07 换种思路写分页查询

更新时间：2019-08-08 09:52:34

> 你若要喜爱你自己的价值，你就得给世界创造价值。
>
> ——歌德

很多时候，业务上会有分页操作的需求，对应的 SQL 类似下面这条：

```
select a,b,c from t1 limit 10000,10;
```

表示从表 t1 中取出从 10001 行开始的 10 行记录。看似只查询了 10 条记录，实际这条 SQL 是先读取 10010 条记录，然后抛弃前 10000 条记录，然后读到后面 10 条想要的数据。因此要查询一张大表比较靠后的数据，执行效率是非常低的。本节内容就一起研究下，是否有办法去优化分页查询。

为了方便验证，首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` (            /* 创建表t1 */
  `id` int(11) NOT NULL auto_increment,
  `a` int(11) DEFAULT NULL,
  `b` int(11) DEFAULT NULL,
  `create_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '记录创建时间',
  `update_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP COMMENT '记录更新时间',
  PRIMARY KEY (`id`),
  KEY `idx_a` (`a`),
  KEY `idx_b` (`b`)
) ENGINE=InnoDB 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<=100000)do               /* 对满足i<=100000的值进行while循环 */
    insert into t1(a,b) values(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100000条数据到表t1的存储过程insert_t1 */
call insert_t1();          /* 运行存储过程insert_t1 */
```

本节会分享两种分页场景的优化技巧：

- 根据自增且连续主键排序的分页查询
- 查询根据非主键字段排序的分页查询

1 根据自增且连续主键排序的分页查询

首先来看一个根据自增且连续主键排序的分页查询的例子：

```
select * from t1 limit 99000,2;
```

```
mysql> select * from t1 limit 99000,2;
+-------+-------+-------+---------------------+---------------------+
| id    | a     | b     | create_time         | update_time         |
+-------+-------+-------+---------------------+---------------------+
| 99001 | 99001 | 99001 | 2019-06-24 16:33:12 | 2019-06-24 16:33:12 |
| 99002 | 99002 | 99002 | 2019-06-24 16:33:12 | 2019-06-24 16:33:12 |
+-------+-------+-------+---------------------+---------------------+
2 rows in set (0.03 sec)
```

该 SQL 表示查询从第 99001开始的两行数据，没添加单独 order by，表示通过主键排序。我们再看表 t1，因为主键是自增并且连续的，所以可以改写成按照主键去查询从第 99001开始的两行数据，如下：

```
select * from t1 where id >99000 limit 2;
```

```
mysql> select * from t1 where id >99000 limit 2;
+-------+-------+-------+---------------------+---------------------+
| id    | a     | b     | create_time         | update_time         |
+-------+-------+-------+---------------------+---------------------+
| 99001 | 99001 | 99001 | 2019-06-24 16:33:12 | 2019-06-24 16:33:12 |
| 99002 | 99002 | 99002 | 2019-06-24 16:33:12 | 2019-06-24 16:33:12 |
+-------+-------+-------+---------------------+---------------------+
2 rows in set (0.00 sec)
```

查询的结果是一致的。我们再对比一下执行计划：

```
mysql> explain select * from t1 limit 99000,2;
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+-------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows  | filtered | Extra |
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+-------+
|  1 | SIMPLE      | t1    | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 99965 |   100.00 | NULL  |
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+-------+
1 row in set, 1 warning (0.00 sec)

mysql> explain select * from t1 where id >99000 limit 2;
+----+-------------+-------+------------+-------+---------------+---------+---------+------+------+----------+-------------+
| id | select_type | table | partitions | type  | possible_keys | key     | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-------+------------+-------+---------------+---------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | t1    | NULL       | range | PRIMARY       | PRIMARY | 4       | NULL | 1000 |   100.00 | Using where |
+----+-------------+-------+------------+-------+---------------+---------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
```

原 SQL 中 key 字段为 NULL，表示未走索引，rows 显示 99965，表示扫描的行数 99965行；

改写后的 SQL key 字段为 PRIMARY，表示走了主键索引，扫描了1000行。

显然改写后的 SQL 执行效率更高。

但是，这条 SQL 在很多场景并不实用，因为表中可能某些记录被删后，主键空缺，导致结果不一致，如下图的实验（整个实验过程为：先删除一条前面的记录，然后再测试原 SQL 和优化后的 SQL）：

```
mysql> delete from t1 where id=10;
Query OK, 1 row affected (0.00 sec)

mysql> select * from t1 limit 99000,2;
+-------+-------+-------+---------------------+---------------------+
| id    | a     | b     | create_time         | update_time         |
+-------+-------+-------+---------------------+---------------------+
| 99002 | 99002 | 99002 | 2019-06-24 16:33:12 | 2019-06-24 16:33:12 |
| 99003 | 99003 | 99003 | 2019-06-24 16:33:12 | 2019-06-24 16:33:12 |
+-------+-------+-------+---------------------+---------------------+
2 rows in set (0.03 sec)

mysql> select * from t1 where id >99000 limit 2;
+-------+-------+-------+---------------------+---------------------+
| id    | a     | b     | create_time         | update_time         |
+-------+-------+-------+---------------------+---------------------+
| 99001 | 99001 | 99001 | 2019-06-24 16:33:12 | 2019-06-24 16:33:12 |
| 99002 | 99002 | 99002 | 2019-06-24 16:33:12 | 2019-06-24 16:33:12 |
+-------+-------+-------+---------------------+---------------------+
2 rows in set (0.00 sec)
```

可以发现两条 SQL 的结果并不一样，因此，如果主键不连续，不能使用上面描述的优化方法。

另外如果原 SQL 是 order by 非主键的字段，按照上面说的方法改写会导致两条 SQL 的结果不一致。所以这种改写得满足以下两个条件：

- 主键自增且连续
- 结果是按照主键排序的

2 查询根据非主键字段排序的分页查询

再看一个根据非主键字段排序的分页查询，SQL 如下：

