# MySQL学习笔记（Day030：锁_3）

MySQL学习



## 一. 事物隔离级别（二）

### 1.1. 脏读

```
--
-- 终端会话1
--
mysql> use burn_test;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed

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

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

mysql> begin;  -- 事物1
Query OK, 0 rows affected (0.00 sec)

mysql> insert t_ru values(1);
Query OK, 1 row affected (0.01 sec)
-- 事物1没有提交


--
-- 终端会话2
--
mysql> use burn_test;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed

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

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

mysql> select * from t_ru;
+------+
| a    |
+------+
|    1 |  -- 事物2中能看到事物1中插入的数据，但是事物1还没有提交
+------+
1 row in set (0.00 sec)

mysql> insert into t_ru values(2);
Query OK, 1 row affected (0.00 sec)
-- 事物2此时还未commit

--
-- 终端会话1
--
mysql> select * from t_ru;
+------+
| a    |
+------+
|    1 |
|    2 |  -- 会话2中的事物2还没有commit，在会话1中的事物1就能读取到数据，此为脏读
+------+
2 rows in set (0.00 sec)
```

### 1.2. 不可重复读

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

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

mysql> insert into t_ru values(1);
Query OK, 1 row affected (0.00 sec)

mysql> commit;  -- 此时提交了事物1，所以在t_ru表中就有了数据
Query OK, 0 rows affected (0.02 sec)

mysql> begin;  -- 再开启一个新的事物
Query OK, 0 rows affected (0.00 sec)

mysql> select * from t_ru;   -- 在该事物中查询数据，得到一条记录 a = 1
+------+
| a    |
+------+
|    1 |
+------+
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> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> insert  into t_ru select 5;
Query OK, 1 row affected (0.00 sec)
Records: 1  Duplicates: 0  Warnings: 0
-- 此时终端会话2的事物未提交

--
-- 终端会话1
--
mysql> select * from t_ru;  -- 在会话1的事物中查询数据仍只有一条，说明RC隔离级别解决了脏读问题
+------+
| a    |
+------+
|    1 |
+------+
1 row in set (0.00 sec)

--
-- 终端会话2
--
mysql> rollback;  -- 回滚，不插入 a = 5 的记录
Query OK, 0 rows affected (0.01 sec)

mysql> begin;  -- 又开启一个事物
Query OK, 0 rows affected (0.00 sec)

mysql> update t_ru set a = 5 where a = 1;  -- 将a=1的记录更新成a=5
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> commit;  -- 并且提交该事物
Query OK, 0 rows affected (0.02 sec)

--
-- 终端会话1
--
-- 此时的终端会话1还在事物中
mysql> select * from t_ru;
+------+
| a    |
+------+
|    5 |  -- 得到了记录 a = 5，会话2中的事物update操作，并且commit，已经影响到了会话1中的结果
+------+
1 row in set (0.00 sec)
```

### 1.3. 幻读

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

mysql> select * from t_ru;
+------+
| a    |
+------+
|    5 |
+------+
1 row in set (0.00 sec)


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

mysql> insert into t_ru values(10);
Query OK, 1 row affected (0.00 sec)

mysql> commit;  -- 会话2中的事物已经提交
Query OK, 0 rows affected (0.03 sec)

--
-- 终端会话1
--
mysql> select * from t_ru;
+------+
| a    |
+------+
|    5 |
|   10 | -- 终端会话1中的事物没提交，就看到了终端会话2中的事物提交的结果
+------+
2 rows in set (0.00 sec)
```

> **不可重复读：**在一个事物中，针对同一条记录，执行两次相同的SQL得到的结果不一样
> **幻读：**在一个事物中，执行两次相同的SQL，得到了不同的结果集（新增了部分记录或者缺失了部分记录）（*不是同一条记录*）

# 二. 锁的算法（一）

- **Record Lock**
  - 单个行记录上的锁
- **Gap Lock**
  - 锁定一个范围，但不包含记录本身
- **Next-Key Lock**
  - **Gap Lock** + **Record Lock**，锁定一个范围，并且锁定记录本身

