SQL数据库中的主键与外键介绍

| 浏览：5057 | 更新：2013-06-23 22:34

一、什么是主键、外键：

关系型数据库中的一条记录中有若干个属性，若其中某一个属性组(注意是组)能唯一标识一条记录，该属性组就可以成为一个主键比如 ：

学生表(学号，姓名，性别，班级)

其中每个学生的学号是唯一的，学号就是一个主键

用户表(用户名、密码、登录级别)

其中用户名是唯一的, 用户名就是一个主键

上机记录表(卡号,学号,姓名、序列号)

上机记录表中单一一个属性无法唯一标识一条记录，学号和姓名的组合才可以唯一标识一条记录，所以 学号和姓名的属性组是一个主键

上机记录表中的序列号不是成绩表的主键，但它和学生表中的学号相对应，并且学生表中的学号是学生表的主键，则称成绩表中的学号是学生表的外键

定义主键和外键主要是为了维护关系数据库的完整性，总结一下：

主键是能确定一条记录的唯一标识，比如，一条记录包括身份证号，姓名，年龄。身份证号是唯一能确定你这个人的，其他都可能有重复，所以，身份证号是主键。

外键用于与另一张表的关联。是能确定另一张表记录的字段，用于保持数据的一致性。比如，A表中的一个字段，是B表的主键，那他就可以是A表的外键。

二、 主键、外键 和索引的区别

主键、外键和索引的区别？

定义： 唯一标识一条记录，不能有重复的，不允许为空 表的外键是另一表的主键, 外键可以有重复的, 可以是空值
该字段没有重复值，但可以有一个空值作用： 用来保证数据完整性 用来和其他表建立联系用的是提高查询排序的速度个数： 主键只能有一个
一个表可以有多个外键 一个表可以有多个惟一索引

聚集索引和非聚集索引的区别？聚集索引一定是唯一索引。但唯一索引不一定是聚集索引。

聚集索引，在索引页里直接存放数据，而非聚集索引在索引页里存放的是索引，这些索引指向专门的数据页的数据。

三、数据库中主键和外键的设计原则

主键和外键是把多个表组织为一个有效的关系数据库的粘合剂。主键和外键的设计对物理数据库的性能和可用性都有着决定性的影响。必须将数据库模式从理论上的逻辑设计转换为实际的物理设计。而主键和外键的结构是这个设计过程的症结所在。一旦将所设计的数据库用于了生产环境

jida2010

个性签名：路由器网www.luyouqiwang.net

作者的经验

这样设置你的无线网络更安全

WPA/WPA2无线密码怎么破解

电信光猫与 TP-LINK路由器怎么连接

httpd.ini这个文件是干嘛用的

路由器上面的WAN口和LAN口是神马意思...

，就很难对这些键进行修改，所以在开发阶段就设计好主键和外键就是非常必要和值得的。

主键：

关系数据库依赖于主键---它是数据库物理模式的基石。主键在物理层面上只有两个用途：

1. 惟一地标识一行。

2. 作为一个可以被外键有效引用的对象。

基于以上这两个用途，下面给出了我在设计物理层面的主键时所遵循的一些原则：

1. 主键应当是对用户没有意义的。如果用户看到了一个表示多对多关系的连接表中的数据，并抱怨它没有什么用处，那就证明它的主键设计地很好。

2. 主键应该是单列的，以便提高连接和筛选操作的效率。

注：使用复合键的人通常有两个理由为自己开脱，而这两个理由都是错误的。其一是主键应当具有实际意义，然而，让主键具有意义只不过是给人为地破坏数据库提供了方便。其二是利用这种方法可以在描述多对多关系的连接表中使用两个外部键来作为主键，我也反对这种做法，理由是：复合主键常常导致不良的外键，即当连接表成为另一个从表的主表，而依据上面的第二种方法成为这个表主键的一部分，然而这个表又有可能再成为其它从表的主表，其主键又有可能成了其它从表主键的一部分，如此传递下去，越靠后的从表，其主键将会包含越多的列了。

3.
永远也不要更新主键。实际上，因为主键除了惟一地标识一行之外，再没有其他的用途了，所以也就没有理由去对它更新。如果主键需要更新，则说明主键应对用户无意义的原则被违反了。

