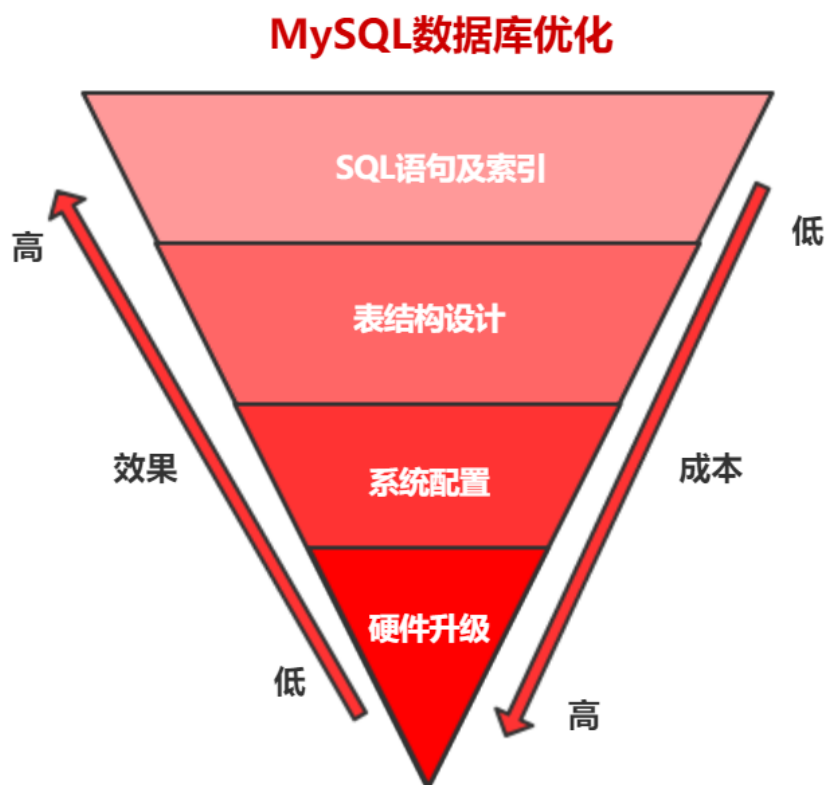


# MySQL性能优化

数据库优化维度有四个：

硬件升级、系统配置、表结构设计、SQL语句及索引。



优化选择：

- 优化成本：硬件升级>系统配置>表结构设计>SQL语句及索引。
- 优化效果：硬件升级<系统配置<表结构设计<SQL语句及索引。

## 1、系统配置优化

### 1.1 保证从内存中读取数据

MySQL会在内存中保存一定的数据，通过LRU算法将不常访问的数据保存在硬盘文件中。

尽可能的扩大内存中的数据量，将数据保存在内存中，从内存中读取数据，可以提升MySQL性能。

扩大`innodb_buffer_pool_size`，能够全然从内存中读取数据。最大限度降低磁盘操作。

确定`innodb_buffer_pool_size` 足够大的方法：

```
mysql> show global status like 'innodb_buffer_pool_pages_%';
+-----+
| Variable_name | Value |
+-----+
| Innodb_buffer_pool_pages_data | 8190 |
| Innodb_buffer_pool_pages_dirty | 0 |
| Innodb_buffer_pool_pages_flushed | 12646 |
| Innodb_buffer_pool_pages_free | 0 | 0 表示已经被用光
| Innodb_buffer_pool_pages_misc | 1 |
| Innodb_buffer_pool_pages_total | 8191 |
+-----+
```

innodb\_buffer\_pool\_size默认为128M，理论上可以扩大到内存的3/4或4/5。

修改 my.cnf

innodb\_buffer\_pool\_size = 750M

如果是专用的MySQL Server可以禁用SWAP

```
#查看swap
cat /proc/swaps
Filename                                Type              Size      Used      Priority
/dev/sda2                              partition         1048572  0         -1
#关闭所有交换设备和文件。
swapoff -a
```

## 1.2 数据预热

默认情况，仅仅有某条数据被读取一次，才会缓存在 innodb\_buffer\_pool。

所以，数据库刚刚启动，须要进行数据预热，将磁盘上的全部数据缓存到内存中。

数据预热能够提高读取速度。

1、对于InnoDB数据库，进行数据预热的脚本是：

```
SELECT DISTINCT
CONCAT('SELECT ',ndxcollist,' FROM ',db,'.',tb,
' ORDER BY ',ndxcollist,';') SelectQueryToLoadCache
FROM
(
    SELECT
        engine,table_schema db,table_name tb,
        index_name,GROUP_CONCAT(column_name ORDER BY seq_in_index)
    ndxcollist
    FROM
    (
        SELECT
            B.engine,A.table_schema,A.table_name,
            A.index_name,A.column_name,A.seq_in_index
        FROM
            information_schema.statistics A INNER JOIN
            (
                SELECT engine,table_schema,table_name
                FROM information_schema.tables WHERE
```

```

        engine='InnoDB'
    ) B USING (table_schema,table_name)
    WHERE B.table_schema NOT IN ('information_schema','mysql')
    ORDER BY table_schema,table_name,index_name,seq_in_index
) A
GROUP BY table_schema,table_name,index_name
) AA
ORDER BY db,tb;

```

将该脚本保存为：loadtomem.sql

2、执行命令：

```
mysql -uroot -proot -AN < /root/loadtomem.sql > /root/loadtomem.sql
```

3、在需要数据预热时，比如重启数据库

执行命令：

```
mysql -uroot < /root/loadtomem.sql > /dev/null 2>&1
```

### 1.3 降低磁盘写入次数

- 增大redolog，减少落盘次数  
innodb\_log\_file\_size 设置为 0.25 \* innodb\_buffer\_pool\_size
- 通用查询日志、慢查询日志可以不开，bin-log开  
生产中不开通用查询日志，遇到性能问题开慢查询日志
- 写redolog策略 innodb\_flush\_log\_at\_trx\_commit设置为0或2  
如果不涉及非常高的安全性（金融系统），或者基础架构足够安全，或者事务都非常小，都能够用 0 或者 2 来减少磁盘操作。

### 1.4 提高磁盘读写性能

使用SSD或者内存磁盘

## 2、表结构设计优化

### 2.1 设计中间表

设计中间表，一般针对于统计分析功能，或者实时性不高的需求（OLTP、OLAP）

### 2.2 设计冗余字段

为减少关联查询，创建合理的冗余字段（创建冗余字段还需要注意数据一致性问题）

### 2.3 拆表

对于字段太多的大表，考虑拆表（比如一个表有100多个字段）

对于表中经常不被使用的字段或者存储数据比较多的字段，考虑拆表

### 2.4 主键优化

每张表建议都要有一个主键（主键索引），而且主键类型最好是int类型，建议自增主键（不考虑分布式系统的情况下雪花算法）。

## 2.5 字段的设计

数据库中的表越小，在它上面执行的查询也就会越快。

因此，在创建表的时候，为了获得更好的性能，我们可以将表中字段的宽度设得尽可能小。

尽量把字段设置为NOTNULL，这样在将来执行查询的时候，数据库不用去比较NULL值。

对于某些文本字段，例如“省份”或者“性别”，我们可以将它们定义为ENUM类型。因为在MySQL中，ENUM类型被当作数值型数据来处理，而数值型数据被处理起来的速度要比文本类型快得多。这样，我们又可以提高数据库的性能。

能用数字的用数值类型

sex 1 0

## 3、SQL语句及索引优化

设计一个表：tbiguser

```
create table tbiguser(
  id int primary key auto_increment,
  nickname varchar(255),
  loginname varchar(255),
  age int ,
  sex char(1),
  status int,
  address varchar(255)
);
```

