43 | 如何使用Redis搭建玩家排行榜？

2019-09-11 陈旸

上一篇文章中，我们使用Redis模拟了多用户抢票的问题，这里再回顾一下原理。我们通过使用WATCH+MULTI的方式实现乐观锁机制，对ticket_count这个键进行监视，当这个键发生变化的时候事务就会被打断，重新请求，这样做的好处就是可以保证事务对键进行操作的原子性，当然我们也可以使用Redis的incr和decr来实现键的原子性递增或递减。

今天我们用Redis搭建一个玩家的排行榜，假设一个服务器存储了10万名玩家的数据，我们想给这个区（这台服务器）上的玩家做个全区的排名，该如何用Redis实现呢？

不妨一起来思考下面几个问题：

1. MySQL是如何实现玩家排行榜的？有哪些难题需要解决？
2. 如何用Redis模拟10万名玩家数据？Redis里的Lua又是什么？
3. Redis如何搭建玩家排行榜？和MySQL相比有什么优势？

使用MySQL搭建玩家排行榜

我们如果用MySQL搭建玩家排行榜的话，首先需要生成10万名玩家的数据，这里我们使用之前学习过的存储过程来模拟。

为了简化，玩家排行榜主要包括3个字段：user_id、score、和create_time，它们分别代表玩家的用户ID、玩家的积分和玩家的创建时间。

王者荣耀英雄等级说明

这里我们可以模拟王者荣耀的英雄等级，具体等级标准如下：

段位	星星总数	说明
青铜	9颗	分为3段，每段晋级需要3星
白银	12颗	分为3段，每段晋级需要4星
黄金	16颗	分为4段，每段晋级需要4星
铂金	25颗	分为5段，每段晋级需要5星
钻石	25颗	分为5段，每段晋级需要5星
星耀	25颗	分为5段，每段晋级需要5星
最强王者	无上限	最高段位，可积累无限星星

如果想要英雄要达到最强王者的段位，那么之前需要积累112颗（9+12+16+25+25+25）星星，而达到最强王者之后还可以继续积累无上限的星星。在随机数模拟上，我们也分成两个阶段，第一个阶段模拟英雄的段位，我们使用随机数来模拟score（数值范围是1-112之间），当score=112的时候，再模拟最强王者等级中的星星个数。如果我们只用一个随机数进行模拟，会出现最强王者的比例变大的情况，显然不符合实际情况。

使用存储过程模拟10万名玩家数据

这里我们使用存储过程，具体代码如下：

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `insert_many_user_scores`(IN START INT(10), IN max_num
BEGIN
DECLARE i INT DEFAULT 0;
-- 模拟玩家英雄的星星数
DECLARE score INT;
DECLARE score2 INT;
-- 初始注册时间
DECLARE date_start DATETIME DEFAULT ('2017-01-01 00:00:00');
-- 每个玩家的注册时间
DECLARE date_temp DATETIME;
SET date_temp = date_start;
SET autocommit=0;

REPEAT
SET i=i+1;
SET date_temp = date_add(date_temp, interval RAND()*60 second);
-- 1-112随机数
SET score = CEIL(RAND()*112);
-- 如果达到了王者，继续模拟王者的星星数
IF score = 112 THEN
        SET score2 = FLOOR(RAND()*100);
        SET score = score + score2;
END IF;
-- 插入新玩家
INSERT INTO user_score(user_id, score, create_time) VALUES((START+i), score, date_temp);
UNTIL i = max_num
END REPEAT;
COMMIT;
END
```

然后我们使用call insert_many_user_scores(10000,100000);模拟生成10万名玩家的得分数据。注意在insert之前，需要先设置autocommit=0，也就是关闭了自动提交，然后在批量插入结束之后再手动进行COMMIT，这样做的好处是可以进行批量提交，提升插入效率。你可以看到整体的用时为5.2秒。

