38 | 要不要使用Memory引擎

**内存表的数据组织结构**

假设两张表t1和t2，其中t1使用Memory引擎，表t2使用InnoDB引擎

create table t1(id int primary key, c int) engine=Memory;

create table t2(id int primary key, c int) engine=innodb;

insert into t1 values(1,1),(2,2),(3,3),(4,4),(5,5),(6,6),(7,7),(8,8),(9,9),(0,0);

insert into t2 values(1,1),(2,2),(3,3),(4,4),(5,5),(6,6),(7,7),(8,8),(9,9),(0,0);

然后，分别执行select \* from t1和select \* from t2

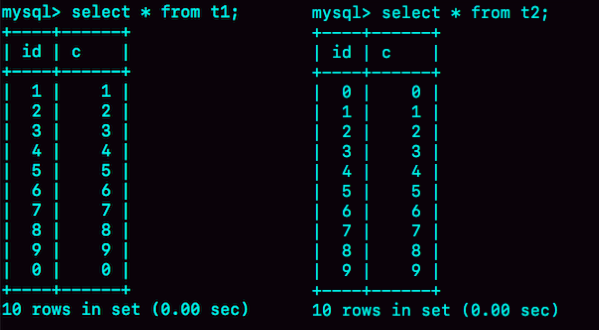


图1 两个查询结果-0的位置

内存表t1的返回结果里面0在最后一行，而InnoDB表t2的返回结果里0是在第一行

表t2用的是InnoDB引擎，它的主键索引id的组织方式，InnoDB表的数据就放在主键索引树上，主键索引B+树，所以表t2的数据组织方式如图所示

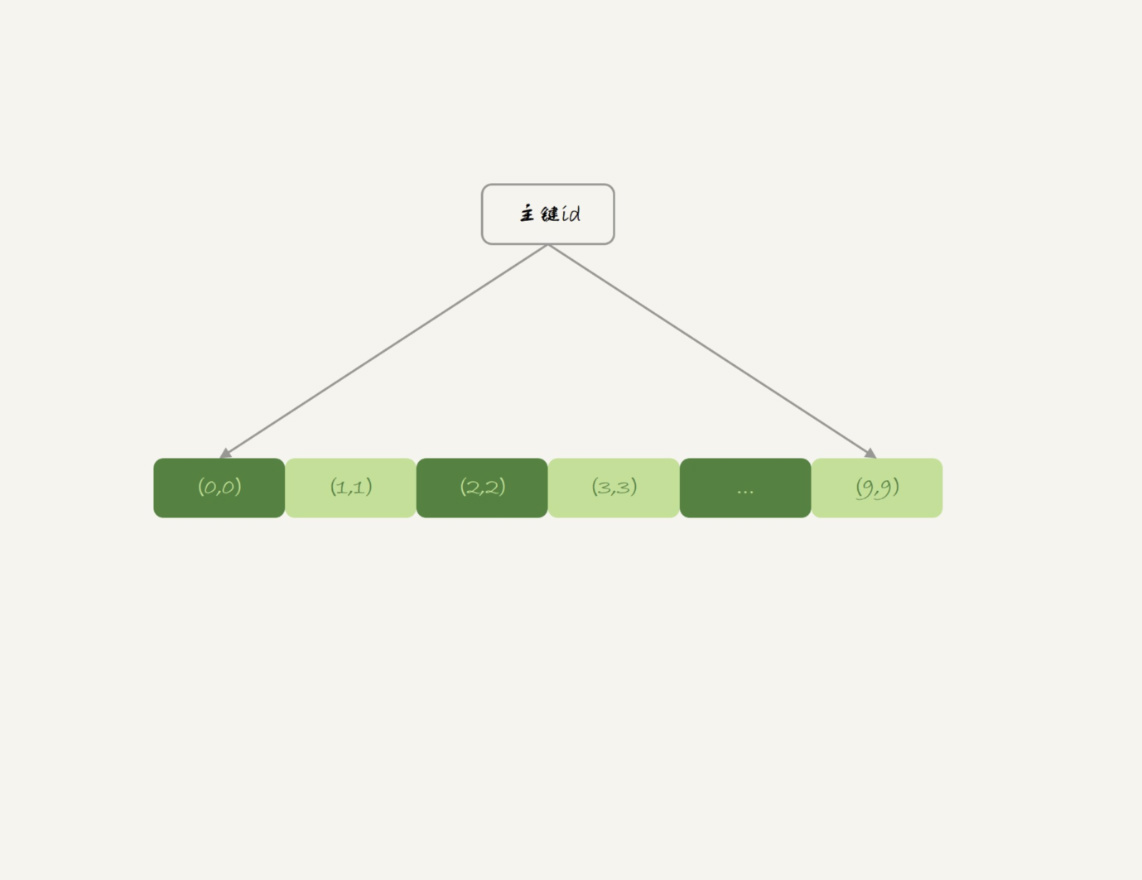


