

# count(\*) 和 count(1) 有什么区别？哪个性能最好？

大家好，我是小林。

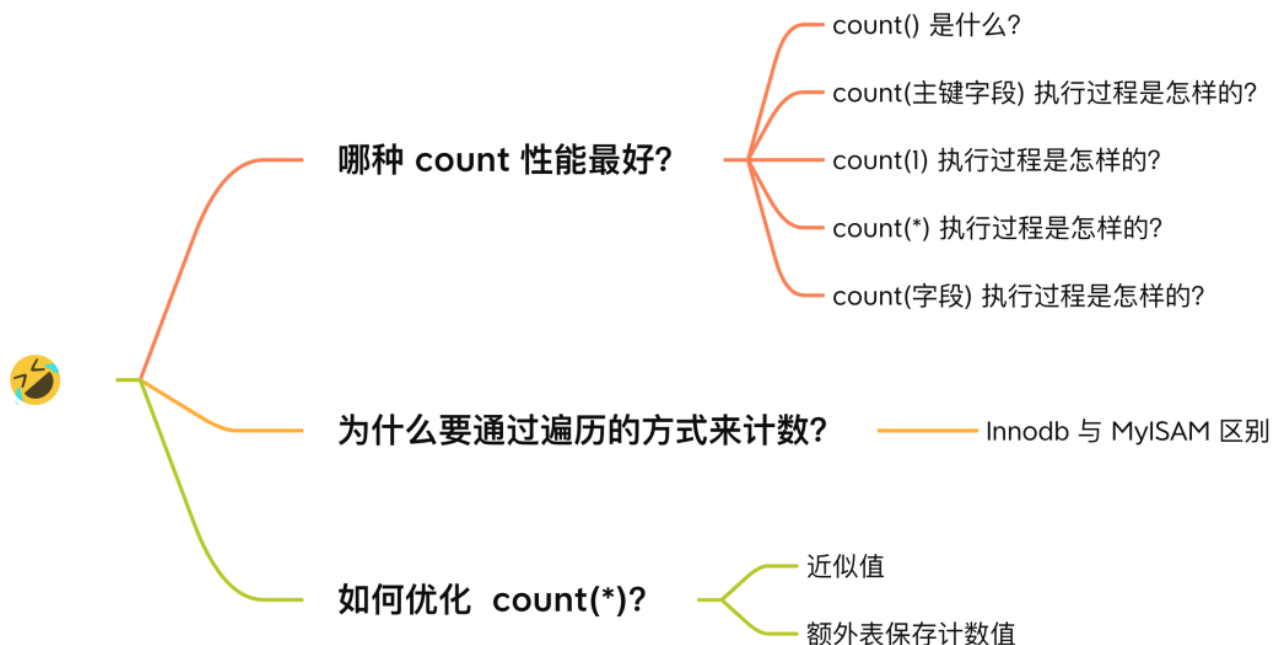
当我们对一张数据表中的记录进行统计的时候，习惯都会使用 count 函数来统计，但是 count 函数传入的参数有很多种，比如 count(1)、count( \* )、count(字段) 等。

到底哪种效率是最好的呢？是不是 count( \* ) 效率最差？

我曾经以为 count( \* ) 是效率最差的，因为认知上 `selete * from t` 会读取所有表中的字段，所以凡事带有 \* 字符的就觉得会读取表中所有的字段，当时网上有很多博客也这么说。

但是，当我深入 count 函数的原理后，被啪啪啪的打脸了！

不多说，发车！



## 哪种 count 性能最好？

我先直接说结论：

按照性能排序：  
**count(\*) = count(1) > count(主键字段) > count(字段)**

要弄明白这个，我们得要深入 count 的原理，以下内容基于常用的 innodb 存储引擎来说明。

## count() 是什么？

count() 是一个聚合函数，函数的参数不仅可以是字段名，也可以是其他任意表达式，该函数作用是统计符合查询条件的记录中，函数指定的参数不为 NULL 的记录有多少个。

