超全面MySQL语句加锁分析（中篇）（求转）

原创 小孩子4919 我们都是小青蛙 2019-05-09

说在前面的话

本文是用来系统阐述在MySQL中，不同语句在各种条件下的加锁情况，并不是解释各种锁是什么（或者说加锁的本质是什么），大家如果不理解什么是 `MVCC` 、 `ReadView` 、 `正经记录锁` 、 `gap锁` 、 `next-key锁` 、 `插入意向锁` 这些概念的，可以参考 `MySQL` 的官方文档，或者直接参照《MySQL是怎样运行的：从根儿上理解MySQL》这本小册（里边有比官方文档更贴心，更详细的解释，文章中涉及到的所有概念均在小册中有详细解释，有疑惑，并且有兴趣的同学可以扫描下边二维码看看）：

建议：

1. 本篇文章不适合碎片化时间阅读，最好使用电脑观看，或者将字体跳到最小效果好一些
2. 可能一下子看不完，关注 + 收藏 + 好看 + 转发一波
3. 不要跳着看
4. 一定要先看过上一篇文章：超全面MySQL语句加锁分析（上篇）（求转）

REPEATABLE READ隔离级别下

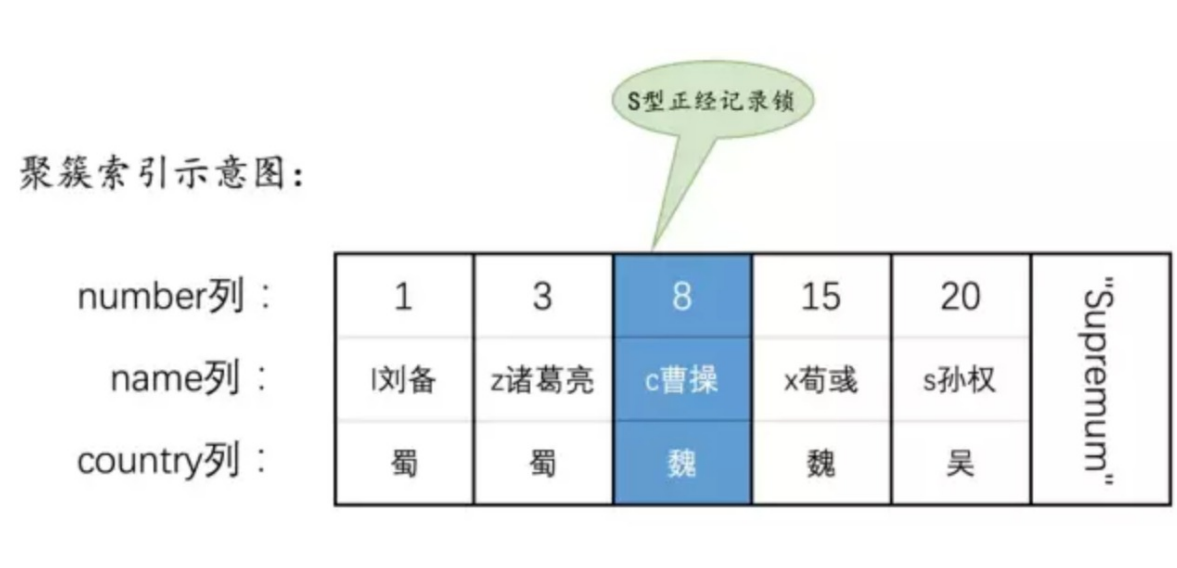
采用 `加锁` 的方式解决并发事务产生的问题时， `REPEATABLE READ` 隔离级别与 `READ UNCOMMITTED` 和 `READ COMMITTED` 这两个隔离级别相比，最主要的就是要解决 `幻读` 问题， `幻读` 问题的解决还得靠我们之前讲过的 `gap锁` 。

对于使用主键进行等值查询的情况

- 使用 `SELECT ... LOCK IN SHARE MODE` 来为记录加锁，比方说：

```
SELECT * FROM hero WHERE number = 8 LOCK IN SHARE MODE;
```

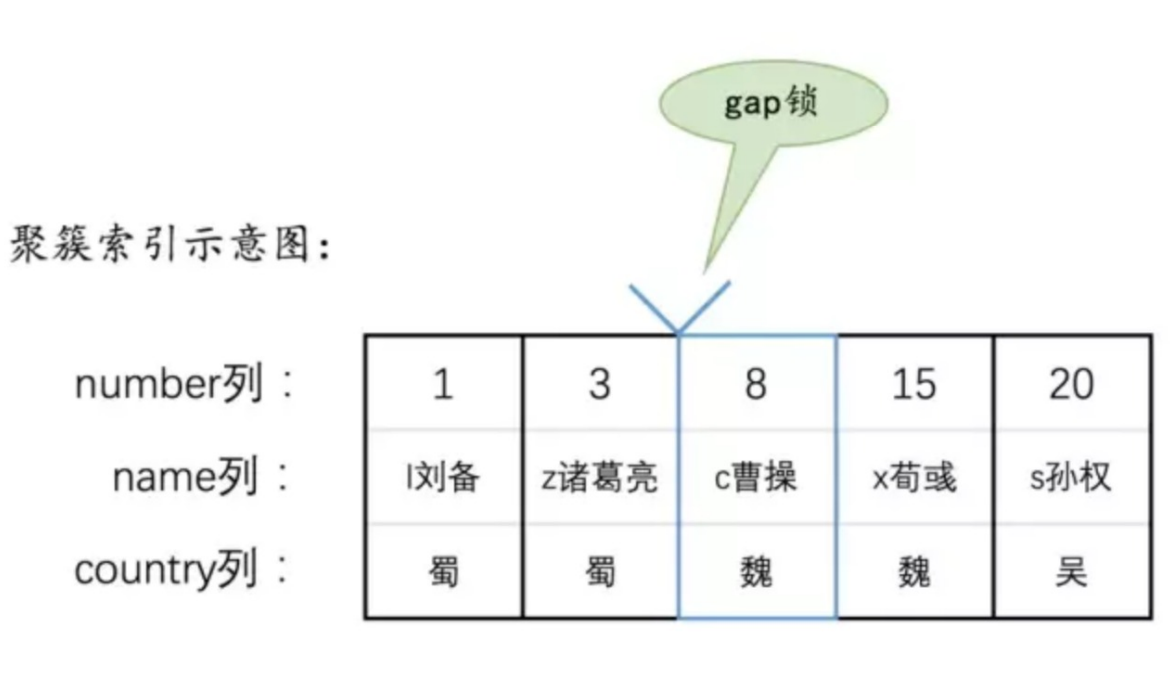
我们知道主键具有唯一性，如果在一个事务中第一次执行上述语句时将得到的结果集中包含一条记录，第二次执行上述语句前肯定不会有别的事务插入多条 `number` 值为 `8` 的记录（主键具有唯一性），也就是说一个事务中两次执行上述语句并不会发生幻读，这种情况下和 `READ UNCOMMITTED / READ COMMITTED` 隔离级别下一样，我们只需要为这条 `number` 值为 `8` 的记录加一个 `S型正经记录锁` 就好了，如图所示：


S型正经记录锁
聚簇索引示意图：
number列：
name列：
country列：
1 3 8 15 20
l刘备 z诸葛亮 c曹操 x荀彧 s孙权
蜀 蜀 魏 魏 吴
"Supremum"

但是如果我们要查询主键值不存在的记录，比方说：

```
SELECT * FROM hero WHERE number = 7 LOCK IN SHARE MODE;
```

