36 | 为什么临时表可以重名

create temporary table temp\_t like t1;

alter table temp\_t add index(b);

insert into temp\_t select \* from t2 where b>=1 and b<=2000;

select \* from t1 join temp\_t on (t1.b=temp\_t.b);

为什么用临时表呢？普通表是不是也可以？

临时表有哪些特征，为什么适合这个场景？

临时表和内存表的概念

·内存表，指定是使用Memory引擎的表，建表语句是create table ... engine=memory,这种表的数据都是保存在内存里，系统重启的时候会被清空，但是表结构还在，除了这两个“奇怪”的特征，冲其他的特征上看，它就是一个正常的表

·临时表 可以使用各种引擎，如果使用InnoDB引擎或者MyISAM引擎临时表，写数据的时候都是在磁盘上，临时表也可以使用Memory引擎

**临时表特征**



图1 临时表特征示例

1. 建表语法是create temporary table
2. 一个临时表只能被创建它的session访问，对其他线程不可见，所以，图中session A创建的临时表t,对于sessionB 就是不可见的
3. 临时表可以与普通表同名
4. Session A内用同名的临时表和普通标的时候，show create语句，以及增删改查语句访问的是临时表
5. Show tables 命令不显示临时表

由于临时表只能被创建它的session访问，所以在这个session结束的时候，会自动删除临时表，由于这个特性，**临时表就特别适合文章开头的Join优化这种场景，**为什么呢

原因包括两方面

1. 不同session创建的临时表可以同名，如果有多个session同时执行join优化，不需要担心表名重复导致建表失败的问题
2. 不需要担心数据删除问题。如果使用普通标，在流程执行过程中客户端发生了异常断开，或者数据库发生异常重启，还需要专门清理过程中生成的数据表。而临时表由于会自动回收，所以不需要这个额外操作。

**临时表的应用**

由于不担心线程间的重名冲突，临时表经常会被用在复杂查询的优化过程中，其中，分库分表系统的垮库查询就是一个典型的使用场景

一个分库分表的场景，就是要把一个逻辑上的大表分散到不同的数据库实例上。比如，将一个大表ht,按字段f,拆分成1024个分表，然后分不到32个数据库实例上，如下图所示

