注：本次实训主要讨论事务隔离级别、索引(主键索引、普通索引或唯一索引)与锁的关系问题，即：在某种级别下以某种方式访问数据表时mysql将如何加锁？

目录

[20.2　加锁验证 1](#_Toc24207)

[20.2.1　REPEATABLE-READ隔离级别+表无显式主键和索引 2](#_Toc10757)

[20.2.2　REPEATABLE-READ隔离级别+表有显式主键但无索引 5](#_Toc23395)

[20.2.3　REPEATABLE-READ隔离级别+表无显式主键但有索引 7](#_Toc18486)

[20.2.4　REPEATABLE-READ隔离级别+表有显示主键和索引 10](#_Toc2665)

[20.2.5　READ-COMMITTED隔离级别+表无显式主键和索引 13](#_Toc18302)

[20.2.6 READ-COMMITTED隔离级别+表有显式主键无索引 14](#_Toc27688)

[20.2.7　READ-COMMITTED隔离级别+表无显式主键有索引 14](#_Toc30110)

[20.2.8　READ-COMMITTED隔离级别+表有显式主键和索引 15](#_Toc4429)

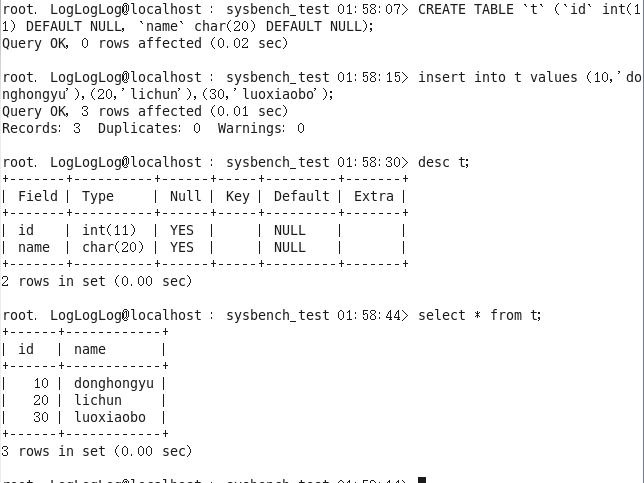
## 20.2　加锁验证

前面已经介绍了InnoDB中相关的锁概念，相信很多MySQL DBA或开发人员对这些锁都有所了解，也知道它们的作用，但问起具体加锁情况时，则很难做出比较详细的解释和清晰的描述，比如**锁和事务隔离级别的关系，以及和主键、索引之间是否有影响**等。

创建一个表t，**没有索引和主键**，并插入测试数据：

CREATE TABLE `t` (`id` int(11) DEFAULT NULL, `name` char(20) DEFAULT NULL);

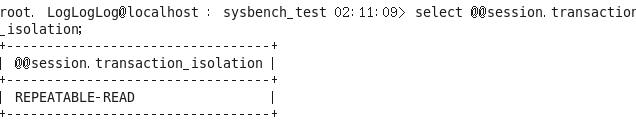
insert into t values (10,'donghongyu'),(20,'lichun'),(30,'luoxiaobo');



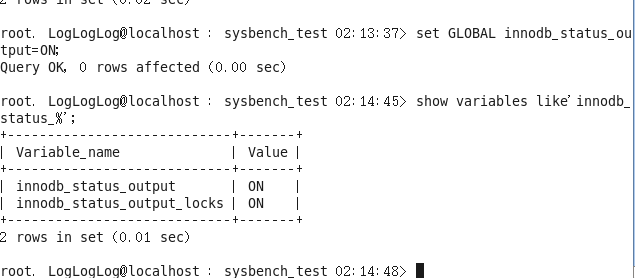
其他准备：

设置全局事务隔离级别为可重复读：

一定要新开会话确认，否则(实验室环境默认为提交读级别)后面一些地方出来结果不对：



打开如下开关以获取更多的锁信息：



由于操作时会输出大量的状态信息，所以设置(菜单“编辑”)字符终端不限定回滚行数：

图形用户界面, 文本, 应用程序

描述已自动生成

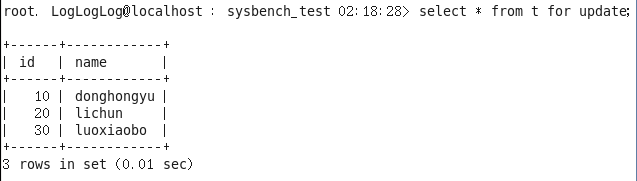
## 20.2.1　REPEATABLE-READ隔离级别+表无显式主键和索引

#手动开启事务，执行语句并采用**for update方式**（当前读），

**(1) 不带where条件的情况**

mysql> begin;

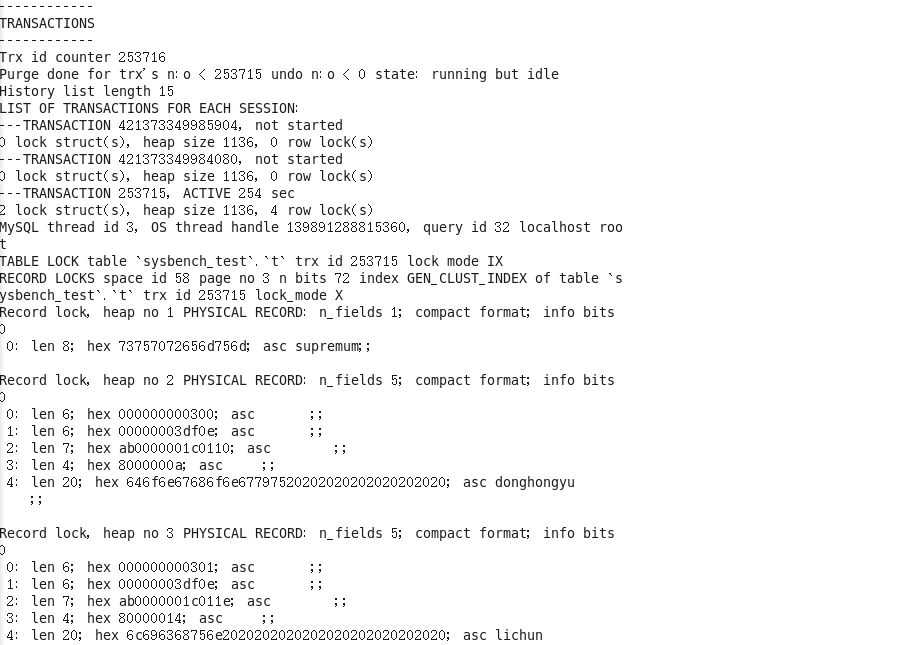
mysql> select \* from t for update;



#使用show engine innodb语句获取结果(另外新建会话，书中主要以mysql8.0中的performance\_schema.data\_locks表分析，但实质结果一样)：

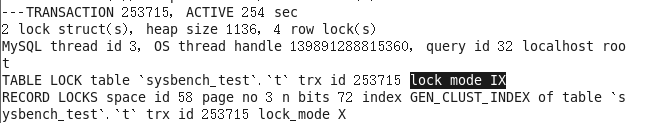
文本

中度可信度描述已自动生成



以下是对结果的分解和解读：

**第1步**：对表**添加了IX锁**（在获取表中某行的独占锁之前，首先必须获取表的IX锁）



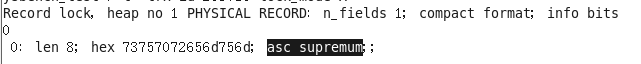
**第2步**：在“supremum”上**添加Next-Key Lock锁（LOCK\_MODE字段显示的是“X”，**而且仅有X**）**。

