13 联合索引有哪些讲究?

更新时间：2019-08-27 10:12:35

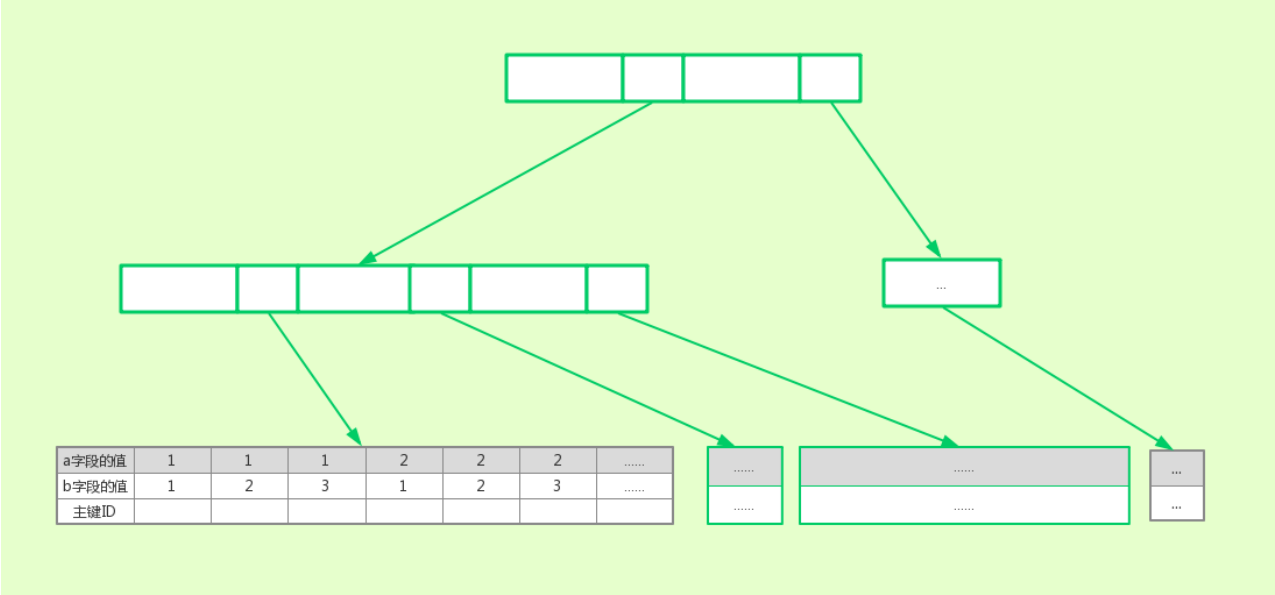
> 没有引发任何行动的思想都不是思想，而是梦想。
>
> —— 马丁

很多时候，单个字段的普通索引已经不能满足我们的需求了，可能需要多个字段的联合索引才能达到最优效果。那么联合索引究竟有哪些作用，又应该怎么去使用呢？本节就一起来探讨这些问题。

1 认识联合索引

联合索引：是指对表上的多个列进行索引。适合 where 条件中的多列组合，在某些场景可以避免回表。

我们拿讲解 order by 时使用的联合索引 B+ 树图进行理解，如下图：



联合索引的键值数量大于 1（比如上图中有 a 和 b 两个键值），与单个键值的 B+ 树一样，也是按照键值排序的。比如图中 a、b 两个字段的值为 (1,1),(1,2),(1,3),(2,1),(2,2),(2,3)，是按(a,b) 进行排序的。因此，**对于 a、b 两个字段都做为条件时，查询是可以走索引的；对于单独 a 字段查询也是可以走索引的。但是对于 b 字段单独查询就走不了索引了**，因为在上图，b 字段对应的值为 1,2,3,1,2,3，显然不是有序的，所以走不了 b 字段的索引。

联合索引的建议：

- where 条件中，经常同时出现的列放在联合索引中。
- 把选择性最大的列放在联合索引的最左边。

老规矩，创建测试表并写入数据：

```
use muke; /* 使用muke这个database */
drop table if exists t11; /* 如果表t11存在则删除表t11 */
CREATE TABLE `t11` ( /* 创建表t11 */
`id` int(11) NOT NULL AUTO_INCREMENT,
`a` int(20) DEFAULT NULL,
`b` int(20) DEFAULT NULL,
`c` int(20) DEFAULT NULL,
`d` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
PRIMARY KEY (`id`),
KEY `idx_a_b_c` (`a`,`b`,`c`)
) ENGINE=InnoDB CHARSET=utf8mb4 ;

insert into t11(a,b,c) values (1,1,1),(1,2,2),(1,2,1),(2,2,2),(2,1,2),(2,3,3),(2,4,4),(3,3,3),(4,4,4),(5,5,5),(6,6,6),(7,7,7),(8,8,8),(1,2,3),(1,2,4); /* 写入一些数据
*/
```

