面试总结大纲

--林银展

2020.12.08

公司：保密

1 面试问题（公司保密）

1.1 什么是回表，二级索引什么情况下不用回表?

```
二级索引查询找到聚簇索引对应的值或者范围，回原表查询，叫回表
当查询列是索引列时，可以不用回表
```

数据库内存分配情况（以128G为实际生产内存总量）

1.2 单单从内存使用机制这一块，如何优化?

数据库内存分配情况（以128G为实际生产内存总量）

```
1. buffer pool 64G-96G都合适，不要说百分比，人家问的是生产分配，不是让你谈理论。
2. redo buffer 分配 2-4G 都可以
3. 4个多缓冲区如何分配
    read_buffer_size  --- 对表进行顺序扫描的请求将分配一个读入缓冲区
    read_rnd_buffer_size  --- mrr再排序二级索引查询结果时，使用到的缓存
    join_buffer_size  --- 执行join语句时，缓存索引分片
    sort_buffer_size  --- 排序缓存空间，对执行结果进行排序
# 答题时，需把握“充分测试”这一点，像内存变更这种不可谓不是牵一发而动全身的变更，它涉及的范围及其影响都相当重大的。
# 还有一些细节部分可以完善，可以关注郭导专家班
```

1.3 Zabbix如何监控mysql主机的参数

```
内存使用量到达多少报警？
例如：
CPU 60%~70%警告，80%发邮件
MEM 60%~70%警告，80%发邮件
IOPS 30000/s警告（raid10，这是大概多少sas盘才能到达，这个需要思考）

当报警时如果内存使用量过高，如何排查原因，遇到如何解决，有没有实际案例？
1.  内存报警主要出现在什么时间段？
2.  业务高峰期是否存在慢语句，尤其是不走索引的慢语句
3.  查看报警时间段的binlog日志，确定这段时间做了什么事情，可能导致内存使用率高
4.  关注错误日志警告信息
5.  统计频繁程度，处理前统计，处理后也得统计确定处理效果。
```

1.4 业务库350G左右，每日产生的binlog大概多大?

业务库的数量级和binlog的大小没有直接关系，业务库是记录的数据量多少，不完全体现数据变化，假如有大量的update操作，那么这些操作会在binlog中反馈，但是对于业务数据库可能只是修改了某个值，可能没增没减。
小公司每日binlog量有100~200M，200M多一些差不多
中型公司有400~500M合理
大型公司可以上10G。

1.5 你们公司数据库服务器的QPS，TPS大概是多少呢?

什么是QPS,TPS?

方法一：**基于 questions 计算qps,基于 com_commit com_rollback 计算tps**

QPS：全名 Queries Per Second，意思是“每秒查询率”，是一台服务器每秒能够响应的查询次数，是对一个特定的查询服务器在规定时间内所处理流量多少的衡量标准。

qps = Questions / Uptime

获取方式：

```
show global status like "questions";
show global status like 'uptime';
```

eg：通过查看show global status得到Questions为2417815，Uptime为364，则：

```
qps = 2417815 / 364 = 6642.34890109890109890110
```

```
# 注意到mysql的status里面还有个Queries: 2417824，比Questions大一点，这两个的区别：
```

Queries：这个状态变量表示，mysql系统接收的查询的次数，包括存储过程内部的查询

Questions：这个状态变量表示，mysql系统接收查询的次数，但是不包括存储过程内部的查询，所以用Questions而不是Queries

TPS： Transactions Per Second 的缩写，每秒处理的事务数目。一个事务是指一个客户机向服务器发送请求然后服务器做出反应的过程。

tps = (Com_commit + Com_rollback) / Uptime

其中Com_commit是已提交事务数目，Com_rollback是回滚的事务数目，Uptime是运行时间。

获取方式：

```
show global status like "Com_commit";
show global status like 'Com_rollback';
show global status like 'uptime';
```

eg：通过查看show global status得到commit为132842， rollback为435， Uptime为364，则：

```
tps = (132842 + 435) / 364 = 366.14560439560439560440
```

面试官问题：

一台32c128g的服务器压测的时候 一般 tps 在200-500之间 qps 在4000-7000之间，具体看磁盘阵列或者固态盘还是机械盘，监控能监控qps和tps。（郭导专家班教你如何烤鸡，考完你就懂）

方法二：**基于 com_* 的status 变量计算tps ,qps**

```
show global status where variable_name
in('com_select','com_insert','com_delete','com_update');
获取间隔1s 的 com_*的值，并作差值运算
    del_diff = (int(mystat2['com_delete'])   - int(mystat1['com_delete']) ) /
diff
    ins_diff = (int(mystat2['com_insert'])   - int(mystat1['com_insert']) ) /
diff
    sel_diff = (int(mystat2['com_select'])   - int(mystat1['com_select']) ) /
diff
    upd_diff = (int(mystat2['com_update'])   - int(mystat1['com_update']) ) /
diff
tps = Com_insert/s + Com_update/s + Com_delete/s
qps = Com_select/s + Com_insert/s + Com_update/s + Com_delete/s
```

Questions 是记录了从mysqld启动以来所有的select，dml 次数包括show 命令的查询的次数。这样多少有失准确性，比如很多数据库有监控系统在运行，每5秒对数据库进行一次show 查询来获取当前数据库的状态，而这些查询就被记录到QPS,TPS统计中，造成一定的"数据污染".

如果数据库中存在比较多的myisam表，则计算还是questions 比较合适，**如果数据库中存在比较多的innodb表，则计算以com_*数据来源比较合适。**

2 个人问题

1. 语言不够干净利索，重复太多
2. 思路有些紊乱
3. 过于急躁，抢了面试官的话（极大忌讳，感谢童哥当面指正）

2020.12.13

1 非本人面试(华为外包)

1.1 聚簇索引如何组织生成表

```
# 重点是B+树结构
1. 叶节点：存储数据行时是有序排列的，然后将数据行的 PAGE 作为叶节点（相邻叶节点，有双向指针）
2. 枝节点：提取叶节点的范围+指针，构建枝节点（相邻枝节点，有双向指针）
3. 根节点：提取枝节点的范围+指针，构建根节点
# 扩展：聚簇索引建立标准
1. 如果表中设置了主键（例如 ID 列），生成表时，会自动根据 ID 列生成索引树；
2. 如果没有设置主键，自动选择第一个不为空的唯一键（unique key）的列作为聚簇索引；
3. 如果以上2点都不成立，自动生成 6字节 隐藏聚簇索引。
```

