MySQL日常维护操作—iCloud

```
insert into sxw select id,product from zt_bug

EXPLAIN SELECT * FROM student WHERE LOCATE('大',`nickname`) LIMIT 10
```

my.ini中添加log-queries-not-using-indexes参数，表示记录下没有使用索引的查询。查找my.cnf： ./mysql --help | grep my.cnf

MHA自动切换后处理步骤.
1.去挂掉的master启动service mysqld start 然后：reset master
2.指向新的主库.一定要看看是slave搞错过

```
CHANGE MASTER TO MASTER_HOST='192.168.100.166',
MASTER_USER='rep',
MASTER_PASSWORD='123456',
MASTER_LOG_FILE='mysql-bin.000003',
MASTER_LOG_POS=535;
```

start slave;
3.启动：service keepalived start
4.登陆MHA的管理节点删除，文件`app1.failover.complete`
5.启动MHA
启动MHA：nohup /soft/mha4mysql-manager-0.56/bin/masterha_manager --conf=/etc/masterha/app1.cnf &
停止MHA：./masterha_stop --config=/etc/masterha/app1.cnf
检查MHA的运行情况：masterha_check_status --conf=/etc/masterha/app1.cnf
app1 (pid:3431) is running(0:PING_OK), master:192.168.100.99

--remove_dead_master_conf 该参数代表当发生主从切换后，老的主库的ip将会从配置文件中移除。
--manger_log 日志存放位置
--ignore_last_failover 在缺省情况下，如果MHA检测到连续发生宕机，且两次宕机间隔不足8小时的话，则不会进行Failover，之所以这样限制是为了避免ping-pong效应。 该参数代表忽略上次MHA触发切换产生的文件 ，默认情况下，MHA发生切换后会在日志目录，也就是上面我设置的/data产生app1.failover.complete文件，下次再次切换的时候如果发现该目录下存在该文件将不允许触发切换，除非在第一次切换后收到删除该文件，为了方便，这里设置为--ignore_last_failover。

#show processlist这里的state列很重要，
`kill -TERM`就相当于`service mysql stop`

Closing tables正在将表中修改的数据刷新到磁盘中，同时正在关闭已经用完的表。这是一个很快的操作，如果不是这样的话，就应该确认磁盘空间是否已经满了或者磁盘是否正处于重负中。
Connect Out：复制从服务器正在连接主服务器
Copying to tmp table on disk：由于临时结果集大于tmp_table_size，正在将临时表从内存存储转为磁盘存储以此节省内存。
Creating tmp table：正在创建临时表以存放部分查询结果。
Sending data：正在处理 SELECT 查询的记录，同时正在把结果发送给客户端。
如果大量的sleep出现：可以设置writ_timeout和interactive_timeout(如果sleep大量出现堆积到超过了max_connect的最大连接数后就只能root用户登陆了。)发生这个问题如程序中长连接多，或没关闭连接

#mysql错误码
Error reading packet from server: Lost connection to MySQL server during query (server_errno=2013) 主从复制出错 网络问题 跨机房出现
ERROR 1040: Too many connections错误，可以过’conn%’通配符查看当前状态的连接

数量
show status like 'threads_connected%’
#show status
#show variables like 'slow%’ 慢查询
#mysqldumpslow ctb-test-slow.log -s t -t 5 查看慢查询日志查询时间最长的前5条

Count: 2 Time=2.79s (5s) Lock=0.00s (0s) Rows=1.0 (2), 告诉我们执行了2次，最大时间是2.79s,总共花费时间5s,lock时间0s,单次返回的结果数是1条记录，2次总共返回2条记录m
-s, 是表示按照何种方式排序，c、t、l、r分别是按照记录次数、时间、查询时间、返回的记录数来排序，ac、at、al、ar，表示相应的倒叙；
-t, 是 top n的意思，即为返回前面多少条的数据；
-g, 后边可以写一个正则匹配模式，大小写不敏感的；[mysqld]
myisam-recover=BACKUP,FORCE
#ANALYZE LOCAL TABLE 优化表
#REPAIR LOCAL TABLE 修复表
SELECT * FROM tb_name PROCEDURE ANALYSE();
#OPTIMIZE LOCAL TABLE 减少碎片提高IO性能回收空间等

如果在集群里则用OPTIMIZE LOCAL TABLE table_name
如果是MyISAM 直接可可运行
如果是innoDB先看看是否开启innodb_file_per_table（是否独享表空间）show variables like ‘indoor_file_per_file’如果是OFF会在MYSQL的datadir下面有一个ibddata1的文件这里存储着数据和索引文件。set global innodb_file_per_table = OFF

