



三歪连MySQL大表怎么DDL变更都不懂



mike_hit 发布于 2022-4-19 15:17

0收藏 9546浏览



作者 | 敖丙

来源 | 敖丙(ID：JavaAudition)

转载请联系授权(微信ID：Aobingcool)



前言

随着业务的发展，用户对系统需求变得越来越多，这就要求系统能够快速更新迭代以满足业务需求，通常系统版本发布时，都要先执行数据库的DDL变更，包括创建表、添加字段、添加索引、修改字段属性等。

在数据量大不大的情况下，执行DDL都很快，对业务基本没啥影响，但是数据量大的情况，而且我们业务做了读写分离，接入了实时数仓，这时DDL变更就是一个的难题，需要综合各方业务全盘考虑。

下面就聊聊这些年我公司在里面，MySQL中的DDL执行方式的变化、大表DDL该如何选择以及DDL执行过程监控。

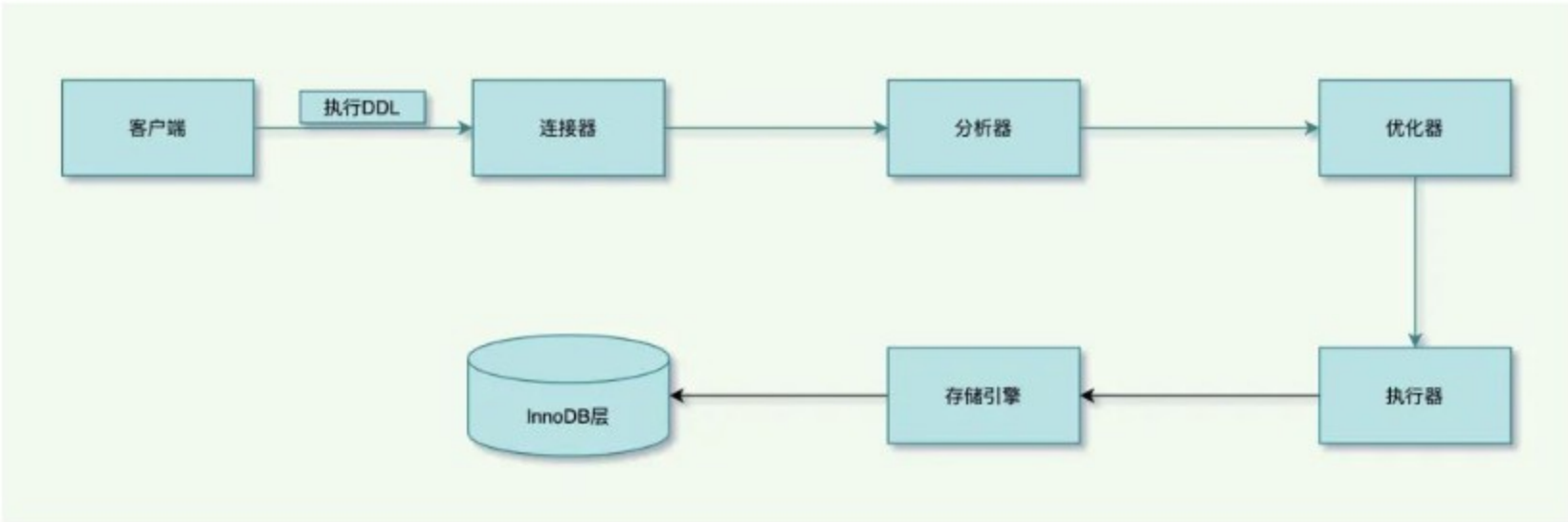
MySQL中的DDL

DDL概述

MySQL中的DDL语句形式比较多，概括一下有以下几类：CREATE，ALTER，DROP，RENAME，TRUNCATE。

这些操作都是隐式提交且原子性，要么成功，要么失败，在MySQL 8.0之前DDL操作是不记录日志的。

今天就聊一下跟系统版本发布相关的数据库结构变更，主要就是ALTER TABLE变更了，DDL变更流程普通的DML变更是类似的，如下所示



注：这里涉及MySQL基础知识，还不知道的朋友翻看下我MySQL基础章节即可。

在早期的MySQL版本，DDL变更都会导致全表被锁，阻塞表上的DML操作，影响业务正常运行，好的一点就是，随着MySQL版本的迭代，DDL的执行方式也在变化。

MetaData元数据

MySQL的元数据（MetaData）跟其他的RDBMS数据库一样的，描述的对象的结构信息，存储在information_sche



mike_hit

LV.1

这个用户很懒，还没有个人简介

15

帖子

0

视频

40

声望

1

粉丝

私信

+关注

最近发布

面试杀手锏：Redis源码之SDS

2022-04-19 16:11:12发布

Redis为什么这么快？

2022-04-19 15:47:29发布

热门推荐

#鸿蒙一夏-2025 HarmonyOS创新赛专场
#问题征集活动 33回复

HarmonyOS 读取系统相册图片并预览 4
回复

云测试提前定位和解决问题 萤火故事屋上
架测试流程 0回复

AI编程神器！Trae+Claude4.0 简单配置
让HarmonyOS开发效率飙升 2回复

【HarmonyOS】鸿蒙应用开发中常用的三
方库介绍和使用示例 1回复

相关问题

三表连表查询+groupby怎么做？ 1回答

mysql 连表查询怎么去重？ 1回答

MySQL 命令行如何导出 DDL？ 1回答

HarmonyOSijkplayer怎么使用，第三方库
看不懂 1回答

mysql怎么从三张表中根据某个字段查询出
想要的数... 1回答

ma架构下，例如常见的TABLES、COLUMNS等，下面例子是创建一个表crm_users，MySQL会自动往Information_schema.tables和columns等相关数据字典表中插入数据，这些数据称为元数据，一般都是静态化，只有表上发生了DDL操作才会实时更新。

```
-- 创建表crm_users
CREATE TABLE `crm_users` (
  `id` bigint NOT NULL AUTO_INCREMENT,
  `name` varchar(20) NOT NULL DEFAULT '' COMMENT '姓名',
  `age` tinyint NOT NULL DEFAULT '0' COMMENT 'age',
  `gender` char(1) NOT NULL DEFAULT 'M' COMMENT '性别',
  `phone` varchar(16) NOT NULL DEFAULT '' COMMENT '手机号',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8mb4 COMMENT='用户信息表'

-- 查看表和列的元数据信息
mysql> select * from information_schema.tables where table_name = 'crm_users'\G
TABLE_CATALOG: def
TABLE_SCHEMA: test
TABLE_NAME: crm_users
TABLE_TYPE: BASE TABLE
ENGINE: InnoDB
VERSION: 10
ROW_FORMAT: Dynamic
TABLE_ROWS: 6525710
AVG_ROW_LENGTH: 94
DATA_LENGTH: 616546304
DATA_FREE: 2097152
AUTO_INCREMENT: 6539682
TABLE_COMMENT: 用户信息表
```

MetaData Lock

MySQL利用MetaData Lock来管理对象的访问，保证数据的一致性，对于一些核心业务表，表上DML操作比较频繁，这个时候添加字段可能会触发MetaData Lock。

