SQL Server 2005/2008

数据建模开发者指南

SQL Server 2005/2008 数据建模开发者指南

本篇文章主要是发现 SQL Server 数据存储的物理对象和原理，本文节选自《SQL Server 数据建模开发者指南》，范围包括 SQL Server 2005 和 2008。你将从本文中了解到各种数据类型、如何使数据主键和外键进行整合以及进行编码存储程序和参数的适当的步骤。作者 Eric Johnson 和 Joshua Jones 还介绍了怎样在 SQL Server 中操作父级和子级表。

SQL Server 2005 和 2008 里的物理数据存储

你已经有了一些用于构建数据模型的逻辑原理，我们现在再来看看这些物理原理。你可以用这些对象构建数据库。构建到物理模式中的大多数对象都基于你在逻辑模式中的对象。无论你用的是哪一种 RDBMS，许多物理原理都是相同的。但是我们今天讨论的是 SQL Server 2008 中的一些有用的原理。了解 SQL Server 的性能对于你用它们来构建模式很重要。在本章中，我们将详细讨论所有 SQL Server 对象，让你大致了解如何在物理模式中使用每一种对象。在第九章中你将会用到这些原理。

SQL Server 2008 数据类型

当你需要存储日期或时间值时，SQL Server 就会提供六种数据类型。懂得用哪种数据类型很重要，因为每个日期和时间数据类型所有的精确性都有些许不同，但是在你计算精确时间和持续时间时它们就有很大的差别了。本章逐一详细介绍了这些数据类型。

- SQL Server 2008 数据类型：Datetime、字符串、自定义及更多（一）
- SQL Server 2008 数据类型：Datetime、字符串、自定义及更多（二）

在 SQL Server 数据库中加强数据的完整性

一般而言，数据完整性就是保持数据连续性并帮助你数据真实、便于检索这样的一个概念。完整性有很多种，参考完整性确保在你插入或更新数据时就附加了表之间的关系。例如，假如你有两个表，一个是 Employee，另一个叫 Vehicle。你要求每辆交通工具（Vehicle）和每个员工（ employee）之间能够进行匹配，这就要通过关系，规则也就是参考完整性规则。通过使用主键和外键实施这种关系。

- 在 SQL Server 数据库中加强数据的完整性（一）
- 在 SQL Server 数据库中加强数据的完整性（二）
- 在 SQL Server 数据库中加强数据的完整性（三）
- 在 SQL Server 数据库中加强数据的完整性（四）

SQL Server 和 T-SQL 数据操纵

除了那些用来保存数据和实现数据完整性的对象之外，SQL Server 还提供了一些允许你编写数据操纵代码的对象。我们可以用这些对象在数据库里进行数据插入、更新、删除或者读数据，还可以利用它们来实施事物规则和实现高级数据完整性。你能在 SQL Server

中构建“应用软件”。一般这些应用软件都很小并且通常以某种方式提供服务或更大的应用软件操纵数据。

- SQL Server 和 T-SQL 数据操纵（一）
- SQL Server 和 T-SQL 数据操纵（二）

SQL Server 中的父表和子表

如何在 SQL Server 中实施子型集群？你有三种选择。首先就是创建一个单独的表表示父型特征并包含所有子型特征。第二种选择就是给每个子型创建表，给每个子表增加父型特征。第三，你可以同时创建父表和子表，和逻辑建模的方式一样更加有效地实现子型集群。 要判断哪种方法正确，你就必须看清楚保存的这些数据。我们会仔细考虑这三种选择中的每一种，并考虑到你使用他们的原因以及每种选择的优点和缺点。

- SQL Server 中的父表和子表（一）
- SQL Server 中的父表和子表（二）

SQL Server 2005 和 2008 里的物理数据存储（一）

本篇文章主要是发现 SQL Server 数据存储的物理对象和原理，本文节选自《SQL Server 数据建模开发者指南》，范围包括 SQL Server 2005 和 2008。你将从本文中了解到各种数据类型、如何使数据主键和外键进行整合以及进行编码存储程序和参数的适当的步骤。作者 Eric Johnson 和 Joshua Jones 还介绍了怎样在 SQL Server 中操作父级和子级图表。

数据模型的物理原理

现在你已经有了一些用于构建数据模型的逻辑原理，我们再来看看这些物理原理。你可以用这些对象构建数据库。构建到物理模式中的大多数对象都基于你在逻辑模式中的对象。无论你用的是哪一种 RDBMS，许多物理原理都是相同的。但是我们今天讨论的是 SQL Server 2008 中的一些有用的原理。了解 SQL Server 的性能对于你用它们来构建模式很重要。

在本章中，我们将详细讨论所有 SQL Server 对象，让你大致了解如何在物理模式中使用每一种对象。在第九章中你将会用到这些原理。

物理存储

首先，我们从能让你在数据库中存储数据的对象开始讨论。你就是在这些对象上建立其他东西的。特别是这些表、视图和数据类型。

表

表是用来建立模块的，这些模块上面还建立有关系型数据库。几乎所有的数据库都以表的形式结束。表由行和列构成。如实体中的单个实例，每行中储存的信息都是关于单个记录的。例如，在员工表中，每一行都存储了一个员工的信息。

表里的列存储的是表里这些行的信息。Employee 表里的 FirstName 列存储的是所有的员工的名字。列映射还要归结于你的逻辑模式，并且就像逻辑模式一样，每个列都配置了一种数据类型。接下来的内容中我们会详细介绍 SQL Server 的数据类型。

当你在表中增加数据的时候，每个列必须包含数据（即使是空字符串）或指定一个 NULL 值。NULL 就是完全没有数据。此外，你还能给每个列指定一个默认值。如果你没有指定一个值就增加了数据，那么你就可以用默认值。默认值可以是一个固定值，如给在数字列中设置值为 12，或者是返回合适的数据类型值的一个函数。如果你没有指定默认值，还在没有指定列值得情况下插入了数据，SQL Server 就会插入一个 NULL。如果列中不允许 NULL 值，你的数据插入就会失败。

你可以在应用软件中将表当作独立的电子表格，如 Microsoft Excel。事实上，一个 Excel 电子表格就是一个表。但是 Excel 不是关系型挂办理系统。数据库只不过就是存储信息的表的集合。当然，在数据库中还有一些其他的对象，但是没有这些标你就没有任何数据。用 Transact-SQL，也叫 T-SQL，你就能在表中操控这些数据。四种基本的数据操作语言（DML）语句定义如下：

- SELECT：允许用户在一个或多个表中检索数据
- INSERT：允许用户在表中增加数据
- UPDATE：允许用户在表中改变数据
- DELETE:允许用户从表中迁移数据

SQL Server 怎样储存表

除了要理解表的定义以外，了解 SQL Server 怎样存储这些表也同样重要，你的列中存储的这些数据类型将命令你如何在磁盘上存储该表，因此也会直接影响到你的数据库性能。SQL Server 的一切都存储在页面上。8K 型连续分配的信息页面都在磁盘上，根据页面上内容的不同就有不同的页面。对于我们来说，我们关注的是数据页面，即存储表数据的页面。你增加到表中的每行都存储在页面上。并且根据行中的数据大小，一个行既能和其他的行一起存储，也能存储在自己的页面上。

在 SQL Server 2005 之前，数据和单独行的整个的占用空间不能超过 8，060 字节（8K）。这是在你设计表的时候你需要考虑、解决的一个硬性的界限。从某种意义上来说，SQL Server 2005 中可以克服这个界限。现在，如果你的行已经超过了 8，060 字节，SQL Server 就会将一个或者多个可变长度列迁移到一个新的页面上，只留下一个 24 字节大小的指针。这并不是说对你的行大小没有限制，你也不需要使你的行大小大于 8，060 字节。为什么不呢？首先，注意我们说 SQL Server 将要迁移可变长度列。这就是说你的固定列长度还是不能超过 8，060 字节。此外，该行的主要数据页面还是不能超过 8K。还记得 24 字节大小的指针吗？从理论上说在主页面上你的指针不能超过 335 个。如果表里有 336 列 varchar(8000)，看起来就很怪。

如果 SQL Server 在幕后管理所有的东西，那你为什么还要关注呢？理由如下：尽管在你超越了 8K 的限制之后，SQL Server 将可变长度范围迁移到新页面上，结果和分段硬件驱动很类似。你在访问的时候需要组合大量的数据，这样自然就增加了处理的时间。作为数据模型创建者，由于考虑到性能你就需要让行的大小小于 8K。对于这一点来说，也有一些例外，这些内容我们在讨论数据类型的时候会详细谈到。记住在 SQL Server 处理存储和页面的方式会比我们在这里讨论的要复杂多了，但是你的数据模型不会影响其他的变量，它更可能会影响表的大小。

