# 背景

参考：

<https://blog.csdn.net/w892824196/article/details/82591115>

mysql5.6开始支持在线ddl，在线ddl能够提供下面的好处；

1. 提高生产环境的可用性
2. 在ddl执行期间，获得性能和并发性的平衡，可以指定LOCK从句与algorithm从句，lock=exclusize会阻塞整个表的访问，lock=shared会允许查询但不允许dml，lock=none允许查询和dml操作，lock=default或是没有指定，mysql使用最低级别的锁，algorithm指定是拷贝表还是不拷贝表直接内部操作
3. **只对需要的地方做改变，不是创建一个新的临时表**。

之前ddl操作的代价是很昂贵的，许多的alter table语句是创建一个新的，按需要的选项创建的空表，然后拷贝已经存在的行到新表中，在更新插入行的索引，在所有的行被拷贝之后，老的表被删除，拷贝的表被重命名成原来表的名。

在5.5和5.1优化了的create index和drop index避免了表拷贝的行为，这个特色叫快速索引创建，5.6增强了在改变的时候dml还能处理，叫在线ddl。

一些alter语句允许并发的dml，但是仍然需要拷贝表，这些操作的表拷贝要比之前版本的快。

当ddl在改变表的时候，表是否被锁住取决于操作的内部工作方式及alter table的lock从句，在线ddl语句总是等待访问表的事务提交或回滚，因为ddl语句在准备的过程中会要求一个短暂的排他请求。因为要记录并发dml操作产生的改变，并在最后应用这些改变，在线的ddl会花费更长的时间。

要看ddl是否使用了临时表还是内部操作的，可以查看语句执行结果中有多少行收到了影响，如果是0行，那么就没有复制表，如果是非0，那么就是复制了表。

参考：

<https://blog.csdn.net/a82831154/article/details/102355035>

<https://www.cnblogs.com/zengkefu/p/5674945.html>

# 概述

DDL是用于描述数据库中要存储的现实世界实体的语言，例如创建数据库、创建表、添加索引、添加字段等。

根据DDL执行过程中是否允许对表做读写操作，可以分为“不允许读和写”、“只允许读”、“允许读和写”三种场景（“只允许读”场景设定为Offline DDL，“允许读和写”场景设定为Online DDL）。

MySQL 5.6版本开始引入Online DDL功能，并在MySQL5.7版本和8.0版本做了功能扩充。相对于Offline DDL，Online DDL在执行期间不仅允许对表进行读操作，还允许写操作。减少DDL对在线业务的影响，同时在某些特定的DDL场景下，Online DDL还可以减少对磁盘IO的消耗以及提升DDL执行效率。

# 种类

DDL的种类有很多，比较常见的包含：

索引操作

主键操作

列操作

外键操作

表操作

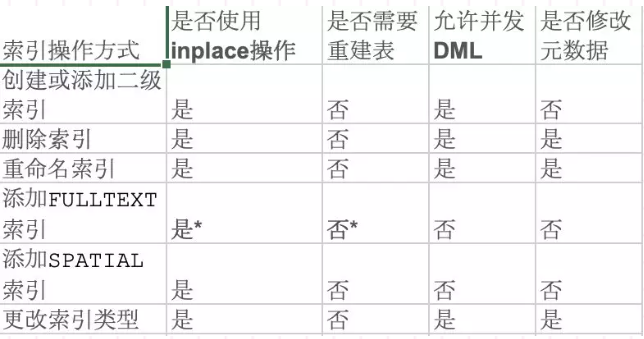
表空间操作

分区操作

每个操作里面又包含了很多种类，比如，索引操作中包含新增索引、删除索引等操作，列操作中有新增列、修改列、删除列等等，这些ddl操作执行过程中的状态究竟是什么样的？我们一一来看。

## 索引DDL操作

可以用下面的表来表示：

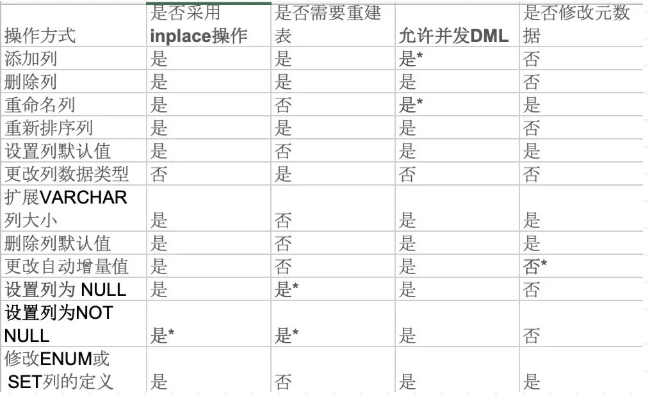


从上面的表中可以看出，创建或者添加二级索引的时候，使用了inplace的操作，不需要重建表，并且允许并发的DML，也就是说，在创建索引的过程中，原表是可读可写的。它数据新增元数据的操作，没有修改数据库的元数据。

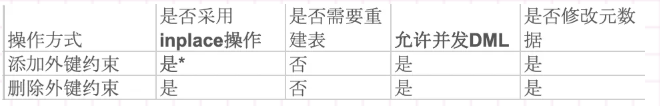
## 主键DDL操作



## 列DDL操作



## 外键操作



## 表操作



# 特点

## 优点

**online ddl操作支持表的本地更改(无需拷贝)和并发DML操作**，一般有以下几个优点：

1、一般的线上环境都是比较忙碌的，想要在一个大表中比较平滑的执行DDL变更几乎不太可能，但是线上的环境又不会接受几分钟的延迟，使用online ddl操作可以尽可能的降低这种影响。