图2 表t2的数据组织方式

主键索引上的值是有序存储的，执行select \* 时，就会按照叶子节点从左到右扫描，所以得到的结果里，0就出现在第一行

与InnoDB引擎不同，Memory引擎的数据和索引是分开的，t1的数据内容如图

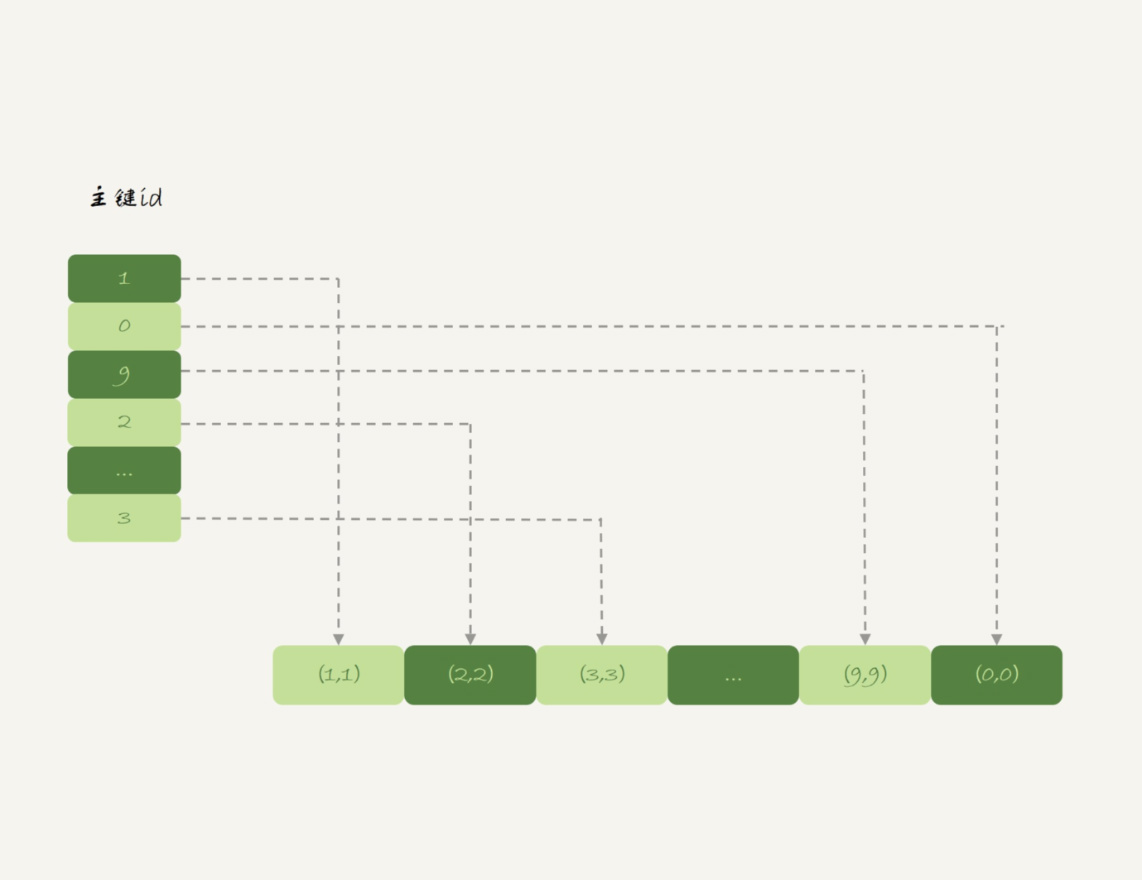


图3 表t1的数据组织

可以看到，内存表的数据部分以数组的方式单独存放，而且主键id索引里，存的是每个数据的位置。主键id是hash索引，可以看到索引上的key并不是有序的。

在内存表t1中，当执行select \* 时，走的全表扫描，也就是顺序扫描这个数组，因此0就是最后一个被读到的，并放入结果集的数据。

InnoDB和Memory引擎的数据组织方式是不同的：

·InnoDB引擎把数据放在主键索引上，其他索引上保存主键id,这种方式，称之为“**索引组织表**”（Index Organizied Table）

·而Memory引擎采用的是把数据单独存放，索引上保存数据位置的数据组织形式，称之为“**堆组织表**”（Heap Organizied Table）

这两个引擎的一些典型不同：

1. InnoDB表的数据总是有序存放的，而内存表的数据就是按照写入顺序存放的
2. 当数据文件有空洞时，InnoDB表在插入时，为了保证数据有序性，只能在固定的位置写入新值，而内存表找到空位就可以插入新值
