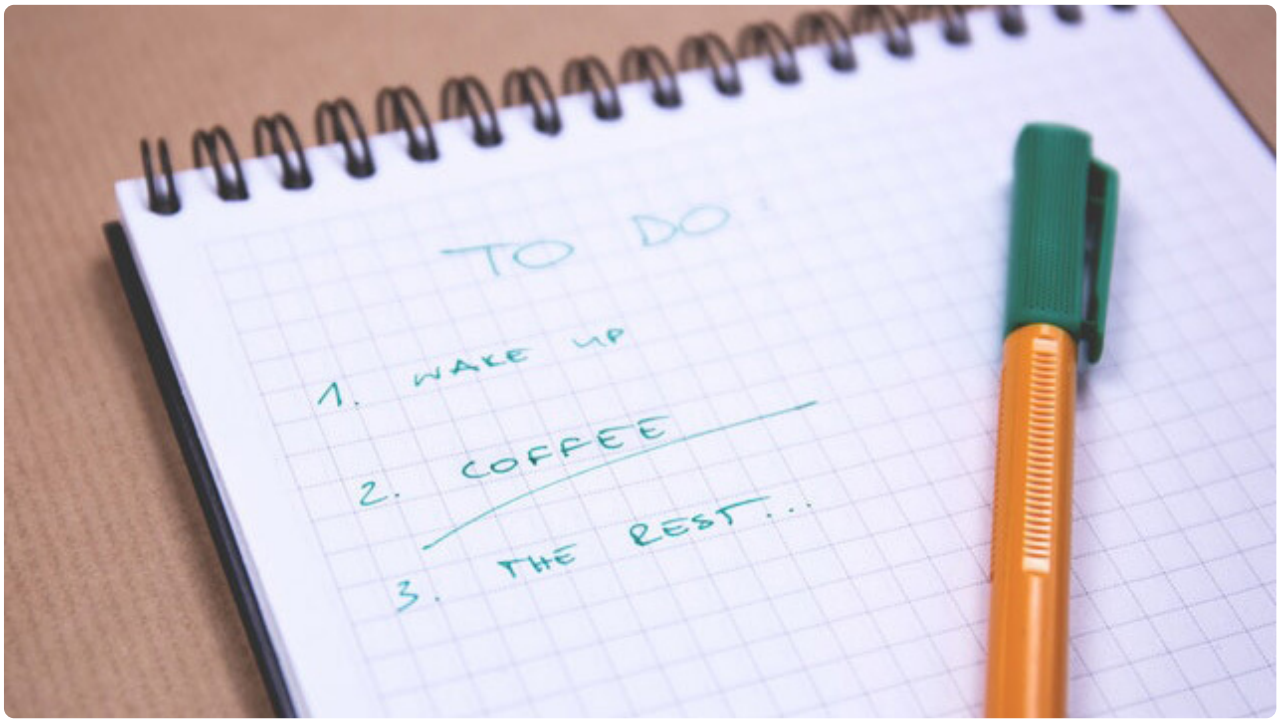
14 为什么MySQL会选错索引？

更新时间：2019-08-29 09:28:29



> 最聪明的人是最不愿浪费时间的人。
>
> ——但丁

在工作中，也许我们有时会遇到这种场景：某条 SQL 明明可以走 a 索引，却走了更慢的 b 索引。 今天我们就来讨论这种现象。

为了方便实验，首先创建测试表并写入测试数据，语句如下：

```
use muke;
drop table if exists t13;
CREATE TABLE `t13` (
  `a` int(11) NOT NULL,
  `b` int(11) NOT NULL,
  `create_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '记录创建时间',
  `update_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP COMMENT '记录更新时间',
  PRIMARY KEY (`a`,`b`),
  KEY `idx_a` (`a`),
  KEY `idx_create_time`(`create_time`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

drop procedure if exists insert_t13; /* 如果存在存储过程insert_t13，则删除 */
delimiter ;;
create procedure insert_t13() /* 创建存储过程insert_t13 */
begin
declare i int; /* 声明变量i */
set i=1; /* 设置i的初始值为1 */
while(i<=100000)do /* 对满足i<=100000的值进行while循环 */
insert into t13(a,b) values(i,i); /* 写入表t13中a字段，值为i当前的值 */
set i=i+1; /* 将i加1 */
end while;
end;;
delimiter ; /* 创建批量写入100000条数据到表t13的存储过程insert_t13 */
call insert_t13(); /* 运行存储过程insert_t13 */
```