文本

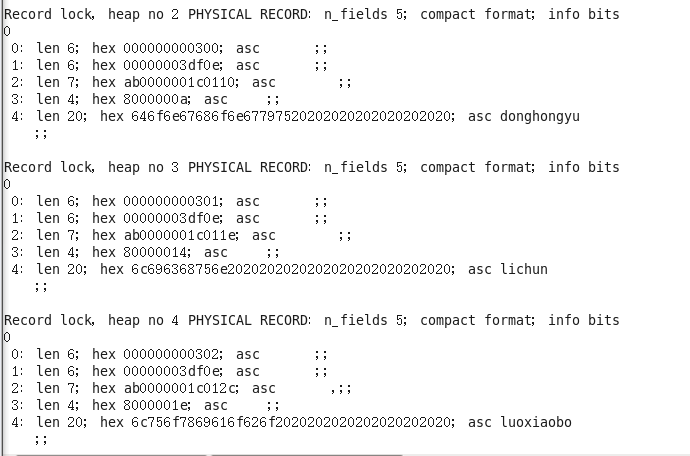
中度可信度描述已自动生成

在REPEATABLE-READ隔离级别下，为了防止发生幻读，例如进行了更新、删除操作，会将最大索引值(当前是id=30)**之后的间隙**锁住并用“**supremum**”**表示高于表中任何一个索引**的值：

换言之，以下锁住了区间[30, supremum]



**第3步**：在3条记录上分别添加Next-Key Lock锁(行锁，不再重复显示LOCK\_MODE)，即锁住了区间(Infimum-页面最小记录,10], (10,20], (20,30]：



**(2) 带有where条件的情况**

这里有读者会有疑问，**这是否和执行语句没有where条件有关？**如果带有where条件，则不会有Next-Key Lock锁。也就是说如果只查询一行记录还会有上面的锁操作吗？下面我们测试一下(结束之前事务)。

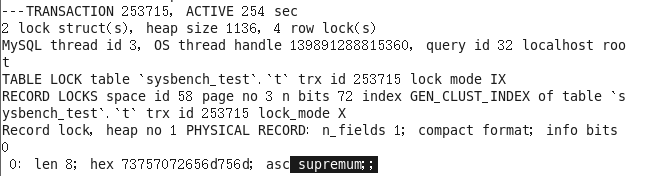
mysql> begin;

Query OK, 0 rows affected (0.00 sec)

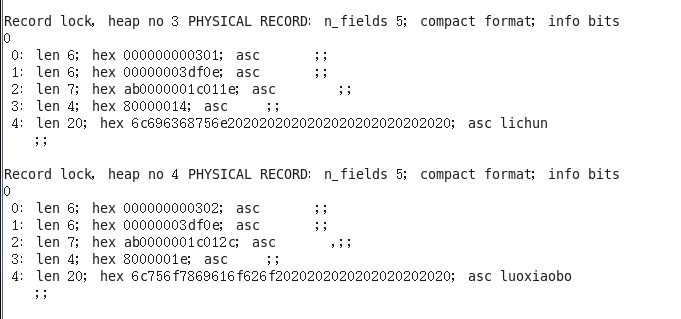
mysql> select \* from t **where id = 10** for update;

我们看到**锁信息与之前一样**，同样会有“supremum pseudo-record”。正如前面所讲的，虽然where条件是id = 10，但是每次插入记录时所生成的聚集索引（DB\_ROW\_ID，按记录归类建立的索引，即使没有指定索引mysql也会自动加这种索引）还是自增的，**每次都会在表的最后插入**，所以就有可能插入id = 10这条记录(注意表t没有主键，如果插入id=10的记录，就会自动跟原id=10的记录通过更新索引“聚集”，但这条记录还是会放到最后)。因此，需要添加一条“supremum pseudo-record”防止数据插入。

总之，后面我们通过插入操作证实“每次都会在表的最后插入”，因此即使是指定where条件“id=10”，还是会保留前面分析的第2步，在“supremum”上添加Next-Key Lock锁：



这里还有一个问题：为什么有了where条件，但是还会在不满足where条件的记录上添加Next-KeyLock锁呢？比如，如下2个锁对应的记录均不符合where条件(这一点后面会结合20.2.5节-事务隔离为“提交读”的情况来对比)：



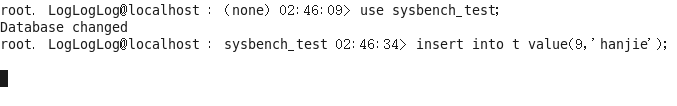
主要也是为了**防止发生幻读**。如果不添加Next-Key Lock锁，这时若有其他会话执行DELETE或者UPDATE语句，则都会造成幻读。

这时(另开会话2)插入任何数据都会被阻塞，

mysql> begin;

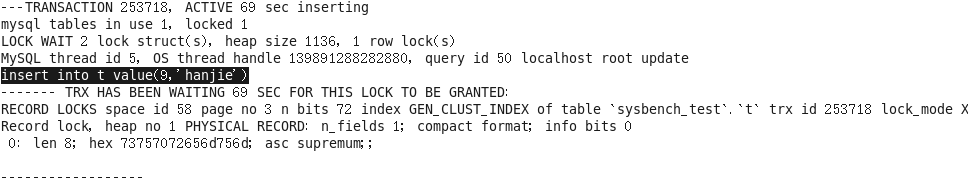
mysql> insert into t values (**9**, 'hanjie');

ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction

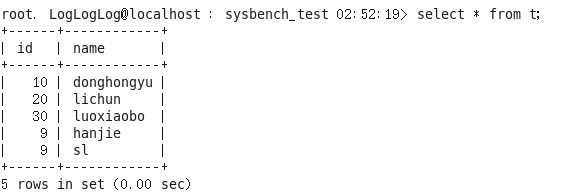


因为**都是在表的最后插入的**(“聚集索引”)，会与“supremum pseudo-record”发生冲突，就是说

虽然插入id=9的数据行，但实际还是要对“到supremum的间隙”加锁，留意下面事务等待的插入意向锁：



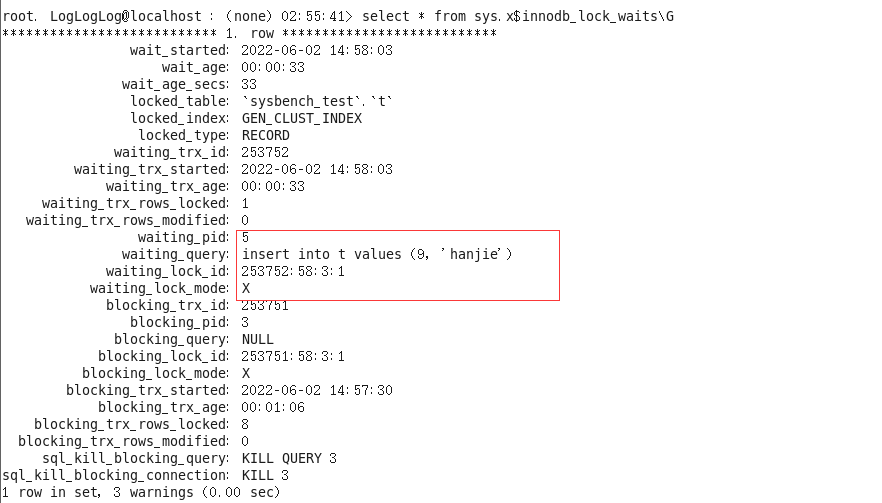
注：会话1事务提交后，会话2重新执行插入操作就可见到记录还是插入在表最后。这里顺便多插入一行id=10的记录以回应前面的讨论。也就是说，表定义本身允许插入相同id的数据行，只是不允许同时两个会话插入。



insert into t value(9,'sl');