由于 `number` 值为 `7` 的记录不存在，为了禁止 `幻读` 现象（也就是避免在同一事务中下一次执行相同语句时得到的结果集中包含 `number` 值为 `7` 的记录），在当前事务提交前我们需要预防别的事务插入 `number` 值为 `7` 的新记录，所以需要在 `number` 值为 `8` 的记录上加一个 `gap锁` ，也就是不允许别的事务插入 `number` 值在 `(3, 8)` 这个区间的新记录。画个图表示一下：


gap锁
聚簇索引示意图：
number列：
name列：
country列：
1 3 8 15 20
l刘备 z诸葛亮 c曹操 x荀彧 s孙权
蜀 蜀 魏 魏 吴

如果在 `READ UNCOMMITTED / READ COMMITTED` 隔离级别下一样查询了一条主键值不存在的记录，那么什么锁也不需要加，因为在 `READ UNCOMMITTED / READ COMMITTED` 隔离级别下，并不需要禁止 `幻读` 问题。

- 其余语句使用主键进行等值查询的情况与 `READ UNCOMMITTED / READ COMMITTED` 隔离级别下的情况类似，这里就不赘述了。

对于使用主键进行范围查询的情况

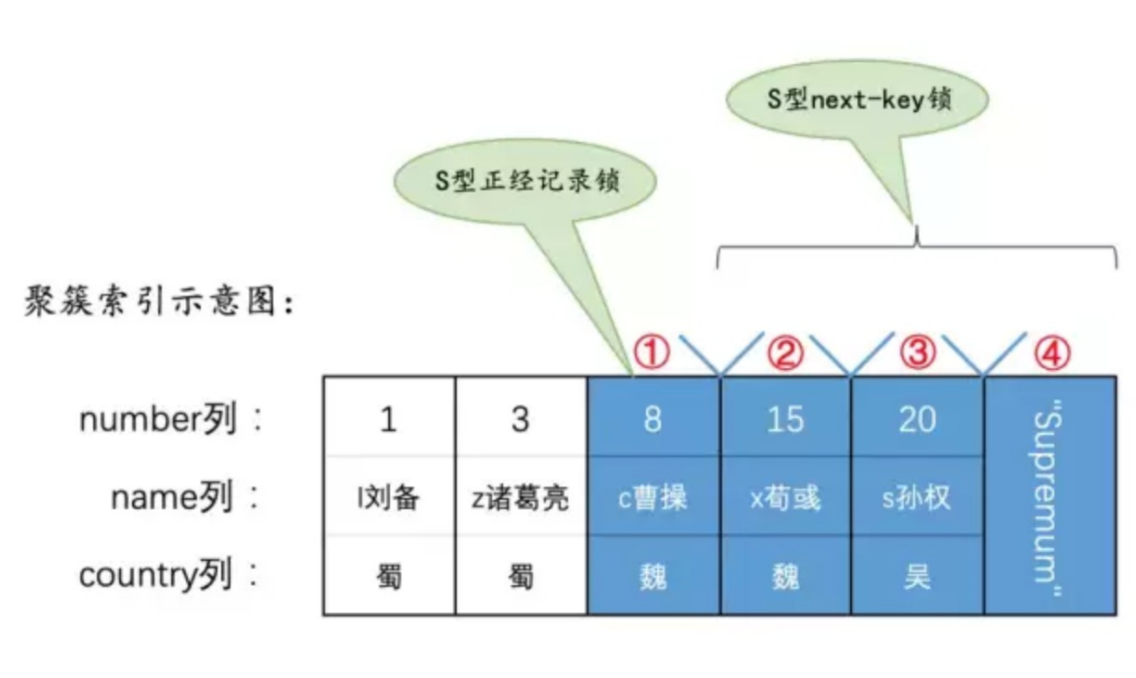
- 使用 `SELECT ... LOCK IN SHARE MODE` 语句来为记录加锁，比方说：

```
SELECT * FROM hero WHERE number >= 8 LOCK IN SHARE MODE;
```

因为要解决幻读问题，所以需要禁止别的事务插入 `number` 值符合 `number >= 8` 的记录，又因为主键本身就是唯一的，所以我们不用担心在 `number` 值为 `8` 的前边有新记录插入，只需要保证不要让新记录插入到 `number` 值为 `8` 的后边就好了，所以：

  - 为 `number` 值为 `8` 的聚簇索引记录加一个 `S型正经记录锁` 。
  - 为 `number` 值大于 `8` 的所有聚簇索引记录都加一个 `S型next-key锁` （包括 `Supremum` 伪记录）。

画个图就是这样子：

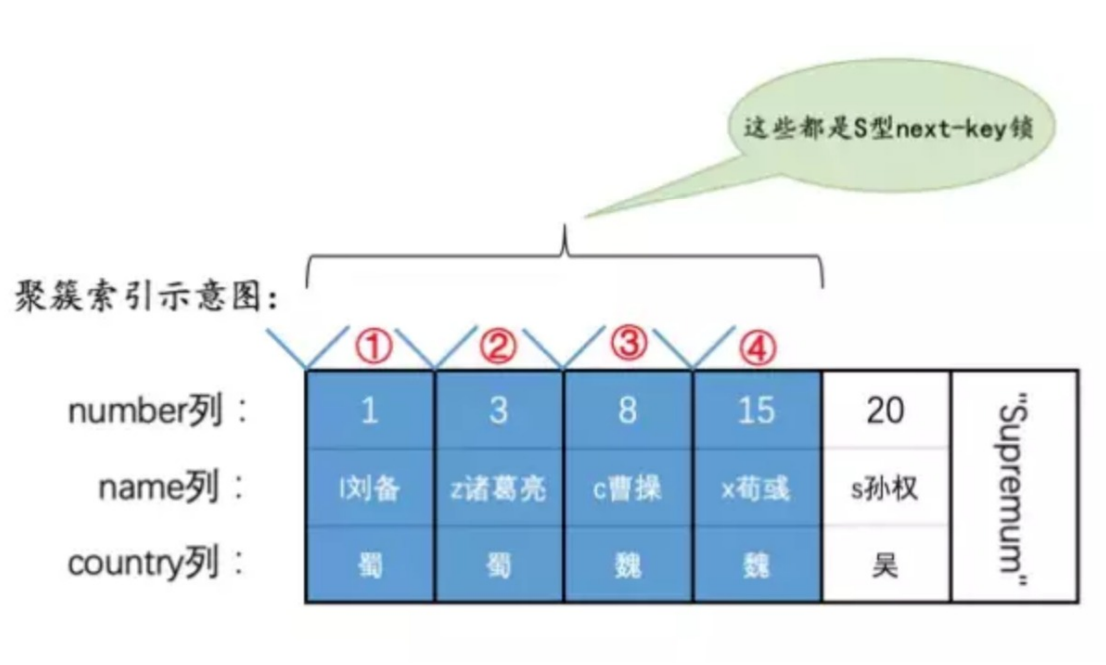

S型next-key锁
S型正经记录锁
聚簇索引示意图：
① ② ③ ④
number列：
name列：
country列：
1 3 8 15 20
l刘备 z诸葛亮 c曹操 x荀彧 s孙权
蜀 蜀 魏 魏 吴
"Supremum"