1.2 离散插入，导致叶子节点重新组织数据，问？这个是如何重新组织的。

这是页分裂的情况，出现页分裂，InnoDB引擎是将产生分裂的页，会按照原页面中50%的数据量进行分裂，然后创建新页，存放被分离的数据，接着将新行插入对应页，更改双向指针指向。譬如
page#5 （23，24，25，26） page已满
page#6 （28，29，30，31） page已满
1.此时，增加id=27的行，由于page已经满了，导致页分裂，分裂过程如下
page#5 门限值（MERGE_THRESHOLD）50%处分裂，page#5保留（23，24），新建page#7，此时分裂出去的25，26填入page#7，此时page#7（25，26），新旧页都有了空间继续插入值。所以按照排序，27应填入page#7，此时page存储情况如下：
page#5 （23，24）
page#7 （25，26，27）
page#6 （28，29，30，31）
2.最后，双向指针变相，page#5--> page#7 --> page#6。分裂页流程完整流程就是这样。
3.页分裂导致的问题：分裂出来的页不是顺序页，大概率是不会在一个簇当中，虽然通过双向指针联系在一起，但是分裂页已经不是顺序IO了，双向指针对分裂页的优化比较有限了。
参考博文：https://blog.csdn.net/weixin_42261489/article/details/108475250

页分裂最好的解决方案就是重建表，可以收回所有碎片，或者alter table xxx engine=innodb;执行一次有一定作用，连续执行也不会更好了。

1.3 MyISAM回行问题?

郭导原话：您公司的还用MyISAM的表存储数据吗？

假如面试官说“没有”，顺着他说，“是啊，MyISAM在被逐渐摒弃，8.0之后，系统表都没有MyISAM的表了，我个人更愿意将学习一个逐渐淘汰的存储引擎的时间，放在主流的存储引擎，比如InnoDB。”（其实我个人心里是这么想的，没有你问个屁！）

假如面试官说“有”，给他提优化建议，将MyISAM引擎更改为InnoDB引擎，回到 Innodb 引擎优势上。
MVCC
支持事务
支持热备
支持CR\DWB
支持外键
支持多缓存
行级锁粒度
等等，啥熟挑啥说，说完顺便给它批量更改MyISAM表为InnoDB表的拼接语句。

```
select concat("alter table ",table_schema,".",table_name," engine=innodb;")
from information_schema.tables where table_schema not in
('mysql','information_schema','performance_schema','sys') and engine != 'innodb'
into outfile '/tmp/alter_engine_InnoDB.sql';
source /tmp/alter_engine_InnoDB.sql
```

1.4 隔离级别

```
# 基础得我不想写，但既然面到了，还是得提一提
# 重点是锁的讲解，即原理分析
RU 幻读、脏读、不可重复读
RC 不可重复读，幻读（record lock）
RR 可重复读，很小概率会发生幻读（record lock、gap lock、next lock）
SR 事务串行化执行，说白了就是没法并发
# 可扩展方向
MVCC下，RC级别和RR级别的区别
```

1.5 备份策略

```
# 备份工具
mysqldump、xtrabackup（XBK）
# 备份策略
一周一备份，每日一个binlog，binlog保留周期为15天

# 原理扩展部分
mysqldump的备份原理
    1. mysqldump是逻辑备份，建议是100G左右的这个量级的数据库，使用这个方式
    2. --master-data=2
        2.1 自动记录备份时的binlog信息（=2 以注释的方式记录）
        2.2 自动锁定所有表，自动解锁（global read lock）
        2.3 因为自动锁表问题，所以最好配合 --single-transaction 参数，减少锁表时间
    3. --single-transaction
        3.1 基于 InnoDB 引擎，开启独立事务，通过全局一致性快照备份表数据，不必锁表备份，可以理解为热备。
        3.2 非 InnoDB 引擎，需要加全局读锁才能备份
        3.3 注意：在备份时，做DDL操作很可能会导致备份失败

XBK的备份原理
    0. 物理备份，其备份效率更多的是取决于磁盘性能。
    1. 对 InnoDB 引擎，支持热备，不锁表。
    2. 对 非InnoDB 引擎，直接全局锁表（FTWRL）备份。拷贝 非InnoDB文件（[包括innodb表的]frm \ myi \ myd ...）。（8.0之后，innodb表的frm文件并入ibd文件，系统库已经没有MyISAM表了）
    3. 问题：
        3.1 xbk只能本地备份，它不认识[mysqld]标签，需要在添加socket文件到[client]
        vim /etc/my.cnf
        ...
        [client]
        socket=/tmp/mysql.sock
        3.2 DDL操作也会让它备份失败，8023之后才默认加上ddl-lock
    4. 恢复时操作注意点：
        4.1 首先，XBK也是基于全局一致性快照进行的备份，它的备份时间点，是快照的时间点，而它备份时产生的数据是落不到实际用户数据文件的，信息存于redo和undo当中，需要做一步prepare的操作，将这部分信息落盘到用户数据文件
        4.2 注意copy-back之后的文件权限，如果是root用户备份的，那么文件权限用户是root，要改回mysql用户
# 此处尚且遗留一个问题，据面试官谈起，XBK备份时，是通过2个线程分别完成备份和记录redu/undo信息的。这2个线程如何工作需要了解透。
```

1.6 说一下，生产中遇到的最难解决的问题。

```
# 你妹的是多没有问题问了，才问这种问题，但也经常问，自由发挥。
```

1.7 水平拆表后，如何保证业务访问到实例

这个问题实际是问中间件如何实现业务访问正常，比如MyCAT如何做到从逻辑库到实例访问的过程，分库分表策略是什么。

面试官带来的扩展问题：

MyCAT分离热点数据，它是怎么一个处理逻辑，使用这个中间件对性能会有什么不利影响，影响多大？在做策略时是如何考量这些不利影响？

2020.12.29

1 面试问题（boss直聘）

1.1 MHA的VIP脑裂问题

以下问题，还没有解决

VIP脑裂的情况分析不了解

MHA的脚本脑裂问题和keepalived脑裂问题相差不大，生产中使用keepalived的概率其实更高。以下是keepalived可能造成脑裂的原因

- 高可用服务器之间的心跳线链路发生故障,导致无法正常通信
- 心跳线坏了(包括断了,老化)
- 网卡及相关驱动坏了,IP配置及冲突问题(网卡直连)
- 心跳线之间连接的设备故障(网卡及交换机)
- 仲裁的机器出问题了(采用仲裁的方案)，priority设置值相同，产生竞争
- 高可用服务器上开启了iptables防火墙阻挠了心跳信息传输
- 高可用服务器上心跳网卡地址等信息配置不正确,导致发送心跳失败
- 其他服务配置不当等原因,如心跳方式不同,心跳广播冲突,软件bug等.

