专家汇总：

SQL Server 十大热门技巧

专家汇总：SQL Server 十大热门技巧

本专题是专家对目前 SQL Server 最热门和实用的十大技巧进行的总结。定义各种数据类型、配置内存设置或有关 SQL Server 表和日志问题的解决等等都是目前人们最关注的话题、也是作为数据库管理员、开发员必须了解的话题。

10、设计 SQL Server 集簇索引以提升性能

SQL Server 的集簇索引是数据库整体架构的一个非常重要的方面。它们经常被忽视、误解，或者如果数据库很小，它们会被认为是不重要的。在本文中作者将主要说明 SQL Server 集簇索引是如何存储在硬盘中的，为什么它们应该一直随着时间增加以及为什么静态的集簇索引是最好的。我同时也将探讨多对多表，为什么它们会被使用，以及集簇索引如何能够让这些表效率更高。

- 设计 SQL Server 集簇索引以提升性能（一）
- 设计 SQL Server 集簇索引以提升性能（二）

9、SQL Server 2005 的 XML 数据类型和 VARCHAR(MAX)

在 SQL Server 2005 中选择 XML 数据类型或 VARCHAR (MAX)数据类型时，数据库管理员和开发员应该知道它们的个子对性能的影响。在本文中，我将向你介绍一个例子，它使用了两个表，一个用于插入和查询 XML 数据，而另一个使用 VARCHAR(MAX)数据类型，从而为你的 SQL Server 环境作出最佳选择。

- SQL Server 2005 的 XML 数据类型和 VARCHAR(MAX)之一
- SQL Server 2005 的 XML 数据类型和 VARCHAR(MAX)之二

8、如何创建与 DB2 链接的 SQL Server 服务器

许多 SQL Server 环境运行着 DB2 服务器，并且两个服务器之间必须进行连接。本文将为你逐步讲解创建 SQL Server 链接服务器的步骤，以帮助解决 SQL Server DBA 和 DB2 DBA 之间的语言鸿沟。

- ❖ 如何创建与 DB2 链接的 SQL Server 服务器（一）
- ❖ 如何创建与 DB2 链接的 SQL Server 服务器（二）

7、SQL Server 数据库设计灾难：不该做什么

如果一个外人仔细地查看你的 SQL Server 数据设计时，你会感觉到窘迫吗？有没有可能在你的表中实现一个外键约束呢？你是否在字段中使用了正确的数据类型？你是否按规范定义的方式进行表/字段命名？数据库体系架构师 Brian Walker 根据他多年的 SQL Server 经验，提出了许多用于改进数据库设计和 SQL Server 性能的建议……

- ❖ SQL Server 数据库设计灾难：不该做什么

6、用存储过程查询 SQL Server 表和其它对象大小

在对象决定 SQL Server 磁盘空间时，微软 sp_spaceused 就有限制。本文中原始的存储程序，sp_spaceused，就是用来计算 SQL Server 中特定的对象空间。我们可以用它来查看数据库的用户表大小概况、计算一组表所占用的空间总大小以及查看前 10 个最大的索引对象。

- 用存储过程查询 SQL Server 表和其它对象大小

5、如何使用向导设置 SQL Server 2005 日志传送

在 SQL Server2005 中建立日志传送时，你可以用许多种方法来实现主服务器和副服务器之间的最优配置。除了一些最佳实践方法之外，我们还要从头到尾遵循 SQL Server MVP Hilary Cotter 日志传送安装过程，其中包括选择合适的数据库恢复模式、事务日志路径和副数据库设置……

- 如何使用向导设置 SQL Server 2005 日志传送（一）
- 如何使用向导设置 SQL Server 2005 日志传送（二）
- 如何使用向导设置 SQL Server 2005 日志传送（三）

4、用存储过程检查 SQL Server 数据库和日志文件大小

解 SQL Server 数据库的大小是许多 DBA 的职责之一，而这个职责你可以轻松通过存储过程 sp_SDS 来完成。sp_SDS 不仅能确定“SQL 数据库空间”，而且它还能监测数据库的增长，提醒 DBA 关于数据或日志文件的增长，执行事务日志备份，甚至提供详细的文件级明细表，这样 DBA 可以压缩文件以获取最大空余空间。本文介绍了完整的 sp_SDS 及其算法。本文还将进一步阐述如何查询数据库对象的大小，包括 SQL Server 表。

- 用存储过程检查 SQL Server 数据库和日志文件大小（一）
- 用存储过程检查 SQL Server 数据库和日志文件大小（二）
- 用存储过程检查 SQL Server 数据库和日志文件大小（三）

3、SQL Server 2005 的 DATETIME 和 SMALLDATETIME 基础

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。在本篇技巧中，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。

- 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）

2、配置 SQL Server 内存设置

SQL Server 中有一些十分重要的内存设置。本篇技巧中，SQL Server 专家 Denny Cherry 将告诉我们 SQL Server 最适合的 RAM 总数、如何进行 AWE 内存管理、最大服务器内存设置以及 32 位和 64 位平台之间的区别。

- 配置 SQL Server 内存设置（一）
- 配置 SQL Server 内存设置（二）

1、SQL Server 中日期/时间值到字符类型的数据转换

Transat-SQL 支持的两种内置日期/时间数据和字符数据数据之间的转换方法。在 2008 年最关注的 10 大技巧中，SQL Server 专家 Robert Sheldon 将向你逐步介绍每种方法的过程和步骤——隐式转换和显式转换。这些步骤包括用手动 CAST 和 CONVERT 函数转换日期/时间值。

- 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）

设计 SQL Server 集簇索引以提升性能（一）

SQL Server 的集簇索引是数据库整体架构的一个非常重要的方面。它们经常被忽视、误解，或者如果数据库很小，它们会被认为是不重要的。

本文阐述了集簇索引对于整个系统性能以及数据库增大时维护的重要性。我将主要说明 SQL Server 集簇索引是如何存储在硬盘中的，为什么它们应该一直随着时间增加以及为什么静态的集簇索引是最好的。我同时也将探讨多对多表，为什么它们会被使用，以及集簇索引如何能够让这些表效率更高。

最后，很重要的是我们会讨论新的 SQL Server 2005 分割表概念，并探讨分割表是如何影响集簇索引的。这将有助于你以后作为出正确的决定性。

集簇索引默认是与匹配主键相匹配的，而主键是定义在 SQL Server 表上的。然而，你可以在任何字段上创建一个集簇索引，然后在另一个字段或多个字段上定义一个主键。这时，这个主键将会作为一个唯一的非集簇索引被创建。典型地，一个集簇索引会与主键相匹配，但这并不是必须的，所以要仔细考虑。对于各种可能出现的情况，我将讨论集簇索引本身，而不管你是否选择将它与主键相匹配。

集簇索引实际上装载了 SQL Server 的数据记录行，所以你的集簇索引存储的地方就是你的数据存储的地方。集簇索引是按数据范围而组织的。比如，1 到 10 之间的值存储为一个范围，而 90 到 110 是另一个范围。因为集簇索引是按范围存储的，如果你需要在一个范围中搜索一个审计日志，使用基于日期字段的集簇索引效率会更高，其中日期字段会用于返回日期范围。非集簇索引更适用于具体值的搜索，比如“等于 DateValue 的日期”，而不是范围搜索，比如“在 date1 和 date2 之间的日期”。

集簇索引的不断增加值

集簇索引必须是基于值不断增加的字段。在前面的例子中，我使用了审计日志的日期字段，审计日志的日期值是不断增加的，而且旧的日期将不会再插入到数据表中。这就是一个“不断增加”字段。另一个不断增加值的好例子就是标识字段，它也是从创建后就持续恒定增加的。

为什么我在这里花这么多时间讨论集簇索引的不断增加值呢？这是因为集簇索引的最重要的属性就是它们是不断增加的并且本质上是静止的。不断增加之所以重要的原因与我之前提到的范围架构有关。如果值不是不断增加的，SQL Server 就必须在现有记录的范围内分配位置，而不是直接将它们放到索引后面的新的范围中。

如果值不是不断增加的，那么当范围的值用完后再出现一个已经用索引范围的值时，SQL Server 将做一个页拆分插入一个索引。在实现时，SQL Server 会将已填满的页拆分成两个单独的页，这两个页此时会有更多的值空间，但这需要更多的资源去处理。你可以通过设置填充参数为 70%来作好预备工作，这样就可以有 30%的自由空间来为后来的值使用。

这个方法的问题是你必须不断地“再索引”集簇索引使它能维持 30%的自由空间。对集簇索引进行再索引会带来繁重的 I/O 负载，因为它必须移动它的实际数据，并且任何非集簇索引都必须重建，这会增加许多的维护时间。

如果集簇索引是不断增加的，你将不需要重建集簇索引。你可以将集簇索引的填充因数设置为 100%，这样随着时间的推移，你就只需要对于不集中的、非集簇索引进行再索引，这样就可以增加数据库在线时间。

不断增加的值将只会在索引的尾部添加新值，并且只在需要的时候才创建新的索引范围。由于新的值都只会不断地添加到索引的尾部而且填充因数为 100%，所以将不再会有逻辑碎片出现。填充因数越高，每一页所填充的记录行就越多。更高的填充因数使得查询时会需要更少的 I/O、RAM 和 CPU 资源。你查询集簇索引中越少的数据类型，JOIN/查询操作速度会更快。同时，因为每一个非集簇索引都要求包括集簇索引键，所以集簇索引键和非集簇索引也会更小。

集簇索引的最佳数据类型是非常狭窄的。对于数据类型大小，它通常是 smallint、int、bigint 或 datetime。当 datetime 值用作集簇索引时，它们是唯一的字段并且通常是不断增加的日期值，这些值通常是作为范围数据查询的。通常，你应该避免组合（多字段）集簇索引，除了以下情况：多对多数据表和 SQL Server 2005 分割表，这种分割表有分割的字段，它包含了集簇索引而允许索引排列。

（作者：Matthew Schroeder 译者：陈柳/曾少宁 来源：TT 中国）

设计 SQL Server 集簇索引以提升性能（二）

多对多表和集簇索引

多对多表有非常快速的 JOIN 并允许快速的从一个记录到另一个记录的重新联合。设想有下面这样的数据结构：

Customer

CustomerID (bigint identity)	Name	Fieldn+

CustomerOrder

CustomerID	OrderID

Orders

OrderID (bigint identity)	Date	Fieldn+

这些结构中的集簇索引是 CustomerID 和 OrderID。组合键是 CustomerID/OrderID。下面这个结构的优点：

JOIN 都是基于集簇索引（比非集簇索引 JOIN 快很多）。

将一个 Order 赋给另一个 Customer 只需要对 CustomerOrder 表作一个 UPDATE 操作，这是改动非常少的，只影响到一个集簇索引。因此，它减少了你在更新一个大表时的数据库锁的时间，如 Orders 表。

使用多对多表就不需要大表中的一些非集簇索引，如 Customer/Orders。因此，它减少大表的维护时间。

这个方法的一个缺点是 CustomerOrder 表的碎片（不是连续的）。然而，这并不是一个大问题，因为这个表是相对较小的，它只有 2 个字段，数据类型也很少，并且只有一个集簇索引。这些本来是在包括 CustomerID 的 Orders 表的非集簇索引的减少，带来的好处是大于额外开销的。

SQL Server 2005 的集簇索引和分割表

SQL Server 2005 中的分割表是一些表面上为一个独立的表，但实际上——在存储子系统中——它们包含能够存储在许多文件组（Filegroup）的多个部分。表的部分是根据一个字段的值来分割成不同文件组的。这种方式的分割表会有几个缺点。这里我会说明几个基本的缺点，希望能让你对相关的原因有一些了解。我建议你使用分割表时先学习它的使用方法。

你可以在这个环境中基于一个字段创建一个集簇索引。但是，如果这个字段不是表用于分割的那个字段，那么集簇索引就被称为非对齐的。如果一个集簇索引是非对齐的，那么任何分区的数据进/出（或合并）都需要你删除集簇索引以及非集簇索引，然后再重建这些索引。这是必要的，因为 SQL Server 并不知道集簇/非集簇索引的哪部分属于哪个表的分区。毫无疑问，这会带来一定的系统停机时间。

分割表上的集簇索引应该总包含常规的集簇字段，它是不断增加和静态的，并且它是用于分割数据库表的。如果集簇索引包含用于分割表的字段，那么 SQL Server 就知道集簇/非集簇索引的哪个部分属于哪个分区。当一个集簇索引包含了用于分割表的字段时，那么这个集簇索引就是“对齐的”。这时表的分区就可以在数据进/出（和合并）而不需要重建集簇/非集簇索引，这样就不会带来额外的系统停机时间。表的 INSERT/UPDATE/DELETE 操作也会更快，因为这些操作只需要关注处于它们自己分区的索引。

总结

SQL Server 集簇索引是数据体系结构的一个重要部分，我希望你已经从本文学习中知道了为什么你需要在一开始就仔细设计好集簇索引。集簇索引应该是窄小的、静态的和不断增加的，这对于将来数据库的健壮性是非常重要的。集簇索引可能帮你实现更快速的 JOIN 和 IUD 操作，并最小化系统的忙时拥塞时间。

最后，我们讨论了 SQL Server 2005 的分割表是如何影响你对集簇索引的使用，集簇索引与分区的“对齐”的意思是什么，以及为什么集簇索引必须按顺序对齐以使分割表正常工作。请继续关注关于将在二月发表的文章《非集簇索引》（第二部分）和三月发表的文章《最优索引维护》（第三部分）。

（作者：Matthew Schroeder　译者：陈柳/曾少宁 来源：TT 中国）

SQL Server 2005 的 XML 数据类型和 VARCHAR(MAX)之一

作为一个数据库管理员，我试着仔细分析性能问题以及如何保证使用 XML 时不要影响 SQL Server 性能。在本文中，我将向你介绍一个例子，它使用了两个表，一个用于插入和查询 XML 数据，而另一个使用 VARCHAR(MAX)数据类型。然后我们观察存储、CPU 和 I/O 的测试结果，从而为你的 SQL Server 环境作出最佳选择。

注意：这里使用的测试仅仅针对基本的表，不使用索引。

如果你对于在 SQL Server 使用 T-SQL 命令和 XML AUTO 的应用性能比较有兴趣，请阅读我的上一篇技巧文章。

XML 数据类型

XML 数据类型会由 SQL Server 在进行检查时与 VARCHAR(MAX)比较，以保证它的内容是合法的 XML。

测试环境描述

对于我的测试，我将使用从一个*.rdl 文件（Reporting Services Report）拷贝的 XML，大小为 265KB。我将创建两个有相同结构的表，只是其中的 MyXML 域，一个表使用 XML 数据类型，而另一个表使用 VARCHAR(MAX)数据类型。

```
    <create table TryVACRCHARDatatype(
id int identity not null,
MyXML VARCHAR(MAX) null
) Go
create table TryXMLDatatype (
```

```
id int identity not null,
MyXML XML null
)Go
```

我将用相同的方法向每一个表插入 XML 数据：

```
set statistics io on
Go
declare @XML XML
--
 My big XML (for space reasons I am not including all of it here) SET @XML = '
<? xml version="1.0" encoding="utf-8"? >
.....
'
insert into TryXMLDatatype (MyXML) values (@XML)
go
declare @Varch VARCHAR(MAX)
-- My big XML (for space reasons I am not including all of it here)
SET @Varch = '<? xml version="1.0" encoding="utf-8"? >
.....
'
insert into TryXMLDatatype (MyXML) values (@Varch)
Go
```

插入 XML 数据到表中

我在 SQL Profiler 中监控上面的插入命令，并得到 I/O 统计，运行两次。

I/O 统计结果：

TryXMLDatatype 表。扫描次数：0，逻辑读取次数：1，物理读取次数：0，预读次数：0，慢速逻辑读取次数：18，慢速物理读取次数：0，慢速预读次数：0。

TryVACRCHARDatatype 表。扫描次数：0，逻辑读取次数：1，物理读取次数：0，预读次数：0，慢速逻辑读取次数：90，慢速物理读取次数：0，慢速预读次数：0。

执行计划显示了两个命令的精确计划。

Profiler 结果（每一个插入的两次执行）：

EventClass	TextData	CPU	Reads	Writes	Duration
SQL:BatchCompleted	declare @XML XML SET @XML = '<?xml version...	47	103	8	781
SQL:BatchCompleted	declare @varch VARCHAR (MAX) SET @varch = ...	16	224	32	616
SQL:BatchCompleted	declare @XML XML SET @XML = '<?xml version...	62	103	8	751
SQL:BatchCompleted	declare @varch VARCHAR (MAX) SET @varch = ...	31	224	32	