> 小贴士：为什么不给Supremum记录加gap锁，而要加next-key锁呢？其实设计InnoDB的大叔在处理Supremum记录上加的next-key锁时就是当作gap锁看待的，只不过为了节省锁结构（我们前边说锁的类型不一样的话不能被放到一个锁结构中）才这么做的而已，大家不必在意。

与 `READ UNCOMMITTED/READ COMMITTED` 隔离级别类似，在 `REPEATABLE READ` 隔离级别下，下边这个范围查询也是有点特殊：

```
SELECT * FROM hero WHERE number <= 8 LOCK IN SHARE MODE;
```

这个语句的执行过程我们在之前唠叨过，在 `READ UNCOMMITTED/READ COMMITTED` 隔离级别下，这个语句会为 `number` 值为 `1` 、 `3` 、 `8` 、 `15` 这4条记录都加上 `S型正经记录锁` ，然后由于 `number` 值为 `15` 的记录不满足边界条件 `number <= 8` ，随后便把这条记录的锁释放掉。在 `REPEATABLE READ` 隔离级别下的加锁过程与之类似，不过会为 `1` 、 `3` 、 `8` 、 `15` 这4条记录都加上 `S型next-key锁` ，但是有一点需要大家十分注意：REPEATABLE READ隔离级别下，在判断number值为15的记录不满足边界条件 number <= 8 后，并不会去释放加在该记录上的锁！！！ 所以在 `REPEATABLE READ` 隔离级别下，该语句的加锁示意图就如下所示：


这些都是S型next-key锁
聚簇索引示意图：
① ② ③ ④
number列：
name列：
country列：
1 3 8 15 20
l刘备 z诸葛亮 c曹操 x荀彧 s孙权
蜀 蜀 魏 魏 吴
"Supremum"

这样如果别的事务想要插入的新记录的 `number` 值在 `(-∞, 1)` 、 `(1, 3)` 、 `(3, 8)` 、 `(8, 15)` 之间的话，是会进入等待状态的。

> 小贴士：很显然这么粗暴的做法导致的一个后果就是别的事务竟然不允许插入number值在(8, 15)这个区间中的新记录，甚至不允许别的事务再获取number值为15的记录上的锁，而理论上只需要禁止别的事务插入number值在(-∞, 8)之间的新记录就好。

- 使用 `SELECT ... FOR UPDATE` 语句来为记录加锁：

和 `SELECT ... LOCK IN SHARE MODE` 语句类似，只不过需要将上边提到的 `S型next-key锁` 替换成 `X型next-key锁` 。

- 使用 `UPDATE ...` 来为记录加锁：

如果 `UPDATE` 语句未更新二级索引列，比方说：

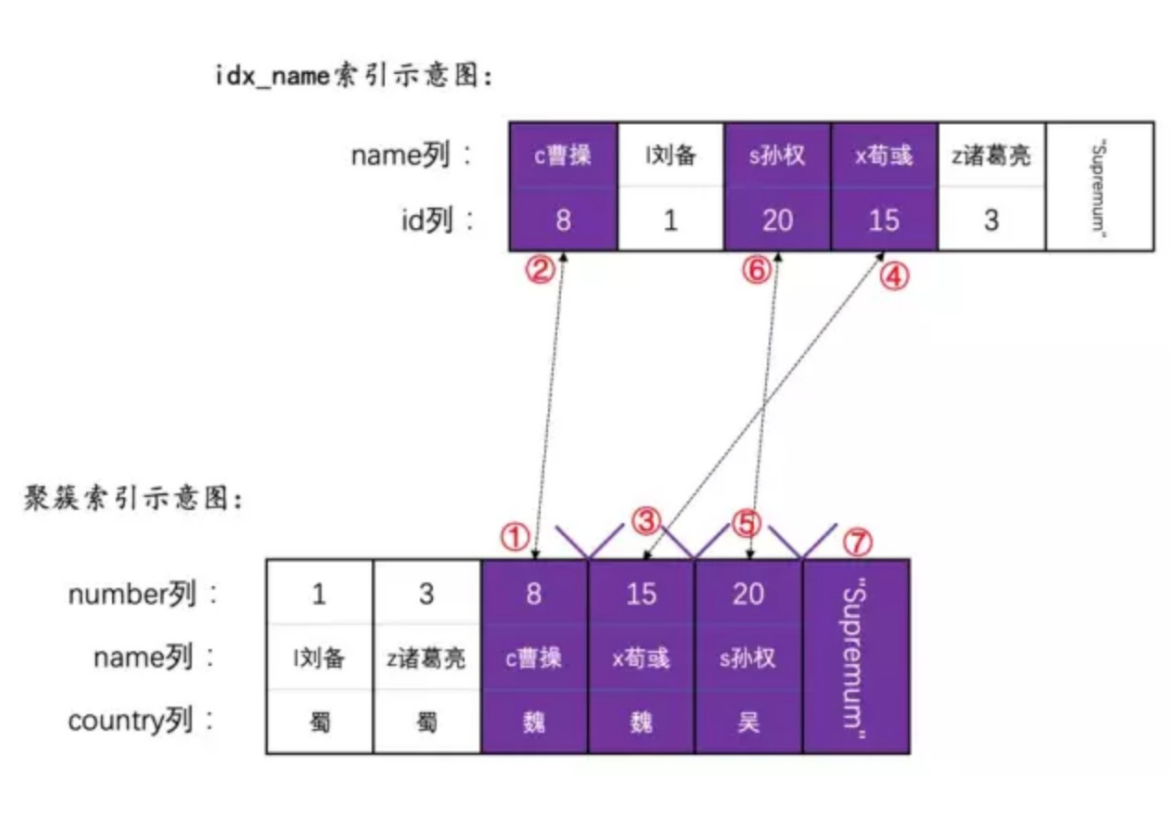
```
UPDATE hero SET country = '汉' WHERE number >= 8;
```

这条 `UPDATE` 语句并没有更新二级索引列，加锁方式和上边所说的 `SELECT ... FOR UPDATE` 语句一致。

如果 `UPDATE` 语句中更新了二级索引列，比方说：

```
UPDATE hero SET name = 'cao曹操' WHERE number >= 8;
```

