**Select加锁查询语句中，WHERE条件是主键等值查询（也即聚簇索引等值查询）时**

start transaction;

select \* from user where id=1 for update;

该事务a提交前：

事务 A 中该sql按序扫描主键索引树，在id =1这个主键索引值上加 X独占型锁。==这样相当于数据页中id=1这一行也被锁住。{由锁住的范围可知是一个Record Lock行锁，由原本加的Next-Key Lock 临键锁退化而来}。



如果有其他事务对 id=1这一行进行更新或者删除操作，这些操作都会被阻塞，因为操作前先扫描主键索引树时，更新或者删除操作需要对主键索引id=1加 X 型锁，而 X 锁和 X 锁之间是互斥关系。

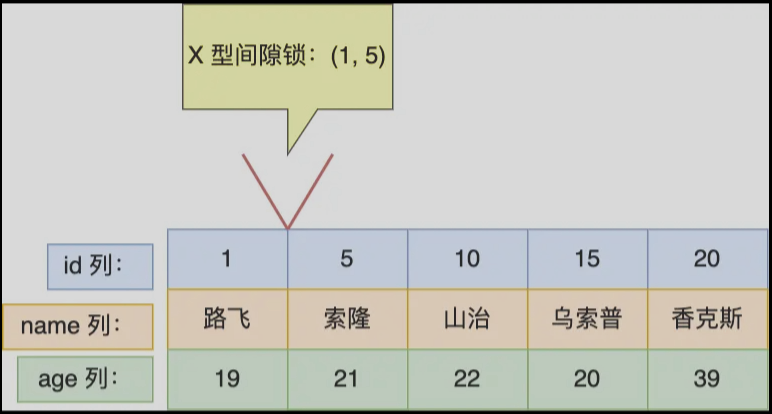
但其他事务是可以再新增一行的，即使新增一行id也为1，也并不会被阻塞，因为新增一行并不会挤现在这一行的位置，但id为1和现有的自增id=1冲突故会报错。

start transaction;

select \* from user where id=2 for update;

该事务a提交前：

事务 A 中该sql按序扫描主键索引树，在 id = 5 这个主键索引值上加X独占型锁，锁住的范围是 (1, 5)。==这样相当于数据页中id在（1,5）的行被锁住。{由锁住的范围可知是一个Gap Lock间隙锁，由原本加的Next-Key Lock 临键锁退化而来}。



如果有其他事务插入 id 值为 2、3、4 这一些行，都会发生阻塞（若update别的行的id，id索引树随之更新后，此时若被改后的id按序应该放到这里被锁的范围来，那么和插入一样也被阻塞，已验证）；

其他事务插入 id = 1 或者 id = 5 的行，并不会发生阻塞，而是报主键冲突的错误。

**Select加锁查询语句中，WHERE条件是主键范围查询**

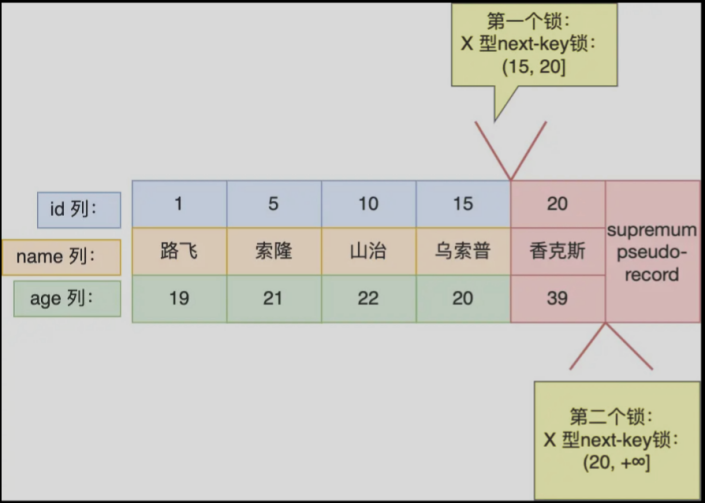
start transaction;

select \* from user where id>15 for update;