向该表中写入10000000条数据

```
CREATE PROCEDURE test_insert()
BEGIN DECLARE i INT DEFAULT 1;
WHILE i<=10000000
DO
insert into tbiguser
VALUES(null,concat('zy',i),concat('zhaoyun',i),23,'1',1,'beijing'); SET i=i+1;
END WHILE ;
commit;
END;
```

执行该存储过程

可以插入10000000条数据

```
mysql> select count(*) from tbiguser;
+-----+
| count(*) |
+-----+
| 10000000 |
+-----+
```

### 3.1 EXPLAIN查看索引使用情况

使用【慢查询日志】功能，去获取所有查询时间比较长的SQL语句 3秒-5秒

使用explain查看有问题的SQL的执行计划，重点查看索引使用情况

```
mysql> explain select * from tbiguser where loginname='zhaoyun1' and
nickname='zy1';
+----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref |
|----|-----|-----|-----|-----|-----|-----|-----|
| 1 | SIMPLE | tbiguser | ref | idx_nickname | idx_nickname | 768 | 768 |
const | 1 | Using index condition; Using where |
+----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

type列，连接类型。一个好的SQL语句至少要达到range级别。杜绝出现all级别。index

key列，使用到的索引名。如果没有选择索引，值是NULL。可以采取强制索引方式。

key\_len列，索引长度。

rows列，扫描行数。该值是个预估值。

extra列，详细说明。注意，常见的不太友好的值，如下：Using filesort，Using temporary 。

常见的索引：

where 字段、组合索引（最左前缀）、索引下推（非选择行不加锁）、覆盖索引（不回表）

on 两边、排序、分组统计

## 3.2 SQL语句中IN包含的值不应过多

MySQL对于IN做了相应的优化，即将IN中的常量全部存储在一个数组里面，而且这个数组是排好序的。但是如果数值较多，产生的消耗也是比较大的。

## 3.3 SELECT语句务必指明字段名称

SELECT \* 增加很多不必要的消耗（CPU、IO、内存、网络带宽）；减少了使用覆盖索引的可能性；当表结构发生改变时，前端也需要更新。所以要求直接在select后面接上字段名。

```
mysql> explain select * from tbiguser ;
+----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref |
|----|-----|-----|-----|-----|-----|-----|-----|
| 1 | SIMPLE | tbiguser | ALL | NULL | NULL | NULL | NULL |
9754360 | NULL |
+----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> explain select id,nickname from tbiguser ;
```

```

+---+-----+-----+-----+-----+-----+-----+
+---+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len |
ref | rows | Extra |
+---+-----+-----+-----+-----+-----+-----+
+---+-----+-----+
| 1 | SIMPLE | tbiguser | index | NULL | NULL | idx_nickname | 768 |
NULL | 9754360 | Using index |
+---+-----+-----+-----+-----+-----+-----+
+---+-----+-----+
1 row in set (0.00 sec)

```

### 3.4 当只需要一条数据的时候，使用limit 1

limit 是可以停止全表扫描的

```

mysql> select * from tbiguser limit 1;
+---+-----+-----+-----+-----+-----+-----+
+---+-----+-----+
| id | nickname | loginname | age | sex | status | address |
+---+-----+-----+-----+-----+-----+-----+
| 1 | zyl | zhaoyun1 | 23 | 1 | 1 | beijing |
+---+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> explain select * from tbiguser limit 1;
+---+-----+-----+-----+-----+-----+-----+
+---+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref |
rows | Extra |
+---+-----+-----+-----+-----+-----+-----+
+---+-----+
| 1 | SIMPLE | tbiguser | ALL | NULL | NULL | NULL | NULL |
9754360 | NULL |
+---+-----+-----+-----+-----+-----+-----+
+---+-----+
1 row in set (0.00 sec)

```

### 3.5 排序字段加索引

```

mysql> explain select * from tbiguser where loginname = 'zhaoyun9999999' order
by id ;
+---+-----+-----+-----+-----+-----+-----+
+---+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref |
| rows | Extra |
+---+-----+-----+-----+-----+-----+-----+
+---+-----+-----+
| 1 | SIMPLE | tbiguser | index | NULL | PRIMARY | 4 | NULL |
| 9754360 | Using where |
+---+-----+-----+-----+-----+-----+-----+
+---+-----+-----+
1 row in set (0.01 sec)

```

```
mysql> explain select * from tbiguser where loginname = 'zhaoyun999999' order
by loginname ;
+----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref |
rows | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | tbiguser | ALL | NULL | NULL | NULL | NULL |
9754360 | Using where |
+----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

### 3.6 如果限制条件中其他字段没有索引，尽量少用or

or两边的字段中，如果有一个不是索引字段，会造成该查询不走索引的情况。

```
mysql> explain select * from tbiguser where nickname='zy1' or
loginname='zhaoyun3';
+----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref |
rows | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | tbiguser | ALL | idx_nickname | NULL | NULL | NULL |
9754360 | Using where |
+----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

### 3.7 尽量用union all代替union

union和union all的差异主要是前者需要将结果集合并后再进行唯一性过滤操作，这就会涉及到排序，增加大量的CPU运算，加大资源消耗及延迟。当然，union all的前提条件是两个结果集没有重复数据。

### 3.8 不使用ORDER BY RAND()

ORDER BY RAND() 不走索引

```
mysql> select * from tbiguser order by rand() limit 10;
+----+-----+-----+-----+-----+-----+-----+-----+
| id | nickname | loginname | age | sex | status | address |
+----+-----+-----+-----+-----+-----+-----+-----+
| 416412 | zy416412 | zhaoyun416412 | 23 | 1 | 1 | beijing |
| 4338012 | zy4338012 | zhaoyun4338012 | 23 | 1 | 1 | beijing |
| 4275017 | zy4275017 | zhaoyun4275017 | 23 | 1 | 1 | beijing |
| 8572779 | zy8572779 | zhaoyun8572779 | 23 | 1 | 1 | beijing |
| 2500546 | zy2500546 | zhaoyun2500546 | 23 | 1 | 1 | beijing |
| 5906410 | zy5906410 | zhaoyun5906410 | 23 | 1 | 1 | beijing |
| 3347200 | zy3347200 | zhaoyun3347200 | 23 | 1 | 1 | beijing |
| 4737955 | zy4737955 | zhaoyun4737955 | 23 | 1 | 1 | beijing |
| 9120355 | zy9120355 | zhaoyun9120355 | 23 | 1 | 1 | beijing |
| 2564477 | zy2564477 | zhaoyun2564477 | 23 | 1 | 1 | beijing |
```