| | session1 | session2 |
|----|--------------------------------------|--|
| 01 | select * from crm_users where id > 0 | |
| 02 | 执行中..... | alter table crm_users add column user_type tinyint not null default 0 comment '用户类型' |
| 03 | 执行中..... | MetaData Lock锁等待, Blocking |
| 04 | 执行完成 | |
| 05 | | 开始执行..... |

```
-- 查看processlist信息
mysql> show processlist;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 154 | root | localhost | test | Query | 173 | User sleep | select * from crm_users where id > 0 |
| 155 | root | localhost | test | Query | 156 | Waiting for table metadata lock | alter table crm_users add column user_type20 tinyint not null default 0 comment '用户类型' |
-- 查看metadata_locks字典信息
mysql> select * from performance_schema.metadata_locks where object_name = 'crm_users';
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| OBJECT_TYPE | OBJECT_SCHEMA | OBJECT_NAME | COLUMN_NAME | OBJECT_INSTANCE_BEGIN | LOCK_TYPE | LOCK_DURATION |
| LOCK_STATUS | SOURCE | OWNER_THREAD_ID | OWNER_EVENT_ID |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| TABLE | test | crm_users | NULL | 140350833633792 | SHARED_READ | TRANSACTION |
| GRANTED | sql_parse.cc:6052 | 285 | 49 |
| TABLE | test | crm_users | NULL | 140350815247792 | SHARED_UPGRADABLE | TRANSACTION |
| GRANTED | sql_parse.cc:6052 | 291 | 1053 |
| TABLE | test | crm_users | NULL | 140350837498848 | EXCLUSIVE | TRANSACTION |
| PENDING | mdl.cc:3696 | 291 | 1060 |
```

可以看到Waiting for table metadata lock等待事件，thread 155正在执行alter table等待thread 154执行的select释放锁，因为DML在执行期间会持有SHARED_READ锁，要执行DDL时获取SHARED_UPGRADABLE（共享可升级锁，缩写为SU，允许并发更新和读同一个表）锁成功，但是获取EXCLUSIVE MetaData Lock锁失败，处于暂挂PENDING状态。



上一篇： 敖丙跟你聊聊MySQL安全...
下一篇： 傻瓜MySQL查询缓存都不...

社区精华内容



DDL执行方式

从MySQL官方文档可以看到，ALTER TABLE的选项很多，跟性能相关的选项主要有ALGORITHM和LOCK。

```
ALTER TABLE tbl_name    ALGORITHM [=] {DEFAULT | INSTANT | INPLACE | COPY}
                           LOCK [=] {DEFAULT | NONE | SHARED | EXCLUSIVE}
```

| ALGORITHM OPTION | DESCRIPTION |
|------------------|--|
| COPY | MySQL早期的变更方式，需要创建修改后的临时表，然后按数据行拷贝原表数据到临时表，做rename重命名来完成创建，在此期间不允许并发DML操作，原表是可读的，不可写，同时需要额外一倍的磁盘空间。 |
| INPLACE | 直接在原表上进行修改，不需创建临时表拷贝数据及重命名，原表会持有Exclusive Metadata Lock，通常是允许并发DML操作。 |
| INSTANT | MySQL 5.8开始支持，只修改数据字典中的元数据，表数据不受影响，执行期间没有Exclusive Metadata Lock，允许并发的DML操作。 |

