MySQL DELETE 删除语句加锁分析

Posted on 2017-09-24 by Harvey MySQL

1. 前言

在MySQL的使用过程中，对SQL加锁的类型经常感到疑惑，这让死锁分析也变得举步维艰。因此需要将MySQL的各种SQL在各个隔离级别下加的锁进行分析，以免再次分析的时候还感到疑惑，也方便用于查询。

本次分析对SQL的删除语句进行分析，主要从以下几种情况进行分析：

1. 非唯一索引删除一条存在的记录
2. 唯一索引删除一条存在的记录
3. 主键删除一条存在的记录
4. 非唯一索引删除一条不存在记录
5. 唯一索引删除一条不存在的记录
6. 主键删除一条不存在的记录
7. 不同的SQL根据主键删除2条记录
8. 非唯一索引删除一条已经标记删除的记录
9. 唯一索引删除一条已经标记删除的记录

在使用之前需要打开innodb lock monitor，这样在查看 ***engine innodb status*** *的时候可以更加清晰的查到到锁的情况*

```
set GLOBAL innodb_status_output_locks=ON;
```

2. SQL的加锁分析

相关表结构

- 普通索引表结构

```
CREATE TABLE `t` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `c1` int(11) NOT NULL DEFAULT '0',
  `c2` int(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (`id`),
  KEY `idx_c1` (`c1`)
) ENGINE=InnoDB AUTO_INCREMENT=15 DEFAULT CHARSET=utf8mb4;
```

- 唯一索引表结构

```
CREATE TABLE `tu` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `c1` int(11) NOT NULL DEFAULT '0',
  `c2` int(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (`id`),
  UNIQUE KEY `uniq_c1` (`c1`)
) ENGINE=InnoDB AUTO_INCREMENT=15 DEFAULT CHARSET=utf8mb4
```

- 表的记录，唯一索引和普通索引的表结构均一样
- 测试的事务隔离级别为RR。

```
+----+----+----+
| id | c1 | c2 |
+----+----+----+
|  2 |  3 |  2 |
|  3 |  5 |  3 |
|  4 |  8 |  4 |
|  5 | 11 |  5 |
|  9 |  9 | 20 |
| 10 |  7 | 10 |
| 11 | 20 | 15 |
| 12 | 30 | 17 |
| 13 | 25 | 16 |
| 14 | 27 | 10 |
+----+----+----+
```

2.1 删除SQL加锁分析

根据非唯一索引删除一条存在记录

```
delete from t where c1=5;
Query OK, 1 rows affected (0.00 sec)
```

```
---TRANSACTION 146749, ACTIVE 9 sec
```

```
4 lock struct(s), heap size 1184, 3 row lock(s), undo log entries 1
MySQL thread id 1, OS thread handle 0x7f61ab1c7700, query id 104 localhost
TABLE LOCK table `test`.`t` trx id 146749 lock mode IX
RECORD LOCKS space id 53 page no 5 n bits 72 index `idx_c1` of table `test`
RECORD LOCKS space id 53 page no 3 n bits 72 index `PRIMARY` of table `test
RECORD LOCKS space id 53 page no 5 n bits 72 index `idx_c1` of table `test`
```

根据非唯一索引进行删除的时候，锁情况为：

1. 4 lock struct(s)：4种锁结构，分别为IX，idx_c1和主键的行锁，还有idx_c1的gap锁
2. 3 row lock(s)：有3个行锁，除去IX的都是算在row lock里面

根据唯一索引删除一条存在记录

```
delete from tu where c1=5;
Query OK, 1 rows affected (0.00 sec)
```

```
---TRANSACTION 146751, ACTIVE 2 sec
3 lock struct(s), heap size 360, 2 row lock(s), undo log entries 1
MySQL thread id 1, OS thread handle 0x7f61ab1c7700, query id 134 localhost
TABLE LOCK table `test`.`tu` trx id 146751 lock mode IX
RECORD LOCKS space id 45 page no 5 n bits 72 index `uniq_c1` of table `test
RECORD LOCKS space id 45 page no 3 n bits 80 index `PRIMARY` of table `test
```

根据唯一索引进行删除的时候，锁情况为：

1. 3 lock struct(s)：3种锁结构，分别为IX，idx_c1和主键的行锁，没有gap锁
2. 2 row lock(s)：有2个行锁，除去IX的都是算在row lock里面，没有gap，因此为2个

