mysql insert 加锁流程

INSERT sets an exclusive lock on the inserted row. This lock is an index-record lock, not a next-key lock (that is, there is no gap lock) and does not prevent other sessions from inserting into the gap before the inserted row.

insert 会对插入成功的行加上排他锁。这个锁不是索引记录锁，也不是next-key lock（更不是gap lock），不会阻止其他并发事务往这条记录之前插入记录。

Prior to inserting the row, a type of gap lock called an insert intention gap lock is set. This lock signals the intent to insert in such a way that multiple transactions inserting into the same index gap need not wait for each other if they are not inserting at the same position within the gap. Suppose that there are index records with values of 4 and 7. Separate transactions that attempt to insert values of 5 and 6 each lock the gap between 4 and 7 with insert intention locks prior to obtaining the exclusive lock on the inserted row, but do not block each other because the rows are nonconflicting.

在插入之前，设置了一个叫做insert intention gap lock（插入意向间隙锁）的间隙锁类型。此锁表示插入这样的意图：即插入到同一索引间隙中的多个事务如果不在间隙内的同一位置插入，则不需要彼此等待。（个人理解，insert intention gap lock 不等于其他gap lock。不会阻塞）。假设有索引记录的值为4和7。当拥有4和7的这个insert intention locks在获得x lock之前，两个试图插入5和6的事务不会相互阻塞对方，锁不冲突。

If a duplicate-key error occurs, a shared lock on the duplicate index record is set. This use of a shared lock can result in deadlock should there be multiple sessions trying to insert the same row if another session already has an exclusive lock. This can occur if another session deletes the row. Suppose that an InnoDB table t1 has the following structure:

如果一个重复主键发生，那么重复键设置在重复索引记录上。（如果多事务insert 插入导致了duplicate-key，那么索引记录被设置为 shared lock 锁）。在多并发情况下,如果一个session 已经有了x-lock，其他并发事务插

入同一行记录，那么这个记录上的s-lock将导致死锁。 如下面例子

```
CREATE TABLE t1 (i INT, PRIMARY KEY (i)) ENGINE = InnoDB;
```

Now suppose that three sessions perform the following operations in order:

Session 1:

```
START TRANSACTION;
INSERT INTO t1 VALUES(1);
```

Session 2:

```
START TRANSACTION;
INSERT INTO t1 VALUES(1);
```

Session 3:

```
START TRANSACTION;
INSERT INTO t1 VALUES(1);
```

Session 1:

```
ROLLBACK;
```

The first operation by session 1 acquires an exclusive lock for the row. The operations by sessions 2 and 3 both result in a duplicate-key error and they both request a shared lock for the row. When session 1 rolls back, it releases its exclusive lock on the row and the queued shared lock requests for sessions 2 and 3 are granted. At this point, sessions 2 and 3 deadlock: Neither can acquire an exclusive lock for the row because of the shared lock held by the other.

A similar situation occurs if the table already contains a row with key value

1 and three sessions perform the following operations in order:

Session 1:

```
START TRANSACTION;
DELETE FROM t1 WHERE i = 1;
```

Session 2:

```
START TRANSACTION;
INSERT INTO t1 VALUES(1);
```

Session 3:

```
START TRANSACTION;
INSERT INTO t1 VALUES(1);
```

Session 1:

```
COMMIT;
```

The first operation by session 1 acquires an exclusive lock for the row. The operations by sessions 2 and 3 both result in a duplicate-key error and they both request a shared lock for the row. When session 1 commits, it releases its exclusive lock on the row and the queued shared lock requests for sessions 2 and 3 are granted. At this point, sessions 2 and 3 deadlock: Neither can acquire an exclusive lock for the row because of the shared lock held by the other.

上面的回话那个报出dead lock，由mysql本身调控。

https://dev.mysql.com/doc/refman/5.5/en/innodb-locks-set.html