39 | 自增主键为什么不是连续的

第4篇提到过自增主键，由于自增主键可以让主键索引尽可能的保持递增顺序插入，避免了页分裂，因此索引更紧凑

业务设计依赖与自增主键的连续性，也就是说，这个设计自增主键是连续的，但实际上，这样的假设是错误的，因为自增主键不能保证是连续的。为什么自增主键会出现“空洞”？

创建一个表t，其中id是自增主键索引，c是唯一索引

CREATE TABLE `t` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`c` int(11) DEFAULT NULL,

`d` int(11) DEFAULT NULL,

PRIMARY KEY (`id`),

UNIQUE KEY `c` (`c`)

) ENGINE=InnoDB;

**自增主键存在哪儿？**

这个空表t里执行insert into t value(null,1,1)；插入一行数据，在执行show create table命令，就可以看到如图所示的结果

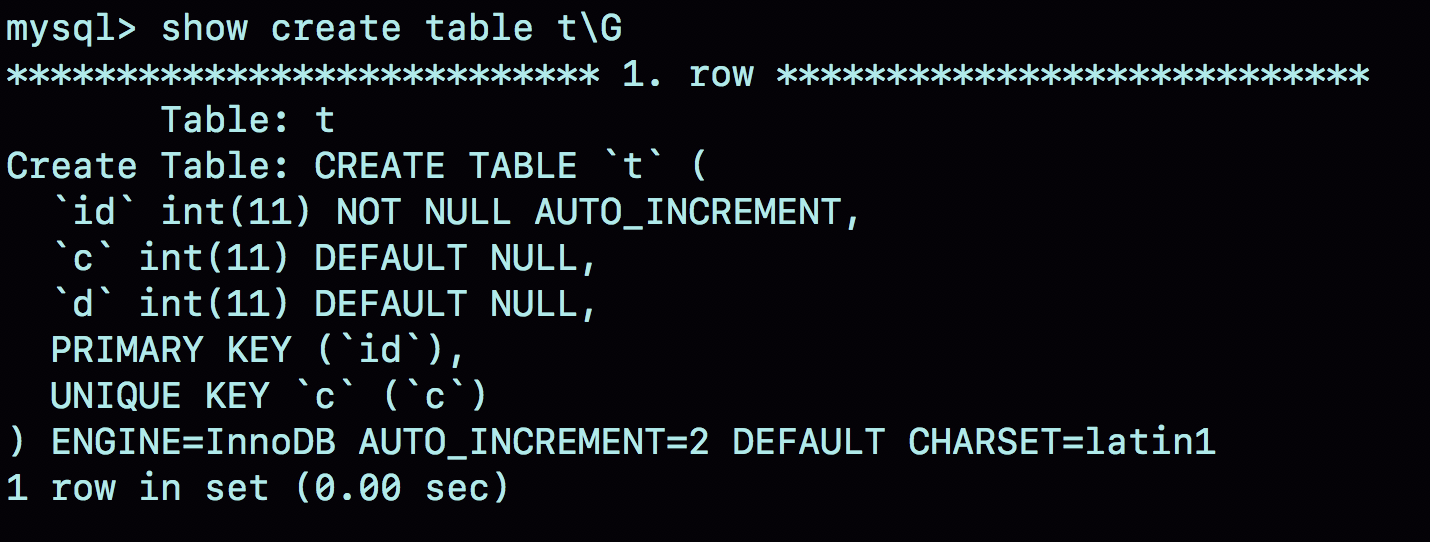


图1 自动生成的AUTO\_INCREMENT值

表里定义里出现了一个AUTO\_INCREMENT=2,表示下一次插入数据时，如果需要自动生成自增值，会生成id=2.