```
mysql> call insert_many_user_scores(10000,100000);
Query OK, 0 rows affected (5.20 sec)
```

如上代码所示，我用score来模拟第一阶段的星星数，如果score达到了112再来模拟score2的分数，这里我限定最强王者阶段的星星个数上限为100。同时我们还模拟了用户注册的时间，这是因为排行榜可以有两种表示方式，第二种方式需要用到这个时间。

第一种表示方式为并列排行榜，也就是分数相同的情况下，允许排名并列，如下所示：

rank	user_id	score
1	10003	112
2	10013	100
2	10015	100
4	10011	98
5	10017	96

第二种为严格排行榜。当分数相同的时候，会按照第二条件来进行排序，比如按照注册时间的长短，注册时间越长的排名越靠前。这样的话，上面那个排行榜就会变成如下所示的严格排行榜。

rank	user_id	create_time	score
1	10003	2017-01-01 05:02:00	112
2	10013	2017-01-02 09:09:13	100
3	10015	2017-01-03 10:08:18	100
4	10011	2017-01-01 12:07:13	98
5	10017	2017-01-05 08:02:36	96

你能看到当10013和10015得分相同的时候，如果按照注册时间来进行排名的话，会将10013排到10015前面。

上面的数据仅仅为示意，下面我们用实际的10万条数据做一个严格排行榜（你可以点击下载地址下载这10万条数据， 也可以自己使用上面的存储过程来进行模拟）首先使用SQL语句进行查询：

```
SELECT (@rownum := @rownum + 1) AS user_rank, user_id, score, create_time
FROM user_score, (SELECT @rownum := 0) b
ORDER BY score DESC, create_time ASC
```

运行结果如下（10万条数据，用时0.17s）：

user_rank	user_id	score	create_time
1	35674	211	2017-01-09 22:12:25
2	37837	211	2017-01-10 16:18:24
3	42377	211	2017-01-12 06:16:07
.....	.....	.....	.....
100000	109984	1	2017-02-04 20:15:55

这里有几点需要说明。

MySQL不像Oracle一样自带rownum统计行编号的功能，所以这里我们需要自己来实现rownum功能，也就是设置MySQL的变量@rownum，初始化为@rownum :=0，然后每次SELECT一条数据的时候都自动加1。

通过开发程序（比如Python、PHP和Java等）统计排名会更方便，这里同样需要初始化一个变量，比如rownum=0，然后每次fetch一条数据的时候都将该变量加1，作为记录的排名。同时，开发程序也可以很方便地实现并列排名，因为程序可以进行上下文的统计，当两名玩家得分相同时，排名相同，否则排名会顺序加1。

如果想要通过SQL来实现，可以写成下面这样：

```
SELECT user_id, score,
    IFNULL((SELECT COUNT(*) FROM user_score WHERE score > t.score), 0) + 1 AS user_rank
FROM user_score t
ORDER BY user_rank ASC
```

这样做的原理是查找比当前分数大的数据行数，然后加1，但是这样执行效率会很低，相当于需要对每个玩家都统计一遍排名。

Lua是什么，如何在Redis中使用

知道如何用MySQL模拟数据后，我们再来看下如何在Redis中完成这一步。事实上，Redis本身不提供存储过程的功能，不过在2.6版本之后集成了Lua语言，可以很方便地实现类似存储过程的函数调用方式。

Lua是一个小巧的脚本语言，采用标准C语言编写，一个完整的Lua解析器大小只有200K。我们之前讲到过采用标准C语言编写的好处就在于执行效率高，依懒性低，同时兼容性好，稳定性高。这些特性同样Lua也有，它可以嵌入到各种应用程序中，提供灵活的扩展和定制功能。

如何在Redis中使用Lua

在Redis中使用Lua脚本的命令格式如下：

```
EVAL script numkeys key [key ...] arg [arg ...]
```

我来说明下这些命令中各个参数代表的含义。

1. script，代表的是Lua的脚本内容。
2. numkeys，代表后续参数key的个数。
3. key就是我们要操作的键，可以是多个键。我们在Lua脚本中可以直接使用这些key，直接通过KEYS[1]、KEYS[2]来获取，默认下标是从1开始。
4. arg，表示传入到Lua脚本中的参数，就像调用函数传入的参数一样。在Lua脚本中我们可以通过ARGV[1]、ARGV[2]来进行获取，同样默认下标从1开始。

下面我们通过2个例子来体会下，比如我们使用eval "return {ARGV[1], ARGV[2]}" 0 cy 123，代表的是传入的key的个数为0，后面有两个arg，分别为cy和123。在Lua脚本中，我们直接返回这两个参数ARGV[1], ARGV[2]，执行结果如下：

```
127.0.0.1:6379> eval "return {ARGV[1], ARGV[2]}" 0 cy 123
1) "cy"
2) "123"
```

比如我们要用这一条语句：

```
eval "math.randomseed(ARGV[1]); local temp = math.random(1,112); redis.call('SET', KEYS[1], temp); return 'ok';"
```

这条语句代表的意思是，我们传入KEY的个数为1，参数是score，arg参数为30。在Lua脚本中使用ARGV[1]，也就是30作为随机数的种子，然后创建本地变量temp等于1到112之间的随机数，再使用SET方法对KEY，也就是用刚才创建的随机数对score这个字段进行赋值，结果如

下：

