14讲count(*)这么慢，我该怎么办?

在开发系统的时候，你可能经常需要计算一个表的行数，比如一个交易系统的所有变更记录总数。这时候你可能会想，一条select count(*) from t 语句不就解决了吗?

但是，你会发现随着系统中记录数越来越多，这条语句执行得也会越来越慢。然后你可能就想了，MySQL怎么这么笨啊，记个总数，每次要查的时候直接读出来，不就好了吗。

那么今天，我们就来聊聊count(*)语句到底是怎样实现的，以及MySQL为什么会这么实现。然后，我会再和你说说，如果应用中有这种频繁变更并需要统计表行数的需求，业务设计上可以怎么做。

count(*)的实现方式

你首先要明确的是，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;
- 而InnoDB引擎就麻烦了，它执行count(*)的时候，需要把数据一行一行地从引擎里面读出来，然后累积计数。

这里需要注意的是，我们在这篇文章里讨论的是没有过滤条件的count(*)，如果加了where 条件的话，MyISAM表也是不能返回得这么快的。

在前面的文章中，我们一起分析了为什么要使用InnoDB，因为不论是在事务支持、并发能力还是在数据安全方面，InnoDB都优于MyISAM。我猜你的表也一定是用了InnoDB引擎。这就是当你的记录数越

来越多的时候，计算一个表的总行数会越来越慢的原因。

那**为什么InnoDB不跟MyISAM一样，也把数字存起来呢?**

这是因为即使是在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，我用一个算count(*)的例子来为你解释一下。

假设表t中现在有10000条记录，我们设计了三个用户并行的会话。

- 会话A先启动事务并查询一次表的总行数；
- 会话B启动事务，插入一行后记录后，查询表的总行数；
- 会话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。

我们假设从上到下是按照时间顺序执行的，同一行语句是在同一时刻执行的。

会话A	会话B	会话C
begin;		
select count(*) from t;		
		insert into t (插入一行);
	begin;	
	insert into t (插入一行);	
select count(*) from t; （返回10000）	select count(*) from t; （返回10002）	select count(*) from t; （返回10001）

图1 会话A、B、C的执行流程

你会看到，在最后一个时刻，三个会话A、B、C会同时查询表t的总行数，但拿到的结果却不同。

这和InnoDB的事务设计有关系，可重复读是它默认的隔离级别，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是否对这个会话可见，因此对于count(*)请求来说，InnoDB只好把数据一行一行地读出依次判断，可见的行才能够用于计算“基于这个查询”的表的总行数。

> 备注：如果你对MVCC记忆模糊了，可以再回顾下第3篇文章《事务隔离：为什么你改了我还看不见？》和第8篇文章《事务到底是隔离的还是不隔离的？》中的相关内容。

当然，现在这个看上去笨笨的MySQL，在执行count(*)操作的时候还是做了优化的。

你知道的，InnoDB是索引组织表，主键索引树的叶子节点是数据，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很多。对于count(*)这样的操作，遍历哪个索引树得到的结果逻

辑上都是一样的。因此，MySQL优化器会找到最小的那棵树来遍历。**在保证逻辑正确的前提下，尽量减少扫描的数据量，是数据库系统设计的通用法则之一。**

如果你用过show table status 命令的话，就会发现这个命令的输出结果里面也有一个TABLE_ROWS用于显示这个表当前有多少行，这个命令执行挺快的，那这个TABLE_ROWS能代替count(*)吗？

你可能还记得在第10篇文章《 MySQL为什么有时候会选错索引？》中我提到过，索引统计的值是通过采样来估算的。实际上，TABLE_ROWS就是从这个采样估算得来的，因此它也很不准。有多不准呢，官方文档说误差可能达到40%到50%。**所以，show table status命令显示的行数也不能直接使用。**

到这里我们小结一下：

- MyISAM表虽然count(*)很快，但是不支持事务；
- show table status命令虽然返回很快，但是不准确；
- InnoDB表直接count(*)会遍历全表，虽然结果准确，但会导致性能问题。