2 个人问题

1. 对于数据的重要性概念不够深
2. 复习不够充分，问题回的不好

2020.12.31

1 面试问题(平安)

1.1 CPU突然高达800%会是什么原因造成的，32核为例?

最可能是锁的问题
1、top -Hp 进程pid
    可以找到打满CPU的线程是哪个，对应的pid就是performance_schema.threads表中的thread_os_id，也就能找processlist_id，通过 show processlist 找到对应的执行用户窗口，找到相应用户信息，还可以通过performance_schema.events_statements_history 表找到对应执行的语句。
2、%Cpu(s): sy us wa
    CPU这3个状态能简单判断锁的情况，只是wait高一般跟IO有关，如果是us高的话，更可能是慢查询导致，如果是sys和wait都高，大几率锁定是锁的问题。

锁排查完整思路：
1.元数据锁排查
    1.1 show processlist; 查看自己的锁类型，案例情况下，当为 Waiting for global read lock
    1.2 select * from performance_schema.metadata_locks;
        ---> PENDING 对象是正在等待元数据锁，在通过库表信息，看谁锁了自己想要的元数据
        ---> GRANTED 对象是已经获取了元数据锁，对应上自己想要的元数据，确定 OWNER_THREAD_ID 的值
        ---> select * from performance_schema.threads where THREAD_ID = num; 确定 PROCESSLIST_ID 的值
    1.3 show processlist; 比对 PROCESSLIST_ID 看对方是谁，正在做什么，在确认可以杀死的话，kill Id

2. 行锁排查
    2.1 确认有没有锁等待：
    show status like 'innodb_row_lock%'
        Innodb_row_lock_current_waits 的值等多少，就有多少个正在锁等待
    select * from information_schema.innodb_trx;
    2.2 查询锁等待详细信息
        select * from sys.innodb_lock_waits;   ---> blocking_pid(锁源的连接线程，假设等于30)
    2.3 通过连接线程找SQL线程
        select * from performance_schema.threads where processlist_id=30; ---> thread_id（假设等于67）
    2.4 通过SQL线程找到 SQL语句
        select thread_id,SQL_TEXT from performance_schema.events_statements_history where thread_id=67;
        show processlist; ---> 找到id=30的用户是谁，拿着语句向对方确认是否可以杀死，释放锁。

1.2 如果CPU和内存使用率都突然暴涨，可能是什么原因，生产遇到过吗？如何解决？在你的理解范围内分析。

可能是缓存失效

redis雪崩、击穿、穿透
    redis雪崩：用户访问量太大，超过了redis缓存能够承载的并发量，缓存服务器意外全部宕机，访问压力全部落回数据库，数据库也撑不住，挂了，重启依然会被新的流量打死，引起了整个架构雪崩式瘫痪。
    redis击穿：存在某些热点key，访问极其频繁，它们同一时间都失效，这时候访问压力就会落到数据库上，导致数据库访问压力迅速暴涨。
    redis穿透：redis作为缓存，查不到时才会访问数据库，数据库 id 是从 1 开始的，假如黑客发过来的请求 id 全部都是负数。这样的话，缓存中不会有，请求每次都“视缓存于无物”，直接查询数据库。这种恶意攻击场景的缓存穿透就会直接把数据库给打死。

1.3 用过针对slowlog的分析工具吗?

```
pt-query-digest
具体使用参考博主：https://www.cnblogs.com/liujingyuan789/p/7281070.html
```

1.4 32C128G数据库服务器，我回答扛大并发时，TPS大概300，QPS大概3500~4000，做raid10有问题吗？面试官认为我的QPS似乎低了。

```
我的回答：
    还好吧，我们的mysql前端是做了缓冲的（redis，mongodb），本身对数据库压力不是很大，而且因为网宿科技的服务器很多，如果服务器都是新的，那开销何其可怕，同时CDN对数据强一致性并没有要求到那么高，也就避免不了数据库服务器用的也是二手的，硬件性能相对差点，不无意外。
# 3500到4000太低，那就答4000到5000。
```

2 个人问题

```
对生产环境关注不够，这是几乎不可弥补的缺陷。
```

2021.01.08

1 1面问题（云和恩墨）

1.1 假如数据库的硬件无差异、网络正常、事务量也不是很大，但是此时主从有相对较大的延时，那么如何排错？请讲明你思路。

```
面试官答案：
0.  首先排除网络、硬件、高并发之后，会引发延迟的大几率发生在SQL语句上
1.  show slave status\G
2.  查看此时relaylog和主库binlog的位置点（或者GTID）差距，同时确认此时回放到位置点（或者GTID）及其相应主库binlog文件
3.  通过 mysqlbinlog 查看这段时间做了什么SQL语句，导致从库延时。可能是有大量的update\delete这种DML操作才导致，可能是不走索引导致，具体的去分析这段时间内的日志，同时慢日志也会有记录这部分SQL语句
4.  查出原因后，索引的问题解决索引，大事务就找到源头，提醒客户注意此处，能的话给出合理建议。这个主从延迟排查的流程一般都是适用的。
```