```
127.0.0.1:6379> eval "math.randomseed(ARGV[1]); local temp = math.random(1,112);
 redis.call('SET', KEYS[1], temp); return 'ok';" 1 score 30
"ok"
127.0.0.1:6379> GET score
"34"
```

然后我们在Redis中使用GET score对刚才设置的随机数进行读取，结果为34。

另外我们还可以在命令中调用Lua脚本，使用的命令格式：

```
redis-cli –eval lua_file key1 key2 , arg1 arg2 arg3
```

使用redis-cli的命令格式不需要输入key的个数，在key和arg参数之间采用了逗号进行分割，注意逗号前后都需要有空格。同时在eval后面可以带一个lua文件（以.lua结尾）。

使用Lua创建10万名玩家数据

如果我们想要通过Lua脚本创建10万名玩家的数据，文件名为insert_user_scores.lua，代码如下：

```
--设置时间种子
math.randomseed(ARGV[1])
-- 设置初始的生成时间
local create_time = 1567769563 - 3600*24*365*2.0
local num = ARGV[2]
local user_id = ARGV[3]
for i=1, num do
  --生成1到60之间的随机数
  local interval = math.random(1, 60)
  --产生1到112之间的随机数
  local temp = math.random(1, 112)
  if (temp == 112) then
      --产生0到100之间的随机数
      temp = temp + math.random(0, 100)
  end
  create_time = create_time + interval
  temp = temp + create_time / 10000000000
  redis.call('ZADD', KEYS[1], temp, user_id+i-1)
end
return 'Generation Completed'
```

上面这段代码可以实现严格排行榜的排名，具体方式是将score进行了改造，score 为浮点数。整数部分为得分，小数部分为时间差。

在调用的时候，我们通过ARGV[1]获取时间种子的参数，传入的KEYS[1]为user_score，也就是创建有序集合user_score。然后通过num来设置生成玩家的数量，通过user_id获取初始的user_id。最后调用如下命令完成玩家数据的创建：

```
redis-cli -h localhost -p 6379 --eval insert_user_scores.lua user_score , 30 100000 10000
```

```
H:\projects\SQL必知必会\redis>redis-cli -h localhost -p 6379 --eval insert_user_
scores.lua user_score , 30 100000 10000
"Generation Completed"
```

使用Redis实现玩家排行榜

我们通过Lua脚本模拟完成10万名玩家数据，并将其存储在了Redis的有序集合user_score中，下面我们就来使用Redis来统计玩家排行榜的数据。

首先我们需要思考的是，一个典型的游戏排行榜都包括哪些功能呢？

1. 统计全部玩家的排行榜
2. 按名次查询排名前N名的玩家
3. 查询某个玩家的分数
4. 查询某个玩家的排名
5. 对玩家的分数和排名进行更新
6. 查询指定玩家前后M名的玩家
7. 增加或移除某个玩家，并对排名进行更新

在Redis中实现上面的功能非常简单，只需要使用Redis我们提供的方法即可，针对上面的排行榜功能需求，我们分别来看下Redis是如何实现的。

统计全部玩家的排行榜

在Redis里，统计全部玩家的排行榜的命令格式为ZREVRANGE 排行榜名称 起始位置 结束为止 [WITHSCORES]。

我们使用这行命令即可：

```
ZREVRANGE user_score 0 -1 WITHSCORES
```

我们对玩家排行榜user_score进行统计，其中-1代表的是全部的玩家数据，WITHSCORES代表的是输出排名的同时也输出分数。

按名次查询排名前N名的玩家

同样我们可以使用ZREVRANGE完成前N名玩家的排名，比如我们想要统计前10名玩家，可以使用：ZREVRANGE user_score 0 9。

```
127.0.0.1:6379> ZREVRANGE user_score 0 9
 1) "109625"
 2) "105792"
 3) "105231"
 4) "103794"
 5) "94252"
 6) "82501"
 7) "67046"
 8) "60677"
 9) "57364"
10) "37094"
```

查询某个玩家的分数

命令格式为ZSCORE 排行榜名称 玩家标识。

时间复杂度为O(1)。

如果我们想要查询玩家10001的分数可以使用：ZSCORE user_score 10001。

