MySQL学习笔记（Day031：锁_4）

MySQL学习

一. 插入意向锁（insert intention lock）

- 插入意向锁 本质 上就是个 `Gap Lock`
  - 普通Gap Lock 不允许 在 （上一条记录，本记录） 范围内插入数据
  - 插入意向锁Gap Lock 允许 在 （上一条记录，本记录） 范围内插入数据
- 插入意向锁的作用是为了 提高并发插入的性能 ， 多个事务 同时写入 不同数据 至同一索引范围（区间）内，并不需要等待其他事务完成，不会发生锁等待

1.1. 插入的过程

假设现在有记录 `10，30，50，70` ；且为 主键 ，需要插入记录 `25` 。

1. 找到 小于等于25的记录 ，这里是 `10`
2. 找到 记录10的下一条记录 ，这里是 `30`
3. 判断 下一条记录30 上是否有锁（*如果有=25的情况，后面再讨论*）
   - 判断 `30` 上面如果 没有锁 ，则 可以插入
   - 判断 `30` 上面如果有 `Record Lock` ，则 可以插入
   - 判断 `30` 上面如果有 `Gap Lock / Next-Key Lock` ，则无法插入，因为锁的范围是 `(10, 30) / (10, 30]` ；在 `30` 上增加 `insert intention lock`（*此时处于 `waiting` 状态*），当 `Gap Lock / Next-Key Lock` 释放时，等待的事物（transaction）将被 唤醒 ，此时 记录30 上才能获得 `insert intention lock` ，然后再插入 记录25