```
select * from t1 order by a limit 99000,2;
```

```
mysql> select * from t1 order by a limit 99000,2;
+-------+-------+-------+---------------------+---------------------+
| id    | a     | b     | create_time         | update_time         |
+-------+-------+-------+---------------------+---------------------+
| 99002 | 99002 | 99002 | 2019-06-24 16:33:12 | 2019-06-24 16:33:12 |
| 99003 | 99003 | 99003 | 2019-06-24 16:33:12 | 2019-06-24 16:33:12 |
+-------+-------+-------+---------------------+---------------------+
2 rows in set (0.08 sec)
```

查询时间是 0.08 秒。

我们来看下这条 SQL 的执行计划：

```
mysql> explain select * from t1 order by a limit 99000,2;
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+----------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows  | filtered | Extra          |
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+----------------+
|  1 | SIMPLE      | t1    | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 99964 |   100.00 | Using filesort |
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+----------------+
1 row in set, 1 warning (0.00 sec)
```

发现并没有使用 a 字段的索引（key 字段对应的值为 null），具体原因可以复习第 4 节 2.2 小节：**扫描整个索引并查找到没索引的行的成本比扫描全表的成本更高，所以优化器放弃使用索引**。

知道不走索引的原因，那么怎么优化呢？

其实关键是**让排序时返回的字段尽可能少**，所以可以让排序和分页操作先查出主键，然后根据主键查到对应的记录，SQL 改写如下（这里参考了《深入浅出 MySQL》18.4.7 优化分页查询）：

```
select * from t1 f inner join (select id from t1 order by a limit 99000,2)g on f.id = g.id;
```

```
mysql> select * from t1 f inner join (select id from t1 order by a limit 99000,2)g on f.id = g.id;
+-------+-------+-------+---------------------+---------------------+-------+
| id    | a     | b     | create_time         | update_time         | id    |
+-------+-------+-------+---------------------+---------------------+-------+
| 99002 | 99002 | 99002 | 2019-06-24 16:33:12 | 2019-06-24 16:33:12 | 99002 |
| 99003 | 99003 | 99003 | 2019-06-24 16:33:12 | 2019-06-24 16:33:12 | 99003 |
+-------+-------+-------+---------------------+---------------------+-------+
2 rows in set (0.02 sec)
```

需要的结果与原 SQL 一致，执行时间 0.02 秒，是原 SQL 执行时间的四分之一，我们再对比优化前后的执行计划：

```
mysql> explain select * from t1 order by a limit 99000,2;
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+----------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows  | filtered | Extra          |
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+----------------+
|  1 | SIMPLE      | t1    | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 99964 |   100.00 | Using filesort |
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+----------------+
1 row in set, 1 warning (0.00 sec)

mysql> explain select * from t1 f inner join (select id from t1 order by a limit 99000,2)g on f.id = g.id;
+----+-------------+------------+------------+--------+---------------+---------+---------+------+-------+----------+-------------+
| id | select_type | table      | partitions | type   | possible_keys | key     | key_len | ref  | rows  | filtered | Extra       |
+----+-------------+------------+------------+--------+---------------+---------+---------+------+-------+----------+-------------+
|  1 | PRIMARY     | <derived2> | NULL       | ALL    | NULL          | NULL    | NULL    | NULL | 99002 |   100.00 | NULL        |
|  1 | PRIMARY     | f          | NULL       | eq_ref | PRIMARY       | PRIMARY | 4       | g.id |     1 |   100.00 | NULL        |
|  2 | DERIVED     | t1         | NULL       | index  | NULL          | idx_a   | 5       | NULL | 99002 |   100.00 | Using index |
+----+-------------+------------+------------+--------+---------------+---------+---------+------+-------+----------+-------------+
3 rows in set, 1 warning (0.00 sec)
```

原 SQL 使用的是 filesort 排序，而优化后的 SQL 使用的是索引排序。

3 总结

本节讲到了两种分页查询场景的优化：

- 根据自增且连续主键排序的分页查询优化
- 查询根据非主键字段排序的分页查询优化

对于其它一些复杂的分页查询，也基本可以按照这两个思路去优化，尤其是第二种优化方式。第一种优化方式需要主键连续，而主键连续对于一个正常业务表来说可能有点困难，总会有些数据行删除的，但是占用了一个主键 id。

4 问题

对于本节生成的测试表 t1，如果主键是自增的，但是中间有部分记录被删了，也就是主键不连续，下面这条 SQL 应该怎么优化？

```
select * from t1 limit 99000,2;
```

你可以把你的优化结果写在留言区一起讨论。

5 参考资料

《MySQL 5.7 Reference Manual》8.2.1.17 LIMIT Query Optimization：https://dev.mysql.com/doc/refman/5.7/en/limit-optimization.html

《深入浅出 MySQL》（第 2 版）：18.4.7 优化分页查询

《高性能 MySQL》（第 3 版）：6.7.5 优化 LIMIT 分页

}

06 让order by、group by查询更快

08 Join语句可以这样优化