```
127.0.0.1:6379> ZSCORE user_score 10001
"94.1504697596"
```

查询某个玩家的排名

命令格式为ZREVRANK 排行榜名称 玩家标识。

时间复杂度为O(log(N))。

如果我们想要查询玩家10001的排名可以使用：ZREVRANK user_score 10001。

```
127.0.0.1:6379> ZREVRANK user_score 10001
(integer) 17153
```

对玩家的分数进行更新，同时排名进行更新

如果我们想要对玩家的分数进行增减，命令格式为ZINCRBY排行榜名称 分数变化 玩家标识。

时间复杂度为O(log(N))。

比如我们想对玩家10001的分数减1，可以使用：ZINCRBY user_score -1 10001。

```
127.0.0.1:6379> ZINCRBY user_score -1 10001
"93.1504697596"
```

然后我们再来查看下玩家10001的排名，使用：ZREVRANK user_score 10001。

```
127.0.0.1:6379> ZREVRANK user_score 10001
(integer) 18036
```

你能看到排名由17153降到了18036名。

查询指定玩家前后M名的玩家

比如我们想要查询玩家10001前后5名玩家都是谁，当前已知玩家10001的排名是18036，那么可以使用：ZREVRANGE user_score 18031 18041。

```
127.0.0.1:6379> ZREVRANGE user_score 18031 18041
 1) "10991"
 2) "10736"
 3) "10206"
 4) "10205"
 5) "10195"
 6) "10001"
 7) "109941"
 8) "109902"
 9) "109820"
10) "109796"
11) "109723"
```

这样就可以得到玩家10001前后5名玩家的信息。

增加或删除某个玩家，并对排名进行更新

如果我们想要删除某个玩家，命令格式为ZREM 排行榜名称 玩家标识。

时间复杂度为O(log(N))。

比如我们想要删除玩家10001，可以使用：ZREM user_score 10001。

```
127.0.0.1:6379> ZREM user_score 10001
(integer) 1
```

这样我们再来查询下排名在18031到18041的玩家是谁，使用：ZREVRANGE user_score 18031 18041。

```
127.0.0.1:6379> ZREVRANGE user_score 18031 18041
 1) "10991"
 2) "10736"
 3) "10206"
 4) "10205"
 5) "10195"
 6) "109941"
 7) "109902"
 8) "109820"
 9) "109796"
10) "109723"
11) "109677"
```

你能看到玩家10001的信息被删除，同时后面的玩家排名都向前移了一位。

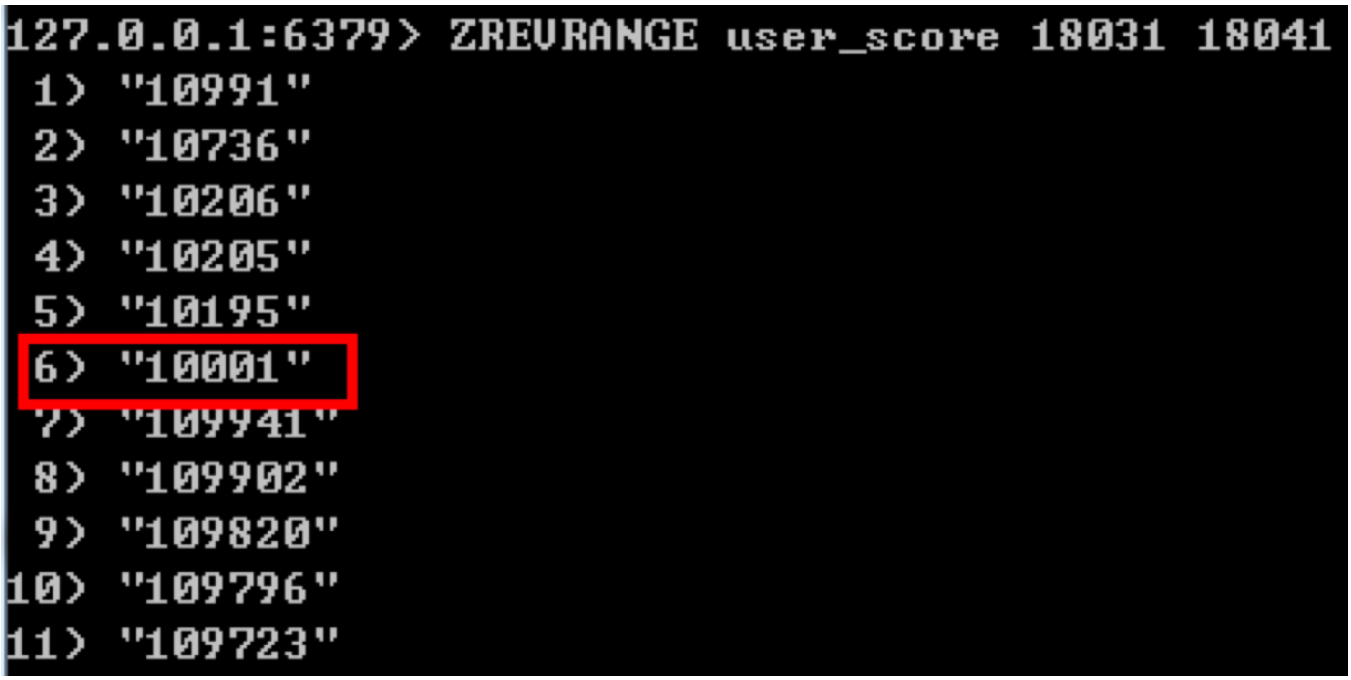
如果我们想要增加某个玩家的数据，命令格式为ZADD 排行榜名称 分数 玩家标识。

时间复杂度为O(log(N))。

这里，我们把玩家10001的信息再增加回来，使用：ZADD user_score 93.1504697596 10001。