注：这项原则对于那些经常需要在数据转换或多数据库合并时进行数据整理的数据并不适用。

4. 主键不应包含动态变化的数据，如时间戳、创建时间列、修改时间列等。

5.
主键应当有计算机自动生成。如果由人来对主键的创建进行干预，就会使它带有除了惟一标识一行以外的意义。一旦越过这个界限，就可能产生认为修改主键的动机，这样，这种系统用来链接记录行、管理记录行的关键手段就会落入不了解数据库设计的人的手中。

四、数据库主键选取策略

我们在建立数据库的时候，需要为每张表指定一个主键，所谓主键就是能够唯一标识表中某一行的属性或属性组，一个表只能有一个主键，但可以有多个候选索引。因为主键可以唯一标识某一行记录，所以可以确保执行数据更新、删除的时候不会出现张冠李戴的错误。当然，其它字段可以辅助我们在执行这些操作时消除共享冲突，不过就不在这里讨论了。主键除了上述作用外，常常与外键构成参照完整性约束，防止出现数据不一致。所以数据库在设计时，主键起到了很重要的作用。

常见的数据库主键选取方式有：

• 自动增长字段

• 手动增长字段

• UniqueIdentifier

• “COMB（Combine）”类型

1自动增长型字段很多数据库设计者喜欢使用自动增长型字段，因为它使用简单。自动增长型字段允许我们在向数据库添加数据时，不考虑主键的取值，记录插入后，数据库系统会自动为其分配一个值，确保绝对不会出现重复。如果使用SQL
Server数据库的话，我们还可以在记录插入后使用@@Identity全局变量获取系统分配的主键键值。

尽管自动增长型字段会省掉我们很多繁琐的工作，但使用它也存在潜在的问题，那就是在数据缓冲模式下，很难预先填写主键与外键的值。

假设有两张表：

Order(OrderID, OrderDate)

OrderDetial(OrderID, LineNum, ProductID, Price)

Order表中的OrderID是自动增长型的字段。现在需要我们录入一张订单，包括在Order表中插入一条记录以及在OrderDetail表中插入若干条记录。因为Order表中的OrderID是自动增长型的字段，那么我们在记录正式插入到数据库之前无法事先得知它的取值，只有在更新后才能知道数据库为它分配的是什么值。这会造成以下矛盾发生：

首先，为了能在OrderDetail的OrderID字段中添入正确的值，必须先更新Order表以获取到系统为其分配的OrderID值，然后再用这个OrderID填充OrderDetail表。最后更新OderDetail表。但是，为了确保数据的一致性，Order与OrderDetail在更新时必须在事务保护下同时进行，即确保两表同时更行成功。显然它们是相互矛盾的。

除此之外，当我们需要在多个数据库间进行数据的复制时（SQL
Server的数据分发、订阅机制允许我们进行库间的数据复制操作），自动增长型字段可能造成数据合并时的主键冲突。设想一个数据库中的Order表向另一个库中的Order表复制数据库时，OrderID到底该不该自动增长呢？

ADO.NET允许我们在DataSet中将某一个字段设置为自动增长型字段，但千万记住，这个自动增长字段仅仅是个占位符而已，当数据库进行更新时，数据库生成的值会自动取代ADO.NET分配的值。所以为了防止用户产生误解，建议大家将ADO.NET中的自动增长初始值以及增量都设置成-1。此外，在ADO.NET中，我们可以为两张表建立DataRelation，这样存在级联关系的两张表更新时，一张表更新后另外一张表对应键的值也会自动发生变化，这会大大减少了我们对存在级联关系的两表间更新时自动增长型字段带来的麻烦。

2手动增长型字段既然自动增长型字段会带来如此的麻烦，我们不妨考虑使用手动增长型的字段，也就是说主键的值需要自己维护，通常情况下需要建立一张单独的表存储当前主键键值。还用上面的例子来说，这次我们新建一张表叫IntKey，包含两个字段，KeyName以及KeyValue。就像一个HashTable，给一个KeyName，就可以知道目前的KeyValue是什么，然后手工实现键值数据递增。在SQL
Server中可以编写这样一个存储过程，让取键值的过程自动进行。代码如下：

CREATE PROCEDURE[GetKey]

@KeyNamechar(10),

@KeyValue intOUTPUT AS UPDATE IntKey SET @KeyValue =KeyValue = KeyValue + 1
WHERE KeyName = @KeyName GO