2、online ddl中支持lock语法，lock语法可以微调对表的并发访问程度：

使用lock=none的方法可以开启表的读取和写入，

使用lock=shared方法可以允许对表进行读取，而关闭表的写入功能，

使用lock=exclusive可以禁止对表进行读写，组织并发查询和DML

换句话来说，lock语法可以平衡数据库服务并发和性能之间的竞争，但是需要注意的是：该方法有可能出现失败的情况，如果该方法不可用，该alter table 的操作会立即停止。

3、online ddl中支持algorithm的语法，该参数有两个取值，一个是copy，另外一个是inplace，来看官方文档说明：

COPY：对原始表的副本执行操作，并将表数据从原始表逐行复制到新表。不允许并发DML。

INPLACE：操作避免复制表数据，但可以在适当位置重建表。在操作的准备和执行阶段可以简短地获取表上的独占元数据锁定。通常，支持并发DML。

默认情况下，MySQL5.7使用inplace的方法，而不是copy表结构的方法。因此，与传统的表复制方法相比，online ddl可以降低磁盘上的消耗和IO上的开销。

简单总结，online ddl的3个优点：

a、降低线上变更表的影响时间

b、平衡数据库服务并发性和性能之间的竞争

c、降低磁盘和IO消耗

官方文档中给的常用的在线变更表结构的例子如下：

ALTER TABLEtbl\_name

ADD PRIMARY KEY(column),

ALGORITHM=INPLACE,LOCK=NONE;

## 系统空间

Online DDL对系统空间的要求：

a、如果DDL需要拷贝表数据，则需要额外的空间来保存中间临时表

b、如果DDL执行过程中支持并发DML，则DML操作产生的临时日志文件需要占用额外的系统空间

c、如果DDL执行过程中需要对数据进行排序，则需要额外的系统空间来存储额外的临时排序文件

## 限制

1、使用lock=none模式的时候，不允许有外键约束，如果表中有外键的时候，使用Online DDL会出现一些问题

2、持有元数据锁的其他事务可能导致Online DDL阻塞，Online DDL可能导致其他需要获取元数据锁的事务超时

3、执行Online DDL的执行线程和并行DML不是同一个执行线程，所以并行的DML在执行过程中可能会报错，Duplicate Key的错误

4、optimize table操作会使用重建表的方法来释放聚集索引中未使用的空间，它类似alter table的操作，因为要重建表，它的处理效率不高。

**5、再对大表进行online ddl的操作时，还需要注意以下3点：**

a、没有任何操作能够停止Online DDL操作或者限制该操作过程中IO和磁盘使用率

b、一旦中间发生问题，回滚的代价非常昂贵

c、大表的Online DDL会导致复制出现巨大的延迟，这一点在主从复制架构中需要考虑到

综上所述，在对大表进行Online DDL的时候，有两种方法：

1、使用pt-osc或者gh-ost等在线变更的工具进行变更√

~~2、提前准备好故障报告，直接在线上进行变更，该方法纯属娱乐×~~

# 元数据锁

Offline DDL和Online DDL最重要的区别：DDL执行过程中是否支持对表写擦做，该区别是由DDL执行过程中加不同的元数据锁决定的。

元数据锁是Server层的锁（不是InnoDB存储引擎层面的），主要用于隔离DML和DDL以及 DDL之间的干扰。

## 类型

DDL中的元数据锁：

|  |  |  |
| --- | --- | --- |
| 类型 | 含义 | 作用域 |
| MDL\_EXCLUSIVE | 排它锁，防止其他线程读写元数据 | Offline & Online DDL |
| MDL\_SHARED\_UPGRADABLE | 允许读表数据，允许写表数据，禁止DDL | Offline & Online DDL |
| MDL\_SHARED\_NO\_WRITE | 允许读表数据，禁止写表数据，禁止DDL | Offline |
| MDL\_SHARED\_READ | 读表数据时加的锁 | DML |
| MDL\_SHARED\_WRITE | 写表数据时加的锁 | DML |

**元数据锁之间的关系：**

1. MDL\_EXCLUSIVE和MDL\_SHARED\_READ互斥

事务1拥有表的MDL\_EXCLUSIVE锁，事务2申请MDL\_SHARED\_READ锁时等待

事务1拥有表的MDL\_SHARED\_READ锁，事务2申请MDL\_EXCLUSIVE锁时等待

1. MDL\_EXCLUSIVE和MDL\_SHARED\_WRITE互斥
2. MDL\_SHARED\_UPGRADABLE和MDL\_SHARED\_UPGRADABLE互斥
3. MDL\_SHARED\_UPGRADABLE和MDL\_SHARED\_READ兼容
4. MDL\_SHARED\_READ和MDL\_SHARED\_WRITE兼容
5. MDL\_SHARED\_NO\_WRITE和MDL\_SHARED\_NO\_READ兼容
6. MDL\_SHARED\_NO\_WRITE和MDL\_SHARED\_WRITE互斥