注意：一个事物 insert 25 且没有提交，另一个事物 delete 25 时，记录25上会有 `Record Lock`

1.2. 插入意向锁的演示

```
--
-- 终端会话1
--
mysql> set tx_isolation="REPEATABLE-READ";
Query OK, 0 rows affected (0.00 sec)

mysql> desc t_lock_1;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| a     | int(11) | NO   | PRI | NULL    |       |
+-------+---------+------+-----+---------+-------+
1 row in set (0.00 sec)

mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from t_lock_1  where a<=13 for update;  -- RR下默认为Next-Key lock，status与之前一致
+----+
| a  |
+----+
| 10 |
| 11 |
| 13 |
+----+
3 rows in set (0.00 sec)


--
-- 终端会话2
--
mysql> set tx_isolation="REPEATABLE-READ";
Query OK, 0 rows affected (0.00 sec)

mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> set innodb_lock_wait_timeout=60;  -- 将锁等待时间改成60秒，原来配置中为5秒，来不及切换测试
Query OK, 0 rows affected (0.00 sec)

mysql> insert into t_lock_1 values (12);
-- waiting......（被阻塞了，在这里等待）


--
-- 终端会话3
--
mysql> show engine innodb status\G
--  ----------省略部分输出----------
---TRANSACTION 25645, ACTIVE 4 sec inserting
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 1136, 1 row lock(s)
MySQL thread id 2, OS thread handle 140512469145344, query id 161 localhost root update
                                  --------------------
insert into t_lock_1 values (12)  -- 等待插入的SQL语句
                                  --------------------
------- TRX HAS BEEN WAITING 4 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 140 page no 3 n bits 72 index PRIMARY of table `burn_test`.`t_lock_1` trx id 25645 lock_mode X locks gap before rec insert intention waiting  -- 插入记录12的事物等待中（被终端会话1中的事物阻塞了），等待 获得 插入意向锁
Record lock, heap no 4 PHYSICAL RECORD: n_fields 3; compact format; info bits 0
 0: len 4; hex 8000000d; asc     ;;
 1: len 6; hex 00000000602b; asc     `+;;
 2: len 7; hex ab000000470128; asc     G (;;

------------------
TABLE LOCK table `burn_test`.`t_lock_1` trx id 25645 lock mode IX
RECORD LOCKS space id 140 page no 3 n bits 72 index PRIMARY of table `burn_test`.`t_lock_1` trx id 25645 lock_mode X locks gap before rec insert intention waiting
Record lock, heap no 4 PHYSICAL RECORD: n_fields 3; compact format; info bits 0
 0: len 4; hex 8000000d; asc     ;;
 1: len 6; hex 00000000602b; asc     `+;;
 2: len 7; hex ab000000470128; asc     G (;;

---TRANSACTION 25644, ACTIVE 36 sec   --  以下是 终端会话1中， <=13 for update产生的Next-Key Lock信息
2 lock struct(s), heap size 1136, 4 row lock(s)
MySQL thread id 5, OS thread handle 140512468612864, query id 113 localhost root cleaning up
TABLE LOCK table `burn_test`.`t_lock_1` trx id 25644 lock mode IX
RECORD LOCKS space id 140 page no 3 n bits 72 index PRIMARY of table `burn_test`.`t_lock_1` trx id 25644 lock_mode X
Record lock, heap no 2 PHYSICAL RECORD: n_fields 3; compact format; info bits 0
 0: len 4; hex 8000000a; asc     ;;
 1: len 6; hex 00000000602b; asc     `+;;
 2: len 7; hex ab000000470110; asc     G  ;;

Record lock, heap no 3 PHYSICAL RECORD: n_fields 3; compact format; info bits 0
 0: len 4; hex 8000000b; asc     ;;
 1: len 6; hex 00000000602b; asc     `+;;
 2: len 7; hex ab0000047011c; asc     G  ;;

Record lock, heap no 4 PHYSICAL RECORD: n_fields 3; compact format; info bits 0
 0: len 4; hex 8000000d; asc     ;;
 1: len 6; hex 00000000602b; asc     `+;;
 2: len 7; hex ab000000470128; asc     G (;;

Record lock, heap no 5 PHYSICAL RECORD: n_fields 3; compact format; info bits 0
 0: len 4; hex 80000014; asc     ;;
 1: len 6; hex 00000000602b; asc     `+;;
 2: len 7; hex ab000000470134; asc     G 4;;


 --
 -- 终端会话1
 --
mysql> commit;  -- 终端会话1 中的事物提交，相当与释放了锁
Query OK, 0 rows affected (0.00 sec)


--
-- 终端会话2
--
mysql> insert into t_lock_1 values(12);  -- 这个是刚才输入的SQL语句，之前是阻塞的
Query OK, 1 row affected (17.40 sec)  -- 因为终端会话1中的事物提交了，所以此时插入成功


--
-- 终端会话3
--
mysql> show engine innodb status\G
--  ----------省略部分输出----------
2 lock struct(s), heap size 1136, 1 row lock(s), undo log entries 1
MySQL thread id 8, OS thread handle 140333220423424, query id 300 localhost root cleaning up
TABLE LOCK table `burn_test`.`t_lock_1` trx id 26177 lock mode IX
RECORD LOCKS space id 140 page no 3 n bits 72 index PRIMARY of table `burn_test`.`t_lock_1` trx id 26177 lock_mode X locks gap before rec insert intention  -- 插入意向锁已经获得了
Record lock, heap no 4 PHYSICAL RECORD: n_fields 3; compact format; info bits 0
 0: len 4; hex 8000000d; asc     ;;          --  在a=13的记录上有插入意向锁
 1: len 6; hex 00000000602b; asc     `+;;
 2: len 7; hex ab000000470128; asc     G (;;
```