通过锁冲突信息可以清楚地看到，是申请添加的插入意向锁与“supremum pseudo-record”发生了冲突，这也**证明了插入时都是在表的最后插入的**。

补充：我们在前面也学习过查询事务锁等待的一些方法，比如重新执行上面的两个会话的事务，同样也可以用sys的工具查询：



但是所得信息主要指导如何解决锁问题(告诉我们杀掉持有锁的会话)，锁本身的信息自然不及这里详细。

## 20.2.2　REPEATABLE-READ隔离级别+表有显式主键但无索引

分如下几种情况进行分析，对于不同的情况加锁方式也有所不同。

● 不带where条件。

● where条件是主键字段。

● where条件包含主键字段和非主键字段。

创建表结构如下：

mysql> show create table dhytest\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Table: dhytest

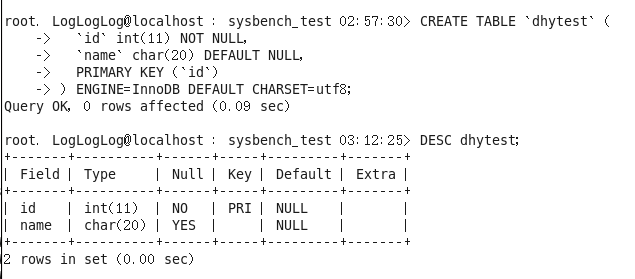
Create Table: CREATE TABLE `dhytest` (

`id` int(11) NOT NULL,

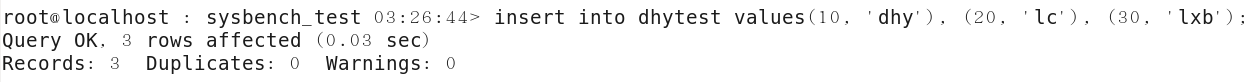
`name` char(20) DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8



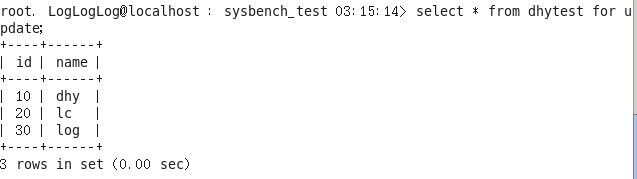
然后像前面那样插入三行数据：



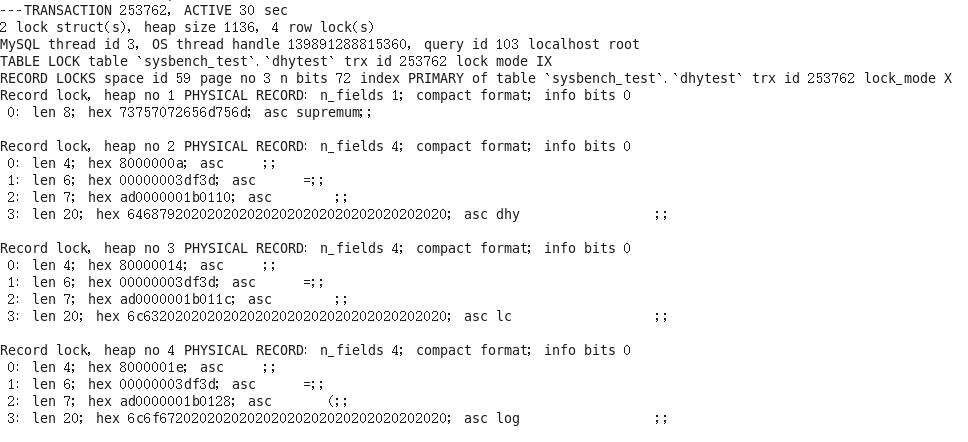
**1．不带where条件lo**

mysql> begin;

mysql> select \* from dhytest for update;(此处估计查询应为dhytest)



查看data\_locks表的加锁情况：



可以看到，没有where条件时加锁方式与20.2.1节(无显式主键也无索引)中不带where条件的加锁方式相同，有where条件就不同了。

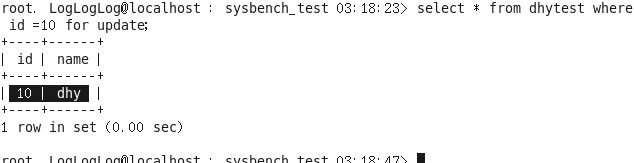
注：差别就在于绿色高亮处，20.2.1节此处生成聚集索引(GEN\_CLUST\_INDEX)。

**2．where条件是主键字段**

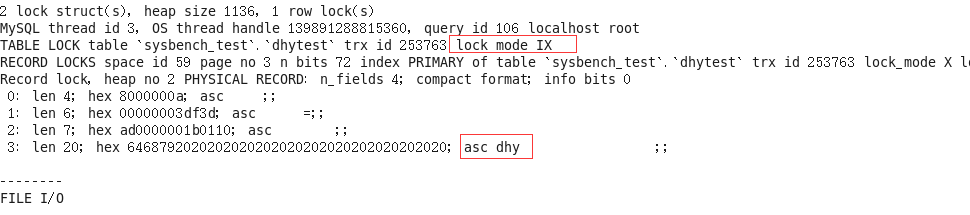
结束前一事务。

mysql> begin;

mysql> select \* from dhytest where id = 10 for update;



查看加锁情况：



注：以上结果一定要对比20.2.1节(无显式主键也无索引)中语句“select \* from t where id = 10 for update;”的相关锁信息来理解。

可以看到，只对表添加了IX锁(20.2.1节中对应处一样) 和对主键添加了记录锁（X,REC\_NOT\_GAP），并且只锁住了where条件id = 10这条记录(不一样)。因为**主键已经保证了唯一性**，所以在插入时就不会是id = 10这条记录了。因此，这里也不需要间隙锁。

注：显然如果我们继续操作插入一行id=10自然不会被允许。这是由主键本身决定的。

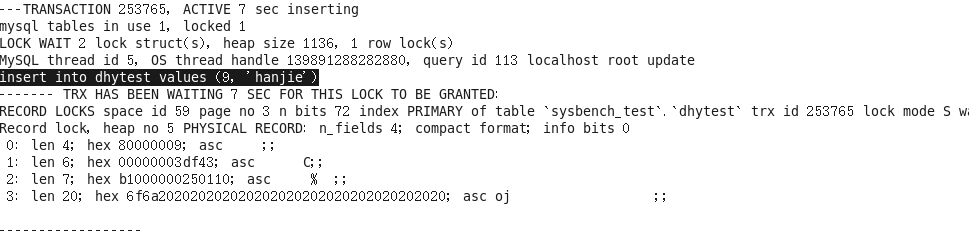


也就是说，表dhytest的定义本身决定了不允许插入重复行，因此不需要像前面那样加锁，只需要锁定对应记录行即可。

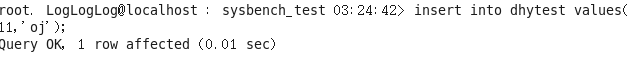
但是只锁定主键id=10的数据行不意味跟其他更新插入操作不冲突，如果我们插入id<10的数据行，同样会引起冲突，另开会话：