## 锁冲突

开启元数据锁统计功能：

update performance\_schema.setup\_instruments set enabled=’YES’ where name=’wait/lock/metadata/sql/mdl’;

查询当前元数据锁：

select \* from performance\_schema.metadata\_locks;

举例：

事务1：lock table t1 write;

事务2：select \* from t1 limit 1 for update;

先执行事务1，再执行事务2，事务2 处于等待状态，按照以下方法分析元数据锁冲突：

1. 查看当前连接处理情况：

select \* from information\_schema.processlist;

事务2当前状态未“Waiting for table metadata lock”，等待元数据锁。

1. 查看当前元数据锁情况：

select OBJECT\_TYPE,OBJECT\_SCHEMA,OBJECT\_NAME,

OBJECT\_INSTANCE\_BEGIN,

LOCK\_TYPE,LOCK\_STATUS,

OWNER\_THREAD

from performance\_schema.metadata\_locks;

事务1拥有t1表的SHARED\_NO\_WRITE类型元数据

LOCK\_TYPE=SHARED\_NO\_READ\_WRITE 元数据锁类型

LOCK\_STATUS=GRANTED 已拥有元数据锁

OWNER\_THREAD\_IN=336520 处理事务1的线程号

事务2申请SHARED\_WRITE类型元数据锁并处于pending状态

LOCK\_TYPE=SHARED\_WRITE 元数据锁类型

LOCK\_STATUS=PENDING 等待获取元数据锁

OWNER\_THRED\_ID=335454 处理事务2的线程号

1. 确定事务2的SQL语句

select THREAD\_ID,PROCESSLIST\_ID,

PROCESSLIST\_DB,PROCESSLIST\_TIME

from performance\_schema.threads where THREADS\_ID=335454;

PROCESSLIST\_ID =335156与第1步中查询结果处于“Waiting for table metadata lock”连接一致。

select THREAD\_ID,CURRENT\_SCHEMA,SQL\_TEXT

from performance\_schema.events\_statements\_current

where THREADS\_ID=335454;

1. 确定事务1的SQL语句

select THREAD\_ID,PROCESSLIST\_ID,

PROCESSLIST\_DB,PROCESSLIST\_TIME

from performance\_schema.threads where THREADS\_ID=336520;

select THREAD\_ID,CURRENT\_SCHEMA,SQL\_TEXT

from performance\_schema.events\_statements\_current

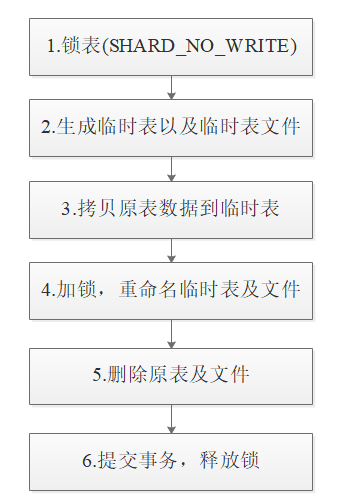
where THREADS\_ID=336520;

从以上查询结果可以判断，事务1阻塞事务2的执行，以及事务1和事务2对应的SQL语句。可以使用kill processlist\_id的方式终止事务1：

kill 336218；

# Offline DDL

## 执行过程



根据元数据锁的关系可知，从第1步加SHARED\_NO\_WRITE元数据锁，一直到第6步释放元数据锁，整个DDL期间，只允许对该表进行查询操作，不允许对该表进行写操作。

