# **11 怎么给字符串字段加索引？**

mysql> create table SUser(

ID bigint unsigned primary key,

email varchar(64),

...

)engine=innodb;

由于要使用邮箱登录，所有的业务代码中一定会出现类似于这样的语句

mysql> select f1, f2 from SUser where email='xxx';

MySql 支持前缀索引

比如，这两个在email字段上创建索引的语句

mysql> alter table SUser add index index1(email);

或

mysql> alter table SUser add index index2(email(6));

两种不同的定义的数据结构的区别是什么

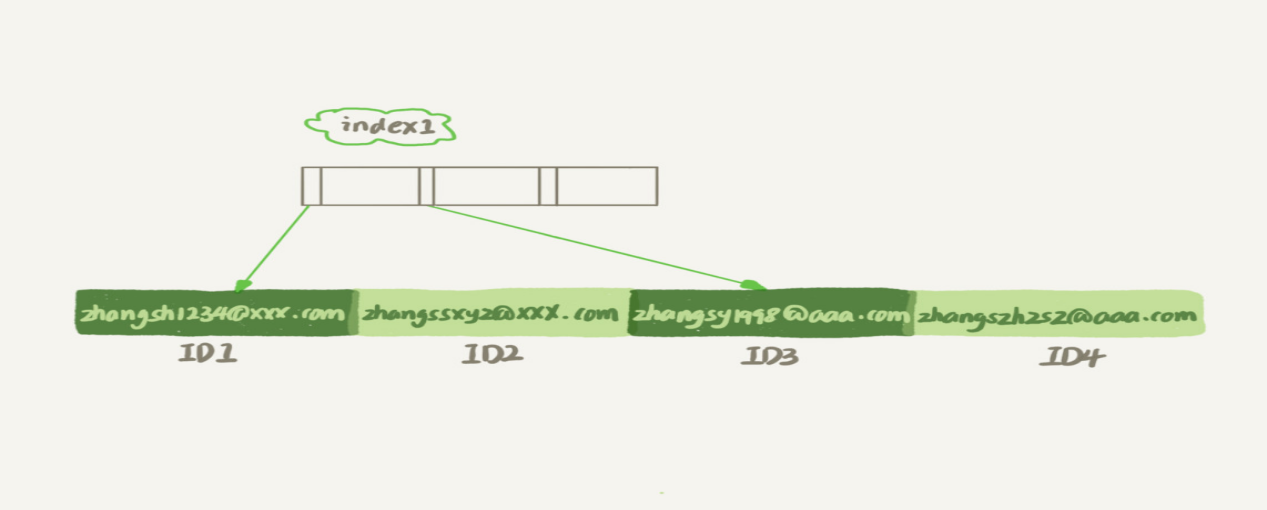


图1 email索引结构

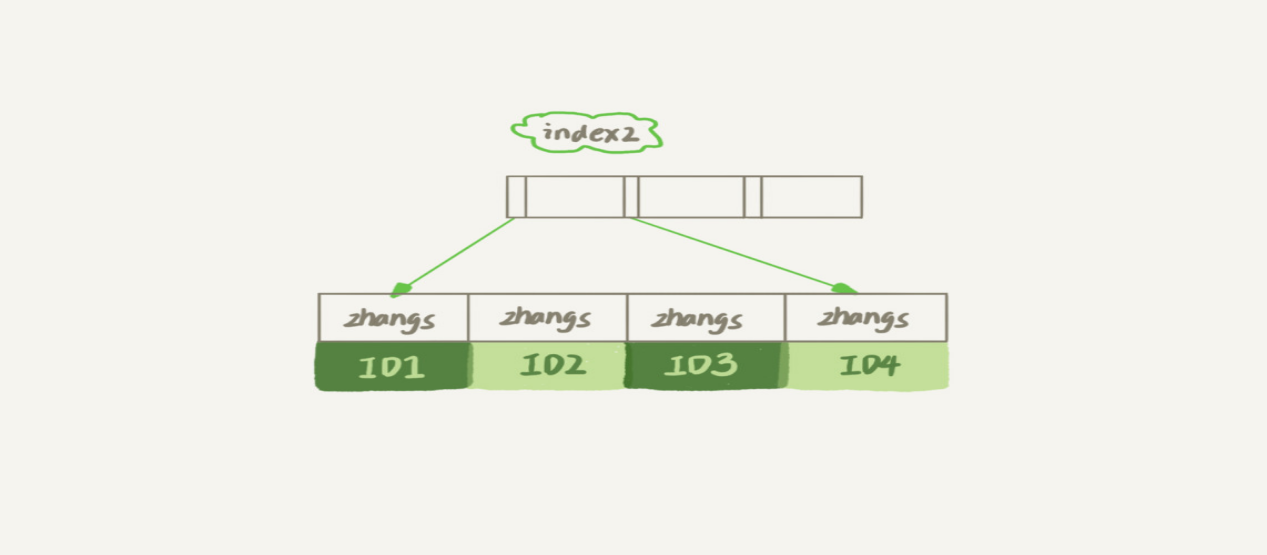


图 2 email(6)索引结构

从图中可以看出 email(6)这个索引结构中每个邮箱字段只取前6个字节（zhangs）所以占用的空间会更小，这就是前缀索引的优势。

但，这可能带来的损失是，可能会增加额外的记录扫描次数

看看下面这个语句，在这个两个索引定义下分别是怎么执行的

Select id ,name ,email from t where [email=’zhangssxyz@xxx.com’;](mailto:email=’zhangssxyz@xxx.com’;)

**如果使用index1(**即email整个字符串的索引结构**)，**执行顺序：

1.从index1索引树上找到满足索引值是‘zhangssxyz@xxx.com’的这条记录，取得ID2的值

2.到主键上查到主键值ID2的行，判断email的值是正确的，将这行记录加入结果集

3.取index1索引树上刚刚查找到的位置的下一行记录，发现已经不满足

[email=’zhangssxyz@xxx.com’的条件了，循环结束。](mailto:email=’zhangssxyz@xxx.com’的条件了，循环结束。)

