# 【译】优化MySQL中的分页

| Giraffe's Home http://yemengying.com/2016/05/28/optimized-pagiantion-mysql/

翻译一篇关于优化MySQL中的分页的文章，原文地址：[Optimized Pagination using MySQL](http://www.xarg.org/2011/10/optimized-pagination-using-mysql/)，谢谢，3Q，康桑阿米达~~~

很多应用往往只展示最新或最热门的几条记录，但为了旧记录仍然可访问，所以就需要个分页的导航栏。然而，如何通过MySQL更好的实现分页，始终是比较令人头疼的问题。虽然没有拿来就能用的解决办法，但了解数据库的底层或多或少有助于优化分页查询。  
我们先从一个常用但性能很差的查询来看一看。

|  |  |
| --- | --- |
| 1  2  3  4 | SELECT \*  FROM city  ORDER BY id DESC  LIMIT 0, 15 |

这个查询耗时0.00sec。So，这个查询有什么问题呢？实际上，这个查询语句和参数都没有问题，因为它用到了下面表的主键，而且只读取15条记录。

|  |  |
| --- | --- |
| 1  2  3  4  5 | CREATE TABLE city (  id int(10) unsigned NOT NULL AUTO\_INCREMENT,  city varchar(128) NOT NULL,  PRIMARY KEY (id)  ) ENGINE=InnoDB; |

真正的问题在于offset(分页偏移量)很大的时候，像下面这样：

|  |  |
| --- | --- |
| 1  2  3  4 | SELECT \*  FROM city  ORDER BY id DESC  LIMIT 100000, 15; |

上面的查询在有2M行记录时需要0.22sec，通过EXPLAIN查看SQL的执行计划可以发现该SQL检索了100015行，但最后只需要15行。大的分页偏移量会增加使用的数据，MySQL会将大量最终不会使用的数据加载到内存中。就算我们假设大部分网站的用户只访问前几页数据，但少量的大的分页偏移量的请求也会对整个系统造成危害。Facebook意识到了这一点，但Facebook并没有为了每秒可以处理更多的请求而去优化数据库，而是将重心放在将[请求响应时间的方差变小](http://highscalability.com/blog/2010/11/4/facebook-at-13-million-queries-per-second-recommends-minimiz.html)。  
对于分页请求，还有一个信息也很重要，就是总共的记录数。我们可以通过下面的查询很容易的获取总的记录数。

|  |  |
| --- | --- |
| 1  2 | SELECT COUNT(\*)  FROM city; |

然而，上面的SQL在采用InnoDB为存储引擎时需要耗费9.28sec。一个不正确的优化是采用SQL\_CALC\_FOUND\_ROWS,SQL\_CALC\_FOUND\_ROWS可以在能够在分页查询时事先准备好符合条件的记录数，随后只要执行一句select FOUND\_ROWS(); 就能获得总记录数。但是在大多数情况下，查询语句简短并不意味着性能的提高。不幸的是，这种分页查询方式在许多主流框架中都有用到，下面看看这个语句的查询性能。

|  |  |
| --- | --- |
| 1  2  3  4 | SELECT SQL\_CALC\_FOUND\_ROWS \*  FROM city  ORDER BY id DESC  LIMIT 100000, 15; |

这个语句耗时20.02sec，是上一个的两倍。事实证明使用SQL\_CALC\_FOUND\_ROWS做分页是很糟糕的想法。  
下面来看看到底如何优化。文章分为两部分，第一部分是如何获取记录的总数目，第二部分是获取真正的记录。

高效的计算行数

如果采用的引擎是MyISAM，可以直接执行COUNT(\*)去获取行数即可。相似的，在堆表中也会将行数存储到表的元信息中。但如果引擎是InnoDB情况就会复杂一些，因为InnoDB不保存表的具体行数。  
我们可以将行数缓存起来，然后可以通过一个守护进程定期更新或者用户的某些操作导致缓存失效时，执行下面的语句：

|  |  |
| --- | --- |
| 1  2  3 | SELECT COUNT(\*)  FROM city  USE INDEX(PRIMARY); |

获取记录

下面进入这篇文章最重要的部分，获取分页要展示的记录。上面已经说过了，大的偏移量会影响性能，所以我们要重写查询语句。为了演示，我们创建一个新的表“news”，按照时事性排序(最新发布的在最前面)，实现一个高性能的分页。为了简单，我们就假设最新发布的新闻的Id也是最大的。

|  |  |
| --- | --- |
| 1  2  3  4 | CREATE TABLE news(  id INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,  title VARCHAR(128) NOT NULL  ) ENGINE=InnoDB; |