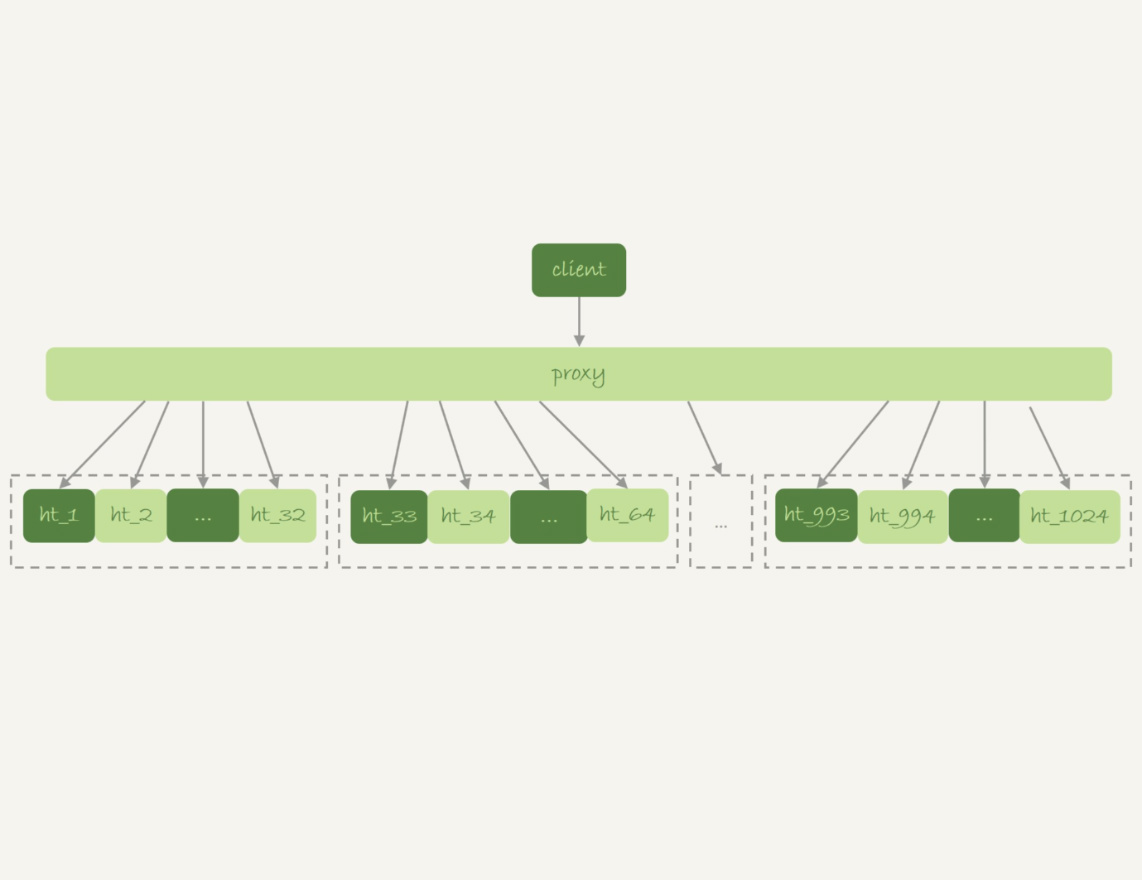


图 2 分库分表简图

一般情况下，这种分库分表系统都有一个中间件层proxy,不过，也有一些方案会让客户端直接连接数据库，也就是没有proxy这层。

在这个架构中，分区key的选择是以“减少夸库和跨表”为依据的，如果大部分的语句都会包含f的等值条件，就用f做分区键，这样，在proxy这一层解析SQL语句后，就能确定将这条语句路由到那个分表做查询

比如这条语句

select v from ht where f=N;

此时，可以通过分表规则（比如，N%1024）来确认需要的数据被放在那个分表上，这种语句只需要访问一个分表，是分库分表最欢迎的语句形式了。

如果这个表还有一索引k，并且查询是这样的

select v from ht where k >= M order by t\_modified desc limit 100;

此时，由于查询条件里没有用到分区字段f,只能到所有的分区中去查找满足条件的所有行，然后统一做order by的操作，这种情况下，有两种比计较长用的思路

**第一种思路 ，**在proxy层的进行代码中实现排序

这种方式的优势是处理速度快，拿到分库的数据以后，直接在内存中参与计算，不过，这个方案的缺点也比较明显。

1. 需要的开发工作量比较大，举例的这条语句还是比较简单的，如果涉及到复杂的操作，比如group by，甚至join这样的操作，对中间层的开发能力要求比较高
2. 对proxy端的压力比较大，尤其是很容易出现内存不够用的CPU瓶颈的问题

**另一种思路，**把个分库的拿到的数据，汇总到一个MySQL实例的一个表中，然后在这个汇总实例上做逻辑操作

比如上面这条语句，执行流程可以类似这样：

·在汇总库上创建一个临时表temp\_ht，表里包含三个字段v,k,t\_modified

·在个分库上执行

select v,k,t\_modified from ht\_x where k >= M order by t\_modified desc limit 100;

·把分库执行的结果插入到temp\_ht表中

·执行

select v from temp\_ht order by t\_modified desc limit 100;