## 监控

打开监控DDL进度功能。

表数据文件大小：3G

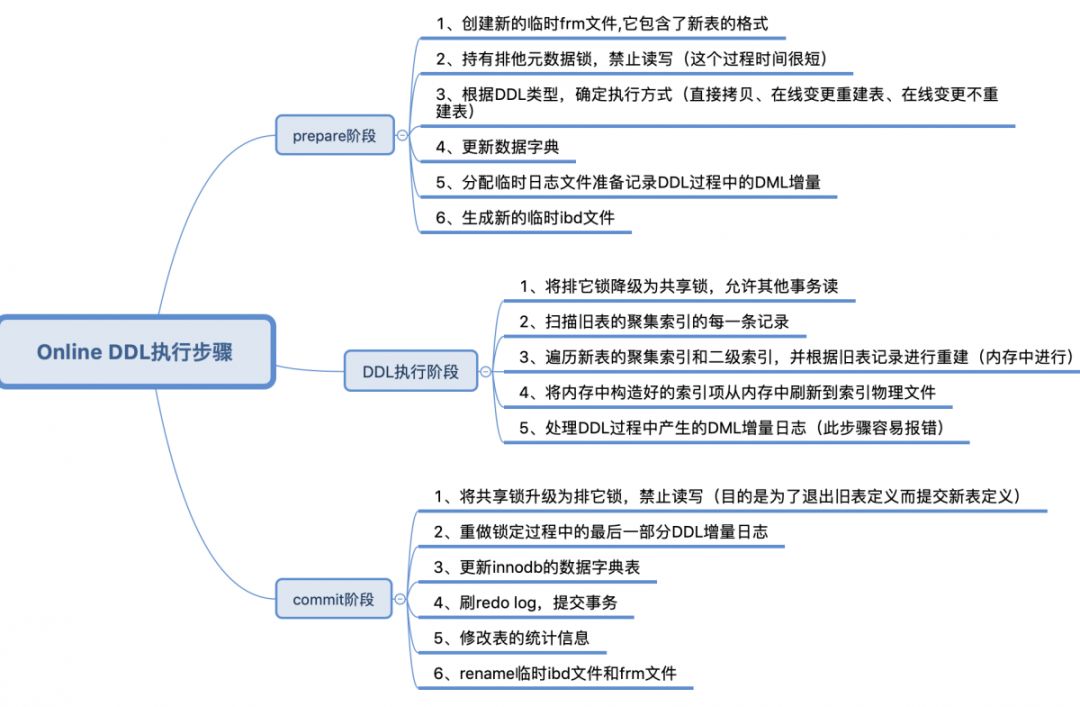
DDL类型：新增字段

1. offline DDL监控信息显示，只有一个阶段copy to tmp table
2. Offline DDL性能比online DDL性能差

分别采用offline DDL和online DDL方式新增字段，offline DDL耗时2min，online DDL耗时38s。

# 过程

online ddl操作的执行过程一般被分为3个阶段，如下：



## 阶段1：初始化阶段（准备阶段）

在初始化阶段，服务器将考虑存储引擎功能，语句中指定的操作以及用户指定的ALGORITHM和LOCK选项，以确定在操作期间允许多少并发。在此阶段，将使用共享的元数据锁来保护当前表定义。

具体步骤：

1、创建新的临时frm文件

2、持有exclusive-mdl锁（MDL\_EXCLUSIVE），禁止DML读写（速度非常快）

3、根据alter类型，确定执行方式，检查存储引擎是否支持inplace，不支持则使用copy---copy

online-rebuild：重新组织表,online-norebuild：改数据字典即可

4、更新数据字典的内存对象

5、分配row\_log对象记录增量（仅rebuild类型需要），记录DDL期间数据修改的日志，如果日志量超过参数 innodb\_online\_alter\_log\_max\_size 的上限，则DDL失败

6、生成临时ibd文件（仅rebuild类型需要)

### ALGORITHM

#### COPY

1、新建带索引(主键索引)的临时表

2、将原表锁定，禁止DML操作，只允许select查询

3、将原表数据拷贝到临时表

4、禁止读写，进行rename，升级字典锁

5、完成创建索引操作

需要记录undo和redo，效率不如inplace，短期占用buffer pool，影响性能

#### INPLACE

1、创建索引(二级索引，主键+普通字段)数据字典

2、加表共享锁S，禁止DML，允许select查询

3、读取聚簇索引，构造新的索引项，排序并插入新索引

4、等待打开当前表的所有只读事务提交

5、创建索引结束

避免重建表带来的IO和CPU消耗，保证DDL期间的性能和并发

#### DEFAULT

1、alter table后面什么都不加的时候，默认是这个方式

2、如果old\_alter\_table为OFF，默认就是inplace方式

3、inplace不支持则进行copy

注意：old\_alter\_table = 0，表示不是使用新建表的方式来建立唯一索引

当一张表的某个字段存在重复值时，这个字段没办法直接加UNIQUE KEY，但是MySQL提供了一个ALTER IGNORE TABLE的方式，可以忽略修改表结构过程中出现的错误，但是要忽略UNIQUE重复值，就需要打开old\_alter\_table，也就是拷贝表的方式来ALTER TABLE。

### 执行方式

根据DDL是否需要重建表空间，可以分为no-build和rebuild两种方式。

#### no-rebuild

no-rebuild不涉及表的重建（例如修改字段名），只修改元数据项（添加索引，会产生二级索引的写入操作），即只在原表路径下产生.frm文件，是代价最小、速度最快的DDL类型。

#### rebuild

rebuild涉及表的重建（例如新增字段），在原表路径下创建新的.frm和.ibd文件，拷贝ibd文件时消耗的IO较多。

DDL执行过程中，并行的DML操作原表，同时会申请row\_log空间记录DML操作，这部分操作会在DDL执行和提交阶段应用到新的表空间中。row\_log空间是一个独立的空间，其大小可通过innodb\_online\_alter\_log\_max\_size控制（默认128M），当DDL过程中，并行的DML超过innodb\_online\_alter\_log\_max\_size容量，就会报错。

Rebuild方式的DDL，对空间有要求，对IO消耗比较大，是代价最大的DDL类型。

### LOCK

控制是否锁表，根据不同的DDL操作类型表现不同，mysql原则是尽量不锁表，但是修改主键这样的昂贵操作不得不锁表

1、LOCK=NONE，允许DDL期间并发读写涉及的表，显式指定时，当不支持对该表的继续写入，则alter语句失败，是ALGORITHM=COPY的默认lock级别