从这张表可以看到，MySQL对于DDL执行方式一直在做优化，目的就是为了提高DDL执行效率，减少锁等待，不影响表数据，同时不影响正常的DML操作。

LOCK选项

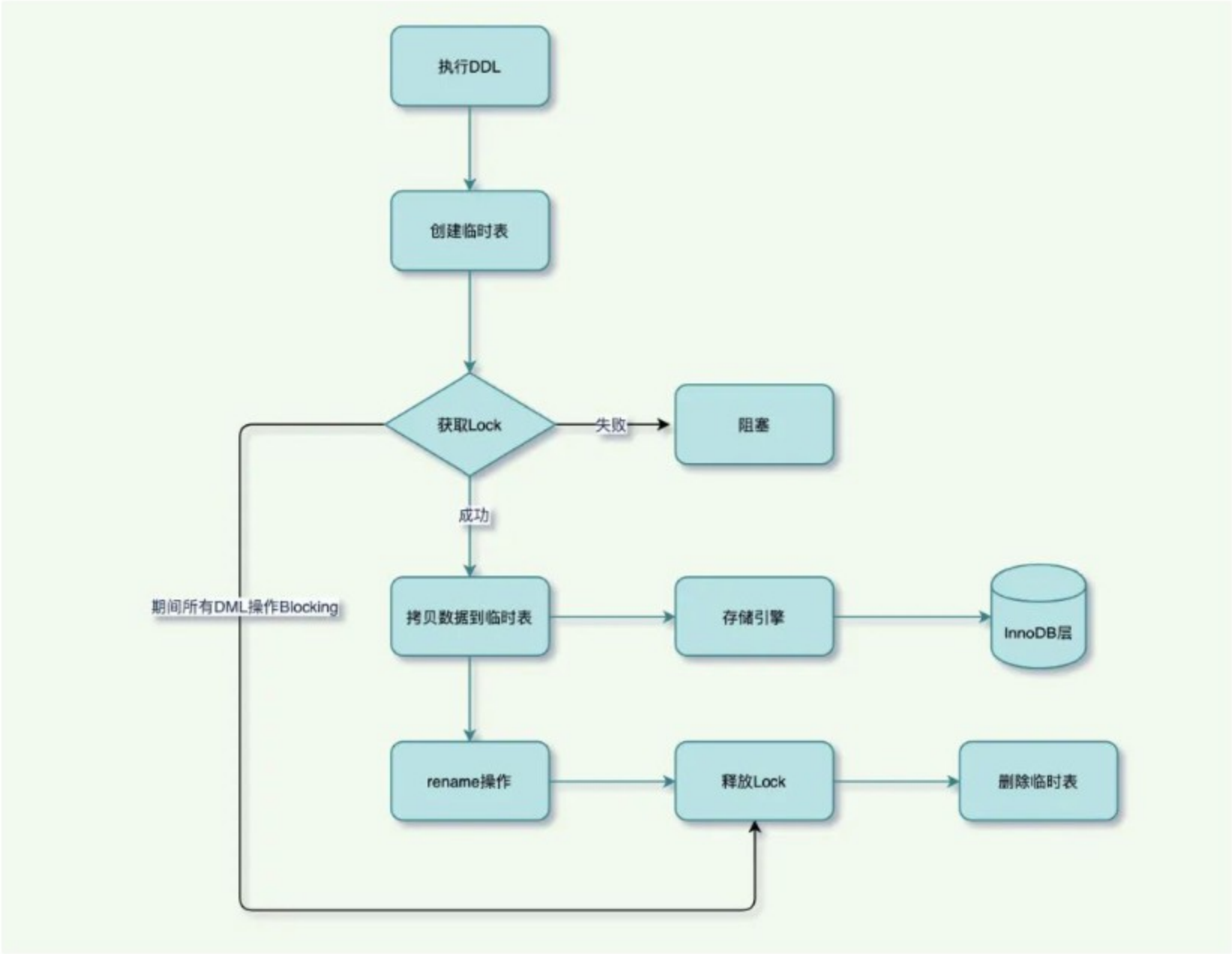
| LOCK OPTION | DESCRIPTION |
|-------------|---|
| DEFAULT | 默认模式：MySQL根据运行情况，在尽量不锁表的情况下自动选择LOCK模式。 |
| NONE | 无锁：允许Online DDL期间进行并发读写操作，如果Online DDL操作不支持对表并发DML操作，则DDL操作失败，对表修改无效。 |
| SHARED | 共享锁：Online DDL操作期间不影响读取，阻塞写入。 |
| EXCLUSIVE | 排它锁：Online DDL操作期间不允许对锁表进行任何操作。 |

下面举例说明下这几种方式的执行过程，先创建测试表，制造一些数据。

```
-- 创建表crm_users
CREATE TABLE `crm_users` (
  `id` bigint NOT NULL AUTO_INCREMENT,
  `name` varchar(20) NOT NULL DEFAULT '' COMMENT '姓名',
  `age` tinyint NOT NULL DEFAULT '0' COMMENT 'age',
  `gender` char(1) NOT NULL DEFAULT 'M' COMMENT '性别',
  `phone` varchar(16) NOT NULL DEFAULT '' COMMENT '手机号',
  `create_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
  `update_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP COMMENT '修改时间',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8mb4 COMMENT='用户信息表'
-- 创建存储过程，方便插入数据
DELIMITER $$
CREATE PROCEDURE insert_user_data(num INTEGER)
BEGIN
  DECLARE v_i int unsigned DEFAULT 0;
  set autocommit= 0;
  WHILE v_i < num DO
    insert into crm_users(`name`, age, gender, phone) values (CONCAT('User',v_i), mod(v_i,120), 'M',
CONCAT('152',ROUND(RAND(1)*100000000)));
    SET v_i = v_i+1;
  END WHILE;
  COMMIT;
END $$
DELIMITER ;
--插入650w数据
mysql> call insert_user_data(6500000);
```