假设 count() 函数的参数是字段名，如下：

```
select count(name) from t_order;
```

这条语句是统计「t\_order 表中，name 字段不为 NULL 的记录」有多少个。也就是说，如果某一条记录中的 name 字段的值为 NULL，则就不会被统计进去。

再来假设 count() 函数的参数是数字 1 这个表达式，如下：

```
select count(1) from t_order;
```

这条语句是统计「t\_order 表中，1 这个表达式不为 NULL 的记录」有多少个。

1 这个表达式就是单纯数字，它永远都不是 NULL，所以上面这条语句，其实是在统计 t\_order 表中有多少个记录。

## count(主键字段) 执行过程是怎样的？

在通过 count 函数统计有多少个记录时，MySQL 的 server 层会维护一个名叫 count 的变量。

server 层会循环向 InnoDB 读取一条记录，如果 count 函数指定的参数不为 NULL，那么就会将变量 count 加 1，直到符合查询的全部记录被读完，就退出循环。最后将 count 变量的值发送给客户端。

InnoDB 是通过 B+ 树来保存记录的，根据索引的类型又分为聚簇索引和二级索引，它们区别在于，聚簇索引的叶子节点存放的是实际数据，而二级索引的叶子节点存放的是主键值，而不是实际数据。

用下面这条语句作为例子：

```
//id 为主键值
select count(id) from t_order;
```

如果表里只有主键索引，没有二级索引时，那么，InnoDB 循环遍历聚簇索引，将读取到的记录返回给 server 层，然后读取记录中的 id 值，就会 id 值判断是否为 NULL，如果不为 NULL，就将 count 变量加 1。




The screenshot shows a MySQL Query Client window with the following elements:

- Host: 127.0.0.01
- Database: test
- Query: `1 explain select count(id) from t_order;`
- Result: Result 1
- Profile: Profile
- Status: Status
- Tab: 主键索引 (Primary Index)

id	select_type	table	partitions	type	possible_keys	key	key_len
1	SIMPLE	t_order	(NULL)	index	(NULL)	PRIMARY	4

但是，如果表里有二级索引时，InnoDB 循环遍历的对象就不是聚簇索引，而是二级索引。



The screenshot shows a MySQL Query Client window with the following elements:

- Host: 127.0.0.01
- Database: test
- Query: `1 explain select count(id) from t_order;`
- Result: Result 1
- Profile: Profile
- Status: Status
- Tab: 二级索引 (Secondary Index)

id	select_type	table	partitions	type	possible_keys	key	key_len	ref
1	SIMPLE	t_order	(NULL)	index	(NULL)	index_order	1023	(NULL)

这是因为相同数量的二级索引记录可以比聚簇索引记录占用更少的存储空间，所以二级索引树比聚簇索引树小，这样遍历二级索引的 I/O 成本比遍历聚簇索引的 I/O 成本小，因此「优化

