字节面试：加了什么锁，导致死锁的?

大家好，我是小林。

之前收到读者面试字节时，被问到一个关于 MySQL 的问题。

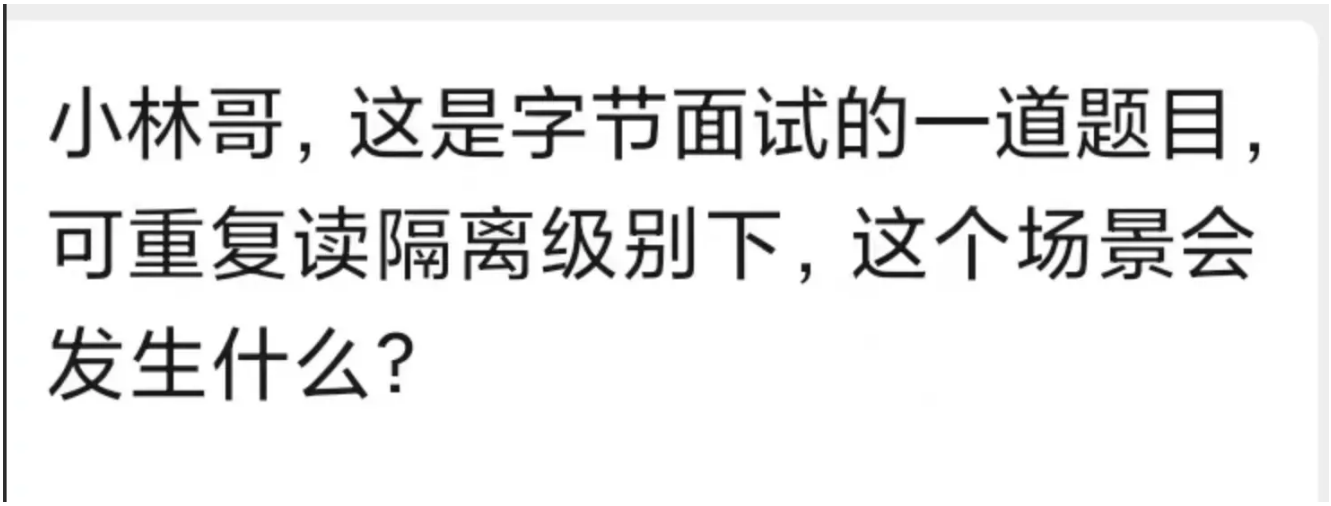
题目描述

(id, no, name, age, score)

(15, S0001, Bob, 25, 34)
(18, S0002, Alice, 24, 77)
(20, S0003, Jim, 24, 5)
(30, S0004, Eric, 23, 91)
(37, S0005, Tom, 22, 22)
(49, S0006, Tom, 25, 83)
(50, S0007, Rose, 23, 89)

事务A执行:
time1: update students set score=100 where id=25
time3: insert into students(id, no, name, age, score) value(25, 's0025', 'sony', 28,90)

事务B执行:
time2: update students set score=100 where id=26
time4: insert into students(id, no, name, age, score) value(26, 's0026', 'ace', 28,90)