COPY

COPY方式的变更流程如下：



根据业务需要，需要在crm_users添加一个字段user_type，采用COPY方式执行变更。

```
--通过profiling跟踪下执行过程
set profiling = 1

mysql> alter table crm_users add column user_type tinyint not null default 0 comment '用户类型',
ALGORITHM=COPY; --使用COPY方式
Query OK, 6539680 rows affected (29.89 sec)

-- 在datadir指定的目录下，看到临时表#sql-564_85及对应临时数据文件
-rw-r----- 1 mysql mysql 75497472 11 28 10:54 crm_users.ibd
-rw-r----- 1 mysql mysql 54525952 11 28 11:10 #sql-564_85.ibd

--processlist相关信息，可以看到是逐步的fetching rows然后加锁
mysql> select * from information_schema.innodb_trx\G
      trx_id: 8414918
      trx_weight: 15499
      trx_mysql_thread_id: 139
      trx_operation_state: fetching rows
      trx_lock_structs: 15493
      trx_lock_memory_bytes: 2007248
      trx_rows_locked: 3468452

--profile信息
mysql> show profile cpu,block io for query 3;
```

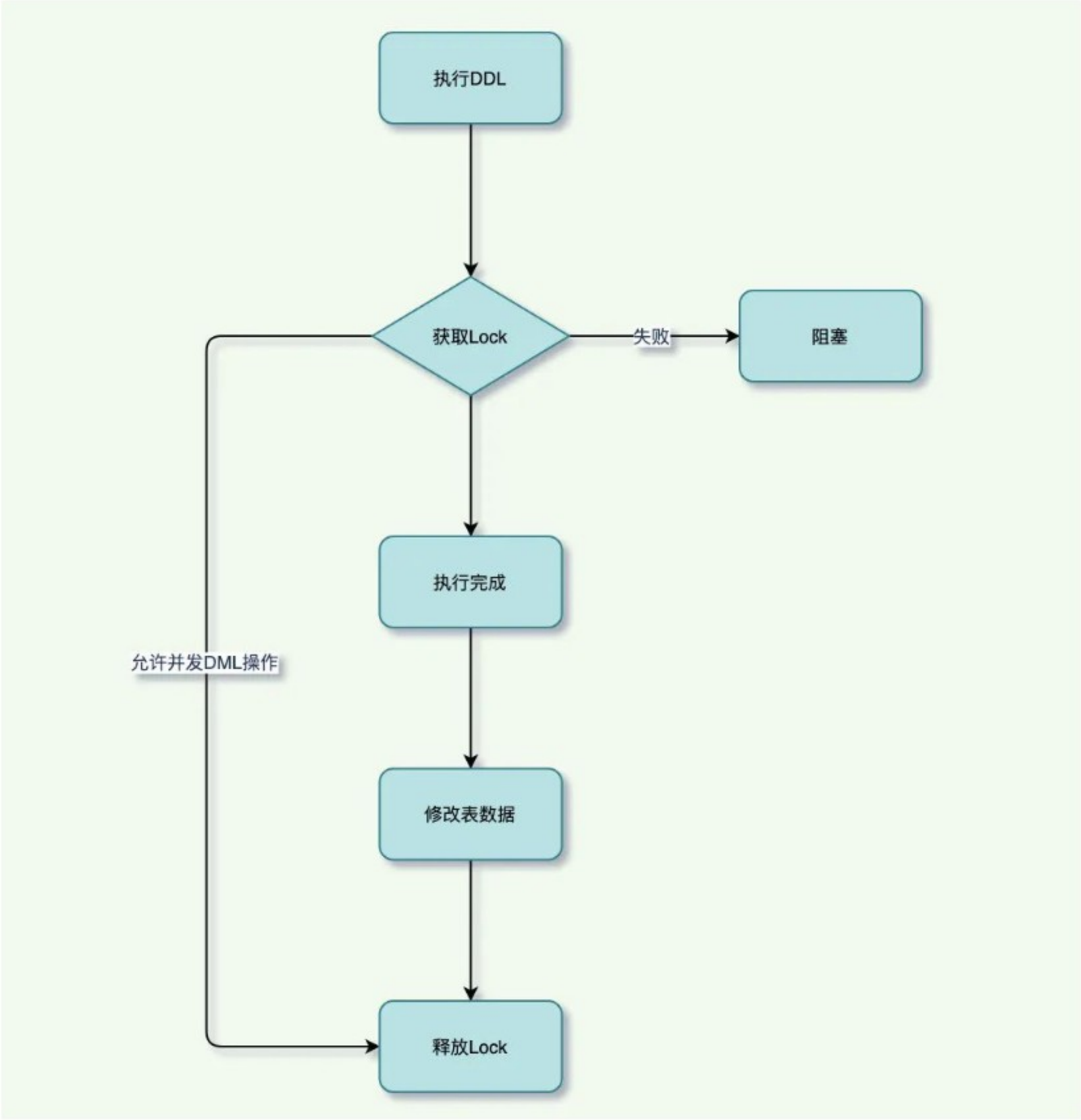
| Status | Duration | CPU_user | CPU_system | Block_ops_in | Block_ops_out |
|---------------------|-----------|-----------|------------|--------------|---------------|
| init | 0.000015 | 0.000012 | 0.000002 | 0 | 0 |
| Opening tables | 0.001802 | 0.000409 | 0.000394 | 0 | 0 |
| setup | 0.000142 | 0.000105 | 0.000037 | 0 | 0 |
| creating table | 0.000197 | 0.000191 | 0.000005 | 0 | 0 |
| After create | 0.046153 | 0.003464 | 0.018263 | 0 | 0 |
| System lock | 0.000017 | 0.000006 | 0.000010 | 0 | 0 |
| copy to tmp table | 29.731857 | 29.397112 | 11.451599 | 0 | 0 |
| rename result table | 0.048384 | 0.012688 | 0.018345 | 0 | 0 |
| end | 0.000532 | 0.000246 | 0.000190 | 0 | 0 |