然后查询锁信息：



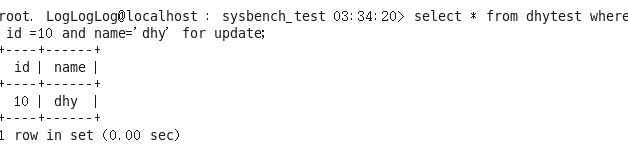
这是因为插入意向锁属于特殊的间隙锁，这里其实锁定的间隙与前面关于id=10的记录锁冲突("asc dhy")。如果插入的位置在id=10之后，则是允许的：



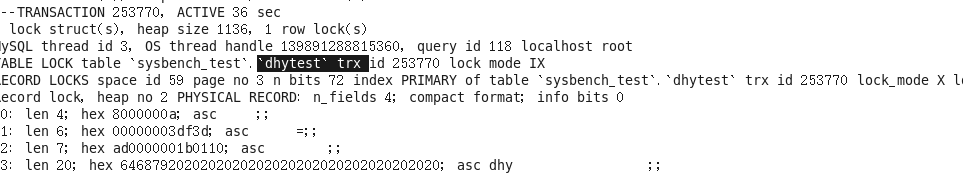
**3．where条件包含主键字段和非主键字段**

mysql> begin;

mysql> select \* from dhytest where id = 10 and name = 'dhy' for update;



查看加锁情况：



可以看到，加锁方式与where条件是主键字段的加锁方式相同，因为**根据主键字段可以直接定位一条记录**。

## 20.2.3　REPEATABLE-READ隔离级别+表无显式主键但有索引

这里分如下几种情况。

● 不带where条件

● 普通索引：

■ where条件是索引字段。

■ where条件包含索引字段和非索引字段。

● 唯一索引：

■ where条件是索引字段。

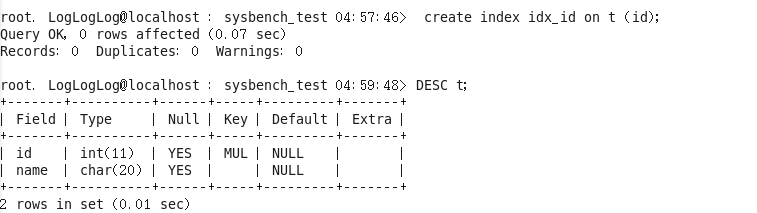
■ where条件包含索引字段和非索引字段。

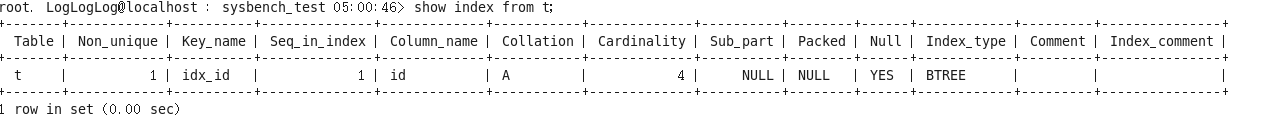
**1．不带where条件**

这种情况的加锁方式与20.2.1节(无主键无索引)中不带where条件的加锁方式相同，相信大家也能理解为何这样加锁，这里就不演示了，我们重点看一下不一样的地方。

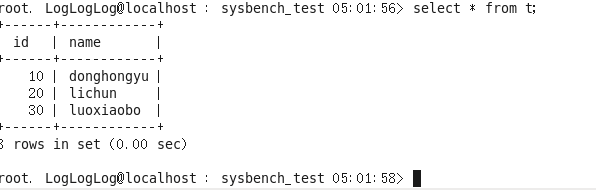
**2．普通索引**

mysql> create index idx\_id on t (id);



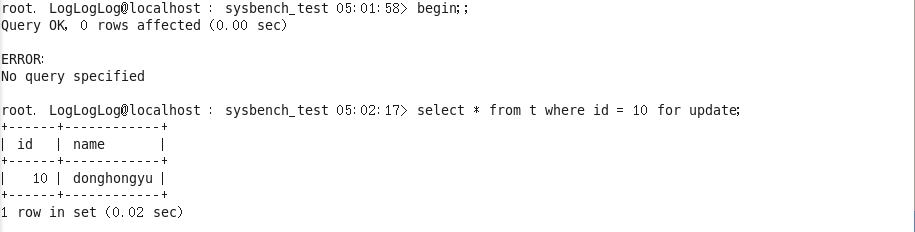


注：为便于讨论，把表t恢复到原来的样子。



mysql> begin;

mysql> select \* from t where **id = 10** for update;



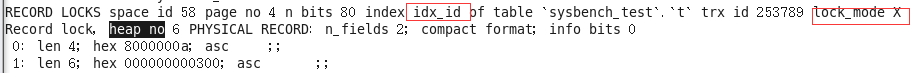
**2.1 当where条件是普通索引字段时**，

加锁顺序是(显示信息略有不同)：

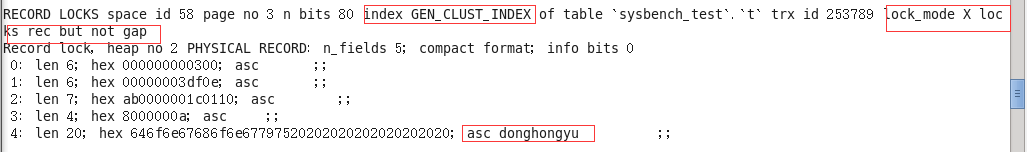
① 对表添加IX锁。



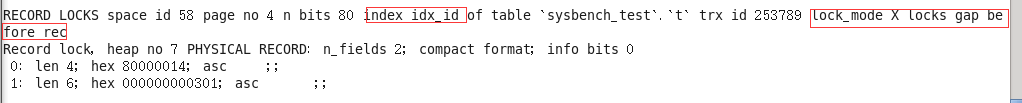
② 对id = 10**对应的索引**添加Next-Key Lock锁，区间是（-∞, 10]。注意“heap no 3”：



③ 对索引对应的**聚集索引**添加X**记录锁**。



④ 为**防止发生幻读**（因为是普通索引，所以**可以再插入id =10**这条记录），对**索引记录区间（10,20）添加间隙锁**。



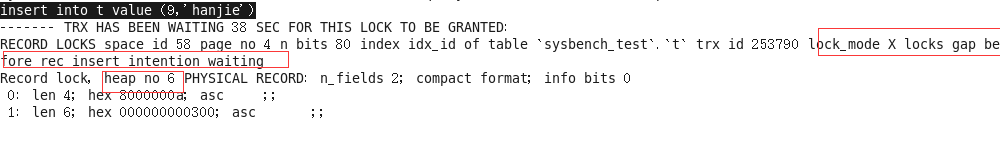
(另开会话)这时如果插入id = 9到id = 19之间的记录都会被阻塞

mysql> begin;

mysql> insert into t values (9, 'hanjie');



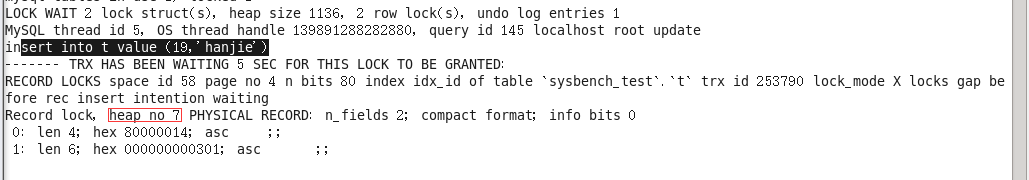
当插入id = 9这条记录时同样需要添加间隙锁（其实是插入意向锁），**与id = 10这条记录的Next-Key Lock锁**(区间为（-∞, 10])**发生了冲突**，所以看到申请X锁、间隙锁、插入意向锁时状态是WAITING(这个在前面也有所讨论，但现在的情况是“有索引”。注意书中实际也只查到一行锁记录处于waiting状态，与以下结果一致)。注意“heap no 3”，刚好与加的第二把锁对应：