3. 数据位置发生变化时，InnoDB表只需 修改主键索引，而内存表需要修改所有索引
4. InnoDB表用主键索引查询是需要走一次索引查找，用普通索引查询时，需要走两次索引查找。而内存表没有这个区别，所有索引的“地位”都是相同的。
5. InnoDB支持变长数据类型，不同记录的长度可能不同；内存表不支持Blob和Text字段，并且定义了varchar(N),实际也当做char(N)，也就是固定长度字符串来存储，因此内存表的每行数据长度相同。

由于内存表的特性，每个数据行被删除后，空出的位置都可以被接下来要插入的数据复用，比如要在表t1中执行

delete from t1 where id=5;

insert into t1 values(10,10);

select \* from t1;

就会看出返回结果，id=10这一行出现在id=4之后，也就是原来id=5这行数据的位置

需要指出的是，表t1的这个主键索引是哈希索引，因此如果执行范围查找，比如

select \* from t1 where id<5;

是用不上主键索引的，需要走全表扫描，可以借此回顾第4篇内容，如果要让内存表支持范围查找，该怎么办？

**Hash 索引和B-Tree索引**

内存表也支持B-Tree索引的，在id列上创建一个B-Tree索引，SQL语句可以这么写

alter table t1 add index a\_btree\_index using btree (id);