其实，这个输出结果很容易引起这样的误解，自增值是保存在表结构定义里的。实际上**，表的结构定义语句存放在后缀名为.frm的文件中，但是并不会保存自增值。**

不同的引擎对于自增值保存策略不同

·MyISAM引擎的自增值保存在数据文件中

·InnoDB 引擎的自增值，其实是保存在了内存中，并且到了MySQL 8.0版本后，才有了“自增值持久化”的能力，也就是才实现了“如果发生重启，表的自增值可以恢复为MySQL重启前的值”具体情况是

·MySQL 5.7及之前的版本，自增值保存在内存里，并没有持久化，每次重启后，第一次打开表的时候，都会去找自增值的最大值max（id），然后将max(id)+1作为这个表当前的自增值

·举例来说，如果一个表当前数据行里最大的id是10，AUTO\_INCREMENT=11,此时，删除id=10的行，AUTO\_INCREMENT还是11，但是如果马上重启实例，重启之后这个表的AUTO\_INCREMENT就会变成10，

也就是说，MySQL重启可能会修改一个表的AUTO\_INCREMENT的值

·在MySQL8.0版本，将自增值的变更记录在了redo log中，重启时候依靠redo log恢复重启之前的值。

**自增机制修改**

如果字段id被定义为AUTO\_INCREMENT,再插入一行数据是，自增值的行为如下

1. 如果插入数据时id字段指定为0、null或未指定值，那么把这个表当前的AUTO\_INCREMENT值填到自增字段
2. 如果插入数据时id字段指定了具体的值，就直接使用语句里指定的值

根据要插入值和当前自增值的大小关系，自增值的变更结果也会有不同的，假设，某次要插入的值是X,当前的自增值是Y

1. 如果X< Y,那么这个表的自增值不变
2. 如果X>= Y,就需要把当前自增值修改为新的自增值

**新的自增值生算法是：**从auto\_increment\_offset开始，以auto\_increment\_increment为步长，持续叠加，直到找到第一个大于X的值，作为新的自增值。

其中，auto\_increment\_offset和auto\_increment\_increment是两个系统参数，分别用来表示自增的初始值和步长，默认值都是1.

备注：在一些场景下，使用的就不全是默认值，比如，双M的主备结构里要求双写的时候，可能会设置auto\_increment\_increment=2，让一个库的自增id是奇数，另一个库的自增id是偶数，避免两个库生成的主键发生冲突

当auto\_increment\_offset和auto\_increment\_increment都是1时，新的自增值生成逻辑很简单，就是

1. 如果准备插入的值>=当前自增值，新的自增值就是“准备插入的值+1”
2. 否则，自增值不变

这就引入了我们文章开头的问题，在这两个参数都是设置为1的时候，自增主键id却不能保证是连续的，这是什么原因呢？

**自增值的修改时机**

假设，表t里面已经有了（1,1,1）这条记录，此时插入一条数据命令

insert into t values(null, 1, 1);

这个语句的执行流程

1. 执行器调用InnoDB引擎接口写入一行，传入的这一行的值是（0,1,1）
2. InnoDB发现用户没有指定自增id的值，获取表t当前的自增值2
3. 将传入的行的值改为（2,1,1）
4. 将表的自增值改成3
5. 继续执行插入数据库操作，由于已经存在c=1的记录，所以报Duplicate key error,语句返回

