09 年必会的十大 SQL Server 开发技巧

09 年必会的十大 SQL Server 开发技巧

本专题为 SQL Server 开发 10 大技巧总结。如果你在工作过程中不了解这些开发技巧，可能会大大影响你的工作兴趣。无论你是现在是不是正将日期/时间数据类型转换成为字符数据类型，还是在 SQL Server2005 中操作 DATETIME 和 SMALLDATETIME，或者用存储程序查找 SQL Server 表大小或用 XQuery 检索 XML 数值等等，本技术专题中的 10 大技巧都是今年你必须了解、知道的话题。

1、SQL Server 中日期/时间值到字符类型的数据转换

本文将介绍 Transat-SQL 支持的两种内置、从时间/日期数据数据和字符数据相互转换的方法：隐式转换和显式转换。本文将向你逐步介绍每种方法的使用方法。在本文中专家还提出了在转换日期/时间值时是如何用到 CAST 和 CONVERT 函数的。

- SQL Server 中日期/时间值到字符类型的数据转换（一）
- SQL Server 中日期/时间值到字符类型的数据转换（二）

2、SQL Server 2005 的 DATETIME 和 SMALLDATETIME 基础

了解 SQL Server 中的 DATETIME、SMALLDATETIME 和 TIMESTAMP 数据类型通常不是一个很直接的过程。另外，你还能在文中了解数据是如何存储在 DATETIME 和 SMALLDATETIME 中的、为什么 TIMESTAMP 会和这两种日期/时间数据类型不同但又经常与两种主要的日期/时间数据类型相混淆。

- SQL Server 2005 的 DATETIME 和 SMALLDATETIME 基础（一）
- SQL Server 2005 的 DATETIME 和 SMALLDATETIME 基础（二）

❖ SQL Server 2005 的 DATETIME 和 SMALLDATETIME 基础（三）

3、用存储过程查询 SQL Server 表和其它对象大小

介绍数据库中用户表大小的概况、10 个最大的索引对象，这是计算 SQL Server 中某个对象特定的磁盘空间的重要理由。Sp_SOS 这一存储程序可以代替不合适的 sp_spaceused 存储程序。Sp_spaceused 是随 SQL Server 发布的一个存储过程，它用来显示 SQL Server 对象所占用的硬盘空间。但是我们往往发现它并不能满足要求。比如，当我想要查看一个特定的 SQL Server 数据库的用户表大小概况时，或者希望看一下前 10 个最大的索引对象，或者需要计算一组表所占用的空间总大小时，sp_spaceused 并不能做到……

❖ 用存储过程查询 SQL Server 表和其它对象大小

4、正确使用 SQL Server 的 datetime 函数：GETDATE、DATENAME 和 DATEPART

Transact-SQL 包含一组函数可以用于检索当前的日期和时间或一个 DATETIME 或 SMALLDATETIME 值的某个部分。比如你可以在 DATETIME 值中提取日、月或年以及季度、周、小时甚至毫秒。在本文中，我们将学习对这些函数进行阐述并举例说明如何使用 SQL Server 中的这些函数来查询数据的方法。

❖ 正确使用 SQL Server 的 datetime 函数（一）

❖ 正确使用 SQL Server 的 datetime 函数（二）

5、在 SQL Server 2005 中使用 XQuery 检索 XML 数据

XQuery 是一个功能强大的、专门为存取 XML 数据而设计的脚本语言。SQL Server2005 支持一个允许在 XML 列、变量或参数中存取值的 XQuery 语言子集。通过调用 XML 数据类型支持的方法，就可以在 Transact-SQL 语句中使用 XQuery。其中两种方法分别是 value()和 query()——这对于在 XML 数据中检索指定的元素是特别有用的。本文将阐述这两种方法并分别举例说明。

- ❖ 在 SQL Server 2005 中使用 XQuery 检索 XML 数据（一）
- ❖ 在 SQL Server 2005 中使用 XQuery 检索 XML 数据（二）
- ❖ 在 SQL Server 2005 中使用 XQuery 检索 XML 数据（三）

6、SQL Server 2000 中监控长期运行任务的存储过程

在 SQL Server 2000 中监控长期运行的任务不是一项简单的任务，但是它是 DBA 的一项最佳实践。本文主要介绍发现易于识别长期运行任务的存储过程的方法，并且它还能通过电子邮件的方式提醒 DBA 环境可能会对 SQL Server 性能造成一定程度的影响。

- ❖ SQL Server 2000 中监控长期运行任务的存储过程

7、在 SQL Server 2005中创建 DDL 表格来审计 DDL 触发活动

在 SQL Server 2005之前，我们只能定义 Data Manipulation Language (DML)触发器。当执行 DML 语句时，如 UPDATE 或 DALETE，这些触发器就会启动。在 SQL Server 2005发布之后，现在可以定义 Data Definition Language (DDL)触发器了。当执行 DDL 语句，如 CREATE TABLE 和 ALTER VIEW，这些类型的触发器就会启动，并且这使得使用 DDL 触发器来审计 SQL Server 中的 DDL 事件更加容易了。

- ❖ 在 SQL Server 2005 中创建 DDL 表格来审计 DDL 触发活动（一）

- 在 SQL Server 2005 中创建 DDL 表格来审计 DDL 触发活动（二）

8、使用 DATEADD 和 DATEDIFF 来计算 SQL Server 的 DATETIME 值

在 SQL Server 数据库中，DATETIME 和 SMALLDATETIME 值是以整数存储的。然而，与整数不同的是，它们不能直接地进行数学运算。尽管如此，有时候还是需要在日期/时间值中添加或减去一个时间间隔。比如，你可能想在一值上加一些月数或天数，或者甚至可能是小时数。你甚至可能想比较两个日期/时间值以便确定它们之间的时间间隔，如相差的天数或年数。为了简化这些类型的计算，Transact-SQL 支持两个重要的日期/时间方法：DATEADD 和 DATEDIFF。

- 使用 DATEADD 和 DATEDIFF 来计算 SQL Server 的 DATETIME 值（一）
- 使用 DATEADD 和 DATEDIFF 来计算 SQL Server 的 DATETIME 值（二）

9、配置 SQL Server 服务代理来发送存储过程数据

在 SQL Server 2005中，Microsoft 引进了一个令人振奋的新特性即服务代理（Service Broker），同时这也给许多数据库管理人员带来了一个新的概念……

- 配置 SQL Server 服务代理来发送存储过程数据（一）
- 配置 SQL Server 服务代理来发送存储过程数据（二）

10、用 SQL Server 2005 CTE 简化查询

SQL Server 2005引进了一个很有价值的新的 Transact-SQL 语言组件：通用表表达式（Common Table Expression，CTE），它是派生表和视图的一个便捷的替代。通过使用 CTE，我们可以创建一个命名结果集来在 SELECT、INSERT、UPDATE 和 DELETE 语句中引用，而无须保存结果集结构的任何元数据。在本文中，作者将阐述如何在 SQL Server 2005中创建 CTE——包括如何使用 CTE 来创建一个递归查询——并举几个例子来说明它们

是如何使用的。注意，本文中所有例子都使用 SQL Server 2005的 AdventureWorks 示例数据库。

- 用 SQL Server 2005 CTE 简化查询（一）
- 用 SQL Server 2005 CTE 简化查询（二）

SQL Server 中日期/时间值到字符类型的数据转换（一）

在本文中，我将解释如何将 DATETIME 和 SMALLDATETIME 数据类型转换成字符数据，以及如何将字符数据转换成日期/时间数据。具体来说，本章将介绍 Transat-SQL 支持的两个内置 SQL Server 数据转换方法——隐式转换和显式转换。在前一篇的技巧《SQL Server 2005 的 DATETIME 和 SMALLDATETIME 数据基础》中，我已经阐述了 SQL Server 是如何使用 DATETIME 和 SMALLDATETIME 数据类型来存储日期/时间数据的。在本文中，我将解释日期/时间数据是如何转换成字符数据和字符数据是如何转换成日期/时间数据以及 Transact-SQL 是如何支持这两种执行这些数据转换的方法——隐式转换和显式转换。

本文的前提是假定你已具备 T-SQL 和 SQL Server 应用知识，并且该部分只涉及日期/时间数据与字符数据之间的相互转换。但是，你还可以转换其它类型的数值，如将 INT 转换为 DATETIME。虽然在大多数情况下，你的主要工作是字符到日期/时间的转换。

隐式转换数据

当你插入数据到 DATETIME 或 SMALLDATETIME 字段中时，SQL Server 会自动尝试将不同类型的数据进行转换。例如，如果你向 DATETIME 字段中插入 CHAR 值，SQL Server 将对数据作转换——如果该值是一个可以接受的格式。如果你在 CHAR 栏中插入 DATETIME 值，SQL Server 也将作自动转换。

让我们来看看几个隐式转换例子以便更好地理解它是如何工作的。为了说明这些转换，我使用了下面的代码在 AdventureWorks 示例数据库中创建 LogInfo 表：

```
USE AdventureWorks
```

```
GO
CREATE TABLE dbo.LogInfo
(
LogID INT PRIMARY KEY,
LogEvent NVARCHAR(30) NOT NULL,
Post_DateTime DATETIME NOT NULL,
Post_SmallDateTime SMALLDATETIME NOT NULL,
Post_NVarChar NVARCHAR(25) NOT NULL
)
```

在这个表中包含了三个用于保存日期/时间信息的字段：Post_DateTime、Post_SmallDateTime 和 Post_NVarChar。字段的名称反映了用于定义字段的数据类型。下面让我们在这些字段中插入数据：

```
INSERT INTO LogInfo
SELECT DatabaseLogID, [Event],
PostTime, PostTime, PostTime
FROM dbo.DatabaseLog
```

这个语句将从 DatabaseLog 表（在 AdventureWorks 数据库）中获取数据，然后插入到 LogInfo 表中。在源表的 PostTime 字段是 DATETIME 数据类型的。注意，这个字段是用于将数据插入到 LogInfo 表的每个日期/时间字段的。

在你填充好表的数据后，你可以使用下面的 SELECT 语句来获取 LogInfo 表的第一行记录：

```
```

```
SELECT * from dbo.LogInfo
WHERE LogID = 1
```

SELECT 语句返回 LogID 值为 1 的记录行的所有字段的值。下面的结果显示了数据是如何存储在表中的。

LogID	LogEvent	Post_DateTime	Post_SmallDateTime	Post_NVarChar
1	CREATE_TABLE	2005-10-14 01:58:27.567	2005-10-14 01:58:00	Oct 14 2005 1:58AM

你可以看到，每个日期/时间值都稍微有些不同。The Post_DateTime 字段存储的是完整日期和时间值。但是，正如我们所预期的，Post_SmallDateTime 字段存储一个缩短的时间（00 表示秒）。最后，Post_NVarChar 存储的是一个与其它两个非常不一样的字符串值。

默认情况下，当 SQL Server 将一个 DATETIME 或 SMALLDATETIME 值转换为一个字符值时，它使用上面显示的格式（Oct 14 2005 1:58AM)。在后面的文章中，你将会知道我们还可以将这种格式修改成其它的一些可用的格式。但是目前而言，我们要知道的重要一点是 SQL Server 是如何隐式转换日期/时间值的。现在让我们来看看 SQL Server 中的显式数据转换。

（作者：Robert Sheldon 译者：曾少宁/陈柳 来源：TT 中国）

SQL Server 中日期/时间值到字符类型的数据转换（二）

显式地转换日期/时间值，你必须使用 CAST 或 CONVERT Transact-SQL 方法。由于 CAST 方法是两者中相对简单的，因此我们从这个开始介绍。下面这个 SELECT 语句使用 CAST 方法将 Post_NVarChar 字段中的字符数据转换成一个 DATETIME 值。

```
SELECT LogID,  LogEvent,
CAST(Post_NVarChar AS DATETIME) AS Post_Converted
FROM dbo.LogInfo
WHERE LogID = 1
```

当你使用 CAST 方法时，你必须指定源字段名称（或其它一些表达式）、AS 关键字和值转换的数据类型——这里是 DATETIME。当你运行这个语句时，值就被转换了，如下面显示的结果：

LogID	LogEvent	Post_Converted
1	CREATE_TABLE	2005-10-14 01:58:00.000

(1 row(s) affected)

注意，Post_Converted 字段（别名赋给 SELECT 子句中的字段）预期是包括完整日期时间值并精确到毫秒的 DATETIME 格式。但是，秒是表示为 00.000。这是因为当 SQL Server 转换原始值时，它会去掉秒而只存储小时和分钟值。当你将值转换回 DATETIME 时，SQL Server 将秒设置为 00.000。

然而，如果日期/时间值是以作为字符串存储的并使用 DATETIME 数据所使用的格式，那么 SQL Server 就会保留秒。比如，下面的 SELECT 语句使用 CAST 方法将字符串值转换为 DATETIME：

```
SELECT CAST('2005-10-14 01:58:27.567' AS DATETIME) AS [Date/Time]
```

下面的结果显示秒和毫秒现在被保存了：

Date/Time
2005-10-14 01:58:27.567

(1 row(s) affected)

除了显式地将 DATETIME（或 SMALLDATETIME）值转换成字符数据，你也可以使用 CAST 方法将 DATETIME 数据转换成字符数据。下面的 SELECT 语句使用 CAST 功能从 Post_DateTime 字段中获取数据：

```
SELECT LogID,  LogEvent,
CAST(Post_DateTime AS VARCHAR(20)) AS Post_Converted
FROM dbo.LogInfo
WHERE LogID = 1
```

正如你所看到的下面显示的结果，句法将值转换成 VARCHAR：

LogID	LogEvent	Post_Converted
1	CREATE_TABLE	Oct 14 2005 1:58AM

(1 row(s) affected)

注意，当 SQL Server 隐式地将 DATETIMER 值转换成 NVARCHA 时，转换的值的格式现在就是你先前看到格式。

现在你了解了如何使用 CAST 方法，那么让我们接着看看 CONVERT 方法。最基本的，CONVERT 方法返回与 CAST 方法一样的结果。比如，与上面的例子一样，下面的语句将 Post_DateTime 值转换成 VARCHAR：

```
SELECT LogID,  LogEvent,
CONVERT(VARCHAR(20),  Post_DateTime) AS Post_Converted
FROM dbo.LogInfo
WHERE LogID = 1
```