1.3. 插入意向锁提高插入的并发性演示

```
--
-- 终端会话1
--
mysql> set tx_isolation="REPEATABLE-READ";
Query OK, 0 rows affected (0.00 sec)

mysql> create table t_lock_5 (a int ,primary key(a));
Query OK, 0 rows affected (0.14 sec)

mysql> insert into t_lock_5 values(5),(20),(50);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> begin;
Query OK, 0 rows affected (0.04 sec)

mysql> select * from t_lock_5 where a<=50 for update;
+----+
| a  |
+----+
|  5 |
| 20 |
| 50 |
+----+
3 rows in set (0.00 sec)


--
-- 终端会话2
--
mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into t_lock_5 values(25);
-- waiting... 被 终端会话1 中的事物阻塞


--
-- 终端会话1
--
mysql> commit;
Query OK, 0 rows affected (0.00 sec)


--
-- 终端会话2
--
mysql> insert into t_lock_5 values(25);
Query OK, 1 row affected (30.46 sec) -- 由于 终端会话1 中的事物提交了，现在获得了插入意向锁


--
-- 终端会话3
--
mysql> show engine innodb status\G
--  ----------省略部分输出----------
2 lock struct(s), heap size 1136, 1 row lock(s), undo log entries 1
MySQL thread id 9, OS thread handle 140333220157184, query id 402 localhost root cleaning up
TABLE LOCK table `burn_test`.`t_lock_5` trx id 26192 lock mode IX
RECORD LOCKS space id 147 page no 3 n bits 72 index PRIMARY of table `burn_test`.`t_lock_5` trx id 26192 lock_mode X locks gap before rec insert intention  -- 插入意向锁已经获得
Record lock, heap no 4 PHYSICAL RECORD: n_fields 3; compact format; info bits 0
 0: len 4; hex 80000032; asc    2;;  -- 记录a=50上已经有了插入意向锁
 1: len 6; hex 00000000664a; asc     fJ;;
 2: len 7; hex c30000002a0128; asc     * (;;


--
-- 终端会话1
--
mysql> begin;  -- 在终端会话1中再开启一个新的事物（此时终端会话2中的事物其实还未提交）
Query OK, 0 rows affected (0.00 sec)

mysql> insert into t_lock_5 values(30);  -- 插入小于50，大于20的值，插入成功
Query OK, 1 row affected (0.00 sec)

mysql> insert into t_lock_5 values(31);  -- 插入小于50，大于20的值，插入成功
Query OK, 1 row affected (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.02 sec)

-- 这样就可以并发的插入记录了，而不需要一个事物等待另一事物
-- 当所有相关的插入的事物都提交后，50上的插入意向锁 便会释放
```

二. Read Committed

```
--
-- 终端会话1
--
mysql> set tx_isolation='READ-COMMITTED';
Query OK, 0 rows affected (0.00 sec)

mysql> select * from t_lock_1 where a <= 13 for update;
+----+
| a  |
+----+
| 10 |
| 11 |
| 13 |
+----+
3 rows in set (0.00 sec)

--
-- 终端会话2
--
mysql> set tx_isolation="REPEATABLE-READ";
mysql> show engine innodb status\G
--  ----------省略部分输出----------
---TRANSACTION 26212, ACTIVE 26 sec
2 lock struct(s), heap size 1136, 3 row lock(s)
MySQL thread id 16, OS thread handle 140333220423424, query id 625 localhost root cleaning up
TABLE LOCK table `burn_test`.`t_lock_1` trx id 26212 lock mode IX
RECORD LOCKS space id 140 page no 3 n bits 72 index PRIMARY of table `burn_test`.`t_lock_1` trx id 26212 lock_mode X locks rec but not gap  -- 在RC隔离级别下，变成了 记录锁（同时会出现幻读问题）
Record lock, heap no 2 PHYSICAL RECORD: n_fields 3; compact format; info bits 0
 0: len 4; hex 8000000a; asc     ;;
 1: len 6; hex 00000000602b; asc     `+;;
 2: len 7; hex ab000000470110; asc     G  ;;

Record lock, heap no 3 PHYSICAL RECORD: n_fields 3; compact format; info bits 0
 0: len 4; hex 8000000b; asc     ;;
 1: len 6; hex 00000000602b; asc     `+;;
 2: len 7; hex ab0000047011c; asc     G  ;;

Record lock, heap no 4 PHYSICAL RECORD: n_fields 3; compact format; info bits 0
 0: len 4; hex 8000000d; asc     ;;
 1: len 6; hex 00000000602b; asc     `+;;
 2: len 7; hex ab000000470128; asc     G (;;
```