```

+-----+-----+-----+-----+-----+-----+-----+
10 rows in set (10.86 sec)

mysql> select * from tbiguser t1 join (select rand()*(select max(id) from
tbiguser) nid ) t2 on t1.id>t2.nid limit 10;
+-----+-----+-----+-----+-----+-----+-----+
-----+
| id      | nickname | loginname      | age | sex | status | address | nid
|
+-----+-----+-----+-----+-----+-----+-----+
-----+
| 6580156 | zy6580156 | zhaoyun6580156 | 23 | 1   | 1       | beijing |
6580155.591089424 |
| 6580157 | zy6580157 | zhaoyun6580157 | 23 | 1   | 1       | beijing |
6580155.591089424 |
| 6580158 | zy6580158 | zhaoyun6580158 | 23 | 1   | 1       | beijing |
6580155.591089424 |
| 6580159 | zy6580159 | zhaoyun6580159 | 23 | 1   | 1       | beijing |
6580155.591089424 |
| 6580160 | zy6580160 | zhaoyun6580160 | 23 | 1   | 1       | beijing |
6580155.591089424 |
| 6580161 | zy6580161 | zhaoyun6580161 | 23 | 1   | 1       | beijing |
6580155.591089424 |
| 6580162 | zy6580162 | zhaoyun6580162 | 23 | 1   | 1       | beijing |
6580155.591089424 |
| 6580163 | zy6580163 | zhaoyun6580163 | 23 | 1   | 1       | beijing |
6580155.591089424 |
| 6580164 | zy6580164 | zhaoyun6580164 | 23 | 1   | 1       | beijing |
6580155.591089424 |
| 6580165 | zy6580165 | zhaoyun6580165 | 23 | 1   | 1       | beijing |
6580155.591089424 |
+-----+-----+-----+-----+-----+-----+-----+
-----+
10 rows in set (0.01 sec)

```

### 3.9 区分in和exists、not in和not exists

区分in和exists主要是造成了驱动顺序的改变（这是性能变化的关键），如果是exists，那么以外层表为驱动表，先被访问，如果是IN，那么先执行子查询。所以IN适合于外表大而内表小的情况；EXISTS适合于外表小而内表大的情况。

关于not in和not exists，推荐使用not exists，不仅仅是效率问题，not in可能存在逻辑问题。如何高效的写出一个替代not exists的SQL语句？

原SQL语句：

```
select colname ... from A表 where a.id not in (select b.id from B表)
```

高效的SQL语句：

```
select colname ... from A表 Left join B表 on where a.id = b.id where b.id is null
```

### 3.10 使用合理的分页方式以提高分页的效率

分页使用 limit m,n 尽量让m 小

利用主键的定位，可以减小m的值



```
mysql> select * from tbiguser limit 9999998, 2;
```

id	nickname	loginname	age	sex	status	address
9999999	zy9999999	zhaoyun9999999	23	1	1	beijing
10000000	zy10000000	zhaoyun10000000	23	1	1	beijing

```
2 rows in set (4.72 sec)
```

```
mysql> select * from tbiguser where id>9999998 limit 2;
```

id	nickname	loginname	age	sex	status	address
9999999	zy9999999	zhaoyun9999999	23	1	1	beijing
10000000	zy10000000	zhaoyun10000000	23	1	1	beijing

```
2 rows in set (0.00 sec)
```

### 3.11 分段查询

一些用户选择页面中，可能一些用户选择的范围过大，造成查询缓慢。主要的原因是扫描行数过多。这个时候可以通过程序，分段进行查询，循环遍历，将结果合并处理进行展示。

### 3.12 不建议使用%前缀模糊查询

例如LIKE"%name"或者LIKE"%name%", 这种查询会导致索引失效而进行全表扫描。但是可以使用LIKE"name%"。

那么如何解决这个问题呢，答案：使用全文索引或ES全文检索

### 3.13 避免在where子句中对字段进行表达式操作

```
select user_id,user_project from user_base where age*2=36;
```

中对字段就行了算术运算，这会造成引擎放弃使用索引，建议改成：

```
select user_id,user_project from user_base where age=36/2;
```

### 3.14 避免隐式类型转换

where子句中出现column字段的类型和传入的参数类型不一致的时候发生的类型转换，建议先确定where中的参数类型。 where age='18'

### 3.15 对于联合索引来说，要遵守最左前缀法则

举例来说索引含有字段id、name、school，可以直接用id字段，也可以id、name这样的顺序，但是name;school都无法使用这个索引。所以在创建联合索引的时候一定要注意索引字段顺序，常用的查询字段放在最前面。

### 3.16 必要时可以使用force index来强制查询走某个索引

有的时候MySQL优化器采取它认为合适的索引来检索SQL语句，但是可能它所采用的索引并不是我们想要的。这时就可以采用forceindex来强制优化器使用我们制定的索引。

### 3.17 注意范围查询语句

对于联合索引来说，如果存在范围查询，比如between、>、<等条件时，会造成后面的索引字段失效。

### 3.18 使用JOIN优化

LEFT JOIN A表为驱动表，INNER JOIN MySQL会自动找出那个数据少的表作用驱动表，RIGHT JOIN B表为驱动表。

**注意：**

1) MySQL中没有full join，可以用以下方式来解决：