该事务a提交前：

事务 A 中该sql按序扫描主键索引树，在 id = 20 这个主键索引值上加X独占型锁，锁住的范围是 (15, 20]。==这样相当于数据页中id为(15, 20]的范围被锁住；{由锁住的范围可知是一个临键锁，就是原本加的Next-Key Lock 临键锁，没退化}。

事务 A 中该sql继续按序扫描主键索引树，在 id =supremum pseudo-record 这个主键索引值上加X独占型锁，锁住的范围是(20, +∞]==这样相当于数据页中id为(20, +∞]范围被锁住。{由锁住的范围可知是一个临键锁，就是原本加的Next-Key Lock 临键锁，没退化}。。



如果其他事务要更新或者删除 id = 20 的行，则会被阻塞；

其他事务要插入id=20的行，会报主键冲突。

其他事务要插入 id 值为 16、17、18、19 的这一些新的行，也会被阻塞（若update别的行的id，id索引树随之更新后，此时若被改后的id按序应该放到这里被锁的范围来，那么和插入一样也被阻塞，已验证）。

其他事务要插入 id 值大于 20 的这一些新的行，也会被阻塞（若update别的行的id，id索引树随之更新后，此时若被改后的id按序应该放到这里被锁的范围来，那么和插入一样也被阻塞，已验证）。

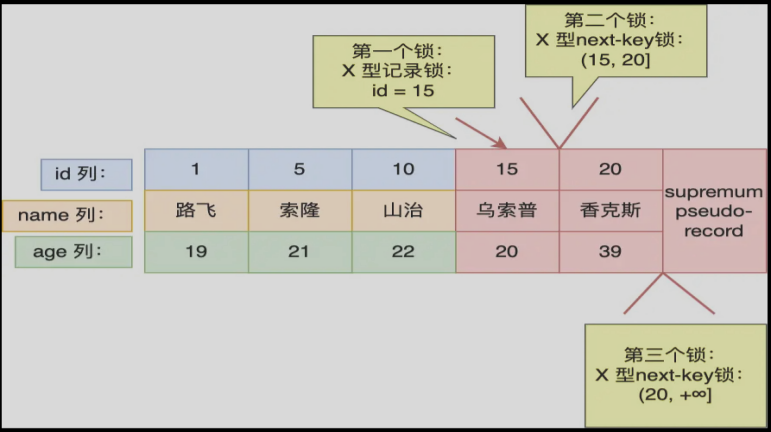
start transaction;

select \* from user where id>=15 for update;

该事务a提交前：

事务 A 中该sql按序扫描主键索引树，在 id = 15 这个主键索引值上加X独占型锁，锁住的范围是id=15。==这就相当于即数据页中id=15这一行被锁住。{由锁住的范围可知是一个Record Lock行锁，由原本的Next-Key Lock 临键锁退化而来}。 事务A继续按序扫描主键索引树，在 id = 20这个主键索引值上加锁，锁住的范围为 (15, 20]。==这就相当于数据页中id为 (15, 20]的行被锁住。{由锁住的范围可知是一个临键锁，就是原本加的Next-Key Lock 临键锁，没退化}

事务 A 中该sql继续按序扫描主键索引树，在 id =supremum pseudo-record 这个主键索引值上加锁，锁住的范围是(20, +∞]。==这就相当于数据页中id为(20, +∞]范围被锁住。{由锁住的范围可知是一个临键锁，就是原本加的Next-Key Lock 临键锁，没退化}



如果其他事务要更新或删除id=15，id=20的行，则会被阻塞；

其他事务要插入id为16、17、18、19的行也会被阻塞（若update别的行的id，id索引树随之更新后，此时若被改后的id按序应该放到这里被锁的范围来，那么和插入一样也被阻塞，已验证）；

