT columns.

13 存储引擎

Show engines;支持的引擎

show variables like '%default\_storage%';当前默认使用的引擎

Tokudb（压缩比高，insert快）,percona xtradb,innodb,marriadb

Alter table t1 engine=innodb;//修改引擎，整理碎片.

整理碎片:导出，重建表再导入。Truncat删除，会回收段,页。

14 InnoDB存储引擎的物理结构

ibdata1:系统数据字典信息(统计信息),元数据信息,

5.6,5.7,8版本包含了不同信息。

5.6(包含了ibtmp1,undo)

5.7(包含了undo)

ibtmp1:临时表空间

undo:undo日志

ib\_logfile0 ib\_logfile1:Redo日志,事务日志

frm:存储表的列信息

ibd:数据行和索引

表空间(Tablespace)

ibdata1:数据库统计信息+undo

ibd:数据行+索引

共享表空间

5.5版本默认模式，需要将所有数据存储到同一表空间，管理混乱

5.6版本以后,共享表空间保留，用来存储 数据字典，undo，临时表

5.7版本,临时表独立出来

8.0版本,undo独立出来

共享表空间信息：

show variables like 'innodb\_data\_file\_path'

ibdata1:12M:autoextend，自动扩展,每次扩展64M

初始化时

**innodb\_data\_file\_path=ibdata1:512M:ibdata2:512M:autoextend**

**innodb\_autoextend\_increment=64**

独立表空间

主要存储用户数据,一个表一个ibd文件,存储数据行和索引信息

基本表结构元数据存储在frm

innodb表= (ibdata+frm)+ibd(段,区,页)

设置参数,建议都用独立表空间

**show variables like 'innodb\_file\_per\_table'**

独立表空间迁移

mysql如何使用物理数据文件进行迁移

1. 创建和原表一致的空表
2. 使用alter table t1 discard tablespace将空表的ibd文件删除,
3. 将原表的ibd 拷贝过来，并修改权限
4. 使用alter table t1 import tablespace 将原表的ibd数据导入

**alter table t1 discard tablespace**

**alter table t1 import tablespace**

MySQL引擎日志

Redo log: ib\_logfile0 ib\_logfile1,重做日志

Undo log: ibdata1 ibdata2,回滚日志 存储在共享表空间(5.7版本) \_

ibtmp1:临时表,在做join union 操作产生的临时数据，用完自动清理。

15 Innodb核心特性

1. A(原子性)C(一致性)I(隔离性)D(持久性)
2. 事务的生命周期性(事务控制语句)

开启事务: begin 或者 start Transaction

标准的事务语句:DML

事务结束:rollback或者commit

1. set autocommit=0/1 ;set global autocommit=0/1;
2. 触发隐式提交的语句

begin

语句1

语句2

begin

set autocommit=1

导致提交的非事务语句:

DDL:alter,create,drop

DCL:grant,revoke,set password

锁定语句: lock tables,unlock tables

导致隐式提交语句:

truncate table,load data infile

select for update

e 重要名词

redo log:重做日志,ib\_logfile0-1 默认50M,轮询使用

redo log buffer: redo内存区域

ibd:存储数据行和索引

data buffer pool:缓冲区池，数据和索引的缓冲

LSN:日志序列号

ibd(磁盘数据页),redo log,data buffer pool,redo buffer都会有LSN号

MySQL每次数据库启动，都会比较磁盘数据页和redo log的LSN，必须要求两者LSN

一致数据库才能启动

WAL:write ahead log 日志优先写的方式持久化

日志是优先于数据写入磁盘

脏页:内存脏页，内存中发生了修改,没写入到磁盘之前，我们把内存页称之为脏页

CKPT:check point 检查点，就是将脏页刷新到磁盘的动作

TXID:事务号,innodb为每个事务生成一个事务号，伴随整个事务

CSR:故障自动恢复机制

**事务日志—redo重做日志**

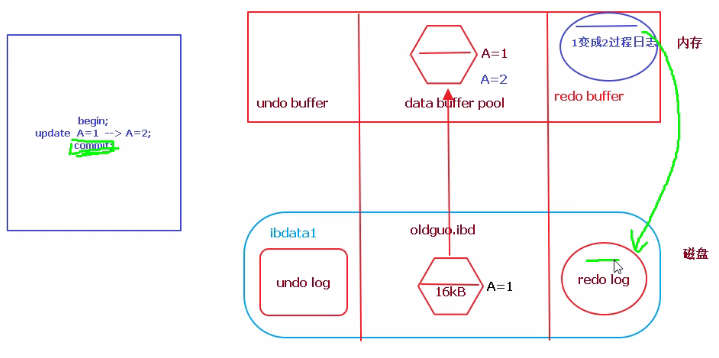
**作用:在ACID特性中，主要保证D特性，同时对CI也有一定功效** 1记录了内存页的数据的变化