这个过程，只需回主键索引取一次数据，所以系统认为只扫描了一行。

**如果使用的是index2(**即email(6)索引结构**)，**执行顺序

1. 从index2索引树找到满足索引值是‘zhangs’的记录，找到的第一个ID1;
2. [到主键索引上查到主键值是ID1的行，判断出email的值不是‘zhangssxyz@xxx.com’，这行记录丢弃](mailto:到主键索引上查到主键值是ID1的行，判断出email的值不是‘zhangssxyz@xxx.com’，这行记录丢弃)
3. 取index2上刚刚查到的位置的下一行记录，发现仍然是‘zhangs’,取出ID2,再到ID索引上取行然后判断，这次值对了，将这行记录加入结果集。
4. 重复上一步，直到index2上取到的值不是‘zhangs’时，循环结束。

这个过程回主键索引取4次数据，也就是扫描4行。

通过这个对比，使用前缀索引后，可能会导致查询语句读数据的此时变多。

但这个查询语句来说，若你定义的index(2)不是email(6)而是email(7),也就是说取email字段的前7个字节来构建索引的话，即满足前缀‘zhangs’的记录只有一个，也能够值查到ID2,只扫描一行

也即是**使用前缀索引，定义好长度，就可以做到既节省空间，有不用额外增加太多的查询成本。**

建立索引时关注的是区分度，区分度越高，键值越少，可以通过统计索引上有多少个不同的值来判断要使用多长的前缀。

使用下面的语句，算出来这个列上有多少个不同的值

mysql> select count(distinct email) as L from SUser

然后，依次选取不同长度的前缀来看这个值，比如4~7个字符的前缀索引，

mysql> select

count(distinct left(email,4)）as L4,

count(distinct left(email,5)）as L5,

count(distinct left(email,6)）as L6,

count(distinct left(email,7)）as L7,

from SUser;

当然，使用前缀索引很可能会损失区分度，所以需要预先设定一个可以接受的损失比例，如5%，然后，在返回的L4~L7中，找出不小于L\*95%的值，假设L6,L7都瞒足，可以选前缀长度为6.

