Like模糊查询如何优化?

典型回答

在MySQL中，使用like进行模糊查询，在一定情况下是无法使用索引的。如下所示：

- 当like值前后都有匹配符时`%abc%`，无法使用索引

```
EXPLAIN SELECT * FROM `test` WHERE `name` LIKE '%abc%' ;

+----+-------------+-------+------------+------+---------------+--------+---
------+--------+-------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key    |
key_len | ref    | rows  | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+--------+---
------+--------+-------+----------+-------------+
| 1  | SIMPLE      | test  | <null>     | ALL  | <null>        | <null> |
<null>  | <null> | 19820 | 11.11    | Using where |
+----+-------------+-------+------------+------+---------------+--------+---
------+--------+-------+----------+-------------+
```

- 当like值前有匹配符时`%abc`，无法使用索引

```
EXPLAIN SELECT * FROM `test` WHERE `name` LIKE '%abc' ;

+----+-------------+-------+------------+------+---------------+--------+---
------+--------+-------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key    |
key_len | ref    | rows  | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+--------+---
------+--------+-------+----------+-------------+
| 1  | SIMPLE      | test  | <null>     | ALL  | <null>        | <null> |
<null>  | <null> | 19820 | 11.11    | Using where |
+----+-------------+-------+------------+------+---------------+--------+---
------+--------+-------+----------+-------------+
```

- 当like值后有匹配符时'abc%'，可以使用索引

```
EXPLAIN SELECT * FROM `test` WHERE `name` LIKE 'abc%' ;

+----+-------------+-------+------------+-------+---------------+----------
+---------+--------+------+----------+-----------------------+
| id | select_type | table | partitions | type  | possible_keys | key      |
key_len | ref    | rows | filtered | Extra                 |
+----+-------------+-------+------------+-------+---------------+----------
+---------+--------+------+----------+-----------------------+
| 1  | SIMPLE      | test  | <null>     | range | idx_name      | idx_name |
153     | <null> | 200  | 100.0    | Using index condition |
+----+-------------+-------+------------+-------+---------------+----------
+---------+--------+------+----------+-----------------------+
```

索引类型为range，并使用索引下推（index condition）

扩展阅读

使用 `like %abc` 真的无法优化了吗?
我们之所以会使用 `%abc` 来查询说明表中的name可能包含以abc结尾的字符串，如果以 `abc%` 说明有以abc开头的字符串。
假设我们要向表中的name写入123abc，我们可以将这一列反转过来，即cba321插入到一个冗余列v_name中，并为这一列建立索引：

```
ALTER TABLE `test` ADD COLUMN `v_name` VARCHAR(50) NOT NULL DEFAULT ''; //为
test表新增v_name列
ALTER TABLE `test` ADD INDEX `idx_v_name`(`v_name`); //为v_name列添加索引
INSERT INTO `test`(`id`,`name`,`v_name`)VALUES(1,'123abc','cba321'); //这里不
但要写name，也要写v_name
```

接下来在查询的时候，我们就可以使用v_name列进行模糊查询了

```
SELECT * FROM `test` WHERE `v_name` LIKE 'cba%'; //相当于反向查询匹配出了
name=123abc的行
```

当然这样看起来有点麻烦，表中如果已经有了很多数据，还需要利用update语句反转name到v_name中，如果数据量大了（几百万或上千万条记录）更新一下v_name耗时也比较长，同时也会增大表空间。

```
UPDATE `test` SET `v_name` = REVERSE(`name`);
```

幸运的是在MySQL5.7.6之后，新增了虚拟列功能（如果不是>=5.7.6，只能用上面的土方法）。为一个列建立一个虚拟列，并为虚拟列建立索引，在查询时where中like条件改为虚拟列，就可以使用索引了。

```
ALTER TABLE `test` ADD COLUMN `v_name` VARCHAR(50) GENERATED ALWAYS AS
(REVERSE(`name`)) VIRTUAL; //创建虚拟列
ALTER TABLE `test` ADD INDEX `idx_name_virt`(`v_name`); //为虚拟列v_name列添加
索引
```

我们再进行查询，就会走索引了

```
EXPLAIN SELECT * FROM `test` WHERE `v_name` LIKE 'cba%';

+----+-------------+-------+-----------+-------+--------------+----------
----+---------+--------+------+----------+-------------+
| id | select_type | table | partitions | type  | possible_keys | key
| key_len | ref    | rows | filtered | Extra       |
+----+-------------+-------+-----------+-------+--------------+----------
----+---------+--------+------+----------+-------------+
| 1  | SIMPLE      | test  | <null>     | range | idx_name_virt |
idx_name_virt | 153     | <null> | 200  | 100.0    | Using where |
+----+-------------+-------+-----------+-------+--------------+----------
----+---------+--------+------+----------+-------------+
```