| | session1 | session2 |
|----|--|---|
| 01 | alter table crm_users add column user_type tinyint not null default 0 comment '用户类型', ALGORITHM=COPY; | |
| 02 | 执行中..... | insert into crm_users(name, age, gender, phone) values('食神',34,'M','12398923212'); |
| 03 | | 阻塞中..... |
| 04 | 执行完成，释放Lock | |
| 05 | | 执行完成 |

从执行过程及profile可以看出，通过COPY方式会创建临时表#sql-564_85，获取System Lock，拷贝数据到临时表，最后做rename表名切换，释放Lock资源，在执行期间不支持并发DML操作。

INPLACE

INPLACE方式是在原表上直接修改，对于添加索引、添加/删除列、修改字段NULL/NOT NULL属性等操作，需要修改MySQL内部的数据记录，需要重建表（Rebuild Table）。



```
mysql> alter table crm_users add column user_type tinyint not null default 0 comment '用户类型', ALGORITHM=INPLACE; -
-使用INPLACE方式
Query OK, 0 rows affected (10.56 sec)

--processlist相关信息
mysql> select * from information_schema.innodb_trx\G
***** 1. row *****
      trx_id: 8415091
      trx_weight: 15
      trx_mysql_thread_id: 143
      trx_operation_state: reading clustered index
      trx_lock_structs: 7
      trx_lock_memory_bytes: 1136
      trx_rows_locked: 4
--profile信息
mysql> show profile cpu,block io for query 5;
+-----+-----+-----+-----+-----+-----+
| Status | Duration | CPU_user | CPU_system | Block_ops_in | Block_ops_out |
+-----+-----+-----+-----+-----+-----+
| init | 0.000021 | 0.000019 | 0.000003 | 0 | 0 |
| Opening tables | 0.001468 | 0.000428 | 0.000427 | 0 | 0 |
| setup | 0.000128 | 0.000108 | 0.000018 | 0 | 0 |
| creating table | 0.000201 | 0.000196 | 0.000006 | 0 | 0 |
| After create | 0.001067 | 0.000144 | 0.000398 | 0 | 0 |
| System lock | 0.000033 | 0.000015 | 0.000017 | 0 | 0 |
| preparing for alter table | 0.016628 | 0.003015 | 0.005750 | 0 | 0 |
| altering table | 10.266910 | 13.443390 | 6.238443 | 0 | 0 |
| committing alter table to stor | 0.169194 | 0.014321 | 0.023877 | 0 | 0 |
| end | 0.000591 | 0.000233 | 0.000091 | 0 | 0 |
```

| | session1 | session2 |
|----|--|--|
| 01 | alter table crm_users add column user_type tinyint not null default 0 comment '用户类型', ALGORITHM=COPY; | |
| 02 | 执行中..... | insert into crm_users(name, age, gender, phone) values('食神',34,M,'12398923212'); |
| 03 | | 执行完成 |
| 04 | 执行完成 | |

从执行过程可以看到，需要获取Exclusive Metadata Lock，修改表数据，释放Lock，在执行期间支持并发DML操作。

INSTANT

MySQL 5.8开始推出的方式，DDL只修改数据字典中的元数据，表数据不受影响，没有Exclusive Metadata Lock，允许并发的DML操作，支持的DDL变更是有限制的，目前主要包括添加字段，添加/删除生成列，修改ENUM或SET列，改变索引类型以及重命名表。

```
mysql> alter table crm_users add column user_type tinyint not null default 0 comment '用户类型', ALGORITHM=INSTANT; -
-使用INSTANT方式
Query OK, 0 rows affected (0.19 sec)
--profile信息
mysql> show profile cpu,block io for query 4;
+-----+-----+-----+-----+-----+-----+
| Status | Duration | CPU_user | CPU_system | Block_ops_in | Block_ops_out |
+-----+-----+-----+-----+-----+-----+
| init | 0.000015 | 0.000014 | 0.000002 | 0 | 0 |
| Opening tables | 0.002232 | 0.000720 | 0.000524 | 0 | 0 |
| setup | 0.000189 | 0.000113 | 0.000076 | 0 | 0 |
| creating table | 0.000258 | 0.000206 | 0.000052 | 0 | 0 |
| After create | 0.000089 | 0.000084 | 0.000005 | 0 | 0 |
| System lock | 0.000026 | 0.000011 | 0.000015 | 0 | 0 |
| preparing for alter table | 0.000018 | 0.000015 | 0.000003 | 0 | 0 |
| altering table | 0.000024 | 0.000022 | 0.000002 | 0 | 0 |
| committing alter table to stor | 0.023843 | 0.008900 | 0.009060 | 0 | 0 |
| end | 0.000192 | 0.000155 | 0.000079 | 0 | 0 |
```

