<https://mp.weixin.qq.com/s/RUZHl9cU642FO-edS4utLA>

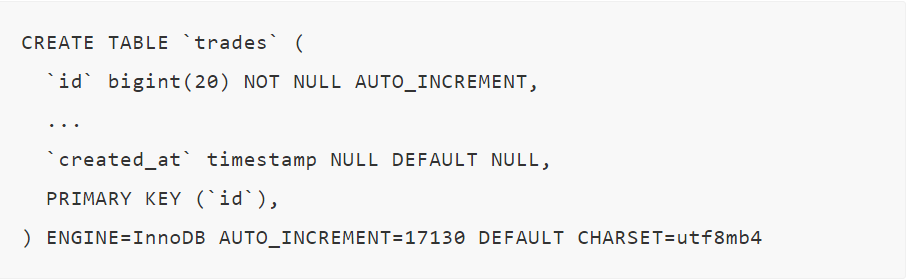
# 为什么 MySQL 的自增主键不单调也不连续

当我们在使用关系型数据库时，主键（Primary Key）是无法避开的概念，主键的作用就是充当记录的标识符，我们能够通过标识符在一张表中定位到唯一的记录，作者在[为什么总是需要无意义的 ID](http://mp.weixin.qq.com/s?__biz=MzU5NTAzNjc3Mg==&mid=2247484059&idx=1&sn=911fd4932294a6027a32bdd735a2550b&chksm=fe795d90c90ed486f97b50ec10556ab9674ebfe695b3c5933cdd10a712b22f6057edcc240cfa&scene=21" \l "wechat_redirect" \t "https://mp.weixin.qq.com/_blank)曾经介绍过为什么不应该使用有意义的字段来充当唯一标识符，感兴趣的读者可以了解一下。

在关系型数据库中，我们会选择记录中多个字段的最小子集作为该记录**在表中的唯一标识符**[^1]，根据关系型数据库对主键的定义，我们既可以选择单个列作为主键，也可以选择多个列作为主键，但是主键在整个记录中必须存在并且唯一。最常见的方式当然是使用 MySQL 默认的自增 ID 作为主键，虽然使用其他策略设置的主键也是合法的，但是不是通用的以及推荐的做法。



MySQL 中默认的 AUTO\_INCREMENT 属性在多数情况下可以保证主键的连续性，我们通过 show create table 命令可以在表的定义中能够看到 AUTO\_INCREMENT 属性的当前值，当我们向当前表中插入数据时，它会使用该属性的值作为插入记录的主键，而每次获取该值也都会将它加一。



在很多开发者的认知中，MySQL 的主键都应该是单调递增的，但是在我们与 MySQL 打交道的过程中会遇到两个问题，首先是记录的主键并不连续，其次是可能会创建多个主键相同的记录，我们将从以下的两个角度回答 MySQL 不单调和不连续的原因：

* 较早版本的 MySQL 将 AUTO\_INCREMENT 存储在内存中，实例重启后会根据表中的数据重新设置该值；
* 获取 AUTO\_INCREMENT 时不会使用事务锁，并发的插入事务可能出现部分字段冲突导致插入失败；

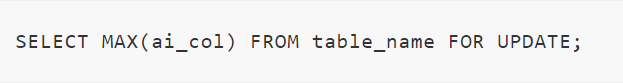
需要注意的是，我们在这篇文章中讨论的是 MySQL 中最常见的 InnoDB 存储引擎，MyISAM 等其他引擎提供的 AUTO\_INCREMENT 实现原理不在本文的讨论范围中。