2 提供了快速持久化功能

3CSR过程中实现前滚操作(ibd磁盘数据页和redo日志 LSN一致)

redo的buffer:数据页的变化信息+数据页的lsn号

ibd磁盘数据读入内存data buffer，至少是一页16kb，



Redo的刷写策略

1 commit

2 刷新当前事务的redobuffer到磁盘，还会将一部分redo buffer未提交的事

务日志也刷新到磁盘

MySQL:启动时，必须保证redo日志文件和数据文件LSN必须一致，如果不一致就会触发

CSR最终保证一致

**回滚日志--Undo日志**

**作用:在ACID特性中，主要保证A特性，同时对CI也有一定功效**

1记录了数据修改之前的状态信息

2 rollback将内存的数据修改恢复到修改之前

3 在CSR中实现未提交数据的回滚操作

4 实现一致性快照，配合隔离级别保证MVCC,读写操作不会互相阻塞

锁

实现了事务之间的隔离,innodb实现了行级锁

innodb隔离级别

**show variables like '%transaction\_isolation%';**

RU :读未提交,脏读（read-uncommitted）

RC :读已提交,会出现不可重复读。(read-committed)

RR :可重复读,有可能出现”幻读”,利用undo快照技术+GAP锁+Next Lock+

必须有索引，来避免幻读数据(REPEATABLE-READ)

SR :可串行化,防止死锁,并发事务差

补充：在RC级别下，可减轻gap+next lock锁问题，但是会出现幻读，一般

为了读一致性，在正常select 添加for update,执行完一定commit;否则 容易出现

锁等待.

Mvcc🡪undo 快照

16 Innodb核心参数

default\_storage\_engine=Innodb

#共享表空间个数和大小

innodb\_data\_file\_path=ibdata1:512M:ibdata2:51M:autoextend

#表空间模式

innodb\_file\_per\_table=1

#双一标准的其中一个.事务提交,刷写redo buffer到磁盘

innodb\_flush\_log\_at\_trx\_commmit=1;

0：每秒redo buffer->os buffer,每秒os buffer->磁盘

1：每个事务提交redo buffer->os buffer, os buffer->磁盘

2：每个事务提交redo buffer->os buffer, 每秒os buffer->磁盘

#写入方式

Innodb\_flush\_method=(O\_DIRECT,fsync)

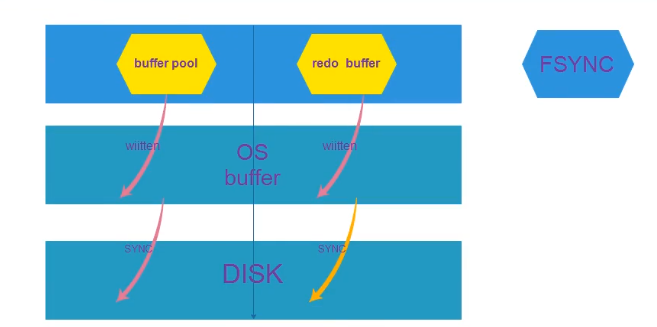
作用：控制redo buffer 和 buffer pool

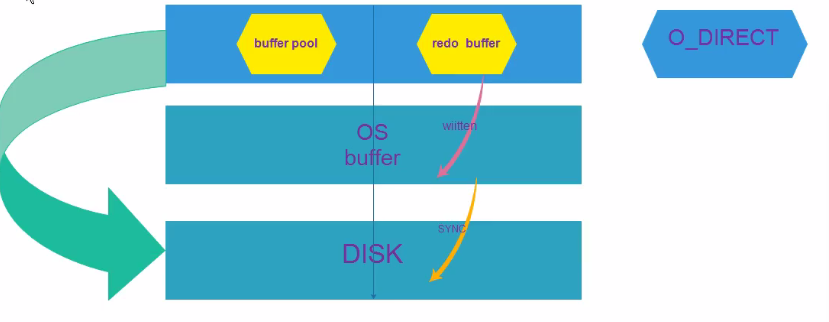
fsync:[redo buffer, buffer pool]先写入os buffer,os buffer再写入磁盘