比对这三种方式的执行效率

| 执行方式/ 项目 | 数据量 (w) | 执行时间 (s) | 重建表 | 修改Meta Data | 修改Data | 允许并发DML |
|-------------|------------|-------------|-----|----------------|--------|---------|
| COPY | 650 | 29.89 | YES | No | Yes | No |
| INPLACE | 650 | 10.56 | YES | No | Yes | Yes |
| INSTANT | 650 | 0.19 | No | Yes | No | Yes |

ONLINE DDL

截止MySQL 8.0，OnLine DDL有三种方式COPY，INPLACE，INSTANT，MySQL会自动根据执行的DDL选择使用哪种方式，一般会优先选择INSTANT方式，如果不支持，就选择INPLANCE方式，再不支持就只能选择COPY方式了。

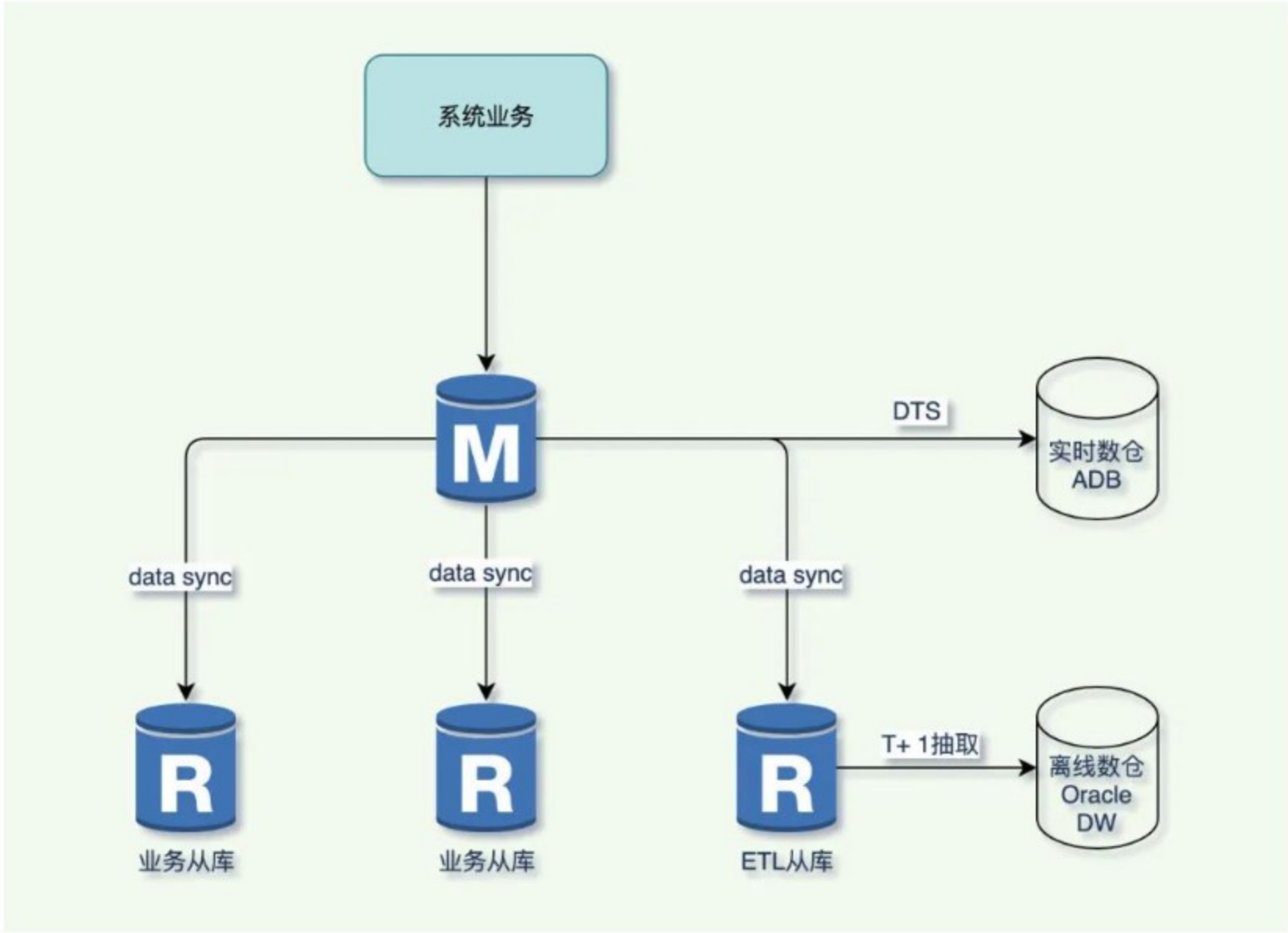
MySQL官方文档也给出了Online DDL的支持矩阵，列下常用的DDL操作，对比项主要包括是否重建表，允许并发的DML操作以及只修改元数据，表数据不受影响。

| Operation | Instant | In Place | Copy | Rebuilds Table | Permits Concurrent DML | Only Modifies Metadata |
|--------------------------------------|---------|----------|------|----------------|------------------------|------------------------|
| Adding a column | Yes | Yes* | Yes | No* | Yes* | Yes |
| Dropping a column | No | Yes | Yes | Yes | Yes | No |
| Renaming a column | No | Yes | Yes | No | Yes | Yes |
| Setting a column default value | Yes | Yes | Yes | No | Yes | Yes |
| Dropping the column default value | Yes | Yes | Yes | No | Yes | Yes |
| Changing the auto-increment value | No | Yes | Yes | No | Yes | No |
| Making a column NULL | No | Yes | Yes | Yes* | Yes | No |
| Making a column NOT NULL | No | Yes | Yes | Yes* | Yes | No |
| Adding a primary key | No | Yes* | Yes | Yes* | Yes | No |
| Dropping a primary key | No | No | Yes | Yes | No | No |
| Creating or adding a secondary index | No | Yes | Yes | No | Yes | No |

| | | | | | | |
|-------------------------|----|------|-----|-----|-----|-----|
| Dropping an index | No | Yes | Yes | No | Yes | Yes |
| Renaming an index | No | Yes | Yes | No | No | No |
| Adding a FULLTEXT index | No | Yes* | Yes | No* | No | No |