## **删除记录**

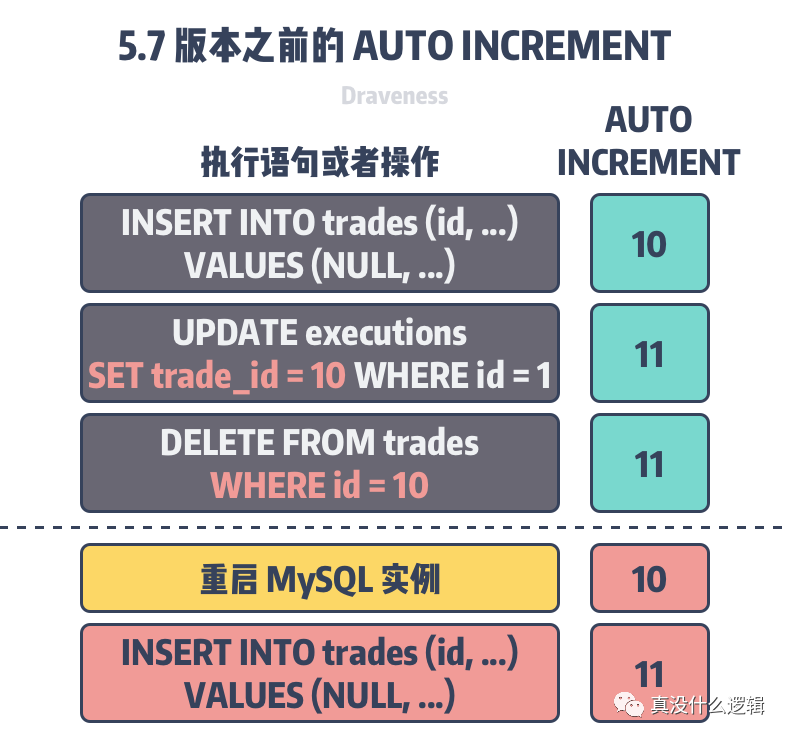
AUTO\_INCREMENT 属性虽然在 MySQL 中十分常见，但是在较早的 MySQL 版本中，它的实现还比较简陋，InnoDB 引擎会在内存中存储一个整数表示下一个被分配到的 ID，当客户端向表中插入数据时会获取 AUTO\_INCREMENT 值并将其加一。



因为该值存储在内存中，所以在每次 MySQL 实例重新启动后，当客户端第一次向 table\_name 表中插入记录时，MySQL 会使用如下所示的 SQL 语句查找当前表中 id 的最大值，将其加一后作为待插入记录的主键，并作为当前表中 AUTO\_INCREMENT 计数器的初始值[^2]。



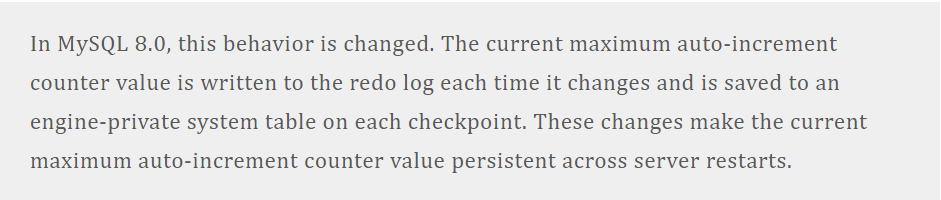
如果让作者实现 AUTO\_INCREMENT，在最开始也会使用这种方法。不过这种实现虽然非常简单，但是如果**使用者不严格遵循关系型数据库的设计规范**，就会出现如下所示的数据不一致的问题：



因为重启了 MySQL 的实例，所以内存中的 AUTO\_INCREMENT 计数器会被重置成表中的最大值，当我们再向表中插入新的 trades 记录时会重新使用 10 作为主键，主键也就不是单调的了。在新的 trades 记录插入之后，executions **表中的记录就错误的引用了新的** trades，这其实是一个比较严重的错误。

然而这也不完全是 MySQL 的问题，如果我们严格遵循关系型数据库的设计规范，使用外键处理不同表之间的联系，就可以避免上述问题，因为当前 trades 记录仍然有外部的引用，所以外键会禁止 trades 记录的删除，不过多数公司内部的 DBA 都不推荐或者禁止使用外键，所以确实存在出现这种问题的可能。

然而在 MySQL 8.0 中，AUTO\_INCREMENT 计数器的初始化行为发生了改变，每次计数器的变化都会写入到系统的重做日志（Redo log）并在每个检查点存储在引擎私有的系统表中[^3]。



当 MySQL 服务被重启或者处于崩溃恢复时，它可以从持久化的检查点和重做日志中恢复出最新的 AUTO\_INCREMENT 计数器，避免出现不单调的主键也解决了这里提到的问题。

## **并发事务**

为了提高事务的吞吐量，MySQL 可以处理并发执行的多个事务，但是如果并发执行多个插入新记录的 SQL 语句，可能会导致主键的不连续。如下图所示，事务 1 向数据库中插入 id = 10 的记录，事务 2 向数据库中插入 id = 11 和 id = 12 的两条记录：



