43 | 要不要使用分区表

初始化语句

CREATE TABLE `t` (

`ftime` datetime NOT NULL,

`c` int(11) DEFAULT NULL,

KEY (`ftime`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1

PARTITION BY RANGE (YEAR(ftime))

(PARTITION p\_2017 VALUES LESS THAN (2017) ENGINE = InnoDB,

PARTITION p\_2018 VALUES LESS THAN (2018) ENGINE = InnoDB,

PARTITION p\_2019 VALUES LESS THAN (2019) ENGINE = InnoDB,

PARTITION p\_others VALUES LESS THAN MAXVALUE ENGINE = InnoDB);

insert into t values('2017-4-1',1),('2018-4-1',1);



图 1 表t 的磁盘文件

初始化插入两条数据，按照分区定义规则，这两行记录分别落在p\_2018和p\_2019这两个分区上

这个表包含了一个.frm文件和4个.ibd文件，每个分区对应一个.ibd文件，也就是说

·对引擎蹭来说，这是4个表

·对Server层来说，这是1个表

**分区表的引擎层行为**

分区表加间隙锁的例子，目的是说明对InnoDB来说，这个是4个表

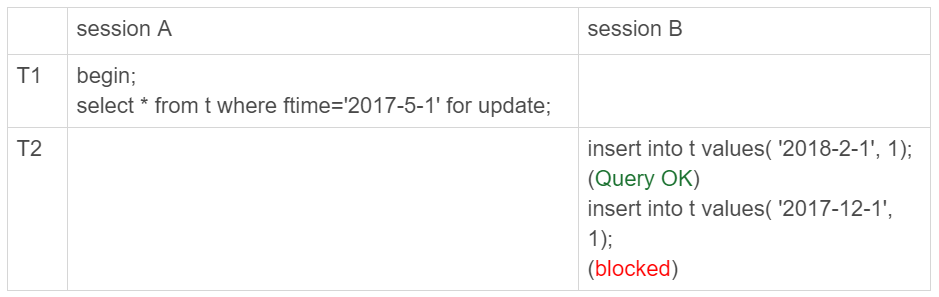


图 2 分区表间隙锁示例

在第21篇介绍间隙锁加锁规则

初始化表t时，吃插入两行数据，ftime的值分别是‘2017-4-1’和‘2018-4-1’.sessionA的select语句对索引ftime上这两行记录之间的间隙加了锁，如果是一个普通表的话，那么T1时刻，在表t的ftime索引上，间隙和加锁状态应该是图3这样

