11 哪些情况需要添加索引？

更新时间：2019-08-20 09:41:30

> 构成我们学习最大障碍的是已知的东西，而不是未知的东西。
>
> —— 贝尔纳

在上一篇文章中，我们知道了索引的重要性。但是，到底哪些情况需要添加索引呢？今天我们就来谈谈这个问题。

首先创建测试表并写入数据：

```
use muke; /* 使用muke这个database */
drop table if exists t9_1; /* 如果表t9_1存在则删除表t9_1 */
CREATE TABLE `t9_1` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`a` int(11) DEFAULT NULL,
`b` int(11) DEFAULT NULL,
`c` int(11) DEFAULT NULL,
`d` int(11) DEFAULT NULL,
PRIMARY KEY (`id`),
KEY `idx_a` (`a`),
KEY `idx_b_c` (`b`,`c`)
) ENGINE=InnoDB  CHARSET=utf8mb4;
drop procedure if exists insert_t9_1; /* 如果存在存储过程insert_t9_1，则删除 */
delimiter ;;
create procedure insert_t9_1() /* 创建存储过程insert_t9_1 */
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 t9_1(a,b,c,d) values(i,i,i,i); /* 写入表t9_1中a、b两个字段，值都为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t9_1的存储过程insert_t9_1 */
call insert_t9_1(); /* 运行存储过程insert_t9_1 */

insert into t9_1(a,b,c,d) select a,b,c,d from t9_1;
insert into t9_1(a,b,c,d) select a,b,c,d from t9_1;
insert into t9_1(a,b,c,d) select a,b,c,d from t9_1;
insert into t9_1(a,b,c,d) select a,b,c,d from t9_1;
insert into t9_1(a,b,c,d) select a,b,c,d from t9_1;
/*  把t9_1的数据量扩大到160万 */
```

目前比较常见需要创建索引的场景有：数据检索时在条件字段添加索引、聚合函数对聚合字段添加索引、对排序字段添加索引、为了防止回表添加索引、关联查询在关联字段添加索引等。我们就一一分析这些需要创建索引的场景：

1 数据检索

用上面的表 t9_1 做测试，首先把没有索引的字段 d 作为条件进行查询：

```
select * from t9_1 where d = 90000;
```

```
mysql> select * from t9_1 where d = 90000;
+---------+-------+-------+-------+-------+
| id      | a     | b     | c     | d     |
+---------+-------+-------+-------+-------+
|   90000 | 90000 | 90000 | 90000 | 90000 |
|  190000 | 90000 | 90000 | 90000 | 90000 |
|  321070 | 90000 | 90000 | 90000 | 90000 |
|  421070 | 90000 | 90000 | 90000 | 90000 |
|  583210 | 90000 | 90000 | 90000 | 90000 |
|  683210 | 90000 | 90000 | 90000 | 90000 |
|  783210 | 90000 | 90000 | 90000 | 90000 |
|  883210 | 90000 | 90000 | 90000 | 90000 |
| 1041955 | 90000 | 90000 | 90000 | 90000 |
| 1141955 | 90000 | 90000 | 90000 | 90000 |
| 1241955 | 90000 | 90000 | 90000 | 90000 |
| 1341955 | 90000 | 90000 | 90000 | 90000 |
| 1441955 | 90000 | 90000 | 90000 | 90000 |
| 1541955 | 90000 | 90000 | 90000 | 90000 |
| 1641955 | 90000 | 90000 | 90000 | 90000 |
| 1741955 | 90000 | 90000 | 90000 | 90000 |
+---------+-------+-------+-------+-------+
16 rows in set (0.44 sec)
```

发现查询时间需要0.44 秒

再把有索引的字段 a 作为条件进行查询

```
select * from t9_1 where a = 90000;
```

```
mysql> select * from t9_1 where a = 90000 ;
+---------+-------+-------+-------+-------+
| id      | a     | b     | c     | d     |
+---------+-------+-------+-------+-------+
|   90000 | 90000 | 90000 | 90000 | 90000 |
|  190000 | 90000 | 90000 | 90000 | 90000 |
|  321070 | 90000 | 90000 | 90000 | 90000 |
|  421070 | 90000 | 90000 | 90000 | 90000 |
|  583210 | 90000 | 90000 | 90000 | 90000 |
|  683210 | 90000 | 90000 | 90000 | 90000 |
|  783210 | 90000 | 90000 | 90000 | 90000 |
|  883210 | 90000 | 90000 | 90000 | 90000 |
| 1041955 | 90000 | 90000 | 90000 | 90000 |
| 1141955 | 90000 | 90000 | 90000 | 90000 |
| 1241955 | 90000 | 90000 | 90000 | 90000 |
| 1341955 | 90000 | 90000 | 90000 | 90000 |
| 1441955 | 90000 | 90000 | 90000 | 90000 |
| 1541955 | 90000 | 90000 | 90000 | 90000 |
| 1641955 | 90000 | 90000 | 90000 | 90000 |
| 1741955 | 90000 | 90000 | 90000 | 90000 |
+---------+-------+-------+-------+-------+
16 rows in set (0.00 sec)
```