如果对 MySQL 加锁机制比较熟悉的同学，应该一眼就能看出**会发生死锁**，但是具体加了什么锁而导致死锁，是需要我们具体分析的。

接下来，就跟聊聊上面两个事务执行 SQL 语句的过程中，加了什么锁，从而导致死锁的。

准备工作

先创建一张 t_student 表，假设除了 id 字段，其他字段都是普通字段。

```
CREATE TABLE `t_student` (
  `id` int NOT NULL,
  `no` varchar(255) DEFAULT NULL,
  `name` varchar(255) DEFAULT NULL,
  `age` int DEFAULT NULL,
  `score` int DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

然后，插入相关的数据后，t_student 表中的记录如下：

id	no	name	age	score
15	S0001	Bod	25	24
18	S0002	Alice	24	77
20	S0003	Jim	24	5
30	S0004	Eric	23	91
37	S0005	Tom	22	22
49	S0006	Tom	25	83
50	S0007	Rose	23	89

开始实验

在实验开始前，先说明下实验环境：

- MySQL 版本：8.0.26
- 隔离级别：可重复读（RR）

启动两个事务，按照题目的 SQL 执行顺序，过程如下表格：

	事务 A	事务 B
	begin	begin
Time 1 阶段	update t_student set score = 100 where **id = 25**;	
Time 2 阶段		update t_student set score = 100 where **id = 26**;
Time 3 阶段	insert into t_student(id, no, name, age,score) value (**25**, 'S0025', 'sony', 28, 90) 阻塞等待.....	
Time 4 阶段		insert into t_student(id, no, name, age,score) value (**26**, 'S0026', 'ace', 28, 90) 阻塞等待.....

可以看到，事务 A 和 事务 B 都在执行 insert 语句后，都陷入了等待状态（前提没有打开死锁检测），也就是发生了死锁，因为都在相互等待对方释放锁。

为什么会发生死锁?

我们可以通过 `select * from performance_schema.data_locks\G;` 这条语句，查看事务执行 SQL 过程中加了什么锁。

接下来，针对每一条 SQL 语句分析具体加了什么锁。

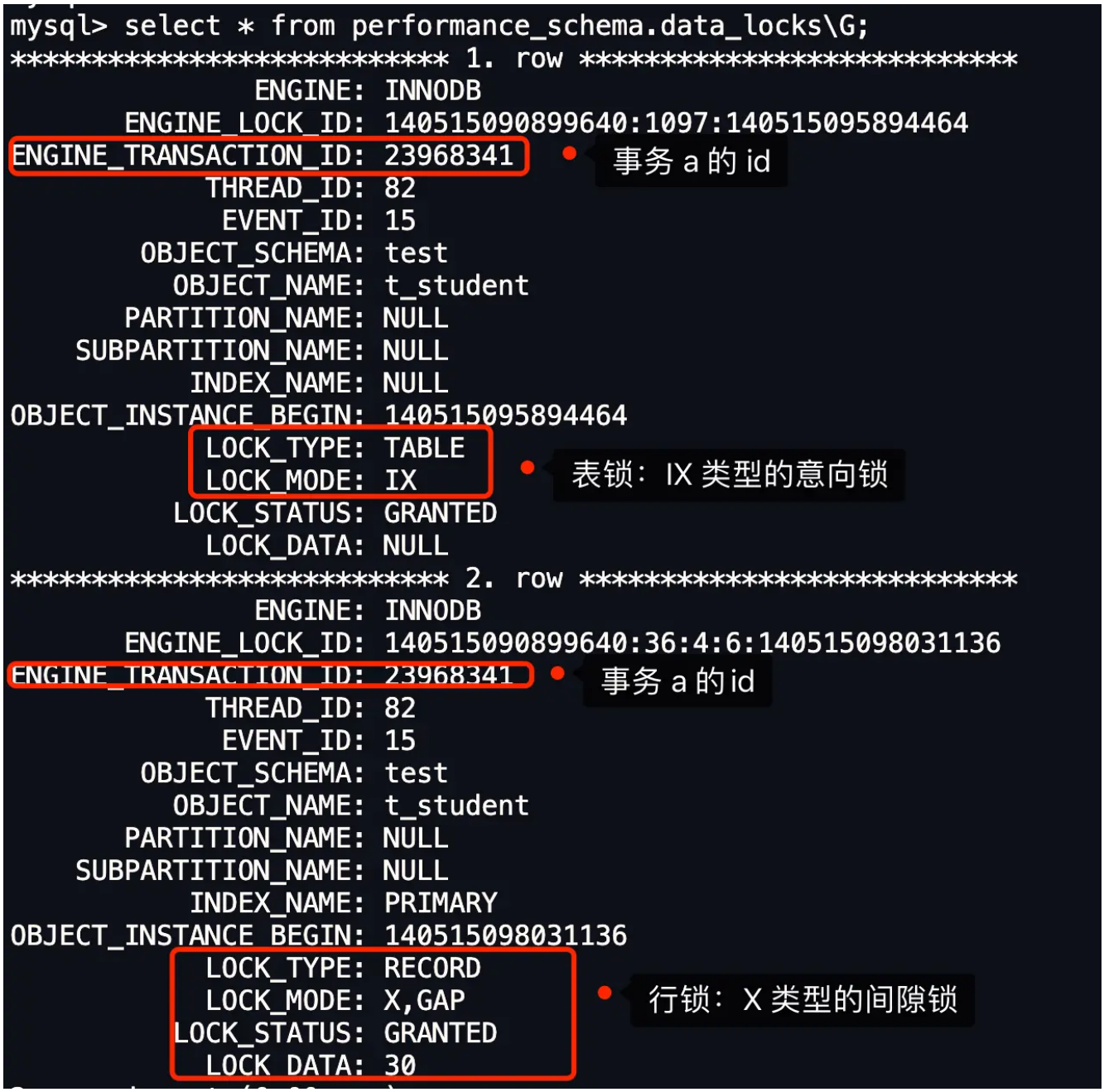
Time 1 阶段加锁分析

Time 1 阶段，事务 A 执行以下语句：