> **在大部分情况下，RC隔离级别下没有GAP锁，但是一些场景下，线上仍可能出现**
> **设置了RC隔离级别后， `binlog_format = row` ，否则会出现主从不一致的情况**

2.1. 非索引列的等值查询

```
--
-- 终端会话1
--
mysql> desc t_lock_3;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| a     | int(11) | NO   |     | NULL    |       |  -- 没有显示的创建主键
+-------+---------+------+-----+---------+-------+
1 row in set (0.00 sec)

mysql> set tx_isolation='READ-COMMITTED';
Query OK, 0 rows affected (0.00 sec)

mysql> select * from t_lock_3 where a = 13 for update;
+----+
| a  |
+----+
| 13 |
+----+
1 row in set (0.00 sec)


--
-- 终端会话2
--
mysql> show engine innodb status\G
--  ----------省略部分输出----------
2 lock struct(s), heap size 1136, 1 row lock(s)
MySQL thread id 16, OS thread handle 140333220423424, query id 632 localhost root cleaning up
TABLE LOCK table `burn_test`.`t_lock_3` trx id 26213 lock mode IX
RECORD LOCKS space id 143 page no 3 n bits 72 index GEN_CLUST_INDEX of table `burn_test`.`t_lock_3` trx id 26213 lock_mode X locks rec but not gap  -- 只有一个记录锁，且锁住的是GEN_CLUST_INDEX
Record lock, heap no 4 PHYSICAL RECORD: n_fields 4; compact format; info bits 0
 0: len 6; hex 000000001206; asc       ;;  -- 主键
 1: len 6; hex 000000006250; asc     bP;;
 2: len 7; hex c50000002a012c; asc     * ,;;
 3: len 4; hex 8000000d; asc     ;;   -- a = 13
```

> 当查询条件是非索引列的等值查询， RR 隔离级别下会锁住每个记录（形成表锁的效果），而 RC 隔离级别下只锁住查询条件的记录本身

2.2. 二级索引列的等值查询

```
--
-- 终端会话1
--
mysql> alter table t_lock_3 add index idx_a (a);
Query OK, 0 rows affected (0.14 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> set tx_isolation='READ-COMMITTED';
Query OK, 0 rows affected (0.00 sec)

mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from t_lock_3 where a = 13 for update;
+----+
| a  |
+----+
| 13 |
+----+
1 row in set (0.00 sec)


--
-- 终端会话2
--
mysql> set tx_isolation='READ-COMMITTED';
Query OK, 0 rows affected (0.00 sec)

mysql> show engine innodb status\G
--  ----------省略部分输出----------
3 lock struct(s), heap size 1136, 2 row lock(s)
MySQL thread id 16, OS thread handle 140333220423424, query id 641 localhost root cleaning up
TABLE LOCK table `burn_test`.`t_lock_3` trx id 26222 lock mode IX
RECORD LOCKS space id 143 page no 4 n bits 72 index idx_a of table `burn_test`.`t_lock_3` trx id 26222 lock_mode X locks rec but not gap   --  index idx_a  二级索引上的 Record Lock
Record lock, heap no 4 PHYSICAL RECORD: n_fields 2; compact format; info bits 0
 0: len 4; hex 8000000d; asc     ;;
 1: len 6; hex 000000001206; asc       ;;

RECORD LOCKS space id 143 page no 3 n bits 72 index GEN_CLUST_INDEX of table `burn_test`.`t_lock_3` trx id 26222 lock_mode X locks rec but not gap   -- index GEN_CLUST_INDEX 聚集索引上的 Record Lock
Record lock, heap no 4 PHYSICAL RECORD: n_fields 4; compact format; info bits 0
 0: len 6; hex 000000001206; asc       ;;
 1: len 6; hex 000000006250; asc     bP;;
 2: len 7; hex c50000002a012c; asc     * ,;;
 3: len 4; hex 8000000d; asc     ;;
```

