数据库主键设计之思考 PDF转换可能丢失图片或格式，建议阅读原文
https://www.100test.com/kao_ti2020/223/2021_2022__E6_95_B0_E6_8D_AE_E5_BA_93_E4_c102_223557.htm 在我们的数据库设计中，不可逃避的就是数据库表的主键，可能有很多朋友没有深入思考过，主键的设计对整个数据库的设计影响很大，因此我们不得不要重视起来。 主键的必要性: 有些朋友可能不提倡数据库表必须要主键，但在我的思考中，觉得每个表都应该具有主键，不管是单主键还是双主键，主键的存在就代表着表结构的完整性，表的记录必须得有唯一区分的字段，主键主要是用于其他表的外键关联，本记录的修改与删除，当我们没有主键时，这些操作会变的非常麻烦。 主键的无意义性： 我强调主键不应该具有实际的意义，这可能对于一些朋友来说不太认同，比如订单表吧，会有“ 订单编号 ”字段，而这个字段呢在业务实际中本身就是应该具有唯一性，具有唯一标识记录的功能，但我是不推荐采用订单编号字段作为主键的，因为具有实际意义的字段，具有“ 意义更改 ”的可能性，比如订单编号在刚开始的时候我们一切顺利，后来客户说“ 订单可以作废，并重新生成订单，而且订单号要保持原订单号一致 ”，这样原来的主键就面临危险了。因此，具有唯一性的实际字段也代表可以作为主键。因此，我推荐是新设一个字段专门用为主键，此主键本身在业务逻辑上不体现，不具有实际意义。而这种主键在一定程序增加了复杂度，所以要视实际系统的规模大小而定，对于小项目，以后扩展不会很大的话，也查允许用实际唯一的字段作主键的。 主键的选择 我们现在在思考一下，应该采用什么来作表的

主键比较合理，申明一下，主键的设计没有一个定论，各人有各人的方法，哪怕同一个，在不同的项目中，也会采用不同的主键设计原则。 第一：编号作主键 此方法就是采用实际业务中的唯一字段的“ 编号 ”作为主键设计，这在小型的项目中是推荐这样做的，因为这可以使项目比较简单化，但在使用中却可能带来一些麻烦，比如要进行“ 编号修改 ”时，可能要涉及到很多相关联的其他表，就象黎叔说的“ 后果很严重 ”.还有就是上面提到的“ 业务要求允许编号重复时 ”，我们再那么先知，都无法知道业务将会修改成什么? 第二：自动编号主键 这种方法也是很多朋友在使用的，就是新建一个ID字段，自动增长，非常方便也满足主键的原则，优点是：数据库自动编号，速度快，而且是增量增长，聚集型主键按顺序存放，对于检索非常有利.数字型的，占用空间小，易排序，在程序中传递也方便.如果通过非系统增加记录（比如手动录入，或是用其他工具直接在表里插入新记录，或老系统数据导入）时，非常方便，不用担心主键重复问题。 缺点：其实缺点也就是来自其优点，就是因为自动增长，在手动要插入指定ID的记录时会显得麻烦，尤其是当系统与其他系统集成时，需要数据导入时，很难保证原系统的ID不发生主键冲突（前提是老系统也是数字型的）.如果其他系统主键不是数字型那就麻烦更大了，会导致修改主键数据类型了，这也会导致其他相关表的修改，后果同样很严重.就算其他系统也是数字型的，在导入时，为了区分新老数据，可能想在老数据主键前统一加一个“ o ”(old)来表示这是老数据，那么自动增长的数字型又面临一个挑战。 第三：Max加一 由于自动编号存在那些问题，所以有些朋友就采用自己生成，同样是

数字型的，只是把自动增长去掉了，采用在Insert时，读取Max值后加一，这种方法可以避免自动编号的问题，但也存在一个效率问题，如果记录非常大的话，那么Max()也会影响效率的.更严重的是并发性问题，如果同时有两人读到相同的Max后，加一后插入的ID值会重复，这已经是有经验教训的了。 第四：自制加一 考虑Max加一的效率后，有人采用自制加一，也就是建一个特别的表，字段为：表名，当前序列值。这样在往表中插入值时，先从此表中找到相应表的最大值后加一，进行插入，有人可能发现，也可能会存在并发处理，这个并发处理，我们可以采用lock线程的方式来避免，在生成此值的时，先Lock，取到值以后，再unLock出来，这样不会有两人同时生成了。这比Max加一的速度要快多了。但同样存在一个问题：在与其他系统集成时，脱离了系统中的生成方法后，很麻烦保证自制表中的最大值与导入后的保持一致，而且数字型都存在上面讲到的“ o ”老数据的导入问题。因此在“ 自制加一 ”中可以把主键设为字符型的。字符型的自制加一我倒是蛮推荐的，应该字符型主键可以应付很多我们意想不到的情况。 第五：GUID主键 目前一个比较好的主键是采用GUID，当然我是推荐主键还是字符型的，但值由GUID生成，GUID是可以自动生成，也可以程序生成，而且键值不可能重复，可以解决系统集成问题，几个系统的GUID值导到一起时，也不会发生重复，就算有“ o ”老数据也可以区分，而且效率很高，在.NET里可以直接使用System.Guid.NewGuid()进行生成，在SQL里也可以使用NewID()生成。优点是： 同 IDENTITY 列相比，uniqueidentifier 列可以通过 NewID() 函数提前得知新增加的

行 ID，为应用程序的后续处理提供了很大方便。 便于数据库移植，其它数据库中并不一定具有 IDENTITY 列，而 Guid 列可以作为字符型列转换到其它数据库中，同时将应用程序中产生的 GUID 值存入数据库，它不会对原有数据带来影响。 便于数据库初始化，如果应用程序要加载一些初始数据，IDENTITY 列的处理方式就比较麻烦，而 uniqueidentifier 列则无需任何处理，直接用 T-SQL 加载即可。 便于对某些对象或常量进行永久标识，如类的 ClassID，对象的实例标识，UDDI 中的联系人、服务接口、tModel标识定义等。 缺点是： GUID 值较长，不容易记忆和输入，而且这个值是随机、无顺序的。 GUID 的值有 16 个字节，与其它那些诸如 4 字节的整数相比要相对大一些。这意味着如果在数据库中使用 uniqueidentifier 键，可能会带来两方面的消极影响：存储空间增大；索引时间较慢。 我也不是推荐GUID最好，其实在不同的情况，我们都可以采用上面的某一种方式，思考了一些利与弊，也方便大家在进行设计时参考。这些也只是我的一点思考而已，而且可能我知识面限制，会有一些误论在里面，希望大家有什么想法欢迎讨论。 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com