```
select * from A left join B on B.name = A.name where B.name is null union all
select * from B;
```

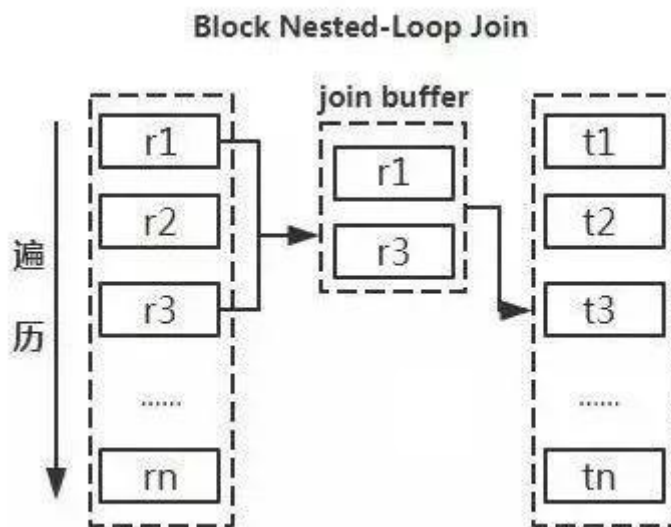
2) 尽量使用inner join，避免left join：

参与联合查询的表至少为2张表，一般都存在大小之分。如果连接方式是inner join，在没有其他过滤条件的情况下MySQL会自动选择小表作为驱动表，但是left join在驱动表的选择上遵循的是左边驱动右边的原则，即left join左边的表名为驱动表。

3) 合理利用索引：

被驱动表的索引字段作为on的限制字段。

4) 利用小表去驱动大表：



从原理图能够直观的看出如果能够减少驱动表的话，减少嵌套循环中的循环次数，以减少IO总量及CPU运算的次数。

## 4、MySQL开发规约

我们知道各大公司都有自己的MySQL开发规约，我们以阿里为例，阿里的MySQL开发规约如下：

### 4.1 建表规约

1、【强制】表达是与否概念的字段，必须使用 is\_xxx 的方式命名，数据类型是 unsigned tinyint

(1 表示是，0 表示否)。

说明：任何字段如果为非负数，必须是 unsigned。

注意：POJO 类中的任何布尔类型的变量，都不要加 is 前缀，所以，需要在设置

从 is\_xxx 到 Xxx 的映射关系。数据库表示是与否的值，使用 tinyint 类型，坚持 is\_xxx 的命名方式是为了明确其取值含义与取值范围。

正例：表达逻辑删除的字段名 is\_deleted，1 表示删除，0 表示未删除。

2、【强制】表名、字段名必须使用小写字母或数字，禁止出现数字开头，禁止两个下划线中间只出现数字。数据库字段名的修改代价很大，因为无法进行预发布，所以字段名称需要慎重考虑。

说明：MySQL 在 Windows 下不区分大小写，但在 Linux 下默认是区分大小写。因此，数据库名、表名、字段名，都不允许出现任何大写字母，避免节外生枝。

正例：aliyun\_admin, rdc\_config, level3\_name

反例：AliyunAdmin, rdcConfig, level\_3\_name

3、【强制】表名不使用复数名词。

说明：表名应该仅仅表示表里面的实体内容，不应该表示实体数量，对应于 DO 类名也是单数形式，符合表达习惯。

4、【强制】禁用保留字，如 desc、range、match、delayed 等，请参考 MySQL 官方保留字。

5、【强制】主键索引名为 pk\_字段名；唯一索引名为 uk\_字段名；普通索引名则为 idx\_字段名。

说明：pk\_ 即 primary key；uk\_ 即 unique key；idx\_ 即 index 的简称。

6、【强制】小数类型为 decimal，禁止使用 float 和 double。

说明：float 和 double 在存储的时候，存在精度损失的问题，很可能在值的比较时，得到不正确的结果。如果存储的数据范围超过 decimal 的范围，建议将数据拆成整数和小数分开存储。

7、【强制】如果存储的字符串长度几乎相等，使用 char 定长字符串类型。

8、【强制】varchar 是可变长字符串，不预先分配存储空间，长度不要超过 5000，如果存储长度大于此值，定义字段类型为 text，独立出来一张表，用主键来对应，避免影响其它字段索引效率。

9、【强制】表必备三字段：id, gmt\_create, gmt\_modified。

说明：其中 id 必为主键，类型为 bigint unsigned、单表时自增、步长为 1。gmt\_create, gmt\_modified 的类型均为 datetime 类型，前者现在时表示主动创建，后者过去分词表示被动更新。

10、【推荐】表的命名最好是加上“业务名称\_表的作用”。

正例：alipay\_task / force\_project / trade\_config

11、【推荐】库名与应用名称尽量一致。

12、【推荐】如果修改字段含义或对字段表示的状态追加时，需要及时更新字段注释。

13、【推荐】字段允许适当冗余，以提高查询性能，但必须考虑数据一致。冗余字段应遵循：

1) 不是频繁修改的字段。

2) 不是 varchar 超长字段，更不能是 text 字段。

正例：商品类目名称使用频率高，字段长度短，名称基本一成不变，可在相关联的表中冗余存储类目名称，避免关联查询。

14、【推荐】单表行数超过 500 万行或者单表容量超过 2GB，才推荐进行分库分表。

说明：如果预计三年后的数据量根本达不到这个级别，请不要在创建表时就分库分表。

15、【参考】合适的字符存储长度，不但节约数据库表空间、节约索引存储，更重要的是提升检索速度。

正例：如下表，其中无符号值可以避免误存负数，且扩大了表示范围。

对象	年龄区间	类型	字节	表示范围
人	150 岁之内	tinyint unsigned	1	无符号值：0 到 255
龟	数百岁	smallint unsigned	2	无符号值：0 到 65535
恐龙化石	数千万年	int unsigned	4	无符号值：0 到约 42.9 亿
太阳	约 50 亿年	bigint unsigned	8	无符号值：0 到约 10 的 19 次方

## 4.2 索引规约

1、【强制】业务上具有唯一特性的字段，即使是多个字段的组合，也必须建成唯一索引。

说明：不要以为唯一索引影响了 insert 速度，这个速度损耗可以忽略，但提高查找速度是明显的；另外，即使在应用层做了非常完善的校验控制，只要没有唯一索引，根据墨菲定律，必然有脏数据产生。

2、【强制】三个表以上禁止 join。需要 join 的字段，数据类型必须绝对一致；多表关联查询时，保证被关联的字段需要有索引。

说明：即使双表 join 也要注意表索引、SQL 性能。

3、【强制】在 varchar 字段上建立索引时，必须指定索引长度，没必要对全字段建立索引，根据实际文本区分度决定索引长度即可。

说明：索引的长度与区分度是一对矛盾体，一般对字符串类型数据，长度为 20 的索引，区分度会高达 90%以上，可以使用  $\text{count}(\text{distinct left}(\text{列名}, \text{索引长度}))/\text{count}(\text{*})$  的区分度来确定。

4、【强制】页面搜索严禁左模糊或者全模糊，如果需要请走搜索引擎来解决。

说明：索引文件具有 B-Tree 的最左前缀匹配特性，如果左边的值未确定，那么无法使用此索引。

5、【推荐】如果有 order by 的场景，请注意利用索引的有序性。order by 最后的字段是组合索引的一部分，并且放在索引组合顺序的最后，避免出现 file\_sort 的情况，影响查询性能。

正例：where a=? and b=? order by c; 索引：a\_b\_c

反例：索引中有范围查找，那么索引有序性无法利用，如：WHERE a>10 ORDER BY b; 索引a\_b 无法排序。

6、【推荐】利用覆盖索引来进行查询操作，避免回表。

说明：如果一本书需要知道第 11 章是什么标题，会翻开第 11 章对应的那一页吗？目录浏览

一下就好，这个目录就是起到覆盖索引的作用。

正例：能够建立索引的种类分为主键索引、唯一索引、普通索引三种，而覆盖索引只是一种查询的一种效果，用 explain 的结果，extra 列会出现：using index。

7、【推荐】利用延迟关联或者子查询优化超多分页场景。

说明：MySQL 并不是跳过 offset 行，而是取 offset+N 行，然后返回放弃前 offset 行，返回 N 行，那当 offset 特别大的时候，效率就非常的低下，要么控制返回的总页数，要么对超过特定阈值的页数进行 SQL 改写。

正例：先快速定位需要获取的 id 段，然后再关联：