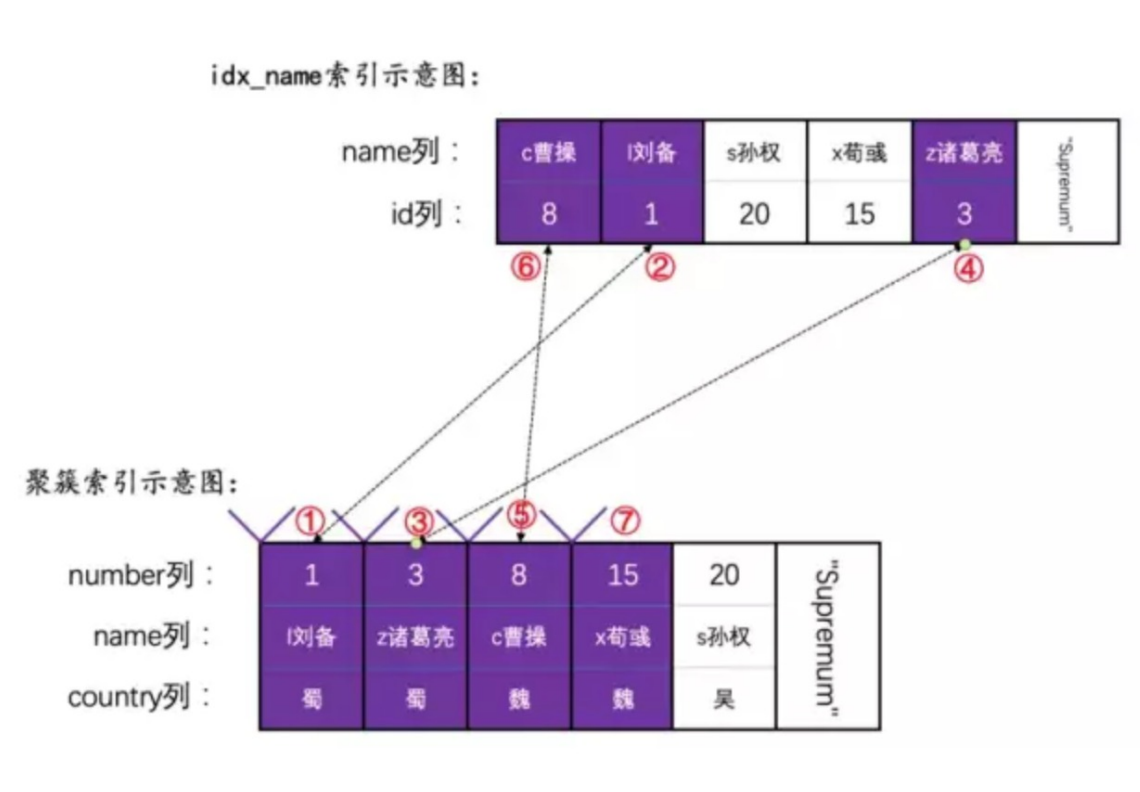
对聚簇索引记录加锁的情况和 `SELECT ... FOR UPDATE` 语句一致，也就是对 `number` 值为 `8` 的聚簇索引记录加 `X型正经记录锁` ，对 `number` 值 `15` 、 `20` 的聚簇索引记录以及 `Supremum` 记录加 `X型next-key锁` 。但是因为也要更新二级索引 `idx_name` ，所以也会对 `number` 值为 `8` 、 `15` 、 `20` 的聚簇索引记录对应的 `idx_name` 二级索引记录加 `X型正经记录锁` ，画个图表示一下：


idx_name索引示意图：
name列：
id列：
c曹操 l刘备 s孙权 x荀彧 z诸葛亮
8 1 20 15 3
"Supremum"
② ⑥ ④
聚簇索引示意图：
① ③ ⑤ ⑦
number列：
name列：
country列：
1 3 8 15 20
l刘备 z诸葛亮 c曹操 x荀彧 s孙权
蜀 蜀 魏 魏 吴
"Supremum"

如果是下边这个语句：

```
UPDATE hero SET name = 'cao曹操' WHERE number <= 8;
```

则会对 `number` 值为 `1` 、 `3` 、 `8` 、 `15` 的聚簇索引记录加 `X型next-key` ，其中 `number` 值为 `15` 的聚簇索引记录不满足 `number <= 8` 的边界条件，虽然在 `REPEATABLE READ` 隔离级别下不会将它的锁释放掉，但是也并不会对这条聚簇索引记录对应的二级索引记录加锁，也就是说只会为 `number` 值为 `1` 、 `3` 、 `8` 的聚簇索引记录对应的 `idx_name` 二级索引记录加 `X型正经记录锁` ，加锁示意图如下所示：


idx_name索引示意图：
name列：
id列：
c曹操 l刘备 s孙权 x荀彧 z诸葛亮
8 1 20 15 3
"Supremum"
⑥ ② ④
聚簇索引示意图：
① ③ ⑤ ⑦
number列：
name列：
country列：
1 3 8 15 20
l刘备 z诸葛亮 c曹操 x荀彧 s孙权
蜀 蜀 魏 魏 吴
"Supremum"

- 使用 `DELETE ...` 来为记录加锁，比方说：

```
DELETE FROM hero WHERE number >= 8;
```

和

```
DELETE FROM hero WHERE number <= 8;
```

这两个语句的加锁情况和更新带有二级索引列的 `UPDATE` 语句一致，就不画图了。

对于使用唯一二级索引进行等值查询的情况

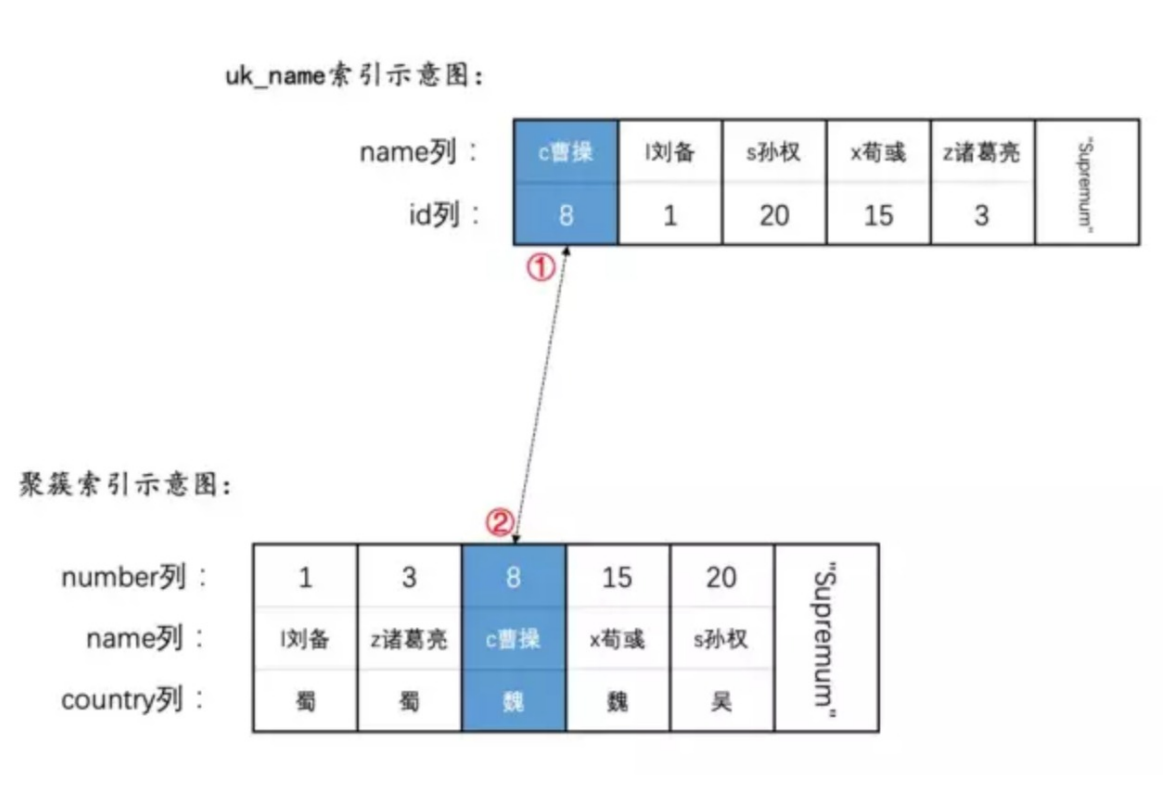
由于 `hero` 表并没有唯一二级索引，我们把原先的 `idx_name` 修改为一个唯一二级索引 `uk_name` ：

```
ALTER TABLE hero DROP INDEX idx_name, ADD UNIQUE KEY uk_name (name);
```

- 使用 `SELECT ... LOCK IN SHARE MODE` 语句来为记录加锁，比方说：

```
SELECT * FROM hero WHERE name = 'c曹操' LOCK IN SHARE MODE;
```