**O\_DIRECT**: buffer pool直接写入磁盘，

redo buffer先写入os buffer,os buffer再写入磁盘

**与innodb\_flush\_log\_at\_trx\_commmit=1配合使用**





#redo日志设置

Innodb\_log\_buffer\_size=16M

Innodb\_log\_file\_size=128M

Innodb\_log\_files\_in\_group=3

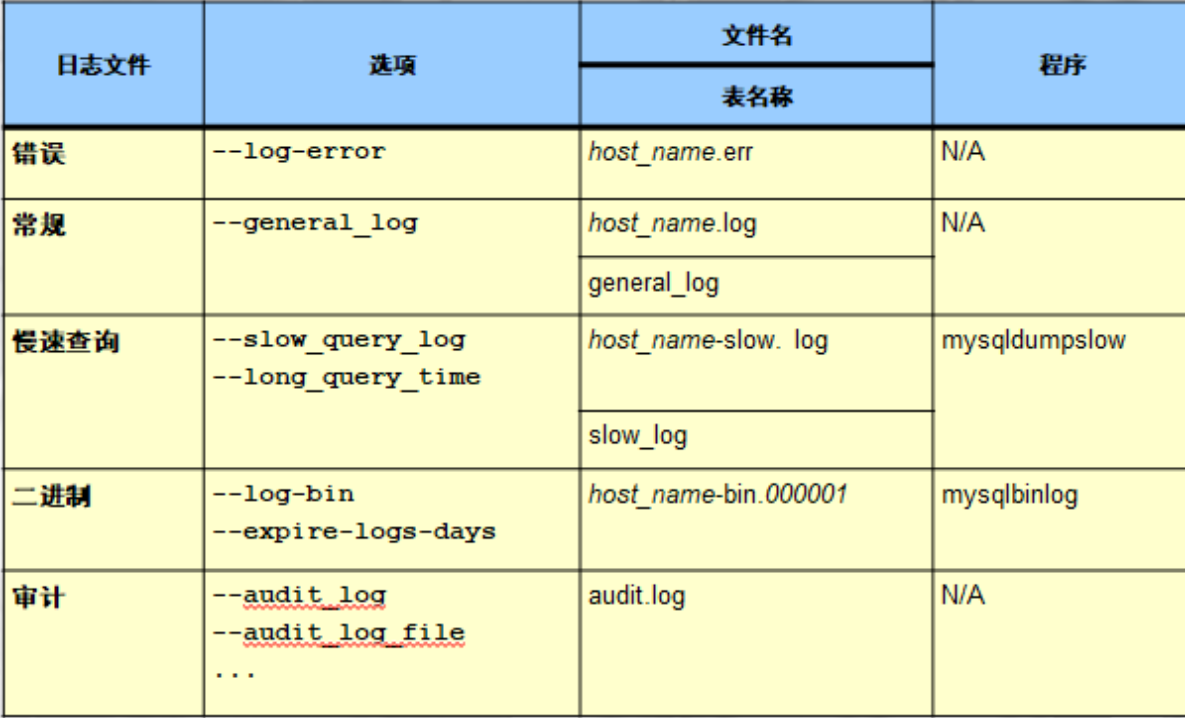
#data buffer脏页刷写策略

Innodb\_max\_dirty\_pages\_pct=75

那些机制会触发写磁盘

CSR，redo满了

MySQL日志管理



1错误日志

作用:排查MySQL运行过程的故障

配置:默认开启.默认路径:datadir/hostname.err

自定义: log-error=/var/log/mysqld.log

2二进制日志(binlog)

作用:a主从模式，b数据恢复

配置:默认未开启.

自定义:

server\_id=5

log\_bin=/data/mysql/binlog/mysql-bin

binlog\_format=row #5.7.7以后版本默认的格式，可省略

/data/mysql/binlog 提前定制好目录，而且要有mysql.mysql的权限

mysql-bin 二进制日志文件名的前缀

记录了数据库所有变更类的操作日志，即DDL，DCL，DML

DDL DCL原样记录,

DML记录格式(statement,row,mixed),通过binlog\_format可以控制

statement:SBR 语句模式记录日志，可读性强,日志量少，但可能出现不准确的情况

row:RBR 行模式,数据行的变化，可读性差，日志量大,不会出现错误，高可用依赖RBR

mixed:MBR 混合模式

二进制日志记录单元

1 event事件,二进制日志的最小单元

DDL：create database study charset utf8每一条语句就是一个事件

DML:一个事务包含了多个语句

begin ; 事件1

语句1：事件2

语句2：事件3

commit; 事件4

2 event事件的开始和结束号码

作用：方便从日志中截取我们想要的日志事件

3 二进日志的管理

查看二进制日志位置 show variables like '%log\_bin%'

查看所有已存在的二进制日志:show binary logs;

手工滚动生成二进制日志 flush logs;

查看正在使用的二进制日志 show master status;

查看二进制日志事件 show binlog events in 'mysql-bin.000002' [limit 5,1];

查看二进制日志内容 mysqlbinlog,注意权限

sudo /app/mysql/bin/mysqlbinlog mysql-bin.000002

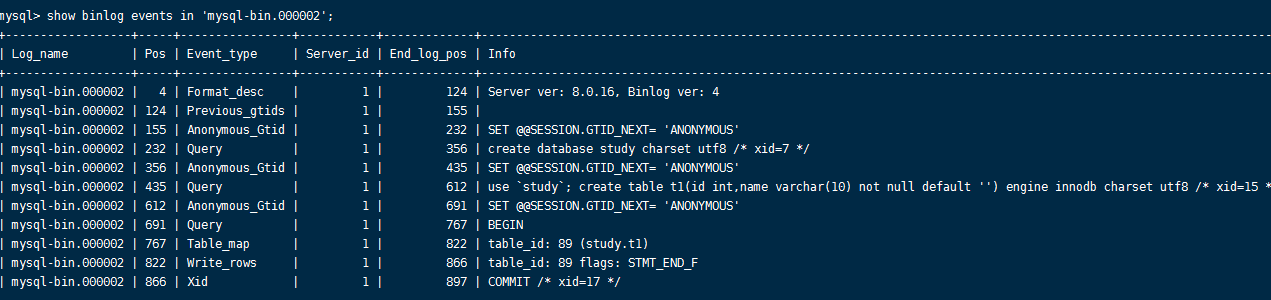
sudo /app/mysql/bin/mysqlbinlog mysql-bin.000002 |grep -v '^SET'