```
SELECT a.* FROM 表 1 a, (select id from 表 1 where 条件 LIMIT 100000,20 ) b where a.id=b.id
```

8、【推荐】SQL 性能优化的目标：至少要达到 range 级别，要求是 ref 级别，如果可以是 consts 最好。

说明：

- 1) consts 单表中最多只有一个匹配行（主键或者唯一索引），在优化阶段即可读取到数据。
- 2) ref 指的是使用普通的索引（normal index）。
- 3) range 对索引进行范围检索。

反例：explain 表的结果，type=index，索引物理文件全扫描，速度非常慢，这个 index 级别比较 range 还低，与全表扫描是小巫见大巫。

9、【推荐】建组合索引的时候，区分度最高的在最左边

正例：如果 where a=? and b=?，如果 a 列的几乎接近于唯一值，那么只需要单建 idx\_a 索引即可。

说明：存在非等号和等号混合时，在建索引时，请把等号条件的列前置。如：where c>? and d=? 那么即使 c 的区分度更高，也必须把 d 放在索引的最前列，即索引 idx\_d\_c。

10、【推荐】防止因字段类型不同造成的隐式转换，导致索引失效。

11、【参考】创建索引时避免有如下极端误解

- 1) 宁滥勿缺。认为一个查询就需要建一个索引。
- 2) 宁缺勿滥。认为索引会消耗空间、严重拖慢更新和新增速度。
- 3) 抵制惟一索引。认为业务的惟一性一律需要在应用层通过“先查后插”方式解决。

## 4.3 SQL 语句

1、【强制】不要使用 count(列名)或 count(常量)来替代 count()，count()是 SQL92 定义的标准统计行数的语法，跟数据库无关，跟 NULL 和非 NULL 无关。

说明：count(\*)会统计值为 NULL 的行，而 count(列名)不会统计此列为 NULL 值的行。

2、【强制】count(distinct col) 计算该列除 NULL 之外的不重复行数，注意 count(distinct col1, col2) 如果其中一列全为 NULL，那么即使另一列有不同的值，也返回为 0。

3、【强制】当某一列的值全是 NULL 时，count(col)的返回结果为 0，但 sum(col)的返回结果为 NULL，因此使用 sum()时需注意 NPE (Null Pointer Exception)问题。

正例：可以使用如下方式来避免 sum 的 NPE 问题：SELECT IF(ISNULL(SUM(g)),0,SUM(g))FROM table;

4、【强制】使用 ISNULL()来判断是否为 NULL 值。

说明：NULL 与任何值的直接比较都为 NULL。

1) NULL<>NULL 的返回结果是 NULL，而不是 false。

2) NULL=NULL 的返回结果是 NULL，而不是 true。

3) NULL<>1 的返回结果是 NULL，而不是 true。

5、【强制】在代码中写分页查询逻辑时，若 count 为 0 应直接返回，避免执行后面的分页语句。

6、【强制】不得使用外键与级联，一切外键概念必须在应用层解决。

说明：以学生和成绩的关系为例，学生表中的 student\_id 是主键，那么成绩表中的 student\_id

则为外键。如果更新学生表中的 student\_id，同时触发成绩表中的 student\_id 更新，即为

级联更新。外键与级联更新适用于单机低并发，不适合分布式、高并发集群；级联更新是强阻

塞，存在数据库更新风暴的风险；外键影响数据库的插入速度。

7、【强制】禁止使用存储过程，存储过程难以调试和扩展，更没有移植性。

8、【强制】数据订正（特别是删除、修改记录操作）时，要先 select，避免出现误删除，确认无误才能执行更新 语句。

9、【推荐】in 操作能避免则避免，若实在避免不了，需要仔细评估 in 后边的集合元素数量，控制在 1000 个

之内。

10、【参考】如果有国际化需要，所有的字符存储与表示，均以 utf-8 编码，注意字符统计函数的区别。

说明：

SELECT LENGTH("轻松工作"); 返回为 12

SELECT CHARACTER\_LENGTH("轻松工作"); 返回为 4

如果需要存储表情，那么选择 utf8mb4 来进行存储，注意它与 utf-8 编码的区别。

11、【参考】TRUNCATE TABLE 比 DELETE 速度快，且使用的系统和事务日志资源少，但 TRUNCATE 无事务且不 触发 trigger，有可能造成事故，故不建议在开发代码中使用此语句。

说明：TRUNCATE TABLE 在功能上与不带 WHERE 子句的 DELETE 语句相同

## 4.4 ORM映射

1. 【强制】在表查询中，一律不要使用 \* 作为查询的字段列表，需要哪些字段必须明确写明。

说明：1) 增加查询分析器解析成本。2) 增减字段容易与 resultMap 配置不一致。3) 无用字段增加网络消耗，尤其是 text 类型的字段。

2. 【强制】POJO 类的布尔属性不能加 is，而数据库字段必须加 is\_，要求在 resultMap 中进行 字段与属性之间的映射。

说明：参见定义 POJO 类以及数据库字段定义规定，在中增加映射，是必须的。

在 MyBatis Generator 生成的代码中，需要进行对应的修改。

3. 【强制】不要用 resultClass 当返回参数，即使所有类属性名与数据库字段一一对应，也需要定义；反过来，每一个表也必然有一个 POJO 类与之对应。

说明：配置映射关系，使字段与 DO 类解耦，方便维护。

4. 【强制】sql.xml 配置参数使用：#{}, #param# 不要使用\${} 此种方式容易出现 SQL 注入。
5. 【强制】iBATIS 自带的 queryForList(String statementName,int start,int size)不推荐使用。  
说明：其实现方式是在数据库取到statementName对应的SQL语句的所有记录，再通过subList 取 start,size 的子集合。  
正例：Map<String, Object> map = new HashMap<>();  
map.put("start", start);  
map.put("size", size);
6. 【强制】不允许直接拿 HashMap 与 Hashtable 作为查询结果集的输出。  
说明：resultClass="Hashtable"，会置入字段名和属性值，但是值的类型不可控。
7. 【强制】更新数据表记录时，必须同时更新记录对应的 gmt\_modified 字段值为当前时间。
8. 【推荐】不要写一个大而全的数据更新接口。传入为 POJO 类，不管是不是自己的目标更新字段，都进行 update table set c1=value1,c2=value2,c3=value3; 这是不对的。执行 SQL 时，不要更新无改动的字段，一是易出错；二是效率低；三是增加 binlog 存储。
9. 【参考】@Transactional 事务不要滥用。事务会影响数据库的 QPS，另外使用事务的地方需要考虑各方面的回滚方案，包括缓存回滚、搜索引擎回滚、消息补偿、统计修正等。
10. 【参考】中的 compareValue 是与属性值对比的常量，一般是数字，表示相等时带上此条件；表示不为空且不为 null 时执行；表示不为 null 值时执行。

## 5、复杂SQL优化实战

### 优化案例

前面用过的tbiguser表有10000000条记录

创建tuser1表和tuser2表，并初始化若干的数据。

```
create table tuser1(
  id int primary key auto_increment,
  name varchar(255),
  address varchar(255)
);

create table tuser2(
  id int primary key auto_increment,
  name varchar(255),
  address varchar(255)
);

mysql> select * from tuser1 ;
+----+-----+-----+
| id | name   | address |
+----+-----+-----+
|  1 | zhangfei | tianjin |
|  2 | zhaoyun | tianjin |
|  3 | diaochan | guangzhou |
|  4 | diaochan | xianggang |
|  5 | diaochan | hebei    |
|  6 | diaochan | dongbei  |
|  7 | diaochan | dongbei  |
|  8 | diaochan | dongbei  |
```

9	diaochan	dongbei
10	diaochan	dongbei
11	1	1
12	1	1
13	1	1
14	1	1
15	1	1
16	1	1
17	1	1
18	1	1
19	1	1
20	1	1

```
+-----+
20 rows in set (0.00 sec)
```

```
mysql> select * from tuser2;
```

id	name	address
1	zhangfei	shanghai
2	zhaozun	shanghai
3	diaochan	guangzhou
4	diaochan	xianggang
5	diaochan	hebei
6	diaochan	dongbei
7	diaochan	dongbei
8	diaochan	dongbei
9	diaochan	dongbei
10	diaochan	dongbei
11	1	1
12	1	1
13	1	1
14	1	1
15	1	1
16	1	1
17	1	1
18	1	1
19	1	1
20	1	1

```
+-----+
20 rows in set (0.00 sec)
```

可以看到tuser1和tuser2表有重复的数据。

需求：tbiguser表按照地区分组统计求和，要求是在tuser1表和tuser2表中出现过的地区

按照需求写出SQL：



```
mysql> select count(id) num , address from tbiguser where address in (select
distinct address from tuser1) group by address union select count(id) num ,
address from tbiguser where address in (select distinct address from tuser2)
group by address ;
+-----+-----+
| num | address |
+-----+-----+
| 105 | tianjin |
| 100 | shanghai |
+-----+-----+
2 rows in set (14.43 sec)
```

通过explain可以看到:

```
mysql> explain select count(id) num , address from tbiguser where address in
(select distinct address from tuser1) group by address union select
count(id) num , address from tbiguser where address in (select distinct
address from tuser2) group by address ;
+----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref |
| rows | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+
| 1 | PRIMARY | <subquery2> | ALL | NULL | NULL | NULL | NULL |
| NULL | Using temporary; Using filesort |
| 1 | PRIMARY | tbiguser | ALL | NULL | NULL | NULL | NULL |
| 9754360 | Using where; Using join buffer (Block Nested Loop) |
| 2 | MATERIALIZED | tuser1 | ALL | NULL | NULL | NULL | NULL |
| 20 | NULL |
| 3 | UNION | <subquery4> | ALL | NULL | NULL | NULL | NULL |
| NULL | Using temporary; Using filesort |
| 3 | UNION | tbiguser | ALL | NULL | NULL | NULL | NULL |
| 9754360 | Using where; Using join buffer (Block Nested Loop) |
| 4 | MATERIALIZED | tuser2 | ALL | NULL | NULL | NULL | NULL |
| 20 | NULL |
| NULL | UNION RESULT | <union1,3> | ALL | NULL | NULL | NULL |
| NULL | NULL | Using temporary |
+----+-----+-----+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)
```

type:为ALL 说明没有索引, 全表扫描

Using temporary: 说明使用了临时表

Using filesort : 说明使用了文件排序

Using where: 没有索引下推, 在Server层进行了全表扫描和过滤

Using join buffer(Block Nested Loop): 关联没有索引, 有关联优化

### 第一次优化:

给address加索引