其他事务要插入id大于20的行也会被阻塞（若update别的行的id，id索引树随之更新后，此时若被改后的id按序应该放到这里被锁的范围来，那么和插入一样也被阻塞，已验证）。

start transaction;

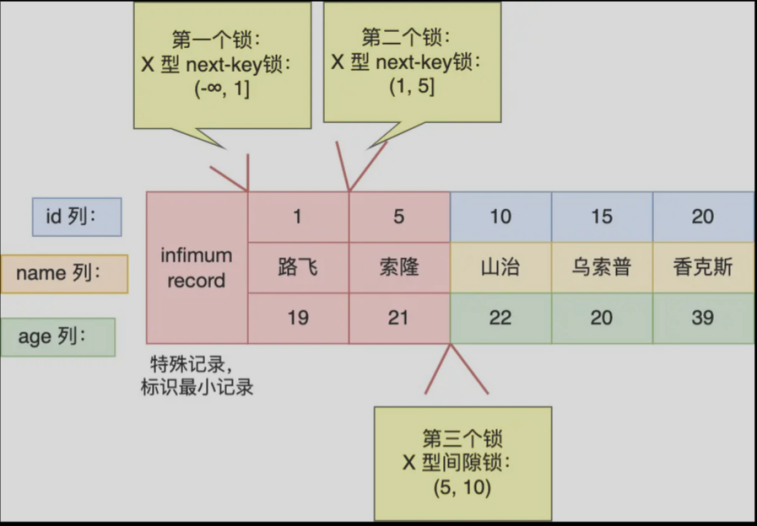
select \* from user where id<6 for update; （改成<=6本例下面结论也不变）

该事务a提交前：

事务 A 中该sql按序扫描主键索引树，在 id = 1 这个主键索引值上加锁，锁住的范围是(-∞, 1]。==这就相当于数据页中id为(-∞, 1]的行被锁住。{由锁住的范围可知是一个临键锁，就是原本加的Next-Key Lock 临键锁，没退化}

事务A 继续按序扫描主键索引树，在 id = 5 这个主键索引值上加锁，锁住的范围为 (1, 5] 。===这就相当于数据页中id为 (1, 5] 的行被锁住。{由锁住的范围可知是一个临键锁，就是原本加的Next-Key Lock 临键锁，没退化}

事务A 继续按序扫描主键索引树，在 id = 10 这个主键索引上加的锁，锁住的范围为 (5, 10) 。===这就相当于数据页中id为 (5, 10) 的行被锁住。{由锁住的范围可知是一个Gap Lock间隙锁，由原本加的Next-Key Lock 临键锁退化而来}



如果其他事务要更新或删除id=1、id=5的行，则会被阻塞；

如果其他事务要插入id<1、id为2、3、4 或 id为6、7、8、9的行也会被阻塞（若update别的行的id，id索引树随之更新后，此时若被改后的id按序应该放到这里被锁的范围来，那么和插入一样也被阻塞，已验证）。

start transaction;

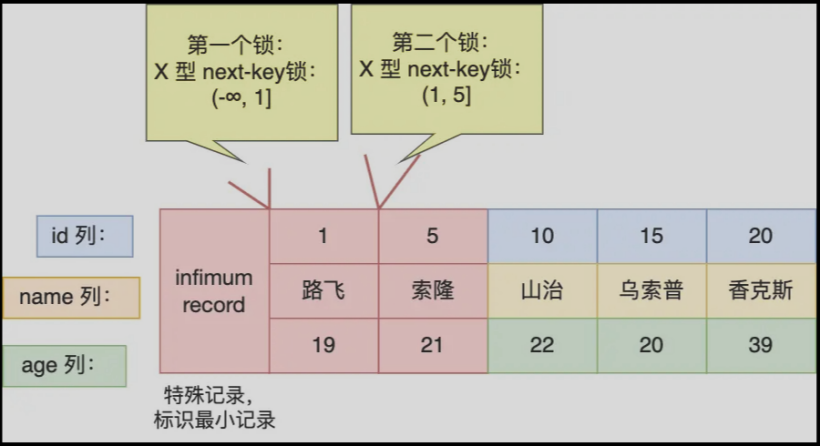
select \* from user where id<=5 for update;