（作者：SearchSQL Server.com 译者：April 来源：TT 中国）

SQL Server 2005 和 2008 里的物理数据存储（二）

视图

视图是就存储 T-SQL 语言的，而 T-SQL 语言则用 SELECT 语句来显示一个表到更多个表的数据，视图参考的表通常指的是视图基表。就像字面意思表示的那样，视图允许你创建各种有潜在信息的图。你可以查阅每个基表制作视图。这一性能让你分割数据并且只显示相关信息。

你访问视图和访问表的方式一样。所有的基本 DML 语句工作原理和视图相反、和表的也相反，当然也有一些例外。如果你有一个视图涉及一个以上的基表，你就可以只用 INSERT、UPDATE 或 DELETE 语句查阅一个基表中的列。如：假设一个视图从两个表中返回的是客户数据。一个表存储的是客户信息，另一个表存储的是该客户地址数据。因此，customer_address 视图的定义如下：

```
CREATE VIEW customer_address
AS
SELECT customer.first_name,
customer.last_name,
customer.phone,
address.address_line1,
address.city,
address.state,
address.zip
FROM customer
```

```
JOIN address
ON address.customer_id = customer.customer_id
WHERE address.type = 'home'
```

如果你只查阅客户表或地址表，你就可以对 customer_address 视图进行 INSERT、UPDATE 和 DELETE 操作。

可能你会问自己，“我为什么要使用视图，而不是直接查阅表呢？”下面是在数据库中使用视图的几个理由。首先，你能使用视图掩盖基础表的复杂性。如果你有一个单独的视图显示客户的地址信息，开发人员和中端用户就能够在视图中访问他们所需要的信息，而不是访问两个表。这项技术排除了用户需要了解真个数据库的必要性，他们可以将精力集中到某一个单独的对象上。在单个视图中启用基表时你就受益匪浅。

使用视图还能允许你在不影响用户的情况下更改表或者数据存储的位置。最后，只要你更新试图定义，让它适应你更改的表，你的用户就不需要知道你已经做了改变。你还可以用视图更好的进行安全管理。如果有用户要查看一些员工数据而不是如社会安全数字或工资这一类的敏感数据，你就可以建一个只显示他们所需要的信息的视图。

最后，考虑怎样在查询你的数据库时使用视图节省时间。每次你都运行 T-SQL 代码，SQL Server 必须首先编码。将人们易读 SELECT 语句转换成 SQL Server 引擎能懂的形式。结果代码就是执行计划。运行视图的执行计划存储在 SQL Server 中，T-SQL 后面的代码也编写出来了。这一过程要耗时间，但是在创建视图的时，编码就已经完成了。这样你每次访问视图时都节省了时间。你第一次访问视图时，SQL Server 就会算出从基表中检索数据的最佳方法，将表结构和索引放在适当的位置。下次访问该视图又会执行这种方案。

依我之见，视图可能就是 SQL Server 中最没有被充分利用的功能。由于某些原因，人们尽量避免使用视图或以一种低效率的方式使用视图。在第十一章中，我们介绍了让你最能受益的视图使用方式。

数据类型

正如在之前提到的一样，每个表中每个列都必须设置存储指定类型的数据。你可以通过把数据类型和列联系起来的方式设置。数据类型就是你用来指定列中能存储的类型、长度、精度和数据尺度。SQL Server 2008 介绍了一些普通的数据类型范畴，每个范畴都包含指定的数据类型。其中的一些数据类型和我们在第二章中了解到的类型很类似。在这一部分中，我们将了解每种 SQL Server 数据类型，还会谈到 SQL Server 引擎如何处理和存储这些数据类型。

（作者：SearchSQLServer.com 译者：April 来源：TT 中国）

SQL Server 2005 和 2008 里的物理数据存储（三）

建模时，了解每个数据类型所需要的空间很重要。相对需要 4 个字节的数据类型来说，需要 2 个字节的数据类型就和它没有很大的区别。但是如果你在一百万或十亿行上增加 2 个字节，你就需要另外 10 个或者 100 个十亿字节的存储器。

SQL Server 2008 有允许 SQL Server 存储引擎将数据压缩到行和页面里的功能，部分功能在 SQL Server 2005 服务包 2 中已经提到。但是这一功能在企业版中受到了限制，并且更多地与行政相关。你对数据存储需求的估计要基于我们刚才谈到的数据，并且还要控制在没有被压缩的存储空间范围内。将数据压缩到数据库里面也就意味着，在数据库建成以后，数据库管理员要将和数据库开发人员通力协作。我们来看一下 SQL Server 2008 中能够用到的数据类型。

数字型数据类型

我们的数据库要存储我们每天用到的各种数据。每一个数据都是唯一的，都要求我们存储各种不同的数据块。数据之间的区别和要求表示 SQL Server 能够支持 11 种数字数据类型。以下对 SQL Server 中可用的数字型数据类型的回顾。同样，表 3.1 是对每种数据类型的详细说明。

表 3.1 Numeric 数据类型

数据类型	值范围	存储大小
bigint	-9，223，372，036，854，775，808 through 9，223，372，036，854，775，807	8 bytes
bit	0 或 1	1 byte (minimum)

decimal	由精度和刻度决定	5 - 17 bytes
float	- 1.79E+308 through - 2.23E - 308, 0,	4 or 8 bytes
int	- 2, 147, 483, 648 to 2, 147, 483, 647	4 bytes
money	- 922, 337, 203, 685, 477.5808 to 922, 337, 203, 685, 477.5807	8 bytes
numeric	由精度和刻度决定	5 - 17 bytes
real	- 3.40E+38 to - 1.18E - 38, 0, and 1.18E - 38 to 3.40E+38	4 bytes
smallint	- 32, 768 to 32, 767	2 bytes
smallmoney	- 214, 748.3648 to 214, 748.3647	4 bytes
tinyint	0 to 255	1 byte

Int

Int 数据类型用来存储所有整数。Int 不存储任何小数点后面的数字，也不会存储任何四舍五入到整数的小数。这类数据类型的存储范围为 - 2，147，483，648 至 2，147，483，647，并且每个 int 数据都需要占用磁盘上的 4 个字节。

Bigint

Bigint 仅仅就是：一个比较大的整数。当你需要比 int 数据类型能支持的更大的数字时，你就可以使用 bigint。使用 bigint 可以将范围从 int 的 20 亿扩大，并且允许你存储范围大约为-900 万的三次方到 900 万的三次方。（一百万的三次方就是 1 后面跟有 18 个零。）更大的数字就需要更多的存储空间，每个 bigint 数据需要 8 个字节。

Smallint

与 Int 数据类型相对的就是 smallint。Smallint 包括的数据范围为－32，768 至 32，767，并且只需要 2 个字节的存储空间。

Tinyint

Int 以外的数据类型就是 tinyint。它只需要一个字节的存储空间，并且能存储从 0 到 255 的数字，tinyint 很适合状况列。注意 tinyint 是唯一一种不能存储负数的 int 数据类型。

Bit

Bit 数据类型就相当于一个标记或者布尔算子。有效值 0、1 或 NULL 组成用于存储或关闭、是或不是、或真或假、完美的 bit 数据类型。Bit 数据类型存储空间更加复杂。存储 1 或 0 这样的数字只需要占用 1 个字节的磁盘空间，而 bit 数据的最小存储空间为 1 个字节。下表中，bit 数据类型列中的数据都是集中存储。这就意味着当你的列中含有 1 个字节到 8 位的数据，它们总共就占用 1 个字节。如果列包含的数字是 9 到 16 位，它们就总共占有 2 个字节，依此类推。SQL Server 也暗中将 TRUE 和 FALSE 字符串分别变成了数字 1 和 0。

Decimal 和 Numeric 数据类型

在 SQL Server 2008 中，Decimal 和 numeric 数据类型基本上是一样的。SQL Server 之前的版本并没有 numeric 数据类型，只在 SQL Server 2005 中才增加了这一项，这样在术语里才和 RDBMS 软件一致。这些数据类型包括小数点右边有小数的一些数据。在用 decimal 或 numeric 数据类型时，你可以指定精度和刻度。精度设定了该数据能够存储的数字。精度可以是从 1 到 38 的任何值，允许后面的小数包括从 1 到 38 的数字。刻度可以是从 0 到你设定的精度。例如，数字 234.67 的精度就是 5、刻度为 2。Decimal 和 numeric 的存储需求空间可以根据精度来调整。表 3.2 就表示基于精度的存储需求空间。