得到结果，流程图如下

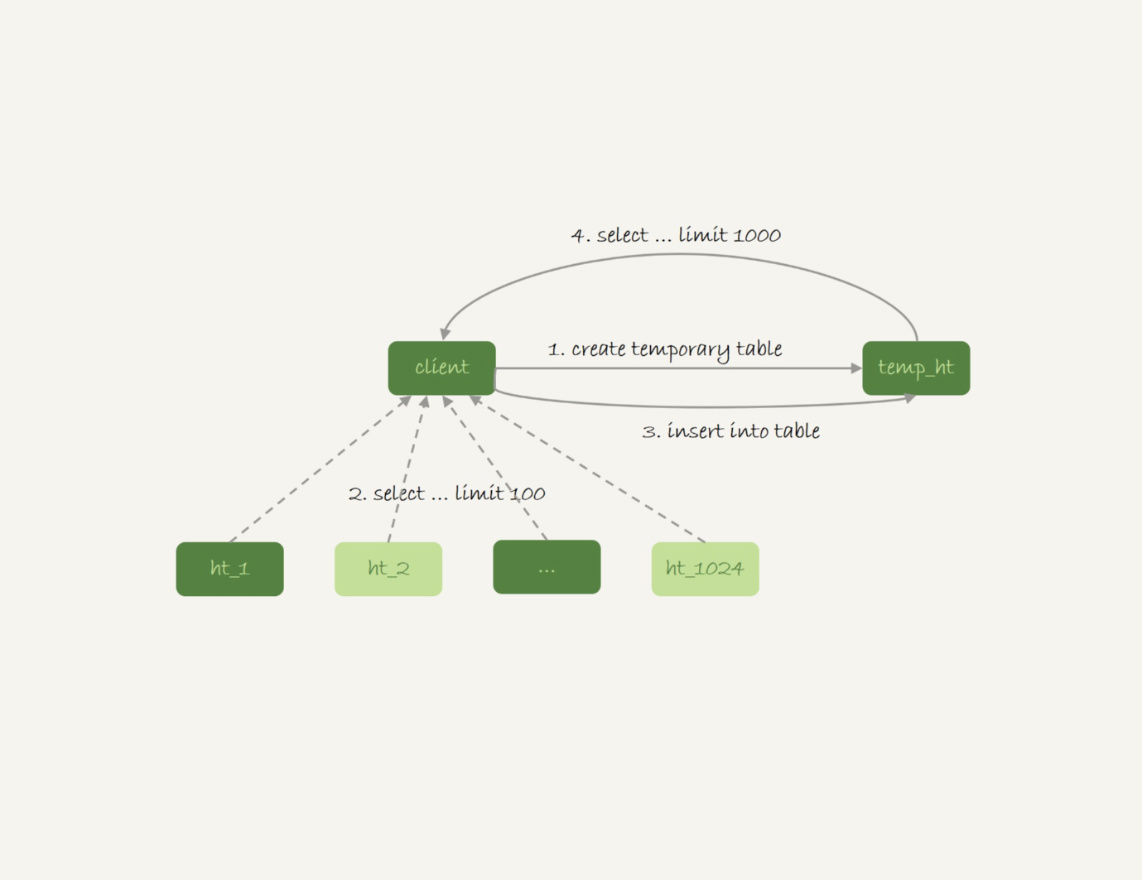


图3 垮库查询流程示意图

**在实践中，每个分库的计算量都不饱和，所以直接把临时表temp\_ht放在32个分库的某一个上**，这是的查询逻辑与图3的类似，

为什么临时表可以重命名？

执行

create temporary table temp\_t(id int primary key)engine=innodb;

MySQL要个这个InnoDB表创建一个frm文件保存表结构定义，还有地方要存表数据

**这个frm 文件放在临时文件目录下，文件名的后缀是.frm，前缀是“#sql{进行id}\_{线程id}\_序列号”**,可以使用select @@tmpdir命令，来显示实例的临时文件目录

关于表中数据的存放方式，在不同的MySQL版本中有不同的处理方式

·5.6以及之前的版本里，MySQL会在临时文件目录下创建一个相同前缀，以.ibd为后缀的文件，用来存放数据文件

·而从5.7版本开始，MySQL引入了一个临时文件表空间，专门用来存放临时文件的数据，因此，就不需要创建ibd文件了

为了便于讨论，举个例子



图 4临时表的表名

这个进程的进程号是1234，session A的id是4，session B的线程id 是5，所以sessionA和sssionB创建的临时表，在磁盘上的文件不会重名

MySQL维护数据表，除了物理上要文件外，内存里有一套机制区分不同的表，每个表对应一个table\_def\_key。

·一个普通表的table\_def\_key的值是由“库名+ 表名”得到的，所以如果要在同一个库下创建两个同名的普通表，创建第二个表的过程中会发现table\_def\_key已经存在了

·对于临时表，table\_def\_key在“库名+表名”基础上，又加入了“server\_id+thread\_id”

SessionA和sessionB创建两个临时表t1,他们的table\_def\_key不同，磁盘文件名也不同，因此可以共存。

在实现上，每个线程都维护了自己的临时表链表，这样每次session内操作表时，先遍历链表，检查是否有这个名字的临时表，如果有就优先操作临时表，如果没有在操作普通表；在session结束的时候，对链表里的每个临时表，执行“DROP TEMPORARY TABLE+表名”操作

此时，binlog中也记录了DROP TMEPORARY TABLE 这条命令，临时表只在线程内可以自己访问，为什么需要写到binlog里面？

这就需要说到**主备复制**了

设想，在主库执行下面的语句

create table t\_normal(id int primary key, c int)engine=innodb;/\*Q1\*/

create temporary table temp\_t like t\_normal;/\*Q2\*/

insert into temp\_t values(1,1);/\*Q3\*/

insert into t\_normal select \* from temp\_t;/\*Q4\*/

如果关于临时表的操作都不记录，那么在备库就只有create table t\_normal表和insert into t\_normal select \* from temp\_t这两个语句的binlog日志，备库执行到insert into t\_normal的时候，就会报错“表temp\_t不存在”