如果MYSQL负载高，先iostart -x 1看看IO是否高iowate(IO处理响应)才4.5，空闲(idle)如果都没事那应该是程序的事。show status like ‘%key_%’;用这里的key_write/key_write_requre如果小就是说明update delete insert很平凡，需要打开delay_key_write了. alter table tb_name delay_key_write=1;但是用了这个在断电或重起的时候要用—myisam-recover或在con设置myisaim-recover=BACKUP,FORCE这样在启动mysql的时候会检查表并同步表和索引。如：misaim-recover=BACKUP,FORCE.这里有好几个参数：

SET GLOBAL `sort_buffer_size=9888`使用SHOW GLOBAL VARIABLES才能看见

DEFAULT(相当于没使用myisam-recover) BACKUP(如果回复时文件更改了把原来的name.MYD改为name.datatime.bak) FORCE(即使.MYD文件将丢掉多个行也进行恢复) QUICK()

show profile;
set profiling = 1;
show profile for query 3
show profile cpu for query 2;
select * from bind_link_event
set profiling = 0
select @@profiling
show process list;
`set @query_id=7;`

```
SELECT STATE, SUM(DURATION) AS Total_R,
       ROUND(
            100 * SUM(DURATION) /
               (SELECT SUM(DURATION)
                FROM INFORMATION_SCHEMA.PROFILING
                WHERE QUERY_ID = @query_id
            ), 2) AS Pct_R,
         COUNT(*) AS Calls,
         SUM(DURATION) / COUNT(*) AS "R/Call"
```

```
      FROM INFORMATION_SCHEMA.PROFILING
 WHERE QUERY_ID = @query_id
GROUP BY STATE
ORDER BY Total_R DESC;
```

mysql> use information_schema;

select user,host,time from information_schema.processlist where user='read_user' ord er by time

/usr/local/mysql/bin/mysqladmin ext -uroot -p -ri10 | grep Key_reads

#查看表里的总大小 SELECT *,concat(round(sum(DATA_LENGTH/1024/1024),2),'MB') A S t FROM information_schema.tables where table_schema='ctb_safetrip' and table_na me='point_info'

SELECT id,pid,sum(lFo) AS lFo FROM bind_link_event GROUP BY pid WITH ROLLUP LIMIT 4 //WITH ROLLUP 不能和ORDER BY同时使用。这是OLAP思想(数据仓库)

SHOW GLOBAL STATUS LIKE '%innodb_data_read%';
Innodb_data_read 46384854470656
Innodb_data_reads 2812101578
SELECT data_read/data_reads=16494
每次IO平均读取字节数=16494
Innodb_data_read 表示Innodb启动后，从物理磁盘上读取的字节数总和。
Innodb_data_reads 表示Innodb启动后，队伍物理磁盘发起的IO请求次数总和。
Innodb_data_read / Innodb_data_reads 得到的比值，越接近16K说明IO压力越倾向于随机IO，越远离16K说明IO从顺序预读中获得性能提升越多
#————————MYSQL 优化 ————————————————

optimizer_switch如果开启(set optimizer_switch="index_condition_pushdown=on")在5.6会使用ICP，使用ICP后在Extra里会看见Using Index Condition，必须是二级索引

先看写入和插入show status like 'indoor_row%'
---
show status like 'handler_%'

Handler_read_key 如果值高说明索引用的高，如果不高则说明索引带不来多少性能
Handler_read_rnd_next 如果值高说明查询效率低需要索引来补救或索引不正确，这个的意思是在数据文件中读下一行。

计算前缀索引的SQL，值接近0.31就可以了,主要看数据是否均匀，前缀索引不能在GROUP BY 和 ORDER BY里使用

```
SELECT
      COUNT( DISTINCT LEFT(user_id,4) )/COUNT(1) AS uid4,
      COUNT(DISTINCT LEFT(user_id,5))/COUNT(1) AS uid5,
      COUNT(DISTINCT LEFT(user_id,6))/COUNT(1) AS uid6
FROM user_coordinate
+--------+--------+--------+
| uid4   | uid5   | uid6   |
+--------+--------+--------+
| 0.0112 | 0.0325 | 0.1057 |
+--------+--------+--------+
```

代理健(Surrogate Key)：在Innodb中插入数据时用自增主健和uuid主健性能不同，自增主健要快而索引也会相对较小，因为有自增主健的时候在插入的时候是顺序插入(相当于

每插入的时候都会把记录放在前一条数据的后面)之样页面会很紧凑，如果是UUID在插入的时候记录主健不一定比前一个大，因此Innodb不能总是把新行插入到索引最后，它会去为新行寻找合适的位置，消耗时间大部分是在于分页和碎片造成。但是自增主健在大并发写入的时候会造成Innodb的内部单点竞争，因为所有的插入都发生在那，并发插入可能会竞争下一个健锁并/或AUTO_INCREMENT锁。

覆盖索引(Covering Index)：包含所有满足查询需要的数据的索引叫覆盖索引，它可以利用索引直接返回select列中的字段，而不用根据索引再次读取数据文件。

limit 优化：

```
SELECT
	pid,uid,uname,time,lat,lng
FROM point_info
INNER JOIN (
	SELECT pid,uid FROM point_info
LIMIT  600000,10 ) AS x USING(pid,uid)
```