sudo /app/mysql/bin/mysqlbinlog --base64-output=decode-rows -vvv

mysql-bin.000002

sudo /app/mysql/bin/mysqlbinlog -d 数据库名 mysql-bin.000002 #过滤一个库的操作

截取二进制日志



sudo /app/mysql/bin/mysqlbinlog --start-position= 232 --stop-position=356

mysql-bin.000002 >tmp.sql

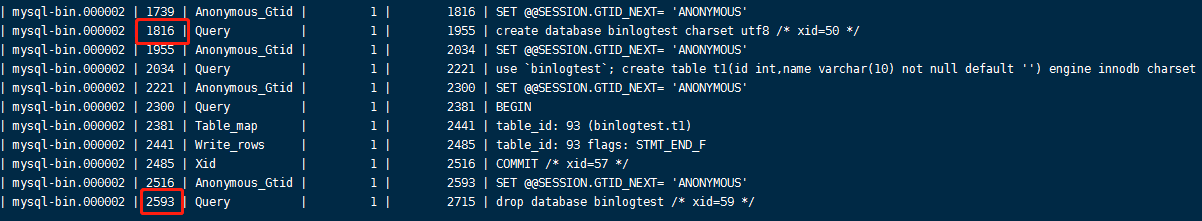
通过binlog恢复数据

模拟创建库，表及插入数据，然后删除库.分析起始和结束点

然后通过binlog恢复

show master status;//确认当前使用binlog

show binlog events in ‘mysql-bin.xxxxx’;//查看事件



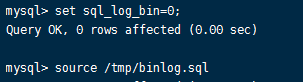
sudo /app/mysql/bin/mysqlbinlog --start-position=1816 --stop-position=2539

mysql-bin.000002 >/bin/tmp.sql

注意：恢复会产生binglog，而恢复的binlog不需要记录，因此在会话里关闭。

session里 set sql\_log\_bin=0;

source tmp/bin.sql;



set sql\_log\_bin=1;

binlog的gtid记录模式的管理

对于binlog中的每一个事务,都会生成一个GTID号码

DDL,DCL 一个event就是一个事务，就会有一个GTID号

DML语句，begin到commit,是一个事务，就是一个GTID号

GTID组成=server\_uuid:TID

server\_uuid 是 datadir目录下的auto.cnf里存在的，初始化时生成的

TID:自增长数据,从1开始.

GTID幂等性

用GTID日志去恢复时，检查当前系统中是否有相同的GTID号，有相同的就自动跳过，

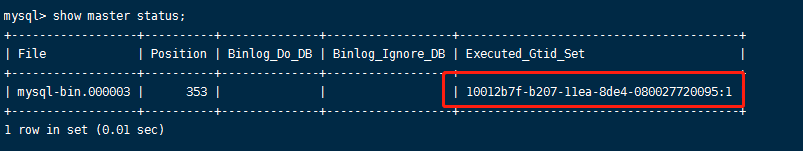
会影响到binlog恢复和主从复制

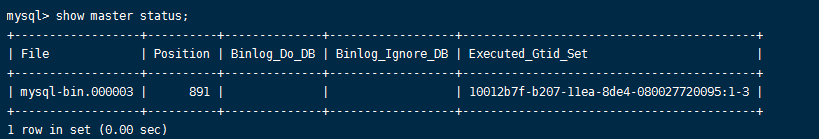
配置开启gtid [my.cnf]