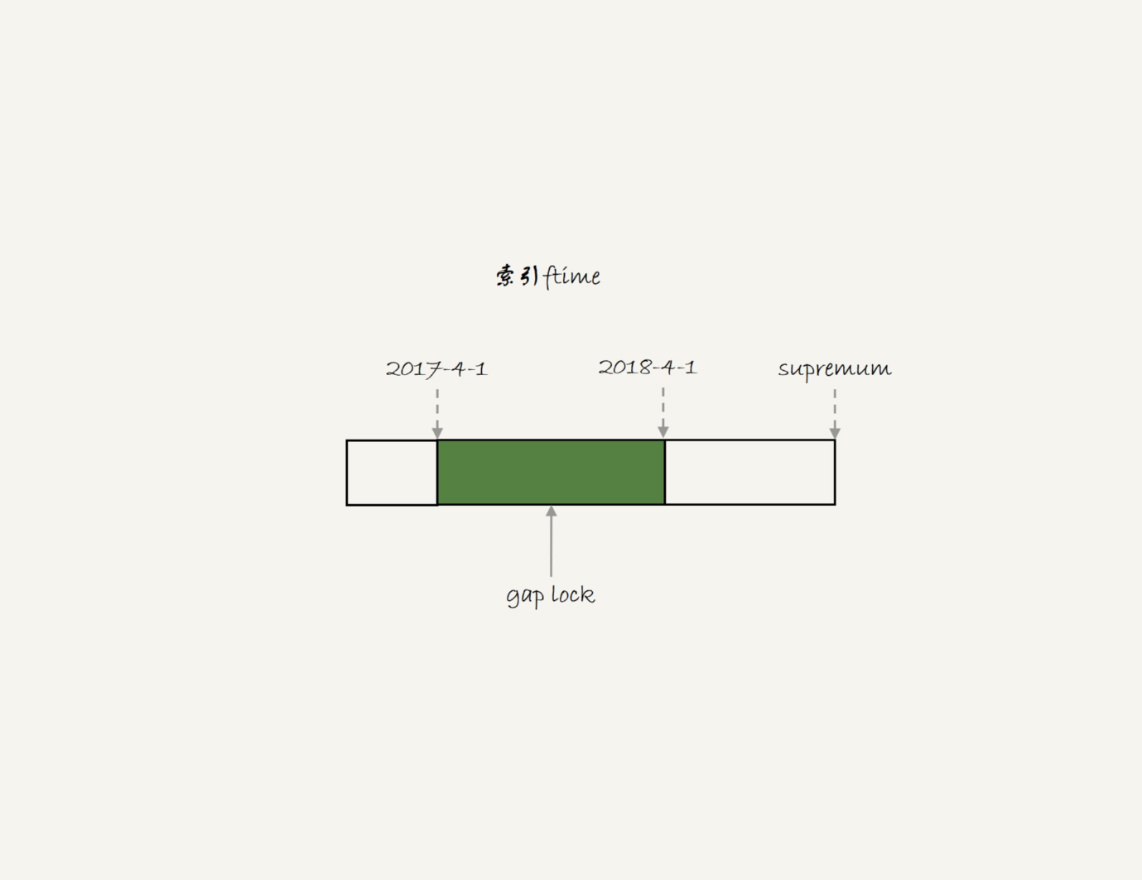


图3 普通表的加锁范围

也就是说，’2017-4-1’和’2018-4-1’这两个记录之间的间隙会被锁住，那么sessionB的两条插入语句应该都要进入锁等待状态

但是，从上面的实验可以看出，sessionB的第一个insert语句是可以执行成功的。这是因为对于引擎来说，p\_2018和p\_2019是两个不相同的表，也就是2017-4-1的下一个记录并不是2018-4-1,而是p\_2018分区的supremum,所以T1时刻，在表t的ftime索引上，间隙和加锁的状态其实是图4这样的