在MYSQL中ALTER MODIFY 字段时都会导致表的构健，列的默认值都保存在.frm文件中。ALTER TABLE test ALTER COLUMN status SET DEFAULT 1 可以跳过构健表，它只是更改了.frm文件。

1.CREATE TABLE tb_ LIKE tb_ 新表
2.alter table hia modify column hi set('Y','N','ON','H') default ‘Y’;修改新表
3.FLUSH TABLES WITH READ LOCK
enum和set枚举是可以使用索引的

show status like 'last_query_cost%’ #sql的随机读取值越低越好。运行一次改变一次

许多数据库服务器都把IN看作多个OR，因为处理逻辑上是相等的，但MYSQL不是，它会对IN里面的数据进行排序，然后进行二分查找这个算法是O(Log n),而相同的OR子句查找效率是O(n),在列表很大的时候用or要比in慢的多。

紧凑索引扫描（Tight index scan）和使用松散索引扫描（Loose index scan）的区别主要在于：紧凑索引扫描需要在扫描索引的时候，读取所有满足条件的索引键，然后再根据读取出的数据来完成 GROUP BY 操作得到相应结果。
松散索引扫描（Loose index scan）实际上就是当 MySQL 完全利用索引扫描来实现 GROUP BY 的时候，并不需要扫描所有满足条件的索引键即可完成操作得出结果。在Extra 信息中有信息显示“Using index for group-by”就是使用了松散索引
#松散索引实例：在customer_id和kind有联合索引，SELECT的字段从左到右只要组合起来是前缀索引都会使用到索引
EXPLAIN SELECT customer_id FROM order_rab WHERE kind > 1#Using where; Using index因为SELECT的字段和WHERE组合起来正好是索引
EXPLAIN SELECT customer_id FROM order_rab ORDER BY kind#Using index; Using file sort
EXPLAIN SELECT * FROM order_rab WHERE customer_id=2 ORDER BY kind #Using where
如果EXPLAIN 出现using index for group-by表示用了松散索引扫描。
ORDER/GROUP BY 优化都好使
EXPLAIN SELECT * FROM order_rab WHERE customer_id=2 AND kind>1 ORDER BY kind#Usingindex condition索引同上
Innodb在使用count的时候有可能使用second index要比cluster index快，因为cluster是和数据row存在一起的而second是单独存放然后有个指针指向primary key。所以cluster就慢在要返回巨大的结果集。

select min(id) from user_coordinate where gender=0
select id from user_coordinate use index(primary) where gender=0 limit 1#高效率，因为主健都是以升序排的

```
explain select * from hia where id=13 order by status#则不会
explain select * from hia where id<13 order by status #Using where; Using filesort 在范围后会多一次排序
```

Extra的解释

1. Using where
这个值表示查询使用了where 语句来处理结果——例如执行全表扫描。如果也用到了索引，那么行的限制条件是通过获取必要的数据之后处理读缓冲区来实现的。
2. Using temporary
这个值表示使用了内部临时(基于内存的)表。一个查询可能用到多个临时表。有很多原因都会导致MySQL 在执行查询期间创建临时表。两个常见的原因是在来自不同表的列上使用了DISTINCT，或者使用了不同的ORDER BY 和GROUP BY 列。
3.of_query_execution_and_use_of_temp_tables。
可以强制指定一个临时表使用基于磁盘的MyISAM 存储引擎。这样做的原因主要有两个：
A 内部临时表占用的空间超过min(tmp_table_size，max_heap_table_size)系统变量的限制
B 使用了TEXT/BLOB 列
3. Using filesort
这是ORDER BY 语句的结果。这可能是一个CPU 密集型的过程。可以通过选择合适的索引来改进性能，用索引来为查询结果排序。详细过程请参考第4 章。
4. Using index
这个值重点强调了只需要使用索引就可以满足查询表的要求，不需要直接访问表数据。请参考第5章的详细示例来理解这个值。
5. Using join buffer
这个值强调了在获取连接条件时没有使用索引，并且需要连接缓冲区来存储中间结果。如果出现了这个值，那应该注意，根据查询的具体情况可能需要添加索引来改进性能。
6. Impossible where
这个值强调了where 语句会导致没有符合条件的行。请看下面的示例：

```
mysql> EXPLAIN SELECT * FROM user WHERE 1=2;
```

7. Select tables optimized away
这个值意味着仅通过使用索引，优化器可能仅从聚合函数结果中返回一行。请看下面的示例：
8. Distinct
这个值意味着MySQL 在找到第一个匹配的行之后就会停止搜索其他行。
9. Index merges
当MySQL 决定要在一个给定的表上使用超过一个索引的时候，就会出现以下格式中的一个，详细说明使用的索引以及合并的类型。
- Using sort_union(...)
- Using union(...)
- Using intersect(...)