然而，注意在 CONVERT 方法中的参数的顺序与 CAST 方法的是不一样的。当使用 CONVERT 时，你首先指定目标数据类型（VARCHAR），然后是源字段(Post_DateTime)的名称，它是由逗号分隔的两个参数，而不是 AS 关键字。当你运行语句时，你会得到下面的结果：

LogID	LogEvent	Post_Converted
1	CREATE_TABLE	Oct 14 2005 1:58AM

(1 row(s) affected)

结果跟前面的例子是一样的。但是，如果你想要以一定的格式显示你的日期/时间值，而不是使用目前我们所看到的格式(Oct 14 2005 1:58AM)。这时，你可以在 CONVERT 方法中添加第三个参数来指定新的格式，如下面的例子所显示的：

```
SELECT LogID,  LogEvent,
CONVERT(VARCHAR(20),  Post_DateTime,  101) AS Post_Converted
FROM dbo.LogInfo
WHERE LogID = 1
```

注意，101 已经作为第三个参数添加到方法中。当指定一个格式时，你必须使用由 T-SQL 支持的预定义代码来表示你想要使用的格式。在这种情况下，101 返回如下所显示格式的结果：

LogID	LogEvent	Post_Converted
1	CREATE_TABLE	10/14/2005

(1 row(s) affected)

Post_Converted 值现在的格式是 10/14/2005，这个也是代码 101 代表的格式。如果你想要你的结果显示为类似于 DATETIME 值所显示的格式，那么你可以指定代码 121，如下面的例子：

```
SELECT LogID,  LogEvent,
CONVERT(VARCHAR(25),  Post_DateTime,  121) AS Post_Converted
FROM dbo.LogInfo
WHERE LogID = 1
```

现在返回的结果是完整日期和时间值，精确到毫秒：

LogID	LogEvent	Post_Converted

1	CREATE_TABLE	2005-10-14 01:58:27.567

(1 row(s) affected)

T-SQL 支持多种预定义的格式。关于用于调用每个格式的格式命名和代码的完整清单，你可以在 Microsoft SQL Server Books Online 中阅读 CAST 和 CONVERT（Transact-SQL）专题。

现在让我们来看一个不同的例子。在下面的 SELECT 语句中，我们使用了 CONVERT 方法将 Post_SmallDateTime 字段栏转换成一个 VARCHAR 字段：

```
   SELECT LogID,  LogEvent,
CONVERT(VARCHAR(25),  Post_SmallDateTime,  121) AS Post_Converted
FROM dbo.LogInfo
WHERE LogID = 1
```

正如前面的例子，日期/时间值显示为 121 格式：

LogID	LogEvent	Post_Converted
1	CREATE_TABLE	2005-10-14 01:58:00.000

(1 row(s) affected)

注意，由于日期/时间值是从 SMALLADATETIME 字段中获取的，因此时间值中的秒是 00.000，这与 SMALLDATETIME 的是一样的。以下是如何以指定更短的长度截断 VARCHAR 数据类型的秒：

```
SELECT LogID,  LogEvent,
```

```
CONVERT(VARCHAR(16), Post_SmallDateTime, 121) AS Post_Converted
FROM dbo.LogInfo
WHERE LogID = 1
```

目前 CONVERT 功能的数据类型参数显示为 VARCHAR（16）而非 VARCHAR（25），与前面的例子一样。下面的结果显示值是如何被截断以便秒不再显示：

LogID	LogEvent	Post_Converted
1	CREATE_TABLE	2005-10-14 01:58

(1 row(s) affected)

这就是所有关于日期/时间值的显式转换方法。当获取这些值时，CAST 和 CONVERT 方法都是方便的工具（注意，这些方法同样可以用于转换其它类型的值）。在接下来的文章中，我将阐述如何从日期/时间字段获取特定的信息，以及如何计算日期/时间值。同时，你现在也已经掌握了如何转换这些值以及以特定格式显示它们的基本用法，这对你是非常有用的。

（作者：Robert Sheldon 译者：曾少宁／陈柳 来源：TT 中国）

SQL Server 2005 的 DATETIME 和 SMALLDATETIME 基础（一）

理解 SQL Server 的日期/时间数据类型是有一定难度的，尤其是混合使用 TIMESTAMP 的时候。在关于日期/时间的这一系列的第一部分，你将了解到关于数据 是如何存储在 DATETIME 和 SMALLDATETIME 的基础知识，以及大致地了解 TIMESTAMP 数据类型——它经常与两种主要的日期/时间数据 类型相混淆。

在 SQL Server 2005 使用日期/时间值有时候会很混淆。因为日期/时间数据类型同时存储日期和时间值，而这些值的操作并不总是一个简单的过程，或者看起来不简单的。

当操作这类数据时，对于这些数据类型有一个基本的了解是很有重要的。在本文中，我将介绍 SQL Server 中的两种基本的日期/时间数据类型——DATETIME 和 SMALLDATETIME，以及数据是如何在这两种类型中存储的。另外，我还将概 括介绍 TIMESTAMP 数据类型，这样你就可以了解到它与两种日期/时间数据类型的不同。

SQL Server 的 DATETIME 数据类型

毫无疑问，DATETIEM 数据类型对你的应用是最有用的。DATETIME 字段（或一个变量）中的值是以两个 4 位整型存储的。第一个整型表示日期，而第二个整型表示时间。

我们是从基准日期 1900 年 1 月 1 日开始的计算日期的整数的。整数表示该日期之前或之后的天数。因此，DATETIME 数据类型仅仅支持由 4 位范围整数表示的日期。这就意味着 DATETIME 字段的一个日期必须处于 1735 年 1 月 1 日到 9999 年 12 月 31 日之间。

第二个在 DATETIME 值的整数，即时间整数，存储了午夜后的 1/300-秒单位的数字。这就意味着时间是以毫秒为单位存储的，并精确到 3.33 毫 秒。因此，当你在 DATETIME 字段中插入一个值，SQL Server 将把时间转换成.000、.003 或.007 秒。下面的表显示了 SQL Server 是圆整时间值的几个例子：

Time example	Rounded to:
10:10:10.989 10:10:10.990 10:10:10.991	10:10:10.990
10:10:10.992 10:10:10.993 10:10:10.994	10:10:10.993
10:10:10.995 10:10:10.996 10:10:10.997 10:10:10.998	10:10:10.997
10:10:10.999	10:10:11.000

这个表的时间是以小时、分钟、秒钟顺序列出的，并且适当地将秒精确到毫秒。

正如你所看到的，DATETIME 字段中的两个整数一起表示一个特定日期的某一特定时间。当你从一个 DATETIME 字段中检索一个值时，它将日期和时 间显示为一系列的数字。比如，下面的 Transact-SQL 语句就是从 DatabaseLog 表中的 PostTime 字段检索数据，这个表也是 AdventureWorks 示例数据库中的一部分：

```
SELECT PostTime FROM dbo.DatabaseLog
WHERE DatabaseLogID = 1
```

PostTime 字段是配置为 DATETIME 数据类型的。当你在 SQL Server Management Studio 中检索值时，所检索到的值在默认情况下以下面的格式返回：

```
PostTime
-----------------------
2005-10-14 01:58:27.567
(1 row(s) affected)
```

注意，日期显示首先是年（2005），接着是月（10），然后是日（14）。然后日期后面是时间，也就是 1 小时、58 分和 27.567 秒。值作为一个整体，它所指的是 2005 年 10 月 14 日，大约是凌晨 1：58 的日期和时间。

（作者：Robert Sheldon 译者：曾少宁／陈柳 来源：TT 中国）

SQL Server 2005 的 DATETIME 和 SMALLDATETIME 基础（二）

SQL Server 的 SMALLDATETIME 数据类型

SMALLDATETIME 数据类型类似于 DATETIEM，只是它的值长度有限。SMALLDATETIME 值是用 2 个 2 位整数存储的。第一个整数表示日期，而第二个整数表示时间。

与 DATETIME 数据类型一样，SMALLDATETIME 数据整数也是相对于基准日期 1900 年 1 月 1 日计算的。但是，整数仅仅表示在基准日期之后 的天数，而不包括之前的日期。这就意味着 SMALLDATETIME 字段中的日期必须处于 1900 年 1 月 1 日到 2079 年 6 月 6 日之间。

SMALLDATETIME 值中的第二个整数——时间整数——存储午夜之后的分钟数。时间并不包括秒数，而且，如果值中包括秒，它们是以下面的方式圆整的：

- 小于或等于 29.998 秒的值向下圆整为前一分钟。
- 大于或等于 29.999 秒的值向上圆整为后一分钟。

下表显示了几个 SQL Server 是圆整时间值的例子：

Time example	Rounded to:
14:22:29.996 14:22:29.997 14:22:29.998	14:22
14:22:29.999 14:22:30.000	14:23

14:22:30.001	

最初这个表中的时间是按照小时、分钟、秒钟、毫秒的顺序列出的。但是，正如你所看到的，这些时间值都被圆整为时和分，而不包括秒。当你查询 SMALLDATETIME 值时，事实上你是可以看到表示秒的数字的，而且这些数字总是显示为 00，但没有毫秒。比如，下面这个 SELECT 语句使用了 CAST 方法来将 DATETIME 值转换成 SMALLDATETIME 值：

```
SELECT CAST(PostTime AS SMALLDATETIME) AS [Date/Time]
FROM dbo.DatabaseLog
WHERE DatabaseLogID = 1
```

与先前的例子一样，这个语句在 DatabaseLog 表中的 PostTime 字段中查询到数据。只是这一次这个语句返回了稍微有些不同的结果：

```
Date/Time
-----------------------
2005-10-14 01:58:00
(1 row(s) affected)
```

正如你所看到的，时间被向下圆整为 01:58。虽然表示秒的两个数字也被包括在内，但是它们总是显示为 00。（注意，CAST 方法，与 CONVERT 方 法相似，都是一个 T-SQL 显式地将一个日期类型转换成另外一个类型的方法。我将在这一系列的后面的文章中继续探讨 CAST 和 CONVERT 方法。你也可 以在 Microsoft SQL Server Books Online 上找到关于这两个方法的资料）。

（作者：Robert Sheldon 译者：曾少宁/陈柳 来源：TT 中国）

SQL Server 2005 的 DATETIME 和 SMALLDATETIME 基础（三）

SQL Server 的 TIMESTAMP 数据类型

另外一个很重要的数据类型是——TIMESTAMP。TIMESTAM 与 DATETIME 和 SMALLDATETIME 是非常不一样的。首先，它基本上 与日期或时间无关。但是，它与行版本化却有着千丝万缕的关系。我只在本文中作阐述，是因为它经常与日期/时间数据类型相混淆。

TIMESTAMP 数据类型字段自动地生成二进制值，它表的每一行提供一个版本戳记。当你每插入一个记录行时，就会有一个版本戳记被插入了 TIMESTAMP 字段中。每次你更新行，TIMESTAMP 值也随之更新。

结果，TIMESTAMP 字段是一个确定一个行最近是否被修改过的很方便的方法——当你正在开发一个支持用户并发的应用时，这是一个非常有用的特性。比 如，你可以使用 TIMESTAMP 值，通过比较 TIMESTAMP 的原始值与最近值来确定事务是结束还是回滚。如果值是一样的，那么你可以结束事务。否则 你必须回滚该事务，因为你很快便知道有另一个用户已经修改了行。

TIMESTAMP 值使用一个数据库计数器，它随着每次行的插入或更新而递增。因为 TIMESTAMP 是一个二进制值，一个包含了 TIMESTAMP 字段的行将自动存储一个类似于下面的值：

```
0x00000000000007DD
```

如果行被更新，那么值也将改变。

当操作 TIMESTAMP 栏时，必须考虑以下几点：

- 这个数据类型是一个递增的数字而且不保存日期/时间数据。
- 这个数据类型不可以用作候选键，比如主键。
- 一个表只能包括一个 TIMESTAMP 字段。
- 这个数据类型的主要目的是支持行版本化。事实上，ROWVERSION 是 TIMESTAMP 数据类型的同义词。

正如你所看到的，TIMESTAMP 字段的范围非常有限。或许除了使用 TIMESTAMP 数据类型来支持行版本化，其它的情况你都应该使用 DATETIME 和 SMALLDATETIME 数据类型，包括当你想要记录一个数据被修改的确切时间。同样的，我在此处提及 TIMESTAMP 数据类型仅 仅是为了更好得区分实际的日期/时间数据类型。

后面的文章中，我将进一步探讨 DATETIME 和 SMALLDATETIME 类型值的操作。你将可以学到如何转换日期/时间值、从这些信息中如何获取特定的信息和对值进行差计算。我甚至还将探讨 SQL Server 2008 中的新的日期/时间数据类型。

(作者：Robert Sheldon 译者：曾少宁/陈柳 来源：TT 中国)

用存储过程查询 SQL Server 表和其它对象大小

Sp_spaceused 是随 SQL Server 发布的一个存储过程，它用来显示 SQL Server 对象所占用的硬盘空间。但是我往往发现它并不能满足要求。比如，当我想要查看一个特定的 SQL Server 数据库的用户表大小概况时，或者希望看一下前 10 个最大的索引对象，或者需要计算一组表所占用的空间总大小时，sp_spaceused 并不 能做到。

所以，我创建了存储过程 sp_SOS，它是 sp_spaceused 的扩展版本，它可以用来计算 SQL Server 对象空间和执行其它的功能。

在sp_SOS中我依然保持了sp_spaceused的核心功能——如，计算数据、索引总和的算法，为一个对象保留和释放空间。同时我还增加了数据库大小计算部分，并将得到一个单独用于数据大小报告的存储程。点击下载代码列表 1：sp_SOS的完整T-SQL定义。

Sp_SOS 有 8 个输入参数，见表 1。

Variable	Data type	Nullable	Default	Default implication
@DbName	sysname	Yes	NULL	Current database 当前数据库
@SchemaName	sysname	Yes	NULL	All schemas 所有的 Schemas
@ObjectName	sysname	Yes	%	Including

				all objects in "LIKE" clause 在“LIKE”子句中包含所有对象
@TopClause	nvarchar(20)	Yes	NULL	All objects. Can be "TOP N" or "TOP N PERCENT" 所有对象。可以是“TOP N”或“TOP N PERCENT”。
@ObjectType	nvarchar(50)	Yes	NULL	All objects that can be sized. Valid values are S(system), U(user), V(indexed view), SQ(service broker queue), IT(internal table) or any combination of

				them 所有可以计算大小的对象。其中有效值包括 S（system，系统），U（user，用户），V（indexed view，索引视图），SQ（service broker queue，服务代理队列），IT（internal table，内部表）或它们的任意组合
@ShowInternalTable	nvarchar(3)	Yes	NULL	Includes internal table. The Parent excludes it in size 包括所有内部表。Parent 除外。