不过如果在最后事务 1 由于插入的记录发生了唯一键冲突导致了回滚，而事务 2 没有发生错误而正常提交，在这时我们会发现当前表中的主键出现了不连续的现象，后续新插入的数据也不再会使用 10 作为记录的主键。

这个现象背后的原因也很简单，虽然在获取 AUTO\_INCREMENT 时会加锁，但是该锁是语句锁，它的目的是保证 AUTO\_INCREMENT 的获取不会导致线程竞争，而不是保证 MySQL 中主键的连续[^4]。

上述行为是由 InnoDB 存储引擎提供的 innodb\_autoinc\_lock\_mode 配置控制的，该配置决定了获取 AUTO\_INCREMENT 计时器时需要先得到的锁，该配置存在三种不同的模式，分别是传统模式（Traditional）、连续模式（Consecutive）和交叉模式（Interleaved）[^5]，其中 MySQL 使用连续模式作为默认的锁模式：

* 传统模式 innodb\_autoinc\_lock\_mode = 0；

在包含 AUTO\_INCREMENT 属性的表中插入数据时，**所有**的 INSERT 语句都会获取**表级别**的 AUTO\_INCREMENT 锁，该锁会在当前语句执行后释放；

* 连续模式 innodb\_autoinc\_lock\_mode = 1；

INSERT ... SELECT、REPLACE ... SELECT  以及 LOAD DATA 等批量的插入操作需要获取**表级别**的 AUTO\_INCREMENT 锁，该锁会在当前语句执行后释放；

**简单的插入语句**（预先知道插入多少条记录的语句）只需要获取获取 AUTO\_INCREMENT 计数器的互斥锁并在获取主键后直接释放，不需要等待当前语句执行完成；

* 交叉模式 innodb\_autoinc\_lock\_mode = 2；

所有的插入语句都不需要获取**表级别**的 AUTO\_INCREMENT 锁，但是当多个语句插入的数据行数不确定时，可能存在分配相同主键的风险；

这三种模式都不能解决 MySQL 自增主键不连续的问题，想要解决这个问题的终极方案是串行执行所有包含插入操作的事务，也就是使用数据库的最高隔离级别 —— 可串行化（Serialiable）。当然直接修改数据库的隔离级别相对来说有些简单粗暴，基于 MySQL 或者其他存储系统实现完全串行的插入也可以保证主键**在插入时的连续**，但是仍然不能避免删除数据导致的不连续。

## **总结**

早期 MySQL 的主键既不是单调的，也不是连续的，这些都是在当时工程上做出的一些选择，如果严格地按照关系型数据库的设计规范，MySQL 最初的设计造成问题的概率也比较低，只有当被删除的主键被外部系统引用时才会影响数据的一致性，但是今天使用方式的不同却增加出错的可能性，而 MySQL 也在 8.0 中持久化了AUTO\_INCREMENT 以避免该问题的出现。

MySQL 中不连续的主键又是一个**工程设计向性能低头**的例子，牺牲主键的连续性来支持数据的并发插入，最终提高了 MySQL 服务的吞吐量，作者在几年前刚刚使用 MySQL 时就遇到过这个问题，但是当时并没有深究背后的原因，今天重新理解该问题背后的设计决策也是个非常有趣的过程。我们在这里简单总结一下本文的内容，重新回到今天的问题 — 为什么 MySQL 的自增主键不单调也不连续：

* MySQL 5.7 版本之前在内存中存储 AUTO\_INCREMENT 计数器，实例重启后会根据表中的数据重新设置，在删除记录后重启就可能出现重复的主键，该问题在 8.0 版本使用重做日志解决，保证了主键的单调性；
* MySQL 插入数据获取 AUTO\_INCREMENT 时不会使用事务锁，而是会使用互斥锁，并发的插入事务可能出现部分字段冲突导致插入失败，想要保证主键的连续需要串行地执行插入语句；

到最后，我们还是来看一些比较开放的相关问题，有兴趣的读者可以仔细思考一下下面的问题：

* MyISAM 和其他的存储引擎如何存储 AUTO\_INCREMENT 计数器？

* MySQL 中的 auto\_increment\_increment 和 auto\_increment\_offset 是用来做什么的？
* auto\_increment\_offset表示自增长字段从那个数开始，他的取值范围是1 .. 65535
* auto\_increment\_increment表示自增长字段每次递增的量，其默认值是1，取值范围是1 .. 65535