## 2.1. Record Lock

例如有记录 `10、 30 、 50`（*只有这三个记录*），且在 `记录30上加锁`（*加锁的SQL暂时不管，只看锁算法本身*）

- 对于在 `记录30上加锁` ，表示 `30` 该记录上加了行锁
- **`[30, 30]`**：只锁定记录30自身

## 2.2. Gap Lock

例如有记录 `10、 30 、 50`（*只有这三个记录*），且在 `记录30上加锁`（*加锁的SQL暂时不管，只看锁算法本身*）

- 对于在 `记录30上加锁` ，表示在 `记录10` 到 `记录30` 之间加锁（*10是30的前一个记录（前一条记录，本记录）*）
- **`(10, 30)`**：锁定该范围，不包含两个边界（*不能插入15、20等在这个范围内的数据，但是可以对10和30做删除或者修改*）
- `Gap Lock解决了幻读问题`

## 2.3. Next-Key Lock

例如有记录 `10、 30 、 50`（*只有这三个记录*），且在 `记录30上加锁`（*加锁的SQL暂时不管，只看锁算法本身*）

- 对于在 `记录30上加锁` ，表示在 `记录10` 到 `记录30` 范围加锁（*Gap Lock*）的同时，在 `记录30` 上同时加上 `行锁`
- **`(10, 30]`**：锁定该范围，且包含记录30本身（*不能插入15、20等在这个范围的数据，同时对 `记录30不能删除或修改` ，记录10可以删除或修改*）

> **注意：** `锁住的是索引`

## 2.4. 锁与隔离级别

### 2.4.1. REPEATABLE READ

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

mysql> create table t_lock_1 (a int primary key);
Query OK, 0 rows affected (0.12 sec)

mysql> insert into t_lock_1 values(10),(11),(13),(20);
Query OK, 4 rows affected (0.01 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> begin;
Query OK, 0 rows affected (0.02 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";
Query OK, 0 rows affected (0.00 sec)

mysql> mysql> show engine innodb status\G
--  ----------省略部分输出----------
---TRANSACTION 24624, ACTIVE 43 sec
2 lock struct(s), heap size 1136, 4 row lock(s)
MySQL thread id 4, OS thread handle 139781911455488, query id 190 localhost root cleaning up
TABLE LOCK table `burn_test`.`t_lock_1` trx id 24624 lock mode IX -- 表上有一个IX意向锁
RECORD LOCKS space id 140 page no 3 n bits 72 index PRIMARY of table `burn_test`.`t_lock_1` trx id 24624 lock_mode X -- lock_mode X 表示的是Next Key lock
Record lock, heap no 2 PHYSICAL RECORD: n_fields 3; compact format; info bits 0
 0: len 4; hex 8000000a; asc     ;;  -- 记录10上有Record lock
 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     ;;  -- 记录11上有Record lock
 1: len 6; hex 00000000602b; asc     `+;;
 2: len 7; hex ab00000047011c; asc     G  ;;

Record lock, heap no 4 PHYSICAL RECORD: n_fields 3; compact format; info bits 0
 0: len 4; hex 8000000d; asc     ;;  -- 记录13上有Record lock
 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     ;;  -- 记录20上有Record lock
 1: len 6; hex 00000000602b; asc     `+;;
 2: len 7; hex ab000000470134; asc     G 4;;
```

> 即在 `RR` 的隔离级别下，锁住的是 $(-\infty, 10]$ , $(10 , 11]$ , $(11 , 13]$ , $(13 , 20]$
> `锁住20` 是因为在RR级别下，（ `<=13` ）将从表（B+树）的第一个记录（ `10` ）开始比对，一直比对到第一个出现大于13的记录（ `20` ）为止。
>
> **总结：**
> **1. RR级别下的锁默认为 Next-Key Lock**
> **2. RR会对游标打开的所有的记录进行加锁**

### 2.4.2. READ COMMITTED