主从延时计算扩展

计算某个时间点的主从延时情况

```
主从延时计算：（immediate_commit_timestamp - original_commit_timestamp）+ sys_time_difference
mysqlbinlog /data/3306/binlog/mysql-bin.000001 | egrep  '^(# original_commit_timestamp=)|^(# immediate_commit_timestamp)|SET @@SESSION.GTID_NEXT=' > /tmp/delay.test
```

```
# 系统时间差（sys_time_difference）
系统时间差主要出现在主从同步前，主机没有做时间同步，导致主从本来就有时间差

验证1：主滞后于从
step1：设置系统时间差
# 从库
db02 [test]>select now();
+---------------------+
| now()               |
+---------------------+
| 2021-01-19 16:16:03 |
+---------------------+
1 row in set (0.00 sec)


# 主库
db01 [test]>select now();
+---------------------+
| now()               |
+---------------------+
| 2021-01-19 16:15:54 |
+---------------------+
1 row in set (0.00 sec)
# 主库时间滞后于从库9秒


step2：
db02 [test]>CHANGE MASTER TO
MASTER_HOST='10.0.1.51',MASTER_USER='repl',MASTER_PASSWORD='123',MASTER_PORT=330
6,MASTER_AUTO_POSITION=1;
db02 [test]>start slave ;
db02 [test]>show slave status\G
...
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
...
        Seconds_Behind_Master: 0
...
# 当前主库时，Seconds_Behind_Master没有显示延迟
# 从库外部制造高写入操作，主库导入t100w
# 从库
15K+[root@db02 ~]# cat test.sh
#!/bin/bash
while true
do
    rm -f /root/test.io
    touch /root/test.io
    dd of=/root/test.io if=/dev/urandom count=1000 bs=1M
done
15K+[root@db02 ~]# sh test.sh

# 主库
db01 [test]>source /root/t100w.sql;

# 从库不断查询，有出现过的最多延时是5秒
Seconds_Behind_Master: 5
# 从库binlog 截取延时时间戳
```

```
15K+[root@db02 data]# mysqlbinlog /data/3306/binlog/mysql-bin.000001 | egrep
'^(# original_commit_timestamp=)|^(# immediate_commit_timestamp)|SET
@@SESSION.GTID_NEXT=' > /tmp/delay.test
-------------------------
/tmp/delay.test 内容部分截取
SET @@SESSION.GTID_NEXT= '75165f81-5a29-11eb-b64f-000c29482365:46'/*!*/;
# original_commit_timestamp=1611044753973220 (2021-01-19 16:25:53.973220 CST)
# immediate_commit_timestamp=1611044766690155 (2021-01-19 16:26:06.690155 CST)
    original_commit_timestamp 是主库提交时的时间戳（对应主库binlog的
immediate_commit_timestamp）
    immediate_commit_timestamp 是从库提交时的时间戳
    immediate_commit_timestamp - original_commit_timestamp = current_delay_time
--------------------------
15K+[root@db02 data]# grep '^#' /tmp/delay.test | awk '{print $4}' | awk -F.
'{print $1}' | xargs -n2
# 处理的是每个gtid下的主从延时对比，第一列original_commit_timestamp，第二列
immediate_commit_timestamp
16:25:45 16:25:54
...
16:25:53 16:26:06
...
# 每组的时间 8<delay_time<14秒，这是因为主本身就滞后于从库8-9秒，而实际延时最多时是5秒，相加
主库滞后表征出来的delay_time就是8到14.

结论：当主库时间落后于从库时，通过从库binlog计算的延时时间差是大于实际延时的时间差的。

验证2：主库超前于从库
# 主库
db01 [(none)]>select now();
+---------------------+
| now()               |
+---------------------+
| 2021-01-21 15:41:35 |
+---------------------+
# 从库
db02 [(none)]>select now();
+---------------------+
| now()               |
+---------------------+
| 2021-01-21 15:41:26 |
+---------------------+
# 主库超前从库大概9秒

# 清空主从并还原初始化状态
# 主库
db01 [test]>drop database test;
db01 [(none)]>reset master;
# 从库
db02 [(none)]>stop slave;
db02 [(none)]>reset slave all;
db02 [(none)]>reset master;
db02 [(none)]>CHANGE MASTER TO
MASTER_HOST='10.0.1.51',MASTER_USER='repl',MASTER_PASSWORD='123',MASTER_PORT=330
6,MASTER_AUTO_POSITION=1;
db02 [(none)]>start slave ;
db02 [(none)]>show slave status\G

# 从库外部制造高写入操作，主库导入t100w
```

```
# 从库
15K+[root@db02 ~]# sh test.sh
# 主库
use test;
source /root/t100w.sql;
# 从库不停地刷show slave status\G,追踪得到最高延时
Seconds_Behind_Master: 8

15K+[root@db02 binlog]# mysqlbinlog /data/3306/binlog/mysql-bin.000001 | egrep
'^(# original_commit_timestamp=)|^(# immediate_commit_timestamp)|SET
@@SESSION.GTID_NEXT=' > /tmp/delay.test
15K+[root@db02 binlog]# grep '^#' /tmp/delay.test | awk '{print $4}' | awk -F.
'{print $1}' | xargs -n2
16:04:20 16:04:11
...
16:05:37 16:05:35
...
# 每组的时间 1<delay_time<9秒，这是因为主本身就超前于从库大概9秒，而实际延时最多时是8秒，相键
主库超前表征出来的delay_time就是1到9.
结论：当主库时间超前于从库时，通过从库binlog计算的延时时间差是小于实际延时的时间差的。
```

总结:

1. Second_Behind_Master 表征的延时是准确的，不管主从系统时间是否同步
2. 当主库系统时间滞后于从库时，从库binlog表征出来的延时大于实际，且

   immediate_commit_timestamp - original_commit_timestamp > 永远大于0，即永远有延时
3. 当主库系统时间超前于从库时，从库binlog表征出来的延时小于实际。且存在

   immediate_commit_timestamp - original_commit_timestamp < 0 的不合理情况，即主库提交时间点，竟然滞后于从库提交时间点

1.2 客户反映，他们有一张60W的表，但是查询极慢。select * from 要2分钟才能输出结果。请问可能是什么原因?