由于唯一二级索引具有唯一性，如果在一个事务中第一次执行上述语句时将得到一条记录，第二次执行上述语句前肯定不会有别的事务插入多条 `name` 值为 `'c曹操'` 的记录（二级索引具有唯一性），也就是说一个事务中两次执行上述语句并不会发生幻读，这种情况下和 `READ UNCOMMITTED / READ COMMITTED` 隔离级别下一样，我们只需要为这条 `name` 值为 `'c曹操'` 的二级索引记录加一个 `S型正经记录锁` ，然后再为它对应的聚簇索引记录加一个 `S型正经记录锁` 就好了，我们画个图看看：

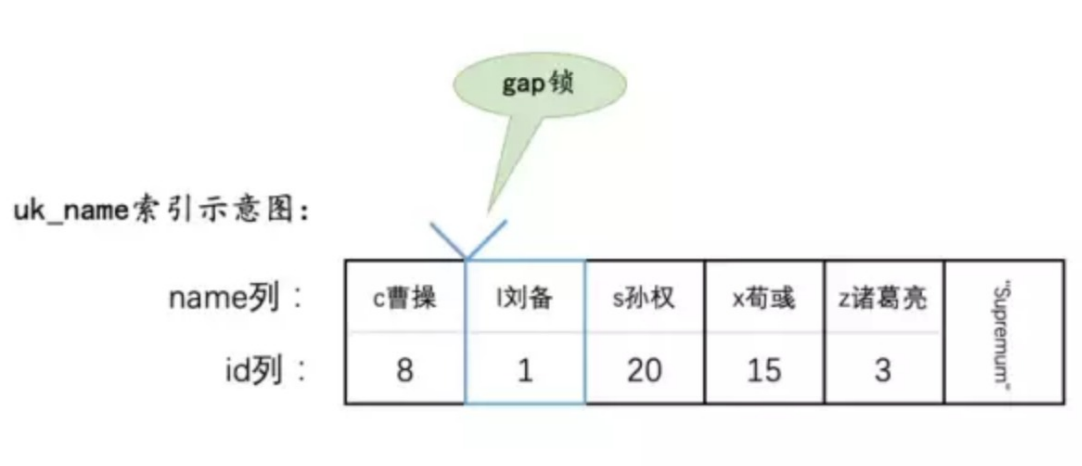

uk_name索引示意图：
name列：
id列：
c曹操 l刘备 s孙权 x荀彧 z诸葛亮
8 1 20 15 3
"Supremum"
①
聚簇索引示意图：
②
number列：
name列：
country列：
1 3 8 15 20
l刘备 z诸葛亮 c曹操 x荀彧 s孙权
蜀 蜀 魏 魏 吴
"Supremum"

注意加锁顺序，是先对二级索引记录加锁，再对聚簇索引加锁。

如果对唯一二级索引列进行等值查询的记录并不存在，比如：

```
SELECT * FROM hero WHERE name = 'g关羽' LOCK IN SHARE MODE;
```

为了禁止幻读，所以需要保证别的事务不能再插入 `name` 值为 `'g关羽'` 的新记录。在唯一二级索引 `uk_name` 中，键值比 `'g关羽'` 大的第一条记录的键值为 `l刘备` ，所以需要在这条二级索引记录上加一个 `gap锁` ，如图所示：


gap锁
uk_name索引示意图：
name列：
id列：
c曹操 l刘备 s孙权 x荀彧 z诸葛亮
8 1 20 15 3
"Supremum"

注意，这里只对二级索引记录进行加锁，并不会对聚簇索引记录进行加锁。

- 使用 `SELECT ... FOR UPDATE` 语句来为记录加锁，比如：

和 `SELECT ... LOCK IN SHARE MODE` 语句类似，只不过加的是 `X型正经记录锁` 。

- 使用 `UPDATE ...` 来为记录加锁，比方说：

与 `SELECT ... FOR UPDATE` 的加锁情况类似，不过如果被更新的列中还有别的二级索引列的话，这些对应的二级索引记录也会被加 `X型正经记录锁` 。

- 使用 `DELETE ...` 来为记录加锁，比方说：

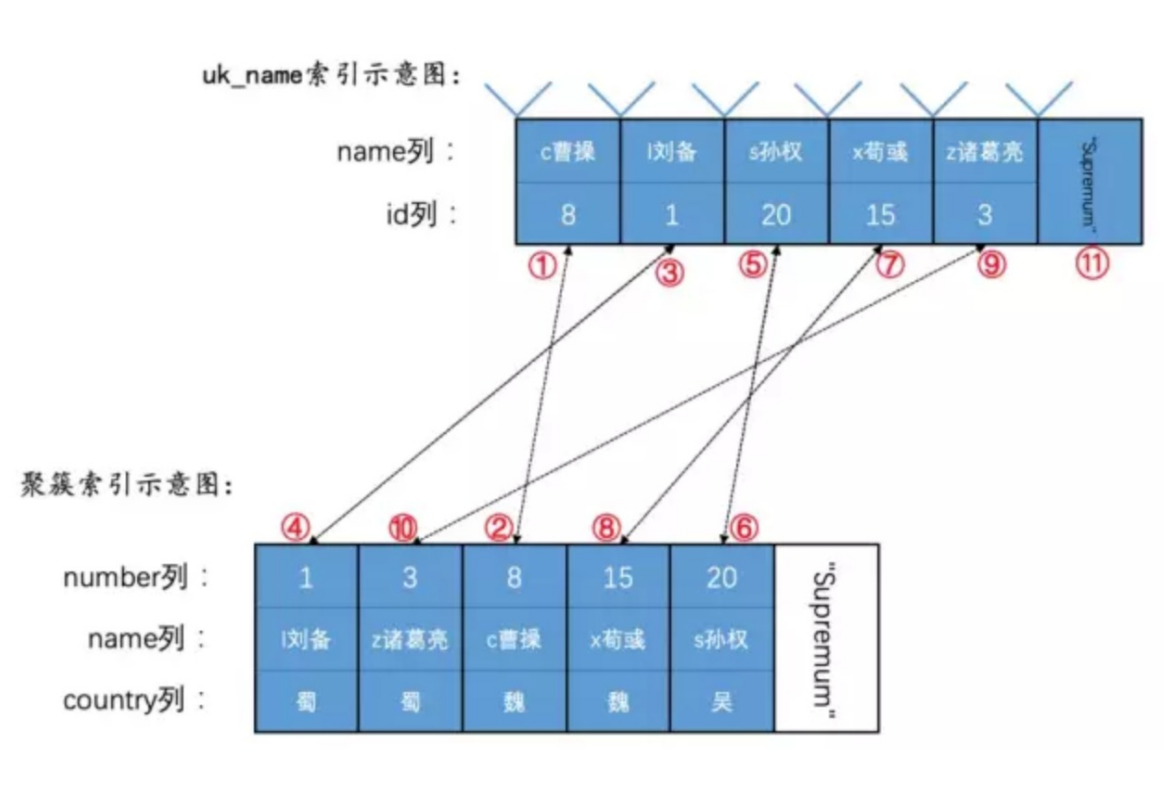
与 `SELECT ... FOR UPDATE` 的加锁情况类似，不过如果表中还有别的二级索引列的话，这些对应的二级索引记录也会被加 `X型正经记录锁` 。