```
# 事务 A
mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> update t_student set score = 100 where id = 25;
Query OK, 0 rows affected (0.01 sec)
Rows matched: 0  Changed: 0  Warnings: 0
```

然后执行 `select * from performance_schema.data_locks\G;` 这条语句，查看事务 A 此时加了什么锁。



从上图可以看到，共加了两个锁，分别是：

- 表锁：X 类型的意向锁；
- 行锁：X 类型的间隙锁；

这里我们重点关注行锁，图中 LOCK_TYPE 中的 RECORD 表示行级锁，而不是记录锁的意思，通过 LOCK_MODE 可以确认是 next-key 锁，还是间隙锁，还是记录锁：

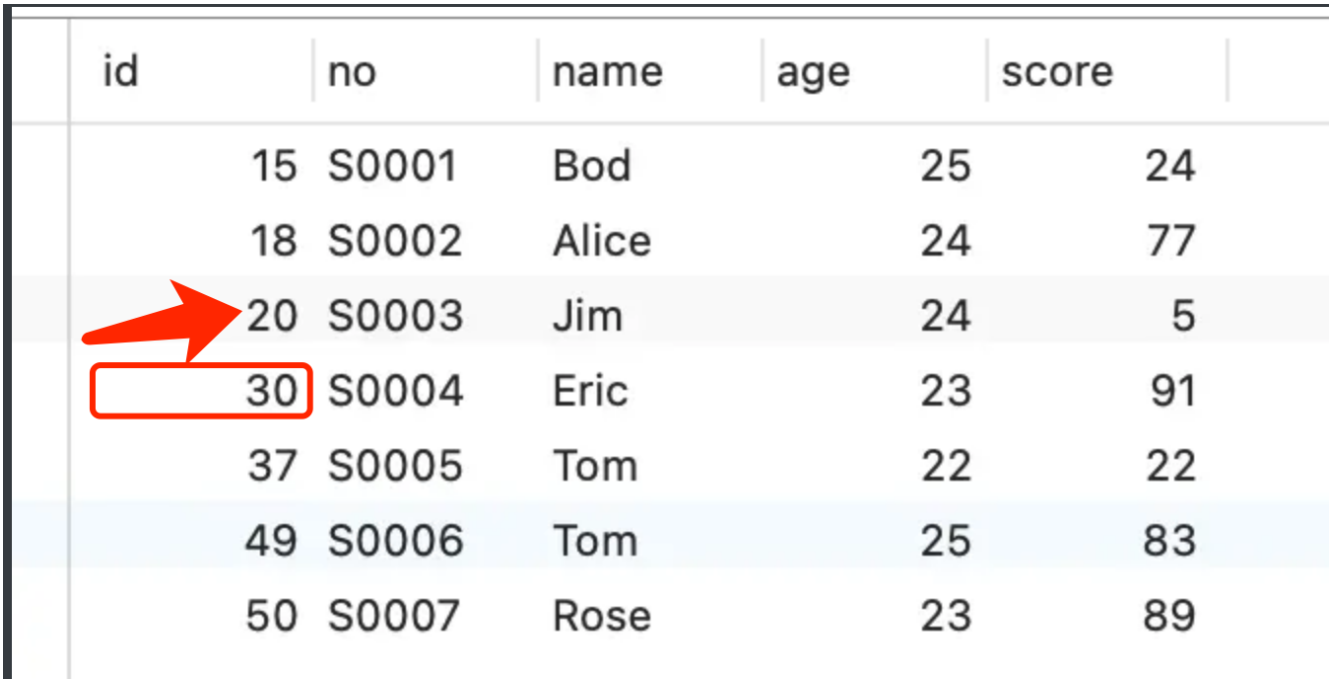
- 如果 LOCK_MODE 为 `X` ，说明是 next-key 锁；
- 如果 LOCK_MODE 为 `X, REC_NOT_GAP` ，说明是记录锁；
- 如果 LOCK_MODE 为 `X, GAP` ，说明是间隙锁；

因此，此时事务 A 在主键索引（INDEX_NAME : PRIMARY）上加的是间隙锁，锁范围是 `(20, 30)` 。

> 间隙锁的范围 `(20, 30)` ，是怎么确定的?

根据我的经验，如果 LOCK_MODE 是 next-key 锁或者间隙锁，那么 LOCK_DATA 就表示锁的范围最右值，此次的事务 A 的 LOCK_DATA 是 30。

然后锁范围的最左值是 t_student 表中 id 为 30 的上一条记录的 id 值，即 20。



id	no	name	age	score
15	S0001	Bod	25	24
18	S0002	Alice	24	77
20	S0003	Jim	24	5
30	S0004	Eric	23	91
37	S0005	Tom	22	22
49	S0006	Tom	25	83
50	S0007	Rose	23	89

因此，间隙锁的范围 `(20, 30)` 。

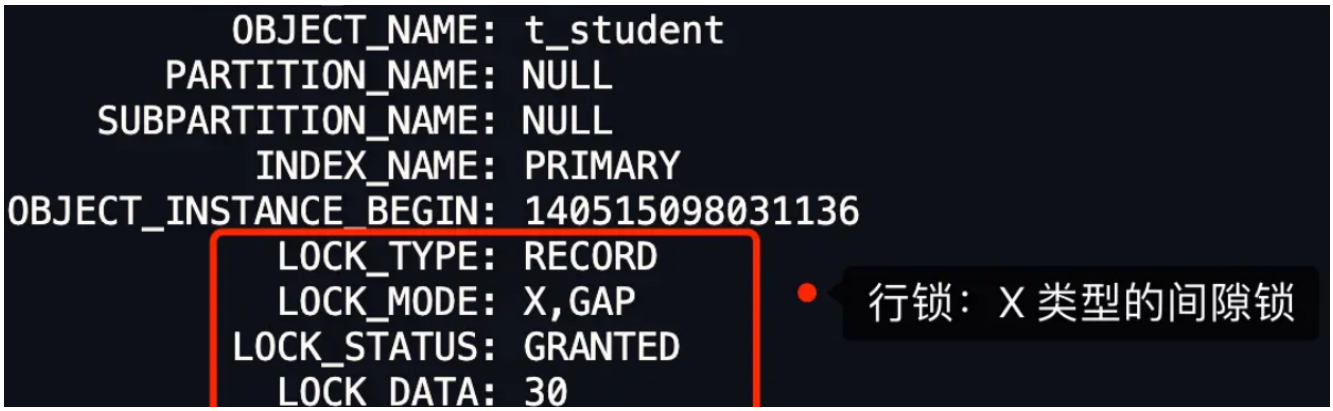
Time 2 阶段加锁分析

Time 2 阶段，事务 B 执行以下语句：