```
--给address加索引
alter table tbiguser add index idx_addr(address);
```

```
alter table tuser1 add index idx_addr(address);
alter table tuser2 add index idx_addr(address);
```

--再次运行SQL

```
select count(id) num , address from tbiguser where address in (select
distinct address from tuser1) group by address union select count(id) num ,
address from tbiguser where address in (select distinct address from tuser2)
group by address ;
```

```
+-----+-----+
| num | address |
+-----+-----+
| 105 | tianjin |
| 100 | shanghai |
+-----+-----+
2 rows in set (13.61 sec)
```

--查看执行计划

```
mysql> explain select count(id) num , address from tbiguser where address in
(select distinct address from tuser1) group by address union select
count(id) num , address from tbiguser where address in (select distinct
address from tuser2) group by address ;
```

```
+-----+-----+-----+-----+-----+-----+-----+
--+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key |
key_len | ref | rows | Extra |
+-----+-----+-----+-----+-----+-----+-----+
--+-----+-----+-----+-----+-----+-----+-----+
| 1 | PRIMARY | tbiguser | index | idx_addr | idx_addr | 768
| NULL | | 9754360 | Using where; Using index |
| 1 | PRIMARY | <subquery2> | eq_ref | <auto_key> | <auto_key> | 768
| demo.tb用户.address | 1 | NULL |
| 2 | MATERIALIZED | tuser1 | index | idx_addr | idx_addr | 768
| NULL | | 20 | Using index |
| 3 | UNION | tbiguser | index | idx_addr | idx_addr | 768
| NULL | | 9754360 | Using where; Using index |
| 3 | UNION | <subquery4> | eq_ref | <auto_key> | <auto_key> | 768
| demo.tb用户.address | 1 | NULL |
| 4 | MATERIALIZED | tuser2 | index | idx_addr | idx_addr | 768
| NULL | | 20 | Using index |
| NULL | UNION RESULT | <union1,3> | ALL | NULL | NULL | NULL
| NULL | | NULL | Using temporary |
+-----+-----+-----+-----+-----+-----+-----+
--+-----+-----+-----+-----+-----+-----+-----+
```

type: index , 说明用到了索引: 覆盖索引

Using temporary : 有临时表

Using where : 没有索引下推, 在Server层进行了全表扫描和过滤

**第二次优化:**

--修改sql

```
select count(id) num , address from tbiguser where address in (select distinct
address from tuser1) or address in (select distinct address from tuser2) group
by address order by address;
```

```
+-----+-----+
| num | address |
```

```

+-----+-----+
| 100 | shanghai |
| 105 | tianjin  |
+-----+-----+
2 rows in set (3.54 sec)
--运行执行计划
explain select count(id) num , address from tbiguser where address in (select
distinct address from tuser1) or address in (select distinct address from
tuser2) group by address order by address;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
--+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref |
| rows | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
--+-----+-----+-----+
| 1 | PRIMARY | tbiguser | index | idx_addr | idx_addr | 768 | |
NULL | 9754360 | Using where; Using index |
| 3 | SUBQUERY | tuser2 | index | idx_addr | idx_addr | 768 | |
NULL | 20 | Using index |
| 2 | SUBQUERY | tuser1 | index | idx_addr | idx_addr | 768 | |
NULL | 20 | Using index |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
--+-----+-----+-----+
3 rows in set (0.00 sec)

```

type: index

没有了临时表

### 第三次优化:

从前面的执行计划可以看出，索引只是使用了覆盖索引，rows=9754360，说明还是几乎扫描了全表的行

利用address索引，先过滤数据

```

mysql> select distinct b.* from tuser1 a,tbiguser b where a.address=b.address;
+-----+-----+-----+-----+-----+-----+-----+-----+
| id | nickname | loginname | age | sex | status | address |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 101 | zy101 | zhaoyun101 | 23 | 1 | 1 | tianjin |
| 102 | zy102 | zhaoyun102 | 23 | 1 | 1 | tianjin |
| 103 | zy103 | zhaoyun103 | 23 | 1 | 1 | tianjin |
| 104 | zy104 | zhaoyun104 | 23 | 1 | 1 | tianjin |
| 105 | zy105 | zhaoyun105 | 23 | 1 | 1 | tianjin |
| 106 | zy106 | zhaoyun106 | 23 | 1 | 1 | tianjin |
| 107 | zy107 | zhaoyun107 | 23 | 1 | 1 | tianjin |
| 108 | zy108 | zhaoyun108 | 23 | 1 | 1 | tianjin |
| 109 | zy109 | zhaoyun109 | 23 | 1 | 1 | tianjin |
| 110 | zy110 | zhaoyun110 | 23 | 1 | 1 | tianjin |
| 111 | zy111 | zhaoyun111 | 23 | 1 | 1 | tianjin |
| 112 | zy112 | zhaoyun112 | 23 | 1 | 1 | tianjin |
| 113 | zy113 | zhaoyun113 | 23 | 1 | 1 | tianjin |
| 114 | zy114 | zhaoyun114 | 23 | 1 | 1 | tianjin |
| 115 | zy115 | zhaoyun115 | 23 | 1 | 1 | tianjin |
| 116 | zy116 | zhaoyun116 | 23 | 1 | 1 | tianjin |
| 117 | zy117 | zhaoyun117 | 23 | 1 | 1 | tianjin |
| 118 | zy118 | zhaoyun118 | 23 | 1 | 1 | tianjin |

```

119	zy119	zhaoyun119	23	1	1	tianjin	
120	zy120	zhaoyun120	23	1	1	tianjin	
121	zy121	zhaoyun121	23	1	1	tianjin	
122	zy122	zhaoyun122	23	1	1	tianjin	
123	zy123	zhaoyun123	23	1	1	tianjin	
124	zy124	zhaoyun124	23	1	1	tianjin	
125	zy125	zhaoyun125	23	1	1	tianjin	
126	zy126	zhaoyun126	23	1	1	tianjin	
127	zy127	zhaoyun127	23	1	1	tianjin	
128	zy128	zhaoyun128	23	1	1	tianjin	
129	zy129	zhaoyun129	23	1	1	tianjin	
130	zy130	zhaoyun130	23	1	1	tianjin	
131	zy131	zhaoyun131	23	1	1	tianjin	
132	zy132	zhaoyun132	23	1	1	tianjin	
133	zy133	zhaoyun133	23	1	1	tianjin	
134	zy134	zhaoyun134	23	1	1	tianjin	
135	zy135	zhaoyun135	23	1	1	tianjin	
136	zy136	zhaoyun136	23	1	1	tianjin	
137	zy137	zhaoyun137	23	1	1	tianjin	
138	zy138	zhaoyun138	23	1	1	tianjin	
139	zy139	zhaoyun139	23	1	1	tianjin	
140	zy140	zhaoyun140	23	1	1	tianjin	
141	zy141	zhaoyun141	23	1	1	tianjin	
142	zy142	zhaoyun142	23	1	1	tianjin	
143	zy143	zhaoyun143	23	1	1	tianjin	
144	zy144	zhaoyun144	23	1	1	tianjin	
145	zy145	zhaoyun145	23	1	1	tianjin	
146	zy146	zhaoyun146	23	1	1	tianjin	
147	zy147	zhaoyun147	23	1	1	tianjin	
148	zy148	zhaoyun148	23	1	1	tianjin	
149	zy149	zhaoyun149	23	1	1	tianjin	
150	zy150	zhaoyun150	23	1	1	tianjin	
151	zy151	zhaoyun151	23	1	1	tianjin	
152	zy152	zhaoyun152	23	1	1	tianjin	
153	zy153	zhaoyun153	23	1	1	tianjin	
154	zy154	zhaoyun154	23	1	1	tianjin	
155	zy155	zhaoyun155	23	1	1	tianjin	
156	zy156	zhaoyun156	23	1	1	tianjin	
157	zy157	zhaoyun157	23	1	1	tianjin	
158	zy158	zhaoyun158	23	1	1	tianjin	
159	zy159	zhaoyun159	23	1	1	tianjin	
160	zy160	zhaoyun160	23	1	1	tianjin	
161	zy161	zhaoyun161	23	1	1	tianjin	
162	zy162	zhaoyun162	23	1	1	tianjin	
163	zy163	zhaoyun163	23	1	1	tianjin	
164	zy164	zhaoyun164	23	1	1	tianjin	
165	zy165	zhaoyun165	23	1	1	tianjin	
166	zy166	zhaoyun166	23	1	1	tianjin	
167	zy167	zhaoyun167	23	1	1	tianjin	
168	zy168	zhaoyun168	23	1	1	tianjin	
169	zy169	zhaoyun169	23	1	1	tianjin	
170	zy170	zhaoyun170	23	1	1	tianjin	
171	zy171	zhaoyun171	23	1	1	tianjin	
172	zy172	zhaoyun172	23	1	1	tianjin	
173	zy173	zhaoyun173	23	1	1	tianjin	
174	zy174	zhaoyun174	23	1	1	tianjin	
175	zy175	zhaoyun175	23	1	1	tianjin	
176	zy176	zhaoyun176	23	1	1	tianjin	