那么，回到文章开头的问题，如果你现在有一个页面经常要显示交易系统的操作记录总数，到底应该怎么办呢？答案是，我们只能自己计数。

接下来，我们讨论一下，看看自己计数有哪些方法，以及每种方法的优缺点有哪些。

这里，我先和你说一下这些方法的基本思路：你需要自己找一个地方，把操作记录表的行数存起来。

用缓存系统保存计数

对于更新很频繁的库来说，你可能会第一时间想到，用缓存系统来支持。

你可以用一个Redis服务来保存这个表的总行数。这个表每被插入一行Redis计数就加1，每被删除一行Redis计数就减1。这种方式下，读和更新操作都很快，但你再想一下这种方式存在什么问题吗？

没错，缓存系统可能会丢失更新。

Redis的数据不能永久地留在内存里，所以你会找一个地方把这个值定期地持久化存储起来。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了一行，Redis中保存的值也加了1，然后Redis异常重启了，重启后你要从存储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里就可以了。异常重启毕竟不是经常出现的情况，这一次全表扫描的成本，还是可以接受的。

但实际上，**将计数保存在缓存系统中的方式，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。**

你可以设想一下有这么一个页面，要显示操作记录的总数，同时还要显示最近操作的100条记录。那么，这个页面的逻辑就需要先到Redis里面取出计数，再到数据表里面取数据记录。

我们是这么定义不精确的：

1. 一种是，查到的100行结果里面有最新插入记录，而Redis的计数里还没加1；
2. 另一种是，查到的100行结果里没有最新插入的记录，而Redis的计数里已经加了1。

这两种情况，都是逻辑不一致的。

我们一起来看看这个时序图。

时刻	会话A	会话B
T1		
T2	插入一行数据R;	
T3		读Redis计数; 查询最近100条记录;
T4	Redis 计数加1;	

图2 会话A、B执行时序图

图2中，会话A是一个插入交易记录的逻辑，往数据表里插入一行R，然后Redis计数加1；会话B就是查询页面显示时需要的数据。

在图2的这个时序里，在T3时刻会话B来查询的时候，会显示出新插入的R这个记录，但是Redis的计数还没加1。这时候，就会出现我们说的数据不一致。

你一定会说，这是因为我们执行新增记录逻辑时候，是先写数据表，再改Redis计数。而读的时候是先读Redis，再读数据表，这个顺序是相反的。那么，如果保持顺序一样的话，是不是就没问题了？我们现在把会话A的更新顺序换一下，再看看执行结果。

时刻	会话A	会话B
T1		
T2	Redis 计数加1;	
T3		读Redis计数; 查询最近100条记录;
T4	插入一行数据R;	
T5		

图3 调整顺序后，会话A、B的执行时序图

你会发现，这时候反过来了，会话B在T3时刻查询的时候，Redis计数加了1了，但还查不到新插入的R这一行，也是数据不一致的情况。

在并发系统里面，我们是无法精确控制不同线程的执行时刻的，因为存在图中的这种操作序列，所以，我们说即使Redis正常工作，这个计数值还是逻辑上不精确的。

在数据库保存计数

根据上面的分析，用缓存系统保存计数有丢失数据和计数不精确的问题。那么，**如果我们把这个计数直接放到数据库里单独的一张计数表C中，又会怎么样呢?**

首先，这解决了崩溃丢失的问题，InnoDB是支持崩溃恢复不丢数据的。

> 备注：关于InnoDB的崩溃恢复，你可以再回顾一下第2篇文章《日志系统：一条SQL更新语句是如何执行的？》中的相关内容。

然后，我们再看看能不能解决计数不精确的问题。

你会说，这不一样吗？无非就是把图3中对Redis的操作，改成了对计数表C的操作。只要出现图3的这种执行序列，这个问题还是无解的吧？

这个问题还真不是无解的。

我们这篇文章要解决的问题，都是由于InnoDB要支持事务，从而导致InnoDB表不能把count(*)直接存起来，然后查询的时候直接返回形成的。

所谓以子之矛攻子之盾，现在我们就利用“事务”这个特性，把问题解决掉。

时刻	会话A	会话B
T1		
T2	begin; 表C中计数值加1;	
T3		begin; 读表C计数值; 查询最近100条记录; commit;
T4	插入一行数据R commit;	