```
# 事务 B
mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> update t_student set score = 100 where id = 26;
Query OK, 0 rows affected (0.01 sec)
Rows matched: 0  Changed: 0  Warnings: 0
```

然后执行 `select * from performance_schema.data_locks\G;` 这条语句，查看事务 B 此时加了什么锁。



从上图可以看到，行锁是 X 类型的间隙锁，间隙锁的范围是 `(20, 30)` 。

> 事务 A 和 事务 B 的间隙锁范围都是一样的，为什么不会冲突?

两个事务的间隙锁之间是相互兼容的，不会产生冲突。

在MySQL官网上还有一段非常关键的描述：

Gap locks in InnoDB are “purely inhibitive”, which means that their only purpose is to prevent other transactions from Inserting to the gap. Gap locks can co-exist. A gap lock taken by one transaction does not prevent another transaction from taking a gap lock on the same gap. There is no difference between shared and exclusive gap locks. They do not conflict with each other, and they perform the same function.

间隙锁的意义只在于阻止区间被插入，因此是可以共存的。**一个事务获取的间隙锁不会阻止另一个事务获取同一个间隙范围的间隙锁**，共享（S型）和排他（X型）的间隙锁是没有区别的，他们相互不冲突，且功能相同。

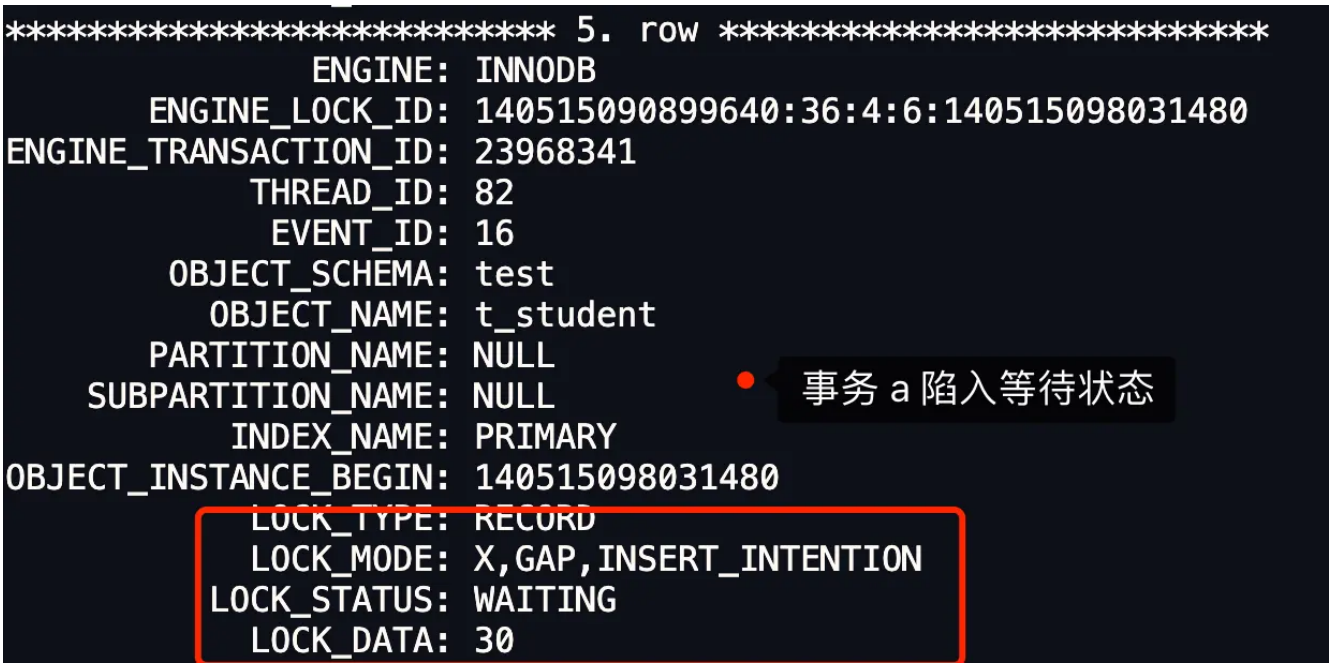
Time 3 阶段加锁分析

Time 3，事务 A 插入了一条记录：