当插入id = 19这条记录时同样需要添加**间隙锁**(插入意向锁锁定间隙(10,20))，与id = 10这条记录的间隙锁（第4锁）发生了冲突，

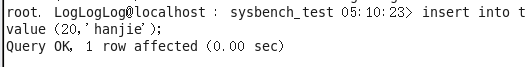


所以看到申请X锁、间隙锁、插入意向锁时状态是WAITING(实际也只查到一行锁，注意等的锁"heap no 2"，刚好对应上面第4步加的锁)。



也就是说：插入id=9和插入id=19阻塞的原因不一样，前者插入在id=10之前，被它的Next-Key Lock锁阻塞，后者插入id=10之后，被索引记录区间（10,20）的间隙锁阻塞。

而插入id = 20这条记录则不会被阻塞，因为它不在间隙锁范围内(前面第4锁锁定间隙是开区间(10,20)，不影响)。



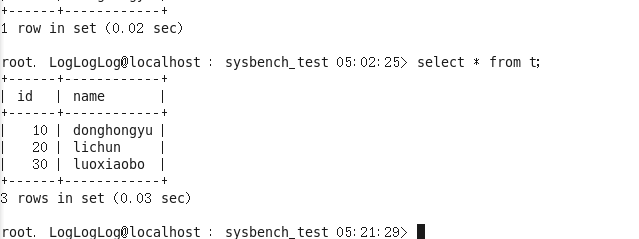
**(2) where条件包含索引字段和非索引字段**

此情况与where条件是普通索引字段的情况相同，这里不再演示。

**3．唯一索引**

**（1）where条件是索引字段**

根据前面的操作，先还原使表t没有重复id：

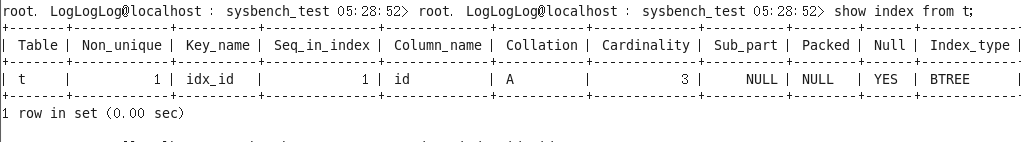


mysql> drop index idx\_id on t;



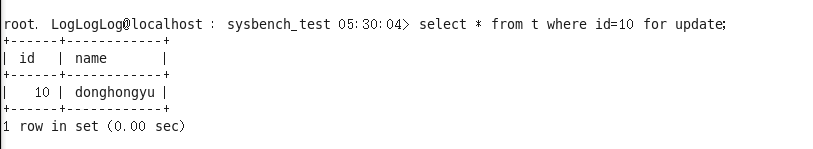
mysql> create **unique index** idx\_id on t (id);





mysql> begin;

mysql> select \* from t where **id = 10** for update;



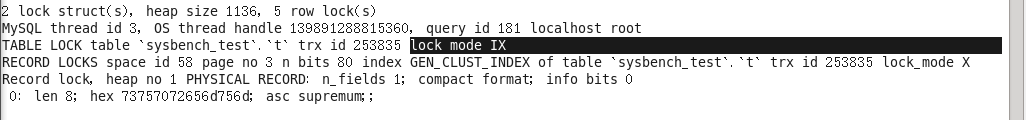
这里与20.2.2节中“where条件是主键字段”的加锁情况相同，表**无显式主键则会把唯一索引作为主键**，因为是主键，所以不能再插入id = 10这条记录了。因此，这里也**不需要间隙锁**。

这里共有3个锁：

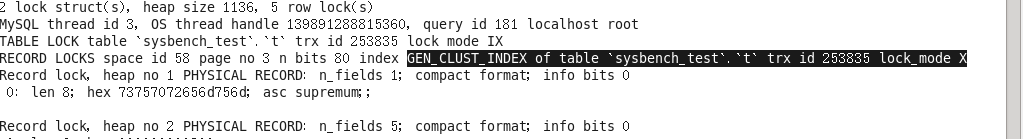
① 对表添加IX锁(自然跟普通索引的一样)。



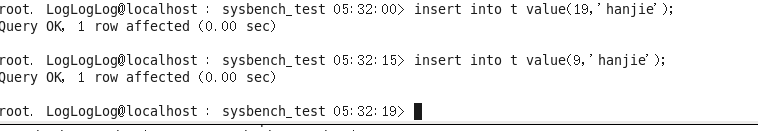
② 对id = 10对应的索引加X记录锁：



③ 对索引对应的聚集索引添加X记录锁(跟普通索引的一样)。



由此可见，并没有加间隙锁。于是前面不允许的插入操作这里都是可以的：



注意这里明显比前面20.2.2节" where条件是主键字段"的限制要宽松。

**（2）where条件包含索引字段和非索引字段**

此情况与where条件是唯一索引字段的情况相同，这里不再演示。

## 20.2.4　REPEATABLE-READ隔离级别+表有显示主键和索引

● 表有显式主键和普通索引：

■ 不带where条件。

■ where条件是普通索引字段。

■ where条件是主键字段。

■ where条件同时包含普通索引字段和主键字段。

● 有显式主键和唯一索引：

■ 不带where条件。

■ where条件是唯一索引字段。

■ where条件是主键字段。

■ where条件同时包含唯一索引字段和主键字段。

**1．表有显式主键和普通索引**

注：把原表t删掉，然后重新建表，要求有主键id和索引



mysql> show create table t \G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

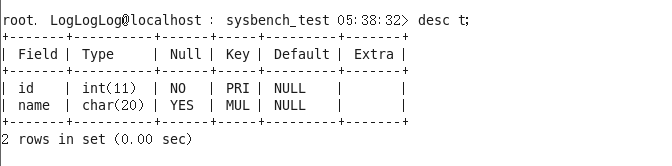
Table: t

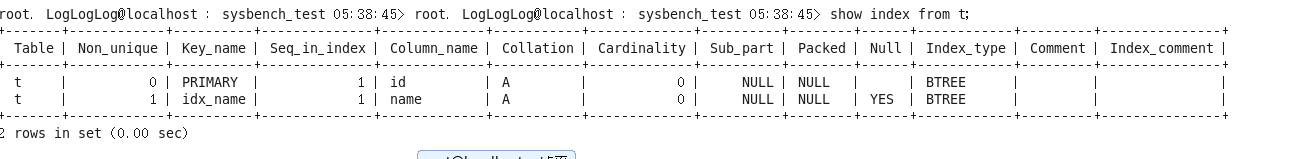
Create Table: CREATE TABLE `t` (

`id` int(11) NOT NULL,

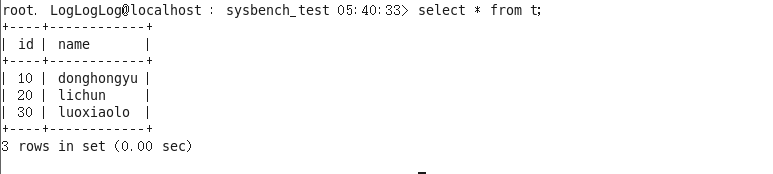
`name` char(20) DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8





重新写入数据：



注意：以下实验另建事务。

**（1）不带where条件**

mysql> begin;

mysql> select \* from t for update;