**前缀索引对覆盖索引的影响**

前面前缀索引会增加扫描行数，这回影响性能，其实，不止这个。

select id,email from SUser where [email='zhangssxyz@xxx.com';](mailto:email='zhangssxyz@xxx.com';)

或

select id,name,email from SUser where email='zhangssxyz@xxx.com';

相比，这个语句只要求返回id和email字段。

所以，如果使用index1(即email整个字符串的索引结构)的话，可以利用覆盖索引，从index1查到结果后直接返回了，不需要回到ID索引再去查一次，而如果使用index2(即email(6)索引结构)的话，就不得不回到ID索引再去判断email字段的值。

即使使用index2的定义emial(18)的前缀索引，这时虽然index2已经包含了所有的信息，但InnoDB还是要回到id索引再查一次，因为系统并不确定前缀索引的定义是否截取完整信息。

**使用前缀索引就不用不上覆盖索引对查询的性能优化了，**这也是使用前缀索引考虑的一个因素

其他方式

**第一种是使用倒叙存储，**如果你存储身份证号的时候把它到过来，每次查询的时候，可以这么写

mysql> select field\_list from t where id\_card = reverse('input\_id\_card\_string');

由于身份证号的后6位没有地址这样的重复逻辑， 所以最后这6为很可能提供了足够的区分度，实际中不要忘记count(distinct)方法去验证。

**第二种使用hash字段，**在表上创建一个整数字段，来保存身份证号的校验码，同时在这个字段上建索引

mysql> alter table t add id\_card\_crc int unsigned, add index(id\_card\_crc)*;*

然后插入新的记录，都同时用crc32()这个函数得到校验码填到这个字段，由于校验码可能存在冲突，也就是说两个不同的身份证号通过crc32()函数得到的结果可能是相同的饿，所以查询语句where部分要判断id\_card的值是否精确相同

mysql> select field\_list from t where id\_card\_crc=crc32('input\_id\_card\_string') and id\_card='input\_id\_card\_string'

这样，索引长度变成了4个字节，比原来下了很多

**使用倒叙存储和使用hash字段两种方法的异同点。**

相同点是，不支持范围查询，倒序存储的字段上创建的索引时按照倒序字符串的方式排序的，hash字段的方式只能支持等值查询。

1.从占用空间来开，倒序存储方式在主键索引上，不会消耗额外的存储空

而hash字段方法需要增加一个字段，当然倒序存储使用4个字节的前缀长度应该是不够的，如果再长一点，这个消耗跟额外这个hash字段也差不多抵消了。

2.在CPU消耗方面，倒序方式每次写和读的时候，都需要额外调用一次reverse()函数，而hash字段的方式需要额外调用一次crc32()函数，如果只从这个两个函数计算复杂度来看的，reverse()函数额外消耗CPU资源会更小些

3.从查询效率上，使用hash字段方式的查询性能相对更稳定一些，因为crc32算出来的值虽然有冲突的概率，但是概率非常小，可以认为每次查询平均扫描行数接近1。而倒序存储方式毕竟还是用前缀索引的方式，也就是还是会增加扫描行数。

总结

1. 直接创建完整索引，这样可能比较占用空间
2. 创建前缀索引，节省空间，但会增加查询扫描次数，并且不能使用覆盖索引
3. 倒序存储，在创建前缀索引，用于绕过字符串本身前缀的区分度不够的问题
4. 创建hash字段索引，查询性能稳定，有额外的存储和计算消耗，跟第三中方式一样，都不支持范围扫描。

思考题

[维护学生信息。“学号@gmail.com”，学号规则十五位数字，前三位是城市编码，四到六位是学校编码，七到十位是入学年份，最后5位是顺序编码，](mailto:维护学生信息。\“学号@gmail.com\”，学号规则十五位数字，前三位是城市编码，四到六位是学校编码，七到十位是入学年份，最后5位是顺序编码，)

系统登录时，验证正确才能继续使用系统，只考虑登录验证这个行为，怎样设计登录名的索引？

[由于学号的规则，无论正向还是反向前缀索引，重复度都比较高，因为维护一个学校里面的，因此前面6位（其中，前面三位是所在城市编号，第四到六位数学校编码），其实是固定的饿，右边后缀都是@gmail.com.因此可以只存入学年份](mailto:由于学号的规则，无论正向还是反向前缀索引，重复度都比较高，因为维护一个学校里面的，因此前面6位（其中，前面三位是所在城市编号，第四到六位数学校编码），其实是固定的饿，右边后缀都是@gmail.com.因此可以只存入学年份) 加顺序编号，他们的长度是9位，而其实在此基础上，可以用数字类型来存这9位数字，比如201100001,这样只占4个字节，其实这个就是一中hash,只是用了最简单的转换规则：字符串转数字的规则，而刚好这个转化后的结果唯一性。