发现查询时间为 0.00 sec，表示执行时间不超过 10 毫秒，非常快。

我们再对比两条 SQL 的执行计划：

```
mysql> explain select * from t9_1 where d = 90000 ;
+----+-------------+-------+------------+------+---------------+------+---------+------+---------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows    | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+------+---------+------+---------+----------+-------------+
|  1 | SIMPLE      | t9_1  | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 1596288 |    10.00 | Using where |
+----+-------------+-------+------------+------+---------------+------+---------+------+---------+----------+-------------+
1 row in set, 1 warning (0.00 sec)

mysql> explain select * from t9_1 where a = 90000 ;
+----+-------------+-------+------------+------+---------------+-------+---------+-------+------+----------+-------+
| id | select_type | table | partitions | type | possible_keys | key   | key_len | ref   | rows | filtered | Extra |
+----+-------------+-------+------------+------+---------------+-------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | t9_1  | NULL       | ref  | idx_a         | idx_a | 5       | const |   16 |   100.00 | NULL  |
+----+-------------+-------+------------+------+---------------+-------+---------+-------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)
```

执行计划中：

前者 type 字段为 ALL，后者 type 字段为 ref，显然后者性能更好（explain 的 type 字段解释如果忘记了，可以复习第 1 节 中 2.1.2 小节）

rows 这个字段前者是 1596288，而后者是 16，有索引的情况扫描行数大大降低。

因此建议数据检索时，在条件字段添加索引。

2 聚合函数

在测试表 t9_1 中，如果要求出无索引字段 d 的最大值，SQL 如下：

```
select max(d) from t9_1;
```

```
mysql> select max(d) from t9_1;
+--------+
| max(d) |
+--------+
| 100000 |
+--------+
1 row in set (0.33 sec)
```

执行时间为 0.33 秒。

再看下求有索引的字段 a 的最大值：

```
select max(a) from t9_1;
```

```
mysql> select max(a) from t9_1;
+--------+
| max(a) |
+--------+
| 100000 |
+--------+
1 row in set (0.00 sec)
```

执行时间为 0.00 秒，表示执行时间不超过 10 毫秒。

相比对没有索引的字段 d 求最大值（花费330毫秒），**显然索引能提升 max() 函数的效率，同理也能提升 min() 函数的效率**。

你是否有印象，我们在第 7 节中的 1.3 小节中有介绍 MySQL 5.7.18 之后版本的 count(*) 特点：从 MySQL 5.7.18 开始，通过遍历最小的可用二级索引来处理 count(*) 语句，如果不存在二级索引，则扫描聚簇索引。原因是：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，所以二级索引树比主键索引树小。因此优化器基于成本的考虑，优先选择的是二级索引。

因此索引对聚合函数 count(*) 也有优化作用。

3 排序

在第 4 节 2.1 小节，我们列出了几种通过添加合适索引优化 order by 的方法，这里再做一次总结（如果对下面的总结不是很理解，可以复习第 4 节的内容，有对每种情况举例说明）：

- 如果对单个字段排序，则可以在这个排序字段上添加索引来优化排序语句；
- 如果是多个字段排序，可以在多个排序字段上添加联合索引来优化排序语句；

- 如果是先等值查询再排序，可以通过在条件字段和排序字段添加联合索引来优化排序语句。

4 避免回表

比如下面这条 SQL：

```
select a,d from t9_1 where a=90000;
```

可以走 a 字段的索引，但是在学了第 8 节后，我们知道了辅助索引的结构，如果通过辅助索引来寻找数据，InnoDB 存储引擎会遍历辅助索引树查找到对应记录的主键，然后通过主键索引回表去找对应的行数据。

但是，如果条件字段和需要查询的字段有联合索引的话，其实回表这一步就省了，因为联合索引中包含了这两个字段的值。像这种索引就已经覆盖了我们的查询需求的场景，我们称为：覆盖索引。比如下面这条 SQL：

```
select b,c from t9_1 where b=90000;
```

可直接通过联合索引 idx_b_c 找到 b、c 的值（联合索引详细讲解将放在第 11 节）。

所以可以通过添加覆盖索引让 SQL 不需要回表，从而减少树的搜索次数，让查询更快地返回结果。

5 关联查询

在第 6 节中，我们讲到了关联查询的一些优化技巧，其中一个优化方式就是：通过在关联字段添加索引，让 BNL变成 NLJ 或者 BKA。这里就不继续举例说明了，如果没印象可以复习下第 6 节的内容。

6 总结

本节讲解了常见需要添加索引的场景：

- 数据检索时在条件字段添加索引
- 聚合函数对聚合字段添加索引
- 对排序字段添加索引
- 为了防止回表添加索引
- 关联查询在关联字段添加索引

后面如果工作中遇到类似场景，首先需要考虑是否需要添加索引，而不是被动地等出现慢查询甚至业务明显响应很慢才去添加索引。

7 问题

你觉得还有哪些场景需要添加索引？欢迎在留言中补充。

8 参考资料

《高性能 MySQL》（第三版）：第 5 章 创建高性能索引

}