这里加锁就比较多了，加锁顺序如下：

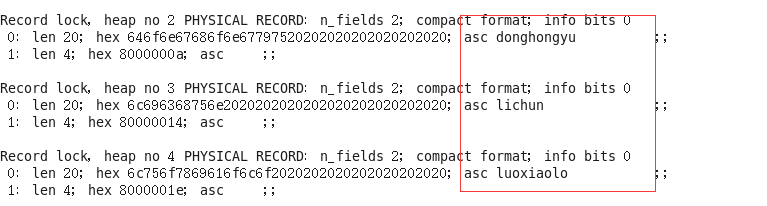
① 对表添加IX锁。



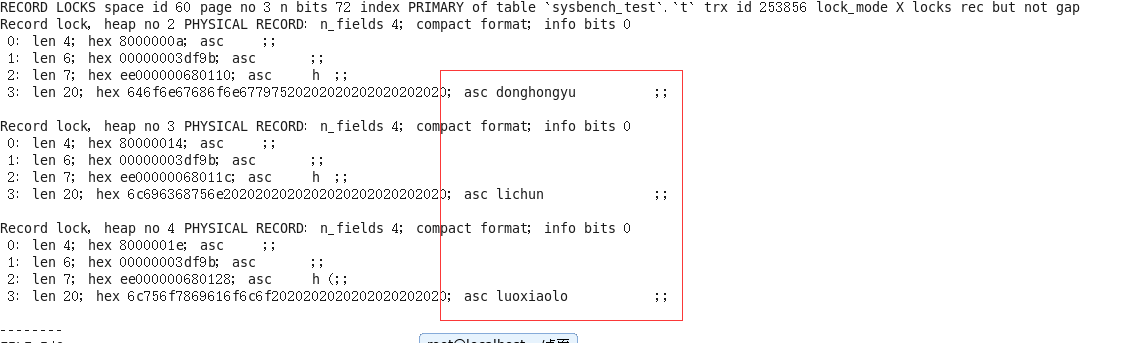
② 对supremum pseudo-record添加Next-Key Lock锁。



③ 对索引添加Next-Key Lock锁。



④ 对主键索引添加X记录锁。



**（2）where条件是普通索引字段**

mysql> select \* from t **where name = 'donghongyu'** for update;

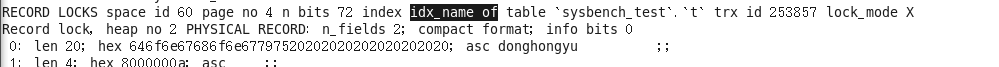
此情况与20.2.3节（无主键有索引）中的“where条件是索引字段”情况相同，只因索引字段不同，**加锁的是name字段**而已。

注：

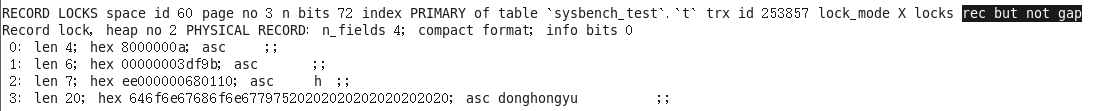
① 对表添加IX锁。



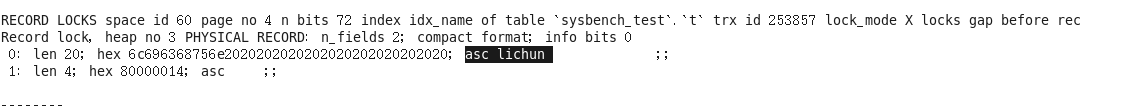
② 对name = 'donghongyu'对应的索引加Next-Key Lock锁：



③ 对name字段的索引对应的id主键(索引)加记录锁



④ 对name字段的索引区间("donghongyu","lichun")加间隙锁：



**（3）where条件是主键字段**

此情况与20.2.2节(有主键无索引)中的“where条件是主键字段”情况相同，这里不再演示。

**（4）where条件同时包含普通索引字段和主键字段**

在这种情况下，要**看SQL执行计划**用到的是主键索引还是普通索引：

如果是主键索引(也就回到本节第(3)种情况)，则与20.2.2节(有主键无索引)中的“where条件是主键字段”情况相同；

如果是普通索引(也就回到本节第(2)种情况)，则与20.2.3节(无主键有索引)中的“where条件是索引字段”情况相同，只因索引字段名称不同，**加锁的是name字段**而已。

## 20.2.5　READ-COMMITTED隔离级别+表无显式主键和索引

注：修改全局事务隔离级别为提交读，并且全部启用新的会话，以防出错。

set @@session.transaction\_isolation = 'read-committed';

insert into t values(10,'donghongyu'),(20,'lichun'),(30,'luoxiaolo');

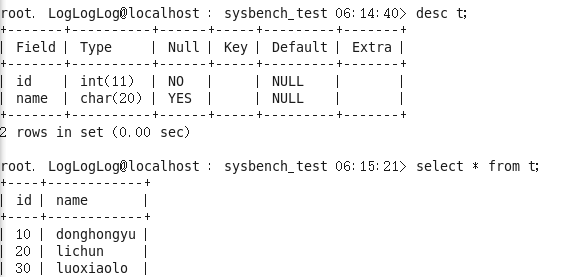
CREATE TABLE `t` (

`id` int(11) NOT NULL,

`name` char(20) DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8

按前面方法重置表t，无主键和索引：

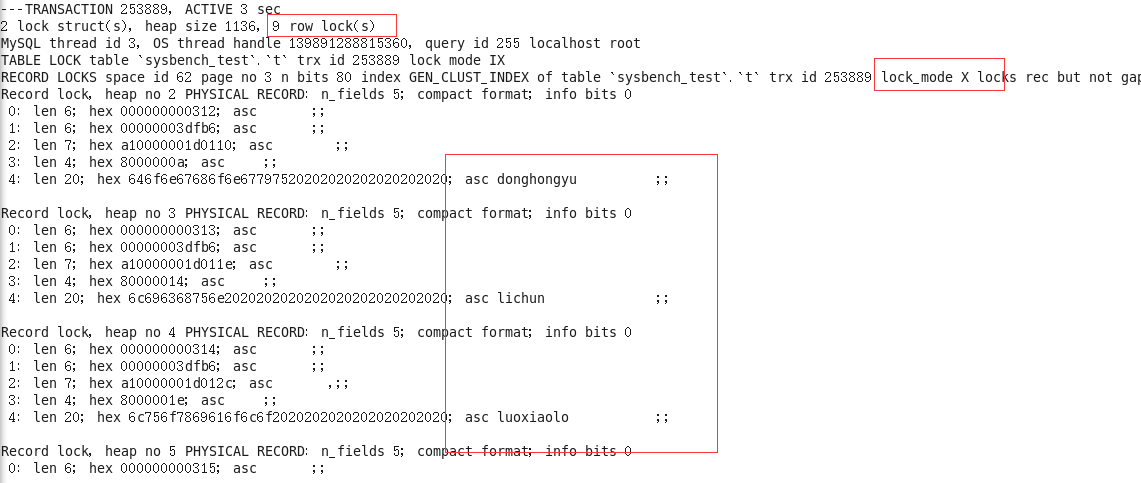


**(1)不带where条件的情况。**

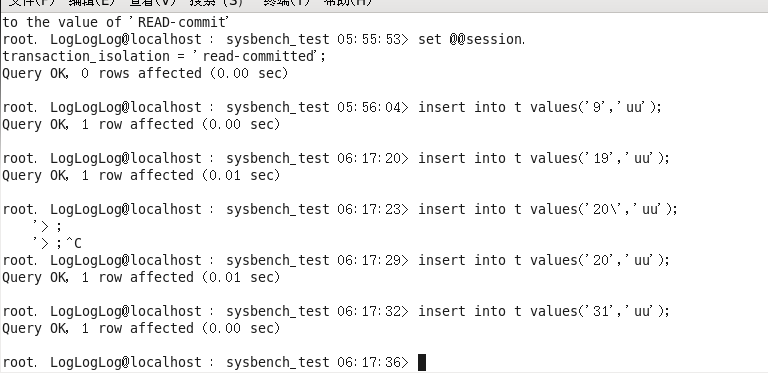
mysql> begin;

mysql> select \* from t for update;