SQL 语句

```
SELECT SUM(IF(status !='A',1,0)) / COUNT(*) * 100 FROM tb
SELECT GROUP_CONCAT(str separator '|') FROM bbcca GROUP BY str
```

#count()优化
计算出customer_id等于1和3的总和
SELECT *,SUM(IF(customer_id=2,1,0)) AS red,sum(IF(customer_id=3,1,0)) AS bai FROM order_rab HAVING red=1
count优化在mysql内部唯一只能用覆盖索引

SELECT SUM(kind) FROM order_rab GROUP BY customer_id WITH ROLLUP#可以统计所有的总和加在最后一列
查询缓存命中率：Qcache_hits/(Qcache_hits+Comm_select)越大越好。
可以根据Qcache_lowmem_prunes的值来了解有有多少查询因为内存不够导致失效

Variable_name	Value
Innodb_buffer_pool_pages_data	70
Innodb_buffer_pool_pages_dirty	0
Innodb_buffer_pool_pages_flushed	0
Innodb_buffer_pool_pages_free	1978
Innodb_buffer_pool_pages_latched	0
Innodb_buffer_pool_pages_misc	0
Innodb_buffer_pool_pages_total	2048
Innodb_buffer_pool_read_ahead_rnd	1
Innodb_buffer_pool_read_ahead_seq	0
Innodb_buffer_pool_read_requests	329
Innodb_buffer_pool_reads	19
Innodb_buffer_pool_wait_free	0
Innodb_buffer_pool_write_requests	0

从上面的值我们可以看出总共2048 pages，还有1978是Free状态的仅仅只有70个page有数据，read请求329次，其中有19次所请求的数据在buffer_pool中没有，也就是说有19次是通过读取物理磁盘来读取数据的，所以很容易也就得出了Innodb_Buffer_Pool的Read命中率大概在为(329-19)/329*100%= 94.22%。 (Innodb_buffer_pool_read_requests-Innodb_buffer_pool_reads)/Innodb_buffer_pool_read_requests*100单从这里的数据来看，我们设置的 Buffer_Pool过大，仅仅使用70/2048*100%=3.4%。Innodb_buffer_pool_pages_data/Innodb_buffer_pool_pages_total=3.4%如>95%则增加indoor_buffer_pool_size，如果<95%则减少，建议使用内存总数的75%

insert into b (`id`,`comment`) select * from a

```
insert into  point_info_hi SELECT * FROM (`point_info`) WHERE lat <= 4103000 and lat >= 3926000 and lng <= 11730000 and lng >= 11525000
```

#MYSQL 不能打开更多文件 [ERROR] Error in accept: Too many open files
打开文件数达到上限了，需要提高上限，或者释放部分已打开的表文件描述符。
MySQL中，有几个地方会存在文件描述符限制：
1、在Server层，整个mysqld实例打开文件总数超过用户进程级的文件数限制，需要检查内核 fs.file-max 限制、进程级限制 ulimit -n 及MySQL中的 open-files-limit 选项，是否有某一个超限了。任何一个条件超限了，就会抛出错误。
2、虽然Server层总文件数没有超，但InnoDB层也有限制，所有InnoDB相关文件打开总数不能超过 innodb-open-files 选项限制。否则的话，会先把最早打开的InnoDB文件描述符关闭，才能打开新的文件，但不会抛出错误，只有告警信息。
相应地，如果提示超出限制，则可以使用下面方法提高上限：
1、首先，提高内核级总的限制。执行：sysctl -w fs.file-max=3264018；
2、其次，提高内核对用户进程级的打开文件数限制。执行：ulimit -n 204800；
3、最后，适当提高MySQL层的几个参数：open-files-limit、innodb-open-files、table-open-cache、table-definition-cach。

stop slave ; start slave io_thread;只启用IO_THREAD
start slave sql_thread只启用SQL线程

InnoDB 工作如何刷新IO和如何执行IO

写：缓冲池->IO线程->操作系统缓存->文件系统((表空间->双写缓冲)，(数据，索引),数据文件)->日志缓冲区->...
读：文件系统((表空间->双写缓冲)，(数据，索引),数据文件)->操作系统缓存->IO线程

mysql的SELECT 通常是非锁定的，但如果是INSERT....SELECT会锁定它读取的所有行。

mysqldum -u root -p123456 -h 192.168.32.71 库名 表名 > *.sql
--all-databeses为全部数据库.