该事务a提交前：

事务 A 中该sql按序在主键索引树中扫描，在 id = 1 这个主键索引值上加锁，锁住的范围是(-∞, 1]。===这就相当于数据页中id为(-∞, 1]的行被锁住。{由锁住的范围可知是一个临键锁，就是原本加的Next-Key Lock 临键锁，没退化}

事务 A 中该sql继续按序扫描主键索引树，在 id = 5 这个主键索引值上加锁，锁住的范围是(1, 5]。===这就相当于数据页中id为(1, 5]的行被锁住。{由锁住的范围可知是一个临键锁，就是原本加的Next-Key Lock 临键锁，没退化}

由于id是主键，不可能重复，因此不再在主键索引树上继续按序扫描。



其他事务如果要更新或删除id=1、id=5的行，则会被阻塞；

如果其他事务要插入id<1、id为2、3、4的行则会被阻塞（若update别的行的id，id索引树随之更新后，此时若被改后的id按序应该放到这里被锁的范围来，那么和插入一样也被阻塞，已验证）。

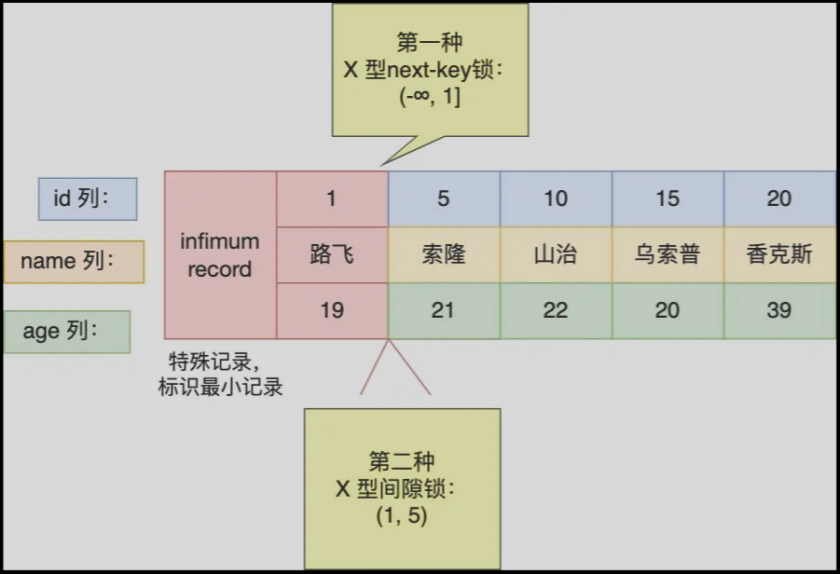
start transaction;

select \* from user where id<5 for update;

该事务a提交前：

事务 A 中该sql按序扫描主键索引树，在 id = 1 这个主键索引值上加锁，锁住的范围是(-∞, 1]。这就相当于数据页中id为(-∞, 1]的行被锁住。{由锁住的范围可知是一个临键锁，就是原本加的Next-Key Lock 临键锁，没退化}

事务 A 中该sql继续按序扫描主键索引树，在 id = 5 这个主键索引值上加锁，锁住的范围是(1, 5）。===这就相当于数据页中id为(1, 5）的行被锁住。{由锁住的范围可知是一个Gap Lock间隙锁，由原本加的Next-Key Lock 临键锁退化而来}



其他事务如果要修改或删除id=1的行，则会被阻塞；

如果其他事务要插入id<1、id为2、3、4的行则会被阻塞（若update别的行的id，id索引树随之更新后，此时若被改后的id按序应该放到这里被锁的范围来，那么和插入一样也被阻塞，已验证）。

（完）