一个比较高效的方式是基于用户展示的最后一个新闻Id。查询下一页的语句如下，需要传入当前页面展示的最后一个Id。

|  |  |
| --- | --- |
| 1  2  3  4 | SELECT \*  FROM news WHERE id < $last\_id  ORDER BY id DESC  LIMIT $perpage |

查询上一页的语句类似，只不过需要传入当前页的第一个Id，并且要逆序。

|  |  |
| --- | --- |
| 1  2  3  4 | SELECT \*  FROM news WHERE id > $last\_id  ORDER BY id ASC  LIMIT $perpage |

上面的查询方式适合实现简易的分页，即不显示具体的页数导航，只显示“上一页”和“下一页”，例如博客中页脚显示“上一页”，“下一页”的按钮。但如果要实现真正的页面导航还是很难的,下面看看另一种方式。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | SELECT id  FROM (  SELECT id, ((@cnt:= @cnt + 1) + $perpage - 1) % $perpage cnt  FROM news  JOIN (SELECT @cnt:= 0)T  WHERE id < $last\_id  ORDER BY id DESC  LIMIT $perpage \* $buttons  )C  WHERE cnt = 0; |

通过上面的语句可以为每一个分页的按钮计算出一个offset对应的id。这种方法还有一个好处。假设，网站上正在发布一片新的文章，那么所有文章的位置都会往后移一位，所以如果用户在发布文章时换页，那么他会看见一篇文章两次。如果固定了每个按钮的offset Id，这个问题就迎刃而解了。Mark Callaghan发表过一篇类似的[博客](http://www.facebook.com/note.php?note_id=206034210932)，利用了组合索引和两个位置变量，但是基本思想是一致的。  
如果表中的记录很少被删除、修改，还可以将记录对应的页码存储到表中，并在该列上创建合适的索引。采用这种方式，当新增一个记录的时候，需要执行下面的查询重新生成对应的页号。

|  |  |
| --- | --- |
| 1  2 | SET @p:= 0;  UPDATE news SET page=CEIL((@p:= @p + 1) / $perpage) ORDER BY id DESC; |

当然，也可以新增一个专用于分页的表，可以用个后台程序来维护。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | UPDATE pagination T  JOIN (  SELECT id, CEIL((@p:= @p + 1) / $perpage) page  FROM news  ORDER BY id  )C  ON C.id = T.id  SET T.page = C.page; |

现在想获取任意一页的元素就很简单了：

|  |  |
| --- | --- |
| 1  2  3  4 | SELECT \*  FROM news A  JOIN pagination B ON A.id=B.ID  WHERE page=$offset; |

还有另外一种与上种方法比较相似的方法来做分页，这种方式比较试用于数据集相对小，并且没有可用的索引的情况下—比如处理搜索结果时。在一个普通的服务器上执行下面的查询，当有2M条记录时，要耗费2sec左右。这种方式比较简单，创建一个用来存储所有Id的临时表即可(这也是最耗费性能的地方)。

|  |  |
| --- | --- |
| 1  2  3  4  5 | CREATE TEMPORARY TABLE \_tmp (KEY SORT(random))  SELECT id, FLOOR(RAND() \* 0x8000000) random  FROM city;  ALTER TABLE \_tmp ADD OFFSET INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT, DROP INDEX SORT, ORDER BY random; |

接下来就可以向下面一样执行分页查询了。

|  |  |
| --- | --- |
| 1  2  3  4  5 | SELECT \*  FROM \_tmp  WHERE OFFSET >= $offset  ORDER BY OFFSET  LIMIT $perpage; |

——————————————–俺只是个分割线———————————————————-  
简单来说，对于分页的优化就是。。。避免数据量大时扫描过多的记录。  
博客比较长，所以翻译的有些粗糙。。。，之后会在好好检查一遍的。在自己做测试时，有些查询时间与作者有点不一致，不过作者这篇博客是写于2011年的，so~不要在意具体数据，领会精神吧~~