2、LOCK=SHARED，DDL期间表上的写操作会被阻塞，但是不影响select

3、LOCK=DEFAULT，让mysql自己判断lock模式，原则是尽量不锁表

4、LOCK=EXCLUSIVE，DDL期间该表不可用，堵塞任何读写请求，使用场景：

最短时间内完成

短时间表不可用能刚接受

注意：

1、任何模式下，online DDL开始之前都需要一个短时间的排它锁X来准备环境

2、当alter命令发出后，会首先等待该表上的其他操作完成

3、alter命令之后的其他请求会出现等待MDL锁

4、alter完成之前，其他DDL也会被阻塞一小段时间？

https://www.cnblogs.com/zengkefu/p/5674945.html

## 阶段2：执行

在此阶段，准备并执行该语句。元数据锁是否升级到排它锁取决于初始化阶段评估的因素。如果需要排他元数据锁，则仅在语句准备期间进行短暂锁定。

具体步骤：

1、降级exclusvie-mdl锁（EXCLUSIVE->SHARED\_UPGRADABLE），允许DML读写

2、扫描原表（old\_table）的聚簇索引每条记录（rec）

3、遍历新表的聚簇索引和二级索引，逐一处理

4、根据记录（rec）构造对应的索引项

5、将构造索引项插入sort\_buffer块进行排序

6、将sort\_buffer块中的排序结果插入新的索引

7、记录DDL执行过程中产生的增量（仅rebuild类型需要）

8、重放row\_log中的操作到新索引上（no-rebuild数据是在原表上更新的）

9、重放row\_log中间产生的DML操作append到row\_log最后一个Block

## 阶段3：提交阶段

在提交表定义阶段，将元数据锁升级为排它锁，以退出旧表定义并提交新表定义，在获取排它锁的过程中，如果其他事务正在占有元数据的排它锁，那么本事务的提交操作可能会出现锁等待。

具体步骤：

1、当前block为row\_log最后一个时，升级到exclusive-mdl索引（MDL\_EXCLUSIVE），禁止读写

2、应用最后row\_log中产生的日志（重做row\_log中最后一部分增量）

3、更新innodb的数据字典

4、提交事务（刷数据的redo日志）

5、修改统计信息

6、rename临时ibd文件，frm文件

7、变更完成

http://blog.itpub.net/22664653/viewspace-2056953

## 总结

1. 执行阶段加的锁是SHARED\_UPGRADABLE，该阶段允许并行读写
2. Prepare阶段和commit阶段，加的锁是EXCLUSIVE，这两个阶段不能并行DML

由此可见，Online DDL并不是全过程允许DML并行。但是prepare和commit阶段的耗时非常短，占整个DDL流程的比例比较小，对业务影响可以忽略不计。反过来，正在执行的业务可能会对DDL产生影响，可能会产生锁冲突的情况。

1. MDL\_SHARED\_UPGRADABLE之间是互斥的，所以可以保证同一张表不会并行执行多个DDL

# 主备机DDL处理

# 监控

# 失败

官方文档上给出了可能失败的几种情况：

1、手工指定的algorithm和存储引擎中的算法出现冲突

2、在一些必须使用排它锁的场合手工指定锁的类型为share或者为none

3、需要拷贝表的时候系统磁盘空间溢出或者DDL过程中的并发DML临时日志文件过大导致超过了参数innodb\_online\_alter\_max\_size的值

4、当前系统有不活跃的事务占用了元数据锁，导致锁等待超时

5、DDL添加唯一二级索引的时候，并发DML中插入了重复键值的记录，此时会造成alter table的操作回滚

## prepare阶段锁冲突

**场景描述：**

在DDL开始时，如果当前有其他长事务（不论是读还是写）涉及该表，则DDL处于Waiting for table metadata lock阶段，可能会锁等待超时“ERROR 1205(HY000):Lock wait timeout exceeded;try restarting transaction”。

**例如：**

事务1：begin;select \* from t1 where id=1;

事务2：alter table t1 add column c1 varchar(10);

如果先执行事务1，并且不提交，再执行事务2，事务2的DDL就会锁等待超时。

**异常影响：**

DDL失败，业务无影响。

**解决办法：**

1. 对长事务处理：等待提交或者强制杀掉
2. 重新发起DDL操作

## commit阶段锁冲突

**场景描述：**

在DDL commit阶段，如果当前有其他长事务（不论是读还是写）涉及该表，则DDL处于Waiting for metadata lock阶段，可能会锁等待超时“ERROR 1205(HY000):Lock wait timeout exceeded;try restarting transaction”，此时DDL失败，对原表无影响，期间产生的DML无影响，继续使用原表。

**异常影响：**

DDL失败回滚，DDL执行和回滚期间的IO消耗可能会对业务性能产生影响。

**解决方法：**

1. 对长事务处理：等待提交或者强制杀掉
2. 重新发起DDL操作

## Row\_log空间不足

**场景描述：**

row log空间每次申请的大小由innodb\_sort\_buffer\_size决定，最大值是由参数innodb\_online\_alter\_log+max\_size，默认是128M，支持动态修改。对于更新频繁的表来讲，如果预计在DDL期间对表的更新操作存储可能超过128M时，需要为本次操作增大该值。当然如果不涉及rebuild操作时，不需要考虑该值。如果提示DB\_ONLINE\_LOG\_TOO\_BIG错误，则是由innodb\_online\_alter\_log\_max\_size空间不足造成的。

**异常影响：**

DDL失败回滚，业务无影响。

**解决办法：**

1. 调大innodb\_online\_alter\_log\_max\_size值

## 备机回放时延

**场景描述：**

备机收到DDL的binlog信息后，做Online DDL操作，DDL耗时比较长的情况下，由于备机回放线程串行处理，阻塞后续的DML的回放，造成回放延迟。

**异常影响：**无

**解决办法：**无

# 分布式数据库实践

## TDSQL

## GoldenDB

### DDL流程设计

#### 离线DDL

OMM界面执行的DDL属于离线DDL，均不锁表，是因为运维人员能保证当前没有业务在执行。

#### Proxy DDL

##### 流程

Proxy发起的DDL处理流程：

Proxy->PM->MDS修改表状态->PM->通知所有Proxy禁表->Proxy向DB发起DDL操作->PM->MDS修改DDL并解禁->PM->将新的元数据推送到所有Proxy

##### 分类

Proxy的DDL分为在线DDL和非在线DDL。

###### 非在线DDL

对于一般的DDL，proxy会对表做禁表操作，在DDL完成之前，表一直处于禁表状态，不允许对表做任何操作，包括增删改查，返回客户端的错误为：ERROR 11204(HY000)：table ‘db.tb1’ si disabed!