注意：XML 插入占用更多的 CPU 而读/写则比较少。同时插入 XML 数据需要时间更长。我将在下一部分分析这个行为。

查询表

```
I ran SELECT * FROM in each table and monitored with Profiler and Statistics I
/O:
SELECT * FROM TryXMLDatatype
Go
SELECT * FROM TryVACRCHARDatatype
Go
```

结果：

注意：域中的值是不同的！

I/O 统计结果：

TryXMLDatatype 表：扫描次数：0，逻辑读取次数：1，物理读取次数：0，预读次数：0，慢速逻辑读取次数：56，慢速物理读取次数：0，慢速预读次数：0。

TryVACRCHARDatatype 表：扫描次数：0，逻辑读取次数：1，物理读取次数：0，预读次数：0，慢速逻辑读取次数：200，慢速物理读取次数：0，慢速预读次数：0。

XML 数据类型占用更少的 I/O。、

同样，执行计划成本显示了两个命令的完全相同的计划。

EventClass	TextData	CPU	Reads	Writes	Duration
SQL:BatchCompleted	SET STATISTICS XML ON	0	0	0	1
SQL:BatchCompleted	SELECT * FROM TryXMLDatatype	0	59	0	308
SQL:BatchCompleted	SELECT * FROM TryVACRCHARDatatype	0	203	0	102
SQL:BatchCompleted	SET STATISTICS XML OFF	0	0	0	1
SQL:BatchCompleted	SET STATISTICS XML ON	0	0	0	0
SQL:BatchCompleted	SELECT * FROM TryXMLDatatype	0	59	0	200
SQL:BatchCompleted	SELECT * FROM TryVACRCHARDatatype	0	203	0	351
SQL:BatchCompleted	SET STATISTICS XML OFF	0	0	0	0

为什么两个表的读取操作差别这么大呢？

注意：查询 XML 的时间更长。时间延长的很多不同的因素是基于机器的活动。我将忽略这些不同点，主要是因为没有 CPU 显示。如果涉及到 CPU 活动，它可能解释了时间延长的不同点但是这并不是重点。

让我查询一下每一个表的字段长度：

```
SELECT datalength(MyXML) FROM TryXMLDatatype
Go
SELECT datalength(MyXML) FROM TryVACRCHARDatatype
Go
```

很意外，XML 域的字符比另一个更少。原因如下：

XML--less I/O:

当插入 XML 数据类型时，“附加”数据，如引号和制表符会从域中删除。结果是更加节约了存储空间。

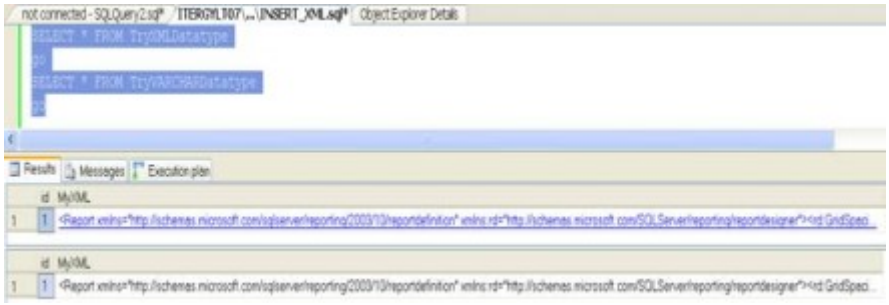
我尝试从 XML 字段拷贝值到 VARCHAR(MAX)字段，如下：

```
truncate table TryVARCHARDatatype
Go
insert into TryVARCHARDatatype (MyXML)
select convert(varchar(max), MyXML) from TryXMLDatatype
Go
```

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

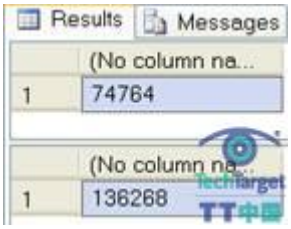
SQL Server 2005 的 XML 数据类型和 VARCHAR(MAX)之一

结果：



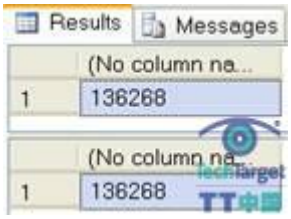
看起来是一样的，但是……

```
SELECT datalength(MyXML) FROM TryXMLDatatype
Go
SELECT datalength(MyXML) FROM TryVACRCHARDatatype
Go
```



这比前面好一些（现在是 136，268，而不是 271，210），但情况仍然相同。如果我将 XML 转化到 VARCHAR(MAX)然后计算长度：

```
SELECT
datalength(convert(varchar(max), MyXML)) FROM TryXMLDatatype
go
SELECT datalength(MyXML)FROM TryVARCHARDatatype
Go
```



长度相同。

这意味着 XML 数据类型的存储方式更加高效。现在两个表包括了相同的数据，让我们再看看 Profiler 的跟踪命令 select * from both tables (three executions)的结果：

TextData	CPU	Reads	Writes	Duration
SELECT * FROM TryXMLDatatype	16	83	0	273
SELECT * FROM TryVARCHARDatatype	15	103	0	182
SELECT * FROM TryXMLDatatype	16	83	0	95
SELECT * FROM TryVARCHARDatatype	0	103	0	100
SELECT * FROM TryXMLDatatype	0	83	0	277
SELECT * FROM TryVARCHARDatatype	0	103	0	195

XML 数据类型的读取数少了 20%。

I/O 统计：

TryXMLDatatype 表：扫描次数：0，逻辑读取次数：1，物理读取次数：0，预读次数：0，慢速逻辑读取次数：56，慢速物理读取次数：0，慢速预读次数：0。

TryVARCHARDatatype 表：扫描次数：0，逻辑读取次数：1，物理读取次数：0，预读次数：0，慢速逻辑读取次数：100，慢速物理读取次数：0，慢速预读次数：0。

同样，这里 XML 数据类型的读操作也效率更高。

查询到 XML 转换为 VARCHAR（MAX）是怎样的呢？

```
SELECT convert(varchar(max), MyXML) FROM TryXMLDatatype
Go
SELECT MyXML FROM TryVARCHARDatatype
Go
```

TextData	CPU	Reads	Writes	Duration
SELECT convert(varchar(max),MyXML) ...	0	23	0	293
SELECT MyXML FROM TryVARCHARDatatype	0	103	0	141
SELECT convert(varchar(max),MyXML) ...	15	23	0	199
SELECT MyXML FROM TryVARCHARDatatype	0	103	0	[illegible]
SELECT convert(varchar(max),MyXML) ...	16	23	0	105
SELECT MyXML FROM TryVARCHARDatatype	0	103	0	[illegible]

TryXMLDatatype 表：扫描次数：1，逻辑读取次数：1，物理读取次数：0，预读次数：0，慢速逻辑读取次数：20，慢速物理读取次数：0，慢速预读次数：9。

TryVARCHARDatatype 表：扫描次数：1，逻辑读取次数：1，物理读取次数：0，预读次数：0，慢速逻辑读取次数：100，慢速物理读取次数：0，慢速预读次数：0。

对于不变行为的唯一度量标准是 I/O 读取。而且，XML 转换查询还会使用 CPU 资源来实现转换——这很正常。

结论

XML 数据类型表示存储纯 XML 数据，它不包括不必要的头尾字符。这带来的结果是通过 I/O 方法实现更划算的存储方式，然而这里仍然有一些 CPU 资源用于验证 XML 是有否效。

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

如何创建与 DB2 链接的 SQL Server 服务器（一）

许多 SQL Server 环境都运行着必须链接到 SQL Server 的 DB2 服务器。链接到一个 DB2 服务器是有挑战性的，因为你必然首先从使用不同术语的 DB2 工程师中得到数值，然后使用那些不熟悉的数值来链接处理。本文将为你逐步讲解创建 SQL Server 链接服务器的步骤，以帮助解决 SQL Server DBA 和 DB2 DBA 之间的语言鸿沟。

安装 DB2 驱动

因为我们所探讨的是 Microsoft 的产品，所以我们将使用 Microsoft 自己的 DB2 驱动程序来演示这个技术。Microsoft 的 DB2 驱动程序是来自于 SQL Server 2005（或 2000）的特性包，因此它需要在任一与 DB2 通信的服务器上分别下载和安装。

链接服务器对话

一旦安装了驱动程序，你就可以尝试创建链接服务器，同时可以看到下面的屏幕：

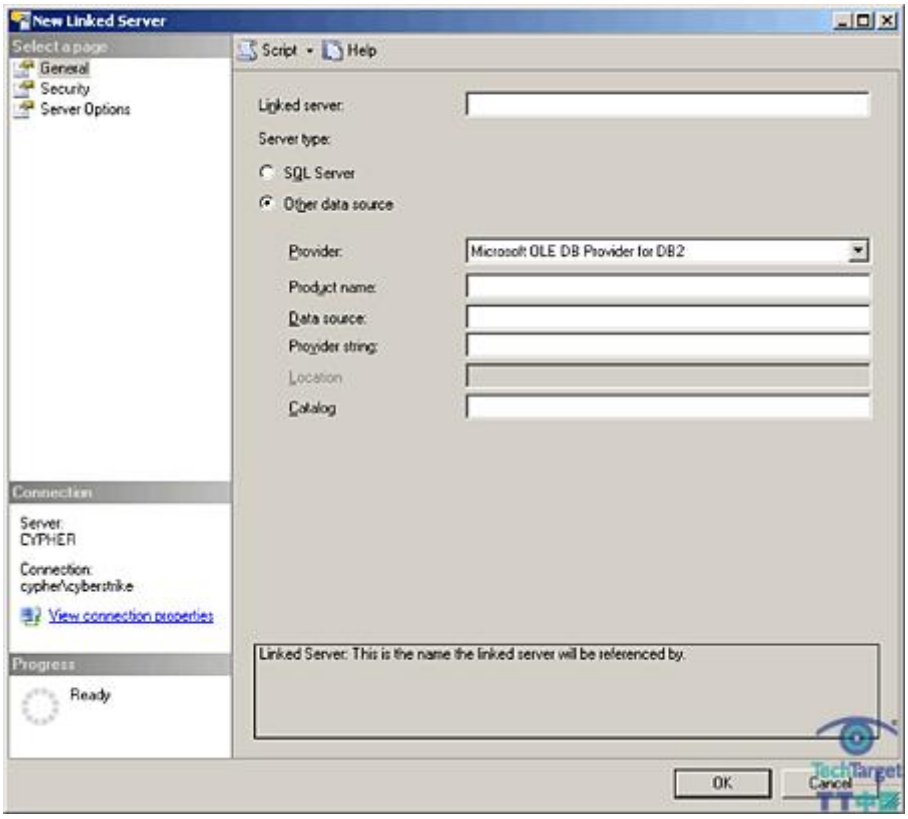


图 A：使用 OLE DB 驱动程序创建链接服务器

Microsoft 并没有提供太多的提示让我们决定应该在这个表单中填写什么样的供应商属性和其它设置。如果你查看的是一个安装了 DB2 驱动的服务器，上面会有一个名为“Microsoft OLE DB Provider for DB2”的程序，该程序有一个 “Data Access Tool”链接。基本上，“Data Access Tool”可以帮你完成链接服务器选项中的供应商字符设置。打开“Data Access Tool”，然后我们将探讨如何为 DB2 的链接服务器设置各种选项。

步骤 1：

右击 DB2 OLE DB UDL，然后选择“New Data Source”。

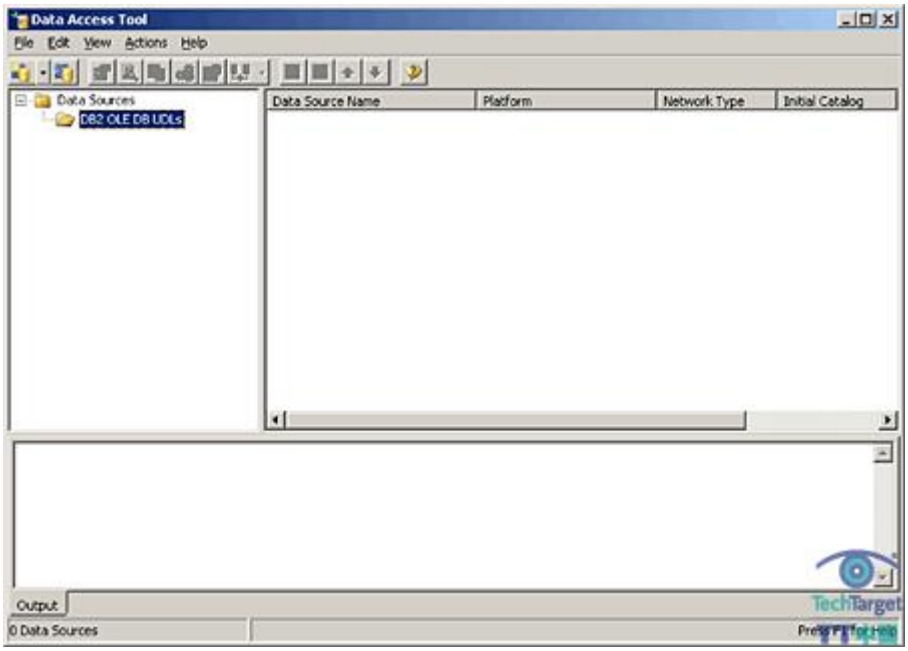


图 B：在 Data Access Tool 中选择“New Data Source”。

步骤 2

选择平台。因为 DB2 有许多支持的平台，你需要与你的 DB2 管理员讨论以找出运行平台。在这个例子中，我们将使用 AS400 DB2 版本。

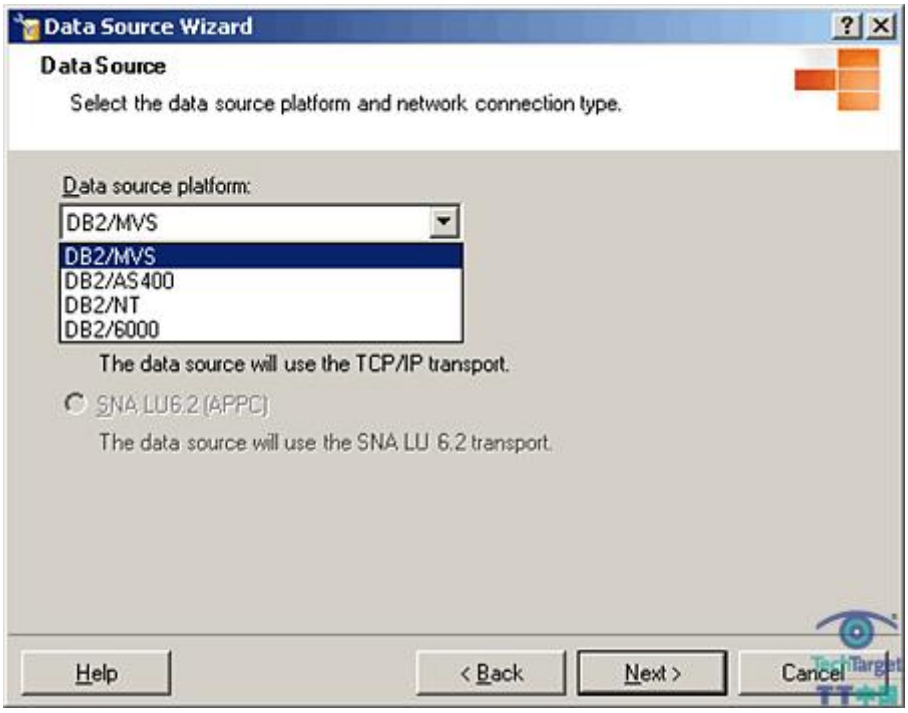


图 C：选择 AS400 DB2 平台创建你的链接服务器。

步骤 3

输入服务器地址。这里最好不要使用 IP 地址，因为它是会随服务器角色变化而改变（相当于集群回滚的 SQL Server）。而 DNS 条目是最佳选择，这里我们将使用 AS400.CYPHER.NET 作为默认端口。你的环境可能有一个自定义的端口——请向你的 DB2 管理员找到该端口。