当查询条件是二级索引列的等值查询时

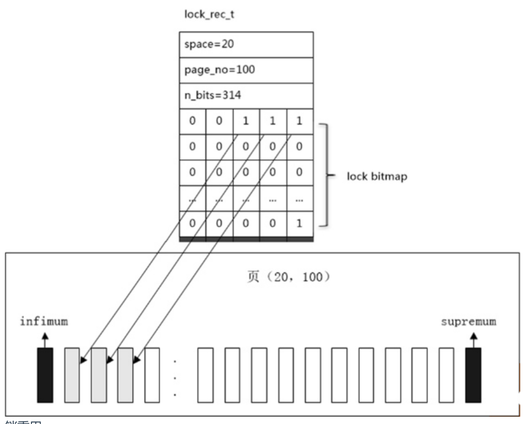
1. RC 模式下，二级索引查询的记录上有一个记录锁， 对应的聚集索引上有一个记录锁
2. RR 模式下，二级索引查询的记录上有一个 Next-Key Lock ，该记录的下一个记录上有一个 Gap-Lock （ *二级索引* ）；对应的聚集索引上有一个记录锁

```
--
-- 终端会话1
--
mysql> set tx_isolation='REPEATABLE-READ';
Query OK, 0 rows affected (0.00 sec)

mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from t_lock_3 where a = 13 for update;
+----+
| a  |
+----+
| 13 |
+----+
1 row in set (0.00 sec)


--
-- 终端会话2
--
mysql> set tx_isolation='REPEATABLE-READ';
Query OK, 0 rows affected (0.00 sec)

mysql> show engine innodb status\G
--  ----------省略部分输出----------
4 lock struct(s), heap size 1136, 3 row lock(s)
MySQL thread id 16, OS thread handle 140333220423424, query id 649 localhost root cleaning up
TABLE LOCK table `burn_test`.`t_lock_3` trx id 26223 lock mode IX
RECORD LOCKS space id 143 page no 4 n bits 72 index idx_a of table `burn_test`.`t_lock_3` trx id 26223 lock_mode X  -- index idx_a 二级索引上的 Next-Key Lock
Record lock, heap no 4 PHYSICAL RECORD: n_fields 2; compact format; info bits 0
 0: len 4; hex 8000000d; asc     ;;
 1: len 6; hex 000000001206; asc       ;;

RECORD LOCKS space id 143 page no 3 n bits 72 index GEN_CLUST_INDEX of table `burn_test`.`t_lock_3` trx id 26223 lock_mode X locks rec but not gap  -- index GEN_CLUST_INDEX 聚集索引上的 Record-lock
Record lock, heap no 4 PHYSICAL RECORD: n_fields 4; compact format; info bits 0
 0: len 6; hex 000000001206; asc       ;;
 1: len 6; hex 000000006250; asc     bP;;
 2: len 7; hex c50000002a012c; asc     * ,;;
 3: len 4; hex 8000000d; asc     ;;

RECORD LOCKS space id 143 page no 4 n bits 72 index idx_a of table `burn_test`.`t_lock_3` trx id 26223 lock_mode X locks gap before rec  -- index idx_a 二级索引上的Gap Lock（针对下一个记录20）
Record lock, heap no 5 PHYSICAL RECORD: n_fields 2; compact format; info bits 0
 0: len 4; hex 80000014; asc     ;;
 1: len 6; hex 000000001207; asc       ;;
```

RR与RC性能对比

三. 锁的算法（二）

- 每个事物每个页 一个锁对象
  - 约100个字节
- 通过位图存放锁信息
  - 内存占用少
- 没有锁升级



- 锁重用

```
--


-- 终端会话1
--

mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from t_lock_1 where a=13 for update;
+----+
| a  |
+----+
| 13 |
+----+
1 row in set (0.00 sec)

mysql> select * from t_lock_1 where a=13 lock in share mode;
+----+
| a  |
+----+
| 13 |
+----+
1 row in set (0.00 sec)


--

-- 终端会话2
--

mysql> show engine innodb status\G
--  ----------省略部分输出----------
2 lock struct(s), heap size 1136, 1 row lock(s)
MySQL thread id 19, OS thread handle 140333220423424, query id 745 localhost root cleaning up
TABLE LOCK table `burn_test`.`t_lock_1` trx id 26224 lock mode IX
RECORD LOCKS space id 140 page no 3 n bits 72 index PRIMARY of table `burn_test`.`t_lock_1` trx id 26224 lock_mode X locks rec but not gap  -- 只有一把锁，不会出现两个锁，进一步降低了开销
Record lock, heap no 4 PHYSICAL RECORD: n_fields 3; compact format; info bits 0
 0: len 4; hex 8000000d; asc     ;;
 1: len 6; hex 00000000602b; asc     `+;;
 2: len 7; hex ab000000470128; asc     G (;;
```