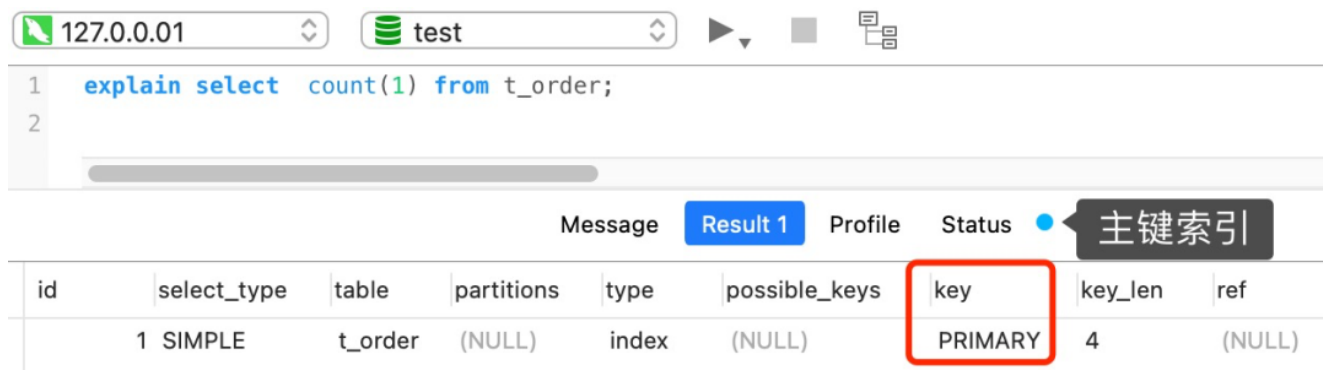
器」优先选择的是二级索引。

## count(1) 执行过程是怎样的？

用下面这条语句作为例子：

```
select count(1) from t_order;
```

如果表里只有主键索引，没有二级索引时。



The screenshot shows a MySQL Query Client window with the following details:

- Host: 127.0.0.01
- Database: test
- Query: `1 explain select count(1) from t_order;`
- Tab: Result 1
- Table: 

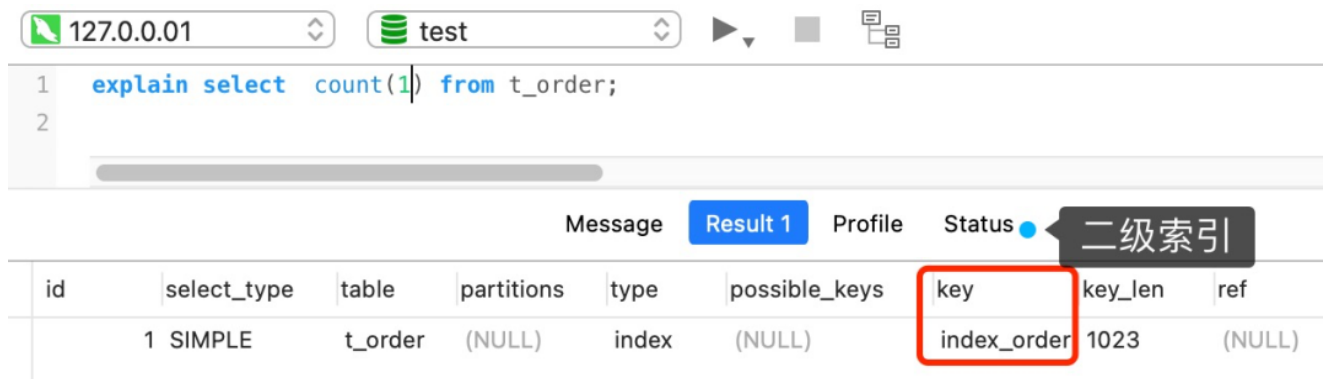
id	select_type	table	partitions	type	possible_keys	key	key_len	ref
1	SIMPLE	t_order	(NULL)	index	(NULL)	PRIMARY	4	(NULL)

A red box highlights the 'key' column value 'PRIMARY', and a callout bubble points to it with the text '主键索引' (Primary Index).

那么，InnoDB 循环遍历聚簇索引（主键索引），将读取到的记录返回给 server 层，**但是不会读取记录中的任何字段的值**，因为 count 函数的参数是 1，不是字段，所以不需要读取记录中的字段值。参数 1 很明显并不是 NULL，因此 server 层每从 InnoDB 读取到一条记录，就将 count 变量加 1。

可以看到，count(1) 相比 count(主键字段) 少一个步骤，就是不需要读取记录中的字段值，所以通常会说 count(1) 执行效率会比 count(主键字段) 高一点。

但是，如果表里有二级索引时，InnoDB 循环遍历的对象就二级索引了。



The screenshot shows a MySQL Query Client window with the following details:

- Host: 127.0.0.01
- Database: test
- Query: `1 explain select count(1) from t_order;`
- Tab: Result 1
- Table: 

id	select_type	table	partitions	type	possible_keys	key	key_len	ref
1	SIMPLE	t_order	(NULL)	index	(NULL)	index_order	1023	(NULL)

A red box highlights the 'key' column value 'index\_order', and a callout bubble points to it with the text '二级索引' (Secondary Index).