表 3.2 Decimal 和 Numeric 存储空间大小

精度	存储大小
1 至 9	5 bytes
10 至 19	9 bytes
20 至 28	13 bytes
29 至 38	17 bytes

货币和货币值数据类型

货币和货币值数据类型存储四个小数位的货币值。这两种类型数据之间的区别就是货币值的存储范围为－922 兆到 922 兆，并且存储大小为 8 个字节；而货币值数据类型只包括从－214，748.3648 到 214，748.3647 之间的值，存储大小为 4 个字节。从功能上说，这些数据类型和 decimal、 numeric 数据类型很相似，但是货币和货币值数据类型还能存储货币符号：$（美元)， ￥（日元)或 £（法郎)。

Float 和 Real 数据类型

Float 和 Real 数据类型为近似数据。每个数据包括科学计数法，由于缺少精度就造成了数据丢失。如果你记不住高中的化学班级，我们可以简单解释一些科学计数法。你基本上存储了一些值的子集，这些值都规定前面或后面可以跟多少个小数。所以你就可以将 1，234，467，890 存储为 1.23E+9。也就是说 1.23 这个数据中的小数点可以向右移 9 位。你可以看到，在用这种方法存储数据时，你丢失了一些小数。在转换成科学计数法时，原数据（1，234，467，890）就变成了 1，230，000，000。

现在返回到数据类型。Float 和 real 数据类型能用科学计数法存储，唯一的区别就是这些值得范围和存储大小。表 3.1 为这些数据类型值的范围。Real 数据类型需要 4 个字节的存储空间，并且固定精度为 7。你可以用这些浮点数据指定精度或整个数据的数字，即从 1 到 53。存储空间大小为 4 个字节（精度小于 25 时）到 8 个字节（精度为 25 到 53）。

(作者：SearchSQLServer.com 译者：April 来源：TT 中国)

SQL Server 2008 数据类型：Datetime、字符串、自定义及更多（一）

日期和时间数据类型

当你需要存储日期或时间值时，SQL Server 就会提供六种数据类型。懂得用哪种数据类型很重要，因为每个日期和时间数据类型所有的精确性都有些许不同，但是在你计算精确时间和持续时间时它们就有很大的差别了。我们依次看看。

Datetime 和 Smalldatetime 数据类型能够存储各种文本形式的日期和时间数据。它们之间的不同之处就是各自能存储的值的范围。Datetime 能够存储从 1753 年 1 月 1 日到 9999 年 12 月 31 日之间的值，并且还能精确到 3.33 毫秒。相反，smalldatetime 只能存储从 1900 年 1 月 1 日至 2079 年 6 月 6 日之间的日期值，还只能精确到 1 分钟。就存储而言，datetime 需要 8 个字节，而 smalldatetime 只需要 4 个字节。

日期和时间

现在 SQL Server 2008 里的新数据类型，能将传统的日和时间数据类型的日期部分和时间部分分开。从字面上来理解，它们是两种数据类型进行的说明：日期部分（月、日和年），或者是时间部分（小时、分钟、秒和纳秒）。因此，如果有必要的话，你就只能在列中存储其中的一部份。

日期数据类型的有效值的范围和 datetime 数据类型的有效值范围一样，也就是说日期型数据类型的值包括 1753 年 1 月 1 日至 9999 年 12 月 31 日。从存储的立场看来，日期型数据类型只需要占用 3 个字节大小，字符长度为 10。

时间型数据类型包括从 00:00:00.0000000 到 23:59:59.9999999 之间的值，能够包括 8 个字符（hh:mm:ss）至 16 个字符（hh:mm:ss:nnnnnnn），代表小数秒。例如，

13:45:25.5 从字面理解就是 1:45:25 加上半秒。你可以指定从 0 到 7 之间时间数据类型的刻度、指定你可以在小数秒中使用多少个数字。时间数据类型的最多需要 5 个字节来存储。

Datetime2

SQL Server2008 中的另一种新数据类型。它和原 datetime 数据类型很相近，只是 datetime2 结合了时间数据类型的精度和刻度选项。你可以在 0 到 7 之间指定刻度，这取决于你想不想划分和存储。这种数据类型的存储大小为 8 个字节，精度假定为 7。

Datetimeoffset

SQL Server 2008 中最后一种新增的日期和时间数据类型就是 datetimeoffset。这是一种标准的日期和时间数据类型，和 datetime2 相似（因为它能够存储精度）。另外，datetimeoffset 还能够存储加减 14 小时的偏移量。这在应用软件中很有用，当你想存储日期和时间以及相关的偏移量，如多个时区。Datetimeoffset 的存储所需空间为 10 个字节。

字符串数据类型

存储字符串和字符数据时，选项和变量很复杂。不论你是否要存储单个字母还是《战争与和平》的整篇文章，SQL Server 提供了字符串型数据类型。幸运的是，如果你知道现有字符串数据类型，那你就可以直接选择正确的一项。

Char 和 Varchar 数据类型

Char 和 Varchar 数据类型可能是最常用的字符串数据类型。它们都是用来存储标准的、非统一码的文本数据。二者之间的区别就在于数据存储。在这两种情况下，你必须指定一个长度并定义列为 char 或 varchar。该长度设定了该列能够存储的字符个数限制。

Char 数据类型在存储时需要的字节个数和你之前指定的长度一样。如果有一个 char(20)，即使你在列中只存储一个 5 个字符的词，你也需要 20 个字节存储。如果存储 varchar，存储空间大小通常是你存储的实际字符数加上 2 个字节。所以 varchar(20)以及一个 5 个字符的单词将占用 7 个字节的空间。另外的 2 个字节包括 SQL Server 的 size reference。每种类型长度可以包括 8000 个字符。

什么时候选择它们其中的一种类型呢？如果单凭经验，就是党所有的数据都有相同的长度时就用 char，当数据发生了很大的变化时，就用 varchar。以下是最佳存储规则。

另一种技巧就是对于较短的列避免使用 varchar。我们已经了解了数据库是用 varchar(2)列的情况，其结果就是浪费空间。假设你的表中有 100 行，每个表都包括一个 varchar(2)列。假定所有的列都是 NULL，那你还需要 2 个字节的存储空间。所以，没有存储任何数据，你就已经占用了和用 char(2)一样多的空间。

第三种特殊的 varchar 函数就是最大长度选项。当你指定长度的最大值时，varchar 列就能存储多达 2^31 - 1 字节的数据，大约为 2 兆个字节或 2GB 的字符数据。如果你认为这还不算多，那就请你打开文本编辑器开始输入，直到能够称一个 2GB 的文件为止。接下来就是等待。有很多信息可以填满单个列。在 2005 版本中增加了 Varchar(max)，它也就取代了之前 SQL Server 版本中的文本数据类型。

(作者：SearchSQLServer.com 译者： April 来源：TT 中国)

SQL Server 2008 数据类型：Datetime、字符串、自定义及更多（二）

Nchar 和 Nvarchar 数据类型

Nchar 和 Nvarchar 数据类型与 char 和 varchar 数据类型工作原理一样，只是 n 用于存储统一码。统一码是在你要存储非英语语言字符串时最常用的，这类字符串需要特殊的字符，如希腊字母 beta (_)。由于统一码有些复杂，所以每个字节都需要 2 个字节，因此一个 nchar 需要两倍长度来存储，nvarchar 需要两倍实际字符数加上必须的 2 个字节。

前面我们已经提到了，现在再来回顾一下 SQL Server 将表存储在 8060 字节的页面。而单个列并不能粘满整个页面，所以简单的数学告诉我们当你使用这些统一码数据类型时，你可达到 8000 个字节、长度可达 4000 个字节，这在 SQL Server 2005 中就替换了以前的 ntext 数据类型。

Binary 和 Varbinary

Binary and Varbinary Binary 和 Varbinary 函数与 char 和 varchar 一样。唯一的区别就是它们包括 binary 信息，如文件或者图像。如以前，varbinary(max)就取代了就的图像数据类型。此外，SQL Server 2008 允许你指定 varbinary(max) 列的文件属性，这样就改变了 BLOB 存储方式。它保存在磁盘 SQL Server 的页面上而不是在文件系统上作为独立的文件保存。

Text、Ntext 和 Image

如之前提到的，text、ntext 和 image 数据类型已经分别由 varchar、nvarchar、和 varbinary 最大长度来代替。但是，如果你现在运行的是老版本或者已经升级到 SQL Server 2005 或 SQL Server 2008，可能你还需要这些数据类型。这种 text 数据类型包括