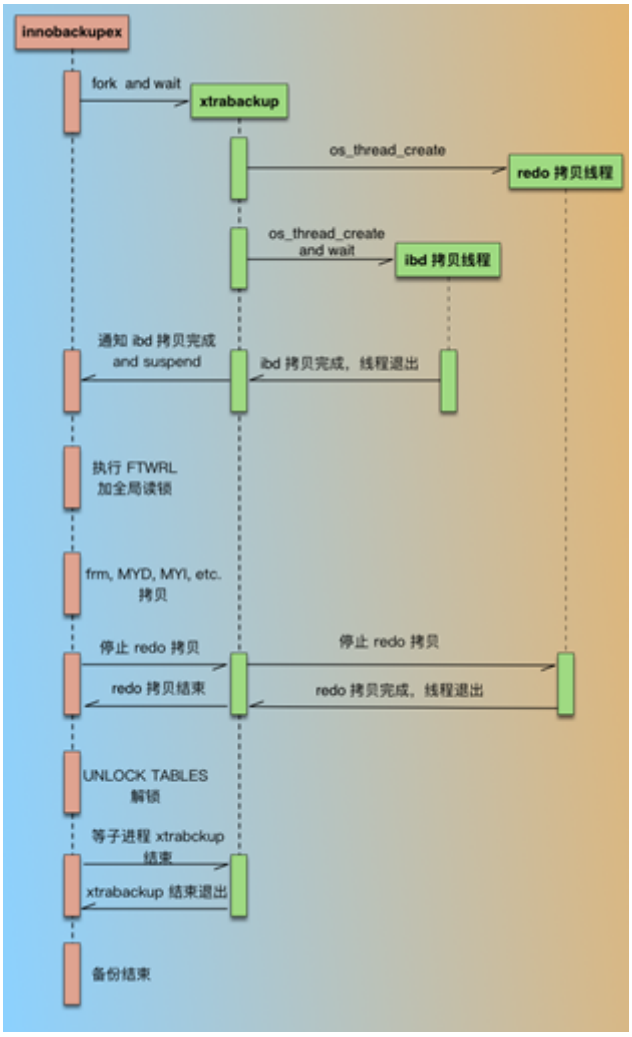
面试官答案：

他们主键列使用uuid列（32位字符串），导致他们的聚簇索引列长度太大，我们知道，即使是二级索引，基本上也要回表做聚簇索引的查询，所以无论怎么查询，都是慢的。

我给的优化方案：

聚簇索引列最好使用无关自增数字列，同时聚簇索引列是表生成后不能更改的，所以我给的建议是重构一张表增加一列无关数字自增列作为主键，而且表数据量级小，导入导出并不复杂，彻底解决聚簇索引的问题。

1.3 XBK具体的备份流程?



1. innobackupex 在启动后，会先 fork 一个进程，启动 xtrabackup 进程，然后就等待 xtrabackup 备份完 ibd 数据文件；
2. xtrabackup 在备份 InnoDB 相关数据时，是有2种线程的，1种是 redo 拷贝线程，负责拷贝 redo 文件，1种是 ibd 拷贝线程，负责拷贝 ibd 文件；redo 拷贝线程只有一个，在 ibd 拷贝线程之前启动，在 ibd 线程结束后结束。xtrabackup 进程开始执行后，先启动 redo 拷贝线程，从最新的 checkpoint 点开始顺序拷贝 redo 日志；然后再启动 ibd 数据拷贝线程，在 xtrabackup 拷贝 ibd 过程中，innobackupex 进程一直处于等待状态（等待文件被创建）。
3. xtrabackup 拷贝完成idb后，通知 innobackupex（通过创建文件），同时自己进入等待（redo 线程仍然继续拷贝）；
4. innobackupex 收到 xtrabackup 通知后，执行FLUSH TABLES WITH READ LOCK (FTWRL)，取得一致性位点，然后开始备份非 InnoDB 文件（包括 frm、MYD、MYI、CSV、opt、par等）。拷贝非 InnoDB 文件过程中，因为数据库处于全局只读状态，如果在业务的主库备份的话，要特别小心，非 InnoDB 表（主要是MyISAM）比较多的话整库只读时间就会比较长，这个影响一定要评估到。
5. 当 innobackupex 拷贝完所有非 InnoDB 表文件后，通知 xtrabackup（通过删文件） ，同时自己进入等待（等待另一个文件被创建）；
6. xtrabackup 收到 innobackupex 备份完非 InnoDB 通知后，就停止 redo 拷贝线程，然后通知 innobackupex redo log 拷贝完成（通过创建文件）；
7. innobackupex 收到 redo 备份完成通知后，就开始解锁，执行 UNLOCK TABLES；
8. 最后 innobackupex 和 xtrabackup 进程各自完成收尾工作，如资源的释放、写备份元数据信息等，innobackupex 等待 xtrabackup 子进程结束后退出。

增量备份
PXB 是支持增量备份的，但是只能对 InnoDB 做增量，InnoDB 每个 page 有个 LSN 号，LSN 是全局递增的，page 被更改时会记录当前的 LSN 号，page中的 LSN 越大，说明当前page越新（最近被更新）。每次备份会记录当前备份到的LSN（xtrabackup_checkpoints 文件中），增量备份就是只拷贝LSN大于上次备份的page，比上次备份小的跳过，每个 ibd 文件最终备份出来的是增量 delta 文件。
MyISAM 是没有增量的机制的，每次增量备份都是全部拷贝的。
增量备份过程和全量备份一样，只是在 ibd 文件拷贝上有不同。

恢复过程
如果看恢复备份集的日志，会发现和 mysqld 启动时非常相似，其实备份集的恢复就是类似 mysqld crash后，做一次 crash recover。
恢复的目的是把备份集中的数据恢复到一个一致性位点，所谓一致就是指原数据库某一时间点各引擎数据的状态，比如 MyISAM 中的数据对应的是 15:00 时间点的，InnoDB 中的数据对应的是 15:20 的，这种状态的数据就是不一致的。PXB 备份集对应的一致点，就是备份时FTWRL的时间点，恢复出来的数据，就对应原数据库FTWRL时的状态。

因为备份时 FTWRL 后，数据库是处于只读的，非 InnoDB 数据是在持有全局读锁情况下拷贝的，所以非 InnoDB 数据本身就对应 FTWRL 时间点；InnoDB 的 ibd 文件拷贝是在 FTWRL 前做的，拷贝出来的不同 ibd 文件最后更新时间点是不一样的，这种状态的 ibd 文件是不能直接用的，但是 redo log 是从备份开始一直持续拷贝的，最后的 redo 日志点是在持有 FTWRL 后取得的，所以最终通过 redo 应用后的 ibd 数据时间点也是和 FTWRL 一致的。
所以恢复过程只涉及 InnoDB 文件的恢复，非 InnoDB 数据是不动的。备份恢复完成后，就可以把数据文件拷贝到对应的目录，然后通过mysqld来启动了。

1.4 如何处理redis的bigkey，bigkey会对redis产生什么影响?

1. redis是单线程，bigkey会占据线程，影响它的性能
2. bigkey要做拆分，避免bigkey存在

1.5 内存泄漏处理?