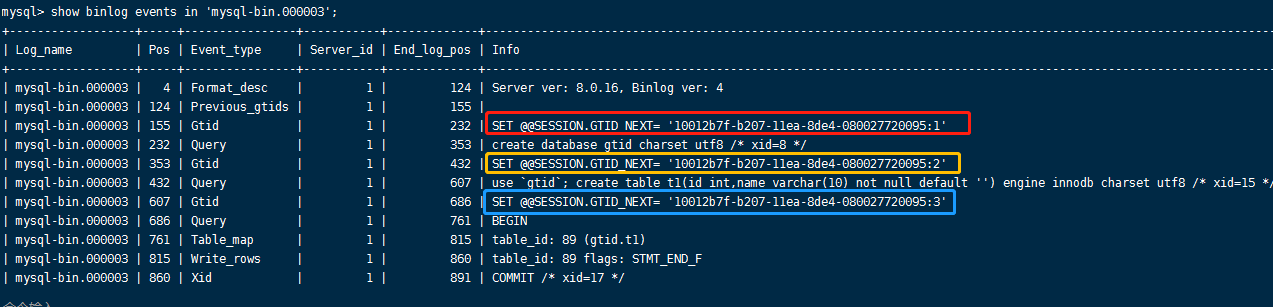
gtid-mode=on

enforce-gtid-consistency=true

查看gtid信息







基于gtid binlog恢复

模拟创建库，表及插入数据，然后删除库. 然后通过binlog gtid恢复

sudo /app/mysql/bin/mysqlbinlog --skip-gtids

--include-gtids='10012b7f-b207-11ea-8de4-080027720095:1-3' mysql-bin.000003 > /tmp/gtid.sql **--skip-gtids 导出时，忽略原有的gtid信息，恢复时生成新的gtid信息,否则由于幂等性，恢复会报错。**

恢复

set sql\_log\_bin=0;

source /tmp/gtid.sql

set sql\_log\_bin=1;

gtid相关参数

--skip-gtids

--include-gtids= '10012b7f-b207-11ea-8de4-080027720095:1-3',

'10012b7f-b207-11ea-8de4-080027720095:5'

--exclude-gtids='10012b7f-b207-11ea-8de4-080027720095:6'

3 慢日志(slow log)

记录SQL运行较慢的语句,优化过程中常用的工具日志

配置:默认未开启.

自定义:

#慢查询开关

slow\_query\_log=1

#文件位置及名字

slow\_query\_log\_file=/data/mysql/slow.log

#设定慢查询时间

long\_query\_time=3

##没走索引的语句也记录

log\_queries\_not\_using\_indexes

分析慢SQL

mysqldumpslow -s c -t 10 /data/mysql/slow.sql

第三方工具

安装 percona-toolkit

./pt-query-diagest /data/mysql/slow.sql

Anemometer 基于pt-query-diagest将mysql慢查询可视化

MySQL备份恢复

1备份策略的设计

备份周期：每天,每周

备份工具：mysqldump（MDP）,XBK（PBK）percona xtrabackup,MEB(MYSQL Enterprise Backup)

mysqlbinlog

备份方式：

逻辑：全备 mysqldump,增量binlog（执行flush logs后，拷贝）

物理: 全备XBK,增量XBK

\*附加：同一版本mysql

可以物理备份[建表语句+ibd数据+discard tablespace+import tablespace]

2检查备份可用性

crontab -l 备份脚本,备份路径,看备份日志,检查备份文件(大小,内容)

vi /etc/crontab

加任务：

　　　　crontab -e

　　　　0 \*/1 \* \* \* command

　　　　0 \*/2 \* \* \* command

查询任务是否加了：

　　　crontab -l

　　　0 \*/1 \* \* \* command

　　　0 \*/2 \* \* \* command

配置格式

\* \*　 \*　 \*　 \*　　command

分钟(0-59)　小时(0-23)　日期(1-31)　月份(1-12)　星期(0-6,0代表星期天)　 命令

第1列表示分钟1～59 每分钟用\*或者 \*/1表示

第2列表示小时1～23（0表示0点）

第3列表示日期1～31

第4列表示月份1～12

第5列标识号星期0～6（0表示星期天）

第6列要运行的命令

相关参数详细说明：

星号（\*）可以用来代表所有有效的值。譬如，月份值中的星号意味着在满足其它制约条件后

每月都执行该命令。

整数间的短线（-）指定一个整数范围。譬如，1-4 意味着整数 1、2、3、4。

用逗号（,）隔开的一系列值指定一个列表。譬如，3, 4, 6, 8 标明这四个指定的整数。

正斜线（/）可以用来指定间隔频率。在范围后加上 /<integer> 意味着在范围内可以

跳过 integer。譬如，0-59/2 可以用来在分钟字段定义每两分钟。间隔频率值还可以和

星号一起使用。例如，\*/3 的值可以用在月份字段中表示每三个月运行一次任务。

3备份类型

热备:对业务影响小 Innodb

温备：长时间锁表备份 MyISAM

冷备:业务关闭的情况下备份

4逻辑备份mysqldump

-u用户名 -p密码 -h地址 -P端口

-A全备 -B库名 库名 …

库 表 表…

mysqldump -uroot -pabc123 -A >full.sql

mysqldump -uroot -pabc123 -Bgtid test >backup\_db\_$(data +%F).sql

mysqldump -uroot -pabc123 gtid t1 t2 >backup\_table.sql

-R 备份存储过程和函数

-E 事件

--triggers 触发器

--master-data=2

1记录备份时binlog日志信息

2 自动锁表，

不加 --single-transaction，温备份

加了--single-transaction，对于innodb不锁表备份（快照备份）

--single-transaction

对于innodb表,进行一致性快照备份，不锁表。

**--set-gtid-purged=ON/OFF 是否生成gtid信息，普通备份可以是OFF，主从复制**

**必须开启.推荐该参数开启**。

--max\_allowed\_packet=128M 控制备份时传输数据的大小

5物理备份XBK(PKB)