2GB 的字符串数据，ntext 包括 1GB 的统一码字符串数据。Image 属于可变长的二进制域，可以包括 2GB 的任何二进制的数据。当你在使用这些数据类型时，你必须用某些函数来写、升级、读这些列，但是你不能就进行简单的升级。记住这三类数据类型已经被替换了，微软很可能会将它们从以后的 SQL Server 版本中去掉。

其他数据类型

除了标准的数字和字符串型数据类型时，SQL Server 2008 提供了其他几种有用的数据类型。这些类型允许你存储 XML 数据、全球唯一标识符（GUIDs）、hierarchical identities 和空间坐标数据类型。还有一种新的存储数据类型，我们会简短地谈到。

Sql_variant

定义为 sql_variant 的列，能最大范围地存储任何可以用其他 SQL Server 数据类型保存的数据。唯一不能保存到 sql_variant 中的数据就是 text、ntext、image、xml、timestamp 或最大长度的数据类型。你可以用 sql_variant 保存各种数据类型到一个表的同一列中。在第 4 章你会了解到，从建模的角度看这并不是最佳的解决方案。也就是说 sql_variant 还有一些更好的用途，如当你从其他地方访问一些不太好的数据时创建 staging table 平台等。存储 sql_variant 所需的空间由你放入列中的数据类型来决定。

Timestamp

这种数据类型有一些误导名称。实际上 timestamp 并没有存储有关时间和日期的信息。它是一种二进制的数字、每次都会对包含 timestamp 列的表自动增加插入或更新操作。Timestamp 列的计数器保存整个数据库，每个表都只有 timestamp 列。这样，你就可以告知你的数据库各种操作的顺序，你还可以实施行版本。

Uniqueidentifier

Uniqueidentifier 可能数据类型中最有意思的，也是很多人辩论的主题。基本上，uniqueidentifier 列包括 GUID，即它是由 32 个任意字符组成的、有连字符分隔的字符串。例如：以下就是有效的 GUID。

```
45E8F437-670D-4409-93CB-F9424A40D6EE
```

你用到 uniqueidentifier 列时，记住以下的一些事情。首先，它们很大、需要 16 个字节的空间保存。第二，和 timestamps 或 identity 列不一样（稍后详见本章的主键部分），在插如数据时，uniqueidentifier 并不是自动被赋与了新的 GUID。在插入数据时你必须用 NEWID 函数生成一个新的 GUID。你还可以给 NEWID()列设定一个默认值。这样你就不需要指定 uniqueidentifier 列的任何别的东西了，SQL Server 会为你插入 GUID。

Xml

Xml 数据类型不数据本书的范围，但是我们还是会对它作一个简要的介绍。用 Xml 数据类型时，SQL Server 可以在列中包括可扩展的标记语言（XML）。另外，你可以将 XML schema 绑在列中来约束存储数据。如 max 数据类型一样，xml 数据类型存储空间最大为 2GB。

表

表数据类型可以保存一套可供后来处理的 T-SQL 语句。该数据保存的方法和保存整个表的方式一样。值得注意的是，表数据类型不能在列中使用，它只能用在 T-SQL 代码参数中。SQL Server 设计不是本书的内容，但是表数据类型在自定义函数中起到了很重要的作用，我们也会简要谈到。

表参数和基表运转方式一样。它们包括一些列，还有检查约束、唯一约束和主键。有了这些基表，表参数就能用于 SELECT、INSERT、UPDATE 和 DELETE 语句中。就像其他本地

参数一样，表参数存在于调用函数范围内并且在调用模块完成执行时就被清除了。要用这些表参数，你就要将它们和其他参数一样对待并规定一个标准的表定义。

Hierarchyid

Hierarchyid 数据类型是系统提供的数据类型，它允许你存储 hierarchical 数据，如数据库表中的机构数据、项目任务或文件系统。只要你有 selfreferencing 数据分层格式，hierarchyid 可以让你存储和查询数据更加有效。Hierarchyid 里的实际数据代表一系列的斜线和指定数值。这种特殊的数据类型只用于非常特殊的实例中。

空间数据类型

SQL Server 2008 还介绍有关存储的空间数据类型。首先两种新的数据类型属于几何学范围，他们允许你保存实际位置的平面数据（距离、向量等等）。另外一种数据类型，属于地理学范围，允许你保存有关地球的数据，如经度和纬度等等。虽然你这样太过于简单，这些数据类型就允许你保存能帮你决定位置和航行路线的信息。

自定义数据类型

除了我们描述的数据类型之外，SQL Server 还允许你创建自定义数据类型。有了自定义数据类型，你就能够创建你所用的表的标准列。在你定义这些自定义数据类型时，你仍然需要使用标准数据类型作为基础。自定义数据类型就是数据类型的一种固定的定义，还包括长度、精度或刻度。

例如，如果你需要在你的数据库中保存电话号码，你就能创建一个电话号码数据类型。如果你创建的电话号码数据类型是 varchar(25)，那么你定义的每一个列就是完全相同的，都是 varchar(25)。回顾一下在第二章中讨论的一样，自定义数据类型属于在 SQL Server 中实际执行的域。我们强烈推荐连贯性的自定义数据类型，在最初的开发和后来的增加期间都能对你的数据模块使用。

（作者：SearchSQLServer.com　译者：April 来源：TT 中国）

在 SQL Server 数据库中加强数据的完整性（一）

参考完整性

我们在第二章理已经讨论过了参考完整性（RI）。现在再来看看你如何在物理数据库中执行参考完整性。一般而言，数据完整性就是保持数据连续性并帮助你数据真实、便于检索这样的一个概念。完整性有很多种，参考完整性确保在你插入或更新数据时就附加了表之间的关系。例如，假如你有两个表，一个是 Employee，另一个叫 Vehicle。你要求每辆交通工具（Vehicle）和每个员工(employee) 之间能够进行匹配，这就要通过关系，规则也就是参考完整性规则。通过使用主键和外键实施这种关系。

主键

SQL Server 中的主键约束和它在逻辑模块中的工作原理一样。主键约束由一个或以上的列组成，这些列能够识别指定表中的行。

创建 PK 的第一步就是鉴别用于创建键的列。大多数时候这一步都是在逻辑建模的时侯完成。在 SQL Server 中怎样才能创建一个好的主键？更重要的是，什么能导致关键弱权（poor key）？你表中的任意一列或组合列可以识别候选键。通常一个表中有很多个候选键。我们要谈的第一个 PK 技巧就是避免字符串列。在你连接两个表时，SQL Server 必须比较主键中的数据和外键中其他表的数据。字符串本来就比数字型数据类型就要花更多时间和处理能量进行比较。

所以我们就选择了数字型数据类型。你应该用什么样的数字呢？整数同常是最好的选择，所以，只有数据类型够大，能够包含潜在的行，你就可以使用 int 数据类型。同样，你还能够创建一个复合 PK（使用多个列以上的 PK），但是如果你能够避免，我们并不建议你用复合 PK。为什么呢？如果在 PK 中有四个列，那么参考这个表的每个表也需要同样

的四个列。它不仅仅需要更长的时间建立这四个列之间的联系，而且你还要对数据存储进行复制，否则就会被消除。

以下是在你从候选键中选择 PK 时，应该遵循的一些规则。

- 避免使用字符串函数。
- 尽量使用整数数据。
- 避免用复合主键

列出了这些规则后，我们再看看一个表，判断使用哪个列来作为我们的主键 PK。图 3.1 表示产品表。该表有一系列的候选键，首先就是型号（model number）。但是，型号只用于具体的制造厂商。所以最佳选择就是包括型号和制造商的复合键。该表中的另外一个候选键为 SKU。SKU（stock-keeping unit）编号通常是只能被购买或出售该产品的公司识别，不包括生产商。

图 3.1：需要主键的产品表

我们看看每个候选键是不是违反了规则。第一个候选键（型号和厂商）违反了所有规则。这一数据属于字符串型数据，还是一个复合键。所以我们只能选择 SKU，它正好可以鉴别该行，它是一个整数并且是一个单独的列。