如果把binlog设置为row格式就好了吧，因为binlog是row格式时，在记录insert into t\_normal的binlog时，记录的这个操作的数据，即：write\_row event里面记录的逻辑是“插入一行数据（1，1）”。

如果当前的binlog\_format=row,那么跟临时表有关的语句，就不会记录到binlog里，只在binlog\_format=statement/mixed时，binlog中才会记录临时表的操作。

这种情况下，创建临时表的语句会传到备库执行，因此备库的同步线程就会创建这个临时表。主库在线程退出的时候，会自动删除临时表，但是备库同步线程是持续在运行的，所以，此时就需要在写一个DROP TEMPORARY TABLE传给备库执行。

问题？MySQL在记录binlog的时候，不论create table还是alter table语句，都是原样记录，甚至连空格都不变，但是如果执行drop table t\_normal,系统记录binlog就会写成：

DROP TABLE `t\_normal` /\* generated by server \*/

也就是给成了标准的格式，为什么要这么做呢？

那就是：drop table 命令是可以一次删除多个表的，比如，在上面的例子中，设置binlog\_format=row,如果主库上执行drop table t\_normal,temp\_t 这个命令，那么binlog中就只能记录

DROP TABLE `t\_normal` /\* generated by server \*/

因为备库上并没有temp\_t，将这个命令重写后再传到备库执行，才不会导致备库同步线程停止

所以，drop table命令记录binlog的时候，就必须对语句做改写“/\*generated by server \* /”说明了这是一个被服务端改写过的命令。

说到主备复制，**还有另外一个问题需要解决：** 主库上不同的线程创建同名的临时表是没关系的，但是传到备库执行是怎么处理的呢？

下面的序列中实例S是M的备库



图 5 主备关系中的临时表操作

主库M上的两个session创建了同名的临时表t1,这两个create temporary table t1语句都会被传到备库S上。

但是，备库的应用日志线程是共用的，就是要在应用线程里面先后执行这个create语句两次，（即使开了多线程复制，也可能被分配到从库的同一个worker中执行），那么会不会导致同步线程报错呢？

显然不会，否则临时表就是一个bug了，也就是说备库线程在执行的时候，要把这两个t1表当做两个不同的临时表来处理，这，又是怎么样实现的呢？

MySQL在记录binlog的时候，会把主库执行这个语句的线程id写到binlog中，这样，在备库的应用线程就能够知道执行每个语句的主库的id,并利用这个线程id来构造临时表的table\_def\_key

1. sessionA 的临时表t1,在备库的table\_def\_key就是：库名+t1+ M的serverid+ sessionA的thread\_id
2. sessionB的临时表t1，在备库的table\_def\_key就是：库名+t1+M的serverid+sessionB的thread\_id

由于table\_def\_key不同，所以这两个表在备库的应用线程里面不会冲突

**思考题**

下面的语句序列是创建一个临时表，并将其改名

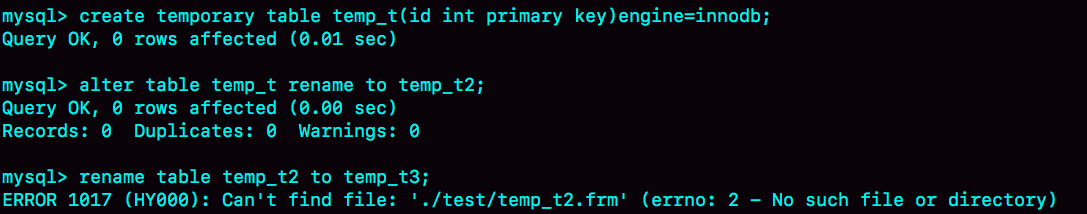


图 6 关于临时表改名的思考题

可以看到，可以利用alter table语法修改临时表的表名，而不使用rename语法，是什么原因？

在实现上，执行rename table时，要求按照“库名/表明.frm”的规则去磁盘找文件，但是临时表在磁盘上的frm文件是放在tmpdir目录下的，并且文件名的规则是“#sql{进行id}\_{线程id}\_序列号.frm”,因此报“找不到文件名”的错误。