@OrderBy	nvarchar(100)	Yes	NULL	By object name, can be any size related column. Valid short terms are N(name), R(row), T(total), U(used), I(index), D(data), F(free or unused) and Y(type) 对象名排序，可以是任意与大小相关的字段。有效的缩写有 N（name），R（row），T（total），U（used），I（index），D（data），F（free or unused）和 Y

				（type）。
@UpdateUsage	bit	Yes	0	Do not run "DBCC UPDATEUSAGE" 不运行 “DBCC UPDATEUSAGE”。

表 1：sp_SOS 的参数变量和它们的特性。

我更喜欢使用类似公式化的样式，以使它更好地解释数字关系。例如，公式“Total(MB) - Unused(MB) == Used(MB) = Index(MB) + Data(MB)”，使用的空间是总大小与未使用大小的差，也是索引与数据的大小之和。sp_spaceused 中的“reserved”字段实际上等于 sp_SOS 的总大小。内部表是 SQL Server 2005 和 SQL Server 2008 的一个新概念。它们是父对象处理 XML 主索引、Service Broker 队列、全文索引和查询通知订阅的中间表。

在计算父对象总和时，类型 202（xml_index_nodes）和 204（fulltext_catalog_map）的内部表应该被加到总大小 中。因此，我在 sp_SOS 中设计了一个包含两个部分的临时基本对象表（##BO）。左半部分包含 6 个字段表示子对象。而右半部分是父对象。当一个父对象 拥有一个子对象，它会显示不同的模型、模型 ID、对象名和对象 ID。否则，名称和 ID 是相同的。当一个内部表显示时，它们的父名称显示在括号中以表示清晰 的关系。

对象类型的开头与 Microsoft SQL Server Books Online 一致。它们可以是系统表（S）、用户表（U）、视图（V）、服务队列（SQ）或内部表（IT）。每一种类型是由一个或多个空格、逗号或分号分 隔。分隔符的数量和顺序并不重要。如果你使用除了这些

允许的分隔符之外的字符时，sp_SOS 会报错并退出。另外这个存储过程是兼容 Unicode 和大小 写敏感的。

Sp_SOS 可以运行在 SQL Server 2000、2005 和 2008 上。在 SQL Server 2000 中，sp_SOS 报告的值是不可以更新的。@UpdateUsage 参数可以为每一次运行指定一个“DBCC UPDATEUSAGE”以确保这些值是下面报告的当前值。但是要注意它可能影响大型数据库的性能。从 SQL Server 2005 开始，sp_spaceused 总是报告正确的数值，它使 DBCC 命令不再有用了。

以下是一些如何使用 sp_SOS 的场景的典型说明：

@DbName 是你想要在 SQL Server 中查找对象空间的数据库名称。如果没有数据库名，它将会使用当前数据库。比如，如果你想快速地查看 AdventureWorks 数据库中的所有数据库对象空间，你可以运行列表 2 中的 T-SQL 语句。

```
USE AdventureWorks;
EXEC dbo.sp_SOS;
```

列表 2：AdventureWorks 数据库中的所有对象占用空间概况。注意所有的参数都是默认值。执行结果显示所有按字母排序的模型对象。

@SchemaName 和@ObjectName 这两个参数都将通配符%作为默认值。这让你对有类似模式名称或拥有者名称的对象组进行求和。我们仍然使 用 AdventureWorks 作为示例数据库，而我们想要按硬盘空间使用状况列出所有类似于“Sales”的 Sales 模型对象。我们还不想显示内部 表。我喜欢先运行“DBCC UPDATEUSAGE”更新任何不正确的值。其中使用的 T-SQL 语句类似于代码列表 3。

```
sp_SOS 'AdventureWorks', 'Sales%', 'Sales%', NULL, ' SQ, ;u v ;iT;',
'no',
'U', 1
```

列表 3：列出 Sales 模型所拥有名称类似于“Sales”的对象，按硬盘使用空间降序排列。

类似于列表 3，在列表 4 中的命令可以获取特定对象的大小信息。注意其中大小表示对象属性 GUI 显示值和 sp_SOS 或 sp_spaceused 返回值的差。此外，注意补充的父对象名在 XML 索引后面以表示它们的关系。

```
sp_SOS 'AdventureWorks', NULL, 'xml_index_nodes_309576141_32000',
NULL, 'IT',
'yes', 'N', 0
```

列表 4：检查一个特定对象的使用空间。因为它是一个内部表，因此模型名被忽略了。显然一个对象不需要排序，所以@OrderBy 参数也被忽略了。

下面是在我管理的一个 SQL Server 2000 数据库中 sp_SOS 运行的 2 个屏幕截图：

1、你可以在图 1 中看到 2 组表，ARCHIVED_HISTORY 和 HISTORY。我经常需要计算 HISTORY 表的总和。通过 sp_SOS，我可以 在结果最后计算值的总和。图 2 显示每一个相关表上执行 DBCC UPDATEUSAGE 的详细信息。箭头(= =>)表示实际的命令，后面是更新的明细。

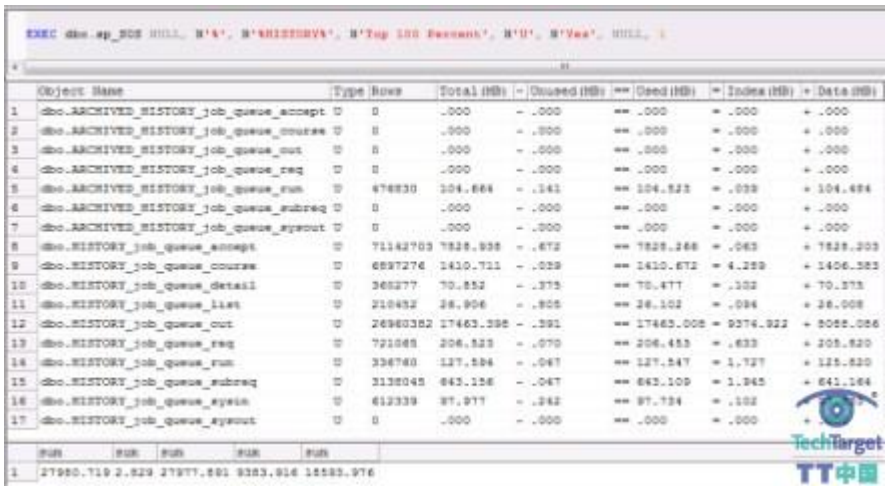


图 1：sp_SOS 显示了一个 SQL Server 2000 用户数据库的分组用户表，它们有相似的名称和统计空间大小。

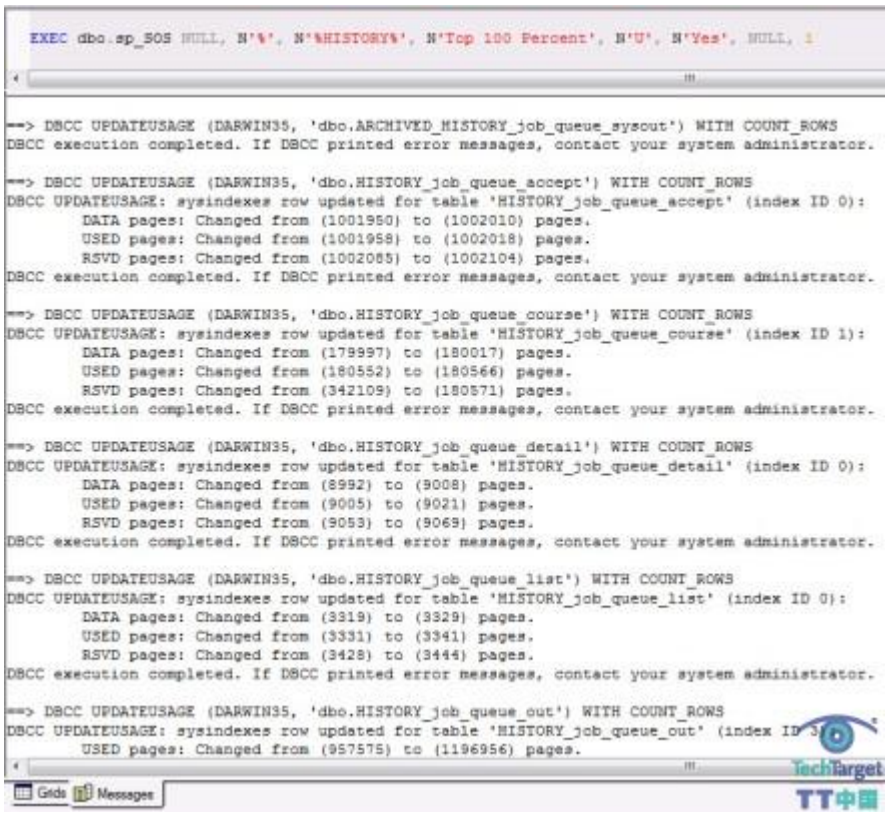


图 2：SQL Server 2000 Query Analyzer 显示 sp_SOS 代码段以及“DBCC UPDATEUSAGE”命令的详细结果。这时，大多数的对象需要更新它们的大小信息。

最后，这是 sp_SOS 在 SQL Server 2008 CTP 的运行情况。使用列表 5 的脚本，我们可以得到图 3 显示的结果。

列表 5：在 SQL Server 2008 CTP 上的 AdventureWorks2008 数据库根据对象类型顺序对所有用户表、视图、内部表和服务队列排序。最后运行 DBCC UPDATEUSAGE，同时显示内部对象。

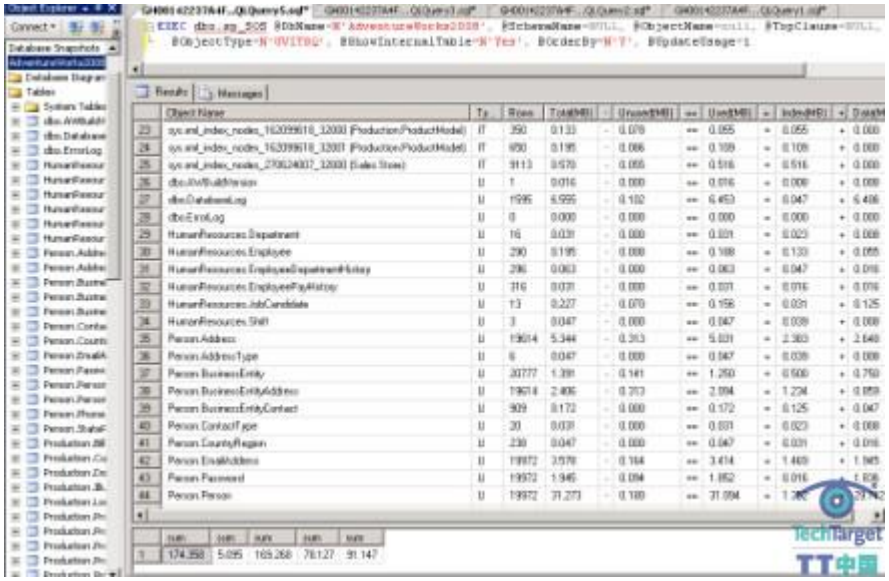


图 3：兼容 SQL Server 2008 CTP 的 sp_SOS 在 AdventureWorks2008 数据库运行后显示根据对象类型排序的所有对象列表。

我不能一一解释 sp_SOS 执行的所有不同的参数设置。如果你发现一些有趣的或与预期行为有冲突的结果时，请给我发一个评论，我将会检查一下看是否能再改进。

你可以阅读我接下来的关于存储过程 sp_SDS 文章。sp_SDS 不仅仅能确定“SQL 数据空间”，它也可以用于监控数据的增长，并在数据或日志文件增长时提醒 DBA，执行一个

日志备份事务，甚至提供文件级的详细分析，这样 DBA 就可以压缩文件以获取更多空余空间。

（作者：Richard Ding 译者：曾少宁／陈柳 来源：TT 中国）

正确使用 SQL Server 的 datetime 函数：GETDATE、DATENAME 和 DATEPART（一）

Transact- SQL 包含一组函数可以用于检索当前的日期和时间或一个 DATETIME 或 SMALLDATETIME 值的某个部分。比如，你可以在 DATETIME 值中 提取日、月或年以及季度、周、小时甚至毫秒。在本文中，我将对这些函数进行阐述并举例说明如何使用 SQL Server 中的这些函数来查询数据的。注意，本文假定你已经具备了一定的 T-SQL、DATETIME 和 SMALLDATETIME 数据类型的知识。想 获得更多这些类型的信息，你可以阅读这一系列的第一部分“SQL Server 2005 的 DATETIME 和 SMALLDATETIME 基础”。

检索当前日期和时间

在 T-SQL 中最便捷的一个函数就是 GETDATE，它根据本地系统的时钟设置检索当前日期和时间。使用 GETDATE，只需直接调用 T-SQL 语句中的函数而不用指定任何参数，如下例所示：

```
SELECT GETDATE() AS [Current Date/Time]
```

在这里，我在 SELECT 中使用 GETDATE 来检索日期/时间值。（注意，即使你不需要任何参数，你必须使用括号。）语句返回的结果如下：

Current Date/Time
2008-07-29 10:45:13.327

默认情况下，GETDATE 函数返回如下格式的时间值。然而，你可以使用 CONVERT 函数修改结果的格式。关于 CONVERT 的使用，可以参考这一技巧系列“从 date/time 值到字符类型的数据转换”。

Transact- SQL 的另外一个简单的函数是 GETUTCDATE，它用于检索当前的 Coordinated Universal Time (UTC)——也就是格林威治标准时间。检索的值是基于本地系统上的时钟和时区设置的。正如你在 GETDATE 中所看到的，你在 Transact-SQL 语句中调用 GETUTCDATE 是不需要包括任何参数的，如下例所示：

```
SELECT GETUTCDATE() AS [UTC Date/Time]
```

