33 加餐：答疑篇（二）

更新时间：2019-11-11 18:13:00

> 如果不想在世界上虚度一生，那就要学习一辈子。
>
> ——高尔基

在第 16 节后面，我们增加了答疑篇（一），最近在新的几篇包括之前的文章中，发现也有同学在留言区讨论了一些新问题，这里就选出几个，在本篇文章讨论一下。

1 第 17 节的问题

在第 17 节的课后问题，有几位同学参与了讨论，看到有些同学可能还不是很理解，这里来聊聊。

建表及初始化语句如下：

```
use muke;

drop table if exists t17_1;

CREATE TABLE `t17_1` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `a` int(11) NOT NULL,
  `b` int(11) NOT NULL,
  `c` int(11) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `uniq_a` (`a`) USING BTREE,
  KEY `idx_c` (`c`)
) ENGINE=InnoDB  CHARSET=utf8mb4;

insert into t17_1(id,a,b,c) values (1,1,1,1),(3,4,2,5),(5,3,4,3),(7,7,7,7);
```

```
select * from t17_1;
```

```
mysql> select * from t17_1;
+----+---+---+---+
| id | a | b | c |
+----+---+---+---+
|  1 | 1 | 1 | 1 |
|  3 | 4 | 2 | 5 |
|  5 | 3 | 4 | 3 |
|  7 | 7 | 7 | 7 |
+----+---+---+---+
4 rows in set (0.00 sec)
```