```
127.0.0.1:6379> ZADD user_score 93.1504697596 10001
(integer) 1
```

然后我们再来看下排名在18031到18041的玩家是谁，使用：ZREVRANGE user_score 18031 18041。

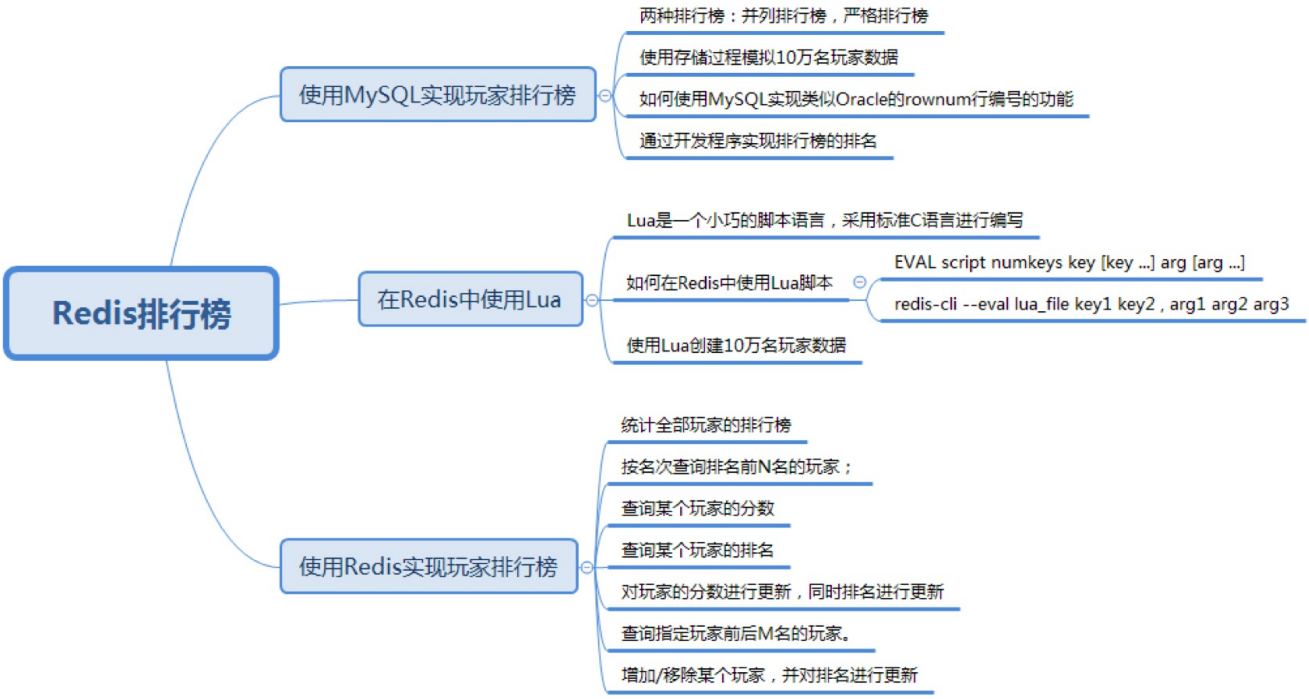


你能看到插入了玩家10001的数据之后，排名又回来了。

总结

今天我们使用MySQL和Redis搭建了排行榜，根据相同分数的处理方式，我们可以把排行榜分成并列排行榜和严格排行榜。虽然MySQL和Redis都可以搭建排行榜，但两者还是有区别的。MySQL擅长存储数据，而对于数据的运算来说则效率不高，比如统计排行榜的排名，通常还是需要使用后端语言（比如Python、PHP、Java等）再进行统计。而Redis本身提供了丰富的排行榜统计功能，不论是增加、删除玩家，还是对某个玩家的分数进行调整，Redis都可以对排行榜实时更新，对于游戏的实时排名来说，这还是很重要的。

在Redis中还集成了Lua脚本语言，通过Lua我们可以更加灵活地扩展Redis的功能，同时在Redis中使用Lua语言，还可以对Lua脚本进行复用，减少网络开销，编写代码也更具有模块化。此外Redis在调用Lua脚本的时候，会将它作为一个整体，也就是说中间如果有其他的Redis命令是不会被插入进去的，也保证了Lua脚本执行过程中不会被其他命令所干扰。



我们今天使用Redis对10万名玩家的数据进行了排行榜的统计，相比于用RDBMS实现排行榜来说，使用Redis进行统计都有哪些优势呢？

我们使用了Lua脚本模拟了10万名玩家的数据，其中玩家的分数score分成了两个部分，整数部分为实际的得分，小数部分为注册时间。例子中给出的严格排行榜是在分数相同的情况下，按照注册时间的长短进行的排名，注册时间长的排名靠前。如果我们将规则进行调整，同样是在分数相同的情况下，如果注册时间长的排名靠后，又该如何编写代码呢？

欢迎你在评论区写下你的思考，也欢迎把这篇文章分享给你的朋友或者同事，一起交流一下。

精选留言

石维康 1

在生成数据时,把"temp = temp + create_time / 10000000000"换成 temp = temp +1 - create_time / 10000000000 哈哈

2019-09-12

DemonLee 1

感觉用redis，最终还是需要结合程序以及MySQL来处理，因为排行榜展示，前端还是需要用户名的，光给个用户id不知道是谁，除非redis有序集合的member包含了用户id和name，请指正。

2019-09-12

ttttt 1

注册时间排名靠后MySQL语法：create_time按照降序排列。