```

| 177 | zy177 | zhaoyun177 | 23 | 1 | 1 | tianjin |
| 178 | zy178 | zhaoyun178 | 23 | 1 | 1 | tianjin |
| 179 | zy179 | zhaoyun179 | 23 | 1 | 1 | tianjin |
| 180 | zy180 | zhaoyun180 | 23 | 1 | 1 | tianjin |
| 181 | zy181 | zhaoyun181 | 23 | 1 | 1 | tianjin |
| 182 | zy182 | zhaoyun182 | 23 | 1 | 1 | tianjin |
| 183 | zy183 | zhaoyun183 | 23 | 1 | 1 | tianjin |
| 184 | zy184 | zhaoyun184 | 23 | 1 | 1 | tianjin |
| 185 | zy185 | zhaoyun185 | 23 | 1 | 1 | tianjin |
| 186 | zy186 | zhaoyun186 | 23 | 1 | 1 | tianjin |
| 187 | zy187 | zhaoyun187 | 23 | 1 | 1 | tianjin |
| 188 | zy188 | zhaoyun188 | 23 | 1 | 1 | tianjin |
| 189 | zy189 | zhaoyun189 | 23 | 1 | 1 | tianjin |
| 190 | zy190 | zhaoyun190 | 23 | 1 | 1 | tianjin |
| 191 | zy191 | zhaoyun191 | 23 | 1 | 1 | tianjin |
| 192 | zy192 | zhaoyun192 | 23 | 1 | 1 | tianjin |
| 193 | zy193 | zhaoyun193 | 23 | 1 | 1 | tianjin |
| 194 | zy194 | zhaoyun194 | 23 | 1 | 1 | tianjin |
| 195 | zy195 | zhaoyun195 | 23 | 1 | 1 | tianjin |
| 196 | zy196 | zhaoyun196 | 23 | 1 | 1 | tianjin |
| 197 | zy197 | zhaoyun197 | 23 | 1 | 1 | tianjin |
| 198 | zy198 | zhaoyun198 | 23 | 1 | 1 | tianjin |
| 199 | zy199 | zhaoyun199 | 23 | 1 | 1 | tianjin |
| 200 | zy200 | zhaoyun200 | 23 | 1 | 1 | tianjin |
| 201 | zy201 | zhaoyun201 | 23 | 1 | 1 | tianjin |
| 202 | zy202 | zhaoyun202 | 23 | 1 | 1 | tianjin |
| 203 | zy203 | zhaoyun203 | 23 | 1 | 1 | tianjin |
| 204 | zy204 | zhaoyun204 | 23 | 1 | 1 | tianjin |
| 205 | zy205 | zhaoyun205 | 23 | 1 | 1 | tianjin |

```

```
+-----+-----+-----+-----+-----+-----+-----+
```

105 rows in set (0.00 sec)

--查看执行计划

```
mysql> explain select distinct b.* from tuser1 a,tbiguser b where
a.address=b.address;
```

```

+----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref |
|----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | a | index | idx_addr | idx_addr | 768 | NULL |
| 20 | Using where; Using index; Using temporary |
| 1 | SIMPLE | b | ref | idx_addr | idx_addr | 768 |
demo.a.address | 2438590 | NULL |

```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
```

```
2 rows in set (0.00 sec)
```

type: ref

rows: 2438590

说明使用了address索引做关联

同理:

```
mysql> select distinct b.* from tuser2 a,tbiguser b where a.address=b.address;
```

id	nickname	loginname	age	sex	status	address
1	zy1	zhaoyun1	23	1	1	shanghai
2	zy2	zhaoyun2	23	1	1	shanghai
3	zy3	zhaoyun3	23	1	1	shanghai
4	zy4	zhaoyun4	23	1	1	shanghai
5	zy5	zhaoyun5	23	1	1	shanghai
6	zy6	zhaoyun6	23	1	1	shanghai
7	zy7	zhaoyun7	23	1	1	shanghai
8	zy8	zhaoyun8	23	1	1	shanghai
9	zy9	zhaoyun9	23	1	1	shanghai
10	zy10	zhaoyun10	23	1	1	shanghai
11	zy11	zhaoyun11	23	1	1	shanghai
12	zy12	zhaoyun12	23	1	1	shanghai
13	zy13	zhaoyun13	23	1	1	shanghai
14	zy14	zhaoyun14	23	1	1	shanghai
15	zy15	zhaoyun15	23	1	1	shanghai
16	zy16	zhaoyun16	23	1	1	shanghai
17	zy17	zhaoyun17	23	1	1	shanghai
18	zy18	zhaoyun18	23	1	1	shanghai
19	zy19	zhaoyun19	23	1	1	shanghai
20	zy20	zhaoyun20	23	1	1	shanghai
21	zy21	zhaoyun21	23	1	1	shanghai
22	zy22	zhaoyun22	23	1	1	shanghai
23	zy23	zhaoyun23	23	1	1	shanghai
24	zy24	zhaoyun24	23	1	1	shanghai
25	zy25	zhaoyun25	23	1	1	shanghai
26	zy26	zhaoyun26	23	1	1	shanghai
27	zy27	zhaoyun27	23	1	1	shanghai
28	zy28	zhaoyun28	23	1	1	shanghai
29	zy29	zhaoyun29	23	1	1	shanghai
30	zy30	zhaoyun30	23	1	1	shanghai
31	zy31	zhaoyun31	23	1	1	shanghai
32	zy32	zhaoyun32	23	1	1	shanghai
33	zy33	zhaoyun33	23	1	1	shanghai
34	zy34	zhaoyun34	23	1	1	shanghai
35	zy35	zhaoyun35	23	1	1	shanghai
36	zy36	zhaoyun36	23	1	1	shanghai
37	zy37	zhaoyun37	23	1	1	shanghai
38	zy38	zhaoyun38	23	1	1	shanghai
39	zy39	zhaoyun39	23	1	1	shanghai
40	zy40	zhaoyun40	23	1	1	shanghai
41	zy41	zhaoyun41	23	1	1	shanghai
42	zy42	zhaoyun42	23	1	1	shanghai
43	zy43	zhaoyun43	23	1	1	shanghai
44	zy44	zhaoyun44	23	1	1	shanghai
45	zy45	zhaoyun45	23	1	1	shanghai
46	zy46	zhaoyun46	23	1	1	shanghai
47	zy47	zhaoyun47	23	1	1	shanghai
48	zy48	zhaoyun48	23	1	1	shanghai
49	zy49	zhaoyun49	23	1	1	shanghai
50	zy50	zhaoyun50	23	1	1	shanghai
51	zy51	zhaoyun51	23	1	1	shanghai
52	zy52	zhaoyun52	23	1	1	shanghai
53	zy53	zhaoyun53	23	1	1	shanghai
54	zy54	zhaoyun54	23	1	1	shanghai
55	zy55	zhaoyun55	23	1	1	shanghai