## count(\*) 执行过程是怎样的？

看到 \* 这个字符的时候，是不是大家觉得是读取记录中的所有字段值？

对于 `selete *` 这条语句来说是这个意思，但是在 `count(*)` 中并不是这个意思。

**count( \\* ) 其实等于 count( 0 )**，也就是说，当你使用 `count( * )` 时，MySQL 会将 \* 参数转化为参数 0 来处理。

```
1 explain select count(*) from t_order;
2 show warnings;
```

	Message	Result 1	Result 2	Profile	Status
Level	Code	Message			
Note	1003	/* select#1 */ select count(0) AS `count(*)` from `test`.`t_order`			

所以，**count(\*) 执行过程跟 count(1) 执行过程基本一样的**，性能没有什么差异。

在 MySQL 5.7 的官方手册中有这么一句话：

*InnoDB handles SELECT COUNT( \\* ) and SELECT COUNT( 1 ) operations in the same way. There is no performance difference.*

翻译：InnoDB以相同的方式处理SELECT COUNT ( \\* ) 和SELECT COUNT ( 1 ) 操作，没有性能差异。

而且 MySQL 会对 `count(*)` 和 `count(1)` 有个优化，如果有多个二级索引的时候，优化器会使用 `key_len` 最小的二级索引进行扫描。

只有当没有二级索引的时候，才会采用主键索引来进行统计。

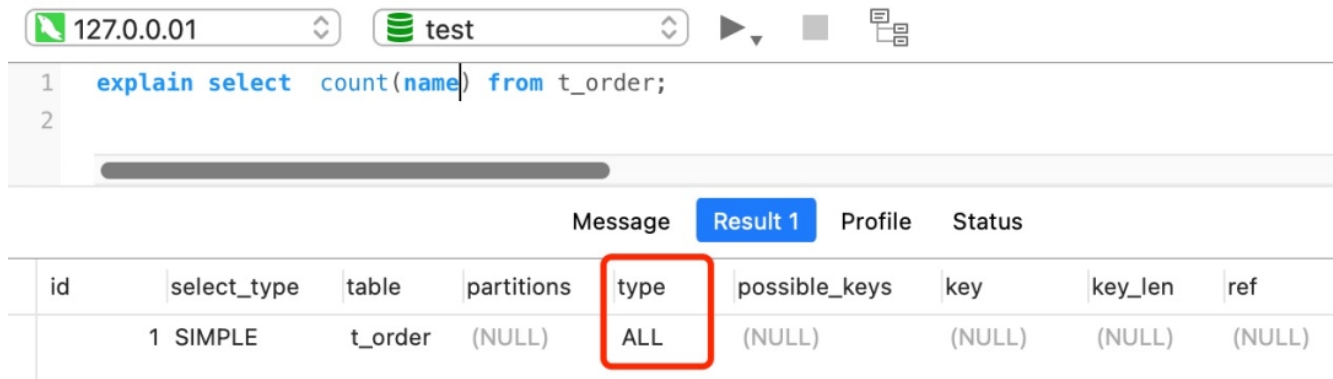
## count(字段) 执行过程是怎样的？

`count(字段)` 的执行效率相比前面的 `count(1)`、`count(*)`、`count(主键字段)` 执行效率是最差的。

用下面这条语句作为例子：

```
// name不是索引，普通字段
select count(name) from t_order;
```

对于这个查询来说，会采用全表扫描的方式来计数，所以它的执行效率是比较差的。



Message Result 1 Profile Status									
id	select_type	table	partitions	type	possible_keys	key	key_len	ref	
1	SIMPLE	t_order	(NULL)	ALL	(NULL)	(NULL)	(NULL)	(NULL)	