```
# Time 3 阶段，事务 A 插入了一条记录
mysql> insert into t_student(id, no, name, age,score) value (25, 'S0025', 'son
    /// 阻塞等待.....
```

此时，事务 A 就陷入了等待状态。

然后执行 `select * from performance_schema.data_locks\G;` 这条语句，查看事务 A 在获取什么锁而导致被阻塞。



可以看到，事务 A 的状态为等待状态（LOCK_STATUS: WAITING），因为向事务 B 生成的间隙锁（范围 `(20, 30)`）中插入了一条记录，所以事务 A 的插入操作生成了一个插入意向锁（`LOCK_MODE:INSERT_INTENTION`）。

> 插入意向锁是什么?

注意！插入意向锁名字里虽然有意向锁这三个字，但是它并不是意向锁，它属于行级锁，是一种特殊的间隙锁。

在MySQL的官方文档中有以下重要描述：

An Insert intention lock is a type of gap lock set by Insert operations prior to row Insertion. 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, respectively, 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.

这段话表明尽管**插入意向锁是一种特殊的间隙锁，但不同于间隙锁的是，该锁只用于并发插入操作**。

如果说间隙锁锁住的是一个区间，那么「插入意向锁」锁住的就是一个点。因而从这个角度来说，插入意向锁确实是一种特殊的间隙锁。

插入意向锁与间隙锁的另一个非常重要的差别是：**尽管「插入意向锁」也属于间隙锁，但两个事务却不能在同一时间内，一个拥有间隙锁，另一个拥有该间隙区间内的插入意向锁（当然，插入意向锁如果不在间隙锁区间内则是可以的）。所以，插入意向锁和间隙锁之间是冲突的**。

另外，我补充一点，插入意向锁的生成时机：

- 每插入一条新记录，都需要看一下待插入记录的下一条记录上是否已经被加了间隙锁，如果已加间隙锁，此时会生成一个插入意向锁，然后锁的状态设置为等待状态（*PS：MySQL 加锁时，是先生成锁结构，然后设置锁的状态，如果锁状态是等待状态，并不是意味着事务成功获取到了锁，只有当锁状态为正常状态时，才代表事务成功获取到了锁*），现象就是 Insert 语句会被阻塞。

Time 4 阶段加锁分析

Time 4，事务 B 插入了一条记录：