你运行这个语句时，你会得到如下的结果：

UTC Date/Time
2008-07-29 17:45:13.327

注意，这里返回的时间比前面显示的例子要晚 7 个小时。但我在一个配置为太平洋时区（在夏时制的白天）的系统上同时运行这些语句。

正如你在最后两个例子中所看到的，这些函数都是包含在 SELECT 列表中的。但是，当用它们来定义你定义表的默认值时，这些函数是特别有用的。比如，下面的 三个语句所创建的 Orders 表格——包括一个 DATETIME 字段（OrderDate）——在表格中插入数据并检索该数据：

```
CREATE TABLE Orders
(
OrderID INT PRIMARY KEY IDENTITY,
```

```
Product VARCHAR(30) NOT NULL,
OrderAmt INT NOT NULL,
OrderDate DATETIME NOT NULL DEFAULT GETDATE()
)
GO
INSERT INTO Orders (Product, OrderAmt)
VALUES('Test Product', 12)
GO
SELECT * FROM Orders
```

OrderDate 字段定义包含一个指定 GETDATE 作为默认值的 DEFAULT 子句。因此，当你在表格中插入行时，当前日期和时间将自动插入列中，下面显示了 SELECT 语句返回的结果：

OrderID	Product	OrderAmt	OrderDate
1	Test Product	12	2008-07-29 10:46:47.420

你可以将这些信息作为一个时间戳来使用，以便在需要时跟踪加入的记录或协助数据审计。这也方便其它使用时间戳检索数据的操作。比如，当决定是否提取或更新数据时，抽取、转换和加载（ETL）程序可能参考时间戳。

检索年、月或日

在某些情况下，你可能想要在 DATETIME 或 SMALLDATETIME 值中检索年、月或日。其中一个函数是使用 YEAR、MONTH 或 DAY 函数来检索必要的数据（作为一个整数）。下面的 SELECT 语句就是一个很好的说明：

```
SELECT YEAR(PostTime) AS [Year],
MONTH(PostTime) AS [Month],
DAY(PostTime) AS [Day]
FROM DatabaseLog
WHERE DatabaseLogID = 1
```

SELECT 子句中包含了三个字段表达式。第一个使用的是 YEAR 函数来检索 DatabaseLog 表格（AdventureWorks 中的样本数据库）中的 PostTime 字段的年。当调用 YEAR 函数时，指定字段的名称（或其它表达式）作为函数的参数。MONTH 和 DAY 函数也是一样的运行方式。在 SELECT 子句中的第二个字段表达式使用 DAY 来检索日。下面的结果显示了语句返回的信息类型：

每一个值都是在 PostTime 字段中提取并作为一个整数返回的。（存储在表中的值是 2005-10-14 01:58:27.567。）

这些函数都是检索年、月或日的简单方式，但是，在某些情况下，你可能想更多的控制返回的值的类型以及这些值的格式。另外，你可能想从日期/时间值中提取时间。幸运的是，Transact-SQL 支持这些函数。

（作者：Robert Sheldon 译者：曾少宁/陈柳 来源：TT 中国）

正确使用 SQL Server 的 datetime 函数：GETDATE、DATENAME 和 DATEPART（二）

检索 date/time 值各部分

与 YEAR、MONTH 和 DAY 函数相似，DATEPART 函数返回一个代表 date/time 值的指定部分的整数值。比如，下面的 SELECT 语句返回与上面的例子一样的结果：

```
SELECT DATEPART(yy,  PostTime) AS [Year],
DATEPART(mm,  PostTime) AS [Month],
DATEPART(dd,  PostTime) AS [Day]
FROM DatabaseLog
WHERE DatabaseLogID = 1
```

首先要注意的是，当调用 DATEPART 时，需要指定两个参数。第一个参数确认检索的 date/time 值的组成部分，而第二个参数是一个源字段。对于第一 个参数，你必须使用一个支持的缩写来指定时间部分。下面的表格列举了你可以检索的日期/时间部分以及你必须用来检索这些部分的缩写：

Date/time part	Abbreviations
year	yy, yyyy
quarter	qq, q
month	mm, m
day of year	dy, y

day	dd， d
week	wk， ww
weekday	dw
hour	hh
minute	mi， n
second	ss， s
millisecond	ms

有些 datetime 部分有多个缩写支持。比如，你可以使用“YY”或“YYY”作为第一个 DATEPART 参数来检索 date/time 值中的年。注意， 表格不仅仅包含年、月或日的缩写。换言之，你还可以检索季度、一年中特定的一天、一年中特定的一周以及工作日，如下面的 SELECT 语句所示：

```
SELECT DATEPART(qq， PostTime) AS [Quarter]，
DATEPART(dy， PostTime) AS [DayOfYear]，
DATEPART(wk， PostTime) AS [Week]，
DATEPART(dw， PostTime) AS [Weekday]
FROM DatabaseLog
WHERE DatabaseLogID = 1
```

正如前面的例子所显示的，每一个 DATEPART 实例都包含两个参数：date/time 部分的缩写和源字段。语句返回如下的结果：

Quarter	DayOfYear	Week	Weekday
4	287	42	6

注意，工作日显示为 6。默认情况下，SQL Server 的每周是从星期天开始的，因此，工作日 6 等同于星期五。

上面的两个例子仅仅检索与日期相关的值。但是，正如下面的表格所显示的，你也可以检索其它与时间相关的数据：

```
SELECT DATEPART(hh,  PostTime) AS [Hour],
DATEPART(mi,  PostTime) AS [Minute],
DATEPART(ss,  PostTime) AS [Second],
DATEPART(ms,  PostTime) AS [Millisecond]
FROM DatabaseLog
WHERE DatabaseLogID = 1
```

在这种情况下，该语句是检索小时、分、秒和毫秒，结果显示如下：

DATEPART 函数的主要的局限是它返回的只是整数，这就是为什么星期五是以 6 显示的。然而，如果你想要显示实际的日和月名称，你可以使用 DATENAME 函数。 DATENAME 函数与 DATEPART 函数的运作完全一样。DATENAME 使用同样数目的参数并支持同样的缩写。比如，如果你检索年、月和日，那么你 只需简单地用 DATENAME 取代 DATEPART 就可以得到正如你在前面的例子中所看到的结果：

```
SELECT DATENAME(yy,  PostTime) AS [Year],
DATENAME(mm,  PostTime) AS [Month],
DATENAME(dd,  PostTime) AS [Day]
FROM DatabaseLog
WHERE DatabaseLogID = 1
```

这样，你将得到下面的结果：

Year	Month	Day
2005	October	14

月的值现在是 October 而不是 10。但是，由于只有一种方式来代表它们，因此，年和日依然是整数。你也可以在其它的日期/时间组件上使用 DATENAME 函数，如下所示：

```
SELECT DATENAME(qq,  PostTime) AS [Quarter],
DATENAME(dy,  PostTime) AS [DayOfYear],
DATENAME(wk,  PostTime) AS [Week],
DATENAME(dw,  PostTime) AS [Weekday]
FROM DatabaseLog
WHERE DatabaseLogID = 1
```

再次，我用 DATENAME 取代 DATEPART，但是保留其它的不改变。语句返回了以下的结果。

Quarter	DayOfYear	Week	Weekday
4	287	42	Friday

注意，季度、一年中第几天和第几周仍然是整数，但是工作日现在是 Friday 而不是 6。你也可以使用 DATENAME 来检索一个 date/time 值的时间组件，但是结果将如你所想的那样，仍然是整数。

这就是 DATEPART 和 DATENAME 函数以及其它用来检索日期/时间值的函数。你可以单独或者混合着使用这些函数来关联值。我建议你尝试着使用这些函 数以便更好地了解它们是如何工作的。在以后的技巧中（第四部分），我将阐述如何在这些值上执行运算以便

添加数据和确定日期范围。同时，你也可以在 Microsoft SQL Server Books Online 上面获得各个函数的更多的信息和例子。

(作者：Robert Sheldon 译者：曾少宁/陈柳 来源：TT 中国)

在 SQL Server 2005 中使用 XQuery 检索 XML 数据（一）

当 Microsoft 发布 SQL Server 2005 时，它引进了一个新的数据类型：XML。与其它的 SQL Server 数据类型一样，你可以使用 XML 数据类型来定义字段、存储过程和用户自定义方法的变量和参数，并且你可以整体存取 XML 数据——即作为一个值 ——就像一个 XML 文档一样。然而，与 XML 文档一样，可能有时候当你想存取的只是 XML 数据中的指定值时，你就需要使用到 XQuery 了。

XQuery 是一个功能强大的、专门为存取 XML 数据而设计的脚本语言。SQL Server2005 支持一个允许在 XML 列、变量或参数中存取值的 XQuery 语言子集。通过调用 XML 数据类型支持的方法，就可以在 Transact-SQL 语句中使用 XQuery。其中两种方法分别是 value()和 query()——这对于在 XML 数据中检索指定的元素是特别有用 的。

在本文中，我将阐述这两种方法并分别举例说明。注意，本文假定你已经熟悉 T-SQL 和 SQL Server 的 XML。

XML value()方法

当在 Transact-SQL 语句中调用 XML 方法时，需要指定一个 XML 字段、变量或参数名称，并且在其后面括号中包含一个周期、方法名称以及一个 XQuery 表达式。比如，在一个命名为 XmlInfo 的 XML 字段上调用一个 value()方法，你可以使用下面的语法：

```
XmlInfo.value(<XQuery expression>)
```

当你调用方法 value()时，该 XML 方法返回一个指定类型的标量（单一的）值。这个 value()方法有两个参数。第一个是检索的元素和值，而第二 个是返回值的数据类型。让

我们举一个例子来说明它是如何工作的。下面的语句在 HumanResources 表中的 JobCandidateID 和 Resume 字段中检索数据。JobCandidate 表格（AdventureWorks 示例数据库的一部分）：

```
SELECT JobCandidateID,  Resume.value('declare namespace ns=
"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume";
(/ns:Resume/ns:Name/ns:Name.Last)[1]',
'nvarchar(30)') AS LastName
FROM HumanResources.JobCandidate
```

SELECT 列表中的第二字段（Resume）是以 XML 数据类型定义的。因此，你可以调用任何 XML 方法。在这里，我调用了 value()方法并且传输了两个参数。每个参数都包含在单引号中并且由一个逗号分隔。现在让我们仔细地看一下第一个参数。

第一个参数被分成两个部分并由分号分隔。第一部分表示的是命名空间并赋予“ns”别名。你必须任意类型化的 XML 字段指定命名空间。每一个类型化的字段 都关联到一个指定的模式。在 SQL Server Management Studio 中，你可以通过检索 XML 列中整体值，然后点击返回值来确定与 XML 文档相关的模式。XML 文档打开的是一个单独的窗口，如图 1 所示。注意， 该模式被定义为根元素的属性。

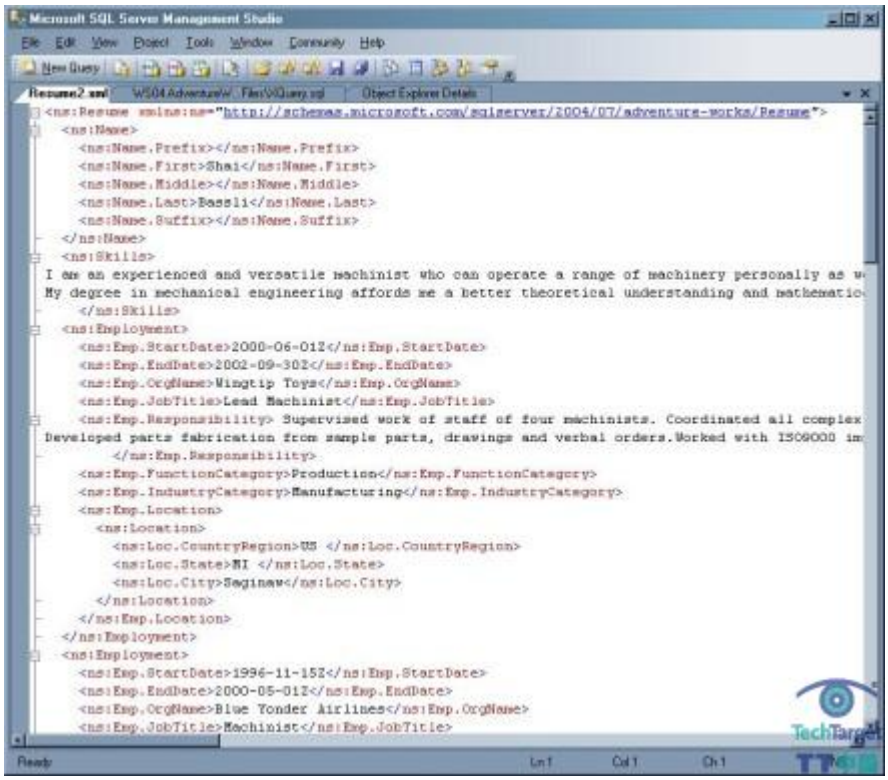


图 1：确定与 XML 文档关联的模式

当调用 value()方法时，在 XML 文档中列出的模式就是作为命名空间的模式。如果 XML 字段是非类型化的（未关联到一个模式），那么就不需要声明命名空间，并且可以省略参数的第一部分。

参数的第二部分定义包含检索的值的 XML 元素。

元素基本上是一个由正斜线分隔的路径名称。当处理类型化的 XML 字段时，你必须先在路径名中的每个节点前加上命令空间别名，然后是冒号。在这里，我在每个节点前加上“ns:”。再次参照图 1，你可以在文档的上方附近看到这个元素。

注意，路径名被包含在括号中并且其后面是[1]。因为 value()方法只能返回多个数量值，因此你必须在括号的后面指定[1]以确保只有一个元素实例 返回，即使在 XML 中只有

一个实例。在你指定了路径之后，你必须指定数据类型。SQL 语句现在就检索每个职位候选人的姓氏并将它作为 NVARCHAR 数据 类型返回。

与其它的语言一样，XQuery 支持广泛的各种不同方法。比如，下面的语句使用的是一个 concat 方法（在第一个参数的第二个部分）来连接名和姓：

```
SELECT JobCandidateID,  Resume.value('declare namespace ns=
"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume";
concat((/ns:Resume/ns:Name/ns:Name.First)[1],  " ",
(/ns:Resume/ns:Name/ns:Name.Last)[1])',
'nvarchar(60)') AS FullName
FROM HumanResources.JobCandidate
```

concat 方法的多个参数是由逗号分隔的。每一个参数都必须串连。注意，第一个和第三个参数使用的是上例中一样的路径架构。更多关于 concat 方法 和所有 XQuery 方法的信息，可以阅读 Microsoft SQL Server 2005 Books Online。

(作者：Robert Sheldon 译者：曾少宁／陈柳 来源：TT 中国)