- 隐式锁

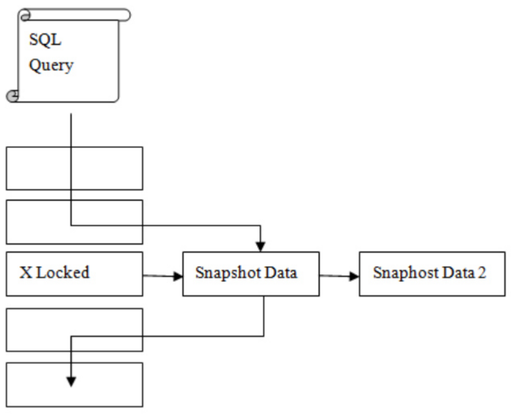
```
--


-- 终端会话1
--

mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into t_lock_1 values(12);
Query OK, 1 row affected (0.00 sec)
-- 此时未提交，  记录12上面应该有锁


--

-- 终端会话2
--

mysql> show engine innodb status\G  -- 但是这里只能看到在表上的IX锁，而看不到记录12的记录锁
                                    -- 因为show出来的都是显示的锁（已经创建了内存对象的）
                                    -- 此时记录12中持有的是一把 隐式锁
--  ----------省略部分输出----------
1 lock struct(s), heap size 1136, 0 row lock(s), undo log entries 1
MySQL thread id 19, OS thread handle 140333220423424, query id 750 localhost root cleaning up
TABLE LOCK table `burn_test`.`t_lock_1` trx id 26225 lock mode IX


--

-- 终端会话3
--

mysql> select * from t_lock_1 where a=12 for update;  -- 另外一个事物中去修改a=12的这条未提交的记录
-- waiting......

-- 此时  隐式锁  转换成了  显示锁

--

-- 终端会话2
--

mysql> show engine innodb status\G    --  此时就显示了锁的信息
--  ----------省略部分输出----------
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 1136, 2 row lock(s)
MySQL thread id 20, OS thread handle 140333220157184, query id 801 localhost root statistics
select * from t_lock_1 where a=12 for update
------- TRX HAS BEEN WAITING 2 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 140 page no 3 n bits 72 index PRIMARY of table `burn_test`.`t_lock_1` trx id 26226 lock_mode X locks rec but not gap waiting   -- 新的事物在等待这个 Record Lock
Record lock, heap no 6 PHYSICAL RECORD: n_fields 3; compact format; info bits 0
 0: len 4; hex 8000000c; asc     ;;
 1: len 6; hex 000000006671; asc     fq;;
 2: len 7; hex df000000270110; asc     '  ;;

------------------
TABLE LOCK table `burn_test`.`t_lock_1` trx id 26226 lock mode IX
RECORD LOCKS space id 140 page no 3 n bits 72 index PRIMARY of table `burn_test`.`t_lock_1` trx id 26226 lock_mode X locks rec but not gap waiting
Record lock, heap no 6 PHYSICAL RECORD: n_fields 3; compact format; info bits 0
 0: len 4; hex 8000000c; asc     ;;
 1: len 6; hex 000000006671; asc     fq;;
 2: len 7; hex df000000270110; asc     '  ;;

---TRANSACTION 26225, ACTIVE 313 sec
2 lock struct(s), heap size 1136, 1 row lock(s), undo log entries 1
MySQL thread id 19, OS thread handle 140333220423424, query id 750 localhost root cleaning up
TABLE LOCK table `burn_test`.`t_lock_1` trx id 26225 lock mode IX
RECORD LOCKS space id 140 page no 3 n bits 72 index PRIMARY of table `burn_test`.`t_lock_1` trx id 26225 lock_mode X locks rec but not gap  --  Record lock
Record lock, heap no 6 PHYSICAL RECORD: n_fields 3; compact format; info bits 0
 0: len 4; hex 8000000c; asc     ;;
 1: len 6; hex 000000006671; asc     fq;;
 2: len 7; hex df000000270110; asc     '  ;;

-- 尽管此时还没有提交（没有创建内存对象），但是可以通过 trx_id 来判断该记录上是否有锁；
-- 通过 在线事物列表 中是否存在该 trx_id 进行判断，因为事物没有提交（即活跃），就表示在这个记录上是有锁的
```

