

后盾网 人人做后盾

www.houdunwang.com

MySQL优化

后盾网 2011-2016 v5.0

选择合理范围内最小的

- 我们应该选择最小的数据范围，因为这样可以大大减少磁盘空间及磁盘I/O读写开销，减少内存占用，减少CPU的占用率。

选择相对简单的数据类型

- 数字类型相对字符串类型要简单的多，尤其是在比较运算是，所以我们应该选择最简单的数据类型，比如说在保存时间时，因为PHP可以良好的处理Linux时间戳所以我们可以将日期存为int(10)要方便、合适、快速的多。

不要使用null

- 为什么这么说呢，因为MySQL对NULL字段索引优化不佳，增加更多的计算难度，同时在保存与处理NULL类型时，也会做更多的工作，所以从效率上来说，不建议用过多的NULL。有些值他确实有可能没有值，怎么办呢？解决方法是数值用整数0，字符串用空来定义默认值即可。

字段类型的选择

- 字符串数据类型是一个万能数据类型，可以储存数值、字符串等。
- 保存数值类型最好不要用字符串数据类型，这样存储的空间显然是会更大，而且在排序时字符串的9是大于22的。如果进行运算时mysql会将字符串转换为数值类型，这种转换是不会走索引的。
- 如果明确数据在一个完整的集合中如男，女，那么可以使用set或enum数据类型，这种数据类型在运算及储存时以数值方式操作，所以效率要比字符串更好，同时空间占用更少

字符串类型的使用

整数

- 整数类型很多比如tinyint、int、smallint、bigint等，那么我们要根据自己需要存储的数据长度决定使用的类型，同时tinyint(10)与tinyint(100)在储存与计算上并无任何差别，区别只是显示层面上，但是我们也要选择适合合适的数据类型长度。可以通过指定zerofill属性查看显示时区别。

浮点数与精度数值

- 浮点数float与double在储存空间及运行效率上要优于精度数值类型decimal，但float与double会有舍入错误而decimal则可以提供更加准确的小数级精确运算不会有错误产生计算更精确，适用于金融类型数据的存储。

总结

- 数值数据类型要比字符串执行更快，区间小的数据类型占用空间更少，处理速度更快，如tinyint可比bigint要快的多
- 选择数据类型时要考虑内容长度，比如是保存毫米单位还是米而选择不同的数值类型

数值类型的选择

什么是索引

- 索引就像一本书的目录一样，我们可以通过一本书的目录，快速的找到需要的页面，但是我们也不能过多的创建目录页，原因是如果某一篇文章删除或修改将发变所有页码的顺序，就需要重新创建目录。
- `select sname from stu where sname="李四"`
- 如果sname使用了索引，上面这个例子就会使用到sname索引

索引

UNIQUE唯一索引

- 不可以出现相同的值，可以有NULL值

INDEX普通索引

- 允许出现相同的索引内容

PRIMARY KEY主键索引

- 不允许出现相同的值，且不能为NULL值，一个表只能有一个primary_key索引

索引类型

创建索引

ALTER TABLE 表名

ADD 索引类型 (unique,primary key,index)

索引名 (字段名) ;

删除索引

Alter table 表名

Drop index 索引名 (drop primary key 删除主键索引) ;

创建与修改索引

- 索引是加快查询操作的重要手段，如果当发生查询过慢时添加上索引后会发现速度大大改观

| id | uname | sex | cid |
|----|-------|-----|-----|
| 7 | 李四 | 男 | 1 |
| 8 | 小花 | 女 | 4 |
| 9 | 赵云 | 男 | 2 |
| 10 | 张三 | 男 | 4 |
| 11 | 马五 | 男 | 1 |
| 12 | 小玉 | 女 | 2 |
| 13 | 后盾向军 | 男 | 3 |

- 如果在上表中查找cid为3的，那么会对整个表进行一行一行的全表扫描，可想而知速度会很慢

索引的优点

如果对cid列创建索引，同样在表中查找cid为3的记录

- 在索引中找到cid为3的行时，而下面的值是4即停止查找，当然数据库在检索索引时也不是从索引表头部开始的，根据相应算法可以快速定位到索引行,也就是说查找cid为3的可以直接定位到那一行索引

| id | uname | sex | cid |
|----|-------|-----|-----|
| 7 | 李四 | 男 | 1 |
| 8 | 小花 | 女 | 4 |
| 9 | 赵云 | 男 | 2 |
| 10 | 张三 | 男 | 4 |
| 11 | 马五 | 男 | 1 |
| 12 | 小玉 | 女 | 2 |
| 13 | 后盾向军 | 男 | 3 |

| cid |
|-----|
| 1 |
| 1 |
| 2 |
| 2 |
| 3 |
| 4 |
| 4 |

索引的优点

- `select a,b,c from a join b join c on a=b and b=c;`
- 在没有添加索引时会执行 $6*6*6=216$ 次查询，如果数据量很大如各个表都有2000条记录，结果会是8000000000（80亿次）次查询，这个结果是很糟糕的
- 如果添加索引后结果为6次

| a |
|---|
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |

| b |
|---|
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |

| c |
|---|
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |

没有添加索引的查询

- 创建索引会使查询操作变得更加快速，但是会降低增加、删除、更新操作的速度，因为执行这些操作的同时会对索引文件进行重新排序或更新
- 创建过多列的索引会大大增加磁盘空间开销
- 不要盲目的创建索引，只为查询操作频繁的列创建索引

