09 为何count(*)这么慢？

更新时间：2019-08-15 11:38:33

> 书是人类进步的阶梯。
>
> ——高尔基

比如你维护着一张电商订单表，业务的需求是查找所有订单数，开发很快能写出对应的 SQL ：

```
select count(*) from order_01;
```

但你是否会发现，如果这张表很大后，这条 SQL 会非常耗时。

今天我们就一起重新认识下 count()，并想办法去优化这类 SQL。

老规矩，先创建测试表并写入数据。

```
use muke; /* 使用muke这个database */
drop table if exists t1; /* 如果表t1存在则删除表t1 */

CREATE TABLE `t1` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `a` int(11) DEFAULT NULL,
  `b` int(11) NOT NULL,
  `c` int(11) DEFAULT NULL,
  `d` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `idx_a` (`a`),
  KEY `idx_b` (`b`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8mb4;

drop procedure if exists insert_t1; /* 如果存在存储过程insert_t1，则删除 */
delimiter ;;
create procedure insert_t1() /* 创建存储过程insert_t1 */
begin
declare i int; /* 声明变量i */
set i=1; /* 设置i的初始值为1 */
while(i<=10000)do /* 对满足i<=10000的值进行while循环 */
insert into t1(a,b,c,d) values(i,i,i,i); /* 写入表t1中a、b两个字段，值都为i当前的值 */
set i=i+1; /* 将i加1 */
end while;
end;;
delimiter ; /* 创建批量写入10000条数据到表t1的存储过程insert_t1 */
call insert_t1(); /* 运行存储过程insert_t1 */

insert into t1(a,b,c,d) values (null,10001,10001,10001),(10002,10002,10002,10002);

drop table if exists t2; /* 如果表t2存在则删除表t2 */
create table t2 like t1; /* 创建表t2，表结构与t1一致 */
alter table t2 engine =myisam; /* 把t2表改为MyISAM存储引擎 */
insert into t2 select * from t1;  /* 把t1表的数据转到t2表 */

CREATE TABLE `t3` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `a` int(11) DEFAULT NULL,
  `b` int(11) NOT NULL,
  `c` int(11) DEFAULT NULL,
  `d` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  CHARSET=utf8mb4;
insert into t3 select * from t1;  /* 把t1表的数据转到t3表 */
```

1 重新认识 count()

1.1 count(a) 和 count(*) 的区别

当 count() 统计某一列时，比如 count(a)，a 表示列名，是不统计 null 的。

比如测试表 t1，我们插入了字段 a 为 null 的数据，我们来对 a 做一次 count()：

```
select count(a) from t1;
```

```
mysql> select count(a) from t1;
+----------+
| count(a) |
+----------+
|    10001 |
+----------+
1 row in set (0.01 sec)
```

实际在数据写入时，写入了 10002 行数据。因此，对 a 字段为 null 的这一行不做统计。

而 count(*) 无论是否包含空值，都会统计。

我们对测试表 t1 执行一次 count(*):

```
select count(*) from t1;
```

```
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|    10002 |
+----------+
1 row in set (0.01 sec)
```

显然，统计的是所有的行。因此，如果希望知道结果集的行数，最好使用 count(*)。

1.2 MyISAM 引擎和 InnoDB 引擎 count(*) 的区别

对于 MyISAM 引擎，如果没有 where 子句，也没检索其它列，那么 count(*) 将会非常快。因为 MyISAM 引擎会把表的总行数存在磁盘上。

首先我们看下对 t2 表（存储引擎为 MyISAM）不带 where 子句做 count(*) 的执行计划：

```
explain select count(*) from t2;
```

```
mysql> explain select count(*) from t2;
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-----------------------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra                       |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-----------------------------+
|  1 | SIMPLE      | NULL  | NULL       | NULL | NULL          | NULL | NULL    | NULL | NULL |     NULL | Select tables optimized away |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-----------------------------+
1 row in set, 1 warning (0.00 sec)
```