全表的数据如下：

```
select * from t11;
```

```
mysql> select * from t11;
+----+------+------+------+---------------------+
| id | a    | b    | c    | d                   |
+----+------+------+------+---------------------+
|  1 |    1 |    1 |    1 | 2019-07-18 13:43:47 |
|  2 |    1 |    2 |    2 | 2019-07-18 13:43:47 |
|  3 |    1 |    2 |    1 | 2019-07-18 13:43:47 |
|  4 |    2 |    2 |    2 | 2019-07-18 13:43:47 |
|  5 |    2 |    1 |    2 | 2019-07-18 13:43:47 |
|  6 |    2 |    3 |    3 | 2019-07-18 13:43:47 |
|  7 |    2 |    4 |    4 | 2019-07-18 13:43:47 |
|  8 |    3 |    3 |    3 | 2019-07-18 13:43:47 |
|  9 |    4 |    4 |    4 | 2019-07-18 13:43:47 |
| 10 |    5 |    5 |    5 | 2019-07-18 13:43:47 |
| 11 |    6 |    6 |    6 | 2019-07-18 13:43:47 |
| 12 |    7 |    7 |    7 | 2019-07-18 13:43:47 |
| 13 |    8 |    8 |    8 | 2019-07-18 13:43:47 |
| 14 |    1 |    2 |    3 | 2019-07-18 13:43:47 |
| 15 |    1 |    2 |    4 | 2019-07-18 13:43:47 |
+----+------+------+------+---------------------+
15 rows in set (0.00 sec)
```

2 联合索引使用分析

在上面我们知道了联合索引的原理，下面我们一起整理下联合索引的使用场景，理清这些，相信我们会更准确、高效的使用联合索引。

2 .1 可以完整用到联合索引的情况

下面我们列出几种可以完整用到联合索引的情况，并查看其执行计划，然后进行简短的分析：

```
select * from t11 where a=1 and b=1 and c=1; /* sql1 */
```

```
mysql> explain select * from t11 where a=1 and b=1 and c=1; /* sql1 */
+----+-------------+-------+------------+------+---------------+-----------+---------+-------------------+------+----------+-------+
| id | select_type | table | partitions | type | possible_keys | key       | key_len | ref               | rows | filtered | Extra |
+----+-------------+-------+------------+------+---------------+-----------+---------+-------------------+------+----------+-------+
|  1 | SIMPLE      | t11   | NULL       | ref  | idx_a_b_c     | idx_a_b_c | 15      | const,const,const |    1 |   100.00 | NULL  |
+----+-------------+-------+------------+------+---------------+-----------+---------+-------------------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)
```

> 这里补充一下 key_len 相关知识点：
>
> explain 中的 key_len 列用于表示这次查询中，所选择的索引长度有多少字节，常用于判断联合索引有多少列被选择了。下表总结了常用字段类型的 key_len：

列类型	KEY_LEN	备注
int	key_len = 4+1	int 为 4 bytes，允许为 NULL，加 1 byte
int not null	key_len = 4	不允许为 NULL
bigint	key_len=8+1	bigint 为 8 bytes，允许为 NULL 加 1 byte
bigint not null	key_len=8	bigint 为 8 bytes
char(30) utf8	key_len=30*3+1	char(n)为：n * 3 ，允许为 NULL 加 1 byte
char(30) not null utf8	key_len=30*3	不允许为 NULL
varchar(30) not null utf8	key_len=30*3+2	utf8 每个字符为 3 bytes，变长数据类型,加 2 bytes
varchar(30) utf8	key_len=30*3+2+1	utf8 每个字符为 3 bytes，允许为 NULL,加 1 byte,变长数据类型，加 2 bytes
datetime	key_len=8+1 (MySQL 5.6.4之前的版本)； key_len=5+1(MySQL 5.6.4及之后的版本)	允许为 NULL，加 1 byte