56	zy56	zhaoyun56	23	1	1	shanghai
57	zy57	zhaoyun57	23	1	1	shanghai
58	zy58	zhaoyun58	23	1	1	shanghai
59	zy59	zhaoyun59	23	1	1	shanghai
60	zy60	zhaoyun60	23	1	1	shanghai
61	zy61	zhaoyun61	23	1	1	shanghai
62	zy62	zhaoyun62	23	1	1	shanghai
63	zy63	zhaoyun63	23	1	1	shanghai
64	zy64	zhaoyun64	23	1	1	shanghai
65	zy65	zhaoyun65	23	1	1	shanghai
66	zy66	zhaoyun66	23	1	1	shanghai
67	zy67	zhaoyun67	23	1	1	shanghai
68	zy68	zhaoyun68	23	1	1	shanghai
69	zy69	zhaoyun69	23	1	1	shanghai
70	zy70	zhaoyun70	23	1	1	shanghai
71	zy71	zhaoyun71	23	1	1	shanghai
72	zy72	zhaoyun72	23	1	1	shanghai
73	zy73	zhaoyun73	23	1	1	shanghai
74	zy74	zhaoyun74	23	1	1	shanghai
75	zy75	zhaoyun75	23	1	1	shanghai
76	zy76	zhaoyun76	23	1	1	shanghai
77	zy77	zhaoyun77	23	1	1	shanghai
78	zy78	zhaoyun78	23	1	1	shanghai
79	zy79	zhaoyun79	23	1	1	shanghai
80	zy80	zhaoyun80	23	1	1	shanghai
81	zy81	zhaoyun81	23	1	1	shanghai
82	zy82	zhaoyun82	23	1	1	shanghai
83	zy83	zhaoyun83	23	1	1	shanghai
84	zy84	zhaoyun84	23	1	1	shanghai
85	zy85	zhaoyun85	23	1	1	shanghai
86	zy86	zhaoyun86	23	1	1	shanghai
87	zy87	zhaoyun87	23	1	1	shanghai
88	zy88	zhaoyun88	23	1	1	shanghai
89	zy89	zhaoyun89	23	1	1	shanghai
90	zy90	zhaoyun90	23	1	1	shanghai
91	zy91	zhaoyun91	23	1	1	shanghai
92	zy92	zhaoyun92	23	1	1	shanghai
93	zy93	zhaoyun93	23	1	1	shanghai
94	zy94	zhaoyun94	23	1	1	shanghai
95	zy95	zhaoyun95	23	1	1	shanghai
96	zy96	zhaoyun96	23	1	1	shanghai
97	zy97	zhaoyun97	23	1	1	shanghai
98	zy98	zhaoyun98	23	1	1	shanghai
99	zy99	zhaoyun99	23	1	1	shanghai
100	zy100	zhaoyun100	23	1	1	shanghai

```
+-----+-----+-----+-----+-----+-----+
100 rows in set (0.00 sec)
```

```
--查看执行计划
```

```
mysql> explain select distinct b.* from tuser2 a,tbiguser b where
a.address=b.address;
```

```
+---+-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref |
|----|-----|-----|-----|-----|-----|-----|-----|
+---+-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+-----+-----+
```

```

| 1 | SIMPLE | a | index | idx_addr | idx_addr | 768 | NULL
| 20 | Using where; Using index; Using temporary |
| 1 | SIMPLE | b | ref | idx_addr | idx_addr | 768 |
demo.a.address | 2438590 | NULL
+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

```

type: ref

rows: 2438590

说明使用了address索引做关联

合并结果集后再分组求和

```

select count(x.id),x.address
from
(select distinct b.* from tuser1 a,tbiguser b where a.address=b.address union
all select distinct b.* from tuser2 a,tbiguser b where a.address=b.address) x
group by x.address;

+-----+-----+
| count(x.id) | address |
+-----+-----+
| 100 | shanghai |
| 105 | tianjin |
+-----+-----+
2 rows in set (0.00 sec)

--查看执行计划
mysql> explain select count(x.id),x.address
-> from
-> (select distinct b.* from tuser1 a,tbiguser b where a.address=b.address
union all select distinct b.* from tuser2 a,tbiguser b where
a.address=b.address) x group by x.address;

+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len |
ref | rows | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | PRIMARY | <derived2> | ALL | NULL | NULL | NULL |
NULL | 97543600 | Using temporary; Using filesort |
| 2 | DERIVED | a | index | idx_addr | idx_addr | 768 |
NULL | 20 | Using where; Using index; Using temporary |
| 2 | DERIVED | b | ref | idx_addr | idx_addr | 768 |
demo.a.address | 2438590 | Distinct |
| 3 | UNION | a | index | idx_addr | idx_addr | 768 |
NULL | 20 | Using where; Using index; Using temporary |
| 3 | UNION | b | ref | idx_addr | idx_addr | 768 |
demo.a.address | 2438590 | Distinct |
| NULL | UNION RESULT | <union2,3> | ALL | NULL | NULL | NULL |
| NULL | NULL | Using temporary |
+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)

```



DERIVED: 派生表

## 最终优化

将派生表写成视图

```
--创建视图
create view v_tuser as select distinct b.* from tuser1 a,tbiguser b where
a.address=b.address union all select distinct b.* from tuser2 a,tbiguser b where
a.address=b.address;

--执行SQL
select count(id) cont ,address from v_tuser group by address order by address;
+-----+-----+
| cont | address |
+-----+-----+
| 100 | shanghai |
| 105 | tianjin |
+-----+-----+
2 rows in set (0.00 sec)
```

**优化结果：从最初的将近14秒优化到不到1秒。**

**优化总结：**

- 开启慢查询日志，定位运行慢的SQL语句
- 利用explain执行计划，查看SQL执行情况
- 关注索引使用情况：type
- 关注Rows：行扫描
- 关注Extra：没有信息最好
- 加索引后，查看索引使用情况，index只是覆盖索引，并不算很好的使用索引
- 如果有关联尽量将索引用到eq\_ref或ref级别
- 复杂SQL可以做成视图，视图在MySQL内部有优化，而且开发也比较友好
- 对于复杂的SQL要逐一分析，找到比较费时的SQL语句片段进行优化