此时，表t1的数据组织形式就变成了这样

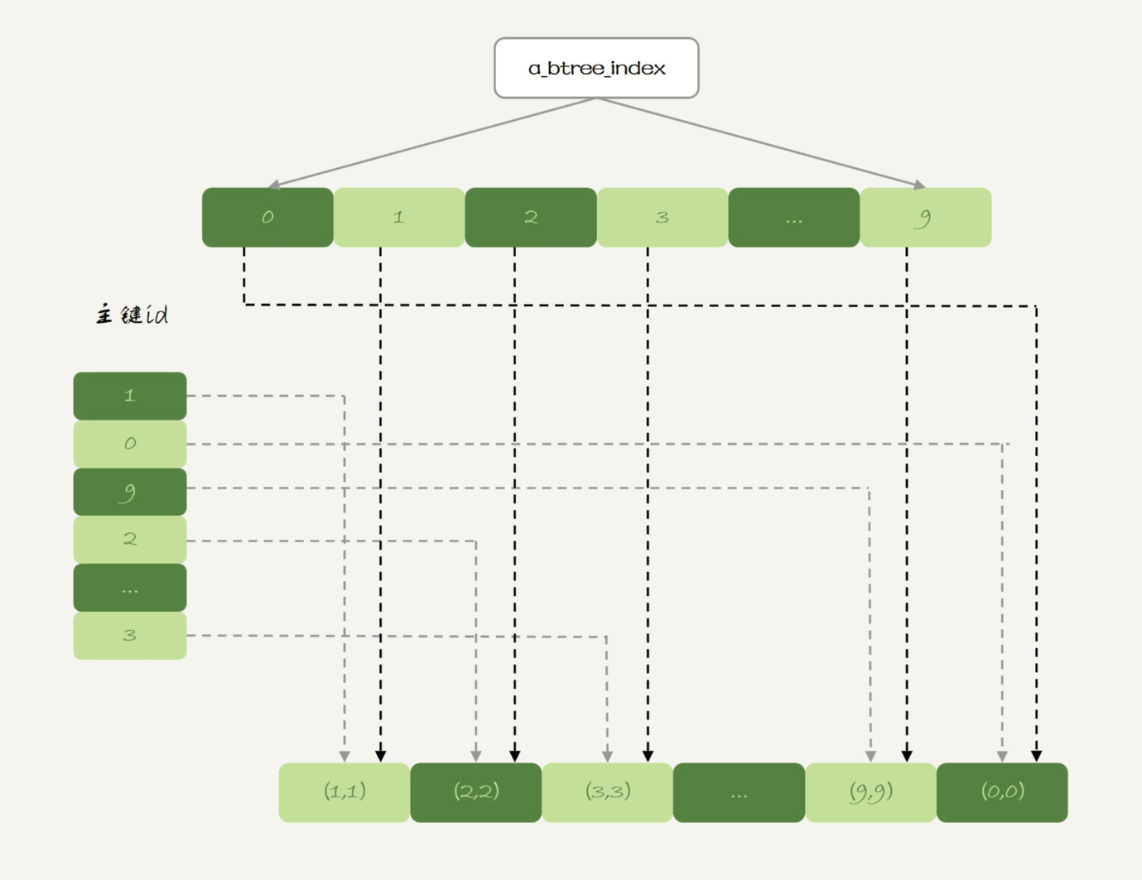


图 4 表t1的数据组织-- 增加B-Tree索引

新增的这个B-Tree索引眼熟，跟InnoDB的b+数索引组织形式类似

作为对比，可以看下两个语句的输出

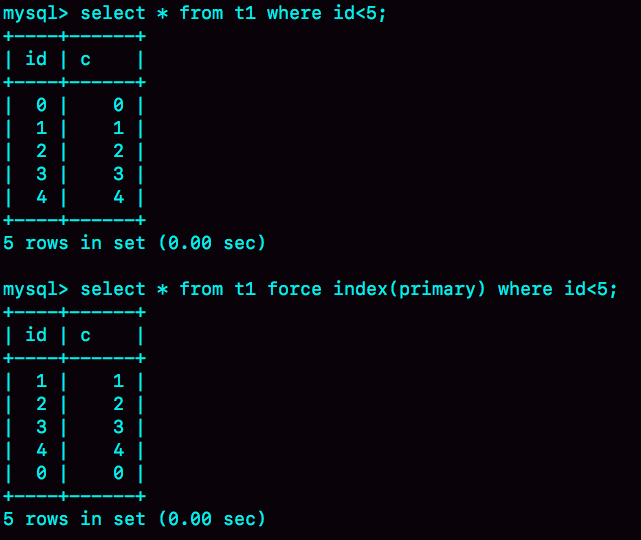


图 5 使用B-Tree和hash索引查找返回结果对比

执行select \* from t1 where id < 5的时候，优化器会选择B-Tree索引，所以返回结果是0到4。使用force index强行使用主键id这个索引，id=0这一行就在结果集的末尾了。