根据主键删除一条存在记录

```
delete from tu where id=2;
Query OK, 1 rows affected (0.00 sec)
```

```
---TRANSACTION 146753, ACTIVE 2 sec
```

```
2 lock struct(s), heap size 360, 1 row lock(s), undo log entries 1
MySQL thread id 1, OS thread handle 0x7f61ab1c7700, query id 147 localhost
TABLE LOCK table `test`.`tu` trx id 146753 lock mode IX
RECORD LOCKS space id 45 page no 3 n bits 80 index `PRIMARY` of table `test
```

根据主键进行删除的时候，锁情况为：

1. 2 lock struct(s)：2种锁结构，分别为IX和主键的行锁，没有gap锁
2. 1 row lock(s)：有1个行锁，就主键记录上的行锁，没有gap，因此为1个

根据非唯一索引删除一条*不存在*记录

```
delete from t where c1 = 4;
Query OK, 0 rows affected (0.00 sec)
```

```
---TRANSACTION 146786, ACTIVE 1 sec
2 lock struct(s), heap size 360, 1 row lock(s)
MySQL thread id 1, OS thread handle 0x7f61ab1c7700, query id 671 localhost
TABLE LOCK table `test`.`t` trx id 146786 lock mode IX
RECORD LOCKS space id 53 page no 5 n bits 80 index `idx_c1` of table `test`
```

根据非唯一索引删除一条 *不存在* 记录，锁情况为：

1. 2 lock struct(s)：2种锁结构，分别为IX和X类型的gap锁
2. 1 row lock(s)：有1个行锁，为非唯一索引的gap锁

根据唯一索引删除一条*不存在* 记录

```
delete from tu where c1 = 4;
Query OK, 0 rows affected (0.00 sec)
```

```
---TRANSACTION 146787, ACTIVE 2 sec
2 lock struct(s), heap size 360, 1 row lock(s)
MySQL thread id 1, OS thread handle 0x7f61ab1c7700, query id 711 localhost
TABLE LOCK table `test`.`tu` trx id 146787 lock mode IX
RECORD LOCKS space id 45 page no 5 n bits 72 index `uniq_c1` of table `test
```

根据唯一索引删除一条 *不存在* 记录,发现和非唯一索引一样，锁情况为：

1. 2 lock struct(s)：2种锁结构，分别为IX和X类型的gap锁
2. 1 row lock(s)：有1个行锁，为唯一索引的gap锁

根据主键删除一条*不存在* 记录

```
delete from tu where id = 6;
Query OK, 0 rows affected (0.00 sec)


---TRANSACTION 146831, ACTIVE 24 sec
2 lock struct(s), heap size 360, 1 row lock(s)
MySQL thread id 1, OS thread handle 0x7f61ab1c7700, query id 881 localhost
TABLE LOCK table `test`.`tu` trx id 146831 lock mode IX
RECORD LOCKS space id 45 page no 3 n bits 80 index `PRIMARY` of table `test
```

根据主键删除一条 *不存在* 记录,发现和非唯一索引一样，锁情况为：

1. 2 lock struct(s)：2种锁结构，分别为IX和X类型的gap锁
2. 1 row lock(s)：有1个行锁，为主键上的gap锁

根据主键删除两条存在的记录

```
有 5 ， 10 这两条记录

delete from tu where id>=5 and id<10;
Query OK, 2 rows affected (0.00 sec)

---TRANSACTION 146900, ACTIVE 35 sec
3 lock struct(s), heap size 360, 3 row lock(s), undo log entries 2
MySQL thread id 1, OS thread handle 0x7f61ab1c7700, query id 995 localhost
TABLE LOCK table `test`.`tu` trx id 146900 lock mode IX
RECORD LOCKS space id 56 page no 3 n bits 80 index `PRIMARY` of table `test
RECORD LOCKS space id 56 page no 3 n bits 80 index `PRIMARY` of table `test

有 5 ， 9 这两条记录
```

```
delete from tu where id>=5 and id<=9;
Query OK, 2 rows affected (0.00 sec)

---TRANSACTION 146912, ACTIVE 12 sec
3 lock struct(s), heap size 360, 3 row lock(s), undo log entries 2
MySQL thread id 1, OS thread handle 0x7f61ab1c7700, query id 1022 localhost
TABLE LOCK table `test`.`tu` trx id 146912 lock mode IX
RECORD LOCKS space id 56 page no 3 n bits 80 index `PRIMARY` of table `test
RECORD LOCKS space id 56 page no 3 n bits 80 index `PRIMARY` of table `test
```