Xtrabackup 针对非InnoDB,进行锁表备份.copy所有的非innodb表文件

针对innodb,立即触发CKPT，会立即记录LSN，copy所有的innodb相关

文件(ibdata1,ibd,frm)，并且将备份过程中产生新的数据变化部分redo进行截取和保存

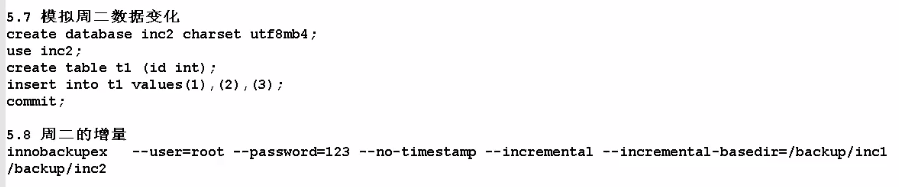
并记录此时最新LSN

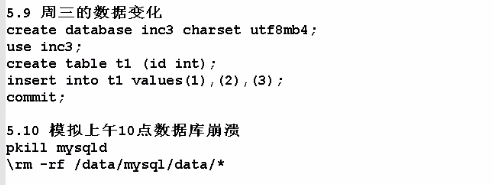
在恢复时,xbk会调用innodb引擎的CSR过程，将数据的LSN 和redo的LSN追平,

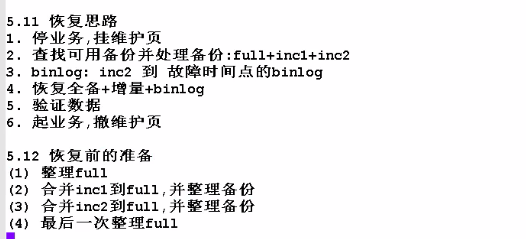
然后进行一致性恢复

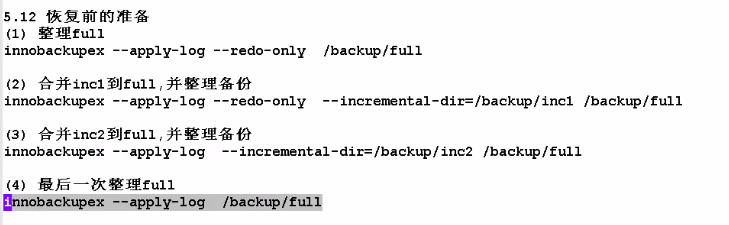
XBK2.4版本

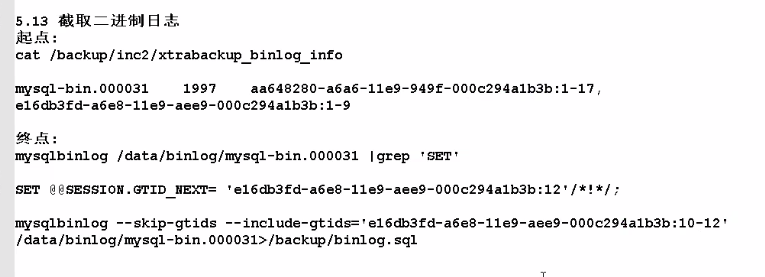


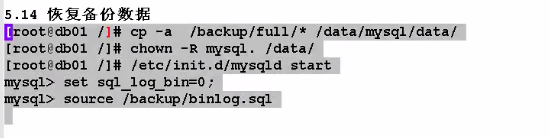














xtrabackup8.0版本全量备份

全备

sudo **xtrabackup** –S /tmp/mysql.sock --user=root --password=abc123 --port=3306

--datadir=/data/mysql **--backup --target-dir=/data/backup**

#停服务,删除现有data目录

sudo /etc/init.d/mysqld stop

恢复前准备

sudo **xtrabackup** **--prepare --target-dir**=/data/backup/

恢复数据

sudo **xtrabackup**  --datadir=/data/mysql **--copy-back --target-dir**=/data/backup/

增量备份

全量备份的目录为：mkdir -p /data/backup

增量备份的目录为： mkdir -p /data/backupIncr

0. 备份操作之前需要停止掉数据库：

1.备份操作：

1.1.全量备份：

**xtrabackup** --defaults-file=/etc/my.cnf -S /tmp/mysql.sock --user=root --password=abc123 **--backup** --parallel=3 **--target-dir=/data/backup/**