其实，内存表的优势是速度快，其中的一个原因就是Memory引擎支持hash索引,更重要的原因是，内存表的所有数据都保存在内存中，而内存的读写速度比磁盘快

为什么不建议在生产环境使用内存表，这里的原因主要包括两个方面：

1. 锁粒度问题
2. 数据持久化问题

**内存表的锁**

内存表不支持行锁，只支持表锁，因此，一张表只要有更新，就会堵住其他所有在这个表上的读写操作

需要注意的是，这里的表锁跟之前介绍的MDL锁不同，但都是表级的锁，

通过场景模拟内存表的表级锁

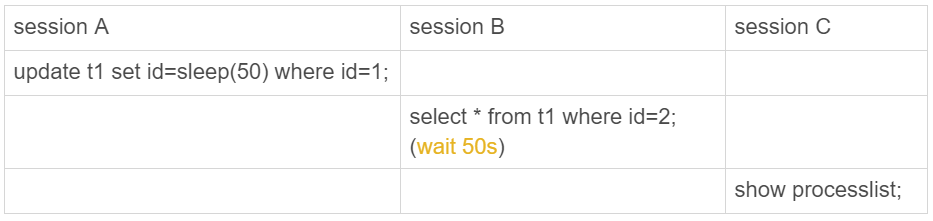
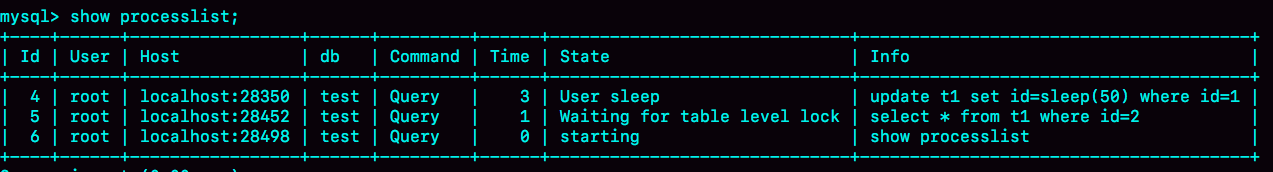


图 6 内存表的表锁--复现步骤

在这个执行序列里，sessionA的update语句需要执行50秒，这个语句执行期间sessionB的查询会进入锁等待状态，sessionC的show processlist结果输出如下