```
# Time 4 阶段，事务 B 插入了一条记录
mysql> insert into t_student(id, no, name, age,score) value (26, 'S0026', 'ace
    /// 阻塞等待.....
```

此时，事务 B 就陷入了等待状态。

然后执行 `select * from performance_schema.data_locks\G;` 这条语句，查看事务 B 在获取什么锁而导致被阻塞。

可以看到，事务 B 在生成插入意向锁时而导致被阻塞，这是因为事务 B 向事务 A 生成的范围为 (20, 30) 的间隙锁插入了一条记录，而插入意向锁和间隙锁是冲突的，所以事务 B 在获取插入意向锁时就陷入了等待状态。

> 最后回答，为什么会发生死锁?

本次案例中，事务 A 和事务 B 在执行完后 update 语句后都持有范围为 `(20, 30)` 的间隙锁，而接下来的插入操作为了获取到插入意向锁，都在等待对方事务的间隙锁释放，于是就造成了循环等待，满足了死锁的四个条件：**互斥、占有且等待、不可强占用、循环等待**，因此发生了死锁。

总结

两个事务即使生成的间隙锁的范围是一样的，也不会发生冲突，因为间隙锁目的是为了防止其他事务插入数据，因此间隙锁与间隙锁之间是相互兼容的。

在执行插入语句时，如果插入的记录在其他事务持有间隙锁范围内，插入语句就会被阻塞，因为插入语句在碰到间隙锁时，会生成一个插入意向锁，然后插入意向锁和间隙锁之间是互斥的关系。

如果两个事务分别向对方持有的间隙锁范围内插入一条记录，而插入操作为了获取到插入意向锁，都在等待对方事务的间隙锁释放，于是就造成了循环等待，满足了死锁的四个条件：**互斥、占有且等待、不可强占用、循环等待**，因此发生了死锁。

读者问答

快乐星球(朋友) 来自山东 置顶 ··· 1

林，对于间隙锁的兼容，我动手测试了一下，发现，只有间隙内为空，这两个间隙锁才兼容互相不阻塞，如果间隙锁内有记录，则两个间隙锁是冲突的，后一个阻塞。本例如 select id>15 and id< 50 for update 两个事务都执行这句，后一个事务就阻塞了，我不清楚这算不算间隙锁，请指教，谢谢。

小林coding(作者)

我测试你说的例子，你这条语句加的是next-key lock，包含记录锁+间隙锁，之所以后一个事务被阻塞，是因为 for update 加的是 x 型的锁，然后 x 型 的记录锁和 x 型的记录锁是冲突的。

小林coding(作者)

就好像两个事务执行 select id = 15 for update ，后一个事务会被阻塞，原因是 x 型 的记录锁和 x 型的记录锁是冲突的。

小林coding(作者)

但是对于单纯的间隙锁，两个事务之间的间隙锁是兼容的，就像本文说的那样。而记录锁的话，要考虑x型和s型的冲突与兼容关系。

快乐星球 来自山东

明白了 感谢 我对间隙锁的理解错误，原来单纯的间隙锁是不包含元素的，感谢解答。