现在我们已经识别了主键，那么怎么在 SQL Server 中设定它呢？有很多创建主键的方法，你可以根据表的情况来选择采用什么样的方法。首先，我们一起来看看在创建表的同时怎样创建主键。以下是创建表同时完成 PK 的脚本。

```
CREATE TABLE Products(
sku                int            NOT NULL PRIMARY KEY,
modelnumber    varchar(25)          NOT NULL,
name           varchar(100)         NOT NULL,
manufacturer    varchar(25)          NOT NULL,
description    varchar(255)         NOT NULL,
warrantydetails varchar(500)        NOT NULL,
price            money(10, 0)          NOT NULL,
weight             decimal(5, 2)      NOT NULL,
shippingweight     decimal(5, 2)      NOT NULL,
height               decimal(4, 2)    NOT NULL,
width               decimal(4, 2)    NOT NULL,
depth                 decimal(4, 2) NOT NULL,
isserialized             bit        NOT NULL,
status             tinyint          NOT NULL,
)
```

（作者：SearchSQLServer.com 译者：April 来源：TT 中国）

在 SQL Server 数据库中加强数据的完整性（二）

你会发现 PRIMARY KEY 语句就在 sku 列的定义得后面。该语句在 sku 列中的表增加 PK，这样既简单、速度又快。

但是，这种方法有一个内在的问题。SQL Server 在数据库中创建 PK，每个 PK 都有一个相应的名字。但是我们并不指定名称，SQL Server 就弥补上了一个。PK_Products_30242045 就是这种情况。这个名称基于表名和一些任意的数字。从表面上看来，这并不是一个很大的问题，但是如果你之后从该表中删除 PK 了呢？如果你适当改变对环境的掌握，那你首先还要为从服务器上摒弃这个键再在创建一个脚本。以前的测试证明了在摒弃这个键时其他洞悉都不会暂停，你只需要在产品上运行这个脚本。问题是如果你用这里的脚本创建了该表，PK 在每台服务器上的名字就不一样了，并且你的脚本也会操作失败。

当你创建这个键的时候如何命名呢？你怎么命名你的键很多时候是由你决定，但是我们在第七章中提供了一些命名指南。这样我们用 pk_product_sku 命名我们的 PK。我们建议的最佳方案就是用这种方式明确命名你所有的外键。我们用以下的脚本从 sku 列定义中清除 PRIMARY KEY 语句并在表定义的末尾增加一个 CONSTRAINT 语句。

```
CREATE TABLE Products(
sku int NOT NULL PRIMARY KEY,
modelnumber varchar(25) NOT NULL,
name varchar(100) NOT NULL,
manufacturer varchar(25) NOT NULL,
description varchar(255) NOT NULL,
```

```
warrantydetails varchar(500) NOT NULL,
price money(10, 0) NOT NULL,
weight decimal(5, 2) NOT NULL,
shippingweight decimal(5, 2) NOT NULL,
height decimal(4, 2) NOT NULL,
width decimal(4, 2) NOT NULL,
depth decimal(4, 2) NOT NULL,
isserialized bit NOT NULL,
status tinyint NOT NULL,
CONSTRAINT pk_product_sku PRIMARY KEY (sku)
)
```

最后一点但并不是最不重要的一点就是，如果该表已经存在了，你还会给主键增加什么呢？首先，你必须保证列中的数据符合主键规则。它不包括 NULL，每个行也必须是唯一的。然后，用另一个简单的脚本就可以了。

```
ALTER TABLE Products
ADD CONSTRAINT pk_product_sku PRIMARY KEY (sku)
```

等等，还有。在这里我们使用 sku 列很好，但是我们还要谈一下其他的 PK 选项。如果你检查整个数据库并在 Products 表上定义 PK，那么你在每个表中得到的就是包括外键的不同的列名。这并不是一间坏事，但是它意味着如果你想增加含有外键的另一列或者需要写一些代码来连接表，你就必须查询数据类型和列名。

那么，如果你所有列表中的 PK 都有相同的名字不是一件好事吗？例如，数据库中的每个表都有一个叫 objectid 的列，并且该列仅有唯一的任意整数。在这种情况下，你就可以在 SQL Server 中用到识别列管理你的整数 PK 值。识别列就是自动在表中增加并插入数字的列。当你的创建了一个识别列（identity column），你就要指定 seed、初始值和

增量，这样每次新记录增加时它就是增量。最普遍的是，seed 和增量都被设为 1，就是说每个新的行都有同一值，也就是它要比之前的值要高 1。

另一个任意的 PK 选项就似乎 GUID。当你需要在数据库之间复制数据时，最通常的情况就就是用 GUID 作 PK，你需要从其他数据库复制过来的数据和现有数据不相冲突。如果你改用恒等式，你就需要用 seed 值消除冲突，例如，1，000，454 这个数字在两个数据库之间就很容易被用到，在复制数据时就出现了冲突。GUIDs 的缺点就是它们比整数要大，并且人们不容易读懂。还有，PK 经常是集群式的，就是说它们是按照顺序来存储的。由于 GUID 是任意的，每次你增加数据时，都会将数据插入到 PK 中间。这也增加了操作了额外负担。在第十章我们还将讨论集群式的 PK 和非集群式的 PK。

我们已经讨论了所有的 PK 选项，最常见的情况就是我们使用的是恒等式列。它们很容易设置并提供了表之间的一贯性。不管你用的是什么方法，都要认真它的考虑优缺点。用错误的方法创建一个 PK 不仅会让你很难编写数据库代码，而且还会使性能降低。

外键

有了主键，外键在 SQL Server 中和在逻辑设置中的运行原理是一样的。外键就是和主键相应的并能建立关系的列或列组合。有相同的数据的列作为主键存在于外键中。正是如此我们才强烈强烈建议你不要用合成主键，这样不仅会给你带来大量的数据复制工作，而且还会在你连接表时增加额外的负担。回到刚才的 employee 和 vehicle 例子，它含有一些抽样资料的表。

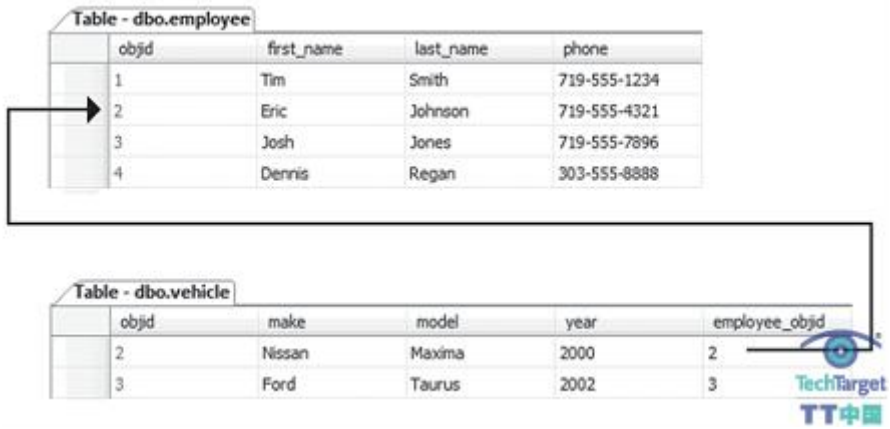

图 3.2 employee 和 vehicle 表显示表之间的关系

你能看出，这两个表都有 objid 列。identity 列就是我们的主键。此外还要注意 vehicle 表有一个 employee_objid 列，该列中包含已经匹配了交通工具的员工的 objid。在 SQL Server 中，vehicle 表中设立了外键，它主要是用来保证你增加到 employee_objid 列中的值是有效值，并且在 employee 表中有相应值。

（作者：SearchSQLServer.com 译者：April 来源：TT 中国）

在 SQL Server 数据库中加强数据的完整性（三）

以下脚本创建了 vehicle 表。你会发现这个脚本和创建先前的那个表的脚本有些不同。首先，在我们设立 objid 列时，我们用的是 IDENTITY(1，1)语句创建的一个恒等式，列中的增量为 1。第二，另外一个 CONSTRAINT 语句增加了外键的关系。在创建外键时，你在参考表中指定包含外键、参考表一个列或列组合和包含主键的列组合。

```
CREATE TABLE dbo.vehicle(
objid int IDENTITY(1, 1) NOT NULL,
make varchar(50) NOT NULL,
model varchar(50)NOT NULL,
year char(4) NOT NULL,
employee_objid int NOT NULL,
CONSTRAINT PK_vehicle PRIMARY KEY (objid),
CONSTRAINT FK_vehicle_employee
FOREIGN KEY(employee_objid)
REFERENCES employee (objid)
)
```

如果你的主键位置适当，创建外键就很不切实际了。你只要在参考表中创建合适的列并增加外键。在第二章开始时我们已经谈到，如果你在设计时需要，表中同样的列可以同在主键和外键中。

创建外键时，你还能够指定升级或删除父级表时做什么。通过默认，如果你想删除父级表中的记录，由于它会造成参考表中的行与行之间的孤立，所以删除操作就失败了。一个孤立的行就是存在于子级表中没有父级表与之对应的表。它能在数据模块中造成两个问