对于使用唯一二级索引进行范围查询的情况

- 使用 `SELECT ... LOCK IN SHARE MODE` 语句来为记录加锁，比方说：

```
SELECT * FROM hero FORCE INDEX(uk_name) WHERE name >= 'c曹操' LOCK IN SHARE MODE;
```

这个语句的执行过程其实是先到二级索引中定位到满足 `name >= 'c曹操'` 的第一条记录，也就是 `name` 值为 `c曹操` 的记录，然后就可以沿着由记录组成的单向链表一路向后找。从二级索引 `idx_name` 的示意图中可以看出，所有的用户记录都满足 `name >= 'c曹操'` 的这个条件，所以所有的二级索引记录都会被加 `S型next-key锁` ，它们对应的聚簇索引记录也会被加 `S型正经记录锁` ，二级索引的最后一条 `Supremum` 记录也会被加 `S型next-key锁` 。不过需要注意一下加锁顺序，对一条二级索引记录加锁完后，会接着对它响应的聚簇索引记录加锁，完后才会对下一条二级索引记录进行加锁，以此类推～ 画个图表示一下就是这样：

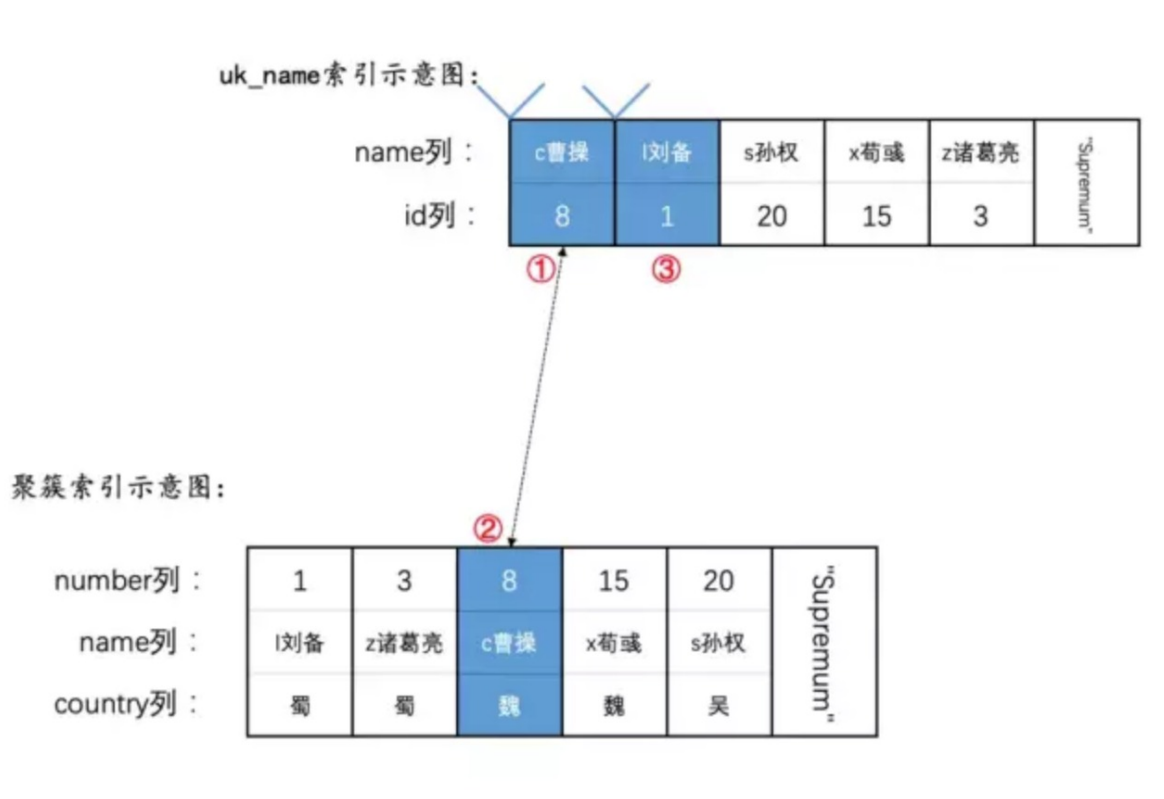

uk_name索引示意图：
name列：
id列：
c曹操 l刘备 s孙权 x荀彧 z诸葛亮
8 1 20 15 3
"Supremum"
① ③ ⑤ ⑦ ⑨ ⑪
聚簇索引示意图：
④ ⑩ ② ⑧ ⑥
number列：
name列：
country列：
1 3 8 15 20
l刘备 z诸葛亮 c曹操 x荀彧 s孙权
蜀 蜀 魏 魏 吴
"Supremum"

稍等一下，不是说 `uk_name` 是唯一二级索引么？唯一二级索引本身就能保证其自身的值是唯一的，那为啥还要给 `name` 值为 `'c曹操'` 的记录加上 `S型next-key锁` ，而不是 `S型正经记录锁` 呢？其实我也不知道，按理说只需要给这条二级索引记录加 `S型正经记录锁` 就好了，我也没想明白设计 `InnoDB` 的大叔是怎么想的，有知道的小伙伴赶紧添加我微信： `xiaohaizi4919` 联系我哈（聊八卦的同学请勿添加）～

再来看下边这个语句：

```
SELECT * FROM hero WHERE name <= 'c曹操' LOCK IN SHARE MODE;
```

这个语句先会为 `name` 值为 `'c曹操'` 的二级索引记录加 `S型next-key锁` 以及它对应的聚簇索引记录加 `S型正经记录锁` 。然后还要给 `name` 值为 `'l刘备'` 的二级索引记录加 `S型next-key锁` ， `name` 值为 `'l刘备'` 的二级索引记录不满足索引条件下推的 `name <= 'c曹操'` 条件，压根儿不会释放掉该记录的锁就直接报告 `server层` 查询完毕了。这样可以禁止其他事务插入 `name` 值在 `('c曹操', 'l刘备')` 之间的新记录，从而防止幻读产生。所以这个过程的加锁示意图如下：


uk_name索引示意图：
name列：
id列：
c曹操 l刘备 s孙权 x荀彧 z诸葛亮
8 1 20 15 3
"Supremum"
① ③
聚簇索引示意图：
②
number列：
name列：
country列：
1 3 8 15 20
l刘备 z诸葛亮 c曹操 x荀彧 s孙权
蜀 蜀 魏 魏 吴
"Supremum"

这里大家要注意一下，设计 InnoDB 的大叔在这里给 `name` 值为 `'l刘备'` 的二级索引记录加的是 `S型next-key锁` ，而不是简单的 `gap锁` 。