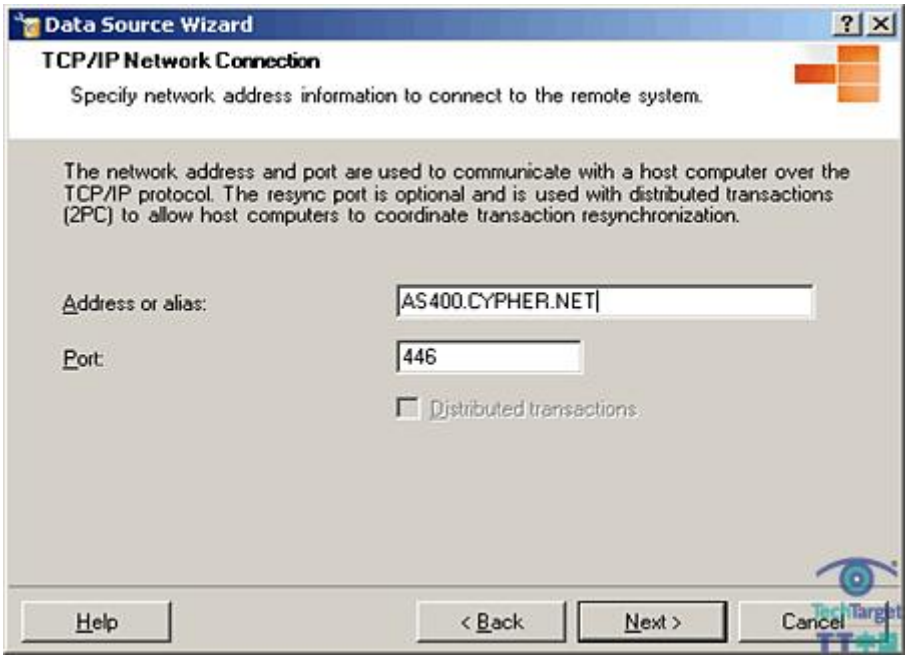


图 D：输入服务器地址 AS400.CYPHER.NET。

步骤 4

下面（图 E）是输入一系列必要的连接信息的对话框。Initial Catalog 选项是很重要的；这是 DB2 的 RDB 名称。一般来说它是初始创建的服务器名，但它也可能是你的 DB2 DBA 所定义的别名。所有其他的选项一般都设为相同的值，该值同时也是你将要尝试访问的库（Library）。DB2 的库可以粗略地看作是 SQL Server 的数据库。

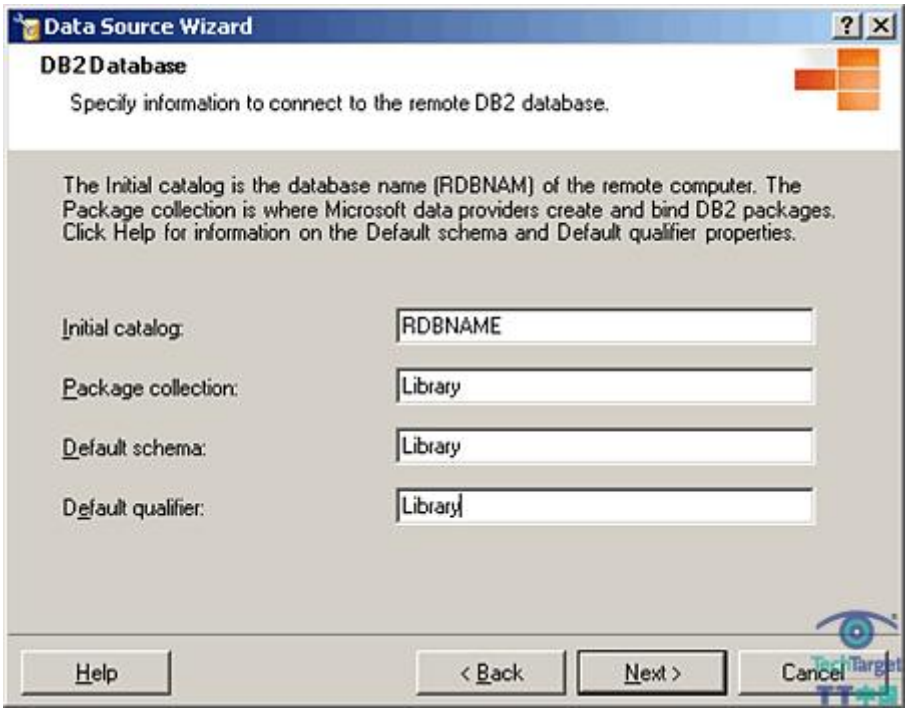


图 E：输入大多数链接服务器连接信息的截图

步骤 5

在下一个对话框（图 F），你可以设置转换的字符集。AS400 使用 EBCDIC 字符集，也就是我们主机的 CCSID 值。（你可以询问你的 DB2 DBA 找到他们系统使用的 CCSID）。PC 代码页是当前 Windows 服务器上使用的代码页。典型地，它会设为 ANSI——US 会使用 Latin1，但你可以根据你所使用的语言来定制使用其它的 Windows 代码页。

一般“Process binary as character”应该置空，这样二进制值才可以作为二进制处理。但有些目标系统会将它转化为字符，这样你也可以传递二进制值。

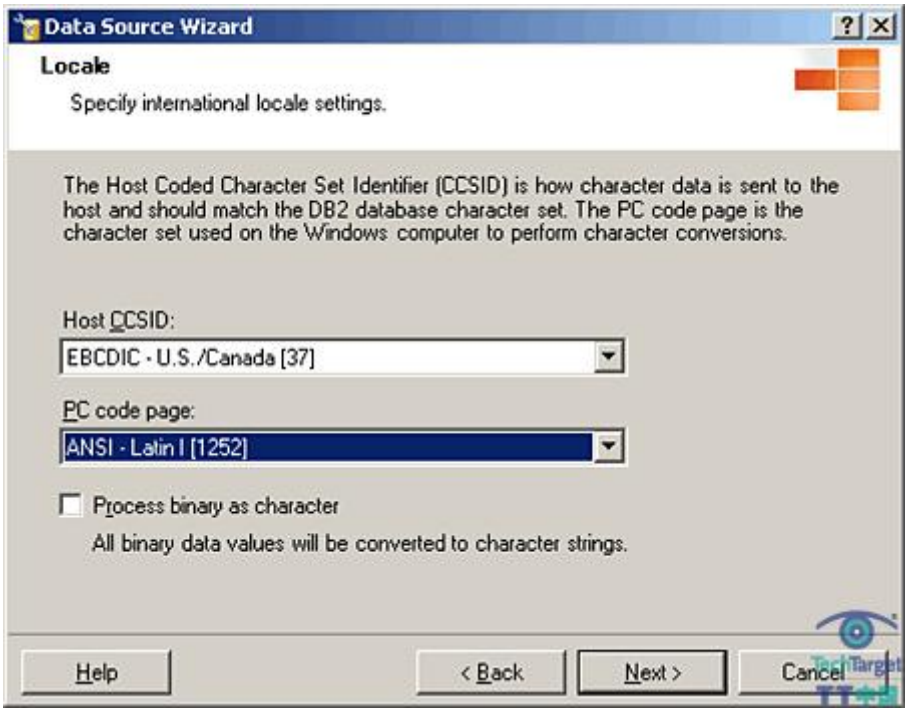


图 F：在大多数环境中，不要选中复框“Process binary as character”。

（作者：Matthew Schroeder 译者：曾少宁／陈柳 来源：TT 中国）

如何创建与 DB2 链接的 SQL Server 服务器（二）

步骤 6

在下一个截图（图 G）中，你可以输入 DB2 DBA 提供给你的用户名/密码。

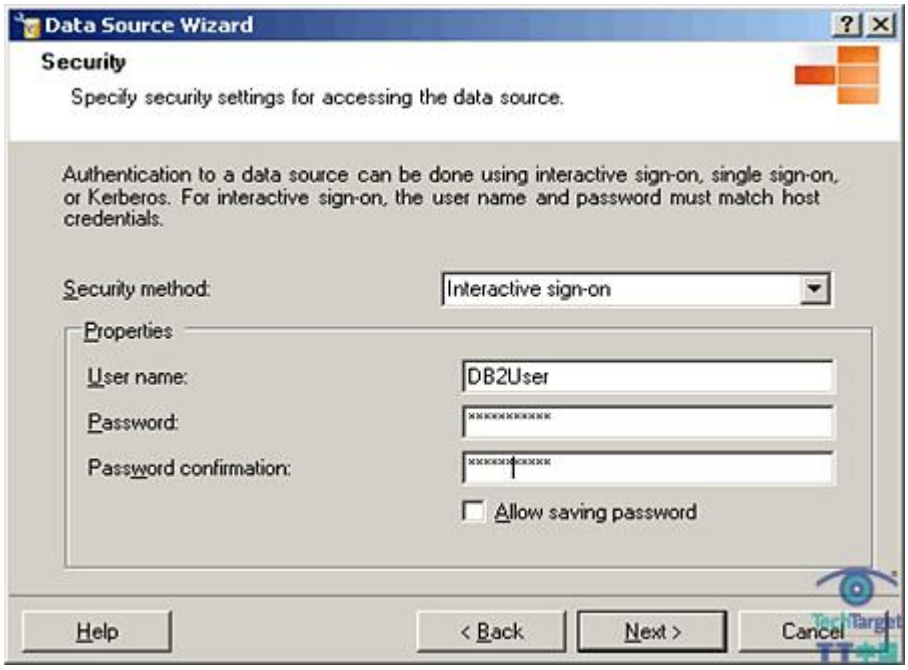


图 G：输入 DB2 DBA 提供的用户或和密码

步骤 7

图 H 中的高级选项应该只设置 DB2 DBA 所提供的信息。在一些环境中，你可能会使用连接池技术来减少系统的负载；而在另一些环境中，这个链接将会是只读的，所以请进行相应的设置。