###### 在线DDL

对于在线DDL，proxy不对表做禁表操作，允许并行DML。需要强调的是，proxy支持的在线DDL和mysql支持的在线DDL不一致，proxy只支持以下几种在线DDL：

|  |  |
| --- | --- |
| 类型 | 操作 |
| INDEX | add index |
| drop index |
| COLUMN | alter table add column |
| PARTITION | add partition |
| drop partition |
| truncate partition |
| reorganize partition |
| coalesce partition |
| remove partition |

Proxy不支持alter table t add column f1 int,add column t2语法。

###### 多分片串行/并行方式

修改proxy.ini文件ddl\_execute\_serial，指定DDL的执行顺序：

0：并行执行

1：串行执行

##### 在线DDL

在proxy上执行的DDL属于在线DDL，有些场景锁表有些场景不锁表：

###### 不锁表

1. 如下客户端下发DDL不锁表

alter table add column(NOT NULL default) --新增列必须在最后，不包含FIRST AFTER等指定位置的DDL

alter table add column 允许NULL --新增列必须在最后，不包含FIRST AFTER等指定列位置的DDL语句

alter table add index+drop index --不锁表

alter table add/drop partition --不锁表

alter table modify/change column类型兼容，详细如下：

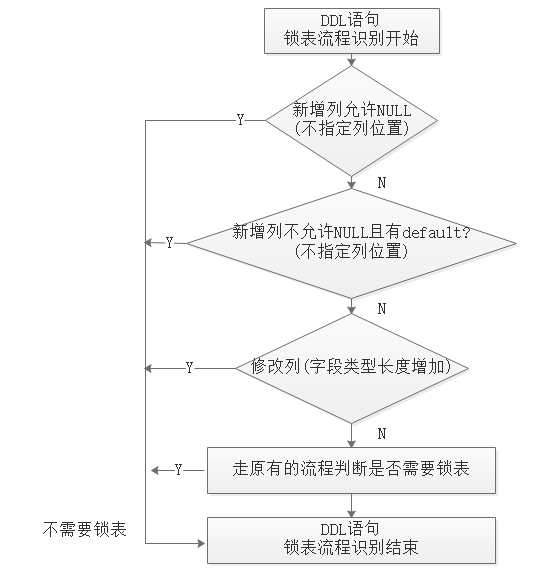
1. 扩长度char、varcha、decimals
2. int类型转换（tinyint->SMALLINT->int->BIGINT）
3. float转换为double，int转换为float double
4. 不停机DDL回滚问题