有 4 , 10 这两条记录

```
delete from tu where id>4 and id<10;
Query OK, 2 rows affected (0.00 sec)

---TRANSACTION 146906, ACTIVE 13 sec
2 lock struct(s), heap size 360, 3 row lock(s), undo log entries 2
MySQL thread id 1, OS thread handle 0x7f61ab1c7700, query id 1011 localhost
TABLE LOCK table `test`.`tu` trx id 146906 lock mode IX
RECORD LOCKS space id 56 page no 3 n bits 80 index `PRIMARY` of table `test
```

有 10 没 7

```
delete from tu where id>=7 and id<=10;
Query OK, 2 rows affected (0.00 sec)

---TRANSACTION 146966, ACTIVE 2 sec
2 lock struct(s), heap size 360, 3 row lock(s), undo log entries 2
MySQL thread id 1, OS thread handle 0x7f61ab1c7700, query id 1172 localhost
TABLE LOCK table `test`.`tu` trx id 146966 lock mode IX
RECORD LOCKS space id 57 page no 3 n bits 80 index `PRIMARY` of table `test
```

有 4没 8

```
delete from tu where id>=4 and id<=8;
Query OK, 2 rows affected (0.00 sec)

---TRANSACTION 146972, ACTIVE 20 sec
3 lock struct(s), heap size 360, 3 row lock(s), undo log entries 2
MySQL thread id 1, OS thread handle 0x7f61ab1c7700, query id 1201 localhost
TABLE LOCK table `test`.`tu` trx id 146972 lock mode IX
RECORD LOCKS space id 57 page no 3 n bits 80 index `PRIMARY` of table `test
RECORD LOCKS space id 57 page no 3 n bits 80 index `PRIMARY` of table `test
```

有3，4两条记录

```
delete from tu where id in (3,4);
Query OK, 2 rows affected (0.00 sec)

---TRANSACTION 146880, ACTIVE 1 sec
2 lock struct(s), heap size 360, 2 row lock(s), undo log entries 2
MySQL thread id 1, OS thread handle 0x7f61ab1c7700, query id 928 localhost
TABLE LOCK table `test`.`tu` trx id 146880 lock mode IX
RECORD LOCKS space id 56 page no 3 n bits 80 index `PRIMARY` of table `test
```

根据主键删除两条的时候，使用in的锁情况为：

1. 2 lock struct(s)：2种锁结构，分别为IX和i主键的行锁，没有gap锁

2. 2 row lock(s)：有2个行锁，就主键记录上的行锁，没有gap，因此为2个

根据主键删除两条的时候，使用>,<,>=,<=,比较符号的锁情况为：

1. 无论如何，匹配到2条记录，因此必须会有2 row lock(s)
2. 如果只有>,<，那么毫无疑问，是不会锁定两个边界的记录，因此他只会锁定边界到边界内的整个范围，锁的类型为X，此时为2 lock struct(s)， 3 row lock(s)
3. 碰到 >= 的时候，判断 >= 的值是否存在，如果存在，则锁定该记录。所以除了IX,X锁，还有行锁，因此存在的时候为3 lock struct(s), 3 row lock(s)。如果不存在，和第二种是一样的，为2 lock struct(s) ， 3 row lock(s) 。

非唯一索引删除一条已经标记删除的记录

Sess1	Sess2	Sess3
begin;		
delete from t where c1=8;		
	begin;	
	delete from t where c1=8;	
		@1 show engine innodb status