对应的执行流程如下

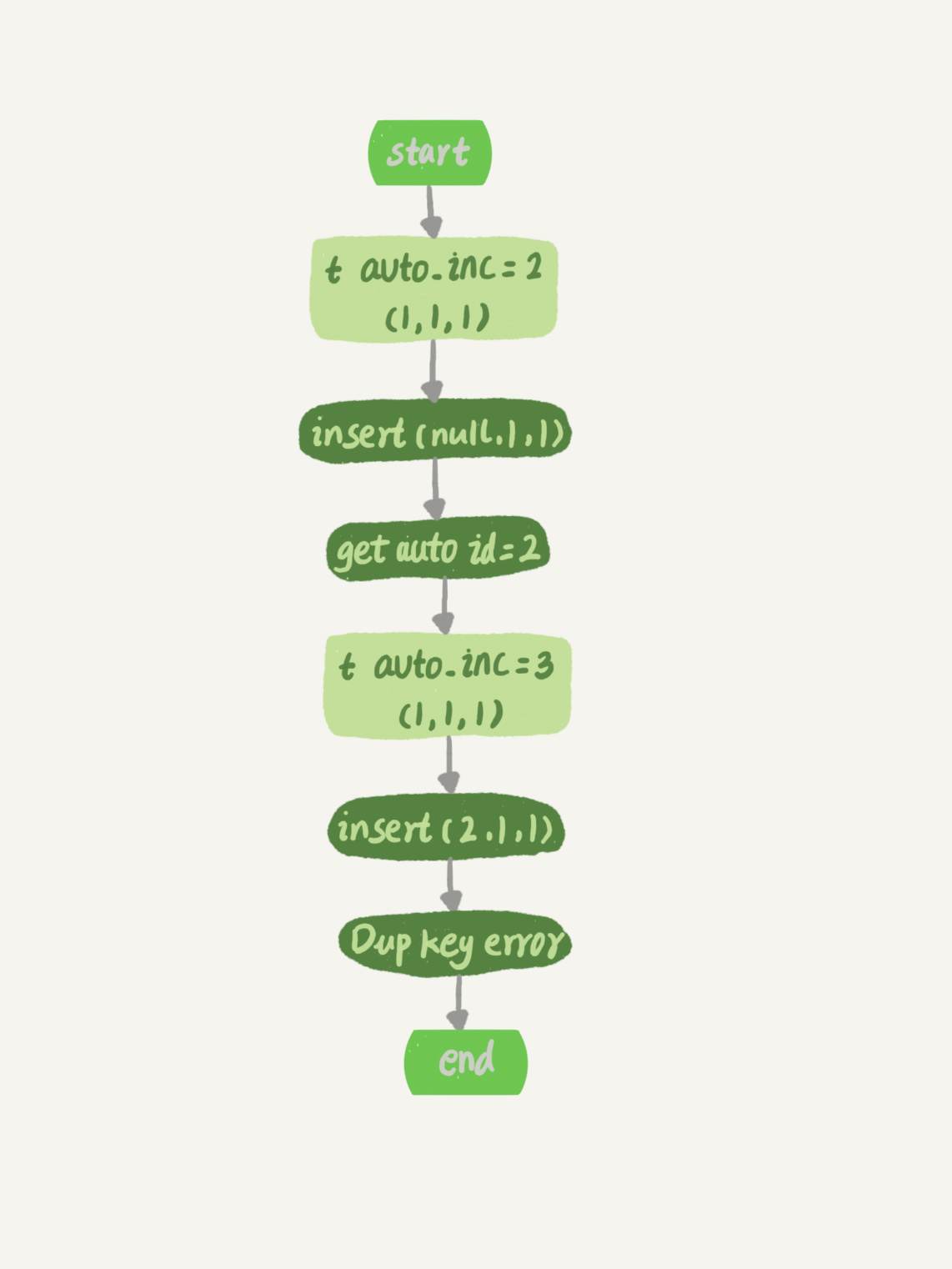


图 2 insert(null,1,1)唯一键冲突

这个表的自增值改成了3，是真正执行插入数据的操作之前，这个语句真正执行的时，因为唯一键c冲突，所以id=2这一行并没有插入成功，但也没有将自增值再改回去。

所以，这之后，再插入新数据行时，拿到的自增id就是3，出现了自增主键不连续的情况

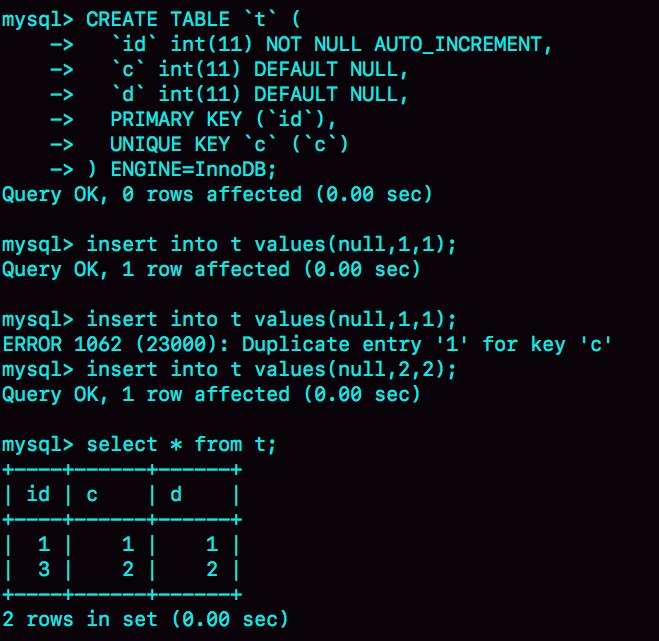


图 3 一个自增主键id不连续的复现步骤

这个操作序列复现了一个自增主键id不连续的现场（没有id=2的行），可见，

**唯一键冲突是导致自增主键id不连续的第一种原因**

同样，事务回滚也会产生类似的现象，就是第二种原因

下面这个语句序列可以构造不连续的自增id,

insert into t values(null,1,1);

begin;

insert into t values(null,2,2);

rollback;

insert into t values(null,2,2);

//插入的行是(3,2,2)

为什么出现唯一键冲突或者回滚时，MySQL没有把表t的自增值改回去呢？如果把表t的当前自增值从3改回2，再插入新数据时，不就可以生成id=2的一行数据了吗？

MySQL设计是为了提升性能，**自增值为什么不能回退**

假设有两个并行执行的事务，在申请自增值的时候，为了避免两个事务申请到相同的自增id,肯定 要加锁，然后顺序申请