导入：`h-3.2# ./mysql -uroot -p < hi.sql`

SHOW TABLE STATUS LIKE ‘a%’

MYSQL 新健用户给权限和收回权限

grant all privileges on active_safetrip.* to sop_user@“%” identified by '89vlz6Z71rAqr5iV'; 给用户分配权限并设置密码
格式：grant 权限 on 数据库.* to 用户名@登录主机 identified by "密码";
如果想指定部分权限给一用户，可以这样来写:
grant select,update on testDB.* to test@localhost identified by '1234';
flush privileges; //刷新系统权限表
set password for 'sxw'@'localhost'=password('654321’);//修改密码
show grants for 'sxw'@'localhost'; //查看用户授权信息
show grants for current_user();//查看当前用户授权信息
revoke select on *.* from 'sxw'@‘localhost';//回收权限
drop user 'sxw'@‘localhost';//删除用户
GRANT ALL PRIVILEGES ON *.* TO shi_root@'localhost' IDENTIFIED BY '123456' WHITH GRANT OPTION;
如果直接UPDATE修改用FLUSH PRIVILEGES才生效
REVOKE select ON *.* FROM 'shixiaowen'@'localhost';
show grants;show grants for 'dab'@'localhost'
skip_grant_tables:管理员忘记密码跳过检测grant表。

通过localhost还是127.0.0.1:用localhost时MYSQL会默认使用unix socket去连接而不是TCP/IP如：
mysql --host=localhost:这个会用Unix socket连接
mysql --host=127.0.0.1或--host=localhost --protocol=tcp:这两个都会能通过TCP/IP连接
所以在指定user@localhost不是多余的。

mysqlreport分析ini配制:

```
[root@ctb-test-1 phpuser]# /usr/bin/mysqlreport --host=localhost --user=root --password=@163.com --outfile=/mnt/mysqlreport.txt
```

buffer uset 最高使用的值，ccurrent当前用到的，这两个加起来可以说明key_buffer是否够用。这是myisam专用。

pt-query-digest查询日志分析工具

每秒对Threads_connected,running和query进行采样

Queries每秒查询数，Threads_connected当前连接数，
```
[root@ctb-test-1 bin]# mysqladmin ext -uroot -p -i1 |awk '/Queries/{q=$4-qp;qp=
$4} /Threads_connected/{tc=$4} /Threads_running/{printf "%5d %5d %5d\n",q,tc,$4
}'
```

每秒分析慢查询log

```
[root@ctb-test-1 bin]# awk '/^# Time:/{print $3,$4,c;c=0}/^# User/{c++}' /mnt/s
low-log/ctb-test-slow.log | head -n10
```
head -n10当前的前10

统计mysql每行值
```
mysql -root -p -e 'SHOW PROCESSLIST\G' | grep State: | sort | uniq -c | sort -r
n
mysql -u root -p -e 'SHOW PROCESSLIST\G' | grep -c 'State: init'
```
统计 state:init有多少

用bin-log恢复数据

```
/server/mysql-5.6.20/bin/mysqlbinlog  -vv --start-position=23410 --stop-positio
n=24638   /log/mysql-5.6.20/mysql-bin.000004 | mysql -u root -p
```

mysql慢查询分析工具：pt-query-digest

安装 wget percona.com/get/pt-query-digest
chmod u+x pt-query-digest
可以分析binlog,tcpdump,general log
```
pt-query-digest --type=binlog localhost.log > slow…..
pt-query-digest --since=12h /mnt/mysql/slow.log > new.log#分析12小时内的日志
--since='2014-04-12 09:30:00' --until='2014-04-12 10:30:00' …..#指定时间内
[root@ctb-test-1 phpuser]# ./pt-query-digest --filter '$event->{fingerprint} =~
 m/^select/i' /mnt/slow-log/ctb-test-slow.log> slow_report4.log#只查看select的
pt-query-digest—filter '($event->{user} || "") =~ m/^root/i' slow.log> slow_rep
ort5.log#某个用户
```

查看服务器运行SQL有什么不一样：pt-upgrade

查看SQL文件
```
pt-upgrade h='localhost' h=192.168.3.92 --user=root --password=zhang@123  aaa.s
ql
```
查看某条SQL
```
pt-upgrade h='localhost' h=192.168.3.92 --user=root --password=zhang@123 --quer
y="select * from user_data.collect_data limit 5"
```

解决事务锁的问题

```
SELECT r.trx_id waiting_trx_id,

       r.trx_mysql_thread_id waiting_thread,
       r.trx_query waiting_query,
       b.trx_id blocking_trx_id,
       b.trx_mysql_thread_id blocking_thread,
       b.trx_query blocking_query
   FROM       information_schema.innodb_lock_waits w
   INNER JOIN information_schema.innodb_trx b  ON
    b.trx_id = w.blocking_trx_id
  INNER JOIN information_schema.innodb_trx r  ON
    r.trx_id = w.requesting_trx_id;
```
打开general_log一段时间show variables like 'general_log',然后根据SHOW ENGINE INNODB STATUS里的thread_id进行关联查找。
gap锁update b set times=3 where id > 3 等3的没事，大于3的并有记录无法update，大于3无记录的无法写入。读没事。