 图 7 内存表的表锁 -- 结果

跟行锁比起来，表锁对并发访问的支持不够好，所以，内存表的锁粒度问题，决定了它在处理并发事务的时候，性能也不会太好。

**数据持久性能问题**

数据放在内存中，是内存表的优势，但也是一个劣势，因为数据库重启的时候，所有的内存表都会被清空

如果数据库异常重启，内存表被清空不会有什么问题。但是高可用架构下，内存表的这个特点简直可以当做bug来看待，为什么这么说？

**看看M-S架构下，使用内存表存在的问题**

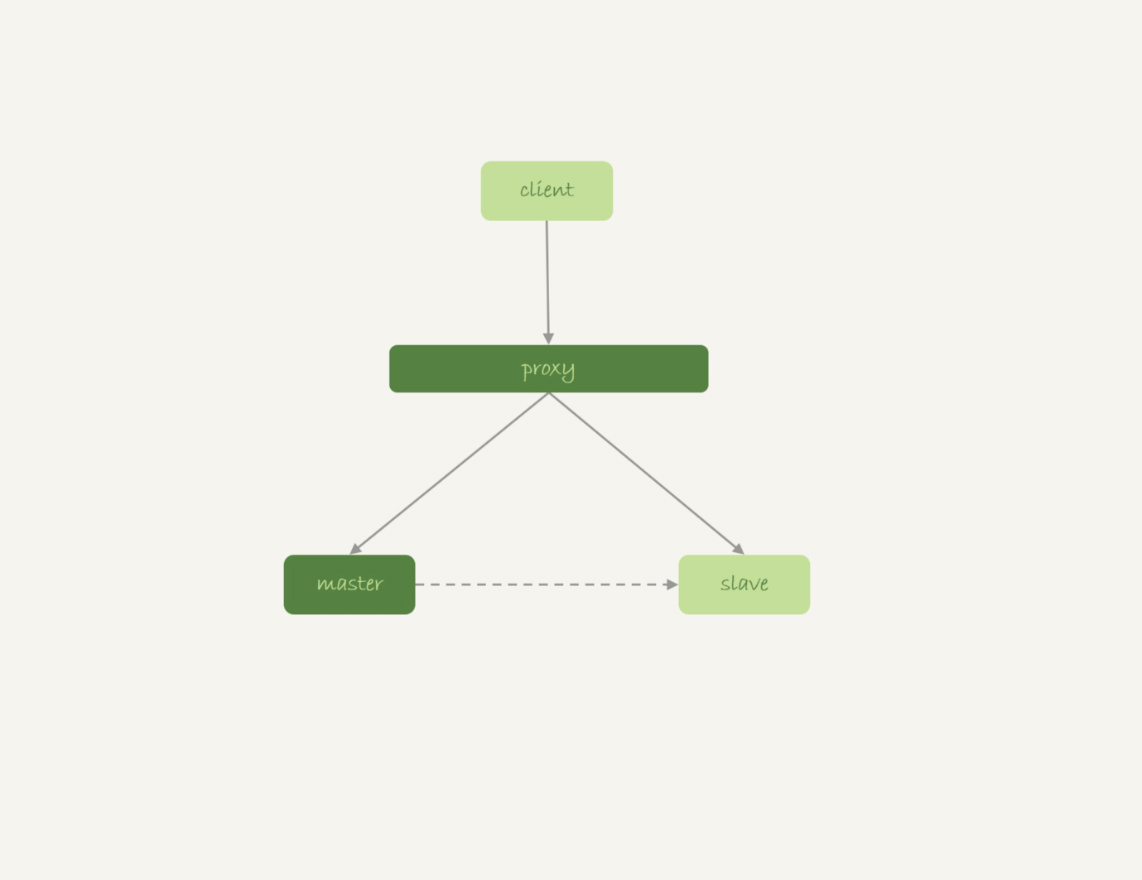


图 8 M-S基本架构

看看下面这个时序

1. 业务正常访问主库
2. 备库硬件升级，备库重启，内存表t1内容被清空
3. 备库重启后，客户端发送一条update语句,修改表t1的数据行，此时备库应用线程就会报“找不到要更新的行”

这样会导致主备同步停止，当然，如果此时发生主备切换的话，客户端会看到，表t1的数据“丢失”了

在图8中这种有proxy的架构里，默认主备切换的逻辑是由数据库系统自己维护的，这样对客户端来说，就是“网络断开”，重连之后，发现内存表数据丢失了。

可能会觉得还好，主备切换，连接会断开，业务端能够感知到异常

但是，接下来内存表的这个特性就会使用现象显得更“诡异”了，由于MySQL知道重启之后，内存表的数据丢失，所以，担心主库重启之后，出现主备不一致，MySQL在实现上做了这样一件事：在数据库重启之后，往binlog里写入一行DELETE FROM t1