索引的弊端

维度

- 数据列中不重复值出现的个数，维度的最大值是数据行的数量
- 如数据表中存在8行数据a ,b ,c,d,a,b,c,d这个表的维度为4
- 要为维度高的列创建索引
- 性别这样的列不适合创建索引，因为维度过低
- 对where，on或group by 及order by 中出现的列使用索引
- 对较小的数据列使用索引，这样会使索引文件更小，同时内存中也可以装载更多的索引键
- 为较长的字符串使用前缀索引
- 不要过多创建索引，除了增加额外的磁盘空间外，对于DML操作的速度影响很大

什么字段该创建索引

- 如果索引列长度过长，这种列索引时将会产生很大的索引文件，不便于操作，可以使用前缀索引方式进行索引
- 前缀索引应该控制在一个合适的点，控制在0.31黄金值即可

例：

```
select count(distinct(left(title,10)))/count(*) from news
```

注：distinct去除重复

增加前缀索引SQL

```
alter table hd_news add index title(title(30));
```

将标题的索引建立在30，这样可以减少索引文件大小，加快索引查询速度

前缀索引

- 如果对表中的多个字段组合进行索引，可以减少文件索引大小，在使用时速度要优于多个索引
- 如果没有左前索引Mysql不执行索引查询

例：

```
Alter table news add index catid(catid,status);
```

如果我们将字段catid与status, status两个字段进行索引

- Select sname from stu where catid="1"
- 以上SQL语句执行索引
- Select sname from stu where catid="1" and status>1
- 以上查询将经过索引
- Select sname from stu where status=1;
- 以上索引不经过索引操作

组合索引

Mysql会对发出的每条SQL进行分析，决定是否使用索引或全表扫描

- 如果发送一条select * from houdunwang where false; 时Mysql是不会执行查询操作的，因为经过SQL分析器的分析后MySQL已经清楚不会有任何语句符合操作

explain select * from houdunwang where false的结果

```
id: 1
select_type: SIMPLE
table: NULL
type: NULL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: NULL
Extra: Impossible WHERE
1 row in set (0.00 sec)
```

Mysql查询优化程序

Select sname from stu where age+10=30;

- 以上SQL语句不会使用索引，因为所有索引列参与了计算

Select sname from stu where left(date,4) <1990;

- 以上SQL不会使用索引，因为使用了函数运算，原理与上面相同

Select * from houdunwang where uname like '后盾%'

- 以上SQL操作索引

Select * from houdunwang where uname like "%后盾%"

- 以上SQL不操作索引

正则表达式不使用索引，这应该很好理解，所以为什么在SQL中很难看到
regexp关键字的原因

Mysql查询优化程序

- 尽量避免数据转换操作如:select title from news where title=900;假设title为char(20)类型，由于每次比较都要将title转为数值型，虽然使用索引，但是会进行全表数据比对

示例：

1. create table a (a char(10));
 2. explain select * from a where a='1' #走索引
- Explain select * from a where a=1 # 不走索引

字符串与数字比较不使用索引

- EXPLAIN可以帮助开发人员分析SQL问题，explain显示了mysql如何使用索引来处理select语句以及连接表，可以帮助选择更好的索引和写出更优化的查询语句。

```
mysql> explain select * from user where mail='22'\G
***** 1. row *****
      id: 1
    select_type: SIMPLE
        table: user
         type: ref
possible_keys: mail
          key: mail
      key_len: 93
         ref: const
        rows: 1
   Extra: Using index condition
1 row in set (0.00 sec)
```

后盾网 houdunwang.com

EXPLAIN

Explain查询后的type值是我们重点关注的位置，通过type可以看出当前使用索引的情况，下表从快到慢排列出了type的值

| 值 | 说明 |
|-----------------|------------------|
| System | 系统表 |
| Const | 通过常量获得 |
| Eq_ref | 一般通过主键获得，只匹配一条记录 |
| Ref | 被驱动表索引 |
| Fulltext | 全文索引 |
| Ref_or_null | 带空值的索引 |
| Indx_merge | 合并索引结果 |
| Unique_subquery | 子查询返回唯一索引 |
| Index_subquery | 子查询返回索引 |
| Range | 索引区间获得 |
| All | 全表遍历 |

EXPLAIN的type值

```
mysql> explain select birthday from hd_user where birthday<"1990/2/2";
```

结果:

id: 1

select_type: SIMPLE 查询类型（简单查询，联合查询，子查询）

table: hd_user 表名

type: range 区间索引（在小于1990/2/2区间的数据）

possible_keys: birthday 可能用到的索引

key: birthday 实际使用到的索引

key_len: 4 最长的索引宽度

ref: const 数据为常量("1990/2/2")

rows: 1 需要查询1行

Extra: Using where; Using index 执行状态说明

EXPLAIN检测MySQL优化

explain select a,b from a join b on a=b;