题。在我们的 employee 和 vehicle 表中，vehicle 表中的 NULL 值表示 vehicle 没有和 employee 匹配。但是考虑到在表中保存了命令和命令的一些细节，这种情况下，命令细节表中孤立的记录就没有用。你就不知道那条命令是属于哪一行的。

你还可以选择其他的方法来避免删除失败。首先，你可以进行删除造作级联，就是说 SQL Server 将删除你想删除的所有子级行和父级行。采用该选项时，你千万要小心。如果你有一些关系并激活了级联删除，你就可以通过删除单个的记录删除很多数据。

第二个选项就是让 SQL Server 对参考表中的 NULL 值设置外键列。该选项创建了孤立的选项，这在前面也已经讨论过。第三，如果有的默认值的话，你可以让 SQL Server 设置外键列返回到列中的默认值中。如果要更新主键值本身的话，你还可以采用其他的选项。同样，SQL Server 能够：（1）进行级联更新，这样子级行就指向正确的、带有新键的父级行；（2）将外键设置为 NULL；或者（3），设置外键恢复其默认值。通常我们并不建议你改变主键的值，但是在很多情况下你会发现你需要这么做。如果你发现在那种情况下需要这么做，那就考虑在外键上设置更新规则。

约束

SQL Server 包含一些加强数据完整性的约束条件。约束，就如同名字暗含的一样，常常用来约束添加到列中的值。我们已经讨论过 SQL Server 中的两种约束：主键和外键。主键约束数据，这样副本和 NULL 值就不会存在于列中。外键保证了添加的值存在于参考表中。你还能实施其他的约束保证数据完整性或加强事务规则。

唯一性约束

唯一性约束和主键约束一样，它们确保副本不在列或列集合中。它们在没有包含主键的列中进行配置。唯一性约束和主键之间的区别是什么呢？在技术角度来看，这二者之间唯一的不同就是唯一性主键允许你添加 NULL 值。但是由于这些值是唯一值，你只能在整个列中添加一个 NULL 值。我们说识别主键时，那是在说候选键。因为候选键还应该识别

行，你应该在候选键上设置唯一性约束。增加唯一性约束和增加外键的方法一样，都是用类似以下的约束语句：

```
CONSTRAINT UNQ_vehicle_vin UNIQUE NONCLUSTERED (vin_number)
```

检查约束

检查约束限制了逻辑表达式添加到列中的值。一个逻辑表达式就是任意一个能得出 TRUE 或 FALSE 值的 SQL 表达式。该表达式可以是任意有效的 SQL 表达式，从简单的比较到复杂的函数。例如，我们想限定添加到工资表中的值。我们用以下表达式对数据进行判断：

```
salary >= 10000 and salary <=150000
```

此行就否决了小于 10000 或大于 150000 的值。

每个列都包含多个检查约束，你或者还可以参考共有一个查询的多个列。在检查约束进行数据判断或评估时，它会允许评估结果正确的值。也就是说如果你的查询判断数据为 NULL，那么该之就可以接受。因此如果你将 NULL 值添加到设计中，但是该检查约束返回的是未知的插入值。这项功能是设计好的，但是它也会导致一个始料未及的结果，所以我们要警惕。

```
ALTER TABLE dbo.Products
ADD CONSTRAINT chk_non_negative_values
CHECK
(
weight >= 0
```

```
AND (shippingweight >= 0 AND shippingweight >= weight)
AND height >= 0
AND width >= 0
AND depth >= 0
)
```

因为那些包含负数的列没有任何意义（重量和高度不能是负数），我们就增加了约束保证数据完整性。现在你想插入负数值时，SQL Server 就会返回以下的错误并且拒绝插入数据。约束还防止了装货重量比实际重量要小。

```
The INSERT statement conflicted with the CHECK constraint
"chk_non_negative_values"
```

（作者：SearchSQLServer.com 译者：April 来源：TT 中国）

在 SQL Server 数据库中加强数据的完整性（四）

你们可以看出，我们创建了一个约束主要检查所有必须包含非负值的列。这种方法唯一的缺点就是很难找出违反约束的数据。但是想像一下如果约束更加复杂，包含的列更多，这种情况很容易发现负值。你只知道在约束中的某个列违反了约束，你就要检查所有的数据找出问题所在。使用哪种方法取决于复杂性和个人喜好。

实施参考完整性

现在我们已经讨论完了 PK，FK 和约束，我们要讨论的最后一个问题就是如何用它们来实施参考完整性。幸运的是，如果你理解了如何创建我们讨论过的每一个对象，那就可以直接用它们实施了。

一对多的关系

一对多的关系是你在数据库中用到的最普遍的关系，用一对多的关系在表中创建外键会节省大量的工作。要创建你所需要的关系，你就必须保证包含外键的列被设置成不允许 NULL 值。不允许 NULL 需要在列中插入一个值，并且增加外键要求该值包括在相关表的主键中。这种类型的关系贯彻了“一或多对一”的基数。 换句话说，你可以有一个单独的行但是你可以拥有无数个行。（本章后面我们会探讨安装高级基数的方法）。外键列中允许 NULL 能让你选择关系，也就是说，我们并不要求参考表和数据相关。如果你在表中跟踪计算机，并用其中的一种关系定义谁正在使用这台计算机，那么外键中的 NULL 就会显示没有被员工使用的计算机。

一对一的关系

一对一关系和一对多关系的实施的方法完全一样。你还会创建一个主键和外键，在这一点上的问题就是 SQL Server 还会允许用户在外键表中插入很多行参考主键表。没有什

么方法能通过默认将数据约束到一对一的关系中。要实施一对一的关系，你就必须要有一些创造性。

第一项选择就是写存储程序（本章下面的内容会讨论更多的存储程序）来进行所有的操作工作，然后增加逻辑阻止表中增加另外的行。这种方法在很多情况下都适用，但是如果没有存储程序你又需要将数据直接下载到表中，这该怎么办呢？实施一对一的关系的另外一种方法就是使用触发器，我们在这里简要讨论了一下。基本上，触发器就是一个能在插入语句之后或代替插入语句的执行代码。用这种方法，你就可以这种一对一的关系进行重新插入。

此外，也可能是最简单的方法，就是你可以在外键中增加唯一性约束。就是说外键中的数据是主键中的值，每个值都只能在参考表中出现一次。这种方法可以很有效地创建被 SQL Server 管理和增强的一对一的关系。

多对多的关系

多对多的关系可能是最复杂的关系之一。即使你可以让两个实体之间拥有一种多对多的关系，你可不能在两个表之间创建多对多的关系。要创建这种关系，你就必须创建第三个表，即连接表以及两个一对多关系。

我们通过一个例子看一下它怎么运作。你现在有两个表，一个为 Student，另一个为 Class，每个表都包括它们的主键 identity（即 objid）。这种情况下你需要多对多关系，因为一个学生可能在以上的班级中，一个班级也可能有一个以上的学生。创建这种关系，你之需要创建有两个列的表：一个包含 student_objid，另一个包含 class_objid。这样你就可以将连接表对 Student 表、另一个连接表对 Class 表之间创建一对多的关系。图 3.3 表明了这种关系。

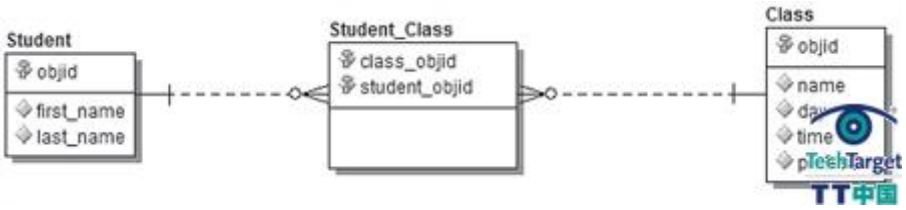

图 3.3 Student 和 Class 表之间的多对多的关系

你会发现这种设置的一些东西。首先，除了是外键，这些列一起能用作 Student_Class 表的主键。怎样才能创建这种多对多的关系？只要它们不违反主键，这个连接表就能够包含主键。也就是说你能够将每个学生和所有他的班级相对应，你也可以将一个特殊班级中所有的学生和那个班级相对应。也就是多对多的关系。

可能听起来很复杂，但是如果你创建了一种多对多的关系并在表中增加了数据，那么它就变得很清晰了。真正弄懂它的最佳方法就是实践，当我们在第九章中建立物理模型时，我们就在更进一步了解多对多的关系，包括一些让它们变得很有用的方法。