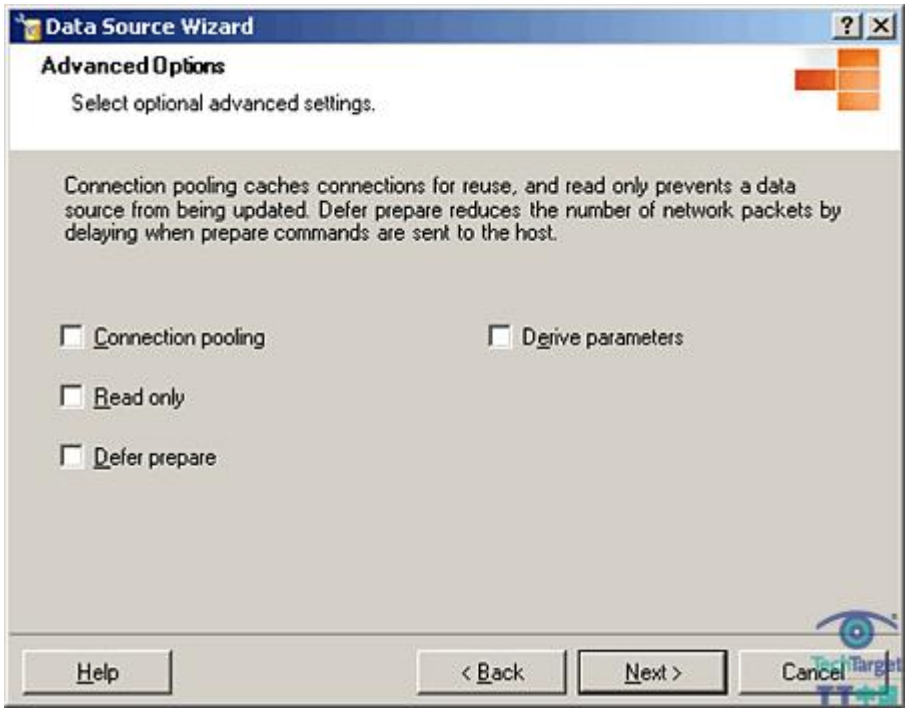


图 H：这些选项应该根据 DB2 DBA 所提供的信息进行设置

步骤 8

在下一个截图中（图 I），你可以选择进行连接，然后它会验证你的设置是否正确。

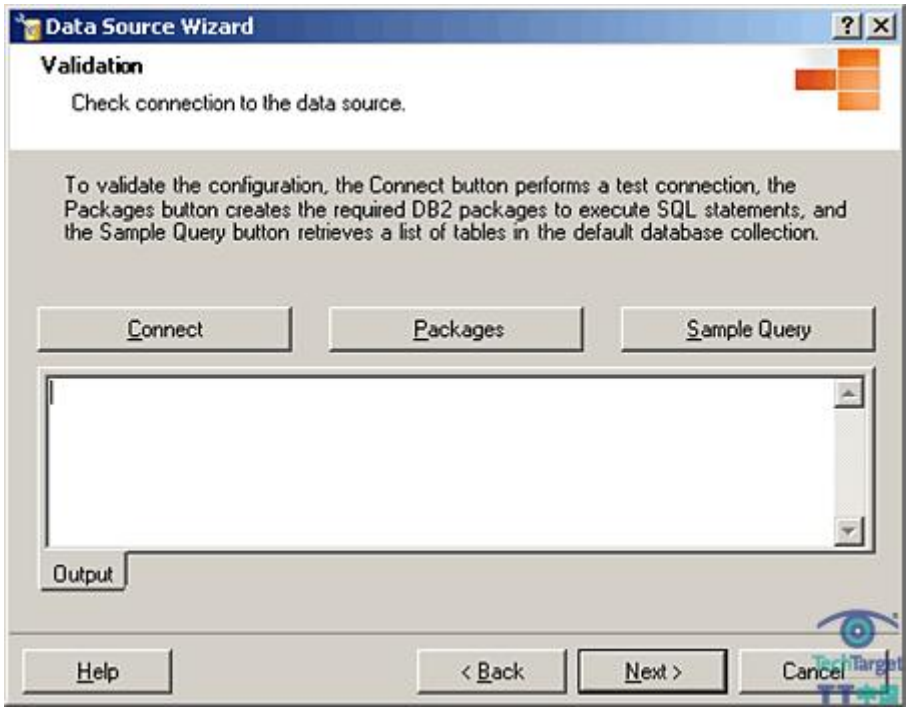


图 I：连接 SQL Server 到 DB2 并验证设置

步骤 9

在最后一个屏幕截图，你将得到你的输出。因为我们将要输入这些信息到一个链接服务器，所以我们仍然保持选中“Universal data link”。

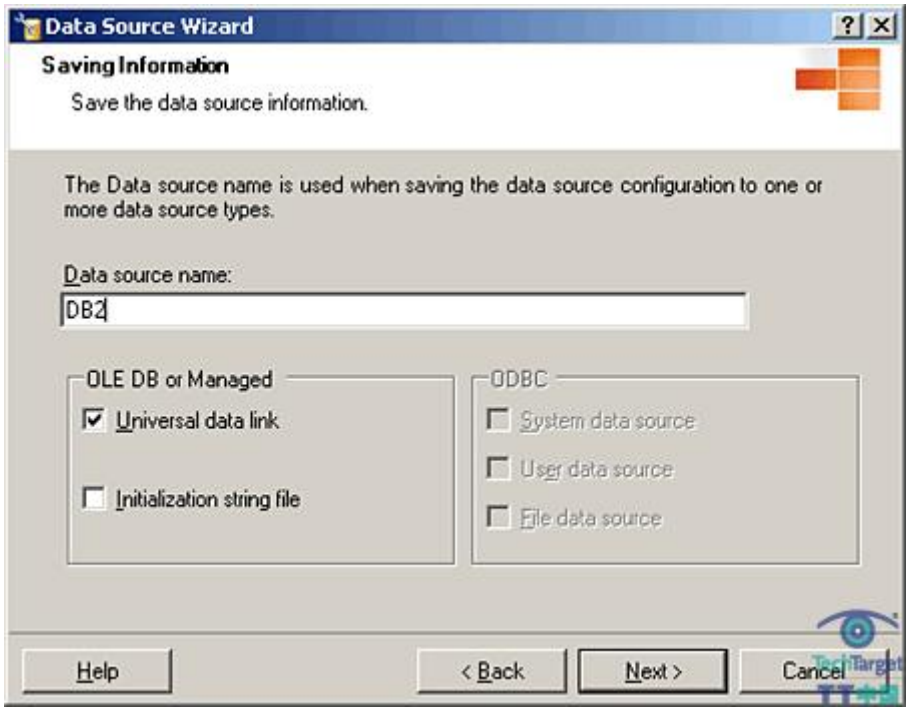


图 J：保持选中“Universal data link”。

验证

图 K 显示你的“Data Access Tool”应该显示的结果。

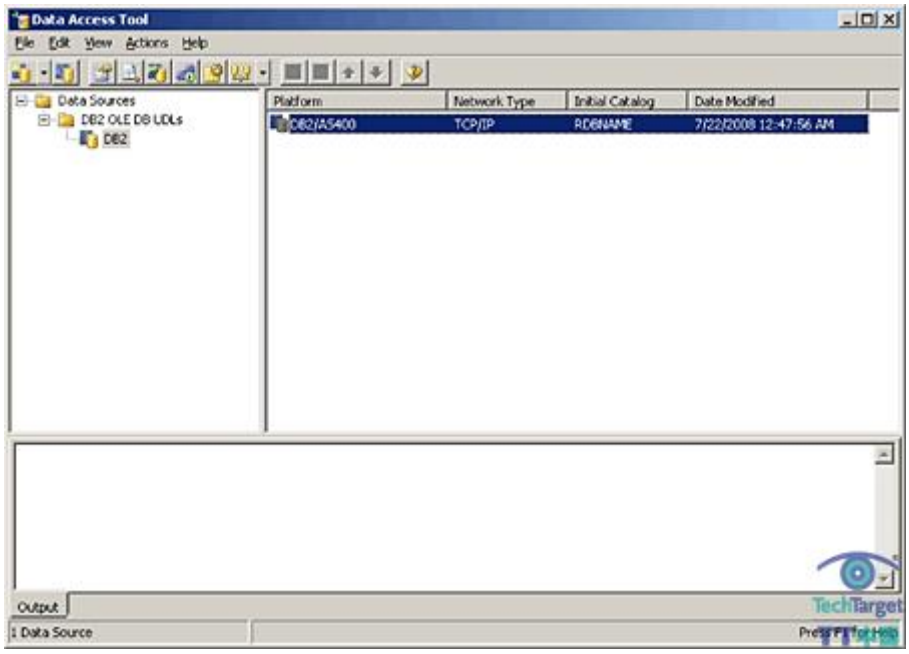


图 K：Data Access Tool 的最后显示

抓取供应商字符

右击 DB2/AS400，点击“Display connection string”。供应商设置字符将会出现在底部窗口。

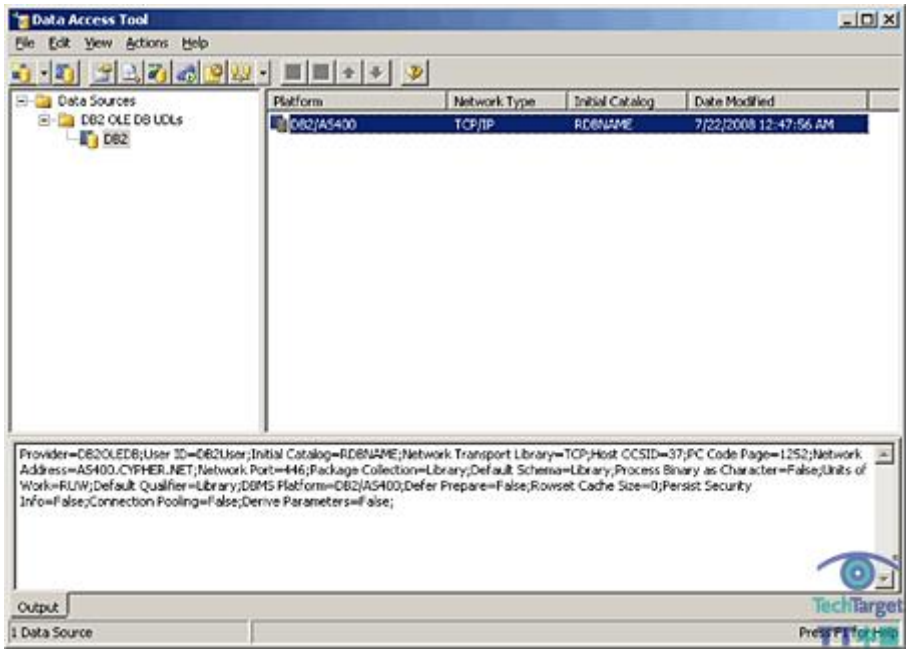


图 L：供应商设置字符出现在底部窗口

建立链接服务器

显然我们需要这些供应商字符信息来创建链接服务器。但是，我们并不想包括用户名或密码，所以我们将我们的字符信息修改为：

1、粘贴修改的供应商字符信息到正确的域中

```
Provider=DB2OLEDB;Initial Catalog=RDBNAME;Network Transport Library=TCP;Host C
CSID=37;PC Code Page=1252;Network Address=AS400.CYPHER.NET;Network Port=446;Pa
ckage Collection=Library;Default Schema=Library;Process Binary as Character=Fa
lse;Units of Work=RUW;Default Qualifier=Library;DBMS Platform=DB2/AS400;Defer
Prepare=False;Rowset Cache Size=0;Persist Security Info=False;Connection Pooli
                    ng=False;Derive Parameters=False;
```

2、定义产品名称——这是连接的标识。

3、数据源/分类应该与大多数系统相匹配，同时它也是服务器的名称。（如图 M）

4、确定你的用户名/密码，如图 N。

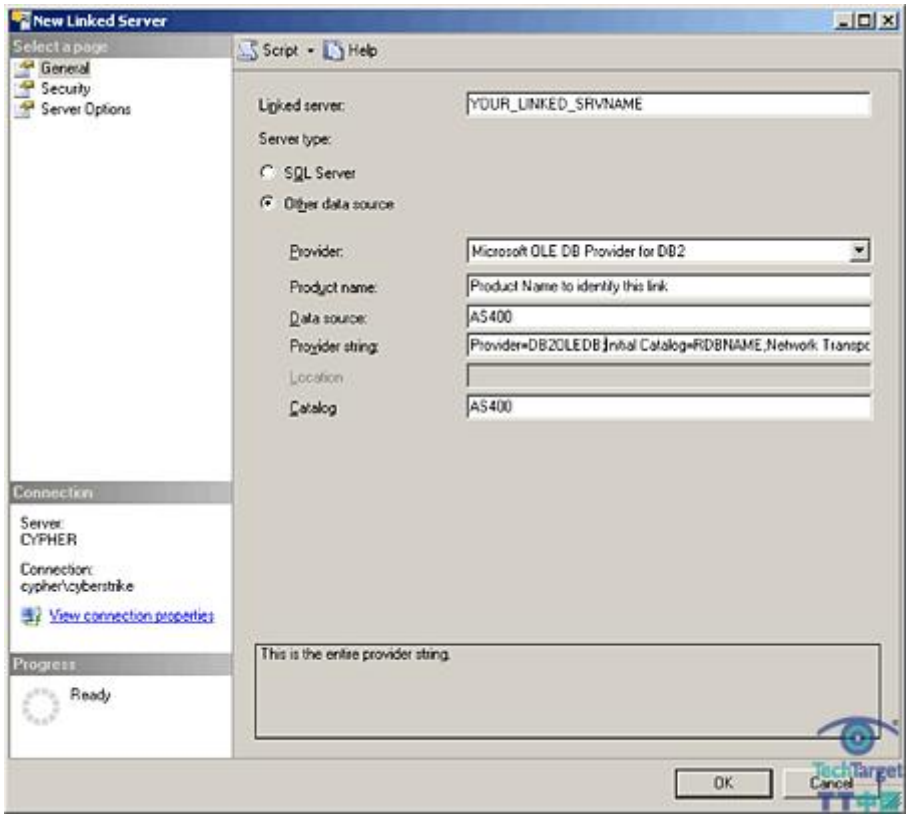


图 M：数据源/分类应该与大多数系统匹配同时作为服务器的名称。

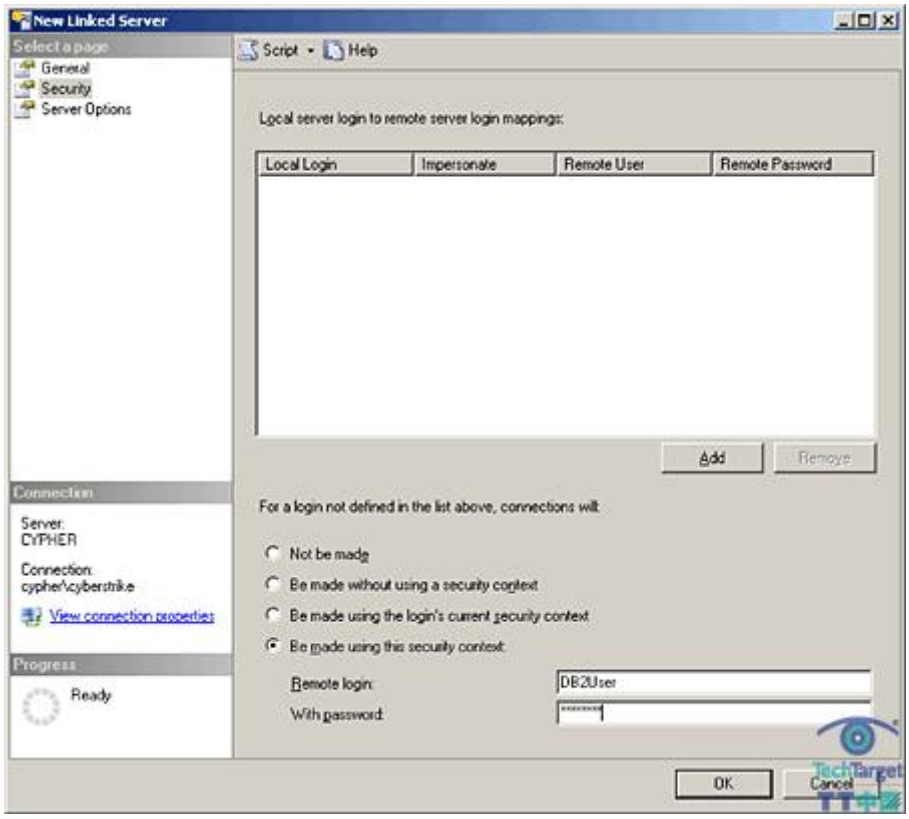


图 N：为你的链接 SQL Server 确定用户名/密码。

注意：

有时，网络连接断开后要求 SQL Server 实例重新启动，以便重新加载 DB2 驱动程序。

我不推荐使用系统名作为 RDB 名称，因为它会根据角色的转换而变化。最好使用一个别名作为 RDB 名称，因为别名可以修改为系统名称，以指向你愿意指向的任何一个服务器。这样，即使当你的链接服务器角色转换时你也不需要修改代码。

如果想要通过一个链接服务器向 DB2 系统作一个 DML 操作（INSERT/UPDATE/DELETE），你需要打开目标对象的日志（通常与 SQL Server 日志相同）。这是因为 SQL Server 能自动地在一个链接服务器上创建 DML 操作的事务，而且这个功能不会被关闭。

现在你已经创建了第一个从 SQL Server 到 DB2 的链接服务器了。我也已经为你提供了一些关于如何增加数据库在角色转换时的灵活性、以及如何向服务器执行 DML 操作的技巧。

*（作者：*Matthew Schroeder *译者：陈柳／曾少宁 来源：TT 中国）*

SQL Server 数据库设计灾难：不该做什么

如果一个外人仔细地查看你的 SQL Server 数据设计时，你会感觉到窘迫吗？有没有可能在你的表中实现一个外键约束呢？你是否在字段中使用了正确的数据类型？你是否按规范定义的方式进行表/字段命名？或者数据库的体系结构是否让人很费解？数据库体系架构师 Brian Walker 根据他多年的 SQL Server 经验，提出了许多用于改进数据库设计和 SQL Server 性能的建议。

一些业务数据库的状况很糟糕。

我曾经在五个星期里检查过三个业务数据库，我至今仍然为我所发现的状况感到震惊。这些都是作为那些能够年产上百万美元公司的关键业务基础服务的 SQL Server 数据库。每天，上百个员工都依靠这些数据库来实现准确性、稳定性和性能。从我个人来说，我不再相信这些数据库能够存储我的 iPod 中的 2，600 首歌曲。

数据库支持它们各自的业务，但它们都受到性能问题的困扰。数据库中有许多问题，但这些业务仍然工作在这些服务器硬件和生产环境 DBA 的问题中。修复这些问题是非常困难的，因为已经有几个应用程序都有适应这些数据库设计缺点的代码。由于额外的开销和功能丢失，业务受到了影响。同时，数据库设计也使 SQL Server 变得脆弱。

过去的问题

那么，我所指的业务数据库到底哪些方面出了问题呢？答案是所有方面，而且非常严重。它们的规范化很差。有一些数据库表没有主键。表之间的许多关系都没有使用外键进行约束。索引的使用也很随意。重要的业务逻辑都堆积在触发器中。许多字段的数据类型是不恰当的。而对于一致性呢？可能参加“美国偶像”第一回合海选的竞争者的造型比这些数据库更有一致性的。

规范化的好处一般是很好理解的，所以我将不会花太多时间进行解释。可以这样说，这些数据库的表都还没有达到 1NF 的要求。表没有主键约束明显违反了 1NF 的规范。而且，有一些表还有包含多种定义值的字段。大多数表的大多数字段都是可为 NULL 的，这也违反了有争议性的 1NF 规范。我个人看来，我认为可为 NULL 的字段一定会给某些情况下的开发人员带来麻烦。

实现外键约束应该是第二特性，而保证数据统一的好处是很重要的原因。这些数据库的一些表包含很多很多的孤立行。这些数据库缺少合适的声明引用完整性（Declarative Referential Integrity，DRI)的结果会给 COBOL 程序员再来巨大的麻烦。其中一个数据库的几个表的状况更令人直冒冷汗的。下面是一个假定的类似的例子。

假定你有一个 Book 表，它有一个外键 AuthorID 字段。在大多数行中，AuthorID 字段包含一个指向 Author 表的外键。在一些行中，AuthorID 字段包含一个指向 Editor 表的外键。而在另一些行中，AuthorID 字段包含指向 Publisher 表的外键。有另一个字段负责指出 AuthorID 字段包含的引用是哪一种类型的。因此，这是不可能实现一个外键约束的。我说得没错吧！

全面的和正确的 DRI 可以生成一些非常有用的 SQL 代码。举例来说，可以为所有的外键生成索引，这可以显著地提高 JOIN 子句调用的性能。生成索可以构成数据库开发的基础，从而我们不需要在性能调优时一个个地添加这些索引。另外一个例子，它还可以生成用于审计或日志目的的 SQL Server 触发器。通常，我认为触发器应该专门用于此类任务，而业务逻辑则由存储过程来完成。

在我所检查的数据库中有许多字段选用了不恰当的 SQL Server 数据类型。它们混淆使用了单字节（标准的）字符串和双字节（国际的）字符串。它们有许许多多固定长度的字符字段，如 20，30，40 和 50。它们有许许多多字段使用了 datetime 数据类型，而其实 smalldatetime 数据类型应该更符合它们的要求。这些数据类型选择浪费了存储空间并且降低了读/写性能。

保持一致性

在这些数据库发现的技术问题让我很不安，而缺少足够的一致性让我更难受。不一致的最主要方面是命名规范。10 个数据库开发人员就会有至少 10 个“最佳的”命名规范。当然，我也有我自己习惯的命名规范，但这里有一个忠告给那些还想创建一个规范的人。这就是：定义一个命名规范，然后再使用它。每一个地方，每时每刻都应该这样做。如果一个数据库中有许多的命名规范是非常糟糕的。

当你在定义一个命名规范时，你要确保考虑到了所有方面。要确定表名的单数或复数。要确定好缩写、大写、特殊字符和前/后缀的使用方法。要为所有对象定义一个命名规范。包括表、字段（主键、外键等）、存储过程、方法、视图、触发器、索引、主键约束、外键约束、检查约束和默认值。任何遗漏都可能造成混乱。

表结构的一致性和可推测性可以带来许多的好处。比如，对旧数据的存档和清理会受到表结构的很大影响。这可能是你处理关系数据库时可能遇到的最复杂的事了。正确的做法是先选择一个父记录行（或一组父记录行），然后在派生表中复制/删除它和每一个相关的记录行。如果所有的表都有一致的结构，你就可以通过简单地调用存储过程来实现存档和清理。

我检查的这些数据库同时使用了几个不同的表结构。主键字段是不可推测的。外键字段也是不可以推测。审计字段也一样不可以推测。同时 IDENTITY 属性的使用也是不可推测的。因此，进行存档和清理需要为每一个表仔细编写特定的 SQL 代码来实现，这给维护带来巨大的工作量。在我个人看来，我认为表结构的不一致可能是许多业务数据库的最严重的设计缺点。

请阅读第二部分《SQL Server 数据库设计灾难：它是如何开始的》，Walker 会为你解释业务数据库糟糕的现状是怎么样的，以及你应该如何避免这些麻烦。

（作者：Brian Walker 译者：陈柳／曾少宁 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。