图4 会话A、B的执行时序图

我们来看下现在的执行结果。虽然会话B的读操作仍然是在T3执行的，但是因为这时候更新事务还没有提交，所以计数值加1这个操作对会话B还不可见。

因此，会话B看到的结果里， 查计数值和"最近100条记录"看到的结果，逻辑上就是一致的。

不同的**count**用法

在前面文章的评论区，有同学留言问到：在select count(?) from t这样的查询语句里面，count(*)、count(主键id)、count(字段)和count(1)等不同用法的性能，有哪些差别。今天谈到了count(*)的性能问题，我就借此机会和你详细说明一下这几种用法的性能差别。

需要注意的是，下面的讨论还是基于InnoDB引擎的。

这里，首先你要弄清楚count()的语义。count()是一个聚合函数，对于返回的结果集，一行行地判断，如果count函数的参数不是NULL，累计值就加1，否则不加。最后返回累计值。

所以，count(*)、count(主键id)和count(1) 都表示返回满足条件的结果集的总行数；而count(字段），则表示返回满足条件的数据行里面，参数"字段"不为NULL的总个数。

至于分析性能差别的时候，你可以记住这么几个原则：

1. server层要什么就给什么；
2. InnoDB只给必要的值；

3. 现在的优化器只优化了count(*)的语义为“取行数”，其他“显而易见”的优化并没有做。

这是什么意思呢？接下来，我们就一个个地来看看。

对于count(主键id)来说，InnoDB引擎会遍历整张表，把每一行的id值都取出来，返回给server层。server层拿到id后，判断是不可能为空的，就按行累加。

对于count(1)来说，InnoDB引擎遍历整张表，但不取值。server层对于返回的每一行，放一个数字“1”进去，判断是不可能为空的，按行累加。

单看这两个用法的差别的话，你能对比出来，count(1)执行得要比count(主键id)快。因为从引擎返回id会涉及到解析数据行，以及拷贝字段值的操作。

对于count(字段)来说：

1. 如果这个“字段”是定义为not null的话，一行行地从记录里面读出这个字段，判断不能为null，按行累加；
2. 如果这个“字段”定义允许为null，那么执行的时候，判断到有可能是null，还要把值取出来再判断一下，不是null才累加。

也就是前面的第一条原则，server层要什么字段，InnoDB就返回什么字段。

但是count(*)是例外，并不会把全部字段取出来，而是专门做了优化，不取值。count(*)肯定不是null，按行累加。

看到这里，你一定会说，优化器就不能自己判断一下吗，主键id肯定非空啊，为什么不能按照count(*)来处理，多么简单的优化啊。

当然，MySQL专门针对这个语句进行优化，也不是不可以。但是这种需要专门优化的情况太多了，而且MySQL已经优化过count(*)了，你直接使用这种用法就可以了。

所以结论是：按照效率排序的话，count(字段)<count(主键id)<count(1)≈count(*)，所以我建议你，尽量使用count(*)。

小结

今天，我和你聊了聊MySQL中获得表行数的两种方法。我们提到了在不同引擎中count(*)的实现方式是不一样的，也分析了用缓存系统来存储计数值存在的问题。

其实，把计数放在Redis里面，不能够保证计数和MySQL表里的数据精确一致的原因，是**这两个不同的存储构成的系统，不支持分布式事务，无法拿到精确一致的视图。**而把计数值也放在MySQL中，就解决了一致性视图的问题。

InnoDB引擎支持事务，我们利用好事务的原子性和隔离性，就可以简化在业务开发时的逻辑。这也是InnoDB引擎备受青睐的原因之一。

最后，又到了今天的思考题时间了。

在刚刚讨论的方案中，我们用了事务来确保计数准确。由于事务可以保证中间结果不被别的事务读到，因此修改计数值和插入新记录的顺序是不影响逻辑结果的。但是，从并发系统性能的角度考虑，你觉得在这个事务序列里，应该先插入操作记录，还是应该先更新计数表呢？