与REPEATABLE-READ隔离级别不同，在READ-COMMITTED隔离级别下添加的锁**都是X记录锁，而不是间隙锁**。



注：另开会话可以插入数据也证实了加的是普通记录锁，不是间隙锁：

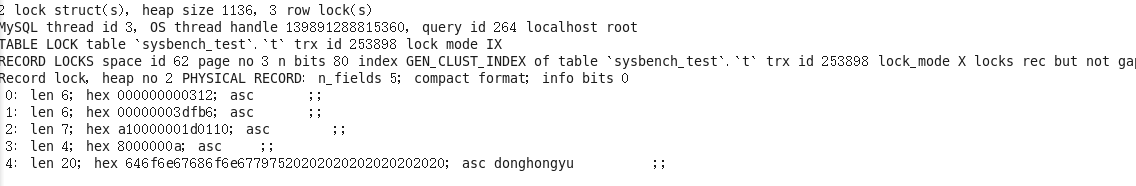


**(2) 带where条件的情况**

mysql> select \* from t where id = 10 for update;

这里与REPEATABLE-READ隔离级别也有所不同，在REPEATABLE-READ隔离级别下会**对不满足id = 10的记录添加X锁**，而在READ-COMMITTED隔离级别下则**不会**。

注：



其实是因为在REPEATABLE-READ隔离级别(对比20.2.1节“(2)带where条件的情况”)下为**避免发生幻读**，对不满足的记录释放锁的时间会晚一些（**持有锁到事务结束**），而在READ-COMMITTED隔离级别下会提前一些时间。

## 20.2.6 READ-COMMITTED隔离级别+表有显式主键无索引

这里分如下几种情况：

● 不带where条件。

此情况与20.2.5节(提交读-无主键无索引)中的“不带where条件”的加锁情况相同，这里不再演示。既然不带where条件因此跟无主键无索引时一样。

● where条件是主键字段。

此情况与20.2.2节(可重复读-有主键无索引)中的“where条件是主键字段”的加锁情况相同，这里不再演示。就是跟可重复读级别一样加锁。

● where条件包含主键字段和非主键字段。

与20.2.2节(可重复读-有主键无索引)中的“where条件包含主键字段和非主键字段”的加锁情况相同，这里不再演示。就是跟可重复读级别一样加锁。

## 20.2.7　READ-COMMITTED隔离级别+表无显式主键有索引

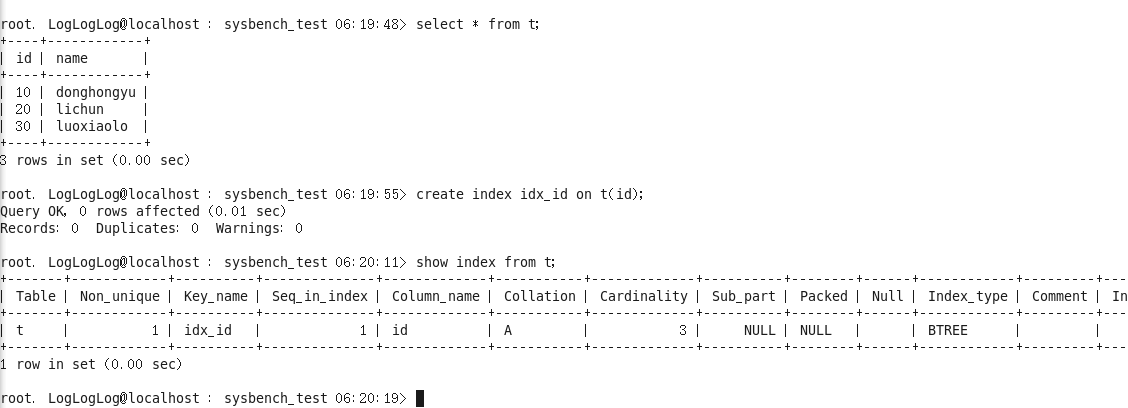
**1. 不带where条件**

此情况与20.2.5节(无主键无索引)中的“不带where条件”的加锁情况相同，这里不再演示。

**2. 普通索引**

**（1）where条件是索引字段**

假设沿着前面20.2.5节的无索引无主键的表t继续，重新按照20.2.3节为表t并添加索引，并且恢复原来的数据：



然后开启事务：

mysql> begin;

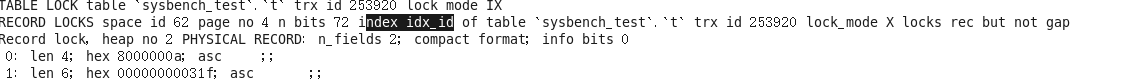
mysql> select \* from t where id = 10 for update;

当where条件是普通索引字段时，加锁顺序如下：

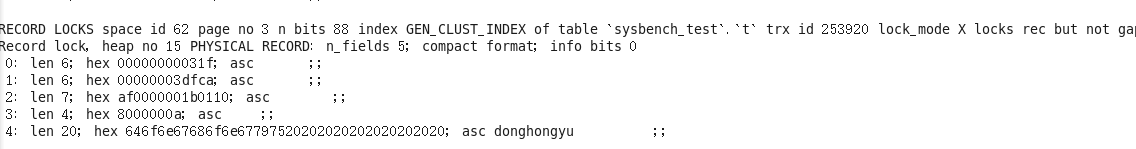
① 对表添加IX锁。



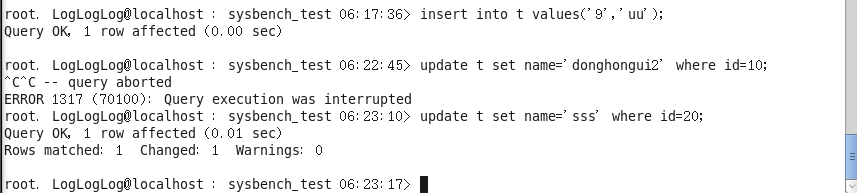
② 对id = 10对应的索引添加X记录锁。



③ 对索引对应的聚集索引添加X记录锁。



相对于REPEATABLE-READ隔离级别(20.2.3节)，**没有了间隙锁**。另开会话操作可发现只是不能修改id=10这一行，但插入和修改其他行被允许(对比20.2.3节的有关讨论)：