目前分析以上几种不禁表方案不影响回滚，所以不需要考虑回滚问题

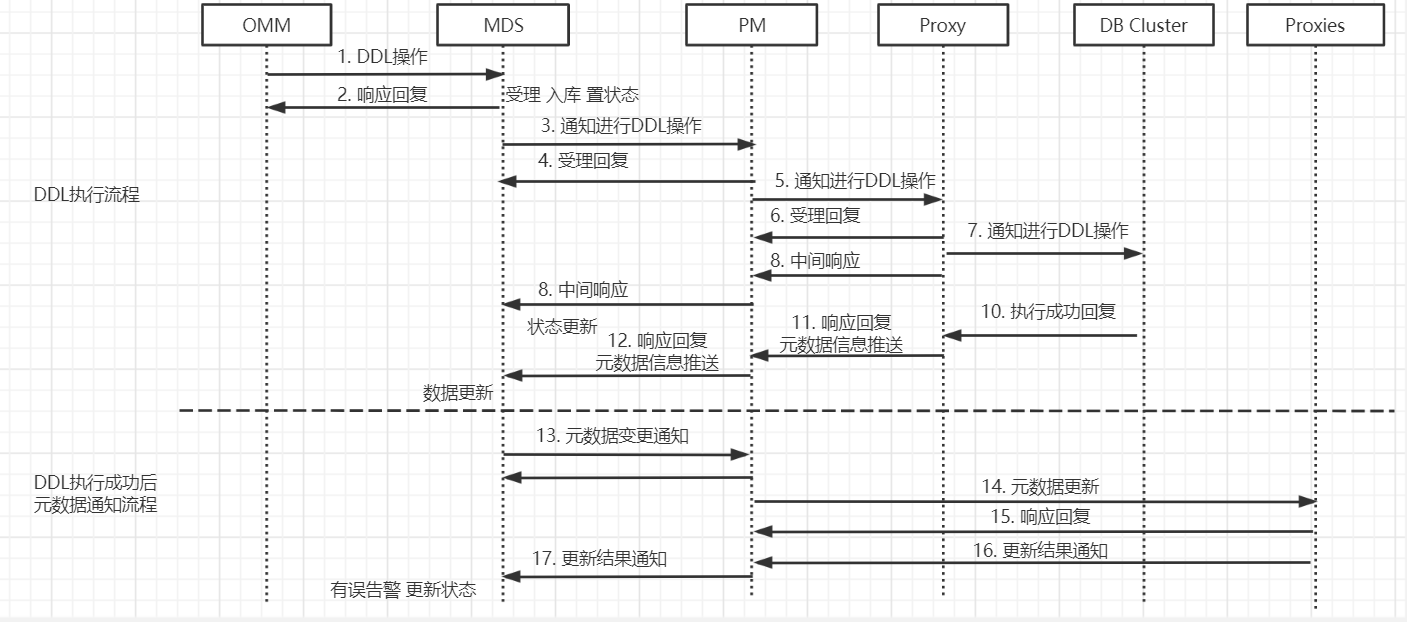
1. 不停机DDL执行失败不需要禁表

不停机DDL执行失败不需要禁表——需要告警

不锁表流程如下：

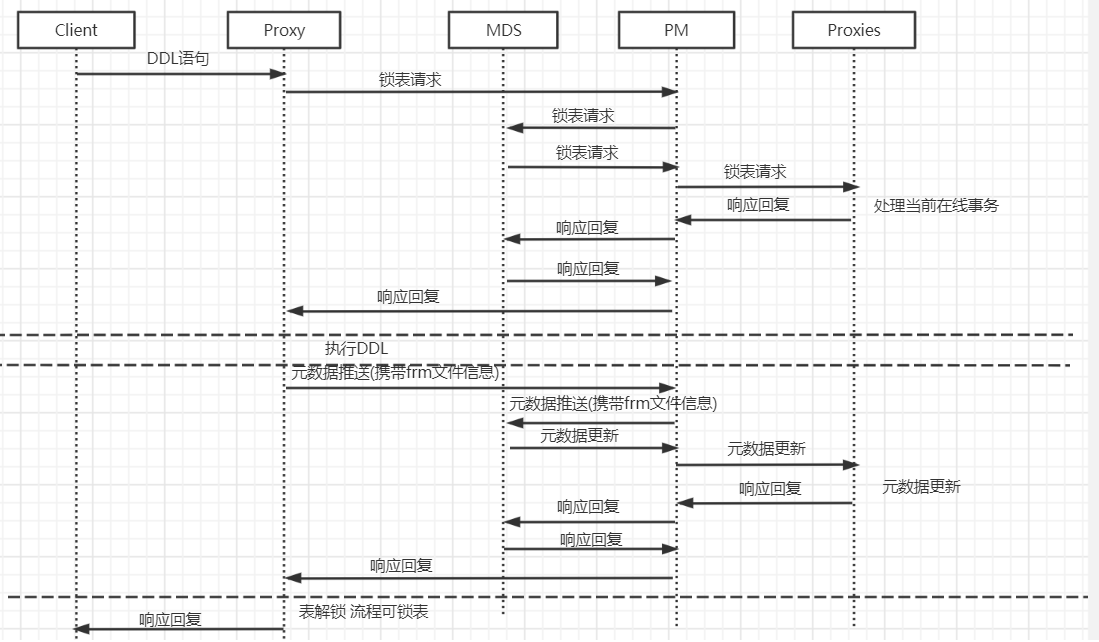


不锁表流程图：



###### 锁表

锁表流程图：



### Proxy和DB差异

|  |  |  |
| --- | --- | --- |
| 场景 | Proxy | DB |
| 新增字段 | 在线DDL，不禁表 | 在线DDL |
| 指定新增字段位置 | 非在线DDL，禁表 | 在线DDL |
| 修改字段注释 | 非在线DDL，禁表 | 在线DDL |
| 增加重复列 | 不报错 | 报错 |
| 批量修改语法 | 不支持 | 支持 |
| 新增字段时，部分分片成功，部分分片失败 | 在线DDL，不禁表  结果：proxy报错，没有禁表，仍然允许业务使用该表 | 分片1原表结构  分片2新表结构 |

### 使用建议

#### 批量处理

通过合并多个DDL，达到性能最大化。例如，如果新增多个字段，可以通过一条DDL完成，这样可以达到只复制一次ibd文件，减少IO消耗和执行时间。

alter table t add column t1,add column t2;

#### 定制DDL的执行方式

语法：alter table xxx,ALGORITHM=xx,Lock=xx;

ALGORITHM与LOCK参数描述：

|  |  |  |
| --- | --- | --- |
| 参数名 | 取值 | 作用 |
| ALGORITHM | COPY | 允许读操作，不允许写操作 |
| INPLACE | 允许读写操作 |
| DEFAULT | 默认值，根据DDL的类型，优选选择INPLACE |
| LOCK | EXCLUSIVE | 对整个表加排它锁，不允许DML并行 |
| SHARED | 对整个表加共享锁，允许读取，不允许写 |
| NONE | 尽可能少加锁，允许读写 |
| DEFAULT | 默认值，根据DDL的类型，尽可能的允许DML |

##### ALGORITH

ALGORITH参数解析：

1. ALGORITHM=COPY相当于offline DDL，ALGORITHM=INPLACE相当于online DDL
2. 支持INPLACE方式的DDL，也支持COPY方式；支持COPY方式的DDL，不一定支持INPLACE方式

例如：

1. 添加字段DDL，支持INPLACE同时支持COPY方式，默认是INPLACE方式，可以强制指定ALGORITHM=COPY，最终是通过offline DDL的方式执行的：

alter table t1 modify column f20 varchar(10),ALGORITHM=COPY;