你可以把你的思考和观点写在留言区里，我会在下一篇文章的末尾给出我的参考答案。感谢你的收听，也欢迎你把这篇文章分享给更多的朋友一起阅读。

上期问题时间

上期我给你留的问题是，什么时候使用alter table t engine=InnoDB会让一个表占用的空间反而变大。

在这篇文章的评论区里面，大家都提到了一个点，就是这个表，本身就已经没有空洞的了，比如说刚刚做过一次重建表操作。

在DDL期间，如果刚好有外部的DML在执行，这期间可能会引入一些新的空洞。

@飞翔 提到了一个更深刻的机制，是我们在文章中没说的。在重建表的时候，InnoDB不会把整张表占满，每个页留了1/16给后续的更新用。也就是说，其实重建表之后不是“最”紧凑的。

假如是这么一个过程：

1. 将表t重建一次；
2. 插入一部分数据，但是插入的这些数据，用掉了一部分的预留空间；
3. 这种情况下，再重建一次表t，就可能会出现问题中的现象。

评论区留言点赞板：

> @W_T 等同学提到了数据表本身紧凑的情况；
> @undiÞned 提了一个好问题， @帆帆帆帆帆帆帆帆 同学回答了这个问题；
> @陈飞 @郜 @wang chen wen 都提了很不错的问题，大家可以去看看。

精选留言

阿建

从并发系统性能的角度考虑，应该先插入操作记录，再更新计数表。

知识点在《行锁功过：怎么减少行锁对性能的影响？》

因为更新计数表涉及到行锁的竞争，先插入再更新能最大程度地减少了事务之间的锁等待，提升了并发度。

2018-12-14 01:44

作者回复

好几个同学说对，你第一个标明出处

2018-12-14 09:20

北天魔狼

老师说过：事务开启后，更新操作放到最后。较少锁等待时间的影响

2018-12-14 06:50

崔根禄

老师：

1.count(*) 不取值，

InnoDB还做遍历表的操作吗，也不用给server层返回值吗？

2.count(1) 不取值，

但是要遍历表。原文中：

“server 层对于返回的每一行，放一个数字”1”进去”

这个”返回的每一行”：到底返回的啥？是每一行记录吗？还是形式的返回空行，然后用1填充？

3. count(1),count(*),count(主键id)

这三个做比较，哪个会快？时间消耗在哪个环节？
是否遍历表；是否取值；返回给server层内容 细节上从哪个角度考虑？

2018-12-14 18:19

Bin
答：先插入操作记录，再更新计数表。
在InnoDB事务中，行锁是在需要的时候才加上，等到事务结束(commit)的时候才释放。
案例中的更新操作很多时候都是更新同一个数据对象，如果是先更新计数表，那么持有的锁时间会更长.

2018-12-14 18:34

某、人
老师我先问个本章之外的问题:
1.rr模式下,一张表上没有主键和唯一键,有二级索引c.如果是一张大表,删除一条数据delete t where c=1.在主库上利用二级索引,在根据虚拟的主键列回表删除还挺快.但是在备库上回放特别慢,而且状态是system lock,是因为binlog event里没有包含虚拟主键列.导致在备库回放的时候,必须全表扫描,耗时特别久?还是其他原因

2.回放过程中,在备库delete一条语句是被阻塞的,insert又是可以的,说明只在记录上的X锁没有gap锁。但是如果在主库session A begin,delete where c=1.在开启一个session B,在主库上操作也是delete阻塞,insert正常.不过等session A执行完成,不提交.insert都阻塞了,说明最后上了gap锁。有点没明白这儿的上锁逻辑是什么?

3.还有就是备库回放binlog,相对于主库的一条update语句流程来说,从库回放哪些流程是省略了的啊,server层的应该都省略了,应该主要是引擎层的回放,这里有点模糊从库是怎么回放的binlog event?
因为第一个问题从库回放的时候,从库上的二级索引貌似没起作用,直接就在聚簇索引上做的更新。