大表DDL方案

在实际业务系统中，业务发展比较快，表的数据量比较大，业务层面又做了读写分离，同时会将MySQL数据实时同步到数据仓库（包括实时数仓和离线数仓），实际的数据库架构如下。



假设这是一个交易系统数据库，订单表booking有8000w数据，且接入到了实时和离线仓库，根据业务需要，在订单表booking添加一个字段，在MySQL 5.7之前添加字段属于高危操作，需要充分考虑对业务的影响，主要存在于两个方面：

1. 在读写分离场景，主从同步延迟导致业务数据不一致
2. 实时数仓ADB不允许源端MySQL表重命名，如果通过COPY方式或者pt-osc、gh-ost等工具都会rename表名，那么就需要从数仓删除该表，重新配置同步(全量 + 增量)，会影响数仓业务

ONLINE DDL方式

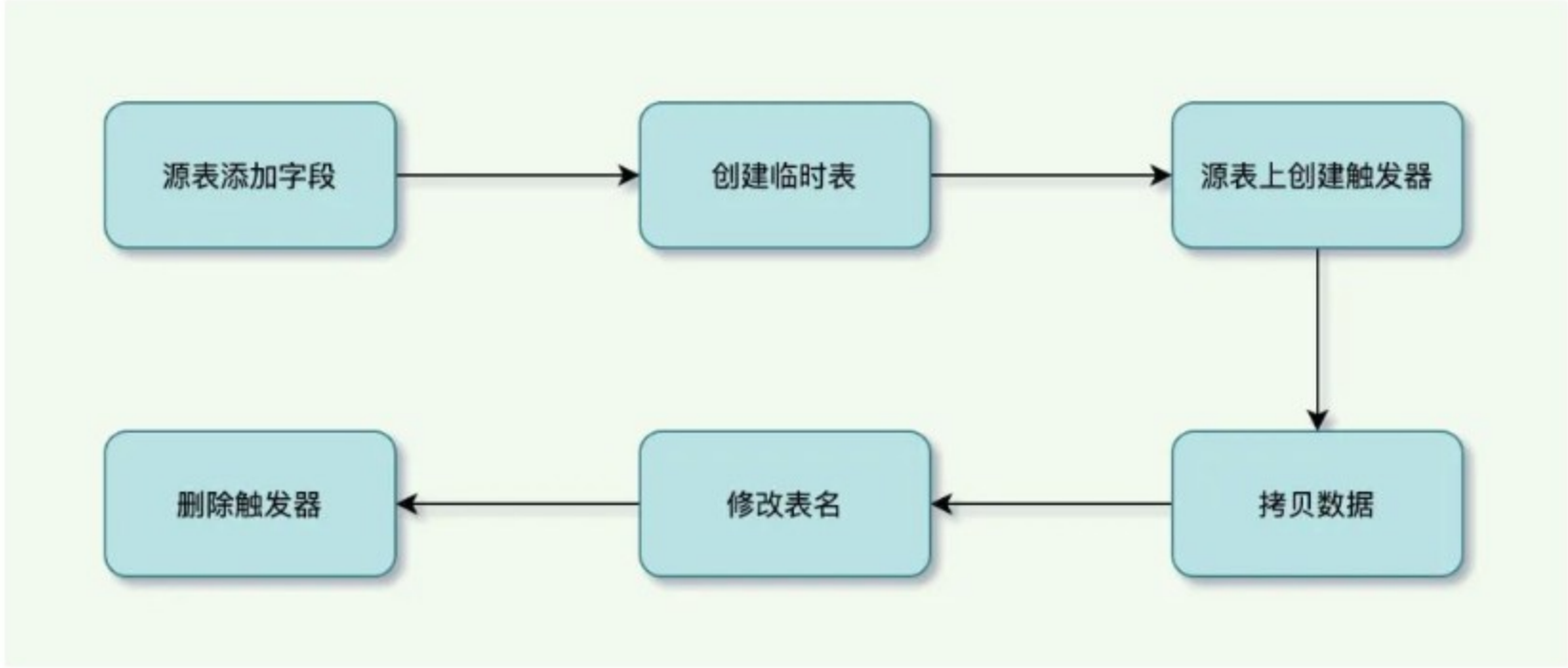
对于MySQL 5.6到5.7的版本，可以使用OnLine DDL的方式变更，对于大表来说，执行时间会很长，好处是在Master上DML操作不受影响，但是会导致主从延时。

假如Master上添加字段执行了20分钟，相应的Slave也要执行20分钟，在这期间Slave一直处于延迟状态，会造成业务数据不一致，比如用户在Master下单成功，由于Slave延迟查询不到订单信息，用户误以为网络原因没有下单成功，又下了一单，导致重复下单的情况。