```
--
-- 终端会话1
--
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_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="READ-COMMITTED";
Query OK, 0 rows affected (0.00 sec)

mysql> show engine innodb status\G
--  ----------省略部分输出----------
---TRANSACTION 24625, ACTIVE 4 sec
2 lock struct(s), heap size 1136, 3 row lock(s)
MySQL thread id 4, OS thread handle 139781911455488, query id 198 localhost root cleaning up
TABLE LOCK table `burn_test`.`t_lock_1` trx id 24625 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 24625 lock_mode X locks rec but not gap -- lock_mode X but not gap 表示的就是 Record lock（记录锁）
Record lock, heap no 2 PHYSICAL RECORD: n_fields 3; compact format; info bits 0
 0: len 4; hex 8000000a; asc     ;;  -- 记录10上有Record lock
 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     ;;  -- 记录11上有Record lock
 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     ;;  -- 记录13上有Record lock
 1: len 6; hex 00000000602b; asc     `+;;
 2: len 7; hex ab000000470128; asc     G (;;
```

即在 RC 的隔离级别下，锁住的是 10，11，13 记录本身（*lock_mode X locks rec but not gap*）

## 2.5. Next-Key lock优化为Record lock

- 锁定的是 一条记录
- 当索引含有唯一约束时

```
--
-- 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_1 where a=13 for update; -- 1:a是主键（唯一约束）2:且返回的记录只有一条
+----+
| 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----------
---TRANSACTION 25128, ACTIVE 49 sec
2 lock struct(s), heap size 1136, 1 row lock(s)
MySQL thread id 2, OS thread handle 140022404839168, query id 52 localhost root cleaning up
TABLE LOCK table `burn_test`.`t_lock_1` trx id 25128 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 25128 lock_mode X locks rec but not gap -- 没有gap的x lock，即为记录锁
Record lock, heap no 4 PHYSICAL RECORD: n_fields 3; compact format; info bits 0
 0: len 4; hex 8000000d; asc     ;;  -- 只对13这个记录加了锁
 1: len 6; hex 00000000602b; asc     `+;;
 2: len 7; hex ab000000470128; asc     G (;;
```

这里可以降级为 Record Lock 是因为返回的记录具有 唯一性 ，不会存在幻读问题。
当唯一索引是 复合索引 时，且查询条件只 包含部分列 的话，其实还是有Gap lock

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

mysql> create table t_lock_2 (a int, b int, primary key(a,b));
Query OK, 0 rows affected (0.15 sec)

mysql> insert into t_lock_2 values(1,2),(1,4),(1,6);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> begin;
Query OK, 0 rows affected (0.03 sec)

mysql> select * from t_lock_2 where b=2 for update;
+---+---+
| a | b |
+---+---+
| 1 | 2 |
+---+---+
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----------
---TRANSACTION 25157, ACTIVE 3 sec
2 lock struct(s), heap size 1136, 4 row lock(s)
MySQL thread id 6, OS thread handle 140022404572928, query id 176 localhost root cleaning up
TABLE LOCK table `burn_test`.`t_lock_2` trx id 25157 lock mode IX
RECORD LOCKS space id 142 page no 3 n bits 72 index PRIMARY of table `burn_test`.`t_lock_2` trx id 25157 lock_mode X  -- 由于 b=2 是复合索引中的一部分，所以没办法降级为record lock，还是使用next-key lock
Record lock, heap no 1 PHYSICAL RECORD: n_fields 1; compact format; info bits 0
 0: len 8; hex 73757072656d756d; asc supremum;;

Record lock, heap no 2 PHYSICAL RECORD: n_fields 4; compact format; info bits 0
 0: len 4; hex 80000001; asc     ;;
 1: len 4; hex 80000002; asc     ;;
 2: len 6; hex 000000006240; asc     b@;;
 3: len 7; hex b90000004e0110; asc     N  ;;

Record lock, heap no 3 PHYSICAL RECORD: n_fields 4; compact format; info bits 0
 0: len 4; hex 80000001; asc     ;;
 1: len 4; hex 80000004; asc     ;;
 2: len 6; hex 000000006240; asc     b@;;
 3: len 7; hex b90000004e0121; asc     N !;;

Record lock, heap no 4 PHYSICAL RECORD: n_fields 4; compact format; info bits 0
 0: len 4; hex 80000001; asc     ;;
 1: len 4; hex 80000006; asc     ;;
 2: len 6; hex 000000006240; asc     b@;;
 3: len 7; hex b90000004e0132; asc     N 2;;
```