在 Extra 字段发现 “Select tables optimized away” 关键字，表示是从 MyISAM 引擎维护的准确行数上获取到的统计值。

而 InnoDB 并不会保留表中的行数，**因为并发事务可能同时读取到不同的行数**。所以执行 count(*) 时都是临时去计算的，会比 MyISAM 引擎慢很多。

我们看下对 t1 表（存储引擎为 InnoDB）执行 count(*) 的执行计划：

```
mysql> explain select count(*) from t1;
+----+-------------+-------+------------+-------+---------------+-------+---------+------+-------+----------+-------------+
| id | select_type | table | partitions | type  | possible_keys | key   | key_len | ref  | rows  | filtered | Extra       |
+----+-------------+-------+------------+-------+---------------+-------+---------+------+-------+----------+-------------+
|  1 | SIMPLE      | t1    | NULL       | index | NULL          | idx_b | 4       | NULL | 10109 |   100.00 | Using index |
+----+-------------+-------+------------+-------+---------------+-------+---------+------+-------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
```

发现使用的是 b 字段的索引 idx_b，并且扫描行数是10109，表示会遍历 b 字段的索引树去计算表的总量。

对比 MyISAM 引擎和 InnoDB 引擎 count(*) 的区别，可以知道：

- MyISAM 会维护表的总行数，放在磁盘中，如果有 count(*) 的需求，直接返回这个数据
- 但是 InnoDB 就会去遍历普通索引树，计算表数据总量

在上面这个例子，InnoDB 表 t1 在执行 count(*) 时，为什么会走 b 字段的索引而不是走主键索引呢？下面我们分析下：

1.3 MySQL 5.7.18 前后 count(*) 的区别

在 MySQL 5.7.18 之前，InnoDB 通过扫描聚簇索引来处理 count(*) 语句。

从 MySQL 5.7.18 开始，通过遍历最小的可用二级索引来处理 count(*) 语句。如果不存在二级索引，则扫描聚簇索引。但是，如果索引记录不完全在缓存池中的话，处理 count(*) 也是比较久的。

新版本为什么会使用二级索引来处理 count(*) 语句呢？

原因是 InnoDB 二级索引树的叶子节点上存放的是主键，而主键索引树的叶子节点上存放的是整行数据，所以二级索引树比主键索引树小。因此优化器基于成本的考虑，优先选择的是二级索引。所以 count(主键) 其实没 count (*) 快。

1.4 count(1) 比 count(*) 快吗？

在前面我们知道 count(*) 无论是否包含空值，所有结果都会统计。

而 count(1)中的 1 是恒真表达式，因此也会统计所有结果。

所以 count(1) 和 count(*) 统计结果没差别。

我们来对比 count(1) 和 count(*) 的执行计划：

```
mysql> explain select count(1) from t1;
+----+-------------+-------+------------+-------+---------------+-------+---------+------+-------+----------+-------------+
| id | select_type | table | partitions | type  | possible_keys | key   | key_len | ref  | rows  | filtered | Extra       |
+----+-------------+-------+------------+-------+---------------+-------+---------+------+-------+----------+-------------+
|  1 | SIMPLE      | t1    | NULL       | index | NULL          | idx_b | 4       | NULL | 10109 |   100.00 | Using index |
+----+-------------+-------+------------+-------+---------------+-------+---------+------+-------+----------+-------------+
1 row in set, 1 warning (0.00 sec)

mysql> explain select count(*) from t1;
+----+-------------+-------+------------+-------+---------------+-------+---------+------+-------+----------+-------------+
| id | select_type | table | partitions | type  | possible_keys | key   | key_len | ref  | rows  | filtered | Extra       |
+----+-------------+-------+------------+-------+---------------+-------+---------+------+-------+----------+-------------+
|  1 | SIMPLE      | t1    | NULL       | index | NULL          | idx_b | 4       | NULL | 10109 |   100.00 | Using index |
+----+-------------+-------+------------+-------+---------------+-------+---------+------+-------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
```