这样，通过调用存储过程，我们可以获得最新键值，确保不会出现重复。若将OrderID字段设置为手动增长型字段，我们的程序可以由以下几步来实现：首先调用存储过程，获得一个OrderID，然后使用这个OrderID填充Order表与OrderDetail表，最后在事务保护下对两表进行更新。

使用手动增长型字段作为主键在进行数据库间数据复制时，可以确保数据合并过程中不会出现键值冲突，只要我们为不同的数据库分配不同的主键取值段就行了。但是，使用手动增长型字段会增加网络的RoundTrip，我们必须通过增加一次数据库访问来获取当前主键键值，这会增加网络和数据库的负载，当处于一个低速或断开的网络环境中时，这种做法会有很大的弊端。同时，手工维护主键还要考虑并发冲突等种种因素，这更会增加系统的复杂程度。

3使用UniqueIdentifierSQL Server为我们提供了UniqueIdentifier数据类型，并提供了一个生成函数NEWID(
)，使用NEWID(
)可以生成一个唯一的UniqueIdentifier。UniqueIdentifier在数据库中占用16个字节，出现重复

的概率非常小，以至于可以认为是0。我们经常从注册表中看到类似

{45F0EB02-0727-4F2E-AAB5-E8AEDEE0CEC5}

的东西实际上就是一个UniqueIdentifier，Windows用它来做COM组件以及接口的标识，防止出现重复。在.NET里管UniqueIdentifier称之为GUID（Global
Unique Identifier）。在C#中可以使用如下命令生成一个GUID：

Guid u =System.Guid.NewGuid();

对于上面提到的Order与OrderDetail的程序，如果选用UniqueIdentifier作为主键的话，我们完全可以避免上面提到的增加网络RoundTrip的问题。通过程序直接生成GUID填充主键，不用考虑是否会出现重复。

UniqueIdentifier字段也存在严重的缺陷：首先，它的长度是16字节，是整数的4倍长，会占用大量存储空间。更为严重的是，UniqueIdentifier的生成毫无规律可言，要想在上面建立索引（绝大多数数据库在主键上都有索引）是一个非常耗时的操作。有人做过实验，插入同样的数据量，使用UniqueIdentifier型数据做主键要比使用Integer型数据慢，所以，出于效率考虑，尽可能避免使用UniqueIdentifier型数据库作为主键键值。

4使用“COMB（Combine）”类型既然上面三种主键类型选取策略都存在各自的缺点，那么到底有没有好的办法加以解决呢？答案是肯定的。通过使用COMB类型（数据库中没有COMB类型，它是Jimmy
Nilsson在他的“The Cost of GUIDs asPrimary Keys”一文中设计出来的），可以在三者之间找到一个很好的平衡点。

COMB数据类型的基本设计思路是这样的：既然UniqueIdentifier数据因毫无规律可言造成索引效率低下，影响了系统的性能，那么我们能不能通过组合的方式，保留UniqueIdentifier的前10个字节，用后6个字节表示GUID生成的时间（DateTime），这样我们将时间信息与UniqueIdentifier组合起来，在保留UniqueIdentifier的唯一性的同时增加了有序性，以此来提高索引效率。也许有人会担心UniqueIdentifier减少到10字节会造成数据出现重复，其实不用担心，后6字节的时间精度可以达到1/300秒，两个COMB类型数据完全相同的可能性是在这1/300秒内生成的两个GUID前10个字节完全相同，这几乎是不可能的！在SQL
Server中用SQL命令将这一思路实现出来便是：

DECLARE @aGuidUNIQUEIDENTIFIER

SET @aGuid =CAST(CAST(NEWID() AS BINARY(10))

+ CAST(GETDATE()AS BINARY(6)) AS UNIQUEIDENTIFIER)

经过测试，使用COMB做主键比使用INT做主键，在检索、插入、更新、删除等操作上仍然显慢，但比Unidentifier类型要快上一些。

以上是对SQL数据库中的主键与外键的简单介绍，如果有出入，还请谅解！

相关经验

换一批

mysql中的怎样使用sql创建多字段的主键的表 0 2013.11.12

mysql中的使用sql创建单一主键的表 0 2013.11.12

sql数据库如何设置主键自增长 3 2014.06.18

我来介绍下用sql语句删除mysql数据库php中通用	0	2013.11.08
mysql 增加外键：(+创建主键)	2	2014.05.27

热门杂志

sql创建表语句设置主键和外键 sql 中的主键、键索引 pl/