**如果使用是如图9所示的双M结构**

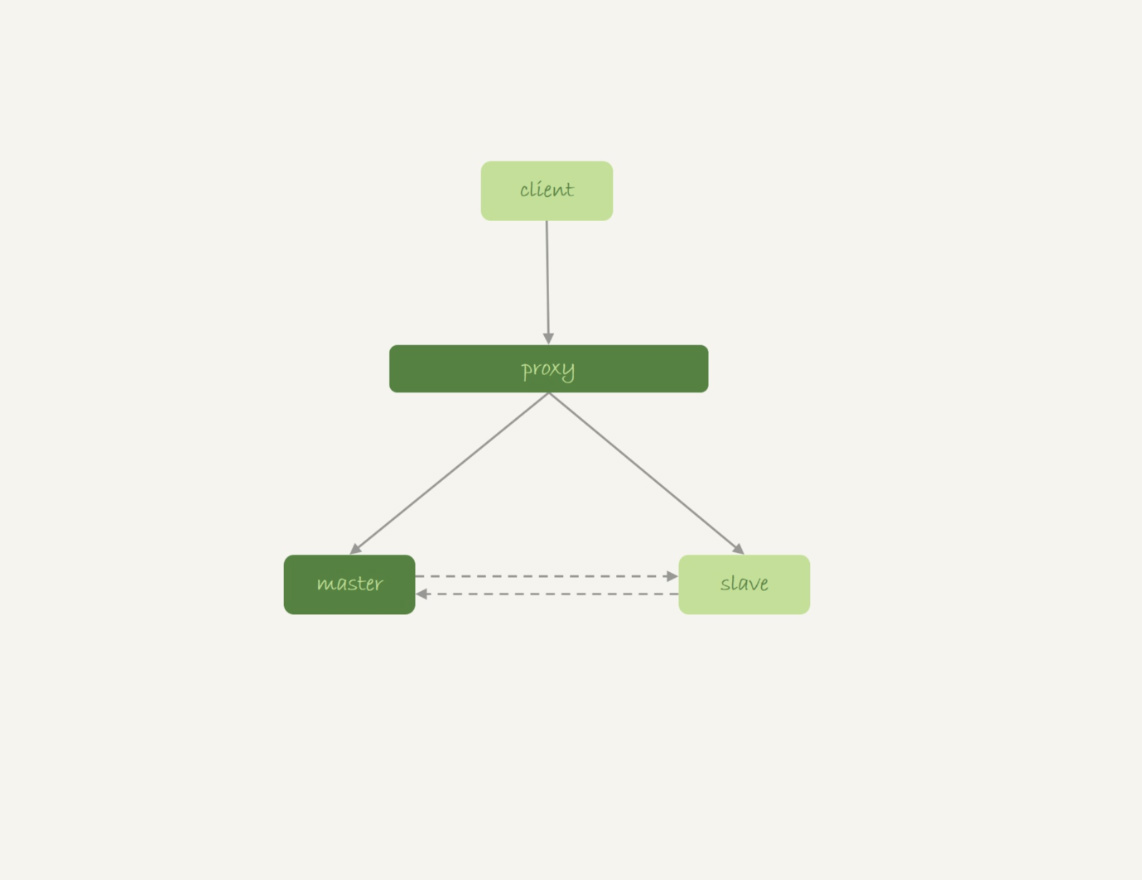


图 9 双M结构

在备库重启时，备库binlog的delete语句就会传到主库，然后把主库内存表的内容删除，这样在使用时候就会发现，主库的内存表数据突然被清空了。

基于上面的分析，内存表并不适合在生产环境上作为普通数据表使用

但是内存表的执行速度快，这个问题，可以这么分析

1. 如果表更新量大，那么并发度是一个很重要的参考指标，InnoDB支持行锁，并发度比内存表好
2. 能发到内存表的数据量都不大，如果考虑是读的性能，一个读QPS很高并且数据量不大的表，即使是使用InnoDB,数据也是都会缓存在InnoDB Buffer Pool里的，因此，使用InnoDB表的读性能也不会差

所以，**建议把普通内存表都用InnoDB表来代替，**但是，有一个场景确实例外

在35和36篇说到的用户临时表。在数量可控，不会耗费过多内存的情况下，可以考虑使用内存表

1. 临时表不会被其他线程访问，没有并发性的问题
2. 临时表重启后也是需要删除的，清空数据这个问题不存在
3. 备库的临时表也不会影响主库的用户线程

回顾35篇join语句优化的例子，创建一个innoDB临时表，语句序列是

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

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);

其实使用内存临时表效果更好，原因有三个

1. 相比于InnoDB表，使用内存表不需要写磁盘，往表temp\_t的写数据的速度更快
2. 索引b使用hash索引，查找的速度比B-Tree索引快
3. 临时表数据只有2000行，占用内存有限