执行计划一样，所以 count(1) 并不比 count(*) 快。

重新认识 count() 之后，你是否有了一些 count() 的优化思路呢？

下面一起讨论下 count() 优化：

2 哪些方法可以加快 count()

2.1 show table status

有时，我们只需要知道某张表的大概数据量，这种情况就可以使用 show table status，具体用法如下：

```
show table status like 't1';
```

```
mysql> show table status like 't1'\G
*************************** 1. row ***************************
           Name: t1
         Engine: InnoDB
        Version: 10
     Row_format: Dynamic
           Rows: 10109
 Avg_row_length: 45
    Data_length: 458752
Max_data_length: 0
   Index_length: 344064
      Data_free: 0
 Auto_increment: 10006
    Create_time: 2019-07-04 08:57:14
    Update_time: 2019-07-04 08:59:31
     Check_time: NULL
      Collation: utf8mb4_general_ci
       Checksum: NULL
 Create_options:
        Comment:
1 row in set (0.00 sec)
```

如上图，Rows 这列就表示这张表的行数。这种方式获取 InnoDB 表的行数非常快。

但是，这个值是个估算值，可能与实际值相差 40% 到 50%。（对于 Rows 这个字段更详细的解释，可以参考官方手册：https://dev.mysql.com/doc/refman/5.7/en/show-table-status.html）

所以，如果需要比较精确的表记录总数，此方法就行不通了。

2.2 用 Redis 做计数器

在有些业务场景，对于某一张表，count() 可能会频繁用到，直接执行 count(*) 可能会比较慢，并且影响数据库性能；使用 show table status 又不准确，此时可以考虑结合 Redis 做计数器。用法大致如下：

首先初始化时，执行一次精确计数：

```
select count(*) from t1;
```

```
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|    10002 |
+----------+
1 row in set (0.00 sec)
```

表此时的总数是 10002，把这个值赋给 Redis 中一个 key，命令如下：

```
set  t1_count  10002
```

```
127.0.0.1:7000> set t1_count 10002
OK
```

当表 t1 写入一条数据时：

```
insert into t1(a,b,c,d) values (10003,10003,10003,10003);
```

把 Redis 中 t1_count 这个 key 的值加 1，命令如下：

```
INCR t1_count
```

```
127.0.0.1:7000> INCR t1_count
(integer) 10003
```

当表 t1 删除一条数据时：

```
delete from t1 where id=10003;
```

把 Redis 中 t1_count 这个 key 的值减 1，命令如下：

```
DECR t1_count
```

```
127.0.0.1:7000> DECR t1_count
(integer) 10002
```

而业务需要查找表 t1 数据量时，只要到 Redis 中执行：

```
get t1_count
```

```
127.0.0.1:7000> get t1_count
"10002"
```

> 这里对 **Redis** 的计数做一些补充：
>
> INCR t1_count 表示为键 t1_count 存储的数字值加 1
> DECR t1_count 表示为键 t1_count 存储的数字值减 1
>
> 如果一次需要增加或者删除多行，用法如下：
>
> ```
> INCRBY t1_count 10
> ```
>
> 表示一次为键 t1_count 存储的数字值加 10。
>
> ```
> DECRBY t1_count 10
> ```
>
> 表示一次为键 t1_count 存储的数字值减 10。