面试官处理方法：MySQL 5.6及之前的版本，存在比较多的内存泄露的问题，MySQL 5.7版本之后优化了很多，遇到严重的内存泄漏，没办法只能重启解决。

2 2面问题(云和恩墨)

（问的贼细贼刁钻，干崩了我的心态）

2.1 讲一讲MySQL体系结构，并以配合开发的角度将体系架构当中的注意点，给予点出

1. sql_mode注意点
   only_full_group_by
2. 统计信息表异步更新带来的问题
3. 执行分析，找出语句问题，评估并优化
   explain/desc
   MySQL profiler（自行了解）
   optimizer（自行了解）
   （还有一个我忘了）

profiling参数

```
# The SHOW PROFILE and SHOW PROFILES statements are deprecated; expect it to be removed in a future MySQL release.
profiling是一个被官方逐渐弃用的工具

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
}

# 默认是off，需要开启分析器
mysql> SET profiling = 1;

# 详细使用查看
https://dev.mysql.com/doc/refman/8.0/en/show-profile.html

# profile工具的缺陷
Profiling is only partially functional on some architectures. For values that
depend on the getrusage() system call, NULL is returned on systems such as
Windows that do not support the call. In addition, profiling is per process and
not per thread. This means that activity on threads within the server other than
your own may affect the timing information that you see.
在体系结构中，性能分析只是部分有用的。对于需要依赖getrusage()函数进行系统调用的值而言，NULL存在不会被系统返回情况，比如不支持调用getrusage()函数的Windows系统。此外，性能分析器针对于每一个进程但并非是线程，这意味着服务中活跃的线程，除了你自己的这个进行下的线程，其他线程也可能影响这个你能看到性能分析的时间。（分析时间可能不够准确的原因）
```

2.2 讲一讲MyCAT的优缺点，分库分表会造成哪些影响?

```
    MyCAT和ProxySQL都存在一个同样的问题，他们需要先给语句做一次解析，然后再分发到下级数据库，再做一次解析，比起无中间件的情况下，语句解析上只有原来的50%效果，这对数据库是有影响的，在做MyCAT分库分表策略时，这点是需要考虑的缺点。
```

2.3 讲一讲proxysql的优缺点，proxysql能给100个节点同时分发流量吗，它是怎么工作的?

```
面试官：
    proxysql做读写分离策略也是有自己的瓶颈的，它依然需要去解析语句，才能分配到各组去，所以它需要消耗一定资源在解析语句上，这注定它必然有自己的瓶颈，下级带个3、5个节点就已经差不多了。（我自己在思考时，类比了负载均衡的四层分发和七层分发，就能理解即使只是简单的解析，在高并发环境下，也会成为瓶颈）
    同时proxysql也有一些缺点，假如开启一个事务的语句是先执行的DQL，它会分配到读组当中，那么后续的DML怎么办呢？这种问题怎么解决？更多的内容自行了解。
```

2.4 你了解mysqlrouter吗?

```
# 自行了解
```

2.5 MHA的vip存在失效问题，即使手动配，也可能配不上去，你知道什么原因吗?

面试官：不仅仅是MHA的VIP漂移脚本，keepalived也可能存在这种问题，因为脚本当中存在arping这么一个命令，这命令可能会清空了MAC地址，那无论你怎么配，都是配不上去的。更多的内容自行了解。

2021.01.14

1 面试问题（拉勾网）

1.1 如果建字段使用了text类型，问：text有什么优缺点，如果你觉得这个类型不适合，如何给开发建议，如何优化?

test类型：本质上mysql提供了两种文本类型
Text：存储普通的字符文本
Blob：存储二进制文本（图片，文件），一般都不会使用blob来存储文件本身，通常是使用一个链接来指向对应的文件本身。
Text：系统中提供的四种text
    Tinytext：系统使用一个字节来保存，实际能够存储的数据为：2^8 + 1
    Text：使用两个字节保存，实际存储为：2^16 + 2
    Mediumtext：使用三个字节保存，实际存储为：2^24 + 3
    Longtext：使用四个字节保存，实际存储为：2^32 + 4
注意：
1、 在选择对应的存储文本的时候，不用刻意去选择text类型，系统会自动根据存储的数据长度来选择合适的文本类型。
2、 在选择字符存储的时候，如果数据超过255个字符，那么一定选择text存储。

具体改造建议参看：https://blog.csdn.net/qq_35190492/article/details/109569211
Text改造建议
使用es存储
在MySQL中，一般log表会存储text类型保存request或response类的数据，用于接口调用失败时去手动排查问题，使用频繁的很低。可以考虑写入本地log file，通过filebeat抽取到es中，按天索引，根据数据保留策略进行清理。

使用对象存储
有些业务场景表用到TEXT，BLOB类型，存储的一些图片信息，比如商品的图片，更新频率比较低，可以考虑使用对象存储，例如阿里云的OSS，AWS的S3都可以，能够方便且高效的实现这类需求。

总结
由于MySQL是单进程多线程模型，一个SQL语句无法利用多个cpu core去执行，这也就决定了MySQL比较适合OLTP（特点：大量用户访问、逻辑读，索引扫描，返回少量数据，SQL简单）业务系统，同时要针对MySQL去制定一些建模规范和开发规范，尽量避免使用Text类型，它不但消耗大量的网络和IO带宽，同时在该表上的DML操作都会变得很慢。

1.2 varchar(10)和varchar(255)，有什么区别（至少有2个）？()内是字节还是字符?

```
1. 能存储的最大字符数肯定不一样。。。
2. 记录字符串长度的位数可能不一样，大于255字节，varchar括号内的数字指的是字符输，假设使用的是utfmb4字符集，那么varchar(255)字符长度可以超过255个字节，那么记录字符串长度，就需要花费2个字节记录，而varchar(10)，一定是1个字节即可。
# 还有吗？
```

1.3 你做过什么线上的优化？如何优化，优化依据是什么？

```
# 有没有实例说明
```

1.4 讲述一下 update/insert 语句，完整的执行过程（MySQL体系架构，select例子已经满足不了面试了）

```

```

1.5 linux系统如何查询进程io占用情况

```
pidstat -d 1
    https://www.cnblogs.com/wx170119/p/11411312.html
iotop -oP

top -Hp 进程号
```