表一 常用字段类型的 key_len

因为 a、b、c 三个字段都是可以为 NULL 的 int 型。根据表1，可以知道三个字段的 key_len 都是 5，所以如果完整使用索引 idx_a_b_c，则 key_len 对应的值为 15。再回到上面 sql1 的执行计划中：key_len 显示是 15，而 key 列对应的是 idx_a_b_c，所以 sql1 完整用到了联合索引 idx_a_b_c。

```
select * from t11 where c=1 and b=1 and a=1;  /* sql2 */
```

```
mysql> explain select * from t11 where c=1 and b=1 and a=1; /* sql2 */
+----+-------------+-------+------------+------+---------------+-----------+---------+-------------------+------+----------+-------+
| id | select_type | table | partitions | type | possible_keys | key       | key_len | ref               | rows | filtered | Extra |
+----+-------------+-------+------------+------+---------------+-----------+---------+-------------------+------+----------+-------+
|  1 | SIMPLE      | t11   | NULL       | ref  | idx_a_b_c     | idx_a_b_c | 15      | const,const,const |    1 |   100.00 | NULL  |
+----+-------------+-------+------------+------+---------------+-----------+---------+-------------------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)
```

跟 sql1 的执行计划一样，因此联合索引各字段都做为条件时，各字段的位置不会影响联合索引的使用。

```
select * from t11 where a=2 and b in (1,2) and c=2;  /* sql3 */
```

```
mysql> explain select * from t11 where a=2 and b in (1,2) and c=2;  /* sql3 */
+----+-------------+-------+------------+-------+---------------+-----------+---------+------+------+----------+-----------------------+
| id | select_type | table | partitions | type  | possible_keys | key       | key_len | ref  | rows | filtered | Extra                 |
+----+-------------+-------+------------+-------+---------------+-----------+---------+------+------+----------+-----------------------+
|  1 | SIMPLE      | t11   | NULL       | range | idx_a_b_c     | idx_a_b_c | 15      | NULL |    2 |   100.00 | Using index condition |
+----+-------------+-------+------------+-------+---------------+-----------+---------+------+------+----------+-----------------------+
1 row in set, 1 warning (0.00 sec)
```

当联合索引前面的字段使用了范围查询，后面的字段做为条件时仍然可以使用完整的联合索引。

```
select * from t11 where a=1 and b=2 order by c;  /* sql4 */
```

```
mysql> explain select * from t11 where a=1 and b=2 order by c;  /* sql4 */
+----+-------------+-------+------------+------+---------------+-----------+---------+-------------+------+----------+-----------------------+
| id | select_type | table | partitions | type | possible_keys | key       | key_len | ref         | rows | filtered | Extra                 |
+----+-------------+-------+------------+------+---------------+-----------+---------+-------------+------+----------+-----------------------+
|  1 | SIMPLE      | t11   | NULL       | ref  | idx_a_b_c     | idx_a_b_c | 10      | const,const |    4 |   100.00 | Using index condition |
+----+-------------+-------+------------+------+---------------+-----------+---------+-------------+------+----------+-----------------------+
1 row in set, 1 warning (0.00 sec)
```

联合索引前面的字段做为条件时，对后面的字段做排序可以使用完整的联合索引。

```
select * from t11 where a=1 order by b,c;  /* sql5 */
```

```
mysql> explain select * from t11 where a=1 order by b,c;  /* sql5 */
+----+-------------+-------+------------+------+---------------+-----------+---------+-------+------+----------+-----------------------+
| id | select_type | table | partitions | type | possible_keys | key       | key_len | ref   | rows | filtered | Extra                 |
+----+-------------+-------+------------+------+---------------+-----------+---------+-------+------+----------+-----------------------+
|  1 | SIMPLE      | t11   | NULL       | ref  | idx_a_b_c     | idx_a_b_c | 5       | const |    5 |   100.00 | Using index condition |
+----+-------------+-------+------------+------+---------------+-----------+---------+-------+------+----------+-----------------------+
1 row in set, 1 warning (0.00 sec)
```

与 sql4 相似，对联合索引第一个字段做条件筛选时，对后面两个字段做排序可以使用完整的联合索引。

```
select a,b,c from t11 order by a,b,c;  /* sql6 */
```

```
mysql> explain select a,b,c from t11 order by a,b,c;  /* sql6 */
+----+-------------+-------+------------+-------+---------------+-----------+---------+------+------+----------+-------------+
| id | select_type | table | partitions | type  | possible_keys | key       | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-------+------------+-------+---------------+-----------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | t11   | NULL       | index | NULL          | idx_a_b_c | 15      | NULL |   15 |   100.00 | Using index |
+----+-------------+-------+------------+-------+---------------+-----------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
```