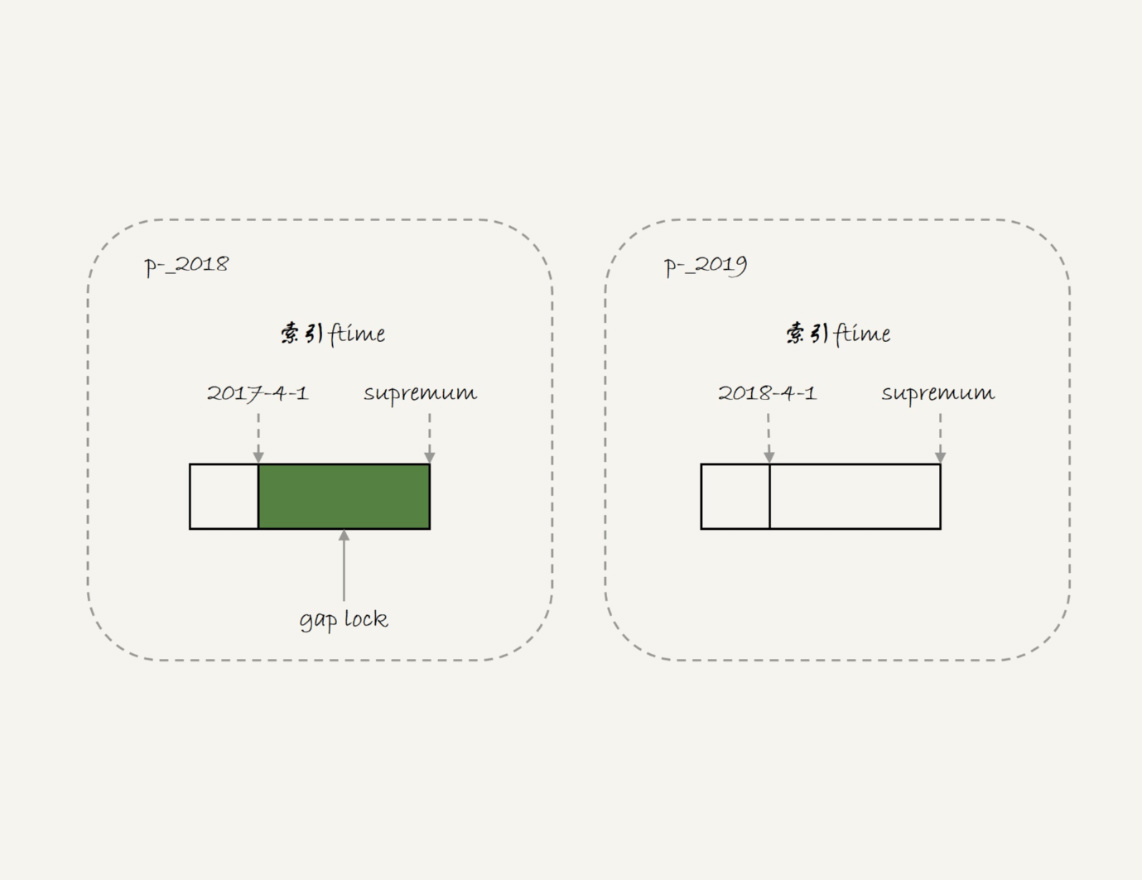


图 4 分区表t的加速范围

有由于分区表的规则，sessionA的select 语句其实只操作了分区p\_2018，因此加锁范围就是图4中深绿色的部分

所以，sessionB要写入一行ftime是2018-2-1是可以成功的，而要写入2017-12-1这个记录，就要等sessionA的间隙锁

图5 就是此时的show engine innodb status的部分结果

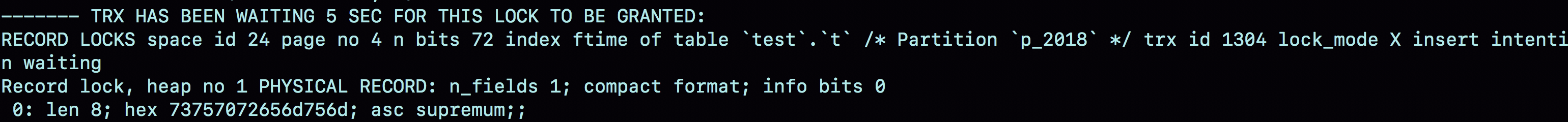


图 5 sessionB 被锁住信息

看完innoDB引擎的例子，再来一个MYIsam分区表的例子

首先用alter table t engine=myisam，把表t改成MyISAM表，然后，在用下面这个例子说明，对于MyISAM引擎来说，这是4个表

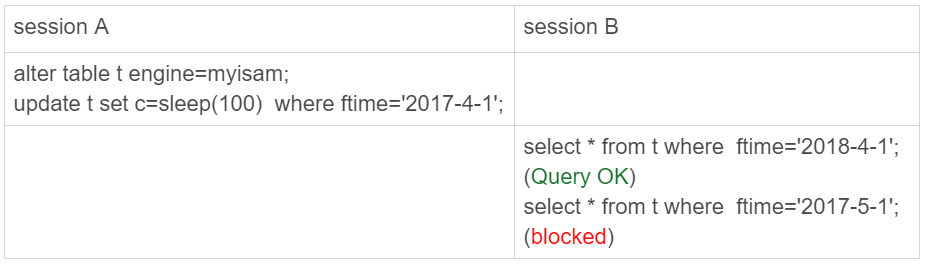


图 6 用MyISAM表锁验证

在sessionA里面，用sleep(100) 将这条语句的执行时间设置为100秒，由于MyISAM引擎只支持表锁，所以这条update语句会锁住整个表t上的锁。