在分析 MySQL 选错索引的情况之前，先讲 show index 的使用，因为后面会用到。

1、show index 的使用

当你需要查看某张表的索引详情时，可以使用命令：

```
show index from t13;
```

```
mysql> show index from t13;
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment |
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| t13   |          0 | PRIMARY  |            1 | id          | A         |       93232 |     NULL | NULL   |      | BTREE      |         |               |
| t13   |          1 | idx_a    |            1 | a           | A         |       96234 |     NULL | NULL   | YES  | BTREE      |         |               |
| t13   |          1 | idx_b    |            1 | b           | A         |           1 |     NULL | NULL   | YES  | BTREE      |         |               |
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
3 rows in set (0.00 sec)
```

对上面几个重要的字段做一下解释：

- Non_unique：如果是唯一索引，则值为 0，如果可以有重复值，则值为 1
- Key_name：索引名字
- Seq_in_index：索引中的列序号，比如联合索引 idx_a_b_c(a,b,c) ，那么三个字段分别对应 1,2,3
- Column_name：字段名
- Collation：字段在索引中的排序方式，A 表示升序，NULL 表示未排序
- Cardinality：索引中不重复记录数量的预估值，该值等会儿会详细讲解
- Sub_part：如果是前缀索引，则会显示索引字符的数量；如果是对整列进行索引，则该字段值为 NULL
- Null：如果列可能包含空值，则该字段为 YES；如果不包含空值，则该字段值为 ' '
- Index_type：索引类型，包括 BTREE、FULLTEXT、HASH、RTREE 等

show index 各字段的详细描述可以参考官方文档：https://dev.mysql.com/doc/refman/5.7/en/show-index.html。

根据上面的截图，可以看到，Cardinality 的值各字段都不相同，那么 Cardinality 的值是如何获取的呢？我们一起来讨论一下。

2、Cardinality 取值

Cardinality 表示该索引不重复记录数量的预估值。如果该值比较小，那就应该考虑是否还有必要创建这个索引。比如性别这种类型的字段，即使加了索引，Cardinality 值比较小，使用性别做条件查询数据时，可能根本用不到已经添加的索引（可以参考第 3 节的第 4 部分：范围查询）。

那么 Cardinality 值的统计频率是怎样的呢？

考虑到如果每次索引在发生操作时，都重新统计字段不重复记录数赋给 Cardinality，将会对数据库带来很大的负担。因此 Cardinality 不是每次操作都重新统计的，而是通过采样的方法来完成的。

Cardinality 统计信息的更新发生在两个操作中：INSERT 和 UPDATE。当然也不是每次 INSERT 或 UPDATE 就更新的，其更新时机为：

- 表中 1/16 的数据已经发生过变化
- 表中数据发生变化次数超过 2000000000

Cardinality 值是怎样统计和更新的呢？

InnoDB 表取出 B+ 树索引中叶子节点的数量，记为 a；随机取出 B+ 树索引中的 8 个（这个数量有参数 innodb_stats_transient_sample_pages 控制，默认为 8）叶子节点，统计每个页中不同记录的个数（假设为 b1，b2，b3，…，b8）。则 Cardinality 的预估值为：

（b1 + b2 + b3 + … b8）* a/8

所以 Cardinality 的值是对 8 个叶子节点进行采样获取的，显然这个值并不准确，只供参考。

下面我们来看下统计 Cardinality 涉及到的几个参数：

- innodb_stats_transient_sample_pages：设置统计 Cardinality 值时每次采样页的数量，默认值为 8。
- innodb_stats_method：用来判断如果对待索引中出现的 NULL 值记录，默认为 nulls_equal，表示将 NULL 值记录视为相等的记录。另外还有 nulls_unequal 和 nulls_ignored。nulls_unequal 表示将 NULL 视为不同的记录，nulls_ignored 表示忽略 NULL 值记录。
- innodb_stats_persistent：是否将 Cardinality 持久化到磁盘。好处是：比如数据库重启，不需要再计算 Cardinality 的值。
- innodb_stats_on_metadata：当通过命令 show table status、show index 及访问 information_chema 库下的 tables 表和 statistics 表时，是否需要重新计算索引的 Cardinality。目的是考虑有些表数据量大，并且辅助索引多时，执行这些操作可能会比较慢，而使用者可能并不需要更新 Cardinality。

3、统计信息不准确导致选错索引

在 MySQL 中，**优化器控制着索引的选择。一般情况下，优化器会考虑扫描行数、是否使用临时表、是否排序等因素，然后选择一个最优方案去执行 SQL 语句**。

而 MySQL 中扫描行数并不会每次执行语句都去计算一次，因为每次都去计算，数据库压力太大了。实际情况是通过统计信息来预估扫描行数。这个统计信息就可以看成 show index 中的 Cardinality。