- 使用 `SELECT ... FOR UPDATE` 语句来为记录加锁：

和 `SELECT ... LOCK IN SHARE MODE` 语句类似，只不过加的是 `X型正经记录锁` 。

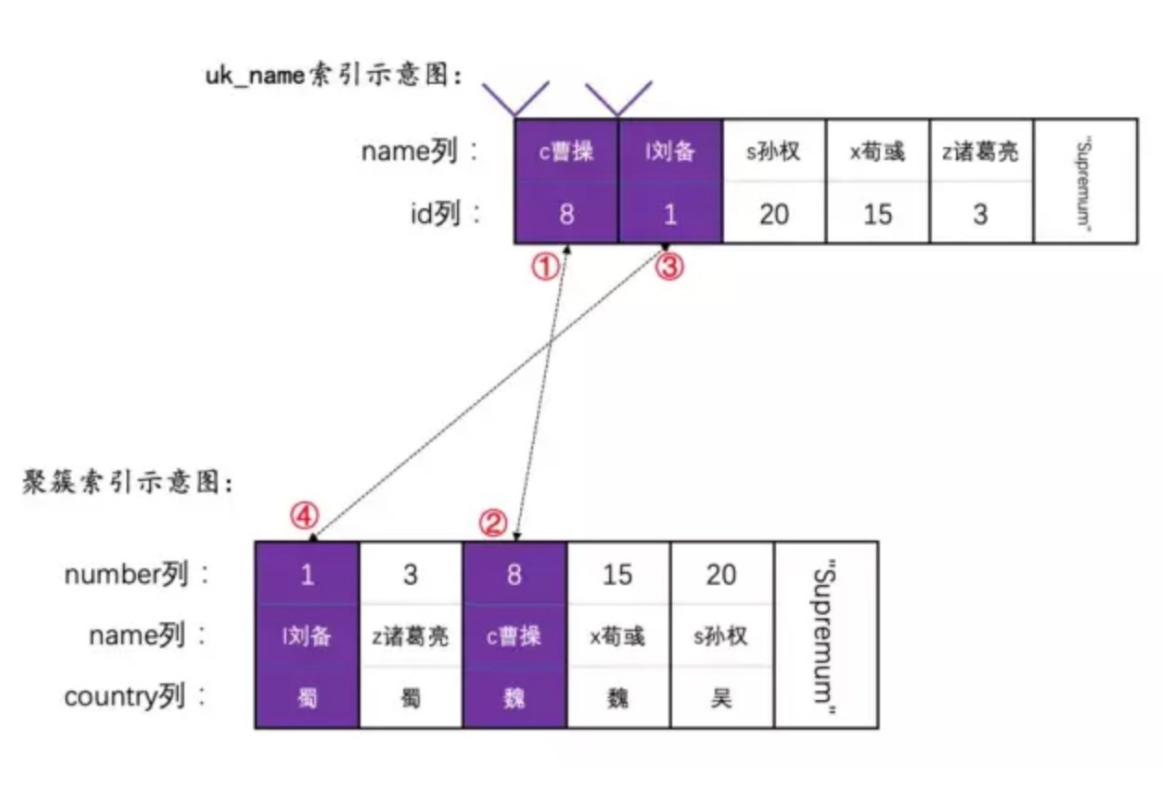
- 使用 `UPDATE ...` 来为记录加锁，比方说：

```
UPDATE hero SET country = '汉' WHERE name >= 'c曹操';
```

假设该语句执行时使用了 `uk_name` 二级索引来进行 `锁定读` （如果二级索引扫描的记录太多，也可能因为成本过大直接使用全表扫描的方式进行 `锁定读` ），而这条 `UPDATE` 语句并没有更新二级索引列，那么它的加锁方式和上边所说的 `SELECT ... FOR UPDATE` 语句一致。如果有其他二级索引列也被更新，那么也会为这些二级索引记录进行加锁，就不赘述了。不过还需要强调一种情况，比方说：

```
UPDATE hero SET country = '汉' WHERE name <= 'c曹操';
```

我们前边说的 `索引条件下推` 这个特性只适用于 `SELECT` 语句，也就是说 `UPDATE` 语句中无法使用，无法使用 `索引条件下推` 这个特性时需要先进行回表操作，那么这个语句就会为 `name` 值为 `'c曹操'` 和 `'l刘备'` 的二级索引记录加 `X型next-key锁` ，对它们对应的聚簇索引记录进行加 `X型正经记录锁` 。不过之后在判断边界条件时，虽然 `name` 值为 `'l刘备'` 的二级索引记录不符合 `name <= 'c曹操'` 的边界条件，但是在REPEATABLE READ隔离级别下并不会释放该记录上加的锁，整个过程的加锁示意图就是：


uk_name索引示意图：
name列：
id列：
c曹操 l刘备 s孙权 x荀彧 z诸葛亮
8 1 20 15 3
"Supremum"
① ③
聚簇索引示意图：
④ ②
number列：
name列：
country列：
1 3 8 15 20
l刘备 z诸葛亮 c曹操 x荀彧 s孙权
蜀 蜀 魏 魏 吴
"Supremum"

- 使用 `DELETE ...` 来为记录加锁，比方说：

```
DELETE FROM hero WHERE name >= 'c曹操';
```

和

```
DELETE FROM hero WHERE name <= 'c曹操';
```

如果这两个语句采用二级索引来进行 `锁定读` ，那么它们的加锁情况和更新带有二级索引列的 `UPDATE` 语句一致，就不画图了。

对于使用普通二级索引进行等值查询的情况

我们再把上边的唯一二级索引 `uk_name` 改回普通二级索引 `idx_name` ：

```
ALTER TABLE hero DROP INDEX uk_name, ADD INDEX idx_name (name);
```

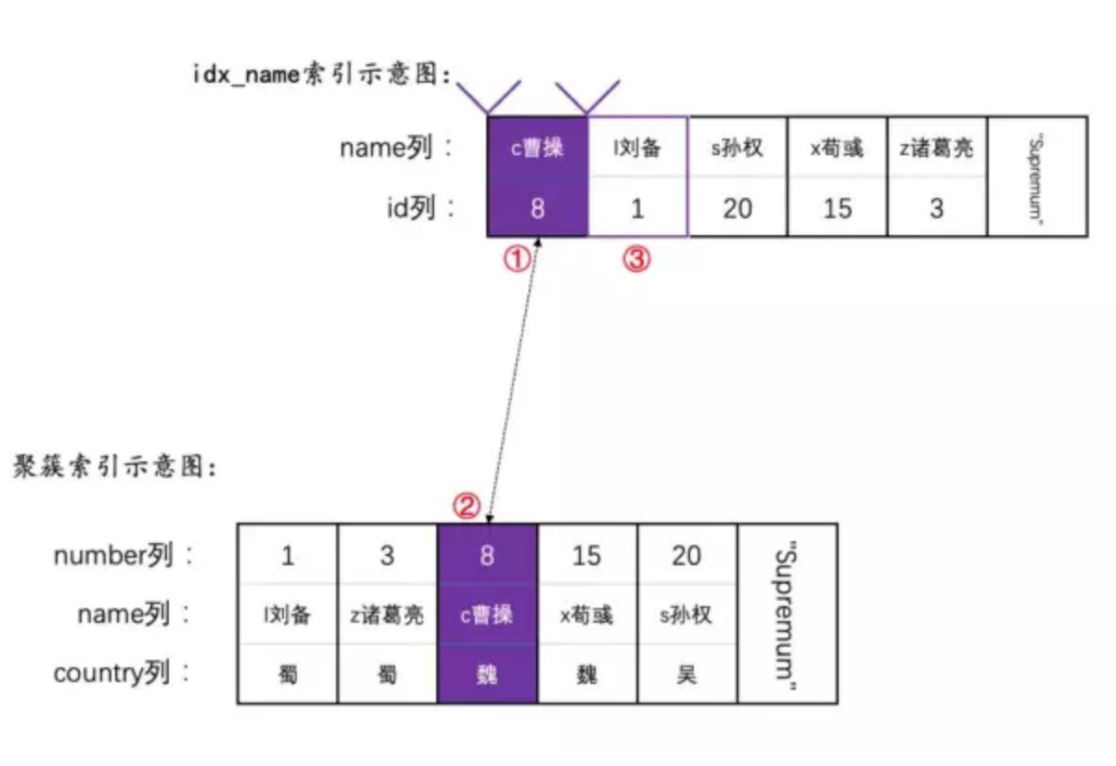
- 使用 `SELECT ... LOCK IN SHARE MODE` 语句来为记录加锁，比方说：

```
SELECT * FROM hero WHERE name = 'c曹操' LOCK IN SHARE MODE;
```