但是结果，sessionB的第一条查询语句是可以正常执行的，第二条语句才进入锁等待状态。

这正式因为MyISA的表锁是在引擎层实现的，sessionA加的表锁，其实是锁在分区p\_2018上，因此，就会堵住在这个分区上执行的查询，落到其他分区的查询是不受影响的

使用分区表的一个重要原因就是单表过大。那么，如果不使用分区表的话，就要使用手动分表的方式。

手动分表和分区表有什么区别

比如，按照年份来划分，就分别创建普通表t\_2017,t\_2018,t\_2019等等，手工分表的逻辑，也是找到需要更新的所有分表，然后依次执行更新，在性能上，这个和分区表并没有实质的差别。

其实这两个方案的区别，主要是在server层上，从server层看，不得不提到分区表的一个饿诟病的问题：打开表的行为。

**分区策略**

每当第一次访问一个分区表时，MySQL需要把所有的分区表都访问一个遍，**一个典型的报错情况**是这样的：如果一个分区表的分区很多，比如超过了1000个，而MySQL启动时，open\_files\_limit参数使用的默认值是1024,那么就会在访问这个表时，由于需要打开所有的文件，导致打开文件的个数超过了上限而报错

下图是一个包含了很多分区的表t\_myisam,执行插入语句后报错的情况

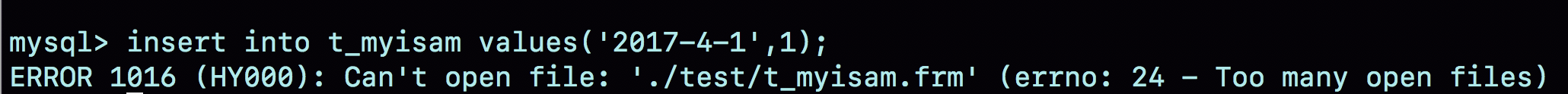


图 7 insert语句报错

可以看到，这条insert语句，明显只需要访问一个分区表，但语句却无法执行。

此时，这个表使用了myisam引擎，因为使用InnoDB引擎，并不会出现这个问题

MyISAM分区表使用的分区策略，称为“**通用分区策略**”（generic partitioning） 每次访问分区都由server层控制，通用分区策略，是MyISAM一开始分区表的时候就存在的代码，在文件管理，表管理的实现上很粗糙，因此有比较严重的性能问题。

从MySQL5.7.17开始，将MyISAM分区表标记为即将弃用（deprecated），意思是“从这个版本开始不建议这么使用”，请使用代替方案，在将来的版本会废弃这个功能。