## 2.6. 非唯一索引的等值查询

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

mysql> create table t_lock_3 (a int not null);  -- 与 t_lock_1 相比，没有显示定义a为主键
Query OK, 0 rows affected (0.12 sec)

mysql> insert into t_lock_3 values(10),(11),(13),(20);
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> begin;
Query OK, 0 rows affected (0.03 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----------
---TRANSACTION 25173, ACTIVE 13 sec
2 lock struct(s), heap size 1136, 5 row lock(s)
MySQL thread id 8, OS thread handle 140022404572928, query id 239 localhost root cleaning up
TABLE LOCK table `burn_test`.`t_lock_3` trx id 25173 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 25173 lock_mode X  -- Next-Key lock
Record lock, heap no 1 PHYSICAL RECORD: n_fields 1; compact format; info bits 0
 0: len 8; hex 73757072656d756d; asc supremum;;

Record lock, heap no 2 PHYSICAL RECORD: n_fields 4; compact format; info bits 0
 0: len 6; hex 000000001204; asc       ;;
 1: len 6; hex 000000006250; asc     bP;;
 2: len 7; hex c50000002a0110; asc     *  ;;
 3: len 4; hex 8000000a; asc     ;;  -- 记录10上加了记录锁

Record lock, heap no 3 PHYSICAL RECORD: n_fields 4; compact format; info bits 0
 0: len 6; hex 000000001205; asc       ;;
 1: len 6; hex 000000006250; asc     bP;;
 2: len 7; hex c50000002a011e; asc     *  ;;
 3: len 4; hex 8000000b; asc     ;;  -- 记录11上加了记录锁

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     ;;  -- 记录13上加了记录锁

Record lock, heap no 5 PHYSICAL RECORD: n_fields 4; compact format; info bits 0
 0: len 6; hex 000000001207; asc       ;;
 1: len 6; hex 000000006250; asc     bP;;
 2: len 7; hex c50000002a013a; asc     * :;;
 3: len 4; hex 80000014; asc     ;;  -- 记录20上加了记录锁
```

> `t_lock_3` 中存在着系统自己生成的主键，当查询条件为 `a=10` 时， 游标 要访问 所有的记录（ 索引），来判断主键对应的a列是否等于10，即需要扫描所有主键，则会将 所有记录（索引）都锁住 。形成了表锁的效果。
>
> 总结：在 `RR` 的隔离级别下，当查询的 列没有索引 时，会 锁住所有记录

## 2.7. 假如RR没有Gap Lock