在 SQL Server 2005 中使用 XQuery 检索 XML 数据（二）

XML 的 query()方法

虽然 value()方法可以很便捷地在 XML 字段中检索一个值。然而，当经常检索多个值时，你必须使用 XML 的 query()方法。query()方法只需要一个参数并返回一个指定的 XML 元素。比如，在下面的 SQL 语句中返回一个 Education 元素以及每个 职位候选人的的子节点。

```
    SELECT JobCandidateID,  Resume.value('declare namespace ns=
"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume";
concat((/ns:Resume/ns:Name/ns:Name.First)[1],  " ",
(/ns:Resume/ns:Name/ns:Name.Last)[1])',
'nvarchar(60)') AS FullName,
Resume.query('declare namespace ns=
"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume";
/ns:Resume/ns:Education') AS Education
FROM HumanResources.JobCandidate
```

query()方法的参数是包含在单一的引号中并分成两个部分。同样，当在一个类型化的字段中检索数据时，你必须指定一个命名空间。命名空间的声明方式 与 value()方法中的一样。在声明了命名空间之后，便可以指定检索的元素。在这种情况下，如图 2 所示，将返回 Education 元素以及所有的子节 点。

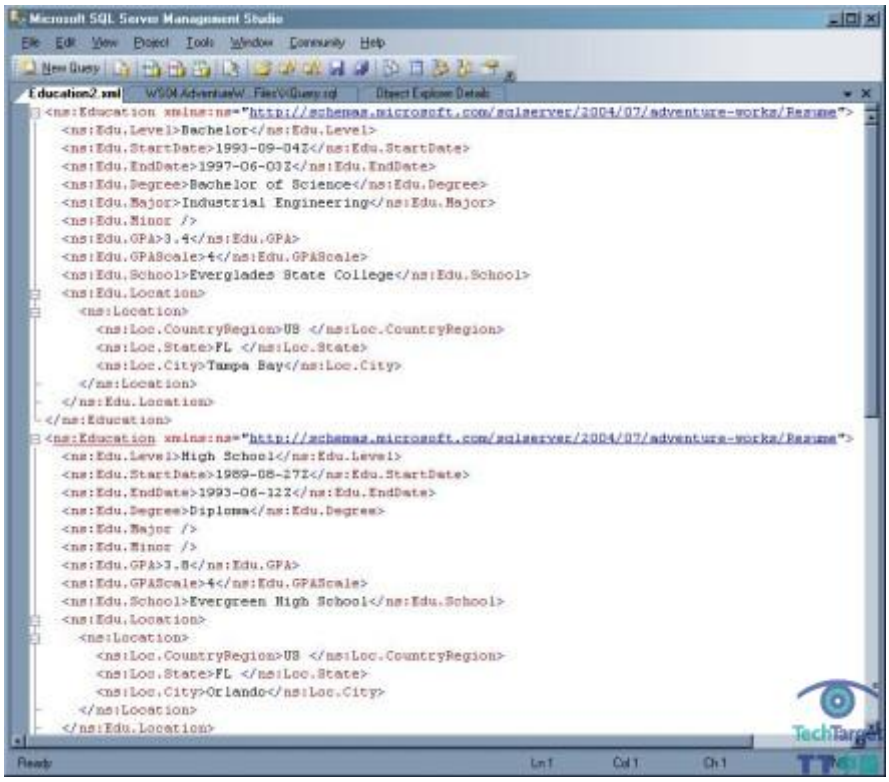


图 2：一旦声明了命名空间，便可以指定检索的 XML 元素。

通过将第二部分的参数转换为 FLWOR（发音为 flower）表达式，我们也可以获得同样的结果。按照定义，该表达式是由“FOR、LET、WHERE、ORDER BY 和 RETURN”子句组成的。但是，注意，目前 SQL Server 并不支持 LET 子句。

下面的 XML query()方法使用的是“FOR”和“RETURN”子句来检索 Education 元素以及子节点：

```
SELECT JobCandidateID,  Resume.value('declare namespace ns=
"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume";
concat((/ns:Resume/ns:Name/ns:Name.First)[1],  " ",
(/ns:Resume/ns:Name/ns:Name.Last)[1])',
```

```
'nvarchar(60)') AS FullName,
Resume.query('declare namespace ns=
"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume";
for $ed in /ns:Resume/ns:Education
return $ed')AS Education
FROM HumanResources.JobCandidate
```

正如你所看到的，FOR 子句包含$ed 变量。你并不需要显式地声明这个变量。在 FOR 子句中使用就可以了。这个变量是用于遍历 Education 元素 的。（只要遵循 SQL Server 的命名规范，变量可以使用任何名称。）而 RETURN 子句引用$ed 变量。因此，它返回每一个 Education 元素和子节点。

当然，一般不会只是使用一个 FLWER 表达式来检索一个参数和子节点。但是，你可以在 FOR 子句（在括号中）的路径名称上包含一个表达式来限制返回的结果。比如，下面的语句限制了结果中 Edu.Level 值为“Bachelor”的 Education 元素：

```
SELECT JobCandidateID,  Resume.value('declare namespace ns=
"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume";
concat((/ns:Resume/ns:Name/ns:Name.First)[1],  " ",
(/ns:Resume/ns:Name/ns:Name.Last)[1])',
'nvarchar(60)') AS FullName,
Resume.query('declare namespace ns=
"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume";
for $ed in /ns:Resume/ns:Education[ns:Edu.Level="Bachelor"]
return $ed') AS Education
FROM HumanResources.JobCandidate
```

注意，我使用的是相等的比较操作符（=）来比较 Edu.Level 元素值和字符串值。这样就构成一个 Boolean 表达式，当结果中包含该元素时为 true。XQuery 支持各种创建 Boolean 表达式的操作符号。你可以在 Microsoft SQL Server 2005 Books online 上阅读支持的操作符。

（作者：Robert Sheldon 译者：曾少宁／陈柳 来源：TT 中国）

在 SQL Server 2005 中使用 XQuery 检索 XML 数据（三）

除了在 FOR 子句中使用 Boolean 表达式，还可以在 WHERE 子句中定义相同的逻辑，如下例所示：

```
SELECT JobCandidateID,  Resume.value('declare namespace ns=
"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume";
concat((/ns:Resume/ns:Name/ns:Name.First)[1],  " ",
(/ns:Resume/ns:Name/ns:Name.Last)[1])',
'nvarchar(60)') AS FullName,
Resume.query('declare namespace ns=
"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume";
for $ed in /ns:Resume/ns:Education
where $ed/ns:Edu.Level="Bachelor"
return $ed') AS Education
FROM HumanResources.JobCandidate
```

在这种情况下，FOR 子句中仅仅包含一个简单的路径，而 WHERE 子句中包含一个 Boolean 逻辑。注意，路径名称使用了$ed 变量来指出了 FOR 子句循环访问的正确的 Education 元素。

在 WHERE 子句中使用这个逻辑可以更易于阅读和编写代码，特别是在使用复杂的 Boolean 表达式的时候。比如，下面的语句使用了 WHERE 子句中的“AND”逻辑操作符号，结果仅限于学士学位中主修学科为商务的记录。

```
SELECT JobCandidateID,  Resume.value('declare namespace ns=
"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume";
concat((/ns:Resume/ns:Name/ns:Name.First)[1],  " ",
(/ns:Resume/ns:Name/ns:Name.Last)[1])',
'nvarchar(60)') AS FullName,
Resume.query(('declare namespace ns=
"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume";
for $ed in /ns:Resume/ns:Education
where $ed/ns:Edu.Level="Bachelor" and $ed/ns:Edu.Major="Business"
return $ed') AS Education
FROM HumanResources.JobCandidate
```

两个路径结果由“AND”操作符号连接。因此，只有当这两个条件都必须为真时才返回 Education 元素。

让我们来看一个包含 ORDER BY 子句的例子。下面的语句检索的是与技术和产品相关的就业信息：

```
SELECT JobCandidateID,  Resume.value('declare namespace ns=
"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume";
concat((/ns:Resume/ns:Name/ns:Name.First)[1],  " ",
(/ns:Resume/ns:Name/ns:Name.Last)[1])',
'nvarchar(60)') AS FullName,
Resume.query('declare namespace ns=
"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume";
for $emp in /ns:Resume/ns:Employment
```

```
where $emp/ns:Emp.FunctionCategory = "Production" or
$emp/ns:Emp.FunctionCategory = "Technology"
order by $emp/ns:Emp.EndDate descending
return $emp') AS Employment
FROM HumanResources.JobCandidate
```

在上面的例子中，语句使用 FOR 子句来确定所需的元素（如，Employment）和 WHERE 子句来限制结果。但是，注意，语句也包含了 ORDER BY 子句。你可以使用这个子句来基于指定节点的排序结果。因此，我基于 Emp.EndDate 元素以降序的形式排序结果。这样，最近的就业信息将出现在列 表的最前面。

正如你所看到的，当检索 XML 数据 时，value()和 query()方法是非常有用的。然而，我这里所涉及到的都只是些皮毛的知识。XQuery 是一个强大的语言，它可以用来编写复杂查 询并明确地按照你需要的方式来返回 XML 数据。由于 XQuery 支持大量各式各样的方法，操作符号和表达式，因此，你可以检索你的 XML 字段、变量和参数 中存储的任意元素和属性——以任意需要的格式。同样的，你可以参考 SQL Server 2005 Books Online 上其它关于 XQuery 的信息以及更多的示例。

（作者：Robert Sheldon 译者：曾少宁／陈柳 来源：TT 中国）

SQL Server 2000 中监控长期运行任务的存储过程

问题：在 SQL Server 2000 中监控长期运行的任务是一项复杂的任务。问题在于只有在第一项任务完成之后，才会写入执行历史（日志）。因此，如果第一步（ 或者只有一步）的任务运行了很长的时间，那么将无法查询任何系统表或者方法来得到它的开始时间。

在本文中，我们将探讨用 T-SQL 修复来检索关于长期运行任务的信息并解决这个问题。

快速标识长期运行 SQL Server 任务

一个可以轻松解决这个问题的方法是在任务中添加虚拟的第一步。虚拟的这一步必须完成一项简单而快速的任务，如 SELECT GETDATE（）。在添加这个步骤之后，当下一步的任务执行时，sysjobhistory 表就会被监控。Sysjobhistory 表中第一步骤的 run_date 和 run_time 域将显示这一步完成的日期和时间，因此，它也将显示下一步开始的日期和时间。

这个例子是在第二个步骤开始两分钟之后显示的运行任务列表：

```
SELECT DISTINCT  name
FROM   msdb.dbo.sysjobs j
WHERE  j.job_id in
       (SELECT h.job_id
        FROM   msdb.dbo.sysjobhistory h
        WHERE  j.job_id = h.job_id
          AND  h.run_status in (4)
        -- run_duration is Elapsed time in the execution
        -- of the job or step in HHMMSS format
        -- (200 = 2 minutes):
          AND run_duration > 200
          AND h.job_id in (SELECT TOP 1 h2.job_id
                           FROM  msdb.dbo.sysjobhistory h2
                           WHERE h.job_id = h2.job_id
                           ORDER BY run_date, run_time desc
                          )
        )

-- Job statuses in sysjobhistory:
-- 0 = Failed
-- 1 = Succeeded
-- 2 = Retry
-- 3 = Canceled
-- 4 = In progress
```

TechTarget
TT中国

怎样在复杂的环境下监控长期运行的 SQL Server 任务？

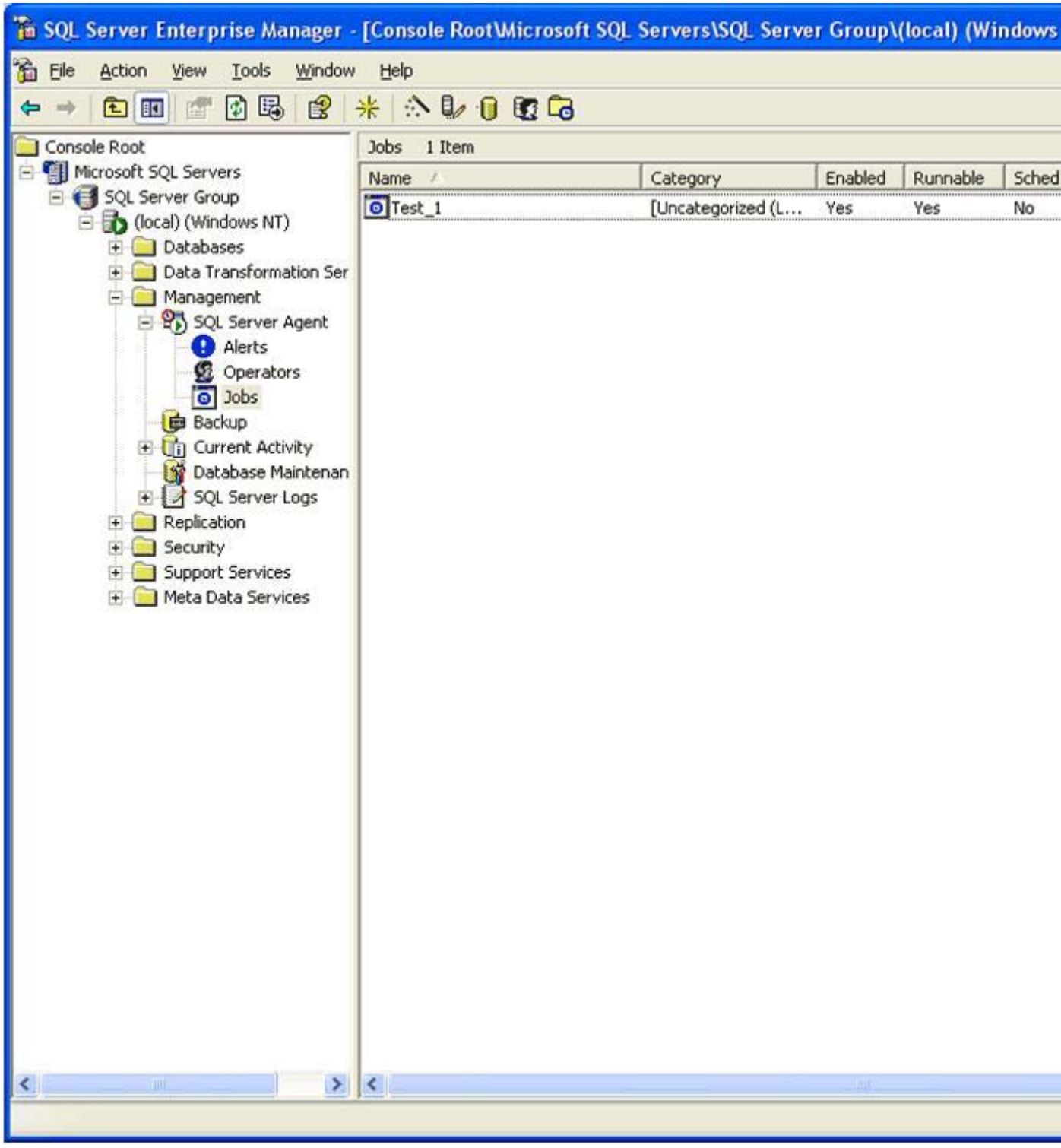
如果在许多服务器上运行上百个 SQL 任务，该怎么办呢？如果任务是由外部供应商建立的，你无法在任务之前增加步骤，又该怎么办？