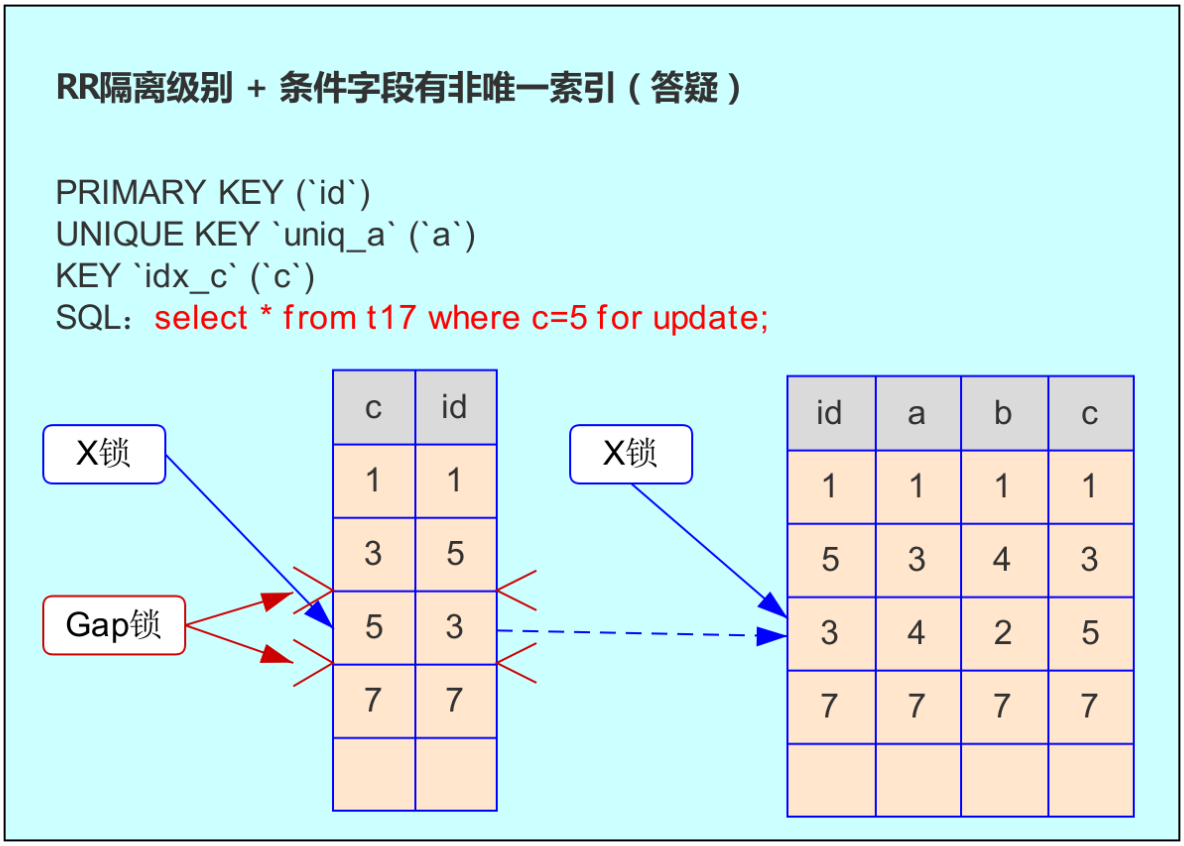
然后进行下面实验：

session1	session2	session3
set session transaction_isolation='REPEATABLE-READ';/* 设置会话隔离级别为 RR*/	set session transaction_isolation='REPEATABLE-READ';/* 设置会话隔离级别为 RR*/	set session transaction_isolation='REPEATABLE-READ';/* 设置会话隔离级别为 RR*/
begin;		
use muke; select * from t17_1 where c=5 for update; … 1 row in set (0.00 sec)		
	insert into t17_1 values (2,2,2,2);/* sql1 */	insert into t17_1 values (6,6,6,6);/* sql2 */
commit;		

在 session1 提交之前，sql1 和 sql2 是否能写入成功？原因是什么？

相信很多做过实验的同学都发现了，sql1 能写入成功，而 sql2 却会等待。什么原因呢？

我们来看下图：



结合第 17 节中:“RR 隔离级别下的非唯一索引查询” 的分析，我们知道：

[3,5] （分表代表 c 和 id 的值）前面的记录都不可能有间隙锁；

[3,5] 与 [5,3] 之间会有间隙锁；

[5,3] 与 [7,7] 之间会有间隙锁。

所以 ：

```
insert into t17_1 values (2,2,2,2);/* sql1 */
```

不需要等待。

```
insert into t17_1 values (6,6,6,6);/* sql2 */
```

会因为间隙锁而等待。

2 order by 使用了索引，在 key_len 中能体现出来吗？

这个问题是在第 13 节留言区有同学提出来的。

首先将建表语句复制过来：

```
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
*/
```

语句为：

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

执行计划的结果如下：

```
mysql> explain select * from t11 where a=1 and b = 2 order by c;
+----+-------------+-------+------------+------+---------------+-----------+---------+-------------+------+----------+-----------------------+
| 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)
```

可以看到实际 key_len 为 10，但是确实使用到联合索引 a、b、c 三个字段。原因是：key_len 只表示 where 中用户条件过滤时被选中的索引列，不包含 order by、group by 这部分被选中的索引。

3 GTID 是什么？

GTID（global transaction id）是对于一个已提交事务的编号，并且是一个全局唯一的编号。同一个事务在同一节点中出现多次的情况也不会再重现。GTID 实际上是由 UUID+TID 组成的。

SERVER_UUID 实际是一个 32 字节 +1 字节（/0）的字符串，SERVER_UUID 跟下面部分有关：

- MySQL 启动时间；
- 线程 LWP 有关；
- 一个随机的内存地址有关。

TID 代表了该实例上已经提交的事务数量，并且随着事务提交单调递增。

> GTID 详细信息可以参考官方文档： https://dev.mysql.com/doc/refman/5.7/en/replication-gtids.html 。

4 MyCAT 是否能实现分布式事务？

MyCAT 可以实现分布式事务（XA），但是可以理解为弱 XA 的事务，比如在提交阶段，若某个节点宕机，没有手段让此事务在故障节点恢复后继续执行。

并且 MyCAT 手册上也说明：不建议一个事务中存在跨多个节点的 SQL 操作，这样锁定的资源更多，并发性会降低很多。

5 总结

本节主要总结了关于专栏的一些新问题。当然，如果各位对于专栏任何一篇文章，或者其它 MySQL 相关知识点存在疑问，欢迎各位在本节底部的留言区讨论。

}

32 MySQL操作规范