四. 一致性的非锁定读

在隔离级别为 RU、RC、RR 时，读取（ select ）操作是不会加锁的，通过行多版本控制（ MVCC ）的方式读取当前执行时间点的记录。

```
--
-- 终端会话1
--
mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from t_lock_1 where a=13;  -- 非锁定的读，不会阻塞
+----+
| a  |
+----+
| 13 |
+----+
1 row in set (0.00 sec)
```

```
--
-- 终端会话2
--
mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from t_lock_1 where a=13; -- 非锁定的读，不会阻塞
+----+
| a  |
+----+
| 13 |
+----+
1 row in set (0.00 sec)
```



> 通过 UNDO 指针的指向，可以读取前一个版本甚至前几个版本的记录（*即通过UNDO来构造版本记录*），从而实现 快照读 （ Snapshot Read ）
>
> 通过 trx_id 判断该记录是否被锁住（在线事物列表），从而决定是否要读取之前的版本（ UNDO ）

五. 死锁

5.1. 死锁的介绍

- 两个或两个以上的事务在执行过程中，因争夺锁资源而造成的一种互相等待的现象
  - AB-BA, A和B互相等待
- 解决死锁
  - 超时， *死锁* 和 *锁超时* 不是同一个东西（ *锁超时 是 解决死锁 的一种方式* ）
    - --innodb_lock_timeout ，默认是50s
- wait-for graph
  - 自动死锁检测

> 数据库中的死锁和程序中的死锁不同，数据库中的死锁是不能完全避免的，且数据库中的死锁有检测机制

5.2. AB-BA死锁演示

```
--
-- 终端会话1
--
mysql> begin;   -- step_1
Query OK, 0 rows affected (0.00 sec)

mysql> delete from t_lock_1 where a=10;   -- step_3
Query OK, 1 row affected (0.00 sec)

mysql> delete from t_lock_1 where a=11;   -- step_6
ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting transaction  -- 出现死锁
```

```
--
-- 终端会话2
--
mysql> begin;   -- step_2
Query OK, 0 rows affected (0.00 sec)

mysql> delete from t_lock_1 where a=11;    -- step_4
Query OK, 1 row affected (0.00 sec)

mysql> delete from t_lock_1 where a=10;    -- step_5
Query OK, 1 row affected (3.98 sec)
```