由于普通的二级索引没有唯一性，所以一个事务在执行上述语句之后，要阻止别的事务插入 `name` 值为 `'c曹操'` 的新记录，设计 `InnoDB` 的大叔采用下边的方式对上述语句进行加锁：

  - 对所有 `name` 值为 `'c曹操'` 的二级索引记录加 `S型next-key锁` ，它们对应的聚簇索引记录加 `S型正经就锁` 。
  - 对最后一个 `name` 值为 `'c曹操'` 的二级索引记录的下一条二级索引记录加 `gap锁` 。

所以整个加锁示意图就如下所示：


idx_name索引示意图：
name列：
id列：
c曹操 l刘备 s孙权 x荀彧 z诸葛亮
8 1 20 15 3
"Supremum"
① ③
聚簇索引示意图：
②
number列：
name列：
country列：
1 3 8 15 20
l刘备 z诸葛亮 c曹操 x荀彧 s孙权
蜀 蜀 魏 魏 吴
"Supremum"

如果对普通二级索引等值查询的值并不存在，比如：

```
SELECT * FROM hero WHERE name = 'g关羽' LOCK IN SHARE MODE;
```

加锁方式和我们上边唠叨过的唯一二级索引的情况是一样的，就不赘述了。

- 使用 `SELECT ... FOR UPDATE` 语句来为记录加锁，比如：

和 `SELECT ... LOCK IN SHARE MODE` 语句类似，只不过加的是 `X型正经记录锁` 。

- 使用 `UPDATE ...` 来为记录加锁，比方说：

与 `SELECT ... FOR UPDATE` 的加锁情况类似，不过如果被更新的列中还有别的二级索引列的话，这些对应的二级索引记录也会被加锁。

- 使用 `DELETE ...` 来为记录加锁，比方说：

与 `SELECT ... FOR UPDATE` 的加锁情况类似，不过如果表中还有别的二级索引列的话，这些对应的二级索引记录也会被加锁。

对于使用普通二级索引进行范围查询的情况

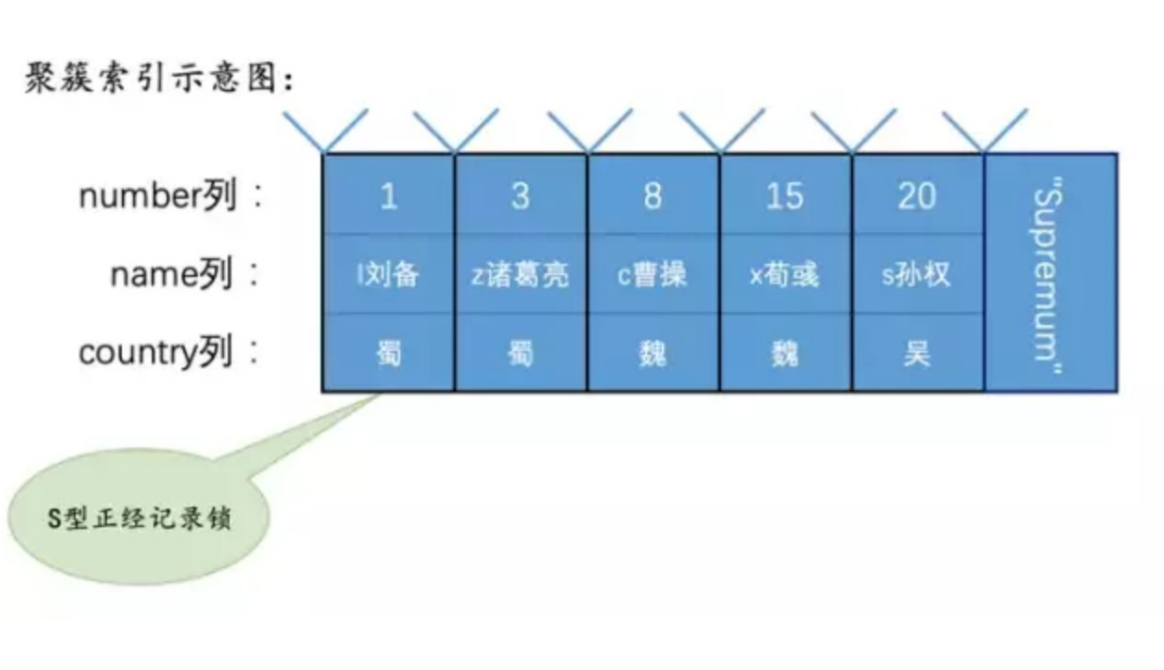
与唯一二级索引的加锁情况类似，就不多唠叨了哈～

全表扫描的情况

比方说：

```
SELECT * FROM hero WHERE country  = '魏' LOCK IN SHARE MODE;
```

由于 `country` 列上未建索引，所以只能采用全表扫描的方式来执行这条查询语句，存储引擎每读取一条聚簇索引记录，就会为这条记录加锁一个 `S型next-key锁` ，然后返回给 `server层` ，如果 `server层` 判断 `country = '魏'` 这个条件是否成立，如果成立则将其发送给客户端，否则会向 `InnoDB` 存储引擎发送释放掉该记录上的锁的消息，不过在REPEATABLE READ隔离级别下，InnoDB存储引擎并不会真正的释放掉锁，所以聚簇索引的全部记录都会被加锁，并且在事务提交前不释放。画个图就像这样：


聚簇索引示意图：
number列：
name列：
country列：
1 3 8 15 20
l刘备 z诸葛亮 c曹操 x荀彧 s孙权
蜀 蜀 魏 魏 吴
"Supremum"
S型正经记录锁

大家看到了么：全部记录都被加了next-key锁！此时别的事务别说想向表中插入啥新记录了，就是对某条记录加 `X` 锁都不可以，这种情况下会极大影响访问该表的并发事务处理能力，所以如果可能的话，尽可能为表建立合适的索引吧～

使用 `SELECT ... FOR UPDATE` 进行加锁的情况与上边类似，只不过加的是 `X型正经记录锁` ，就不赘述了。

对于 `UPDATE ...` 语句来说，加锁情况与 `SELECT ... FOR UPDATE` 类似，不过如果被更新的列中还有别的二级索引列的话，这些对应的二级索引记录也会被加 `X型正经记录锁` 。

和 `DELETE ...` 的语句来说，加锁情况与 `SELECT ... FOR UPDATE` 类似，不过如果表中还有别的二级索引列的话，这些对应的二级索引记录也会被加 `X型正经记录锁` 。

未完待续

下一章节继续唠叨INSERT语句的加锁情况，敬请期待。如果文中有任何问题，请联系作者：xiaohaizi4919（正经微信，扯犊子的请勿添加）。

关注小青蛙，全都是技术干货哈：

我们都是小青蛙
不做懒惰之蛙，不做井底之蛙
好好学本领，来把害虫抓
动动大拇指，长按二维码关注
微信公众平台

原文链接

大家可以点击原文链接，查看《MySQL是怎样运行的：从根儿上理解MySQL》的完整内容，通俗到爆炸💥的MySQL进阶读物，原来学习可以这么有趣（也可以扫下边的二维码直接观看）：

阅读原文