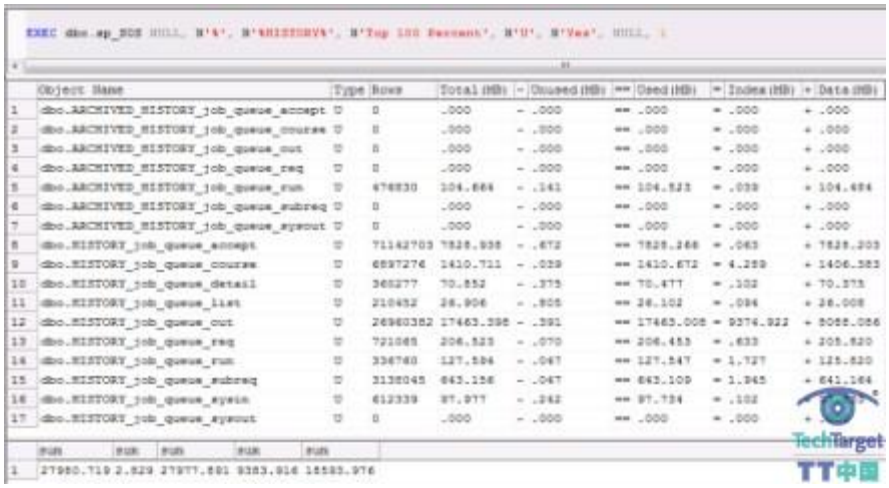


图 1：sp_SOS 显示了一个 SQL Server 2000 用户数据库的分组用户表，它们有相似的名称和统计空间大小。

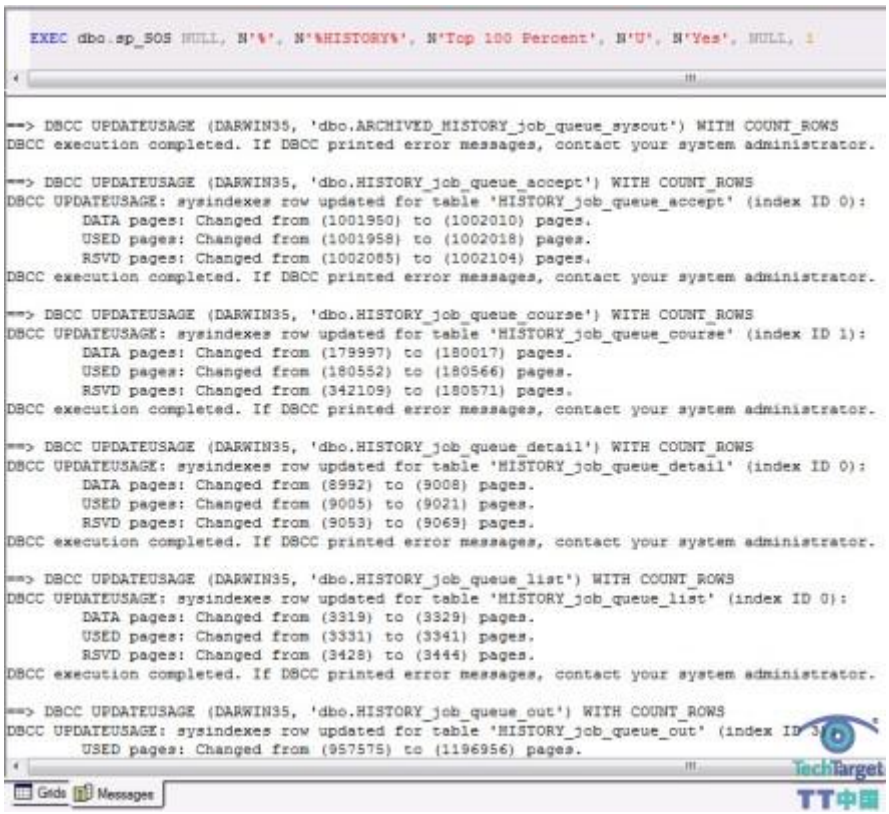


图 2：SQL Server 2000 Query Analyzer 显示 sp_SOS 代码段以及“DBCC UPDATEUSAGE”命令的详细结果。这时，大多数的对象需要更新它们的大小信息。

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。

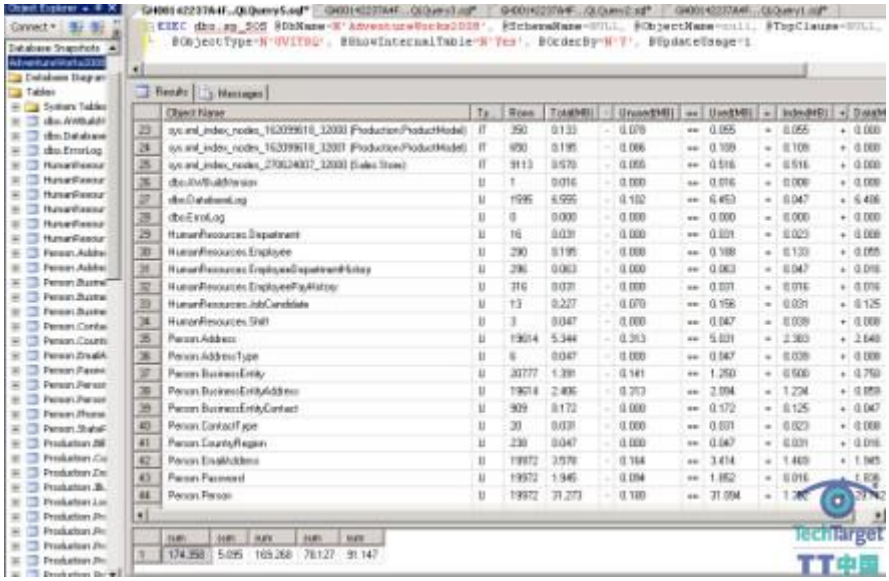


图 3：兼容 SQL Server 2008 CTP 的 sp_SOS 在 AdventureWorks2008 数据库运行后显示根据对象类型排序的所有对象列表。

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：Brian Walker 译者：陈柳／曾少宁 来源：TT 中国）

如何使用向导设置 SQL Server 2005 日志传送（一）

在 SQL Server2005 中建立日志传送时，你可以用许多种方法来实现主服务器和副服务器之间的最优配置。除了一些最佳实践方法之外，我们还要从头到尾遵循 SQL Server MVP Hilary Cotter 日志传送安装过程，其中包括选择合适的数据库恢复模式、事务日志路径和副数据库设置。

日志传送是 SQL Server 内置的一项高可用性技术，它在本质上是一个持续备份和恢复操作。日志传送先从主（源）服务器上拷贝数据库备份和事务日志备份，然后在一个或多个副（目的地）服务器上重建这些数据库和事务日志备份。数据库和事务日志会以备用或无恢复模式在副服务器上重建，这样后续事务日志在主服务器上备份然后传送（或拷贝）到副服务器并应用。

- 备用模式允许用户访问重建的数据库，但不允许对它们做任何修变，只允许将来再将日志备份恢复到数据库。
- 无恢复模式不允许用户访问数据库，但是允许将来再事务日志备份恢复到数据库。
- 日志传送可以使主（源）和副或备用服务器在每个数据库上实现同步。

下面是一些需要注意的关于日志传送的重要说明：

- 除了只读、tempdb、模型和任何不在完全恢复模式或简单恢复模式的数据库，你可以传送所有的数据库。
- 在碎片整理/索引/重索引操作中，事务日志的备份文件可能变得非常大。为了减小事务日志的大小，你可以选择使用完全和批量日志恢复模式（批量日志操作能减少索引操作日志）。

- 某些操作会“中断”日志传送，比如将日志恢复模式从完全或批量模式改为简单模式。
- 日志传送操作在备份操作中暂停。
- 延迟时间（未同步的主服务器到副服务器的时间）可以有所不同；比如在日志传送过程中，你会每 10 分钟备份一次事务日志。而在 1:00a.m 开始的事务日志备份可能需要 10 分钟才能完成。可能还需要再花费 5 分钟的时间将备份拷贝到目的服务器。这就意味着你需要承担 15 分钟丢失数据的风险。假设你的服务器在 1：09 a.m 时死机了将发生什么呢？你的备份将无法生成或者拷贝到副/备用服务器上，而你的副服务器上的数据将可能是只到 12:50 a.m 的。
- 所有依赖项都必须先在副服务器上准备好（比如，登陆、DTS 或者 SSIS 包、SQL Server 任务和其它外部依赖）。
- 客户端必须手动地重定向到备用服务器。
- 应用必须知道数据库使用了日志传送——如果主服务器掉线，数据将在客户端排队等待，直到备用服务器重新连线。最佳实践做法就是手动进行重定向；否则，应用将可能过早或错误地重定向到备用服务器。

日志传送很受欢迎，这是因为它是一项很好理解的技术，而且并没有任何实际的距离限制，它可以在因特网上运用，目的数据库可以被访问但只限于只读方式，同时在备用 SQL Server 上发生错误后的 30 天内不需要使用授权。但是，备用服务器需要一个 OS 使用授权。

在本文中，我们将探讨如何使用 SQL Server 2005 包含的安装向导来在 SQL Server 上设置日志传送的。

*（作者：*Hilary Cotter *译者：陈柳／曾少宁 来源：TT 中国）*

如何使用向导设置 SQL Server 2005 日志传送（二）

使用日志传送向导

右键点击数据库，然后选择“Tasks”-“Ship Transaction Logs”，打开 SQL Server 2005 的日志传送向导，如图 1。

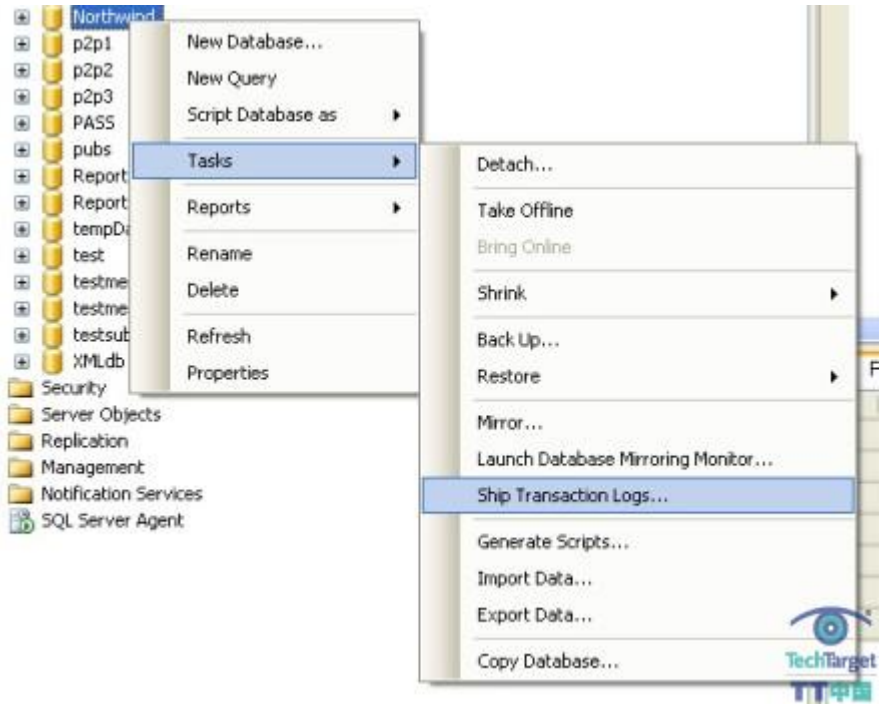


图 1：打开日志传送向导。

你可能会看到图 2 中显示的对话框。SQL Server 所提供给你的是这个数据库并不是在一个完整或批量日志恢复模式上。

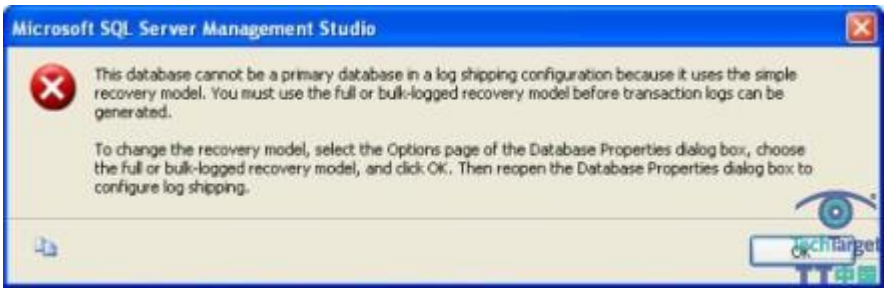


图 2：对话显示数据库是一个简单的恢复模式。

若要改变你的数据库的恢复模式，先右键点击数据库，选择“Properties”，接着在“Options”栏中点击“Recovery Model”下拉列表，然后选择“Full”或“Bulk_logged”恢复模式。如图 3 所示。最后点击“OK”。

注意：如果你选择了这些恢复模式并且你对你的数据库进行备份或者已经备份了数据库，你将需要安排数据库备份调度来维护你的事务日志。日志传送向导将为你配置此项功能。

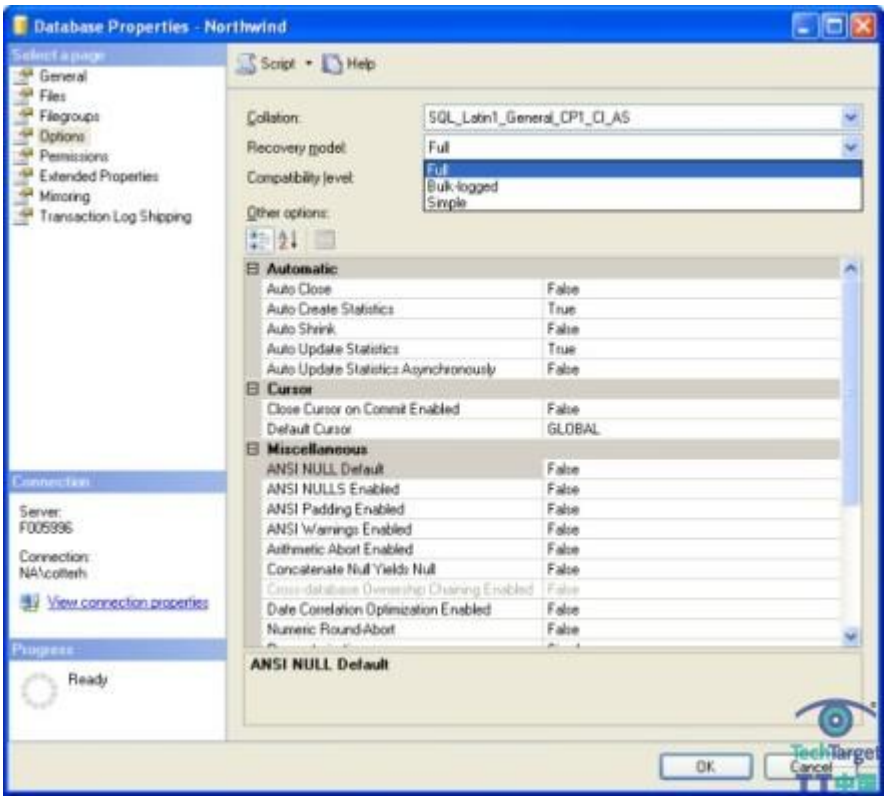


图 3：更改数据库恢复模式。

当为你的数据库设置了正确的恢复模式后，右键点击数据库，然后再次选择“Ship Transaction Logs”。你将得到如图 4 所显示的对话框。

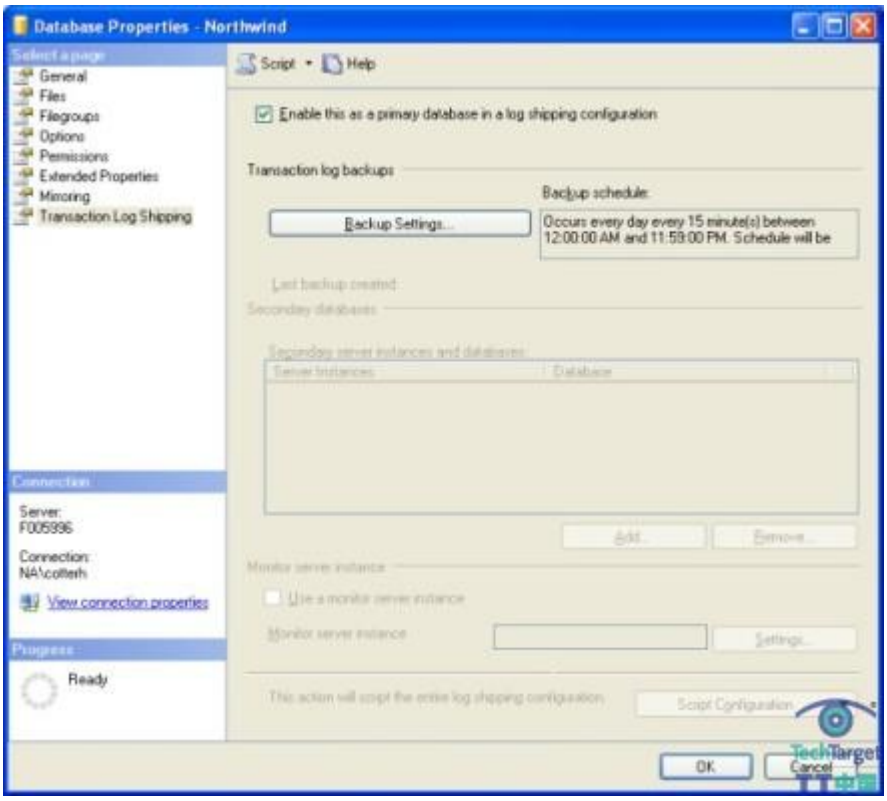


图 4：配置日志传送。

注意：在这个图中，我选中了复选框来将它作为日志传送配置中的主数据库。请务必选择这个复选框。点击“Backup Settings”，将显示如图 5 所示的“Transaction Log Backup Settings”窗口。