1.6 binlog format的格式，以及他们的优缺点（面试官要求连mixed的优缺点也得说，而并不是我一句生产不用就不解释可以过关的）？

据面试官声称，小米使用的就是mixed模式

```

```

2 面试问题（石墨文档）

2.1 知道MongoDB的命名空间吗？

1. MongoDB内部有预分配空间的机制，每个预分配的文件都用0进行填充。

2. 数据文件每新分配一次，它的大小都是上一个数据文件大小的2倍，每个数据文件最大2G。

3. MongoDB每个集合和每个索引都对应一个命名空间，这些命名空间的元数据集中在16M的*.ns文件中，平均每个命名占用约 628 字节，也即整个数据库的命名空间的上限约为24000。

4. 如果每个集合有一个索引（比如默认的_id索引），那么最多可以创建12000个集合。如果索引数更多，则可创建的集合数就更少了。同时，如果集合数太多，一些操作也会变慢。

5. 要建立更多的集合的话，MongoDB 也是支持的，只需要在启动时加上“ --nssize ”参数，这样对应数据库的命名空间文件就可以变得更大以便保存更多的命名。这个命名空间文件（.ns文件）最大可以为 2G。

6. 每个命名空间对应的盘区不一定是连续的。与数据文件增长相同，每个命名空间对应的盘区大小都是随分配次数不断增长的。目的是为了平衡命名空间浪费的空间与保持一个命名空间数据的连续性。

7. 需要注意的一个命名空间 $freelist ，这个命名空间用于记录不再使用的盘区（被删除的Collection或索引）。每当命名空间需要分配新盘区时，会先查看 $freelist 是否有大小合适的盘区可以使用，如果有就回收空闲的磁盘空间。

面试官经典语录：即使学十年只能用上一次，你也要保证这次100%的成功。

2.2 update ... limits 10，这个语句在主库执行和从库回放时，会不会引起主从不一致，会有什么不一样的地方呢？或者说，主库select * limit 10 和 从库select * limit 10 会完全一样吗？

可能会引起主从不一致，无论是update还是select，如果通过的是扫描全表方式（无查询条件、查询条件不走索引等）获取数据的，那么获取的是将是一个无序随机的结果，虽然结果集是一样的，但是limit截取结果集的前10行，这10行具有随机性，主库从库在不同实例中，是不能排除前10行随机结果是不一样的，如果是update语句，出现这种情况时，不可避免的要出现的主从不一致。所以主从环境下，limit通常要配合order by，去截取排序后的结果，那么这就不会有问题了，或者查询条件能正确走向索引（其实建立时索引已经是排序过的，本质还是排序）。
下图是我们进行一次id<=5的先删后增的limit 10/limit 10 offset 999990，虽然id（这个id无索引）顺序上还是1到100w但是实际上limit不是顺序的前10行，可见如果不排序，使用limit，即使主从表内数据完全一致，也可能存在彼此之间是乱序，一旦更新或者删除，未来可能出现主从不一致。
建议：

1. 使用索引，索引本身就有排序再生成
2. 与order by搭配

```
db01 [(none)]>select *from test.t100w limit 10;
+------+--------+------+------+---------------------+
| id   | num    | k1   | k2   | dt                  |
+------+--------+------+------+---------------------+
|    6 | 584039 | QJ   | VWlm | 2019-08-12 11:41:16 |
|    7 | 541486 | vc   | ijKL | 2019-08-12 11:41:16 |
|    8 | 771751 | 47   | ghLM | 2019-08-12 11:41:16 |
|    9 | 752847 | aQ   | CDno | 2019-08-12 11:41:16 |
|   10 | 913759 | ej   | EFfg | 2019-08-12 11:41:16 |
|   11 | 854170 | sW   | bcWX | 2019-08-12 11:41:16 |
|   12 | 857349 | 25   | 89kl | 2019-08-12 11:41:16 |
|   13 |  27778 | Vs   | mnij | 2019-08-12 11:41:16 |
|   14 | 636589 | TM   | 34PQ | 2019-08-12 11:41:16 |
|   15 | 220092 | j2   | 78op | 2019-08-12 11:41:16 |
+------+--------+------+------+---------------------+
10 rows in set (0.00 sec)

db01 [(none)]>select *from test.t100w limit 10 offset 999
+---------+--------+------+------+---------------------+
| id      | num    | k1   | k2   | dt                  |
+---------+--------+------+------+---------------------+
|  999996 | 202946 | DL   | LMmn | 2019-08-12 11:53:17 |
|  999997 | 798547 | yx   | UVLM | 2019-08-12 11:53:17 |
|  999998 | 540949 | jv   | stEF | 2019-08-12 11:53:17 |
|  999999 | 853903 | nD   | VWAB | 2019-08-12 11:53:17 |
| 1000000 |  57542 | 5l   | fg01 | 2019-08-12 11:53:17 |
|       1 |  25503 | 0M   | IJ56 | 2019-08-12 11:41:16 |
|       2 | 756660 | rx   | bc67 | 2019-08-12 11:41:16 |
|       3 | 876710 | 2m   | tu67 | 2019-08-12 11:41:16 |
|       4 | 279106 | E0   | VWtu | 2019-08-12 11:41:16 |
|       5 | 641631 | At   | rsEF | 2019-08-12 11:41:16 |
+---------+--------+------+------+---------------------+
```

2.3 innodb_thread_concurrency参数的作用

故障案例：我们业务里存在频繁的批量update，但业务量级不会很大，不过密集的时候，他会将数据库打满，而且我们许多业务共用同一个数据库实例，这种密集服务可能只是存在一个无关紧要的服务里，但是他把数据库给打满了，CPU被占满了，影响到了核心业务。

innodb_thread_concurrency 参数可以规避这种事情，为什么?