```
SELECT (@rownum := @rownum + 1) AS user_rank, user_id, score, create_time
FROM user_score, (SELECT @rownum := 0) b
ORDER BY score DESC, create_time DESC
```

2019-09-11

felix 0

咨询老师一个关于ip匹配的索引问题：

有一个IP的库表，每一条记录了一个开始ip和结束ip，然后想批量匹配ip，查询为何没有用上“联合索引KEY `ip_range_int` (`start_int`,`end_int`) USING BTREE”？要怎么设置索引才有效？

```
CREATE TABLE `t_dt_ip` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`start_ip` char(15) DEFAULT NULL,
`end_ip` char(15) DEFAULT NULL,
`location` varchar(100) DEFAULT NULL,
`start_int` int(10) unsigned DEFAULT '0',
`end_int` int(10) unsigned DEFAULT '0',
PRIMARY KEY (`id`),
KEY `ip_range` (`start_ip`,`end_ip`) USING BTREE,
KEY `ip_range_int` (`start_int`,`end_int`) USING BTREE
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;

explain update t_tmp_ip t, t_dt_ip i
set t.ip_id = i.id
where INET_ATON(t.ip_address) between i.start_int and i.end_int;
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered |
```

Extra |
| 1 | UPDATE | t | NULL | ALL | NULL | NULL | NULL | NULL | 1000 | 100.00 | NULL |
| 1 | SIMPLE | i | NULL | ALL | ip_range_int | NULL | NULL | NULL | 541942 | 11.11 | Range ch
ecked for each record (index map: 0xC) |

甚至加上单个字段索引也没有用？？
alter table `t_dt_ip` add index indx_t_dt_ip_start_int (start_int);
mysql> explain select * from t_dt_ip i join t_tmp_ip t on 1= 1 where t.ip_address >= i.start_int li
mit 1;
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered |
Extra |
| 1 | SIMPLE | t | NULL | ALL | NULL | NULL | NULL | NULL | 73126 | 100.00 | NULL |
| 1 | SIMPLE | i | NULL | ALL | ip_range_int,indx_t_dt_ip_start_int | NULL | NULL | NULL | 5419
42 | 33.33 | Range checked for each record (index map: 0xC) |

2019-09-14