对联合索引的字段同时做排序时（但是排序的三个字段顺序要跟联合索引中三个字段的顺序一致），可以完整用到联合索引。

2.2 只能使用部分联合索引的情况

有些场景只能用到部分联合索引，这里就列出几种情况。

```
select * from t11 where a=1 and b=1; /* sql11 */
```

```
mysql> explain select * from t11 where a=1 and b=1; /* sql11 */
+----+-------------+-------+------------+------+---------------+-----------+---------+-------------+------+----------+-------+
| id | select_type | table | partitions | type | possible_keys | key       | key_len | ref         | rows | filtered | Extra |
+----+-------------+-------+------------+------+---------------+-----------+---------+-------------+------+----------+-------+
|  1 | SIMPLE      | t11   | NULL       | ref  | idx_a_b_c     | idx_a_b_c | 10      | const,const |    1 |   100.00 | NULL  |
+----+-------------+-------+------------+------+---------------+-----------+---------+-------------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)
```

当条件只包含联合索引的前面部分字段时，可以用到部分联合索引。

```
select * from t11 where a=1 and c=1; /* sql12 */
```

```
mysql> explain select * from t11 where a=1 and c=1; /* sql12 */
+----+-------------+-------+------------+------+---------------+-----------+---------+-------+------+----------+-----------------------+
| id | select_type | table | partitions | type | possible_keys | key       | key_len | ref   | rows | filtered | Extra                 |
+----+-------------+-------+------------+------+---------------+-----------+---------+-------+------+----------+-----------------------+
|  1 | SIMPLE      | t11   | NULL       | ref  | idx_a_b_c     | idx_a_b_c | 5       | const |    5 |    10.00 | Using index condition |
+----+-------------+-------+------------+------+---------------+-----------+---------+-------+------+----------+-----------------------+
1 row in set, 1 warning (0.00 sec)
```

对于联合索引 idx_a_b_c（a,b,c），如果条件中只包含 a 和 c，则只能用到联合索引中 a 的索引。c 这里是用不了索引的。**联合索引 idx_a_b_c(a,b,c) 相当于 (a)、(a,b)、(a,b,c) 三种索引，称为联合索引的最左原则。**

```
select * from t11 where a=2  and b in (3,4) order by c; /* sql13 */
```

```
mysql> explain select * from t11 where a=2  and b in (3,4) order by c; /* sql13 */
+----+-------------+-------+------------+-------+---------------+-----------+---------+------+------+----------+-----------------------------------------------+
| id | select_type | table | partitions | type  | possible_keys | key       | key_len | ref  | rows | filtered | Extra                                         |
+----+-------------+-------+------------+-------+---------------+-----------+---------+------+------+----------+-----------------------------------------------+
|  1 | SIMPLE      | t11   | NULL       | range | idx_a_b_c     | idx_a_b_c | 10      | NULL |    2 |   100.00 | Using index condition; Using filesort |
+----+-------------+-------+------------+-------+---------------+-----------+---------+------+------+----------+-----------------------------------------------+
1 row in set, 1 warning (0.00 sec)
```

这里可以复习第 4 节 2.4，**当联合索引前面的字段使用了范围查询，对后面的字段排序使用不了索引排序**，也就是只能用到联合索引前面两个字段 a 和 b 的索引。

2.3 可以用到覆盖索引的情况

什么是覆盖索引？

从辅助索引中就可以查询到结果，不需要回表查询聚集索引中的记录。

使用覆盖索引的优势：因为不需要扫描聚集索引，因此可以减少 SQL 执行过程的 IO 次数。

```
select b,c from t11 where a=3; /* sql21 */
```

```
mysql> explain select b,c from t11 where a=3; /* sql21 */
+----+-------------+-------+------------+------+---------------+-----------+---------+-------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key       | key_len | ref   | rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+-----------+---------+-------+------+----------+-------------+
|  1 | SIMPLE      | t11   | NULL       | ref  | idx_a_b_c     | idx_a_b_c | 5       | const |    1 |   100.00 | Using index |
+----+-------------+-------+------------+------+---------------+-----------+---------+-------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
```