这种方式会导致主从延迟，但是不会影响实时数仓的业务，根据业务情况，只能选择在业务低峰期执行了。

pt-osc工具

为了解决DDL变更导致主从延时对业务的影响，会想到用大表变更利器pt-osc（pt-online-schema-change）或者gh-ost工具来做，这两个工具执行过程及原理大同小异，变更流程如下（不考虑外键，按照MySQL规范不允许使用外键）：



1. 创建一个新的表，表结构为修改后的数据表，用于从源数据表向新表中导入数据。
2. 在源表上创建触发器，用于记录从拷贝数据开始之后，对源数据表继续进行数据修改的操作记录下来，用于数据拷贝结束后，执行这些操作，保证数据不会丢失。
3. 拷贝数据，从源数据表中拷贝数据到新表中。
4. 修改外键相关的子表，根据修改后的数据，修改外键关联的子表。
5. rename源数据表为old表，把新表rename为源表名，并将old表删除。

6. 删除触发器。

执行pt-osc的时候也需要获取一个Exclusive Metadata Lock，如果在此期间表上有DML操作正在进行，pt-osc操作会一直处于暂挂PENDING状态，这个时候表上正常DML操作都会被阻塞，MySQL活动连接数瞬间暴涨，CPU使用率100%，依赖的该表的接口都会报错，所以要选择在业务低峰期执行，同时做好MetaData Lock锁的监控以便业务不受影响，来看一个例子：

```
pt-online-schema-change \
--host="192.168.22.101" \
--port=3306 \
--chunk-size=1000 \
--user="xxxx" \
--password="xxxx" \
--charset="utf8mb4" \
--max-lag=1 \
--recursion-method="hosts" \
--check-interval=2 \
D=trade, t=booking \
--alter="add remark varchar(50) not null default '' comment '系统备注'" \
--no-version-check \
--execute
```

D=trade, t=booking：数据库trade，表名booking。

--chunk-size=1000：每次拷贝的数据行数。

--max-log = 1：确保从库延迟不超过1s，超过就停止拷贝数据。

--check-interval=2：表示等待2s之后继续拷贝数据。

--recursion-method="hosts"：如果不是使用默认端口3306，那么使用hosts方式来查找从库更可靠。

一般MySQL binlog格式都是ROW，pt-osc在拷贝数据的过程也会产生大量的binlog，也可能导致主从延时，需要控制好每次拷贝数据的大小和频率，在执行期间，也会降低DML的并发度。

MySQL 8.0变更方式

用过Oracle的都知道，DDL变更都是修改元数据，上亿的表在Oracle中DDL变更都是瞬间完成。

令人激动的是，MySQL 8.0也推出了INSTANT方式，真正的只修改MetaData，不影响表数据，所以它的执行效率跟表大小几乎没有关系。建议新系统上线用MySQL的话尽量使用MySQL 8.0，老的数据库也可以升级到MySQL 8.0获取更好的性能。

官方文档对INSTANT的解释：

INSTANT: Operations only modify metadata in the data dictionary. No exclusive metadata locks are taken on the table during preparation and execution, and table data is unaffected, making operations instantaneous. Concurrent DML is permitted. (Introduced in MySQL 8.0.12)

既要解决主从同步，又要解决rename数仓不同步的问题，目前只有INSTANT方式满足需求了。

监控DDL执行进度

在大表执行DDL变更的时候，非常关心它的执行进度，MySQL 5.7之前是没有好的工具去监控，基本只能坐等了。在MySQL 8.0可以通过开启performance_schema，打开events_stages_current事件进行监控。

相关推荐

同事删库跑路后，我**连表名都不能**修改了？

 IIIJDASHJF • 7149浏览 • 0回复

傻瓜**MySQL**查询缓存**都不知道...**

 mike_hit • 8153浏览 • 0回复

mysql单**表**亿级数据分页**怎么**优化？

 qezhu521 • 9650浏览 • 0回复

MySQL索引分类，90%的开发**都不知道**

 netcat20000 • 7941浏览 • 0回复

MySQL索引是**怎么**支撑千万级**表**的快速查找？

 ImCrow • 1.1w浏览 • 0回复

oppo后端16**连问**（三）

 chujichenxuyuan • 7269浏览 • 0回复

年度发布解读 | PolarDB for **MySQL**：**DDL**的优化和演进

 p_wdn • 5384浏览 • 0回复

桥接设计模式(99%的面试官**都不懂**)（一）

 mb628db8cb0e5b3 • 7258浏览 • 0回复

桥接设计模式(99%的面试官**都不懂**)（二）

 mb628db8cb0e5b3 • 7708浏览 • 0回复

MySQL 分库分表

 爱新觉羅、高 • 6133浏览 • 0回复

MySQL全面瓦解4：数据定义–**DDL**

 爱新觉羅、高 • 5779浏览 • 0回复

大表分页查询非常慢，**怎么办**？

 fatherlaw • 8313浏览 • 0回复

他**连**动态规划的一个模型**三个特征都不懂**

 danielmou • 4851浏览 • 0回复

分库分表经典15**连问**

 Bald_eagle • 4942浏览 • 0回复

MySQL索引15**连问**，抗住！

 Bald_eagle • 5244浏览 • 0回复


MySQL 8.0.29 instant **DDL** 数据腐化问题分析

 史前动物 • 3414浏览 • 0回复

MySQL 8.0.29 instant **DDL** 数据腐化问题分析

 heatdog • 2828浏览 • 0回复

聊聊 **MySQL** 的 Online DDL

 laomugua • 3150浏览 • 0回复

三种方案优化 2000w 数据大表！

 Bald_eagle • 2333浏览 • 0回复