1. 假设事务A申请到了id=2,事务B申请到了id=3,那么此时表t的自增值是4，之后继续执行
2. 事务B正确提交了，但事务A出现了唯一键冲突
3. 如果允许事务A把自增id回退，也就是把表t的当前自增值改回2，那么就会出现这样的情况：表里面已经有id=3的行，而当前的自增id值是2.
4. 接下来，继续执行其他事务就会申请到id=2,然后在申请id=3，此时，就会出现插入语句报错“主键冲突”

而为了解决这个主键冲突，有两种办法：

1. 每次申请id之前，先判断表里面是否已经存在这个id。如果存在，就跳过这个id,但是，这个方法的成本很高。因为，本来申请id是一个很快的动作，现在还要再主键索引数上判断id是否存在
2. 把自增id的锁范围扩大，必须等到一个事务执行完成并提交，下一个事务才能再申请自增id,这个方法的问题，就是锁粒度太大，系统并发能力大大下降。

可见，这两个方法导致性能问题，造成了这些麻烦的罪魁祸首，就是我们假设的这个“允许自增id回退”的前提导致的

因此，InnoDB放弃了这个设计，语句执行失败也不回退自增id,也正式因为这样，所以才保证了自增id是递增的，但不保证是连续的

**自增锁的优化**

自增id并不是一个事务锁，而是每次申请完就马上释放，一遍允许别的事务在申请，其实，在MySQL5.1版本之前，并不是这样的

MySQL5.0版本的时候，自增锁的范围是语句级别，如果一个语句申请了一个表自增锁，这个锁会等语句执行结束后才释放，这样设计会影响并发度。

MySQL5.1.22版本引入了一个新策略，新增参数innodb\_autoinc\_lock\_mode,默认值是1.