从MySQL8.0开始，就不允许创建MyISAM分区表了，至于允许创建已经实现了本地分区策略的引擎，目前来开，只有InnoDB和NDB这两个引擎支持本地分区策略

**分区表的server层行为**

如果从server层看的话，一个分区表就只是一个表

如图8和图9所示，分别是这个例子操作序列和执行结果图

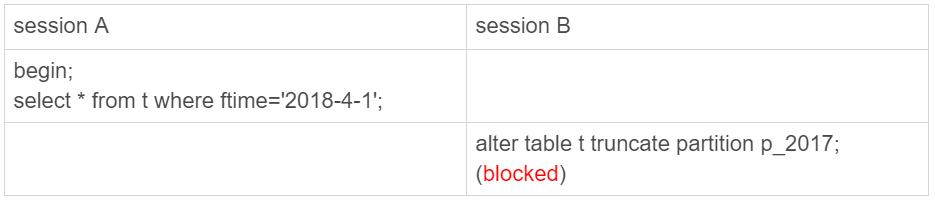


图 8 分区表的MDL锁

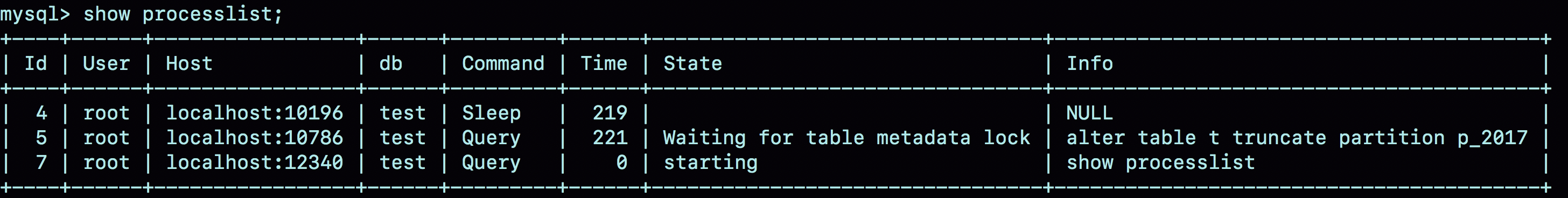


图 9 show processlist结果

虽然sessionB只需要操作p\_2017这个分区，但是由于sessionA持有整个表t的MDL锁，就导致了sessionB的alter语句被堵住了。

这也是DBA经常说的，分区表，在做DDL的时候，影响会更大，如果使用的是普通分表，那么在truncate一个分表的时候，肯定不会跟另外一个分表 上的查询语句出现MDL锁冲突

小结

1. MySQL在第一次打开分区表时，需要访问所有的分区表
2. 在sever层，认为这是同一张表，因此所有分区公用同一个MDL锁
3. 在引擎层，认为这是不同的表，因此MDL锁之后的执行过程，会根据分区表规则，只访问必要的分区。

而关于“必要的分区”的判断，就是根据SQL语句中的where条件，结合分区规则来实现的，比如上面的例子中，where ftime=’2018-4-1’，根据分区规则year函数算出来的值是2018,那么会落在p\_2019这个分区。

但是，如果这个where条件改成 where ftime > = ‘2018-4-1’，虽然查询结果相同，但是此时根据where条件，就要访问p\_2019和p\_others这两个分区。

如果查询语句的where条件中没有分区key，那只能访问所有分区。当然这并不是分区表的问题。即使是使用业务分表的方式，where条件中没有使用分表的key，也必须访问所有的分表。

**分区表的应用场景**

分区表的一个显而易见的优势是对业务透明，相对于用户分表来说，使用分区表的业务代码更简洁。还有，分区表可以很方便的清理历史数据。

如果一项业务跑的时间足够长，往往就会有根据时间删除历史数据的要求，此时，按照时间分区的分区表，就可以直接通过altert table t drop parttion... 这个语法删掉分区，从而删掉过期的历史数据

这个alter table t drop partion 操作是直接删除分区文件，效果跟drop普通表类似，与使用delete语句删除数据相比，优势是速度快，对系统影响小。

**总结**

本章以范围分区（range）为例，实际上，MySQL还支持hash分区，list分区等分区方法，<https://dev.mysql.com/doc/refman/8.0/en/partitioning-types.html>可以通过手册

实际使用时，分区表跟用户分表比起来，有两个绕不开的问题，：一个是第一次访问的时，需要访问所有分区，另一个是公用MDL锁。

因此，如果使用分区，就不要创建太多的分区，否则性能不好。

两个问题需要注意

1. 分区不是越细越好。实际上，单表或单分区的数据一千万行，只要没有特别大的索引，对于现在的硬件能力来说都已经是小表。
2. 分区表不需要提前预留太多，使用之前创建即可，比如按月分区，每年年底时把下一年度的12个新分区创建上即可，对于没有数据的历史分区，要及时的drop掉。

**思考题**

举例的表中没有用到自增主键，假设现在要创建一个自增字段id,MySQL要求分区表中的主键必须包含分区字段，如果要在表t的基础上做修改，会怎么定义这个表的主键呢？为什么这么定义呢？