在第二章中，我们谈到了基数。基数仅用来介绍一个表和另一个表相关联的行数。基数通常是由客户的事物规则衍生而来的。有了一对一的关系，SQL Server 本身没有支持基数的方法。用主键和外键，你很容易就能够增强一或多对多、零或多对多、或者一对一的基数，就像之前我们讨论的一样。

如果每个父级记录只能包括有限的子级记录，你要创建一种关系该怎么做？例如，用 employee 或 vehicle 表，你可能会想限制数据，这样每个员工就至多有能够和五辆车匹配。此外，员工根本就没有必要要一两小汽车。这种关系的基数就是零对五对多。要增强这种需求，你就要进行一些创新。在这种情况下，你可以使用触发器计算出和一个员工匹配的汽车数量。如果与员工匹配的汽车的数量超过 5，那你就要重新插入。

每种情况都是唯一的。在一些情况中你可能会用查询约束或者另一种 PK、FK 组合来实现基数。你需要检查决定的最佳方法的条件是什么。

（作者：SearchSQLServer.com　译者：April 来源：TT 中国）

SQL Server 和 T-SQL 数据操纵（一）

设计

除了那些用来保存数据和实现数据完整性的对象之外，SQL Server 还提供了一些允许你编写数据操纵代码的对象。我们可以用这些对象在数据库里进行数据插入、更新、删除或者读数据，还可以利用它们来实施事物规则和实现高级数据完整性。你能在 SQL Server 中构建“应用软件”。一般这些应用软件都很小并且通常以某种方式提供服务或更大的应用软件操纵数据。

存储过程

最常见的情况是，我们 SQL Server 中编写代码时，你就会同时进行存储程序的操作（SP）。SP 很容易编写，还能保存 T-SQL 代码。SP 是在视图中进行汇编的，在第一次执行计划时 SP 和视图很相似。二者之间的区别就是：除进行数据抽取之外，SP 还能执行 T-SQL 代码和参数。它们和其他程序语言中的模块很相似。你可以用一个程序，并让它执行操作，或能传递参数并从 SP 中返回参数。

和列一样，我们配置参数来允许一个具体的数据类型。 所有相同的数据类型都用于配置参数，它们限定了一种你能将参数迁移到 SP 的数据类型。参数一般有两种类型：输入和输出。输入参数给 SP 提供它们在执行参数期间的数据，输出参数则将数据返回到流程程序（calling process）。除了检索数据外，输出参数还可以用来给 SP 提供数据。你可以在设计员工数据 SP 时做这些事情，如果确实有这个员工就进行数据更新，如果这个员工不存在就插入一个新纪录。在这种情况下，你可以用一个 EmployeeID 参数映射员工主键。该参数会接受你想更新的这个员工的 ID，还将在你插入一个新员工时返回这个新员工的 ID。

SP 还有一个返回值，该返回值能将整数返回到 calling process 中。返回值通常用于给 calling process 提供有关存储程序操作成功的信息。返回值和输出参数之间的不同就是返回值没有名称并且你只能在每个 SP 中获得一个返回值。此外，即使你没有指定返回值，SP 也总是在返回值中返回一个整数。通过默认，一旦你指定了其他的东西，SP 就返回 0 (zero)。正是这个原因，指定成功操作的时候就会用到 0，非零值就会用来返回错误条件。

SP 有很多用途，最常见的就是管理输入和恢复数据。通常 SP 用来映射你存储的实体。如果在你的数据库中有一个学生数据，你也许就名为 sp_add_student、sp_update_student 以及 sp_retrieve_student_data.的一些 SP。这些 SP 就会包含允许你指定最终要写入表的所有学生数据。

如同视图一样，SP 为用户简化了数据库的复杂性，比仅仅进行重复性运行 T-SQL 要更加有效。另外，如果你要更改你的数据库，用 SP 你就不需要更新应用软件代码。只要 SP 接受相同的参数并在你更改之后返回同样的数据，你的应用软件代码就不需要改变。在第 11 章中我们将详细介绍使用存储程序的方法。

自定义函数

和程序语言一样，T-SQL 提供了自定义函数(UDFs)。UDFs 用于获取输入参数、执行操作并将结果返回到 calling process 中。听起来是不是和存储程序一样？的确是一样，但是它们之间还有一些重大的区别。第一件事就是你要注意 UDFs 的命名方法。我们看一看下面命名 SP 的代码。

```
DECLARE @num_in_stock int
EXEC sp_check_product_stock @sku = 4587353,
@stock_level = @num_in_stock OUTPUT
PRINT @num_in_stock
```

你还会发现：第一、你必须用一个变量来保存存储程序的返回值。如果你之后是用的这个值，你就需要使用这个变量，这很简单。现在我们调用 UDF 返回同样的信息：

```
DECLARE @num_in_stock int
SET @num_in_stock = dbo.CheckProductStock (4587353)
PRINT @num_in_stock
```

现在我们开看看调用返回相同信息的 UDF。

这个代码看起来很相像，但是调用的这个函数和其他程序语言中调用的函数一样。你可能还会反问自己：“它们之间有什么区别？”除了调用函数并将返回值迁移到变量中之外，你还能够调用内置于其他代码中的 UDFs。想一想以下返回新的 employee ID 的 UDF 例子。这个调用函数就内置于 employee 表的插入语句中。用这种方法调用 UDFs 就能避免写一些多余的代码来存储后来用到的返回变量。

```
INSERT INTO employee (employeeid, firstname, lastname)
VALUES (dbo.GetNewEmployeeID(), 'Eric', 'Johnson')
```

第一点很大的区别就是 UDFs 是它们返回的一种数据。 能够返回单个值的 UDFs 叫做纯量函数。这种函数返回的数据可以定义为除 text、ntext、image 和 timestamp 之外的任何一种数据类型。扼要说来，我们刚才的所有的例子都包括纯量函数值。

UDFs 还能够定义为运算符计算表值函数（TVF）：即能够返回表数据类型。另外，TVF 还能在其他 T-SQL 代码中调用，并且能当作表来用。我们能运用以下的代码将 employee ID 迁移到这个函数中并返回表。

```
```

```
SELECT * FROM dbo.EmployeeData(8765448)
```

你还可以将计数表函数和其他函数或基表相连。最初开发人员用 UDFs 编写数据库的 T-SQL 代码，但是你还可以用它来实施模块中的事务规则。UDFs 还能够用于检查约束或者触发器帮助你保持数据的完整性。

触发器

触发器和约束是物理数据库中加强数据完整性和事务规则两种最普遍的方法。触发器用 T-SQL 脚本保存。和存储程序很相似，我们运行触发器就是为对表或试图发出 DML 语言（而不是 SELECT 语言）。SQL Server 中有两种 DML 触发器。

（作者：SearchSQLServer.com 译者：April 来源：TT 中国）

SQL Server 和 T-SQL 数据操纵（二）

AFTER 触发器只能存在于表中，它可以用来处理 DML 语句，完成操作以后，触发器代码就开始运行。例如，如果程序要在表中插入一个新员工，就要用到插入触发器。触发器中的代码发出插入请求部分任务完成以后就开始运行了。管理事务部属于本章范围，但是你应该知道，因为触发器和 DML 语句在相同的环境中运行，你可以对受影响的数据进行更改，直到重新执行这个语句。如果不符合事务规则要求，AFTER 触发器对于改变事务规则和取消修正就很有用。

在执行 AFTER 触发器的过程中，你要访问两个虚函数表（virtual table），一个为 Inserted，另一个叫 Deleted。Deleted.表中包含修正行的副本，和它们在删除或更新语句之前一样。Inserted 表在插入或更新之后有着和基表相同的数据。当你仍然要参考 DML 语句前后的数据时，这样的安排就允许你在基表中修改数据。

这些特殊的临时表只有在触发器代码执行时才能够使用，也只能够通过触发器来处理。在创建 AFTER 触发器时，你可以用一个触发器进行插入、更新和删除。换句话说就是我们能够设置一个触发器同时进行插入和更新，并且还能配置一个不同的触发器进行删除。另外，你可以让多个触发器运行同一个语句。例如，两个触发器就可以进行更新。如果你让多个触发器运行同一个语句，那对这些触发器的顺序有限定。用系统存储程序 sp_settriggerorder，你就可以指定先启动哪一个触发器，最后启动哪一个触发器。否则，它们就在中途启动。实际上，这不是一个很大的问题。我们还见过用两个以上的触发器运行同一个指定的语句这样的表。