1. 这个参数的值被设置为0时，表示采用之前MySQL5.0版本的策略，即语句执行结束后才释放
2. 这个参数的被设置为1时

·普通insert语句，自增锁在申请之后马上释放；

·类似insert...select这样的批量插入数据的语句，自增锁还是要等语句结束后才释放

1. 这个参数的值被设置为2时，所有的申请自增主键的动作都是申请后就释放

**为什么默认设置下，insert...select要使用语句级别的锁？为什么这个参数的默认值不是2？**

答案是：这么设计还是为了数据的一致性



图 4 批量插入数据的自增锁

这个例子里，往表t1中插入了4行数据，然后创建了一个相同的结构的表t2,然后两个session同时执行想表t2中插入数据的动作

如果sessionB申请了自增值以后马上就释放自增锁，那么就可能出现这样的情况

·sessionB 先插入两个记录（1，1，1），（2,2,2）

·然后，sessionA来申请自增id得到id=3，插入（3,5,5）

·之后，sessionB 继续执行，插入两条记录（4,3,3），（5,4,4）

可能会说，这没关系吧，但是，如果现在在binlog\_format=statement，可以设想下，binlog会怎么记录？

由于两个session是同时执行插入数据命令的，所以binlog里面对表t2的更新日志只要两种情况：要么先记sessionA的，要么先记sessionB的

但不论哪一种，这个binlog拿去从库执行，或者用来恢复临时实例，备库和临时实例里面，sessionB这个语句执行出来，生成的结果里面，id都是连续的，此时，这个库发生了数据不一致。

出现这个问题什么原因呢？

这是因为原库sessionB的insert语句，生成的id不连续。这个不连续的id,用statement格式的binlog来串行执行，是执行不出来的

要解决这个问题，有两种思路

1. 一种思路是，让原库的批量插入语句，固定生成连续的id值。所以，自增锁直到语句执行结束才释放，就是为了达到这个目的。
2. 另一种思路是，在binlog里面把插入数据的操作都如实记录进来，到备库执行的时候，步子依赖于自增主键去生成。这种情况，其实是innodb\_automic\_lock\_mode设置为2,同时binlog\_format设置为row.

因此，**在生产上，尤其是insert...select这种批量插入数据的场景，从并发插入数据性能的考虑，建议这样设置：innodb\_automic\_lock\_mode=2，并且binlog\_format=row**。这样做，既能提升并发，有不会出现数据一致性问题。

**这里说的批量插入数据，包括insert...select、replace...select和load data语句**

在普通的insert语句里面包含多个value值的情况下，即使innodb\_automic\_lock\_mode设置为1，也不等于语句执行完才释放锁，因为这类语句在申请自增id的时候，是可以精确计算出需要多少个id的，然后一次性申请，申请完成后锁就可以释放了。

批量插入数据的语句，之所以需要这么设置，是因为“不知道要预先先申请多少个id”

既然预先不知道要申请多少个自增id，那么一种直接的想法就是需要一个时申请一个，但是如果一个select...insert语句要插入10万行数据，按照这个逻辑就要申请10万次，显然，这种申请自增id的策略，在大批量插入数据的情况下，不但速度慢，还会影响并发插入的性能。

因此，对于批量插入数据的语句，MySQL有一个批量申请自增id的策略

1. 语句执行过程中，第一次申请自增id,会分配1个
2. 1个用完以后，这个语句第二次申请自增id,会分配2个。
3. 2个用完以后，还是这个语句，第三次申请自增id，会分配4个
4. 依次类推，同一个语句去申请自增id，每次申请到自增id个数都是上一次的两倍

举个例子，看看这个例子

insert into t values(null, 1,1);

insert into t values(null, 2,2);

insert into t values(null, 3,3);

insert into t values(null, 4,4);

create table t2 like t;

insert into t2(c,d) select c,d from t;

insert into t2 values(null, 5,5);

Insert ...select ，实际上往表t2中插入了4行数据，但是，这4行数据是分三次申请的自增id，第一次申请到了id=1,第二次被分配了id=2和id=3,第三次被分配到了id=4到id=7.

由于这条语句实际值用了4个id,所以id=5到7就被浪费了。字后，在执行insert into t2 values(null,5,5)实际上插入的数据是（8，5，5）

**这就是主键id出现自增id不连续的第三种原因**