而从上面说到 Cardinality 的更新原理可以看出，它的值不一定准确的，因此有时可能就是因为它的值不精准导致选错了索引。这种情况可以使用下面的命令重新统计信息：

```
analyze table t13;
```

4、单次选取的数据量过大导致选错索引

有时，我们也会遇到这种情况，如果单次选取的数据量过大，可能也会导致“选错”索引。

我们来看下面这个例子：

```
select a from t13 where a>80000 limit 1000;  /* sql1 */
```

sql1 的执行计划如下：

```
mysql> explain  select a from t13 where a>80000 limit 1000;
+----+-------------+-------+------------+-------+---------------+-------+---------+------+-------+----------+--------------------------+
| id | select_type | table | partitions | type  | possible_keys | key   | key_len | ref  | rows  | filtered | Extra                    |
+----+-------------+-------+------------+-------+---------------+-------+---------+------+-------+----------+--------------------------+
|  1 | SIMPLE      | t13   | NULL       | range | PRIMARY,idx_a | idx_a | 4       | NULL | 38342 |   100.00 | Using where; Using index |
+----+-------------+-------+------------+-------+---------------+-------+---------+------+-------+----------+--------------------------+
1 row in set, 1 warning (0.00 sec)
```

```
select a from t13 where a>70000 limit 1000;  /* sql2 */
```

sql2 的执行计划如下：

```
mysql> explain  select a from t13 where a>70000 limit 1000;
+----+-------------+-------+------------+-------+---------------+---------+---------+------+-------+----------+--------------------------+
| id | select_type | table | partitions | type  | possible_keys | key     | key_len | ref  | rows  | filtered | Extra                    |
+----+-------------+-------+------------+-------+---------------+---------+---------+------+-------+----------+--------------------------+
|  1 | SIMPLE      | t13   | NULL       | range | PRIMARY,idx_a | PRIMARY | 4       | NULL | 50103 |   100.00 | Using where; Using index |
+----+-------------+-------+------------+-------+---------------+---------+---------+------+-------+----------+--------------------------+
1 row in set, 1 warning (0.00 sec)
```

我们发现 sql1 满足条件的数据量相对较少时，是走的索引 idx_a；而在 sql2 中，满足条件的数据量相对比较多时，就走了主键索引（key 字段的值为 PRIMARY）。

我们在“哪些情况需要添加索引”一节中提到，InnoDB 二级索引树的叶子节点上存放的是主键，而主键索引树的叶子节点上存放的是整行数据。因此实际对于 sql2 来说，走普通索引 key_b 效率会高些。因此这条 sql 可以使用 force index 来强制走索引 key_b，sql 如下：

```
select a from t13 force index(idx_a) where a>70000 limit 1000;
```

```
mysql> explain select a from t13 force index(idx_a) where a>70000 limit 1000;
+----+-------------+-------+------------+-------+---------------+-------+---------+------+-------+----------+--------------------------+
| id | select_type | table | partitions | type  | possible_keys | key   | key_len | ref  | rows  | filtered | Extra                    |
+----+-------------+-------+------------+-------+---------------+-------+---------+------+-------+----------+--------------------------+
|  1 | SIMPLE      | t13   | NULL       | range | idx_a         | idx_a | 4       | NULL | 50128 |   100.00 | Using where; Using index |
+----+-------------+-------+------------+-------+---------------+-------+---------+------+-------+----------+--------------------------+
1 row in set, 1 warning (0.00 sec)
```

5、总结

本节介绍了 show index 的各个字段的含义，并重点说明了 Cardinality 的取值原理。

通过学习了 Cardinality 的取值原理，我们知道了它的值只是一个估值，因此当我们遇到它的值与实际值相差很大时，可以考虑使用：analyze table xxx; 重新获取统计信息。

另外举例说明了单次选取的数据量过大也有可能导致优化器选错索引，这种时候，可以尝试使用 force index 让 sql 强制走某个索引。

6、问题

在 sql2 中，优化器为什么会选择主键索引？

7、参考资料

《MySQL 内核：InnoDB 存储引擎》卷1：第 5.5 节 Cardinality 值 和第 5.6 节 B+ 树索引的使用。

《MySQL 5.7 参考手册》：13.7.5.22 SHOW INDEX Syntax

}

13 联合索引有哪些讲究?

15 全局锁和表锁什么场景会用到