INSTEAD OF 触发器属于不同的类型。这些触发器是在你所希望方式中运行的。INSTEAD OF 触发器中的代码在 DML 语句中启用，而正是这些语句才能启用触发器。和 AFTER 触发器不一样，INSTEAD OF 触发器能够定义视图和表。你可以用它们来克服视图有多个基表的界限。和之前提到的一样，如果你限定了更新结果只影响单个的基表，你就可

以更新你的视图。你还可以用 INSTEAD OF 触发器更新一个视图的所有列并使用触发器对基表进行适当更新。你还可以用 INSTEAD OF 触发器通过完全改变 DML 语句执行操作实现高级数据的完整性或事务规则。

当然，你还能够控制触发器嵌套和递归式行为。随着嵌套式触发器的开启，一个触发器就执行一个 DML 语句并引起另一个触发器的开启。例如，在 TableA 中插入一行就会引起 TableA 插入一个触发器开启。这就是触发器嵌套(一个触发器引起另一个触发器的开启)，这就是默认行为（default behavior）。伴随着嵌套触发器的开启，SQL Server 允许嵌入 32 个触发器。无论有没有设置嵌套的选项，INSTEAD OF 触发器能够进行嵌套。

CLR 整合

从 SQL Server 2005 发布时起，我们就了解如何和.NET Framework Common Language Runtime（CLR）进行结合。简单地说，CLR 整合能让你在 SQL Server 对象里用.NET 程序设计语言。你可以创建存储过程、自定义函数、触发器。CLR 自定义类型能在 Microsoft .NET 里多种高级语言。这一层面的程序设计不属于本专题的内容，但是你需要注意 SQL Server 有使用 CLR 的性能。很有可能你会碰到一位想用 CLR 的开发人员，或者你发现自己需要实施一个用标准的 SQL Server 对象和 T-SQL 语言不容易实现的、复杂的事务规则。所以如果你懂代码或有一位懂代码的朋友，你就可以用 CLR 创建一个函数增强复杂规则。

（作者：SearchSQLServer.com　译者：April 来源：TT 中国）

SQL Server 中的父表和子表（一）

我们在第二章中讨论过父型和子型。这些实体有多种真实的模块。例如，我们也许有一个父型电话、附带有一些有线或无线的子型电话。我们将这些对象区分开放进一个子型集群里。因为即使都是电话，不同的类型都需要我们归为不同的属性。例如，一部无线电话，我们就需要知道听筒能够在多大的范围内接听以及接电话的频率。对于一部有线电话，你就要看看线的长度。这些区别都属于子型，这些电话之间的共同点包含在父型特征之内。

如何在 SQL Server 中实施子型集群？你有三种选择。首先就是创建一个单独的表表示父型特征并包含所有子型特征。第二种选择就是给每个子型创建表，给每个子表增加父型特征。第三，你可以同时创建父表和子表，和逻辑建模的方式一样更加有效地实现子型集群。

要判断哪种方法正确，你就必须看清楚保存的这些数据。我们会仔细考虑这三种选择中的每一种，并考虑到你使用他们的原因以及每种选择的优点和缺点。

父表

当子表中没有包含或者和父表中的数据没有区别的时候，不可以用采用这一选项。例如，一个保存了员工数据的集群。在建模时，你会发现同时也付给了小时工工资。所以你就要决定用子表和父表进行区别。在仔细考虑所有的要求之后，你决定这两种类型之间的真正的区别就是：你要保存每个公司付给工资的员工的年工资；你还要保存小时工的工资以及小时工上班时间的长短。

在这个例子中，子表之间的区别很小，所以你可以用子表创建子型集群。在这种情况下，你可以创建一个单独的、包括所有员工类型的表，它还包含子表的三种所有的特征，即工资、小时工资以及上班时间。无论你何时插入一个小时工，你就需要把这个数据添加

到每小时费用和上班时间（小时）这两列中。那么工资这一列就会变成 NULL。而对于有薪水的员工，你要做的事情刚好和它相反

用上面这种方法创建这种图表类型会让你很轻松就找到员工数据，因为所有这些数据都在同一个地方。唯一的缺点就是你必须实现员工类型和他适当的列之间的逻辑性。只有在子型实体其他的特征很少的时候，父表的实施才能行之有效。如果它们之间有很大的区别，那你就可以设定指定行的列为 NULL，这样它就会用一种较好的方式将数据汇集在一起。

子表

子表中包含的数据和一般的数据不一样，这主要是因为子表很小，你很有可能安装子表本身。这实际上是一种相反的数据布局，它会提示你只能用父型模式。

即如你现在在为相机零售店创建一个表。你可以为这个店里销售的产品建一个子集群，因为每个产品都属于不同的类型。你需要保存每种产品的产品型号，物料编号和产品的实用性，也些也就是它们相似的地方。对于相机来说，你还需要知道它最快的快门速度、每秒帧数、反光镜大小、电池型号、距离物体的最近距离和最小光圈。并且三脚架也有一些数据，你需要保存最小和最大的高度、在哪个平面上可以安装枢轴、镜头类型。每个购买了相机的人知道这里我列出的这些区别还仅仅是表面上的。你需要了解每种类型的一些其他特征来准确介绍所有选项。

所有的这些属性对每种子型来说都是唯一的，它们之间也只有很少的共同点，这样就造成你只能实施子表。你在做这些时，每个子表都会自己存储共同的数据。也就是说，相机、镜头和三脚架表有相应的列保存模式数据、SKU 型号和实用性。你用这种方法查询实施的数据，就需要支持你查询你要找的这种产品的合适的表。

（作者：SearchSQLServer.com　译者：April 来源：TT 中国）

SQL Server 中的父表和子表（二）

父表和子表

你可能会想：如果有一些共同的属性并且子表之间有很大的不同，你就需要同时安装父表和子表。这里有一个很好的例子：子型集群存储的是客户的支付信息。不论你的客户的支付形式是电子支票、信用卡、礼券还是现金，你都需要知道一些其他的信息。任何一种支付形式你都要知道：支付的人是谁、什么时候收到的款、数量以及支付状态。但是这些支付形式中每一种都要求你去了解支付的一些细节。如信用卡，你就要知道卡号、卡的类型、安全密码以及有效期。如果是电子支票，你就需要知道银行帐号、流水号、支票号码，或者是司机的执照号码。礼券就很简单了，你只需要知道卡号和余额。如果是现金，你就不需要保存其他的数据。

以上这种需要同时安装父表和子表，Payment 表可能包括很多的细节、单个的信用卡、礼券，支票表会包含每种支付形式相应的信息。我们没有创建现金表，因为我们不需要保存 Payment 表之外的其他现金支付信息。

用这种方法安装子型集群，你还要保存子表之间的区别，通常一个较短的密码在父表中存储成一个列指定适当的子表。我们建议你尽可能使用单个字符，因为它们很小并且还给人们提供了比数字提供的更多的内容。在这个例子中，你可将信用卡保存为 CC，将礼券保存为 G、E 为电子支票、C 为现金（注意我们用 CC 来保存信用卡来区别现金）。在查询支付信息时，你可以加上基于这样一种鉴别工具的恰当的支付类型。

如果你只需要从父表和子表中查找数据，这种方法有两大优势：你只需要查找一个表，你也不需要检索其他数据。但是，另一方面就是如果你同时要从子表和父表中查找数据，你就必须判断你要哪个子表，然后将这连个表连接起来。另外，你可能会发现你还需要父表和多个子表的信息，这将增加你的查询量，因为你要将多个表连接起来。

子表和父表：最后一件事

安装子表和父表方法有时可以灵活。如果你要花大量的时间来了解数据、了解将数据拆分进多个表中而不是将它们捆绑得更紧的意义，你就应该能决定最佳解决方案。不要害怕产生一些测试数据、通过性能测试运行多种选择方式来确保你做出正确的选择。我们在建立物理模式时会用到子型集群和适和其他复杂情况的选项。

总结

在本章中，我们谈到了 SQL Server 中一些可用的对象，你可以用它们来安装你的物理模式。了解这些对象在很多方面都很重要。你当你在设计你的逻辑模式时必须记住这一点，这样你才能牢记 SQL Server。这一点在你之后建立和安装物理模式时也会起到重要的作用。可能不会对你建立的每个数据库用 SQL Server 中的每一个对象，但是你要清楚你的选择。我们会在以后向大家介绍如何创建物理模型，在那个时候我们会介绍你如何用这些物理模型解决问题的方法。

在下一章中，我们会谈到 标准化（normalization），那时我们会通过例题方案、挖掘真实事例介绍本书最基本的部分。

（作者：SearchSQLServer.com 译者：April 来源：TT 中国）