通过 a 字段上的条件，去联合索引 idx_a_b_c 的索引树上可以直接查找到 b 字段和 c 字段的值，不需要回表，因此 sql21 使用到了覆盖索引。

```
select c from t11 where a=1 and b=1 ; /* sql22 */
```

```
mysql> explain select c from t11 where a=1 and b=1 ; /* sql22 */
+----+-------------+-------+------------+------+---------------+-----------+---------+-------------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key       | key_len | ref         | rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+-----------+---------+-------------+------+----------+-------------+
|  1 | SIMPLE      | t11   | NULL       | ref  | idx_a_b_c     | idx_a_b_c | 10      | const,const |    1 |   100.00 | Using index |
+----+-------------+-------+------------+------+---------------+-----------+---------+-------------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
```

跟 sql21 类似，在联合索引 idx_a_b_c 的索引树上，通过 a 和 b 的值可以直接找到 c 的值，因此 sql22 使用的也是覆盖索引。

```
select id from t11 where a=1 and b=1 and c=1; /* sql23 */
```

```
mysql> explain select id from t11 where a=1 and b=1 and c=1; /* sql23 */
+----+-------------+-------+------------+------+---------------+-----------+---------+-------------------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key       | key_len | ref               | rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+-----------+---------+-------------------+------+----------+-------------+
|  1 | SIMPLE      | t11   | NULL       | ref  | idx_a_b_c     | idx_a_b_c | 15      | const,const,const |    1 |   100.00 | Using index |
+----+-------------+-------+------------+------+---------------+-----------+---------+-------------------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
```

通过 a、b、c 三个字段的值，去联合索引树的叶子节点找到主键 id，不需要回表，因此 sql23 也使用了覆盖索引。

2.4 不能使用联合索引的情况

上面讲到的都是能用到联合索引的情况，下面再讲不能使用联合索引的情况。也请各位留意，避免使用不当导致无法利用联合索引。

```
select * from t11 where b=1; /* sql31 */
```

```
mysql> explain select * from t11 where b=1; /* sql31 */
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | t11   | NULL       | ALL  | NULL          | NULL | NULL    | NULL |   15 |    10.00 | Using where |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
```

如果只使用联合索引后面的字段做为条件查询，则使用不了联合索引（联合索引最左匹配）。

```
select * from t11 order by b; /* sql32 */
```

```
mysql> explain select * from t11 order by b; /* sql32 */
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+----------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra          |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+----------------+
|  1 | SIMPLE      | t11   | NULL       | ALL  | NULL          | NULL | NULL    | NULL |   15 |   100.00 | Using filesort |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+----------------+
1 row in set, 1 warning (0.00 sec)
```

与 sql31 相似，对联合索引后面的字段做排序操作，也使用不了联合索引。

```
select * from t11 where c=1; /* sql33 */
```

```
mysql> explain select * from t11 where c=1; /* sql33 */
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | t11   | NULL       | ALL  | NULL          | NULL | NULL    | NULL |   15 |    10.00 | Using where |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
```

与 sql31 类似。c 字段单独做条件使用不了索引。

```
select * from t11 where b=1 and c=1; /* sql34 */
```

```
mysql> explain select * from t11 where b=1 and c=1; /* sql34 */
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | t11   | NULL       | ALL  | NULL          | NULL | NULL    | NULL |   15 |     6.67 | Using where |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
```

联合索引中，如果第一个字段在条件中没有出现，那么联合索引的后面所有字段作为条件都无法使用这个联合索引。

3 总结

这节内容讲解了联合索引的实现原理，并举例说明了几种使用联合索引的情况：

- 可以完整用到联合索引的情况
- 只能使用部分联合索引的情况
- 可以用到覆盖索引的情况
- 不能使用联合索引的情况

可以结合自己的需求考虑是否通过创建联合索引代替单个字段的索引，在使用时也需要清楚联合索引的使用技巧，避免出现走不了索引的情况。

4 问题

在我们本节创建的测试表 t11 中，如果预计有大量下面的 SQL 会执行：

```
select c,d from t11 where c=1;
```

那应该怎么设计索引让查询效率最高？

5 参考资料

《MySQL 技术内幕》第 2 版 5.6.2 联合索引

}

12 普通索引和唯一索引有哪些区别？

14 为什么MySQL会选错索引？