备份事务日志可以有两种方式：

- 网络路径
- 本地路径

如果你是备份到网络路径，你将需要较多的事务日志备份时间，但可以减少在主服务器上的空间需求。网络路径必须能到达你想要存储事务日志备份的副服务器的位置。大多数 DBA 使用网络共享来备份文件，因为他们希望在副服务器上的事务日志备份文件，以此应对主服务器出现死机的状况。

本地路径选项将事务日志备份在主服务器的本地路径上。请确认事务日志备份不要备份在与数据库数据文件或日志文件存放的同一物理驱动器上。如果它们被备份到同一物理驱动器上，将导致 I/O 抢占和整体 SQL Server 性能降低。

图 6 显示一个完整的 Transaction Log Backup Settings 窗口。注意在共用名之后有一个美元符号。这个符号的作用是对用户隐藏共享，并且只有知道共用名的用户才能够进行访问。这是一个良好的安全实践。

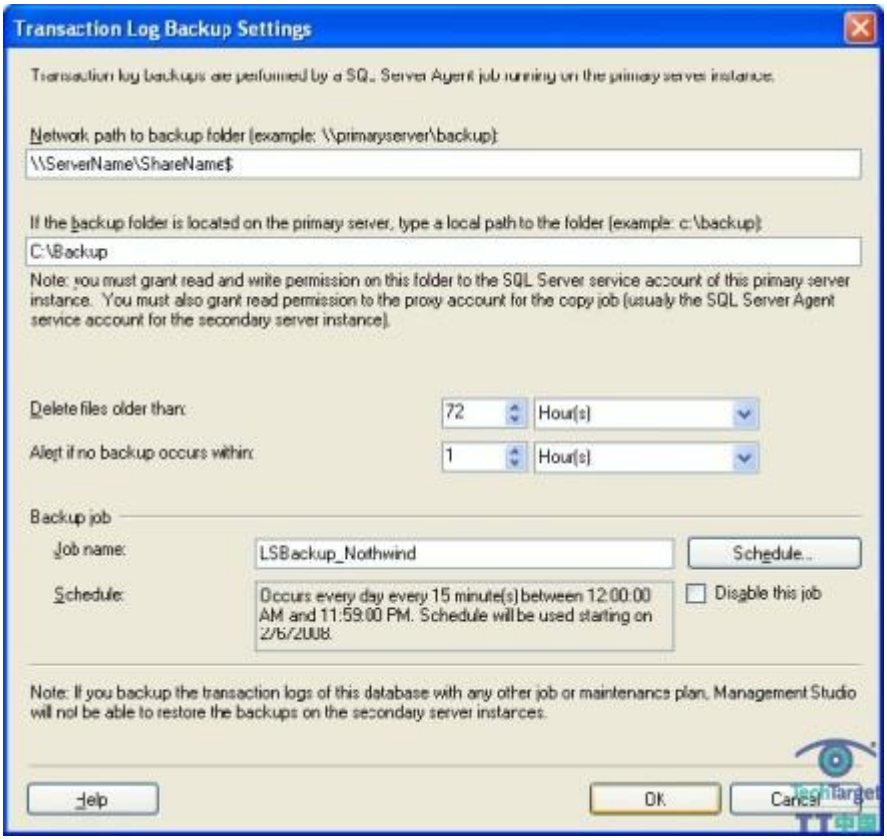


图 6：Transaction Log Backup Settings 显示一个网络路径隐藏共享。

其中有一个选项是设置保留事务日志备份时间长度，而另一个选项是设置在特定时期没有备份时的提醒临界值。我发现了在默认情况下事务日志备份保留三天是足够的。因此，我通常建立一个相对较小的警报，如 20 分钟，但是当发生数据库备份时，就不再有

日志备份，因此如果你的数据库备份需要一个多小时，那么你将会不断接收到提醒。所以要设置一个适合你的环境的值。并且请记住：过分频繁的警示将导致你忽略关键的警示。

在你设置了事务日志备份位置、保留期间、警报时间和时间表，点击“OK”。接着你将返回“Database Properties”对话框，如图 4 所示。点击“Next”按钮，进入副“Database Settings”对话框，如图 7 所示。

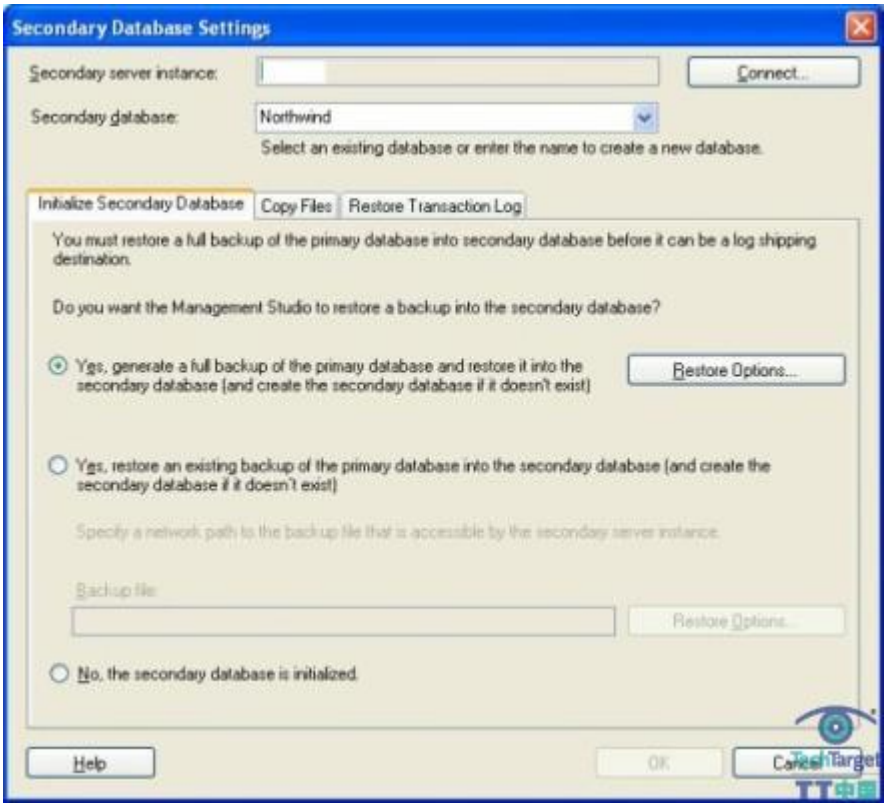


图 7：副数据库设置。

（作者：Hilary Cotter 译者：陈柳／曾少宁 来源：TT 中国）

如何使用向导设置 SQL Server 2005 日志传送（三）

在此对话框中，你可以选择副服务器（你将把事务日志拷贝到这个服务器上）以及副数据库（日志所要传送到的数据库名称）。

注意下面这些针对 Initialize Secondary Database 的选项：

- Yes，generate a full backup of the primary and restore it into the secondary database.这个选项创建了一个你想要进行日志传送的数据库的备份，然后将它恢复到副服务器上。
- Yes，restore an existing backup of the primary database.如果你想要使用一个之前备份的数据库，就可以使用这个选项。这里有一个选项用于选择路径和备份名称。
- No，the secondary database is initialized.Use this option if:在以下情况使用这个选项：

1、你想要进行日志传送的数据库备份已经恢复到副数据库了。

2、主数据库已经是完整或批量日志恢复模式。

3、从备份发生后主数据库上没有再做任何事务日志备份，或者它们已经完成，并恢复到副数据库上了。

4、副数据库已经使用无恢复选项重建好了。

一旦你已经配置了最适合你的选项，点击“Copy Files”标签，如图 8 所示。

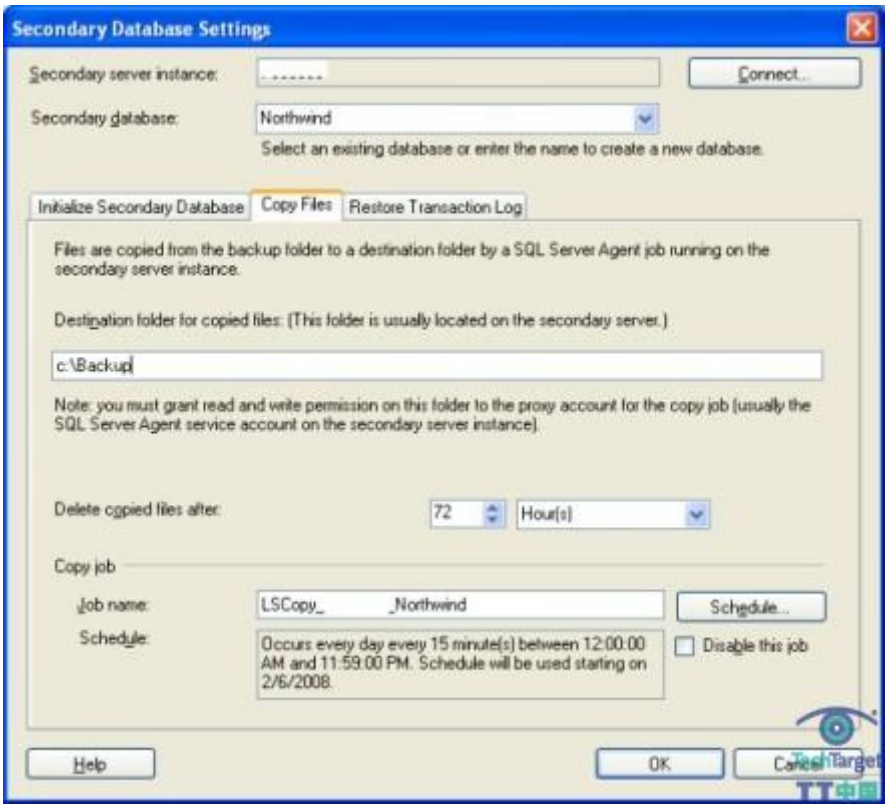


图 8：Copy Files 对话框。

在“Copy Files”标签中选择你从主服务器到副服务器上拷贝文件的存放位置。如果你在Transaction Log Backup Settings（如图 1 所示）中选择一个网络路径，那么网络途径必须映射到副服务器的物理位置上，同时你必须在此处输入路径（如，如果\\ServerName\ShareName$是本地路径C:\Backup的共用名，那么就在此处输入这个路径）。

你也可以使用事务日志备份存储在主服务器上的网络路径。选择你想要保留事务日志备份的时间长度——当然，必须认识到，保留时间可能与你在“Transaction Log Backup Settings”对话框（图 1）中的设置冲突。

你同样还可以选择你希望日志拷贝到副服务器上的频率。

一旦你配置好了拷贝的事务日志备份文件的选项，点击“Restore Transaction Log ”标签，如图 9 所示。

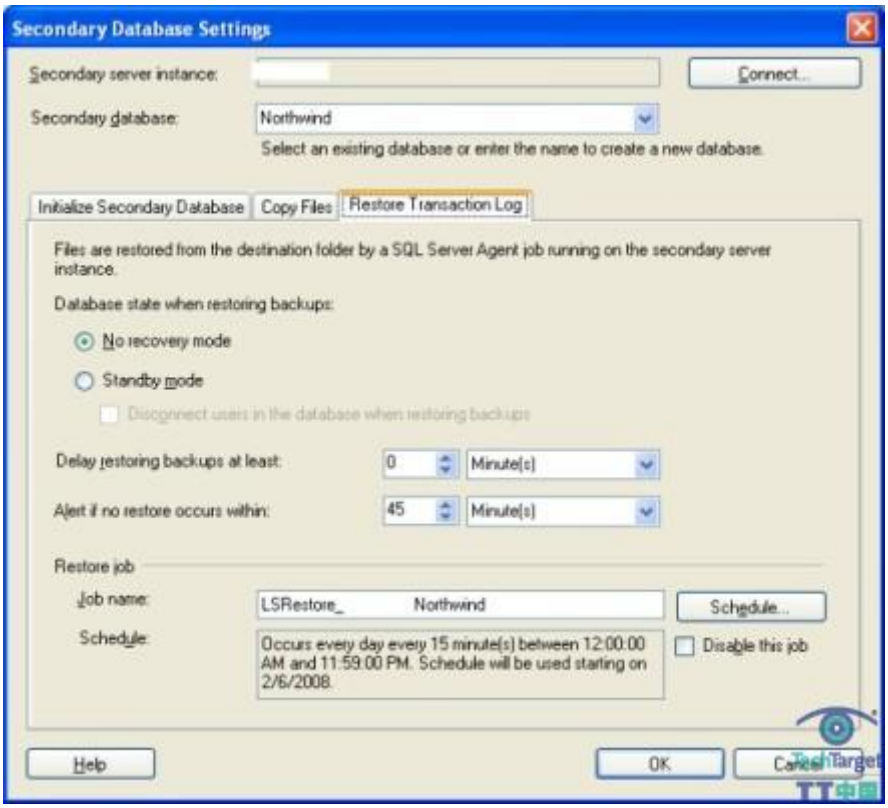


图 9：Restore Transaction Log 标签

数据库恢复状态选项：

- 无恢复模式——这是一个默认选项。在这个选项中，目的地数据库将无法使用。
- 备用模式——使用这个选项，在下一个事务日志备份应用之前，目的地数据库仅仅是只读形式。在事务日志被应用后，数据库重新将返回只读模式。这样就只允许只读访问并且用户将无法对数据库作任何修改（如，创建索引），而且在下一个数据库备份被应用时，他们将被断开。

还有一个选项是通过设置一组小时或分钟数来延迟恢复事务日志备份。有些企业想要保持他们的备用服务器与他们的源服务器几个小时不同步。

在默认情况下，如果事务日志在定义的“Alert if no restore occurs within”间隔的时间内没有恢复，那么就会有一个告警信息发出。通常情况下，这个警告是发生在主服务器进行备份操作的时候。

你还有一个选项就是设置事务日志恢复发生的频率。这些设置是在 Restore 任务对话窗口配置的。

你设置好后，点击“OK”，然后你将看到“Database Properties”对话框（如图 10 所示）的“Transaction Log Shipping”对话框。