通过 Redis 计数的方式，获取表的数据量比 show table status 准确，并且速度也比较快。

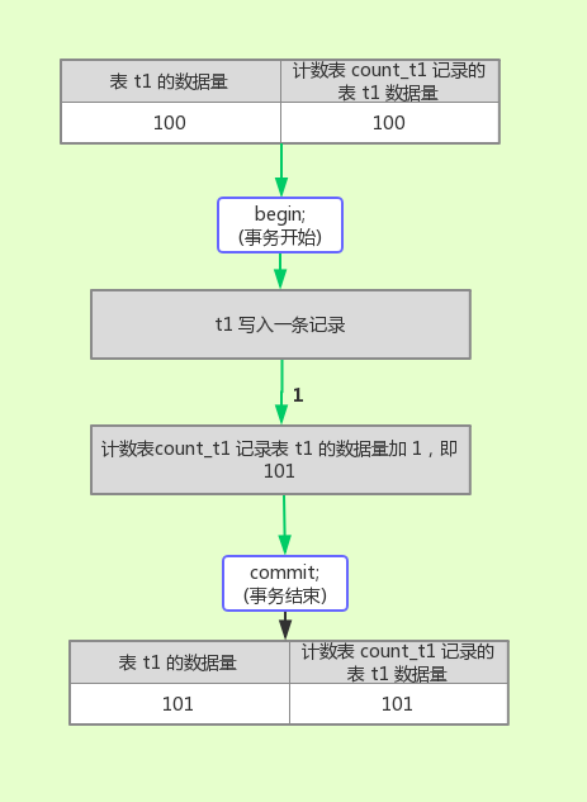
但是这种方法还是有缺点的。试想，在表 t1 写入数据到 Redis ，再到把 t1_count 加 1，总会存在一个时间差，如果这中间另外一个 session 去读取 Redis 中 t1_count 的值，此时 t1_count 的值没增加，但是表的实际数据行已经增加了，是不是结果就不准确了呢？我们在看下有没有更好的办法？

2.3 增加计数表

还是按照 2.2 中的方式，只是计数这一步操作我们用 MySQL 中一张 InnoDB 表来代替。

而数据写入操作和计数操作都放在一个事务中，就可以避免 2.2 中出现计数不准确的情况。

我们通过下图来看下计数整个过程：



因为放在同一个事务里，在图中 1 这个位置点，因为事务还没提交，所以表 t1 写入一条记录本身就对其它 session 不可见，此时其它 session 去执行 select count(*) from t1 和查计数表 count_t1 的记录都是一样的，为 101 。不会出现用 Redis 计数时，表实际总数与计数器的值不一致的情况。

3 总结

本节首先讲解了 count(a) 和 count(*) 的区别。曾经遇到过这种情况，某个同事想要统计表的数据总量，因为考虑到某个字段（比如字段名就是 a 吧）有索引，就写成了 select count(a)，碰巧 a 字段存在 null，导致结果不准确。

然后对比了 MyISAM 引擎和 InnoDB 引擎 count(*) 的区别，也说明了为什么 MyISAM 引擎执行count(*) 可以这么快，并提到了使用二级索引来处理 count(*) 语句比使用主键索引处理 count(*) 效率更高。还有就是 count(1) 和 count(*) 其实执行效率差不多。

后面提到几种优化 count() 的方式：

- show table status：能快速获取结果，但是结果不准确；
- 用 Redis 做计数器：能快速获取结果，比 show table status 结果准确，但是并发场景计数可能不准确；
- 增加 InnoDB 计数表：能快速获取结果，利用了事务特性确保了计数的准确，也是比较推荐的方法。

4 问题

对于本节的测试表 t1，我们如果按照下面这条 SQL 统计表的总数据量，得到的值会准确吗？

```
select count(*) from t1 force index (idx_a);
```

注意，a 字段存在 null。

你可以通过实验验证一下，欢迎将你的理解写在留言中。

5 参考资料

《高性能 MySQL》（第三版）：6.7.1 优化 COUNT() 查询

《MySQL 5.7 Reference Manual》：14.6.1.6 Limits on InnoDB Tables

《MySQL 5.7 Reference Manual》：12.20.1 Aggregate (GROUP BY) Function Descriptions

《MySQL 5.7 Reference Manual》：13.7.5.36 SHOW TABLE STATUS Syntax

}

08 Join语句可以这样优化

10 为什么添加索引能提高查询速度?