添加另外一个步骤到所有现有的任务中可能会相当的耗时或者不被允许。

让我们来讨论另一个检测长期运行 SQL 任务的方法。

在此处，我的目标是在一个位置上集中检查长期运行的任务，收集相关的信息并存入中央表中。注意：在我的环境下的长期运行 SQL Server 2000 任务运行的时间是超过 4 到 4.5 个小时。

在 SQL Server 2000 Enterprise Manager 中，任务状态显示在任务列表中：


SQL Server Enterprise Manager - [Console Root\Microsoft SQL Servers\SQL Server Group\(local) (Windows
File Action View Tools Window Help
Console Root
Microsoft SQL Servers
SQL Server Group
(local) (Windows NT)
Databases
Data Transformation Ser
Management
SQL Server Agent
Alerts
Operators
Jobs
Backup
Current Activity
Database Maintenan
SQL Server Logs
Replication
Security
Support Services
Meta Data Services
Jobs 1 Item
Name
Category
Enabled
Runnable
Sched
Test_1
[Uncategorized (L...
Yes
Yes
No

如果在任务列表面板打开的时候，你运行 SQL Profiler 来监控运行的情况时，你将得到以下结果：

```
exec msdb..sp_help_job
```

由于我要将任务结果存储在中央表中，因此，我不能使用 sp_help_job 存储过程。

如果你执行 INSERT… EXEC msdb..sp_help_job 这个命令，将输出结果保存在表中，那么由于存储过程 sp_help_job 调用了内部存储过程，因此这个命令将失败。

（作者：Michelle Gutzait　译者：曾少宁／陈柳 来源：TT 中国）

在 SQL Server 2005 中创建 DDL 表格来审计 DDL 触发活动（一）

在 SQL Server 2005 之前，我们只能定义 Data Manipulation Language (DML)触发器。当执行 DML 语句时，如 UPDATE 或 DALETE，这些触发器就会启动。在 SQL Server 2005 发布之后，现在可以定义 Data Definition Language (DDL)触发器了。当执行 DDL 语句，如 CREATE TABLE 和 ALTER VIEW，这些类型的触发器就会启动，并且这使得使用 DDL 触发器来审计 SQL Server 中的 DDL 事件更加容易了。

其中一个可以用来审计 DDL 事件的方法是先创建一个表来存储相关的事件数据，然后创建一个 DDL 触发器来记录事件。在本文中，我将阐述每个步骤并举例说 明每个概念。对于所举的例子，我将在传输 SQL Server 2005 的 AdventureWorks 示例数据库中创建一个审计表和 DDL 触发器。注意，本文的前提条件就是假定你已经非常熟悉 Transact- SQL 和 DDL 概念。

创建 DDL 审计表

审计表存储每次指定类型的 DDL 事件发生时产生的事件相关信息。比如，如果从数据库中删除一个视图，那么就会产生一个 DROP_VIEW 事件。我们可以使用 DDL 触发器来捕捉事件信息并存储在表中。

每个审计表都必须至少包含一个 XML 字段来存储事件相关数据。后面将介绍 SQL Server 是如何生成 XML 格式的数据的。当然，这个表也必须包含一个主键字段。下面的语句在 AdventureWorks 数据库中创建了一个基本的审计表：

```
CREATE TABLE dbo.EventLog
(EventID INT PRIMARY KEY IDENTITY,
```

```
EventInstance XML NOT NULL)
GO
```

注意，我已经将 EventID 列作为主键，以及 EventInstance，并介绍了字段是如何保存与每个事件相关的 XML 数据的。每次 DDL 事件生成时，就会有一行添加到表格中。然后，我们可以检索 EventInstance 字段的内容来查看指定事件的信息。

创建 DDL 触发器

在创建了审计表之后，我们必须定义 DDL 触发器。下面的 CREATE TRIGGER 语句定义了一个触发器，当每次在 AdventureWorks 数据库中发生 DDL 事件时，它将在 EventLog 表中插入事件相关的数据：

```
CREATE TRIGGER LogEvents
ON DATABASE
AFTER DDL_DATABASE_LEVEL_EVENTS
AS
INSERT INTO dbo.EventLog (EventInstance)
VALUES (EVENTDATA())
```

让我们详细地来看看每一行的代码以便更好的理解它。CREATE TRIGGER 子句只是简单地标识新的触发器的名称，也就是 LogEvents。第二行——ON DATEBASE——表示触发器将在数据库级上创建。当然，也可以在服务器上创建触发器，它会在服务器发生 DDL 事件时启动，这时就应该使用 ON ALL SERVER 选项。然而，对于这个例子，我们关注的仅仅是与 AdventureWorks 数据库相关的 DDL 事件。

下一行代码——AFTER DDL_DATABASE_LEVEL_EVENTS——是一个 AFTER 子句。这个子句的第一部分是一个 AFTER 关键字，它表示只有在相关的操作（特 别是子句的第二部

分）已经成功的执行了才启动触发器。如果不用 AFTER，可以用 FOR 关键字，这就意味着一发生 DDL 事件就启动事件。因此，我更喜欢在 它们成功的运行了之后才记录这些操作。

AFTER 子句的下一个 部分指定了事件类型或组。因此你必须规定哪件 DDL 事件能够引发触发器启动。因为我想要审计数据库级的所有 DDL 事件，所以我选择了 DDL_DATABASE_LEVEL_EVENTS 选项（一个事件组）。但是，你可以选择其它的组或某个事件类型。如果指定一个以上的事件组或类型，那 么需要使用逗号分隔选项。更多关于每个事件类型和组的详细信息，可以阅读 Microsoft SQL Server Books Online 上的“CREATE TRIGGER (Transact-SQL)”专栏。

当我指定了事件组之后，我使用了一个 AS 关键字，然后是 INSERT 语句。这个语句在每次触发器启动时插入一行数据到 EventLog 表中。我通过调用 EVENDATE（）系统方法获得了 EventInstance 字段的值，它可以检索事件相关的数据（以 XML 格式）。当事件发生时，该方法提供所需的值。

这就是设置基本的审计解决方法的所有代码。现在，让我们来验证一下结果。

(作者：Robert Sheldon 译者：曾少宁 来源：TT 中国)

在 SQL Server 2005 中创建 DDL 表格来审计 DDL 触发活动（二）

测试审计方法

最佳的测试方法就是运行一对针对 AdventureWorks 数据库的 DDL 语句。下面的语句创建了 Person、Contact2 表，接着删除了表：

```
SELECT FirstName,  LastName,  EmailAddress
INTO Person.Contact2
FROM Person.Contact
GO
DROP TABLE Person.Contact2
```

两个语句都应该在 AdventureWorks 上生成 DDL 事件，接着启动 LogEvents 触发器。为了验证触发器是否正确地记录了这两个事件，只需简单的运行下面的 SELECT 语句：

```
SELECT * FROM dbo.EventLog
```

语句必须返回两行。在每一行，EventInstance 字段都必须包含一个与具体事件相关的 XML。如果查看第一行的 XML，那么结果应该是这样的：

```
<EVENT_INSTANCE>
  <EventType>CREATE_TABLE</EventType>
  <PostTime>2008-03-25T14:10:20.377</PostTime>
  <SPID>52</SPID>
  <ServerName>WS04</ServerName>
  <LoginName>DOMAIN01\Administrator</LoginName>
  <UserName>dbo</UserName>
  <DatabaseName>AdventureWorks</DatabaseName>
  <SchemaName>Person</SchemaName>
  <ObjectName>Contact2</ObjectName>
  <ObjectType>TABLE</ObjectType>
  <TSQLCommand>
    <SetOptions ANSI_NULLS="ON" ANSI_NULL_DEFAULT="ON"
ANSI_PADDING="ON" QUOTED_IDENTIFIER="ON" ENCRYPTED="FALSE" />
    <CommandText>SELECT FirstName, LastName, EmailAddress
INTO Person.Contact2
FROM Person.Contact
</CommandText>
  </TSQLCommand>
</EVENT_INSTANCE>
```


注意，事件数据包含了事件类型、创建的对象以及用于创建表格的命令和其它的信息。如果只想从 EventInstance 字段中检索特定的信息，可以使用 XQuery 表达式来访问单个元素值。比如，下面的 SELECT 语句使用了 XML value()方法来检索事件类型、模式名和对象名：

```
SELECT EventID,
EventInstance.value('(//EventType)[1]',
'nvarchar(30)') AS EventType,
EventInstance.value('concat((//SchemaName)[1],
".",  (//ObjectName)[1])',  'nvarchar(60)')
AS ObjectName
FROM dbo.EventLog
```

正如你所看到的，我先通过指定 XML 数据类型定义的 EventInstance 字段调用了 value()方法。我可以以这种方法调用任意的 XML 方法。 value()方法包含两个参数。第一个定义了我所要检索的 XML 元素。第二个定义了数据类型。第一个参数必须包含在括号中，并且后面加上[1]，因为 value()方法总是返回标量值，。即使指定的元素是在 XML

（如本例子）中是唯一的也必须指定[1]。当运行这个 SELECT 语句时，会得到下面的结果：

EventID	EventType	ObjectName
1	CREATE_TABLE	Person.Contact2
2	DROP_TABLE	Person.Contact2

(2 row(s) affected)

结果显示的是 CREATE_TABLE and DROP_TABLE 事件类型，这正是我所运行的 DDL 语句所应该产生的结果。因为 SQL Server 现在支持 Data Definition Language (DDL)触发器，因此可以非常容易地审计由这些语句生成的事件。正如我所阐述的，一个简单的审计事件的方法就是在 SQL Server 上创建一个审计表，并定义一个触发器。然而，我们也可以使用 DDL 触发器来执行一些操作，而不是在审计表中插入一行。比如，当一个特别的 DDL 事件发生时，发送一封邮件到一个指定的接受者。当创建一个 DDL 触发器时，我们可以定义一个或多个 Transact-SQL 语句来完成一些特定的操 作，这样就可以通过创建触发器来执行各种不同的任务。

（作者：Robert Sheldon 译者：曾少宁/陈柳 来源：TT 中国）

使用 DATEADD 和 DATEDIFF 来计算 SQL Server 的 DATETIME 值（一）

在 SQL Server 数据库中，DATETIME 和 SMALLDATETIME 值是以整数存储的。然而，与整数不同的是，它们不能直接地进行数学运算。尽管如此， 有时候还是需要在日期/时间值中添加或减去一个时间间隔。比如，你可能想在一值上加一些月数或天数，或者甚至可能是小时数。你甚至可能想比较两个日期/时 间值以便确定它们之间的时间间隔，如相差的天数或年数。为了简化这些类型的计算，Transact-SQL 支持两个重要的日期/时间方法：DATEADD 和 DATEDIFF。

在关于 DATETIME 值这一系列文章的第四部分，我阐述了如何使用这两个方法并举例说明它们是如何工作的。为了演示这些方法，我使用了下面的 Transact-SQL 代码在 AdventureWorks 示例数据库中创建了一个 Sales.Orders 表：

```
USE AdventureWorks
GO
IF EXISTS (SELECT table_name
FROM information_schema.tables
WHERE table_schema = 'Sales'
AND table_name = 'Orders')
DROP TABLE Sales.Orders
GO
CREATE TABLE Sales.Orders
(
OrderID INT NOT NULL,
OrderDate DATETIME NOT NULL,
```

```
DelivDate DATETIME NOT NULL
)
GO
INSERT INTO Sales.Orders
VALUES(1001, GETDATE(), '2008-09-08 18:27:10.750')
```

表的定义包含了 OrderDate 和 DelivDate 字段，两者都是 DATETIME 数据类型。在我创建了表之后，我在表中插入了一行用于测试 DATEADD 和 DATEDIFF 方法的数据。

使用 DATEADD 方法

在一些情况下，你可能想添加一个时间间隔到 DATETIME 或 SMADDDATETIME 值中——或者减去一个时间间隔。比如，你可能需要在一个指定的日期中增加或减去一个月。你可以使用 DATEADD 方法来执行这个计算。这个方法运用了下面的语法：

```
DATEADD(<date/time_part>, <number>, <date>)
```

<date/time_part>占位符指的是日期/时间值中增加或减少的增量/余差（如日或月）。下表列出了可以使用的日期/时间部分，以及代表这些部分的缩写：

Date/time part	Abbreviations
year	yy, yyyy
quarter	qq, q
month	mm, m
day of year	dy, y
day	dd, d
week	wk, ww

weekday	dw
hour	hh
minute	mi， n
second	ss， s
millisecond	ms

比如，如果你想在一个日期/时间值中增加一小时，可以使用 hh 缩写。在某些情况下，日期/时间部分支持两个缩写，如周可以用 wk 或 ww 支持。

<number>占位符指的是所增加的数值（一个整数）。比如，如果在日期中增加 10 天，就是 10。但是，注意，如果是减去时间间隔，它必须是一个负整数。比如，从天数中减去 10，就必须是-10。

<date>占位符指的是增加或减少的指定间隔的日期/时间值。它可能是一个日期/时间格式的字符串值，或者是方法返回的一个日期/时间值，又或者是常见的 DATETIME 或 SMALLDATETIME 字段。

让我们举例来说明它是如何工作的。在下面的 SELECT 语句中，我增加三个月到 Sales.Orders 表中 OrderDate 值：

```
SELECT OrderDate， DATEADD(mm， 3， OrderDate) AS NewDate
FROM Sales.Orders
WHERE OrderID = 1001
```