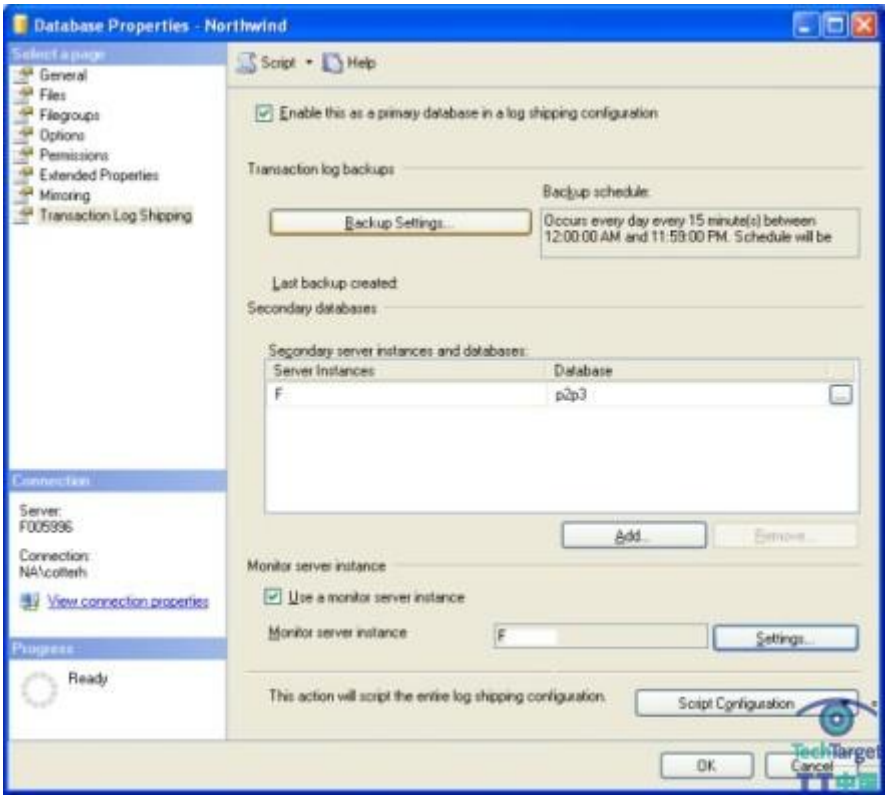


图 10：Transaction Log Shipping 标签

在此对话框中，注意我们是如何配置服务器 F 作为我们的副服务器，同时该数据库 p2p3 是我们日志传送 Northwind 数据库的目地数据库。你也可以日志传送到第二个副/备用服务器——这个可能是你的 DR 站点的另外一个副服务器。

对于任何具备大量日志传输数据库的企业，你可能都想要创建一个监控服务器。选择“Use a monitor server instance”（图 10）然后点击设置按钮来配置一个 Log Shipping Monitor 服务器，就可以创建一个检测服务器。如图 11 所示。

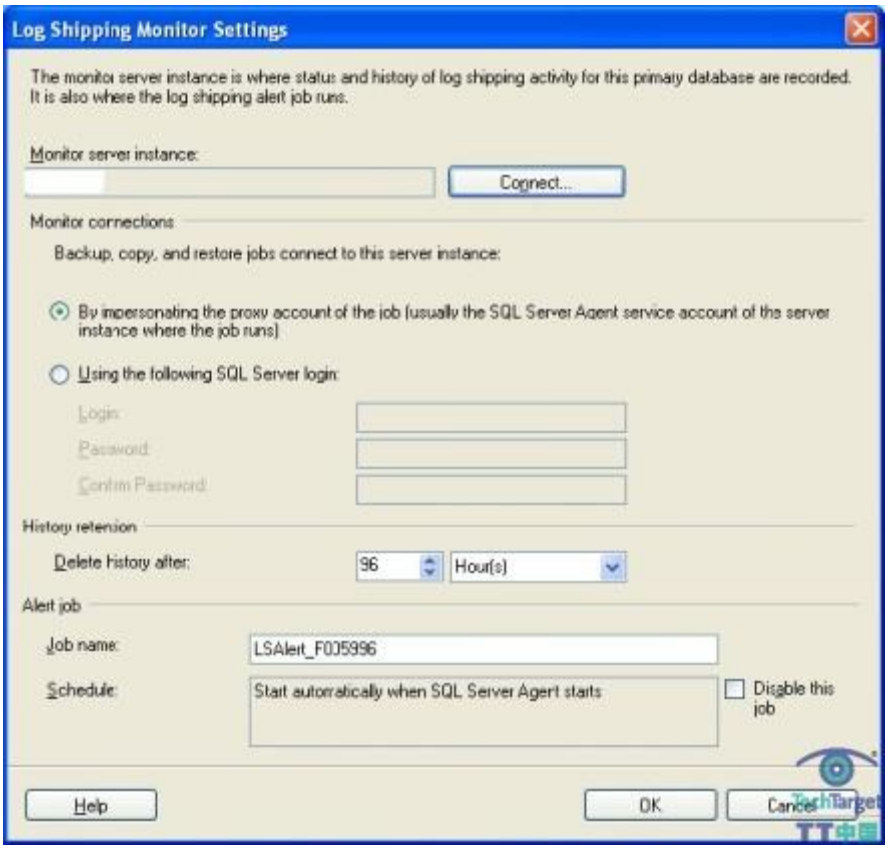


图 11：创建一个监控服务器。

使用连接按钮，你可以连接到你想用作监控的服务器。定义连接方式，或者通过 Windows 认证，或者通过 SQL Server 登录。你可以定义工作历史保留时间，但通常使用默认值都是一个不错的选择，然后定义如何发送警报。默认选项（当 SQL Server Agent 启

动时自动启动）是一个不错的选择，因为这时警报几乎都是实时的。你可以选择一个每小时提醒或者选择一个任意时间间隔。

总结

这样我们就完成了使用 SQL Server 2005 的日志传送的介绍。这个向导有许多的选择并且对于新手而言可能有些困惑。但是，在大多数情况下默认值都是最优选项，并且我已经指出了在什么情况下最好选择非默认选项。

（作者：Hilary Cotter 译者：陈柳／曾少宁 来源：TT 中国）

用存储过程检查 SQL Server 数据库和日志文件大小（一）

了解 SQL Server 数据库的大小是许多 DBA 的职责之一，而这个职责你可以轻松通过存储过程 sp_SDS 来完成。sp_SDS 不仅能确定“SQL 数据库空间”，而且它还能监测数据库的增长，提醒 DBA 关于数据或日志文件的增长，执行事务日志备份，甚至提供详细的文件级明细表，这样 DBA 可以压缩文件以获取最大空余空间。

本文介绍了完整的 sp_SDS 及其算法。相比文章《存储过程 sp_SOS》，本文将进一步阐述如何查询数据库对象的大小，包括 SQL Server 表。

列表 1：sp_SDS 的 T-SQL 定义。

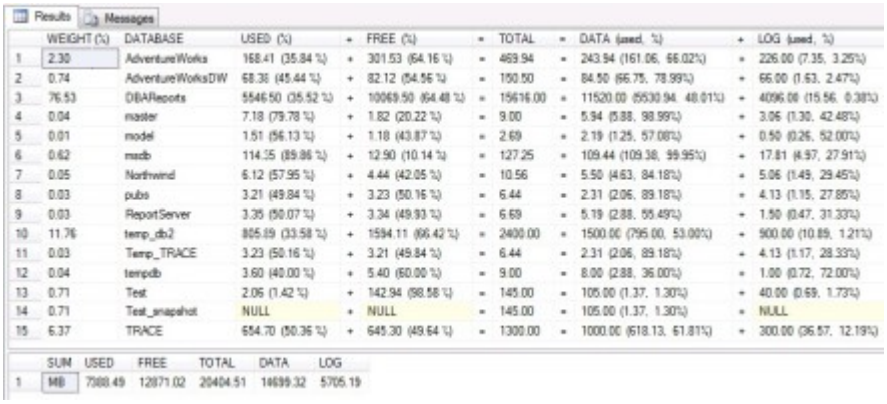
现在，我将解释这个 SQL Server 存储过程是如何查找数据库的大小以及是如何使用它的。

sp_SDS 大部分输入变量是相当明确的。我们需要查询大小的数据库是 @TargetDatabase。@Level 指的是报告详细程度，可以是数据库级或者单个数据库文件级的。它的默认值是数据库级的。它会为每一个数据库显示一个汇总。一个比特值是 @UpdateUsage。它的默认值是 0，表示我们并不希望在 SQL Server 2005 和 SQL Server 2008 中运行“DBCC UPDATEUSAGE”。在 SQL2000 中，sysindexes 表中的值有时候不能及时更新。因此，为了获得准确的读取值，我们需要运行 DBCC 命令。@Unit 参数表明报告测量单位应该是什么，即 KB、MB 或 GB。如果没有指定，那么使用的测量单位是兆字节的。

图 1，你可以看到列表 2 代码执行的屏幕截图。默认情况下，该报告是数据库级的汇总，每一行代表一个数据库。

```
USE master;
EXEC dbo.sp_SDS;
```

列表 2：Sp_SDS 不需要参数的最简单的执行形式（所有输入变量都是 null）。



	WEIGHT (%)	DATABASE	USED (%)	FREE (%)	TOTAL	DATA (used, %)	LOG (used, %)
1	2.30	AdventureWorks	168.41 (35.84 %)	301.53 (64.16 %)	469.94	243.94 (161.06, 66.02%)	226.00 (7.35, 3.25%)
2	0.74	AdventureWorksDW	68.36 (45.44 %)	82.12 (54.56 %)	150.50	84.50 (66.75, 78.99%)	66.00 (1.63, 2.47%)
3	76.53	DBAReports	5546.50 (35.52 %)	10069.50 (64.48 %)	15616.00	11520.00 (5530.94, 48.01%)	4096.00 (15.56, 0.38%)
4	0.04	master	7.18 (79.78 %)	1.82 (20.22 %)	9.00	5.94 (5.88, 98.99%)	3.06 (1.30, 42.48%)
5	0.01	model	1.51 (56.13 %)	1.18 (43.87 %)	2.69	2.19 (1.25, 57.08%)	0.50 (0.26, 52.00%)
6	0.62	msdb	114.35 (89.86 %)	12.90 (10.14 %)	127.25	109.44 (109.38, 99.95%)	17.81 (4.97, 27.91%)
7	0.05	Northwind	6.12 (57.95 %)	4.44 (42.05 %)	10.56	5.50 (4.63, 84.18%)	5.06 (1.49, 29.45%)
8	0.03	pubs	3.21 (49.84 %)	3.23 (50.16 %)	6.44	2.31 (2.06, 89.18%)	4.13 (1.15, 27.85%)
9	0.03	ReportServer	3.35 (50.07 %)	3.34 (49.93 %)	6.69	5.19 (2.88, 55.49%)	1.50 (0.47, 31.33%)
10	11.76	temp_db2	805.69 (33.58 %)	1594.11 (66.42 %)	2400.00	1500.00 (795.00, 53.00%)	900.00 (10.89, 1.21%)
11	0.03	Temp_TRACE	3.23 (50.16 %)	3.21 (49.84 %)	6.44	2.31 (2.06, 89.18%)	4.13 (1.17, 28.33%)
12	0.04	tempdb	3.60 (40.00 %)	5.40 (60.00 %)	9.00	8.00 (2.88, 36.00%)	1.00 (0.72, 72.00%)
13	0.71	Test	2.06 (1.42 %)	142.94 (98.58 %)	145.00	105.00 (1.37, 1.30%)	40.00 (0.69, 1.73%)
14	0.71	Test_snapshot	NULL	NULL	145.00	105.00 (1.37, 1.30%)	NULL
15	6.37	TRACE	654.70 (50.36 %)	645.30 (49.64 %)	1300.00	1000.00 (618.13, 61.81%)	300.00 (36.57, 12.19%)

	SUM	USED	FREE	TOTAL	DATA	LOG
1	MB	7388.49	12871.02	20404.51	14699.32	5705.19

图 1：在测试 SQL Sever 2005 上运行没有输入参数的 sp_SDS 的结果。它生成一个数据库级的以兆字节形式报告的汇总。

第一列“Weight (%)”计算的是所指定的数据库所占总数据库大小的百分比。比如，AdventureWorks 全部为 469.94MB，除以总数的 20，404.51 MB，就是 0.023（即 2.3%）。

根据报告，我们很容易可以看出 DBAReports 占用了大多数的数据库空间（大约是 77%）。这一栏中的数据可以让 DBA 很容易看到数据库大概的空间使用情况。而剩下的值相关的列被安排在一个类似于公式的结构中。

正如我在上一篇关于 sp_SOS 文章中提到的，我喜欢用算术公式来表示一些看起来很费解的关系。因此，“TOTAL”列是左边和右边的相加结果。这两个等式位于不同方向的中心。一个显示被占用的空间和剩余的自由空间。另外一个显示数据库的数据和日志构成。下面就是以数学公式表示数字是如何得出的（仍然以 AdventureWorks 为例）：

```
TOTAL (469.94 MB) = USED (168.41 MB) + FREE (301.53 MB)
TOTAL (469.94 MB) = DATA (243.94 MB) + LOG (226.00 MB)
USED (%) (35.84 %) = USED (168.41 MB) / TOTAL (469.94 MB)
FREE (%) (64.16 %) = FREE (301.53 MB) / TOTAL (469.94 MB)
DATA (used %) (66.02 %) = used (161.06 MB) / DATA (243.94 MB)
LOG (used %) (3.25 %) = used (7.35 MB) / LOG (226.00 MB)
```

值得注意的是在结果中数据库 Test_snapshot 的一些值是 null。Test_snapshot 是一个快照数据库，设计中它是不允许有日志文件的。此外，在报告的底线有一个汇总行显示了每一列的小计。

(作者：Richard Ding 译者：陈柳／曾少宁 来源：TT 中国)

用存储过程检查 SQL Server 数据库和日志文件大小（二）

通常情况下，DBA 在检查数据库空间时，都会运行系统存储过程 sp_spaceused，或者从 SQL Server Management Studio 获取磁盘使用报告。将 sp_SDS 与其它的方法作比较是非常有趣的。图 2 的图表显示的是它们之间比较的几个画面组合。它包含四个部分。第 1 部分是数据级的 sp_SDS 结果。第 2 部分显示 sp_spaceused 的查询结果。第 3 部分与第 1 部分相类似，但它是文件级的。第 4 部分表示 SSMS 绘制的饼图。相关的值颜色突出并与各个部分相连接。

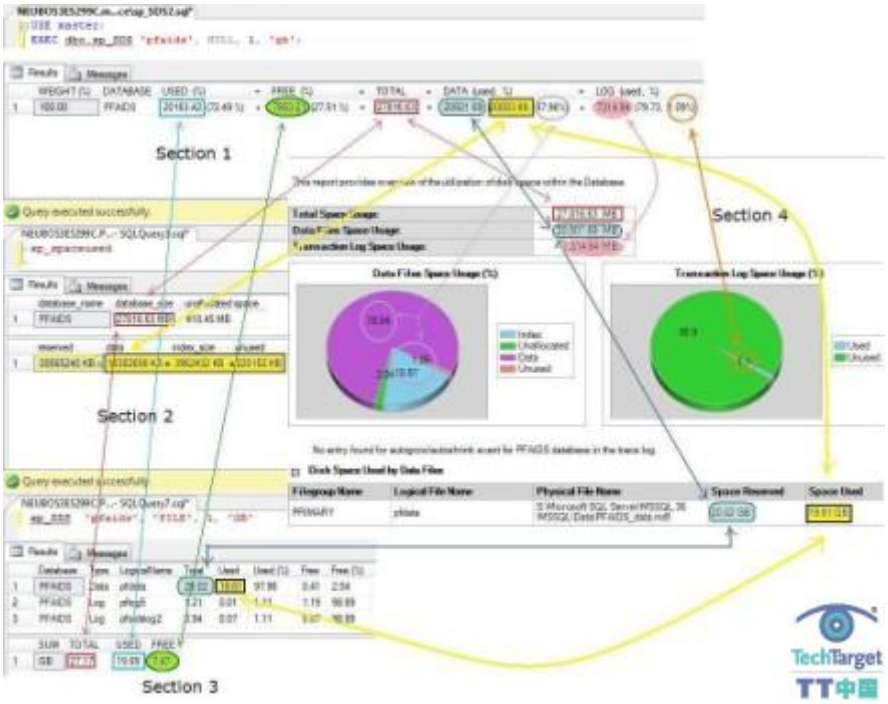


图 2：图表显示数据库级 sp_SDS、文件级 sp_SDS、sp_spaceused 和磁盘使用饼状图报告之间的比较。

在 SQL Server 2000 中，Enterprise Manager 的 Taskpad 视图与 SQL Server 2005 所绘制的有所不同。如图 3，我所管理的其中一个数据库，叫做“LANEPMSI”，它包含 54 个

数据文件和 1 个日志文件。目前，我们先不考虑为什么一个 4.5GB 的数据库需要这么多的数据文件——这个是厂商的选择。我所要强调的是，这么多的文件数对于 DBA 而言，计算这些文件大小是很困难的。如果使用 sp_SDS 来获取数据就像在列表 3 中的 T-SQL 代码按 F5 键（刷新）一样容易。图 4 反映了执行列表 3 中的 T-SQL 语句的结果。

```
USE master;
EXEC dbo.sp_SDS 'LANEPMSI', 'DATABASE', 1, 'MB';
EXEC dbo.sp_SDS 'LANEPMSI', 'file', 1, 'MB';
```