```
记录：  1 3 5 7

     +
     |         tx1        |      tx2
     |                    |
     |    +-----------+   |
     |    |           |   |
     |    |   begin;  |   |
     |    |           |   |
     |    | del <= 7; |   |
     |    |           |   |
     |    +-----------+   |
Time |                    |  +-----------+
     |                    |  |           |
     |                    |  |  begin;   |
     |                    |  |           |
     |                    |  | insert 6; |
     |                    |  |           |
     |                    |  |  commit;  |
     |                    |  +-----------+
     |    +-----------+   |
     |    |           |   |
     |    |  commit;  |   |
     |    |           |   |
     |    +-----------+   |
     |                    |
     v
```

> 如果在 `RR` 级别下，且 没有Gap锁 ，则此时 `tx1` 锁住的是 `1，3，5，7`（ 记录锁 ），并 标记为删除 ，同时 `tx2` 可以插入记录 `6` 。即此时表中的记录为6。
> 而在 `Log` 中记录的是 `insert 6;del<=7;`（ *先提交的在前面* ），如果此时有slave机器进行同步，该表中的记录为空。
> 此时， 主从 机器上的 数据不一致 。
>
> 当 有了Gap Lock 后， `tx2` 的 `insert 6;` 需要等待 `tx1` 的 `del<=7;` 执行完成后才能执行，此时的Log为 `del<=7;insert 6;` ，也就不会有数据不一致的问题（ 符合隔离性要求 ）

## 2.8. 二级索引与锁

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

mysql> create table t_lock_4(a int, b int, primary key(a), key(b));
Query OK, 0 rows affected (0.15 sec)

mysql> insert into t_lock_4 values(1,1),(3,1),(5,3),(7,6),(10,8);
Query OK, 5 rows affected (0.00 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> select * from t_lock_4 where b=3 for update;
+---+------+
| a | b    |
+---+------+
| 5 |    3 |
+---+------+
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----------
---TRANSACTION 25185, ACTIVE 3 sec
4 lock struct(s), heap size 1136, 3 row lock(s)
MySQL thread id 9, OS thread handle 140022404572928, query id 304 localhost root cleaning up
TABLE LOCK table `burn_test`.`t_lock_4` trx id 25185 lock mode IX
RECORD LOCKS space id 144 page no 4 n bits 72 index b of table `burn_test`.`t_lock_4` trx id 25185 lock_mode X  -- Next-Key lock  （index b 二级索引）
Record lock, heap no 4 PHYSICAL RECORD: n_fields 2; compact format; info bits 0
 0: len 4; hex 80000003; asc     ;;   -- 二级索引 key=3
 1: len 4; hex 80000005; asc     ;;   -- value = 5（主键）

RECORD LOCKS space id 144 page no 3 n bits 72 index PRIMARY of table `burn_test`.`t_lock_4` trx id 25185 lock_mode X locks rec but not gap  -- Record Lock  （index PRIMARY 聚集索引）
Record lock, heap no 4 PHYSICAL RECORD: n_fields 4; compact format; info bits 0
 0: len 4; hex 80000005; asc     ;;  -- b=3时对应的主键a=5的记录锁
 1: len 6; hex 00000000625a; asc     bZ;;
 2: len 7; hex cd0000002f0128; asc     / (;;
 3: len 4; hex 80000003; asc     ;;

RECORD LOCKS space id 144 page no 4 n bits 72 index b of table `burn_test`.`t_lock_4` trx id 25185 lock_mode X locks gap before rec  -- lock_mode X locks gap 即为 Gap Lock
Record lock, heap no 5 PHYSICAL RECORD: n_fields 2; compact format; info bits 0
 0: len 4; hex 80000006; asc     ;;
 1: len 4; hex 80000007; asc     ;;

-- 所以二级索引锁住的范围是 (1, 3],（3,  6）
-- 主键索引只锁住了a=5的这条记录 [5, 5]

-- 如果不锁住(3,6)还是可以有别的事物插入b=3的记录，从而产生幻读。



-- 以下SQL语句在终端会话2中执行的效果
select * from t_lock_4 where a = 5 lock in share mode; -- 不可执行，因为a=5上有一把记录锁
insert into t_lock_4 values(4, 2); -- 不可以执行，因为b=2在(1, 3]内