在MySQL出现错误（ 非死锁 ）的时候，线程中的事物是 中止 的，需要 显示 的提交或回滚；
而 死锁 时，MySQL会 自动回滚

假如一个事物中有10个步骤，当执行到步骤8时

1. 出现普通错误，在步骤8中停止，只要重新执行步骤8即可，然后再继续执行后续操作
2. 出现死锁时，该事物中，步骤8之前的操作都会自动回滚

```
mysql> show engine innodb status\G    --  显示最近的死锁信息，重启后就看不到了
--  ----------省略部分输出----------
------------------------
LATEST DETECTED DEADLOCK
------------------------
2016-02-13 15:23:27 0x7fa1dfbcd700
*** (1) TRANSACTION:
TRANSACTION 26240, ACTIVE 9 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 3 lock struct(s), heap size 1136, 2 row lock(s), undo log entries 1
MySQL thread id 23, OS thread handle 140333220423424, query id 963 localhost root updating
delete from t_lock_1 where a=10
*** (1) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 140 page no 3 n bits 72 index PRIMARY of table `burn_test`.`t_lock_1` trx id 26240 lock_mode X locks rec but not gap waiting
Record lock, heap no 2 PHYSICAL RECORD: n_fields 3; compact format; info bits 32
 0: len 4; hex 8000000a; asc     ;;
 1: len 6; hex 00000000667f; asc     f ;;
 2: len 7; hex 67000000330ab9; asc g   3  ;;

*** (2) TRANSACTION:
TRANSACTION 26239, ACTIVE 14 sec starting index read, thread declared inside InnoDB 5000
mysql tables in use 1, locked 1
3 lock struct(s), heap size 1136, 2 row lock(s), undo log entries 1
MySQL thread id 22, OS thread handle 140333220157184, query id 964 localhost root updating
delete from t_lock_1 where a=11
*** (2) HOLDS THE LOCK(S):
RECORD LOCKS space id 140 page no 3 n bits 72 index PRIMARY of table `burn_test`.`t_lock_1` trx id 26239 lock_mode X locks rec but not gap
Record lock, heap no 2 PHYSICAL RECORD: n_fields 3; compact format; info bits 32
 0: len 4; hex 8000000a; asc     ;;
 1: len 6; hex 00000000667f; asc     f ;;
 2: len 7; hex 67000000330ab9; asc g   3  ;;

*** (2) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 140 page no 3 n bits 72 index PRIMARY of table `burn_test`.`t_lock_1` trx id 26239 lock_mode X locks rec but not gap waiting
Record lock, heap no 3 PHYSICAL RECORD: n_fields 3; compact format; info bits 32
 0: len 4; hex 8000000b; asc     ;;
 1: len 6; hex 000000006680; asc     f ;;
 2: len 7; hex 68000000330259; asc h   3 Y;;

*** WE ROLL BACK TRANSACTION (2)
------------
TRANSACTIONS
------------
Trx id counter 26246
Purge done for trx's n:o < 26245 undo n:o < 0 state: running but idle
History list length 316
LIST OF TRANSACTIONS FOR EACH SESSION:
---TRANSACTION 421809722470000, not started
0 lock struct(s), heap size 1136, 0 row lock(s)
---TRANSACTION 421809722468176, not started
0 lock struct(s), heap size 1136, 0 row lock(s)
---TRANSACTION 26240, ACTIVE 748 sec
3 lock struct(s), heap size 1136, 2 row lock(s), undo log entries 2
MySQL thread id 23, OS thread handle 140333220423424, query id 963 localhost root cleaning up
TABLE LOCK table `burn_test`.`t_lock_1` trx id 26240 lock mode IX
RECORD LOCKS space id 140 page no 3 n bits 72 index PRIMARY of table `burn_test`.`t_lock_1` trx id 26240 lock_mode X locks rec but not gap
Record lock, heap no 3 PHYSICAL RECORD: n_fields 3; compact format; info bits 32
 0: len 4; hex 8000000b; asc     ;;
 1: len 6; hex 000000006680; asc     f ;;
 2: len 7; hex 68000000330259; asc h   3 Y;;

RECORD LOCKS space id 140 page no 3 n bits 72 index PRIMARY of table `burn_test`.`t_lock_1` trx id 26240 lock_mode X locks rec but not gap
Record lock, heap no 2 PHYSICAL RECORD: n_fields 3; compact format; info bits 32
 0: len 4; hex 8000000a; asc     ;;
 1: len 6; hex 000000006680; asc     f ;;
 2: len 7; hex 6800000033027b; asc h   3 {;;


mysql> show variables like 'innodb%dead%';
+------------------------------+-------+
| Variable_name                | Value |
+------------------------------+-------+
| innodb_print_all_deadlocks | ON    |  -- 会将死锁信息打印到err_log（搜索 deadlock 关键字）
+------------------------------+-------+
1 row in set (0.00 sec)
```