commit;		
		@2 show engine innodb status

```
@1 show engine innodb status


---TRANSACTION 146981, ACTIVE 12 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 360, 1 row lock(s)
MySQL thread id 363, OS thread handle 0x7f61ab1c7700, query id 2804 localho
delete from t where c1=8
------- TRX HAS BEEN WAITING 12 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 54 page no 4 n bits 80 index `idx_c1` of table `test`
------------------
TABLE LOCK table `test`.`t` trx id 146981 lock mode IX
RECORD LOCKS space id 54 page no 4 n bits 80 index `idx_c1` of table `test`
---TRANSACTION 146980, ACTIVE 16 sec
4 lock struct(s), heap size 1184, 3 row lock(s), undo log entries 1
MySQL thread id 355, OS thread handle 0x7f61ab145700, query id 2802 localho
TABLE LOCK table `test`.`t` trx id 146980 lock mode IX
RECORD LOCKS space id 54 page no 4 n bits 80 index `idx_c1` of table `test`
RECORD LOCKS space id 54 page no 3 n bits 80 index `PRIMARY` of table `test
RECORD LOCKS space id 54 page no 4 n bits 80 index `idx_c1` of table `test`

@2 show engine innodb status

---TRANSACTION 146981, ACTIVE 50 sec
3 lock struct(s), heap size 360, 1 row lock(s)
MySQL thread id 363, OS thread handle 0x7f61ab1c7700, query id 2804 localho
TABLE LOCK table `test`.`t` trx id 146981 lock mode IX
RECORD LOCKS space id 54 page no 4 n bits 80 index `idx_c1` of table `test`
RECORD LOCKS space id 54 page no 4 n bits 80 index `idx_c1` of table `test`
```

非唯一索引删除一条已经标记删除的记录的锁情况为：

- 加锁等待时: 2 lock struct(s)，持有IX锁，等待记录上的X锁
- 加锁成功时：3 lock struct(s)，持有IX,行锁，和gap锁，这个和非唯一索引删除一条不存在的记录是基本一样的，多了个因Sess1 提交成功后多获得的行锁。

唯一索引删除一条已经标记删除的记录

Sess1	Sess2	Sess3
begin;		
delete from tu where c1=8;		
	begin;	
	delete from tu where c1=8;	
		@1 show engine innodb status
commit;		
		@2 show engine innodb status

```
@1 show engine innodb status

---TRANSACTION 146984, ACTIVE 2 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 360, 1 row lock(s)
MySQL thread id 363, OS thread handle 0x7f61ab1c7700, query id 2842 localho
delete from tu where c1=8
------- TRX HAS BEEN WAITING 2 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 57 page no 4 n bits 80 index `uniq_c1` of table `test
------------------
TABLE LOCK table `test`.`tu` trx id 146984 lock mode IX
RECORD LOCKS space id 57 page no 4 n bits 80 index `uniq_c1` of table `test
---TRANSACTION 146983, ACTIVE 9 sec
3 lock struct(s), heap size 360, 2 row lock(s), undo log entries 1
MySQL thread id 355, OS thread handle 0x7f61ab145700, query id 2839 localho
TABLE LOCK table `test`.`tu` trx id 146983 lock mode IX
RECORD LOCKS space id 57 page no 4 n bits 80 index `uniq_c1` of table `test
RECORD LOCKS space id 57 page no 3 n bits 80 index `PRIMARY` of table `test

@2 show engine innodb status

---TRANSACTION 146984, ACTIVE 23 sec
3 lock struct(s), heap size 360, 1 row lock(s)
MySQL thread id 363, OS thread handle 0x7f61ab1c7700, query id 2842 localho
TABLE LOCK table `test`.`tu` trx id 146984 lock mode IX
RECORD LOCKS space id 57 page no 4 n bits 80 index `uniq_c1` of table `test
RECORD LOCKS space id 57 page no 4 n bits 80 index `uniq_c1` of table `test
```

唯一索引删除一条已经标记删除的记录的锁情况为：

- 加锁等待时: 2 lock struct(s) ，持有IX锁，等待记录上的X锁
- 加锁成功时：3 lock struct(s)，持有IX,行锁，和gap锁，和非唯一索引删除一条标记为已删除的记录的情况一模一样。

3. 总结

1. 在非唯一索引的情况下，删除一条存在的记录是有gap锁，锁住记录本身和记录之前的gap
2. 在唯一索引和主键的情况下删除一条存在的记录，因为都是唯一值，进行删除的时候，是不会有gap存在
3. 非唯一索引，唯一索引和主键在删除一条不存在的记录，均会在这个区间加gap锁
4. 通过非唯一索引和唯一索引去删除一条标记为删除的记录的时候，都会请求该记录的行锁，同时锁住记录之前的gap
5. RC 情况下是没有gap锁的，除了遇到唯一键冲突的情况，如插入唯一键冲突。