在线DDLpt-online-schema-change

1.安装percona-toolkit

```
yum install perl-DBI
```

```
yum install perl-DBD-MySQL
yum install perl-Time-HiRes
yum install perl-IO-Socket-SSL
```

或：`yum install perl perl-DBI perl-DBD-MySQL perl-IO-Socket-SSL perl-Time-HiRes -y`

```
perl Makefile.PL
```

如果报错：`[root@ctb-db-s2 percona-toolkit-2.2.15]# perl Makefile.PL`

```
Can't locate ExtUtils/MakeMaker.pm in @INC (@INC contains: /usr/local/lib64/perl5 /usr/local/share/perl5 /usr/lib64/perl5/vendor_perl /usr/share/perl5/vendor_perl /usr/lib64/perl5 /usr/share/perl5 .) at Makefile.PL line 1.
BEGIN failed--compilation aborted at Makefile.PL line 1.
```

则安装**perl-devel**如下：

```
[root@ctb-db-s2 percona-toolkit-2.2.15]# yum -y install perl-devel
make && make install
```

收集指定进程IO使用情况

```
pt-ioprofile -p 8871 --cell=sizes
```

使用参数 `--cell=sizes`，该参数将结果已 `B/s` 的方式展示出来：

增加字段：

```
./pt-online-schema-change -u root -h 127.0.0.1  -p @163.com --alter='add column vidss int' --execut D=ctb_safetrip,t=water_user_car
```

删除字段

```
./pt-online-schema-change -u root -h 127.0.0.1 -p @163.com --alter "drop column vid" --execut D=ctb_safetrip,t=water_user_car
```

添加索引：

```
./pt-online-schema-change -u root -h 127.0.0.1 -p @163.com --alter "add index user_id_indx(user_id)" --execut D=ctb_safetrip,t=water_user_car
```

格式如下：**--critical-load="Threads_running=200"** 或者**--critical-load="Threads_running:200"**。根据参数要求，修改了**Threads_runnings**的阀值，成功执行了

删除索引：

```
./pt-online-schema-change -u root -h 127.0.0.1 -p @163.com --alter "drop index user_id_indx" --execut D=ctb_safetrip,t=water_user_car
```

查找沉余索引pt-duplicate-key-checker

```
./pt-duplicate-key-checker -u root -p @163.com --databases=ctb_safetrip  --socket=/var/lib/mysql/mysql.sock --noclustered

./pt-online-schema-change -u root -h 127.0.0.1  -p @163.com --alter=add column hash_pix char(5) NOT NULL DEFAULT '0' COMMENT 'geo哈 希值前辍5公里'" --execut D=ctb_safetrip,t=point_edog2

./pt-online-schema-change -u root -h 127.0.0.1 -p @163.com --alter="add column `hash_pix` char(5) NOT NULL DEFAULT '0' COMMENT 'geo哈 希值前辍5公里'" --execut D=ctb_safetrip,t=point_edog2
```

查看当前从库的延迟

./pt-slave-delay -h 192.168.100.156 -u root -p123456 在从库上运行，查看当从库同步的延迟

查看主从的数据是否一致

pt-table-checksum h='127.0.0.1',u='root',p='123456',P=3306 -d ctb_safetrip --nocheck-replication-filters --replicate=ctb_safetrip.checksums 在主库存上运行

安装xtrabackup

如果遇到冲突：`rpm -ivh MySQL-client-5.5.34-1.linux2.6.x86_64.rpm --nodeps --force`

[root@localhost soft]# rpm -ivh percona-xtrabackup-2.2.9-5067.el6.x86_64.rpm

warning: percona-xtrabackup-2.2.9-5067.el6.x86_64.rpm: Header V4 DSA/SHA1 Signature, key ID cd2efd2a: NOKEY

error: Failed dependencies:

libaio.so.1()(64bit) is needed by percona-xtrabackup-2.2.9-5067.el6.x86_64

libaio.so.1(LIBAIO_0.1)(64bit) is needed by percona-xtrabackup-2.2.9-5067.el6.x86_64
libaio.so.1(LIBAIO_0.4)(64bit) is needed by percona-xtrabackup-2.2.9-5067.el6.x86_64
rsync is needed by percona-xtrabackup-2.2.9-5067.el6.x86_64
[root@localhost soft]# yum -y install perl perl-devel libaio libaio-devel perl-Time-HiRes perl-DBD-MySQL
yum install perl-DBI
yum install perl-DBD-MySQL
yum install perl-Time-HiRes
yum install perl-IO-Socket-SSL
如果还不行：再安装MySQL-devel-5.6.13-1.el6.x86_64.rpm MySQL-shared-5.6.13-1.el6.x86_64.rpm MySQL-shared-compat-5.6.13-1.el6.x86_64.rpm