1. 修改字段数据类型DDL，只支持COPY方式，不支持INPLACE方式，默认是COPY方式，如果强制指定ALGORITHM=INPLACE，报错：

alter table t1 modify column f14 varchar(10),ALGORITHM=INPLACE;

ERROR 1864:ALGORITHM=INPLACE is not supported.Reason:cannot change column type INPLACE,Try ALGORITHM=COPY

##### LOCK

LOCk参数解析：

1. LOCK=SHARED相当于Offline DDL级别的锁，LOCK=NONE相当于online DDL级别的锁
2. LOCK参数不能指定为低于对应DDL级别的锁（锁级别EXCLUSIVE > SHARED > NONE）

例如，修改字段的数据类型属于offline DDL，需要加SHARED锁，不允许写操作，如果LOCK指定为NONE就会报错：

alter table t1 modify column f14 int,lock=none;

ERROR 1864:LOCK=NONE is not supported.Reason:cannot change column type INPLACE,Try LOCK=SHARED

1. LOCK参数可以指定为高于对应DDL级别的锁

例如，添加字段属于online DDL，允许读写操作，如果LOCK指定为EXCLUSIVE，则不允许读写操作

alter table t1 add column f20 varcahr(10),lock=exclusive;

1. 建议不要显式制定LOCK，由DDL自行判断加合适的锁

#### 采用最小代价方式

1. 从对业务的影响、对资源消耗以及执行时间等角度考虑，不同类型的DDL代价差别比较大

Online DDL的no-rebuild方式代价最小，不仅允许并行DML，而且不消耗IO资源，执行通常很快；

Online DDL rebuild方式的DDL代价中等，允许并行DML，但是对空间和IO资源要求高；

Offline DDL，不仅影响业务，同时对空间和IO资源要求高，性能相对Online DDL rebuild低。

Online DDL no-rebuild < Online DDL rebuild < Offline DDL

特别是在多种DDL组合的需求时，计算最小代价的方式。

1. 对于online DDL rebuild方式，并行的DML对DDL执行时间和IO影响，DDL尽量在业务量小的时候进行

#### Proxy和DB分开执行DDL

通过在proxy和DB上分别执行DDL，来达到proxy上不支持的DDL语法，以及达到最佳性能。

样例场景：

Proxy不支持“alter table ta add column f1 int,add column t2,add column t3;”语法，如果分成3个DDL执行，代价将是上述语法的3倍。

最佳实践：

1. 直连每个分片主DB执行“alter table t add column f1 int,add column t2,add column t3;”，使DB上新增3个字段
2. 待第一步完成后，再通过proxy执行3个拆分后的DDL，是元数据和DB上的表结构保持一致

#### 新增字段

|  |  |
| --- | --- |
| 方案 | 从proxy发起，每次新增一个字段；增加多个字段时，分多个时间段，每个时间段只增加一个字段 |
| 语句 | alter table t\_name add column c\_name data\_type; |
| 是否允许在线读写操作 | 是 |
| 磁盘IO影响 | 产生大量IO，对数据库性能有影响 |
| 影响DDL执行时长因素 | 表数据量，DDL过程中的在线写操作数量 |
| 执行限制 | 1. 不允许在later语句中加first、after关键字； 2. 选择业务量少的时间段执行 |

#### 修改字段类型

|  |  |
| --- | --- |
| 方案 | 从proxy发起 |
| 语句 | alter table t\_name modify column c\_name newData\_type; |
| 是否允许在线读写操作 | 否 |
| 磁盘IO影响 | 产生大量IO，对数据库性能有影响 |
| 影响DDL执行时长因素 | 表数据量，DDL过程中的在线写操作数量 |
| 执行限制 | 1. 确认执行DDL期间没有在线业务对表进行读写，否则涉及在线业务会失败；   2、选择业务量少的时间段执行 |

#### 修改字段comment

|  |  |
| --- | --- |
| 方案 | 从proxy发起 |
| 语句 | alter table t\_name modify column c\_name data\_type comment ‘xxx’; |
| 是否允许在线读写操作 | 否，时间极短 |
| 磁盘IO影响 | 无 |
| 影响DDL执行时长因素 | 无 |
| 执行限制 | 1. 确认执行DDL期间没有在线业务对该表进行读写，否则涉及在线业务会失败； 2. 选择业务量少的时间段执行 |

#### 新增索引

|  |  |
| --- | --- |
| 方案 | 从proxy发起 |
| 语句 | alter table t\_name add index idx\_name(c\_name1,...); |
| 是否允许在线读写操作 | 是 |
| 磁盘IO影响 | 产生大量IO，对数据库性能有影响 |
| 影响DDL执行时长因素 | 表数据量 |
| 执行限制 | 选择业务量少的时间段执行 |

#### 新增分区

|  |  |
| --- | --- |
| 方案 | 从proxy发起 |
| 语句 | alter table t\_name add partition(partition p\_name values less than(xxx)); |
| 是否允许在线读写操作 | 是 |
| 磁盘IO影响 | 无 |
| 影响DDL执行时长因素 | 无 |
| 执行限制 | 选择业务量少的时间段执行 |