如果表a与表b的字段a与b都没有创建索引，查询后的结果如下：

***** 1. row *****

id: 1

select_type: SIMPLE

table: a

type: ALL

ALL全表扫描

possible_keys: NULL

key: NULL

key_len: NULL

ref: NULL

rows: 6289

预计需要读取6289条记录

Extra:

***** 2. row *****

id: 1

select_type: SIMPLE

table: b

type: ALL

全表扫描

possible_keys: NULL

key: NULL

key_len: NULL

ref: NULL

rows: 6709

预计要扫描6709条记录

Extra: Using where; Using join buffer

2 rows in set (0.00 sec)

2张表都执行全表扫描

先对表a执行全表扫描，然后a表中的每一条记录都对b表进行一次全表扫描

扫描行数：6289*6709

多表查询优化

如果为表b添加索引后的结果

***** 1. row *****

id: 1

select_type: SIMPLE

table: a

type: ALL

全表扫描

possible_keys: NULL

key: NULL

key_len: NULL

ref: NULL

rows: 6289

Extra:

***** 2. row *****

id: 1

select_type: SIMPLE

table: b

type: ref

通过引用值 (a表中的a字段值)来在索引中定位

possible_keys: b

key: b

key_len: 5

ref: demo.a.a

rows: 33

预计查找的行数

Extra: Using where; Using index

使用索引

2 rows in set (0.00 sec)

对a表中的每行记录与b表匹配33行
扫描行数: 6289*33

EXPLAIN检测MySQL优化

排序列频繁的列使用索引

- explain select * from user where mail ='houdunwangxj@gmail.com' order by uid
- order by 所用字段uid建立索引

```
hdxj — mysql — 71x15
mysql> explain select * from user where mail ='cc' order by mail\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: user
         type: ref
possible_keys: mail
          key: mail
       key_len: 93
         ref: const
        rows: 1
   Extra: Using index condition
1 row in set (0.00 sec)

mysql>
```

```
hdxj — mysql — 71x15
mysql> explain select * from user where mail ='cc' order by username\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: user
         type: ref
possible_keys: mail
          key: mail
       key_len: 93
         ref: const
        rows: 1
   Extra: Using index condition; Using where; Using filesort
1 row in set (0.00 sec)

mysql>
```

ORDER BY

- select * from user where qq="22";与select * from user where qq=22;结果是不同的

```
mysql> desc select * from user where qq="22"\G
***** 1. row *****
      id: 1
select_type: SIMPLE
   table: user
   type: ref
possible_keys: qq
      key: qq
   key_len: 48
      ref: const
     rows: 1
  Extra: Using index condition
1 row in set (0.00 sec)
```

```
mysql> mysql> desc select * from user where qq=22\G
***** 1. row *****
      id: 1
select_type: SIMPLE
   table: user
   type: ALL
possible_keys: qq
      key: NULL
   key_len: NULL
      ref: NULL
     rows: 368804
  Extra: Using where
1 row in set (0.00 sec)
```

记录条数

后盾网 houdunwang.com

使用相同类型

当Mysql性能下降时，通过开启慢查询来获得哪条SQL语句造成的响应过慢，进行分析处理。当然开启慢查询会带来CPU损耗与日志记录的IO开销，所以我们要间断性的打开慢查询日志来查看Mysql运行状态。

慢查询能记录下所有执行超过long_query_time时间的SQL语句, 用于找到执行慢的SQL, 方便我们对这些SQL进行优化.

是否开启慢查询

- `show variables like "%slow%";`

查询慢查询SQL状况

- `show status like "%slow%"`

慢查询时间

- `show variables like "long_query_time"`

慢查询slow

开启记录慢查询

- 修改mysql配置文件my.ini加入
 1. `slow_query_log = on`
 2. `slow_query_log_file = d:/mysql_slow_houdunwang.log`
 3. `long_query_time = 2`低版本mysql使用以下方式指定日志show variables like “%slow%”查看确定
 - `log-slow-queries = d:\wamp\www\mysqlslowquery.log`

执行慢查询SQL

- `select sleep(3);`
- 也可以通过php循环插入多条记录不加索引进行测试

对慢查询的SQL我们要结合explain语句分析，对慢查询的sql进行进一步优化，或者结合学习过的memcache、nosql、分库、分表等措施进行处理或增加硬件。

慢查询实验

尽量不要使用like的%xx%操作，如果环境允许使用分词技术

如果可以使用between取代limit操作，尤其是在起始数较大时

- `Select * from user where uid limit 123456 ,30`
- 改为
- `Select * from user where uid between 123456 and 123486`

尽量不要使用函数

不要使用select * 这样也可减少内存使用

排序操作会消耗较多的 CPU 资源，所以尽量少排序

使用count(*)的效率要高于count(uid)

使用join代替子查询，效率要高得多

多表操作，关联的外键类型要与主表类型一至，可以使查找的效率更高。

合理化建议