增加一台备库

1.备份主库：
/usr/bin/innobackupex --defaults-file=/etc/my.cnf --user=root --password=CTBgeili_~ --stream=tar /mnt/backup |gzip 1>/mnt/backup/wxlk_user_20150819.tar.gz &
--throttle=#每秒IO次数,限制backup时使用的I/O操作量，使备份对数据库正常业务的影响最小化
--parallel=4 并行个数
--databases=ctb_safetrip
例：/usr/bin/innobackupex --defaults-file=/etc/my.cnf --user=root --password=123456 --parallel=40 --throttle=100 --stream=tar /backup/ | gzip 1>/backup/aha.gz&
然后：tar -ixzvf 进行解压
2.在从库操作
service mysqld stop
innobackupex --defaults-file=/etc/my.cnf --user=root --password=123456 --apply-log /backup
innobackupex --defaults-file=/etc/my.cnf --user=root --password=123456 --copy-back /backup
如果报错
[root@localhost mysql-5.6.20]# service mysqld start
Starting MySQL. ERROR! The server quit without updating PID file (/server/mysql-5.6.20/data/localhost.pid).
权限不够：chmod -R 777或755 data/
然后再启动
然后在恢复后的data目录找xtrabackup_info文件里面有坐标和LSN

主从增量备份备份备份备份

--incremental-basedir原来的全备URL
--incremental 上一次备份的
1.innobackupex --defaults-file=/etc/my.cnf --user=root --password=123456 /backup/a 第一次的全备
2.innobackupex --defaults-file=/etc/my.cnf --user=root --password=123456 --incremental-basedir=/backup/a --incremental /backup/b
3.innobackupex --defaults-file=/etc/my.cnf --user=root --password=123456 --incremental-basedir=/backup/b/ --incremental /backup/c

主从增量恢复恢复恢复恢复

实例

innobackupex --apply-log --redo-only BASE-DIR BASE-DIR指完整的全部备份目录
innobackupex --apply-log --redo-only BASE-DIR --incremental-dir=INCREMENTAL-DIR-1 INCREMENTAL-DIR-1指第一次增量备份的目录
innobackupex --apply-log BASE-DIR --incremental-dir=INCREMENTAL-DIR-2 BASE-DIR指完整的全部备份目录 INCREMENTAL-DIR-2第二次增量备份的目录

开始恢复：

--redo-only --apply-log组，强制备份日志时只redo ,跳过rollback。这在做增量备份时非常必要。 所以在全量备份和增量备份时，需要加上该参数
1.innobackupex --defaults-file=/etc/my.cnf --user=root --password=123456 --apply-log --redo-only /backup/a/
2.innobackupex --defaults-file=/etc/my.cnf --user=root --password=123456 --apply-log --redo-only /backup/a/ --incremental-dir=/backup/b
3.innobackupex --defaults-file=/etc/my.cnf --user=root --password=123456 --apply-log /backup/a/ --incremental-dir=/backup/c/
4.innobackupex --defaults-file=/etc/my.cnf --user=root --password=123456 --appl

```
y-log /backup/a/ --incremental-dir=/backup/d/
02.回滚未完成的日志(需要注意的是恢复的时候,我们只恢复全备份文件就可以了)
innobackupex --defaults-file=/etc/my.cnf --user=root --password=123456 --apply-
log /backup/a/
innobackupex --defaults-file=/etc/my.cnf --user=root --password=123456 --copy-b
ack /backup/a/


innobackupex --defaults-file=/etc/my.cnf --user=root --password=123456  --incre
mental-basedir=/backup/a/ --incremental /backup/b/
```

修改密码：mysql> update user set password=123456 where user='root'

如果进不去可以使用以下方法：

1.关闭mysql

service mysqld stop

2.屏蔽权限

mysqld_safe --skip-grant-table

第二步回车之后，会出现两行，其中有Starting demo from ******，这时需要新开一个窗口进行数据库的登陆操作。

3.新开起一个终端输入

mysql -u root mysql

mysql> UPDATE user SET Password=PASSWORD('123456') where USER='root';

mysql> FLUSH PRIVILEGES;//记得要这句话，否则如果关闭先前的终端，又会出现原来的错误

mysql> \q

----------主从维护----------

可以用pt-table-checksum查看主从数据是否一致情况，通常在主库上运行该工具。如下：

pt-table-checksum —replicate=test.checksum <master_host>

命令将检查所有的表，并把结果插入到test.checksum表中。

```
pt-table-checksum h='127.0.0.1',u='root',p='123456',P=3306 -d ctb_safetrip --
nocheck-replication-filters --replicate=ctb_safetrip.checksums
```

在从库报错binlog可以在主库查看SHOW BINLOG EVENTS IN ‘mysql-bin.000223’ FROM 13634\G

增加一台从库

1.

```
grant replication slave,replication client on *.* to 'repl'@'192.168.100.%'
identified by '123456’;
```

2.运行

```
change master to master_host='192.168.100.156',
        master_user='repl',
        master_password='123456',
        master_log_file='mysql-bin.000001',
        master_log_pos=0
```