insert into t_lock_4 values(6, 5); -- 不可以执行，因为b=5在(3, 6)内
insert into t_lock_4 values(8, 6); -- 可以执行，(8, 6)均不在锁住的范围内
insert into t_lock_4 values(2, 0); -- 可以执行，(2, 0)均不在锁住的范围内
insert into t_lock_4 values(6, 7); -- 可以执行，(6, 7)均不在锁住的范围内


--------------------------------------------------------------------------------

insert into t_lock_4 values(6, 6); -- 不可以执行，二级索引除了看key，还要看主键的值，
                                   -- (6, 6)在(7, 6)前面，在二级索引锁住的范围(3, 6)中
                                   -- 即(6, 6)中的b=6插在(3,6)中间，而(8, 6)中的b=6插在(3,6)之后

-- a :  1  3  5  7  10
-- b :  1  1  3  6  8
-- 其实二级索引锁住的不仅仅是二级索引的Key本身，还有对应的value，也就是主键
-- 「1:1」  「1:3」  「3:5」      「6:7」         「8:10」  -- 锁住的是(「3:5」, 「6:7」)
--                          ^             ^
--                          |             |
--                        「6:6」        「8:6」
--
-- b相同的情况下，a也是排序的；a也是二级索引的一部分


---TRANSACTION 421498906885744, not started
0 lock struct(s), heap size 1136, 0 row lock(s)
---TRANSACTION 25201, ACTIVE 18 sec inserting
mysql tables in use 1, locked 1
LOCK WAIT 4 lock struct(s), heap size 1136, 5 row lock(s), undo log entries 1
MySQL thread id 15, OS thread handle 140022404572928, query id 753 localhost root update
insert into t_lock_4 values(6, 6)
------- TRX HAS BEEN WAITING 3 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 144 page no 4 n bits 80 index b of table `burn_test`.`t_lock_4` trx id 25201 lock_mode X locks gap before rec insert intention waiting
Record lock, heap no 5 PHYSICAL RECORD: n_fields 2; compact format; info bits 0
 0: len 4; hex 80000006; asc     ;;
 1: len 4; hex 80000007; asc     ;;

------------------
TABLE LOCK table `burn_test`.`t_lock_4` trx id 25201 lock mode IX
RECORD LOCKS space id 144 page no 3 n bits 80 index PRIMARY of table `burn_test`.`t_lock_4` trx id 25201 lock mode S
Record lock, heap no 5 PHYSICAL RECORD: n_fields 4; compact format; info bits 0
 0: len 4; hex 80000007; asc     ;;
 1: len 6; hex 00000000625a; asc     bZ;;
 2: len 7; hex cd0000002f0134; asc     / 4;;
 3: len 4; hex 80000006; asc     ;;

RECORD LOCKS space id 144 page no 3 n bits 80 index PRIMARY of table `burn_test`.`t_lock_4` trx id 25201 lock mode S locks gap before rec
Record lock, heap no 7 PHYSICAL RECORD: n_fields 4; compact format; info bits 0
 0: len 4; hex 80000006; asc     ;;
 1: len 6; hex 000000006271; asc     bq;;
 2: len 7; hex de000000520110; asc     R  ;;
 3: len 4; hex 80000006; asc     ;;

RECORD LOCKS space id 144 page no 4 n bits 80 index b of table `burn_test`.`t_lock_4` trx id 25201 lock_mode X locks gap before rec insert intention waiting
Record lock, heap no 5 PHYSICAL RECORD: n_fields 2; compact format; info bits 0
 0: len 4; hex 80000006; asc     ;;
 1: len 4; hex 80000007; asc     ;;
```

> 二级索引上加了 `Record Lock` 或 `Next-Key Lock` ，则对应的主键加 `Record Lock` ；
> 二级索引加 `Gap Lock` ，则对应的主键不加锁；

---

**问题**：为什么在**2.4.1**的例子中 `20` 上加的是**Next-Key Lock**，而**2.8**的例子中 `6` 上加的是**Gap Lock**？

1. 针对上述两个例子的查询，都是通过游标打开后，进行扫描访问
2. 在**2.4.1**的例子中，查询的条件（ `a <=13 for update` ）是 主键 查询，所以在 记录13 的下一个 记录20 上是增加**Next-Key Lock**（*与Purge有关，后续会讲*）
3. 在**2.8**的例子中，查询条件（ `b = 3 for update` ）是 二级索引 查询，所以在 记录3 的下一个 记录6 上增加的是**Gap Lock**
4. 简而言之，主键查询加**Next-Key Lock**，而二级索引查询加**Gap Lock**