实际是提高了并发度来解决的
阅读顺序
https://www.jianshu.com/p/5dbb5e095c9a/
https://blog.csdn.net/weixin_33963189/article/details/90249754/
http://blog.itpub.net/7728585/viewspace-2658990/

（1）简介
此参数用来设置innodb线程的并发数量，默认值为0表示不限制。
（2）配置依据

在官方doc上，对于innodb_thread_concurrency的使用，也给出了一些建议，如下：
如果一个工作负载中，并发用户线程的数量小于64，建议设置innodb_thread_concurrency=0；
如果工作负载一直较为严重甚至偶尔达到顶峰，建议先设置innodb_thread_concurrency=128，
并通过不断的降低这个参数，96，80，64等等，直到发现能够提供最佳性能的线程数，
例如，假设系统通常有40到50个用户，但定期的数量增加至60，70，甚至200。你会发现，
性能在80个并发用户设置时表现稳定，如果高于这个数，性能反而下降。在这种情况下，
建议设置innodb_thread_concurrency参数为80，以避免影响性能。
如果你不希望InnoDB使用的虚拟CPU数量比用户线程使用的虚拟CPU更多（比如20个虚拟CPU），
建议通过设置innodb_thread_concurrency 参数为这个值（也可能更低，这取决于性能体现），
如果你的目标是将MySQL与其他应用隔离，你可以l考虑绑定mysqld进程到专有的虚拟CPU。
但是需 要注意的是，这种绑定，在myslqd进程一直不是很忙的情况下，可能会导致非最优的硬件使用率。在这种情况下，
你可能会设置mysqld进程绑定的虚拟 CPU，允许其他应用程序使用虚拟CPU的一部分或全部。
在某些情况下，最佳的innodb_thread_concurrency参数设置可以比虚拟CPU的数量小。
定期检测和分析系统，负载量、用户数或者工作环境的改变可能都需要对innodb_thread_concurrency参数的设置进行调整。

128   -----> top cpu
设置标准：
1、当前系统cpu使用情况，均不均匀
top

2、当前的连接数，有没有达到顶峰
show status like 'threads_%';
show processlist;
（3）配置方法：
innodb_thread_concurrency=8
方法：
    1. 看top ,观察每个cpu的各自的负载情况
    2. 发现不平均,先设置参数为cpu个数,然后不断增加(一倍)这个数值
    3. 一直观察top状态,直到达到比较均匀时,说明已经到位了.

2.4 mysql-tree（format=json）展示执行分析计划？（再次提到profile工具，更完整的做分析）

```
explain format=json  select * from test.t100w where id=10;
explain ANALYZE  select * from test.t100w where id=10;
```

2.5 RDB和AOF的区别?

RDB：类似于快照，当前内存里的数据的状态持久化到硬盘
优点：压缩格式/恢复速度快
缺点：不是实时的，可能会丢数据，操作比较重量

AOF：类似于mysql的binlog，可以设置成每秒/每次操作都以追加的形式保存在日志文件中
优点：安全，最多只损失1秒的数据，具备一定的可读性
缺点：文件比较大，恢复速度慢

RDB持久化配置
```
save 900 1
save 300 10
save 60 10000
```

AOF持久化配置
```
appendonly yes
```

```
appendfilename "redis.aof"
appendfsync everysec

# 当AOF和RDB同时存在的时候，redis会优先读取AOF的内容
```

2.6 binlog2sql使用要注意什么?

```
# 这个问题，我没有答上。面试官并没有直接回答，而是劝我，要真正了解一个工具之后才去使用这个工具，避免在使用时因为不够了解工具，而出现更大的灾难，确保操作的100%成功。
1. 不能看见DDL的操作过程；
2. 不能看见删表删库的操作过程；
3. 8.0会发生行顺序混乱，譬如：查询 update 语句 变更 1<=id<=6 的行，它查询的结果排序可能混乱，原本是行序号是1234 5 6，它偏偏有可能显示1234 6 5 这种，那么在复制语句时就要注意对应正确的id值。
```

3 面试问题（百度）

3.1 做运维时，系统优化方面你做了哪些?

```
1. 关闭numa
2. 大页内存
3. 更改文件句柄，优化tcp参数、swap分区使用临界点
4. io调度策略
等等
```

3.2 数据库的监控指标，你们设置了哪些?

```
# 如果问如何实现，那就很难回答了，毕竟没有实际经验
内存、事务、线程、QPS、TPS、锁、锁等待、参数的评估指标。
```

3.3 SQL语句很复杂，甚至有子查询，假设分析编号是1、1、2，你怎么看出他的执行计划、执行流程?（第三次遇到提到profile工具的面试）

```
profile工具、mysql_tree
```

面试官对于面试者也是不熟悉的，百度这个面试官更多的是在测试我的沟通能力上，设置题干并没有多少原理的东西，但是他个人面试时，碰到过很多难以沟通的面试者，他的第一个问题是问我公司的自动化部署流程，**考核的就是沟通协调能力。**

4 个人总结

```
我给自己的复习时间太短暂了，有些名词说完就忘，尤其是nosql部分。
```

2012.1.15

1 面试问题（宁德时代新能源）

1.1 如果数据库hang死是 redo log 没办法写入，那会是什么原因导致的?

redo log buffer此时已经满了，触发CKPT，但是还是并发太大，脏页太多，刷写占据了IO，io还是被占满，redo log buffer还是无法刷写到日志，此时就会引起数据库hang死。

1.2 gtid同步中断如何解决？（实际也是从库误写入的案例，才造成这种结果）

```
# 从库执行
mysql> create database oldboy;

#主库执行，出现报错，但是不会影响主库创建oldboy库
mysql> create database oldboy;
2021-01-18T15:18:00.931789Z 10 [ERROR] [MY-010584] [Repl] Slave SQL for channel
'': Error 'Can't create database 'oldboy'; database exists' on query. Default
database: 'oldboy'. Query: 'create database oldboy', Error_code: MY-001007
2021-01-18T15:18:00.931831Z 10 [Warning] [MY-010584] [Repl] Slave: Can't create
database 'oldboy'; database exists Error_code: MY-001007
2021-01-18T15:18:00.931856Z 10 [ERROR] [MY-010586] [Repl] Error running query,
slave SQL thread aborted. Fix the problem, and restart the slave SQL thread with
"SLAVE START". We stopped at log 'mysql-bin.000001' position 673

#从库报错，指示SQL报错，停止复制。
show status slave\G
Last_SQL_Error: Error 'Can't create database 'oldboy'; database exists on query.
Default database: 'oldboy'. Query: 'create database oldboy'
...
           Retrieved_Gtid_Set: 087d35ae-599b-11eb-a72c-000c29482365:1-3
            Executed_Gtid_Set: 087d35ae-599b-11eb-a72c-000c29482365:1-2,
132e9a11-599b-11eb-a347-000c29482365:1      # 这是从库误写入的
...

# 注入空事物的方法：
stop slave;
set gtid_next='087d35ae-599b-11eb-a72c-000c29482365:3';
begin;commit;
set gtid_next='AUTOMATIC';

这里的087d35ae-599b-11eb-a72c-000c29482365:3 也就是你的slave sql thread报错的GTID，
或者说是你想要跳过的GTID。
最好的解决方案：重新构建主从环境
```