显然容易造成幻读。总之，提交读隔离级别造成幻读的原因就在于没有像可重复读级别那样加间隙锁。

**3．唯一索引**

（1）where条件是索引字段

此情况与20.2.3节(无主键有索引)中的“where条件是索引字段”的加锁情况相同，这里不再演示。

（2）where条件包含索引字段和非索引字段

此情况与20.2.3节(无主键有索引)中的“where条件是索引字段”的加锁情况相同，这里不再演示。

## 20.2.8　READ-COMMITTED隔离级别+表有显式主键和索引

这里分如下几种情况。

● 有显式主键和普通索引：

■ 不带where条件。

■ where条件是普通索引字段。

■ where条件是主键字段。

■ where条件同时包含普通索引字段和主键字段。

● 有显式主键和唯一索引：

■ 不带where条件。

■ where条件是唯一索引字段。

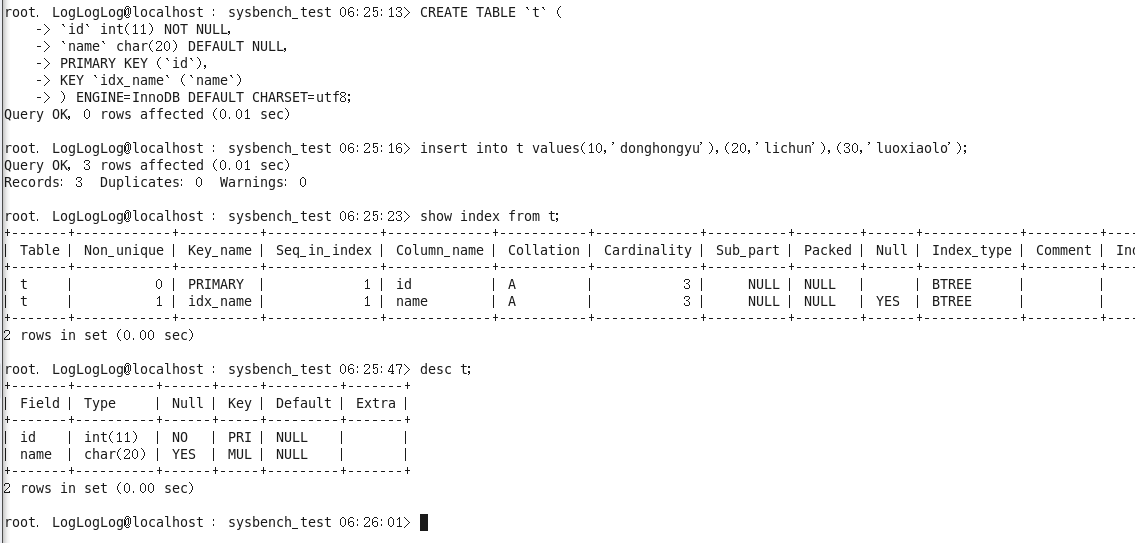
■ where条件是主键字段。

■ where条件同时包含唯一索引字段和主键字段。

**1．表有显式主键和普通索引**

按20.2.4节开始重建表t

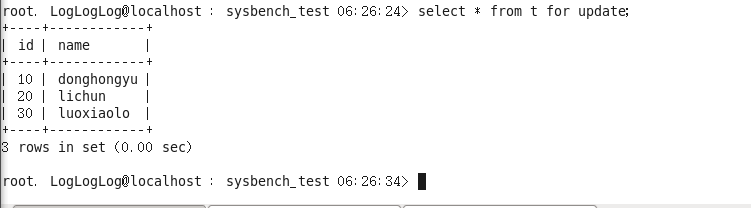
记得把表数据填回：



**（1）不带where条件**

mysql> begin;

mysql> select \* from t for update;



此时添加的锁全部是记录锁，并没有间隙锁。3种锁结构，共6个行锁：



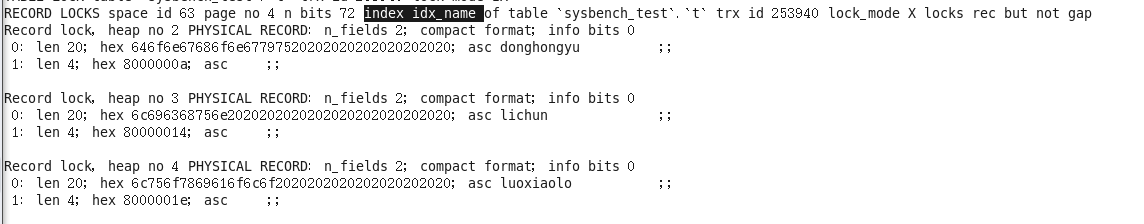
加锁顺序如下：

① 对表添加IX锁。

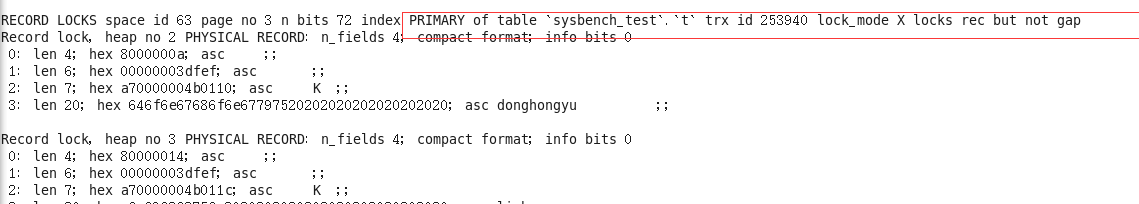


对索引添加Next-Key Lock锁。

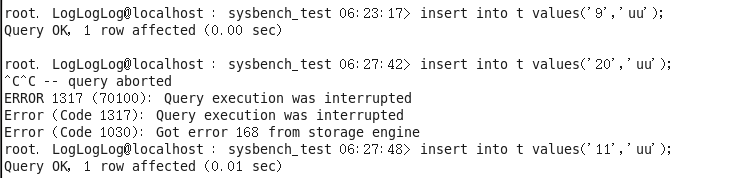
② 加索引的记录锁



③ 对主键索引添加X记录锁。



根据以上实际所得结果，以及前面一路过来的讨论可知，提交读级别有着更宽松的事务同步读写限制(在另一会话操作)：



只有数据行真正被锁定。

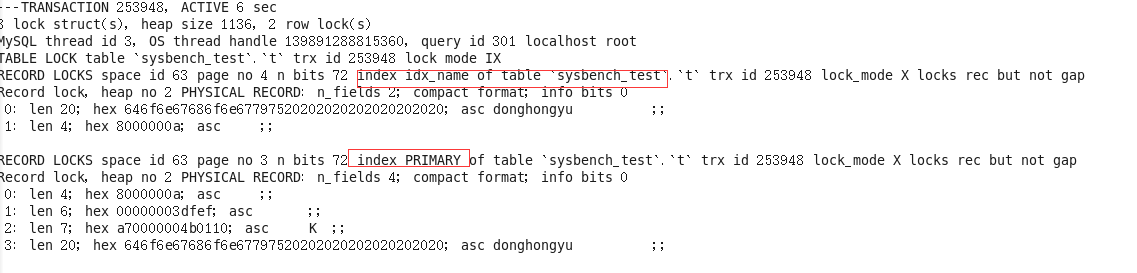
**（2）where条件是普通索引字段**

mysql> begin;

mysql> select \* from t where name = 'donghongyu' for update;

此情况与20.2.7节中的“where条件是索引字段”情况相同，只不过因索引字段不同，加锁的是name字段。

注：锁定的就是name = 'donghongyu'的索引和主键。



注：经过以上一系列“组合式”讨论和操作，我猜作者写到这里也有点郁闷。不如自己做个总结吧(欢迎补充)。

总结：我们大概可以获得如下几点结论：

1. 可重复读与提交读的加锁区别：可重复读级别通过加间隙锁防止幻读的出现。提交读不加间隙锁而仅仅加“NOT GAP”的记录锁，因此其他事务只是不允许修改锁定的记录，

2. 不加where条件与加where条件：锁定的范围自然不同。

3. 主键/索引的有无：记住书中所说(281页)“在MySQL中记录锁都是添加在索引上的，即使表中没有索引，也会在默认创建的聚集索引上添加记录锁。”，自然就容易理解了。

Thank you Teacher Chen!!! Good bye Teacher Chen!!!