感谢老师
2018-12-14 15:10

作者回复

1. 对，这个是个bug, 从库上会全表扫描。MariaDB 的版本有解决这个问题。生产上我们最好不允许没有主键的表

2. 按照你问的,gap锁没问题了。delete 被锁是因为行锁吧。从库重放就是因为走全表扫描按行锁下来触发的

3. 出现这个问题肯定是binlog设置了row格式。
这样binlog里面有所有值。如果你有主键的话，就是主键查，没有的话...就是全表了

2018-12-14 15:28

斜面镜子 Bill
先插入操作纪录，再更新计数表，因为计数表相当于热点行，加锁时间需要考虑足够短！
2018-12-14 13:08

帆帆帆帆帆帆帆帆
如果字段上有索引，且字段非空，count(字段)的效率就不是最差的了吧。
2018-12-14 09:31

作者回复

还是的。

注意：count(id)也是可以使用普通索引的

2018-12-14 12:46

陈天境

碰到大部分情形都是带条件查询的count,，这个怎么解？

2018-12-14 08:54

作者回复

索引条件过滤完后还多少行？如果行数少（几百行？）就没关系直接执行了

2018-12-14 09:05

银太@巨益科技

实际场景中基本没遇到不加条件查行数的业务，这种有什么好的建议吗？

2018-12-14 22:36

丽丽

以前看书说可重复还是会出现幻读。但是现在A会话前后两次都读取1000。是不是MySQL 的可重复读已经不会出现幻读了？

2018-12-14 22:23

作者回复

不是... 这个跟幻读无关哦

2018-12-14 23:16

陈天境

索引条件过滤完后还多少行？如果行数少（几百行？）就没关系直接执行了

——还有上万行，这种怎么处理？

2018-12-14 21:06

作者回复

其实上万还好，如果再更多，就要考虑另外计数了，不过这种就没有固定方法了...

2018-12-14 22:24

果然如此

一、请问计数用这个MySQL+redis方案如何：

1.开启事务（程序中的事务）

2.MySQL插入数据

3.原子更新redis计数

4.如果redis更新成功提交事务，如果redis更新失败回滚事务。

二、.net和java程序代码的事务和MySQL事务是什么关系，有什么相关性？

2018-12-14 20:19

Ryoma

我的意思是InnoDB为什么不像MyISAM一样，把count(*)的结果存储在磁盘上；然后通过事务id这种机制，在查询时能够返回当前事务能看见数据的行数——这个只是我的脑洞

是不是因为这个count(*)的意义不大，所以在InnoDB没有做这个实现

2018-12-14 20:01

作者回复

对，只要就是不准，不满足一致性条件

2018-12-14 22:26

Justin

一直想问一个问题 如果在数据量很大的表中增加一个字段 innodb底层是怎么处理的呢？

2018-12-14 18:22

老师您好：

之前看到文章说在5.7对于count(*)做了一次内部handler函数调用，可以快速获得表总数，不需要去遍历所有行数，这种情况能给提一下吗，也和count(1)效果差不多吗？

2018-12-14 17:53

燕羽阳

count(*)操作

如果只有主键索引，由于主键索引下是行数据，会导致全表扫描，大量读盘。

如果有二级索引，由于二级索引下是主键和行字段，数据量很小。此时count会很快。

我这边测试没加二级索引，需要4分半

加了二级索引，只需要0.13秒

2018-12-14 17:52

坏淡一个

如果经过where条件筛选的数据，数据量仍然很大怎么办呢？比如管理台有很多查询条件，像省市县，只选了某个省，还有几百万的数据，虽然分页了，但还需要统计总数count(*)，速度很慢怎么办呢？这种情况一直没想到有什么完美的解决方案，只能从业务上入手吗？

2018-12-14 17:32

燕羽阳

count(*) 不可以直接统计主键索引的数量么？ 为何还要全表扫描？

2018-12-14 17:23

mongo

请问一个问题：我很想搞明白mysql这个程序的组成部分，最基本的增删改查功能的实现流程，内部的表文件和索引文件等都是怎么存储和怎么使用的。这个专栏很全面，但是我看的很吃力，很多基础知识我还没有掌握，我想请问有什么参考书籍推荐？

2018-12-14 16:55

念

解决效率问题只有上面说的那种写表加redis方法了吗，没有其他的方法了吗

2018-12-14 15:16

作者回复

额其实我是为了展开文章说明后面的方案，方法应该还有的，你建议一个

2018-12-14 16:54