因此，对35篇的语句序列做一个改写，将临时表temp\_t改成内存临时表，并且在字段b上创建一个hash索引

create temporary table temp\_t(id int primary key, a int, b int, index (b))engine=memory;

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);

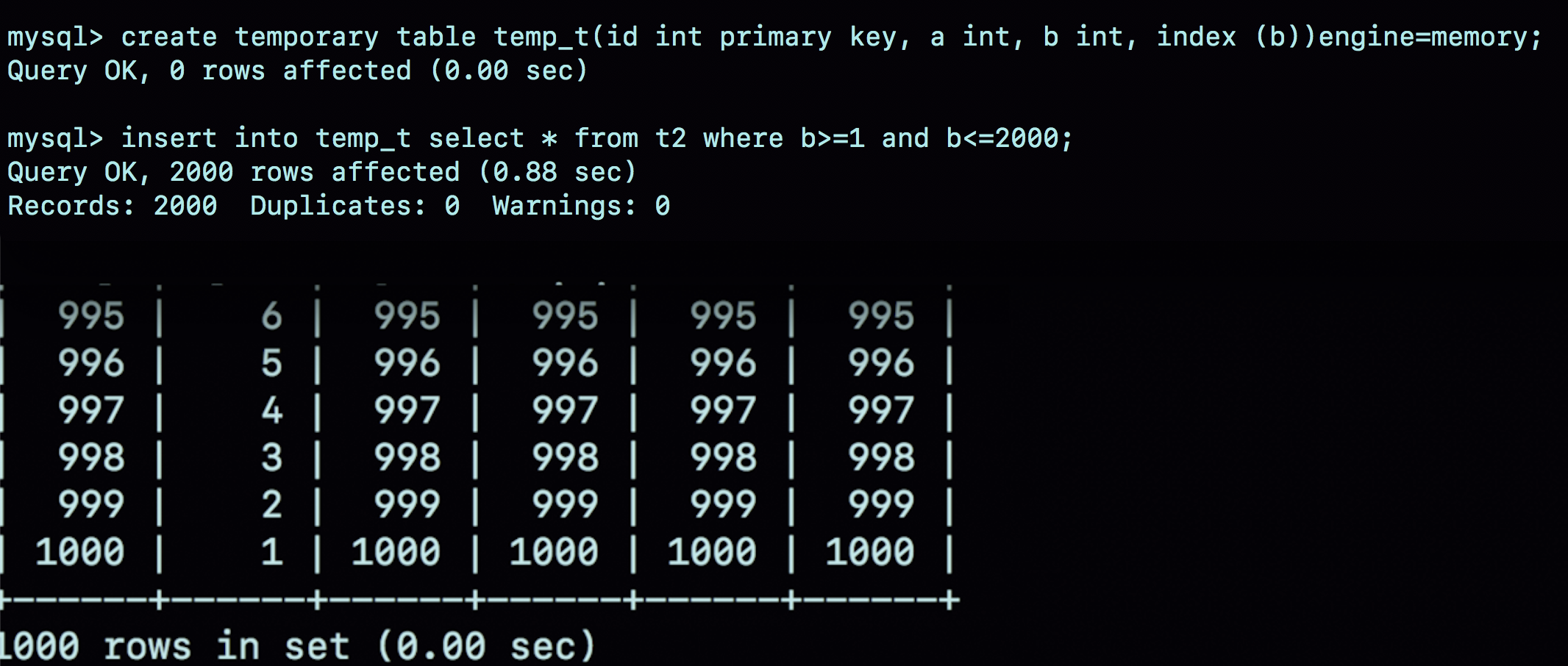


图 10 使用内存临时表的执行结果

不论是导入数据的时间，还是执行join的时间，使用内存临时表的速度都比使用InnoDB临时表要更快一些

**思考题**

一个数据库上，发现了一个内存表，备库重启之后肯定是会导致备库的内存表数据被清空，进而导致主备同步停止，此时，最好的选择是将他修改成InnoDB引擎表。

假设当时的业务暂时不允许修改引擎，可以有什么自动化逻辑，来避免主备同步停止呢？