# 14\_count(\*)这么慢，我该怎么办？

**Count(\*)的实现方式**

**在不同的MySQL引擎中，count(\*)有不同的实现方式**

**· MyISAM 引擎把一个表的总行数存在了磁盘上，因此执行count（\*）的时候会直接返回这个数，效率很高**

**·InnoDB 引擎就麻烦了，它执行count（\*）的时候，需要把数据一行一行的从引擎里面读取出来，然后累计计数**

此处讨论的事没有加过滤条件的count(\*),如果加了where 条件,MyISAM表也是不能返回得这么快。

**为什么InnoDB不跟MyISAM一样，把数字存起来呢？**

因为即使是同一个时刻的多个查询，由于多版本并发控制（MVCC）的原因，InnoDB表“应该返回多少行”也是不确定的，这里，用一个算count(\*)的例子来解释。

假设表T中 有10000条记录，我们设计了三个用户并行的会话

A,先启动事务并查询一次表的总行数

B,启动事务，插入一行记录后，查询表的总行数

C,先启动一个单独的语句，插入一行记录后，查询表的总行数

按时间顺序执行，同一行语句是在同一时刻执行的

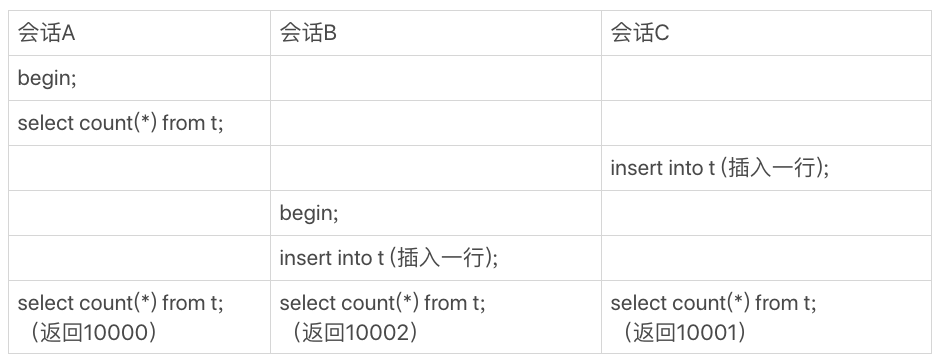


图 1 A,B,C的执行流程

三个会话A,B,C同时查询t的总行数，但拿到的结果却不同

这和InnoDB的事务有关系，可重复读是它的默认隔离级别，在代码上就是通过多版本并发控制，也就是mvcc来实现的，每一行记录都要判断自己的是否对这个会话可见，因此对于count(\*)来说，InnoDB只好把一行行的读出一次判断，可见的行才能够计算“基于这个查询”的表的总行数。

InnoDB是索引组织表，主键索引数的叶子节点是数据，而普通索引树的叶子节点是主键值。所以，普通索引树比主键索引树小的多。对于count(\*)这样的操作，遍历那个索引树得到结果逻辑上都是一致的。因此，MySQL优化器找到最下的那颗树来遍历。**在保证逻辑正确的前提下，尽量减少扫描的数量，是数据库系统设计的通用法则之一。**

用show table status命令的话，就会发现这个命令的输出结果里面有一个TABLE\_ROWS用于显示这个表当前有多少行，这个命令执行的挺快的，那这个TABLE\_ROWS能代替count(\*)吗

TABLE\_ROWS是通过采样估计出来的，误差最多40-50%，**所以，show table status命令显示的行数也不能直接使用**

**·MyISAM表虽然count(\*)快， 但不支持事务**

**·show tables status命令返回很快，但是不准确**

**·InnoDB表直接count(\*)会遍历表，虽然准确，但导致性能问题**

如果一个页面经常要显示系统的操作总记录数，到底该怎么办？答案是：自己计数

**用缓存系统保存计数**

对于更新频繁的库来说，第一时间会想到缓存系统来支持

Redis的数据不能永久不的保存在内存里，所以会找一个地方把这个值存起来，但即使这样，仍然可能丢失更新，试想如果，刚刚在数据表中插入一行，Redis中保存的值也加了1，然后Redis异常重启了，重启之后从存储redis数据的地方把这个值读回来，而刚刚加1的这个计数操作却丢失了。

当然了，这还是有解的，比如redis异常重启以后，到数据库里面单独执行一次count(\*)获取真实的行数，再把这个值写回到redis就可以了，异常重启毕竟不是经常发生，一次扫描全表，还是可以接收的

**将技术保存在缓存系统的方式，还不只是丢失更新的问题。即使Redis正常工作，这个值还是逻辑上不精确的。**

可以设想下有这么一个页面，