注意，SELECT 列表使用了 DATEADD 方法。这个方法有三个参数：mm 指月，3 指月数，而 OrderDate 是一个 DATETIME 值。因此，当查询返回值时，每个 OrderDate 值都会增加三个月时间，如下的结果所示：

OrderDate	NewDate
2008-08-27 13:36:16.280	2008-11-27 13:36:16.280

如上所示，日期 August 27 已经被改为 November 27。而且，这样的运算还不仅限于日期。下面我在 OrderDate 值中增加三个小时：

```
SELECT OrderDate,  DATEADD(hh,  3,  OrderDate) AS NewTime
FROM Sales.Orders
WHERE OrderID = 1001
```

DATEADD 的第一个参数现在是 hh，而不是 mm，因此，只有小时被改变了，如下结果所示：

OrderDate	NewTime
2008-08-27 13:36:16.280	2008-08-27 16:36:16.280

日期/时间值也可以减去一定的日期或时间间隔。在下例中，我从 OrderDate 值中减去了三天：

```
SELECT OrderDate,  DATEADD(dd,  -3,  OrderDate) AS PastDate
FROM Sales.Orders
WHERE OrderID = 1001
```

注意，DATEADD 的第一个参数现在是 dd。同时，注意，第二个参数是一个负数，这意味着将有三天被减去，如下所示：

OrderDate	PastDate

2008-08-27 13:36:16.280	2008-08-24 13:36:16.280

这样，新的日期是 August 24 而不是 August 27。

这样，上面的例子演示如何在从数据库查询到日期/时间值后再对它进行修改。而 DATEADD 方法同样也可以用来插入日期/时间数据。因为 DATEADD 方法 返回一个 DATETIME 值。（如果所提供的日期对应的方法是 SMALLDATETIME，那么它将返回一个 SMALLDATETIME 值。）在下面的例 子中，我添加了一行数据到 Sales.Orders 表中，然后使用 SELECT 语句来检索这个行：

```
INSERT INTO Sales.Orders
VALUES(1002, GETDATE(), DATEADD(dd, 10, GETDATE()))
GO
SELECT * FROM Sales.Orders
WHERE OrderID = 1002
```

注意，VALUES 子句包含了表中每个字段的值。对于 OrderDate 值，我使用 GETDATE() 方法来获取当前的日期和时间。对于 DelivDate 字段，我使用 DATEADD 方法以及相应的三个参数。第一个参数 dd 表示将要添加到日期中的是天数。第二个参数 10 意味着将添加 10 天到日期中。最后，第 三个参数是 GETDATE 方法。因此，10 天将添加到目前的日期和时间中并插入到 DelivDate 字段。这就是 SELECT 语句生成的结果：

OrderID	OrderDate	DelivDate
1002	2008-08-27 13:40:22.357	2008-09-06 13:40:22.357

正如所期待的，DelivDate 值比 OrderDate 晚 10 天。

现在让我们来检测一个使用了 DATEADD 方法的 UPDATE 语句。在下面的语句中，我从 DelivDate 值中减去了三天，然后显示了结果：

```
UPDATE Sales.Orders
SET DelivDate = DATEADD(dd,  -3,  DelivDate)
WHERE OrderID = 1002
GO
SELECT * FROM Sales.Orders
WHERE OrderID = 1002
```

这次我在 SET 子句中使用了 DATEADD——我将 DelivDate 值设为 DATEADD 方法返回的结果。这个方法指定天数（dd）为第一个参数，-3 为 第二个参数，而 DelivDate 字段为第三个参数。这就意味着该方法将返回一个比原始日期早三天的日期，并将 DelivDate 设置为新的日期，如下结 果显示：

OrderID	OrderDate	DelivDate
1002	2008-08-27 13:40:22.357	2008-09-03 13:40:22.357

你应该记得，INSERT 语句（在前一个例子）添加了一个 DelivDate 值为 September 6 的行。但是，这个值现在是 September 3，比原来早了三天。

（作者：Robert Sheldon 译者：曾少宁/陈柳 来源：TT 中国）

使用 DATEADD 和 DATEDIFF 来计算 SQL Server 的 DATETIME 值（二）

使用 DATEDIFF 方法

DATEDIFF 方法可以计算两个日期之间的时间间隔，并返回一个代表间隔的整数。这个方法使用下面的语法：

```
DATEDIFF(<date/time_part>, <start_date>, <end_date>)
```

<date/time_part>占位符指的是两个日期中需要比较的部分。比如，你想确认开始日期和结束日期之间的小时数或天数。

除了工作日(dw, w)缩写之外，<date/time_part>占位符使用的缩写与 DATEADD 方法一样。DATEDIFF 不支持工作日比较。

<start_date>占位符指的是比较的开始日期，而<end_date>占位符指的是结束日期。换言之，方法将返回开始日期和结束日期之间的具体时间或日期间隔。

让我们举例来说明它是如何工作的。下面的 SELECT 语句计算了 Sales.Orders 表中 OrderDate 和 DelivDate 值之间的时间间隔：

```
SELECT OrderDate, DelivDate,
DATEDIFF(dd, OrderDate, DelivDate) AS DaysDiff
FROM Sales.Orders
WHERE OrderID = 1002
```

在 这个语句中，我在 SELECT 列表中使用了 DATEDIFF。这个方法的第一参数指定间隔必须是天数（dd），而第二个参数指定 OrderDate 作为开 始日期，然后第三个参数指定 DelivDate 作为结束日期。因此，DATEDIFF 将计算 OrderDate 和 DelivDate 之间的天数，在此例子 中，它是 7 天，如下结果所示：

OrderDate	DelivDate	DaysDiff
2008-08-27 13:40:22.357	2008-09-03 13:40:22.357	7

当然，它也可以用来计算各种时间间隔，如下例的语句所示：

```
SELECT OrderDate,  DelivDate,
DATEDIFF(hh,  OrderDate,  DelivDate) AS HoursDiff
FROM Sales.Orders
WHERE OrderID = 1002
```

在这种情况下，方法的第一个参数是小时（hh）而非天数。因此，方法将返回 OrderDate 和 DelivDate 值之间相差的小时数，如下结果所示：

OrderDate	DelivDate	HoursDiff
2008-08-27 13:40:22.357	2008-09-03 13:40:22.357	168

两个值之间相差 168 个小时。

与 DATEADD 方法一样，DATEDIFF 方法并不仅限于用在 SELECT 语句中。比如，DATEDIFF 可以用在 UPDATE 语句的 WHERE 子句中， 以确定哪一行需要更新。在下例中，我使用了 DATEDIFF 来指定这些在 OrderDate 和 DelivDate 值之间天数少于 8 的行。

```

```

```
UPDATE Sales.Orders
SET DelivDate = DATEADD(dd, 3, DelivDate)
WHERE DATEDIFF(dd, OrderDate, DelivDate) < 8
GO
SELECT OrderID, OrderDate, DelivDate,
DATEDIFF(dd, OrderDate, DelivDate) AS DaysDiff
FROM Sales.Orders
```

在 前面的例子中，DATEDIFF 方法返回了 OrderDate 和 DelivDate 值之间的天数。然后这个数目将与 8 作比较。如果天数少于 8，那么这一行 将被更新；否则，该行将不改变。对于这些需要更新的行，我使用 DATEADD 方法来增加三天到 DelivDate 值中。然后，我运行一个 SELECT 语句 来返回 Sales.Orders 表的数据以及计算每个行中两个日期的不同，如下结果所示：

OrderID	OrderDate	DelivDate	DaysDiff
1001	2008-08-27 13:36:16.280	2008-09-08 18:27:10.750	12
1002	2008-08-27 13:40:22.357	2008-09-06 13:40:22.357	10

结果显示现在在两个日期（第二行中）之间是相差 10 天，而非原来的 7 天。

在表定义中使用 DATEADD 和 DATEDIFF

DATEADD 和 DATEDIFF 方法也可以用在表定义中。例如，字段定义的 DEFAULT 子句中可以用 DATEADD 方法或使用 DATEDIFF 方法来创建一个计算得 来的字段。在下面的 Transact-SQL 代码中，我首先创建了使用 DATEADD 和 DATEDIFF 的表，然后添加一行数据到表中，最后检索表的数 据：

```
USE AdventureWorks
GO
IF EXISTS (SELECT table_name
FROM information_schema.tables
WHERE table_schema = 'Sales'
AND table_name = 'Orders')
DROP TABLE Sales.Orders
GO
CREATE TABLE Sales.Orders
(
OrderID INT NOT NULL,
OrderDate DATETIME NOT NULL DEFAULT GETDATE(),
DelivDate DATETIME NOT NULL DEFAULT DATEADD(dd, 10, GETDATE()),
DaysDiff AS DATEDIFF(dd, OrderDate, DelivDate)
)
GO
INSERT INTO Sales.Orders(OrderID)
VALUES(1001)
GO
SELECT OrderID, OrderDate, DelivDate, DaysDiff
FROM Sales.Orders
```

在 CREATE TABLE 语句中，我创建了四个字段，其中三个存储日期/时间数据。OrderDate 字段直接使用 GETDATE 来生成默认值。DelivDate 字段 也有一个默认值。然而，这个默认值是基于 DATEADD 返回的结果的，同时，在这种情况下，我使用方法增加 10 天到 GETDATE 返回的值存储到 DelivDate 字段中。最后，DaysDiff 字段是一个计算得来的字段，它的值是使用 DATADIFF 来计算 OrderDate 和 DelivDate 值之间的天数差。

在表定义之后，我插入一数据行到表中。因为所有的日期/时间值都是自动生成的，因此我仅仅需要插入 OrderID 值，如下所示：

OrderID	OrderDate	DelivDate	DaysDiff
1001	2008-08-27 13:42:50.433	2008-09-06 13:42:50.433	10

DATEADD 和 DATEDIFF 方法不仅仅在表定义中非常有用，同样也适用于查询和数据修改语句。通过 DATEADD，我们可以将日期/时间值增加和减少一定值，而通 过 DATEDIFF，我们可以计算日期/时间值之间的时间间隔。更多详细的关于这些方法的信息，可以阅读 Microsoft SQL Server Books Online。

（作者：Robert Sheldon 译者：曾少宁/陈柳　来源：TT 中国）

配置 SQL Server 服务代理来发送存储过程数据（一）

在 SQL Server 2005 中，Microsoft 引进了一个令人振奋的新特性即服务代理（Service Broker），同时这也给许多数据库管理人员带来了一个新的概念。这个概念，进程外消息，是一个开发人员在一些产品中使用多年的应用，如 Microsoft Message Queue (MSMQ)。实现这个功能，需要运行一个 SQL Server 命令，并且对该命令的数据进行处理。但是，因为用户不需要命令的输出，因此它不需要立刻进行处理。这里就是消息队列发挥作用地方。

正常情况下，当用户在 SQL Server 中启动存储过程时，他们必须等待存储过程完成数据处理，然后再进行下一个动作。通过使用服务代理，他们可以让实际数据处理排队等候一下。在这 种情况下，用户将运行存储过程而非数据处理。我们将输入数据打包作为一个消息，然后将消息发到队列去。我们配置队列自动读取信息，并处理信息中的数据。

消息可以在同一个数据库中发送，可以在一个服务器上的一个数据库发送到另一个数据库，或在服务器之间包括因特网上的服务器之间发送。这个过程就如同邮件一 样。当我给我的朋友发一封邮件，他接收到邮件之后可以进行读取（信息处理），接着可以就根据内容做点什么。服务代理就是以这样的方式运作的。

设置服务代理发送和接收消息的四个步骤：

在实际发送和接收消息之前，服务代理有几个组件需要配置。比如，我们将消息输入一个队列来作自动处理。这些消息的处理可以是自动的，或者是某个 Windows 应用所需要的，或者队列中读取消息的服务。

与其它对象一样，服务代理名称在数据库中必须是唯一的。然而，如果准备在服务器之间发送消息，那么在设置服务代理对象名称时，就必须加倍细心。一般情况下， 我们都推荐将系统名称和功能名称组合起来作为服务代理对象名称。这样可以确保在整个企业

中，对象名称都是唯一的。当在内部使用了服务代理的系统之间发送消 息并且想要避免对象命名问题时，你会发现这样做是非常方便的。可以使用 UNC 形式名称来保证唯一性。比如，我们将从 tcp://SearchSQLServer/开始对象的命名。

1、 第一步需要在服务代理中设置的对象是消息类型，它告诉 SQL Server 消息的基本信息。服务代理消息可以是任意数据类型：文本、二进制、XML、数据等等。推荐使用的是 XML，因为它可以包含其它任意数据类型。 我们使用 CREATE MESSAGE TYPE 命令来创建消息类型。其中有四个验证选项。如果想要执行有效的 XML 格式化，可以选择 WELL_FORMED_XML 或 VALID_XML WITH SCHEMA COLLECTION（使用一个 XSD 来执行 XML 模式）。

```
CREATE MESSAGE TYPE [tcp://SearchSQLServer/SampleMessageType] AUTHORIZATION
dbo
VALIDATION = NONE
```

2、 下一步我们需要配置的对象是契约（Contract）。它告诉 SQL Server 哪些消息类型是相关的。我个人倾向于在一个特定进程中对所有的通信使用一个的消息类型，以保持简单。然而，Microsoft SQL Server Books OnLine 的例子在发送系统使用一个消息类型，而接收系统使用另一个类型。

```
CREATE CONTRACT [tcp://SearchSQLServer/SampleContract] AUTHORIZATION dbo
(
[tcp://SearchSQLServer/SampleMessageType] SENT BY ANY
)
```

3、现在让我们来看看服务代理上的实际队列。这个队列有点像一个表格。在这里存储了等待处理的的消息。与表格不一样的是，你无法定义队列的模式。因此，推荐使用 XML 来发送，这样你才可以在 XML 中来定义模式。

```
CREATE QUEUE [tcp://SearchSQLServer/SampleQueue] AUTHORIZATION dbo
```

下一个服务代理的配置对象是服务。服务是服务代理用来在数据库中消息传送到正确的队列，同时将契约绑定到队列消息中。

```
CREATE SERVICE [tcp://SearchSQLServer/SampleService] AUTHORIZATION dbo
ON QUEUE [tcp://SearchSQLServer/SampleQueue]
(
[tcp://SearchSQLServer/SampleContract]
)
```

(作者：Denny Cherry 译者：曾少宁/陈柳　来源：TT 中国)