当然如果你要查询 `like 'abc%'` 和 `like '%abc'`，你只需要使用一个union

```
EXPLAIN SELECT * FROM `test` WHERE `v_name` LIKE 'cba%' //第一部分查询的是虚拟列
UNION SELECT * FROM `test` WHERE `name` LIKE 'abc%'; //第二部分查询的是原name列

+--------+-------------+-----------+-----------+-------+--------------+-
-------------+---------+--------+--------+----------+--------------------
-+
| id     | select_type  | table      | partitions | type  | possible_keys |
key           | key_len | ref    | rows   | filtered | Extra
|
+--------+-------------+-----------+-----------+-------+--------------+-
-------------+---------+--------+--------+----------+--------------------
-+
| 1      | PRIMARY      | test       | <null>     | range | idx_name_virt |
idx_name_virt | 153     | <null> | 200    | 100.0    | Using where
```

```
|
| 2      | UNION        | test       | <null>     | range | idx_name      |
idx_name      | 153     | <null> | 200    | 100.0    | Using index condition
|
| <null> | UNION RESULT | <union1,2> | <null>     | ALL   | <null>        |
<null>        | <null>  | <null> | <null> | <null>   | Using temporary
|
+--------+--------------+------------+------------+-------+---------------+-
-------------+---------+--------+--------+----------+--------------------
-+
```

可以看到，除了union result合并俩个语句，另外俩个查询都已经走索引了。如果你只想需要查询name，甚至可以使用覆盖索引进一步提升性能

```
EXPLAIN SELECT REVERSE(`v_name`) `test` WHERE `v_name` LIKE 'cba%' //第一部分
查询的是虚拟列,注意把v_name反转过来就拿到name的值了
UNION SELECT `name` FROM `test` WHERE `name` LIKE 'abc%'; //第二部分查询的是原
name列

+--------+--------------+------------+------------+-------+---------------+-
-------------+---------+--------+--------+----------+--------------------
----+
| id     | select_type  | table      | partitions | type  | possible_keys |
key           | key_len | ref    | rows   | filtered | Extra
|
+--------+--------------+------------+------------+-------+---------------+-
-------------+---------+--------+--------+----------+--------------------
----+
| 1      | PRIMARY      | test       | <null>     | range | idx_name_virt |
idx_name_virt | 153     | <null> | 200    | 100.0    | Using where; Using
index |
| 2      | UNION        | test       | <null>     | range | idx_name      |
idx_name      | 153     | <null> | 200    | 100.0    | Using where; Using
index |
| <null> | UNION RESULT | <union1,2> | <null>     | ALL   | <null>        |
<null>        | <null>  | <null> | <null> | <null>   | Using temporary
|
+--------+--------------+------------+------------+-------+---------------+-
-------------+---------+--------+--------+----------+--------------------
```

```
----+
```

虚拟列可以指定为VIRTUAL或STORED，VIRTUAL不会将虚拟列存储到磁盘中，在使用时MySQL会现计算虚拟列的值，STORED会存储到磁盘中，相当于我们手动创建的冗余列。所以：如果你的磁盘足够大，可以使用STORED方式，这样在查询时速度会更快一些。

如果你的数据量级较大，不使用反向查询的方式耗时会非常高。你可以使用如下sql测试虚拟列的效果：

```
//建表
CREATE TABLE test (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(50),
  INDEX idx_name (name)
) CHARACTER SET utf8;

//创建一个存储过程，向test表中写入2000000条数据，200条数据中abc字符前包含一些随机字符（用于测试like '%abc'的情况），200条数据中abc字符后包含一些随机字符（用于测试like 'abc%'的情况），其余行不包含abc字符
DELIMITER //

CREATE PROCEDURE InsertTestData()
BEGIN
  DECLARE i INT DEFAULT 1;

  WHILE i <= 2000000 DO
    IF i <= 200 THEN
      SET @randomPrefix1 = CONCAT(CHAR(FLOOR(RAND() * 26) + 65), CHAR(FLOOR(RAND() * 26) + 97), CHAR(FLOOR(RAND() * 26) + 48));
      SET @randomString1 = CONCAT(CHAR(FLOOR(RAND() * 26) + 65), CHAR(FLOOR(RAND() * 26) + 97), CHAR(FLOOR(RAND() * 26) + 48));
      SET @randomName1 = CONCAT(@randomPrefix1, @randomString1, 'abc');
      INSERT INTO test (name) VALUES (@randomName1);
    ELSEIF i <= 400 THEN
      SET @randomString2 = CONCAT(CHAR(FLOOR(RAND() * 26) + 65), CHAR(FLOOR(RAND() * 26) + 97), CHAR(FLOOR(RAND() * 26) + 48));
      SET @randomName2 = CONCAT('abc', @randomString2);
      INSERT INTO test (name) VALUES (@randomName2);
    ELSE
```

```
      SET @randomName3 = CONCAT(CHAR(FLOOR(RAND() * 26) + 65),
CHAR(FLOOR(RAND() * 26) + 97), CHAR(FLOOR(RAND() * 26) + 48));
      INSERT INTO test (name) VALUES (@randomName3);
    END IF;

    SET i = i + 1;
  END WHILE;
END //

DELIMITER ;

//调用存储过程，这里执行的会很慢
call InsertTestData();

//建立虚拟列
alter table test add column `v_name` varchar(50) generated always as
(reverse(name));
//为虚拟列创建索引
alter table test add index `idx_name_virt`(v_name);

//使用虚拟列模糊查询
select * from test where v_name like 'cba%'
union
select * from test where name like 'abc%'

//不使用虚拟列模糊查询
select * from test where name like 'abc%'
union
select * from test where name like '%abc'
```