1.2.增量备份：

**xtrabackup** --defaults-file=/etc/my.cnf -S /tmp/mysql.sock --user=root --password=abc123

**--backup** --parallel=3 **--target-dir=/data/incr1 --incremental-basedir=/data/backup**

1.3.增量备份：

**xtrabackup** --defaults-file=/etc/my.cnf -S /tmp/mysql.sock --user=root --password=abc123

**--backup** --parallel=3 **--target-dir=/data/incr2 --incremental-basedir=/data/incr1**

2.恢复操作(将增量备份合并到全备,整理全备)：

2.1 准备全备份的日志：

**xtrabackup --prepare --apply-log-only --target-dir**=/data/backup

2.2 准备增量备份的日志：

**xtrabackup --prepare --apply-log-only --target-dir=/data/backup**

**--incremental-dir=/data/incr1**

**xtrabackup --prepare --target-dir=/data/backup --incremental-dir=/data/incr2**

2.3 全备份准备：

# xtrabackup --prepare --target-dir=/data/backup

2.4 拷回数据：

**xtrabackup** -S /tmp/mysql.sock --user=root --password=abc123 --datadir=/data/mysql

**--copy-back --target-dir**=/data/backup/

2.5 修改数据目录的权限和属性：

chmod -R 755 /data/mysql

2.6 启动实例：

/etc/init.d/mysqld start

高可用

Percona XtraDB Cluster（简称PXC集群）提供了MySQL高可用的一种实现方法

相比那些比较传统的基于主从复制模式的集群架构MHA和MM+keepalived，galera cluster最突出特点就是解决了诟病已久的数据复制延迟问题，基本上可以达到实时同步

MySQL Group Replication（MGR）框架让MySQL具备了自动主从切换和故障恢复能力

MySQL主从复制

**主从复制基于binglog来实现**

**主库发生新的操作，都会记录binglog**

**从库取得主库的binglog进行回放**

**主从复制的过程是异步**

1基于GTID主从搭建

GTID（Global Transaction ID）是对于一个已提交事务的唯一编号,并且是一个全局（主从复制）唯一编号

GTID= server\_uuid:tid

核心特性：全局唯一，具备等幂性

核心参数：

gtid-mode=on 启用gtid类型，否则为传统复制架构

enforce-gtid-consistency=true 强制gtid的一致性

log-slave-updates=1 slave更新是否记入日志，强制slave更新二进制日志

1 配置好2个实例分别为3306(主),3307(从）

[mysqld]

user=mysql

basedir=/app/mysql

datadir=/data/mysql

socket=/tmp/mysql.sock

port=3306

mysqlx\_port=33060

pid-file=/data/mysql/mysql3306.pid

log-error=/data/mysql/mysql3306.err

server\_id=1

log\_bin=/data/mysql/binlog/mysql-bin

binlog\_format=row

gtid-mode=on

enforce-gtid-consistency=true

log-slave-updates=1

slow\_query\_log=1

slow\_query\_log\_file=/data/mysql/slow.log

long\_query\_time=1

log\_queries\_not\_using\_indexes

[mysql]

socket=/tmp/mysql.sock

[mysqld]

user=mysql

basedir=/app/mysql

datadir=/data/3307

socket=/tmp/3307.sock

port=3307

mysqlx\_port=33070

pid-file=/data/3307/mysql3307.pid

log-error=/data/3307/mysql3307.err

server\_id=11

log\_bin=/data/3307/binlog/mysql-bin

binlog\_format=row

gtid-mode=on

enforce-gtid-consistency=true

log-slave-updates=1

slow\_query\_log=1

slow\_query\_log\_file=/data/3307/slow.log

long\_query\_time=1

log\_queries\_not\_using\_indexes

[mysql]

socket=/tmp/3307.sock

2 分别启动mysql实例

若存在数据，将主库 开启mysqldump **--set-gtid-purged=ON 备份的数据** 导入到从库后

再构建主从。

https://dev.mysql.com/doc/refman/5.7/en/replication-gtids-failover.html#replication-gtids-failover-copy

在主库上 创建复制用户

create user repl@'10.0.2.\*' identified by 'abc123';

grant replication slave on \*.\* to [repl@'10.0.2.\*](mailto:repl@'10.0.2.*)';

flush privileges;

测试同步当前进度

flush logs;//flush log

show master stauts;//查看主库当前使用的二进制日志文件和位置

a99db85a-c809-11ea-8b69-080027720095:21

根据需要同步的位置,在从库上执行

CHANGE MASTER TO

MASTER\_HOST='10.0.2.15',

MASTER\_USER='repl',

MASTER\_PASSWORD='abc123',

MASTER\_PORT=3306,

master\_auto\_position=1; //读取relaylog一个事务的GTID

start slave

其他问题

SET GTID\_NEXT='a99db85a-c809-11ea-8b69-080027720095:6';

BEGIN;COMMIT;

SET GTID\_NEXT='AUTOMATIC';

删除二进制日志

PURGE BINARY LOGS before '2016-09-01 17:20:00';

PURGE BINARY LOGS TO 'mysql-bin.000029';

#重置

reset slave

reset master

搭建主从使用中碰到的问题解决

Master:

mysql> show master status\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

File: binlog.000002

Position: 160205927

Binlog\_Do\_DB:

Binlog\_Ignore\_DB:

Executed\_Gtid\_Set: 8a6962d2-b907-11e4-bebc-080027880ca6:1-153,

9b09b44a-b907-11e4-bebd-080027880ca6:1,

ab8f5793-b907-11e4-bebd-080027880ca6:1-2

Slave:

mysql> show slave status\G

Retrieved\_Gtid\_Set: 8a6962d2-b907-11e4-bebc-080027880ca6:1-153,

9b09b44a-b907-11e4-bebd-080027880ca6:1

Executed\_Gtid\_Set: 8a6962d2-b907-11e4-bebc-080027880ca6:1-153,

9b09b44a-b907-11e4-bebd-080027880ca6:1,

ab8f5793-b907-11e4-bebd-080027880ca6:1-4

We can check if the slave has any errant transactions by executing the following SQL:

mysql> SELECT GTID\_SUBSET('8a6962d2-b907-11e4-bebc-080027880ca6:1-153,ab8f5793-b907-11e4-bebd-080027880ca6:1-4', 9b09b44a-b907-11e4-bebd-080027880ca6:1') as is\_subset\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

is\_subset: 0

1 row in set (0.00 sec)

Looks like there are errant transactions. How do we identify them? We can use another function, GTID\_SUBTRACT()

mysql> SELECT GTID\_SUBTRACT('8a6962d2-b907-11e4-bebc-080027880ca6:1-153,ab8f5793-b907-11e4-bebd-080027880ca6:1-4', 9b09b44a-b907-11e4-bebd-080027880ca6:1') as mising\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

mising: ab8f5793-b907-11e4-bebd-080027880ca6:3-4

1 row in set (0.01 sec)

How to solve issues caused by errant transactions?

To inject empty transactions we can use the following SQL:

mysql> SET gtid\_next='ab8f5793-b907-11e4-bebd-080027880ca6:3';

mysql> begin ; commit;

mysql> SET gtid\_next='ab8f5793-b907-11e4-bebd-080027880ca6:4';

mysql> begin ; commit;

mysql> SET gtid\_next=automatic;

Another approach is to remove the GTID’s from history:

Stop slave:

mysql> STOP SLAVE;

mysql> SHOW MASTER STATUS\G

mysql> RESET slave ALL;

Set GTID\_PURGED to a correct GTID set. based on data from SHOW MASTER STATUS. You should exclude errant transactions from the set.

mysql> SET GLOBAL GTID\_PURGED='8a6962d2-b907-11e4-bebc-080027880ca6:1-153,

9b09b44a-b907-11e4-bebd-080027880ca6:1, ab8f5793-b907-11e4-bebd-080027880ca6:1-2';

start slave.

mysql> START SLAVE\G

检查从库

show master status;

show slave status\G;

2传统主从搭建

配置好2个实例分别为3306(主),3307(从）

basedir=/app/mysql

datadir=/data/mysql

socket=/tmp/mysql.sock

port=3306

mysqlx\_port=33060

pid-file=/data/mysql/mysql3306.pid

log-error=/data/mysql/mysql3306.err

server\_id=1

log\_bin=/data/mysql/binlog/mysql-bin

binlog\_format=row

slow\_query\_log=1

slow\_query\_log\_file=/data/mysql/slow.log

long\_query\_time=1

log\_queries\_not\_using\_indexes

[mysql]

socket=/tmp/mysql.sock

[mysqld]

user=mysql

basedir=/app/mysql

datadir=/data/3307

socket=/tmp/3307.sock

port=3307

mysqlx\_port=33070

pid-file=/data/3307/mysql3307.pid

log-error=/data/3307/mysql3307.err

server\_id=11

log\_bin=/data/3307/binlog/mysql-bin

binlog\_format=row

slow\_query\_log=1

slow\_query\_log\_file=/data/3307/slow.log

long\_query\_time=1

log\_queries\_not\_using\_indexes

[mysql]

socket=/tmp/3307.sock

在主库上 创建复制用户

create user repl@'10.0.2.\*' identified by 'abc123';

grant replication slave on \*.\* to [repl@'10.0.2.\*](mailto:repl@'10.0.2.*)';

flush privileges;

备份主库数据库，并导入到从库

mysqldump -uroot -p123 –A –master-data=2 --single-transaction –E –R –triggers > /tmp/full.sql

从库 ：

set sql\_bin\_log=0;

source /tmp/ful.sql

从 full.sql里 知道文件，起始位置

开启

CHANGE MASTER TO

MASTER\_HOST='10.0.2.15',

MASTER\_USER='repl',

MASTER\_PASSWORD='abc123',

MASTER\_PORT=3306,

MASTER\_LOG\_FILE='mysql-bin.000033',

MASTER\_LOG\_POS=4,

MASTER\_CONNECT\_RETRY=10;

重新改变change master to

STOP SLAVE;

RESET slave ALL;

CHANGE MASTER TO ……

Start slave

主库 Show master status\G;

从库 show master status\G;

检查主从复制状态: show slave status\G ;

Slave\_IO\_Running:Yes

Slave\_SQL\_Running:Yes

3主从复制原理