配置 SQL Server 服务代理来发送存储过程数据（二）

4、 最后一个对象是路由，但它是可选的。路由只有在数据库服务器之间发送消息时才是必须的。路由与接收消息服务的本地副本器密不可分。如果服务器接收的消息是 另外一个使用数据库镜像的服务器的镜像，那么就必须使用一个 MIRROR_ADDRESS 参数。如果想让路由自动删除，那么可以设置 LIFETIME 标 志。

```
CREATE ROUTE [tcp://SearchSQLServer/SampleRoute] AUTHORIZATION dbo
WITH SERVICE_NAME = '[tcp://SearchSQLServer/SampleService]',
BROKER_INSTANCE = 'AB2F3EB9-6662-4AAF-8682-A9A48C3BDD3B',
ADDRESS = 'TCP://RemoteServer:8888',
MIRROR_ADDRESS = 'TCP://MirrorServer:8888'
```

BROKER_INSTANCE 参数是 service_broker_guid 字段的值，它来自远程服务器上的数据库的 sys.databases 视图。

为了实现队列之间的消息发送，必须使用两个命令。第一个是 BEGIN DIALOG 命令，然后是 SEND 命令。使用 BEGIN DIALOG 命令在发送和接收服务器之间来创建一个会话。如果使用一个已有的会话，那么就不需要使用 BEGIN DIALOG 命令。在使用 BEGIN DIALOG 命令创建了新的会话之后，就可以用 SEND 命令在会话上发送消息，其中消息的发送是通过 BEGIN DIALOG 命令返回的句柄实现。

```
DECLARE/font> @dialog_handle UNIQUEIDENTIFIER,
@XMLData XML ;
```

```
SET @XMLData = (SELECT * FROM sys.tables FOR XML AUTO)
BEGIN DIALOG @dialog_handle
FROM SERVICE [tcp://SearchSQLServer/SampleService]
TO SERVICE 'tcp://SearchSQLServer/SampleService'
ON CONTRACT [tcp://SearchSQLServer/SampleContract];
SEND ON CONVERSATION @dialog_handle
MESSAGE TYPE [tcp://SearchSQLServer/SampleMessageType]
(@XMLData);
```

RECEIVE 命令是用来查看和处理消息。RECEIVE 命令与 SELECT 非常相似，都会返回数据。但是，当使用 RECEIVE 命令时，消息只能接收一次。在消息接收 之后，如果它是会话中最后的消息——很多人都只在会话中输入一个消息，那么就使用 END CONVERSATION 命令来关闭它。RECEIVE 命令可以与 WAITFOR 命令一起使用，而 WHILE 循环用来处理程序的一次运行中的队列上的所有消息。

```
DECLARE @dialog_handle UNIQUEIDENTIFIER,
@XMLData XML ;
RECEIVE TOP (1) @dialog_handle = conversation_handle,  @XMLData =
CAST(message_body AS XML)
FROM [tcp://SearchSQLServer/SampleQueue]
END CONVERSATION @dialog_handle
SELECT @XMLData
```

结束会话时，事实接收队列会发送一个消息到发送队列，以此来通知发送队列、服务和 SQL Server，会话结束了。这个结束会话消息必须由发送队列处理以便将数据从队列中删除。建议在发送队列上使用基本程序来自动清除这些消息。

```
CREATE PROCEDURE usp_ProcessAck
AS
DECLARE @xml AS XML
DECLARE @dialog_handle as uniqueidentifier
WHILE 1=1
SET @dialog_handle = NULL
WAITFOR ( RECEIVE TOP (1) @dialog_handle = conversation_handle,  @xml =
cast(message_body as xml)
FROM [tcp://SearchSQLServer/SampleQueue]),  TIMEOUT 1000
IF @dialog_handle IS NULL
break
END CONVERSATION @dialog_handle
END
GO
```

完 成这个程序之后，当消息到达时，可以通过 QUEUE 的 ACTIVATION 命令来使用 ALTER QUEUE 命令设置一个队列自动地运行该程序。你可以设置程序的并行执行数目，将 MAX_QUEUE_READERS 设置为大于 1。在高负荷的系统中，或 当处理花费一定时间时，附加队列读取器可以通过并行方式加速数据处理。

```
ALTER QUEUE [tcp://SearchSQLServer/SampleQueue]
WITH ACTIVATION (STATUS=ON,
PROCEDURE_NAME = dbo.usp_ProcessAck,
MAX_QUEUE_READERS = 2,
EXECUTE AS SELF)
```

SQL Server Service Broker 的安装可能有点复杂，并且在因特网上相关的文档也不是很多。但是，一旦安装并正确运行了服务代理，它可以提供一个坚如磐石的内部或外部数据库通信平台，实现系统之间数据的快速一致发送。

（作者：Denny Cherry 译者：曾少宁/陈柳 来源：TT 中国）

用 SQL Server 2005 CTE 简化查询（一）

SQL Server 2005 引进了一个很有价值的新的 Transact-SQL 语言组件：一个通用表表达式（Common Table Expression，CTE），它是派生表和视图的一个便捷的替代。通过使用 CTE，我们可以创建一个命名结果集来在 SELECT、INSERT、 UPDATE 和 DELETE 语句中引用，而无须保存结果集结构的任何元数据。在本文中，我将阐述如何在 SQL Server 2005 中创建 CTE——包括如何使用 CTE 来创建一个递归查询——并举几个例子来说明它们是如何使用的。注意，本文中所有例子都使用 SQL Server 2005 的 AdventureWorks 示例数据库。

在 SQL Server 2005 中创建一个基本 CTE

我们可以在 SELECT、INSERT、UPDATE 或 DELETE 语句之前添加一个 WITH 子句来构成一个 CTE。下面的语法显示了 WITH 子句的基本构造和 CTE 定义：

```
[WITH <CTE_definition> [, ...n]]
<SELECT, INSERT, UPDATE, or DELETE statement that
calls the CTEs>
<CTE_definition>::=
CTE_name [(column_name [, ...n ])]
AS
(
CTE_query
)
```

如上面语句所示，你可以在可选的 WITH 子句中定义多个 CTE。CTE 定义包含 CTE 名称、CTE 字段名称、AS 关键字和括号中的 CTE 查询。注意，CTE 字段名称的数目必须与 CTE 查询返回的字段数目相匹配。另外，如果 CTE 查询提供所有字段名称，那么字段名称是可选的。

现在我们已经对 SQL Server 的 CTE 语法有了基本的了解，下面让我们来看一个 CTE 定义的例子，以便更好地理解这个语法。下面的例子定义了一个命名为 ProductSold 的 CTE，接着在 SELECT 语句中引用了 CTE：

```
WITH ProductSold (ProductID, TotalSold)
AS
(
SELECT ProductID, SUM(OrderQty)
FROM Sales.SalesOrderDetail
GROUP BY ProductID
)
SELECT p.ProductID, p.Name, p.ProductNumber,
ps.TotalSold
FROM Production.Product AS p
INNER JOIN ProductSold AS ps
ON p.ProductID = ps.ProductID
```

这里可以看到，WITH 子句在引用 CTE 的 SELECT 语句之前。WITH 子句的第一行包含了 CTE 的名称（ProductSold）以及在 CTE 的两个字段名称(ProductID 和 TotalSold)。接着是 AS 关键字，紧接着是括号中的 CTE 查询。这样，CTE 查询返回了每个产品的销售总数。

当 Product 表与 CTE 联接时，WITH 子句后面的 SELECT 语句根据 ProductID，引用了 CTE 的名称。这个语句就像调用一个表格或视图一样 调用 CTE。但是，与表格与视图不同

的是，CTE 只对 WITH 子句的语句有效。如果在一个后续语句中引用 CTE——而没有重新定义 CTE——就会出错。

使用通用表表达式的其中一个优点是你可以在调用语句中多次引用 CTE。比如，下面的语句定义了一个命名为 Employees 的 CTE，接着在跟在 WITH 子句的 SELECT 语句中两次调用 CTE：

```
    WITH Employees (EmpID,  MgrID,  FName,  LName,  Email)
AS
(
SELECT e.EmployeeID,  e.  ManagerID,
c.FirstName,  c.LastName,  c.EmailAddress
FROM HumanResources.Employee e
INNER JOIN Person.Contact c
ON e.ContactID = c.ContactID
)
SELECT e.FName + ' ' + e.LName AS EmpName,
e.Email AS EmpEmail,
m.FName + ' ' + m.LName AS MgrName,
m.Email AS MgrEmail
FROM Employees e
LEFT OUTER JOIN Employees m
ON e.MgrID = m.EmpID
```

在这个例子中，SQL Server CTE 查询返回一个员工 ID、姓名和邮件地址以及他们的经理 ID 的列表。然后，跟在 WITH 子句后的 SELECT 语句将与 CTE 联接来返回经理的姓名和邮件地址。你可以通过使用派生表（子查询）来获得相同的结果，但是这就意味着需要多次重复相同的子查询，同时也使代码更加复杂。

（作者：Robert Sheldon 译者：曾少宁／陈柳 来源：TT 中国）

用 SQL Server 2005 CTE 简化查询（二）

在 WITH 子句中创建多个 CTE

在 CTE 语法中可以看到，我们可以在 WITH 子句中定义多个 CTE，然后在接下来的语句中按照需要多次调用这些 CTE。下面的例子说明了这是如何实现的。下面的 WITH 子句包含了两个 CTE 定义：

```
WITH
Cost (ProductID,  AvgCost)
AS
(
SELECT ProductID,  AVG(StandardCost)
FROM Production.ProductCostHistory
GROUP BY ProductID
),
Sold (ProductID,  AvgSold)
AS
(
SELECT ProductID,  AVG(OrderQty)
FROM Sales.SalesOrderDetail
GROUP BY ProductID
)
SELECT p.ProductID,  p.Name,
(AvgCost * AvgSold)AS TotalCost
```

```
FROM Sold s
INNER JOIN Production.Product p
ON s.ProductID = p.ProductID
INNER JOIN Cost c
ON p.ProductID = c.ProductID
```

创建一个递归通用表表达式

SQL Server 中 CTE 最有价值的功能是创建递归查询的功能——一种反复自身引用以返回数据子集的查询类型。递归查询最常用于返回层次式数据。比如，在 AdventureWorks 数据库中的 Employee 表包含了每个员工的经理 ID。事实上，经理 ID 是该经理管理的员工 ID。因此， Employee 表格包含了从 CEO 往下的整个管理层次报告结构。

可以通过创建一个 CTE 查询来定义一个 CTE 来检索这个分层结构，其中这个查询使用一个 UNION ALL、UNION、INTERSECT 或 EXCEPT 操作符来联接多个 SELECT 语句。下面让我们来看个例子。

在这个 WITH 子句中，CTE 查询包含两个用 UNION ALL 操作符联接的 SELECT 语句：

```
WITH Reports (EmpLevel, EmpID, ContactID, MgrID)
AS
(
SELECT 1, EmployeeID, ContactID, ManagerID
FROM HumanResources.Employee
WHERE ManagerID IS NULL
UNION ALL
SELECT r.EmpLevel + 1, e.EmployeeID, e.ContactID,
e.ManagerID
```

```
FROM HumanResources.Employee e
INNER JOIN Reports r
ON e.ManagerID = r.EmpID
)
SELECT r.EmpLevel,  r.EmpID,
c.FirstName + ' ' + c.LastName AS EmpName,
MgrID
FROM Reports r
INNER JOIN Person.Contact c
ON r.ContactID = c.ContactID
ORDER BY r.EmpLevel,  r.MgrID,  r.EmpID
```

在 CTE 查询中的第一个 SELECT 语句仅仅检索顶层员工——CEO。为了检索到顶层员工，可以使用一个指定 ManagerID 值为 NULL 的 WHERE 子句实现。

换言之，这个人选并不直接向另一个经理报告。注意：SELECT 语句中的第一个字段是 1。它用于标识这个查询返回的员工是在最高层，第一层。

第二个 SELECT 语句（在 UNION ALL 操作符之后）基于经理和员工 ID 将 Employee 表格与 Reports CTE 联接。通过这种方式的自引用，SQL Server 自动将其作为递归查询并按照需要重复检索以返回每个层次的员工。每次查询运行时，第一个字段值设为 1，然后每层都递增 1。

在 WITH 子句后的 SELECT 语句将 Reports CTE 与 Contact 表联接来检索员工的姓名。下面的查询结果显示了这个语句返回的数据样本：

EmpLevel	EmpID	EmpName	MgrID
1	109	Ken Sanchez	NULL

2	6	David Bradley	109
2	12	Terri Duffy	109
2	42	Jean Trenary	109
2	140	Laura Norman	109
2	148	James Hamilton	109
2	273	Brian Welcker	109
3	2	Kevin Brown	6
3	46	Sariya Harnpadoungsataya	6
3	106	Mary Gibson	6
3	119	Jill Williams	6
3	203	Terry Eminhizer	6
3	269	Wanida Benshoof	6
3	271	John Wood	6
3	272	Mary Dempsey	6
3	3	Roberto Tamburello	12
3	66	Janaina Bueno	42
3	102	Dan Bacon	42
3	117	François Ajenstat	42
3	128	Dan Wilson	42
3	149	Ramesh Meyyappan	42
3	150	Stephanie Conroy	42
3	176	Karen Berg	42
3	30	Paula Barreto de Mattos	140
3	71	Wendy Kahn	140

3	103	David Barber	140
3	139	David Liu	140
3	21	Peter Krebs	148
3	44	A. Scott Wright	148
3	200	Hazem Abolrous	148
3	218	Gary Altman	148
3	268	Stephen Jiang	273
3	284	Amy Alberts	273
3	288	Syed Abbas	273
4	4	Rob Walters	3
4	9	Gail Erickson	3
4	11	Jossef Goldberg	3

结果是根据员工的级别排列的。注意，第二行到第七行显示的是 MgrID 值为 109 的数据，它是第一行显示的顶层员工的 ID。下面的行反映了相同分层的数据。

递归 CTE，与 SQL Server 中的其它通用表表达式一样，提供了强大的数据检索功能。与视图不同的是，它并不需要保存元数据。与派生表不同的是，它并不需要重复不必要的代 码。CTE 可以将代码分成不相关的单元，这有助于简化代码复杂性。对于递归查询，CTE 则更加好用。刚开始使用 CTE 时，你可能必须花点时间来适应它们， 但是一旦熟悉了，那么你将乐在其中。

（作者：Robert Sheldon 译者：曾少宁/陈柳 来源：TT 中国）