3.start slave

记划内把从库提升为主库：

1.停止向老的主库写入 FLUSH TABLES WITH READ LOCK 让备库根上日志。

2.把备库配置为新的主库设置read-only为0

3.在新主库上stop slave

4.在新主库备库运行CHANGE MASTER TO并指定合适的值。然后再执行RESET SLAVE使其断开与老主库的连接，并丢掉master.info里的信息

5.执行show master status记录新主库的二进制日志坐标。

6.关闭旧主库，将客户端连接到新主库，在每台备库上执行SHOW MASTER STATUS后再进行CHANGE MASTER TO写上相应的坐标和点

可以用select master_pos_wait(file,pos,timeout)在slave上查看返回的值代表所等的秒数，timeout<=0时为同步完，超时为-1

如果一主二从，主宕机后：

A办法：可以show slave status查看A slave的Master_Log_File:mysql-bin.000788和Read_Master_Log_Pos:765876的值，然后再看另一台从库存B slave是Read_Master_Log_Pos:765870。A机器的Read_Master_Log_Pos较大说明较新，可以用它来提升主库存。用两台从库的Read_Master_Log_Pos相减得到的数就是落后的字节数如上：765876-765870＝5。然后在A机器的mysql-bin.000788里找到最后执行完的文件SHOW BINLOG EVENTS IN 'mysql-bin.00078'如最后一行的pos号是8167，就用8167-5=8162.然后在落后的B节点执行CHANGE MASTER TO。。。。。。。。。。。。MASTER_LOG_FILE='mysql-bin.000788',MASTER_LOG_POS=8162

B办法：先在从库存用START SLAVE UNTIL MASTER_LOG_FILE='mysql-bin.000788' MASTER_LOG_POS= 765876让slave赶上新备库。再用

SELECT MASTER_POS_WAIT('master-bin.0000', 765876)看看是否同步完。然后在slave上运行change master to进行切换。在新主上运行show master status。然后进行相应的设置。

```
RESET MASTER
change master to master_host='192.168.100.156',
        master_user='repl',
        master_password='123456',
        master_log_file='master log bin',
        master_log_pos=master position number
start slave
```

MYSQL 报错类型

```
[root@DB-Mysql2 ~]# service mysqld start
Starting MySQL.The server quit without updating PID file (/[FAILED]mysql/DB-Mysql2.pid).
```

看看是否指定datadir=mysql/data/

再看看sock文件路径和pid

如果都没问题就初始化

```
[root@localhost ~]# cd /usr/local/mysql
[root@localhost mysql]# chown -R mysql.mysql .
[root@localhost mysql]# su - mysql
[mysql@localhost ~]$ cd /usr/local/mysql
[mysql@localhost mysql]$ scripts/mysql_install_db
[mysql@localhost mysql]$ /usr/local/mysql/bin/mysqld_safe --user=mysql & 后台启动
service mysql start
```

分区操作笔记

表的拆分有两个纬度，表分区只有一个纬度；

表的拆分可以多台机器，表分区只能在一台机器；

表的拆分可以无限，表分区只能1024个分表。

新表分区

```
CREATE TABLE tb_name(
filed.....
)ENGINE=INNODB
PARTITION by RANGE(字段)(
PARTITION p0 VALUES LESS THAN(201002),
PARTITION p1 VALUES LESS THAN(201003),
PARTITION p0 VALUES LESS THAN MAXVALUE
)
```

新增分区

```
mysql> ALTER TABLE sale_data ADD PARTITION (PARTITION p201010 VALUES LESS THAN (201011));
```

删除分区

```
mysql> ALTER TABLE table_name DROP PARTITION p0
```

将p2010Q1 分区，拆分为s2009 与s2010 两个分区

```
mysql> ALTER TABLE sale_data REORGANIZE PARTITION p2010Q1 INTO (
   PARTITION s2009 VALUES LESS THAN (201001),
   PARTITION s2010 VALUES LESS THAN (201004)
);
```

以上介绍的分区都是必须是Int整型才可以分区，如果不是要以DATA或其它函数转换，COLUMNS可以使用字符串

使用COLUMNS分区

```
CREATE TABLE `bbcc` (
 `id` int(11),
 `str` char(11) DEFAULT  NULL)
 partition by  list columns(`str` )(
 partition p0 values in('sxw','sb'),
 partition p1 values in('nb','c')
 )
```

使用带主健的列RANGE分区

```
CREATE TABLE tb_name(
`id` int(11) unsigned NOT NULL AUTO_INCREMENT COMMENT '自增id',
`str` int(11) DEFAULT  0,
`act` int(1) default 0,
PRIMARY KEY (`id`,`str`)
)ENGINE=INNODB
PARTITION by  RANGE(`str`)(
PARTITION p0 VALUES LESS THAN(3),
PARTITION p1 VALUES LESS THAN(5),
PARTITION p2 VALUES LESS THAN MAXVALUE
)
```