MySQL 使用 like “%x“，索引一定会失效吗?

大家好，我是小林。

昨天发了一篇关于索引失效的文章：谁还没碰过索引失效呢

我在文末留了一个有点意思的思考题：

> 最后留一个很有意思的思考题给大家。
>
> - **题目1**：一个表有多个字段，其中 name 是索引字段，其他非索引，id 拥有自增主键索引。
> - **题目2**：一个表有2个字段，其中 name 是索引字段，id 拥有自增主键索引。
>
> 上面两张表，分别执行以下查询语句：
>
> - select * from s where name like "xxx"
> - select * from s where name like "xxx%"
> - select * from s where name like "%xxx"
> - select * from s where name like "%xxx%"
>
> 针对题目 1 和题目 2 的数据表，哪些触发索引查询，哪些没有？

这个思考题其实是出自于，我之前这篇文章「一条 SQL 语句引发的思考」中留言区一位读者朋友出的问题。

很多读者都在留言区说了自己的想法，也有不少读者私聊我答案到底是什么?

所以，我今晚就跟大家聊聊这个思考题。

题目一

题目一很简单，相信大家都能分析出答案，我昨天分享的索引失效文章里也提及过。

「题目 1」的数据库表如下，id 是主键索引，name 是二级索引，其他字段都是非索引字段。

id	name	age	address	phone
1	张某	26	北京市海淀区	1300000001
2	林某	18	深圳市南山区	1300000002
3	陈某	30	广州市海珠区	1300000003
4	周某	34	深圳市南山区	1300000004
5	曾某	25	上海市松江区	1300000005
6	黄某	28	深圳市宝安区	1300000006
7	谢某	38	北京市海淀区	1300000007
8	钟某	23	广州市海珠区	1300000008
9	吴某	28	上海市浦东新区	1300000009

这四条模糊匹配的查询语句，第一条和第二条都会走索引扫描，而且都是选择扫描二级索引（index_name），我贴个第二条查询语句的执行计划结果图：

```
1  EXPLAIN select * from t_user where name like "xxx%"
```

Message | Result 1 | Profile | Status

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	t_user	(NULL)	range	index_name	index_name	123	(NULL)	1	100.00	Using index condition

而第三和第四条会发生索引失效，执行计划的结果 type= ALL，代表了全表扫描。

```
1  EXPLAIN select * from t_user where name like "%xxx"
```

Message | Result 1 | Profile | Status

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	t_user	(NULL)	ALL	(NULL)	(NULL)	(NULL)	(NULL)	9	11.11	Using where

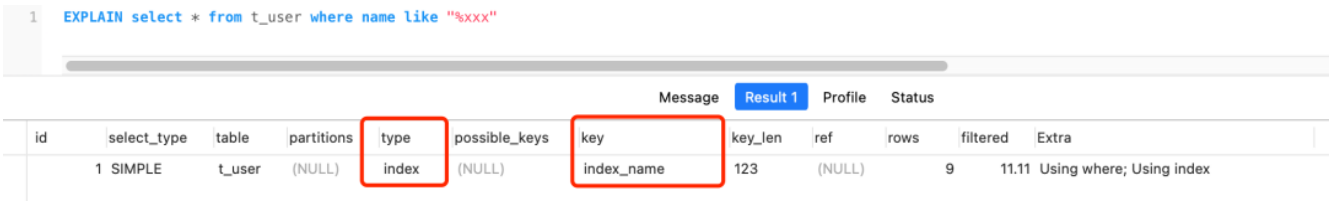
题目二

题目 2 的数据库表特别之处在于，只有两个字段，一个是主键索引 id，另外一个是二级索引 name。

id	name
1	张某
2	林某
3	陈某
4	周某
5	曾某
6	黄某
7	谢某
8	钟某
9	吴某

针对题目 2 的数据表，第一条和第二条模糊查询语句也是一样可以走索引扫描，第二条查询语句的执行计划如下，Extra 里的 Using index 说明用上了覆盖索引：

我们来看一下第三条查询语句的执行计划（第四条也是一样的结果）：



从执行计划的结果中，可以看到 key=index_name，也就是说用上了二级索引，而且从 Extra 里的 Using index 说明用上了覆盖索引。

这是为什么呢?

首先，这张表的字段没有「非索引」字段，所以 `select *` 相当于 `select id,name` ，然后**这个查询的数据都在二级索引的 B+ 树，因为二级索引的 B+ 树的叶子节点包含「索引值+主键**

值」，所以查二级索引的 B+ 树就能查到全部结果了，这个就是覆盖索引。

但是执行计划里的 type 是 `index` ，这代表着是通过全扫描二级索引的 B+ 树的方式查询到数据的，也就是遍历了整颗索引树。

而第一和第二条查询语句的执行计划中 type 是 `range` ，表示对索引列进行范围查询，也就是利用了索引树的有序性的特点，通过查询比较的方式，快速定位到了数据行。

所以，type=range 的查询效率会比 type=index 的高一些。

> 为什么选择全扫描二级索引树，而不扫描聚簇索引树呢?

因为二级索引树的记录东西很少，就只有「索引列+主键值」，而聚簇索引记录的东西会更多，比如聚簇索引中的叶子节点则记录了主键值、事务 id、用于事务和 MVCC 的回滚指针以及所有的剩余列。

再加上，这个 select * 不用执行回表操作。

所以， MySQL 优化器认为直接遍历二级索引树要比遍历聚簇索引树的成本要小的多，因此 MySQL 选择了「全扫描二级索引树」的方式查询数据。

> 为什么这个数据表加了非索引字段，执行同样的查询语句后，怎么变成走的是全表扫描呢?

加了其他字段后， `select * from t_user where name like "%xx";` 要查询的数据就不能只在二级索引树里找了，得需要回表操作才能完成查询的工作，再加上是左模糊匹配，无法利用索引树的有序性来快速定位数据，所以得在二级索引树逐一遍历，获取主键值后，再到聚簇索引树检索到对应的数据行，这样实在太累了。

所以，优化器认为上面这样的查询过程的成本实在太高了，所以直接选择全表扫描的方式来查询数据。

从这个思考题我们知道了，使用左模糊匹配（like "%xx"）并不一定会走全表扫描，关键还是看数据表中的字段。

如果数据库表中的字段只有主键+二级索引，那么即使使用了左模糊匹配，也不会走全表扫描（type=all），而是走全扫描二级索引树(type=index)。

再说一个相似，我们都知道联合索引要遵循最左匹配才能走索引，但是如果数据库表中的字段都是索引的话，即使查询过程中，没有遵循最左匹配原则，也是走全扫描二级索引树(type=index)，比如下图：

```
show CREATE TABLE t;

CREATE TABLE `t` (
  `a` int NOT NULL,
  `b` int NOT NULL,
  `c` int NOT NULL,
  `id` int NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`id`) USING BTREE,
  KEY `abc` (`a`,`b`,`c`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci

explain select * from t where c = 1;
```

Message | Result 1 | Profile | Status

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	t	(NULL)	index	abc	abc	12	(NULL)	3	33.33	Using where; Using index

就说到这了，下次见啦