## 小结

count(1)、count(\*)、count(主键字段)在执行的时候，如果表里存在二级索引，优化器就会选择二级索引进行扫描。

所以，如果要执行 count(1)、count(\*)、count(主键字段) 时，尽量在数据表上建立二级索引，这样优化器会自动采用 key\_len 最小的二级索引进行扫描，相比于扫描主键索引效率会高一些。

再来，就是不要使用 count(字段) 来统计记录个数，因为它的效率是最差的，会采用全表扫描的方式来统计。如果你非要统计表中该字段不为 NULL 的记录个数，建议给这个字段建立一个二级索引。

## 为什么要通过遍历的方式来计数？

你可能会好奇，为什么 count 函数需要通过遍历的方式来统计记录个数？

我前面将的案例都是基于 InnoDB 存储引擎来说明的，但是在 MyISAM 存储引擎里，执行 count 函数的方式是不一样的，通常在没有任何查询条件下的 count(\*)，MyISAM 的查询速度要明显快于 InnoDB。

使用 MyISAM 引擎时，执行 count 函数只需要  $O(1)$  复杂度，这是因为每张 MyISAM 的数据表都有一个 meta 信息有存储了 row\_count 值，由表级锁保证一致性，所以直接读取 row\_count 值就是 count 函数的执行结果。

而 InnoDB 存储引擎是支持事务的，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，所以无法像 MyISAM 一样，只维护一个 row\_count 变量。

举个例子，假设表 t\_order 有 100 条记录，现在有两个会话并行以下语句：

会话A	会话B
begin;	
select * from t_order (返回100)	
	insert into t_order (插入一条记录)
select * from t_order (返回100)	select * from t_order (返回101)

在会话 A 和会话 B 的最后一个时刻，同时查表 t\_order 的记录总个数，可以发现，显示的结果是不一样的。所以，在使用 InnoDB 存储引擎时，就需要扫描表来统计具体的记录。

而当带上 where 条件语句之后，MyISAM 跟 InnoDB 就没有区别了，它们都需要扫描表来进行记录个数的统计。

## 如何优化 count(\*)?

如果对一张大表经常用 count(\*) 来做统计，其实是很不好的。

比如下面我这个案例，表 t\_order 共有 1200+ 万条记录，我也创建了二级索引，但是执行一次 select count(\*) from t\_order 要花费差不多 5 秒！

127.0.0.01 test

```
1 select count(*) from t_order;
2 |
```

Message Result 1 Profile Status

count(*)
12746284

+ - ✓ ✕ select count(\*)... 4.785s elapsed

面对大表的记录统计，我们有没有什么其他更好的办法呢？

## 第一种，近似值

如果你的业务对于统计个数不需要很精确，比如搜索引擎在搜索关键词的时候，给出的搜索结果条数是一个大概值。

Google mysql

全部 图片 图书 视频 新闻 更多

找到约 192,000,000 条结果 (用时 0.51 秒)

<https://www.mysql.com> 翻译此页

**MySQL**

MySQL Database Service is a fully managed database service to deploy cloud-native applications. HeatWave, an integrated, high-performance query accelerator ...

这时，我们就可以使用 `show table status` 或者 `explain` 命令来表进行估算。

执行 `explain` 命令效率是很高的，因为它并不会真正的去查询，下图中的 `rows` 字段值就是 `explain` 命令对表 `t_order` 记录的估算值。



127.0.0.01

test

1

2

explain select count(\*) from t\_order;

Message

Result 1

Profile

Status

table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
t_order	(NULL)	index	(NULL)	PRIMARY	4	(NULL)	12707308	100.00	Using index

+ - ✓ ✕

explain select c...

0.004s elapsed

## 第二种，额外表保存计数值

如果是想精确的获取表的记录总数，我们可以将这个计数值保存到单独的一张计数表中。

当我们在数据表插入一条记录的同时，将计数表中的计数字段 + 1。也就是说，在新增和删除操作时，我们需要额外维护这个计数表。