列表 3：在 SQL Server2000 数据库中以数据库级和文件级执行 sp_SDS。

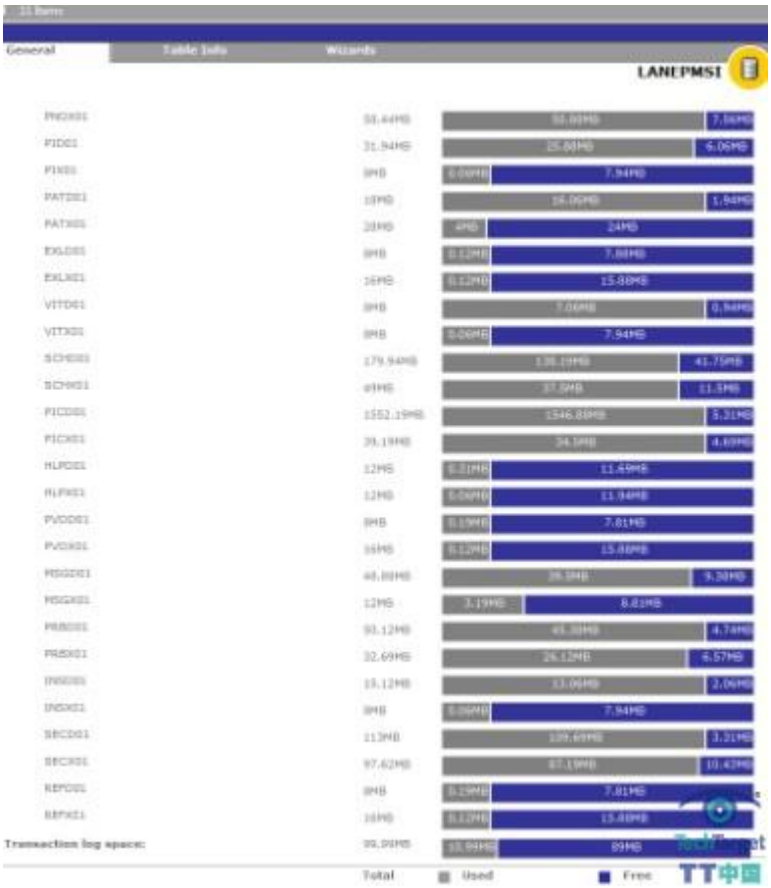


图 3：一个 SQL Server 2000 数据库包含大量的数据文件，这使得显示数据库空间汇总、已使用空间和未使用空间 e 非常困难。

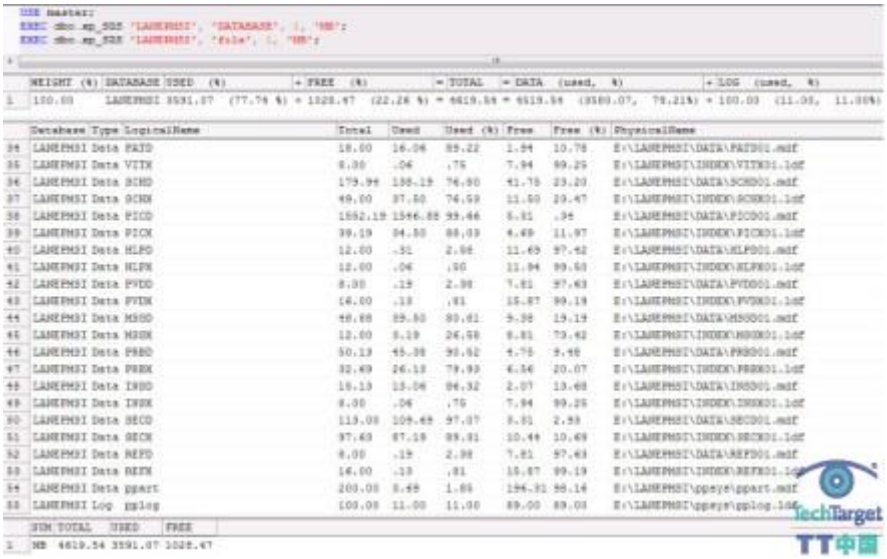


图 4：Sp_SDS 可以让你查看已使用、未使用、数据和日志空间总数。它同时还将数据库分为单独数据和日志文件，并显示了每个文件之间相应值。

Sp_SDS 兼容 SQL Server 2000、2005 和 2008。图 5 显示了在 SQL Server 2008 同时运行 sp_SDS 和系统存储过程 e 中 sp_spaceused 的情况。你可以比较图 5 和图 6 的数据，它们都显示了一个硬盘使用报告（Disk Usage Report）。

图 5：Sp_SDS 兼容 SQL Server 2008。在 AdventureWorks2008 示例数据库中运行 sp_SDS 和 sp_spaceused 的结果比较。

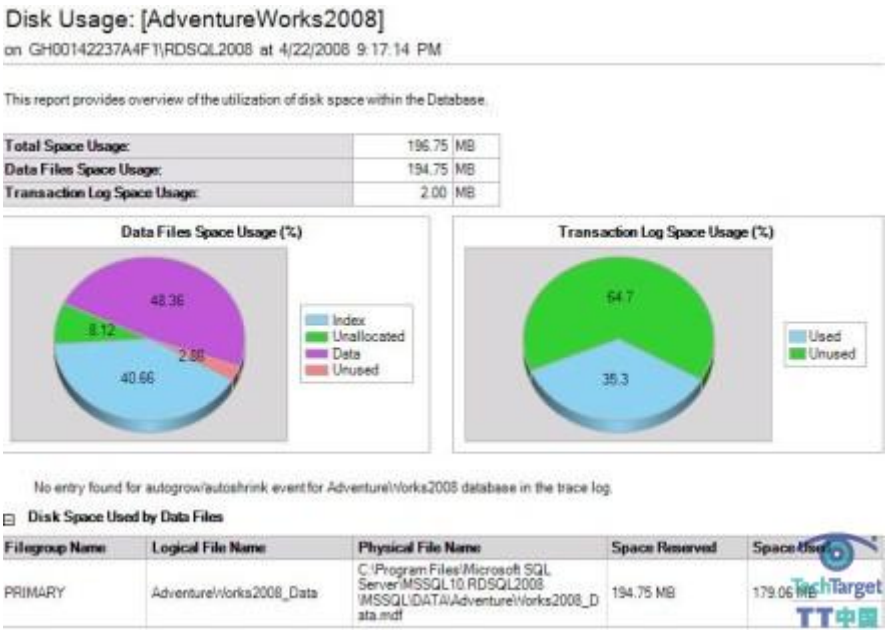


图 6：在 SQL Server 2008 上 AdventureWorks2008 示例数据库的一个 Disk Usage 报告饼状图。

Sp_SDS 和它的计算方法对于各种 DBA 任务是非常有用的。你是否曾经被要求为一个管理会议准备一份数据库空间使用报告呢？你可以使用 sp_SDS 来快速生成一个整洁的表格格式报告。一个 DBA 的责任就是监控数据库的增长。你可以安排一个日常工作来运行 sp_SDS 并将结果存储在一个表中。随着时间的推移，你将建立起一个数据仓库，并用它来分析数据库增长趋势。

创造数据或日志增长的提醒是存储过程 sp_SDS 的另外一个实际应用。如果大小超过了某个临界值，你可以发出警告或者采取其它的措施，如执行一个事务日志备份工作。有时候 DBA 并不需要压缩整个数据库。他只需要缩小占用最大存储空间的一个或几个数据文件。在这种情况下，参数@Level = 'FILE'的 sp_SDS 可以帮助 DBA 快速地确定那个（些）

文件应该压缩。当由于生产环境中磁盘空间不足而需要重建数据库时，SQL Server 数据库空间的详细的文件级统计分析对于 DBA 正确转移数据库文件是非常有用的。

(作者：Richard Ding 译者：陈柳／曾少宁 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：陈柳／曾少宁 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 14:22:30.001	14:23

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：陈柳／曾少宁 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：陈柳／曾少宁 来源：TT 中国)

配置 SQL Server 内存设置（一）

与其它的应用一样，SQL Server 应用需要内存才能运行。但是，与大多数应用不同的是，SQL Server 仅仅允许你决定它可以使用多少内存。这是很有用的，因为 SQL Server 需要更多的内存。

服务器内存选项是在 GUI 中设置或使用 sp_configure 存储过程的调整“最大化服务器内存（MB）设置”来设置的。

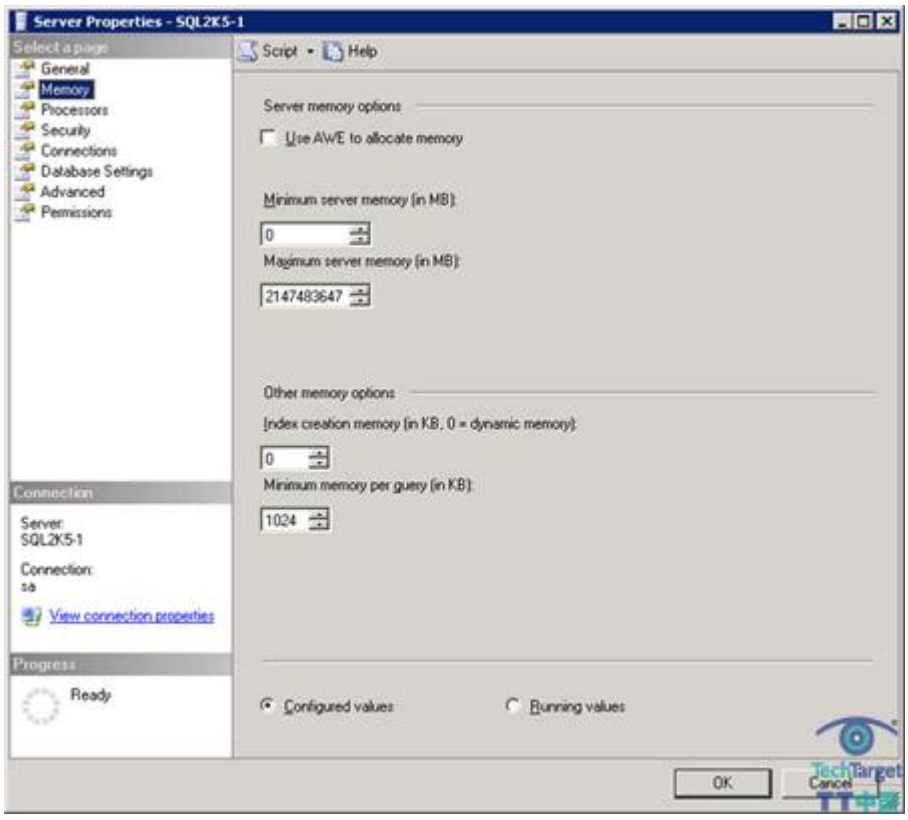


图 1：设置服务器内存属性

与大多数应用不同的是，你可以通过启用 AWE（Address Windowing Extensions)内存设置来配置 SQL Server 使用超过 2GB 的 RAM，这可以在相同的内存设置 GUI 上设置或者通

过 sp_configure 存储过程来调整“AWE 启用”设备。这两个都 是高级设置，在没激活“显示高级选项”设置之前是无法看到的。

SQL Server 需要的 RAM 总数

有一说法很适合 Microsoft Windows：当拿不定主意时，就多买些 RAM。这个说法对于 SQL Server 就不是很适用了，除非 SQL Server 只拥有非常小的数据量。SQL Server 要求服务器上至少有 512 RAM，而 Microsoft 推荐使用 1GB RAM。我个人的建议是服务器上至少有 1.5 GB RAM，1GB RAM 用于 SQL Server， 512 MB RAM 用于 Windows。如果 SQL Server 使用了服务器上所有的内存，而 Windows 没有足够的内存来运行，那么 SQL Server 的运行将会类似于内存不足情况。当 Windows 开始在硬盘驱动器进行越来越多的 RAM 页面调度时，查询响应时间将会增加，CUP 使用率将上升以及磁盘 I/O 将暴涨。

虽然跟 SQL Server 2000 不同的是，SQL Server 2005 没有 RAM 限制，但是软件选择依然是很重要的。当选择你的操作系统（OS）时，你一定要选择符合你的内存要求的正确的版本。Windows 2003 Enterprise Edition 支持最多 64GB RAM——远远超过 Windows 2003 Data Center Edition 要求。因此，购买一个 128GB RAM 的服务器和 Windows 2003 Enterprise Edition 将会浪费一半的内存。

最小和最大服务器内存设置

在 SQL Server 中有两种数字内存设置——最小服务器内存和最大服务器内存。虽然关于最小服务器内存设置是如何工作的存在一些争论，但是最大服务器内存是很明确的：它只不过就是 SQL Server 可使用的最高的内存总数。

很多人认为最小服务器内存设置是指在 SQL 第一次启动时，SQL Server 应该使用多少内存，但是事实并非如此。最小服务器内存设置是一种低水印设置。如果 Windows 需要从 SQL Server 回收内存，它将要求 SQL Server 释放它占用的内存。SQL Server 将返回内存到操作系统，直到占用的内存总量达到最小服务器设置值。

通常情况下，我建议的最大内存设置为低于服务器内存总额 512MB 的值。但当服务器上大约有 8GB RAM 时，我会将这一建议值改为 1GB 内存。我之所以这样做是因为有这样大内存的系统上通常运行着很多系统进程——比如备份软件、大量 DTS/SSIS 包 运行等等——所以额外的内存对于操作系统是很有益的。

（作者：Denny Cherry 译者：陈柳／曾少宁 来源：TT 中国）

配置 SQL Server 内存设置（二）

多个 SQL Server 实例的内存设置

当处理多个实例时，决定内存设置是相当棘手的。当你只有一个实例时，你只需要简单地先确定 OS 需要的 RAM，然后将剩下的内存设置给数据库。随着实例地 增多，你必须仔细决定每个实例需要多少内存。对于数据库较小并且每秒内事务较少的 SQL Server 实例，显然比有更大数据库的实例需要较少的内存资源。当作这些决定时，你一定要清楚你正在使用的是哪个 SQL Server 版本，以及它是如何计算 SQL Server 将使用的程序缓存总数的。在 32 位平台上，程序缓存必须分配在主应用的内存空间（RAM 的 2GB 空间），即使启用了 AWE。你可以在 SQL Server with Mr. Denny 博客上阅读更多关于程序缓存的内容。

32-bit platforms	
SQL Server 2000	50% of the allocated memory or 1 GB, whichever is lower
SQL Server 2005&1t;SP2	50% of the allocated memory or 1 GB, whichever is lower
SQL Server 2005>= SP2	50% of the allocated memory or 1 GB, whichever is lower
64-bit platforms	
SQL Server 2000	50% of the allocated memory or 1 GB, whichever is lower
SQL Server 2005&1t;SP2	75% of the first 8 GB + 50% of the next 56 GB + 25% of the RAM over 64 GB
SQL Server 2005>= SP2	75% of the first 4 GB + 10% of the RAM over 4 GB

AWE 内存管理

Microsoft Windows 的 Address Windowing Extensions API 允许应用开发者在 32 位系统上访问超过 2GB 的内存。在 Windows Server 2000 中，AWE 只能在 Advanced Server 和 Data Center 版本的操作系统上使用。在 Windows Server 2003 中，这三个服务器版本 AWE 都适用。为了使用 AWE，你必须通过添加/PAE 开关到你的 boot.ini 文件上以启用物理地址扩展。在 Windows 2003 SP1 中，当安装了超过 2GB 的 RAM 时，Windows 将自动启动 PAE。

另外一个需要添加到 boot.ini 文件中的开关是/3GB 开关。/3GB 开关使 SQL Server 能够访问高于 3GB 的 RAM。

Microsoft 已经撰写了一篇非常出色的标题为“如何配置 SQL Server 使用超过 2 GB 的物理内存”的 KB 文章。但是，有很多时候/3GB 开关是不应该使用。其中就包括运行 Windows 2003 Data Center 版本，以及有超过 16GB 的 RAM 的系统。

当在 32 位系统上运行 SQL Server 时，除非你正在使用超过 2GB 的 RAM，否则你不能启动 AWE。因为这样做可能导致出现 SQL Server 性能问题。

x64/64i 平台的变化

在目前的 64 位平台上，内存使用方面已经作了很大的改进。虽然 32 位平台要求你使用 AWE 和 PAE 访问超过 2GB 的 RAM，但是 64 位平台没有这些限 制。在 64 位平台上，所有内存都可用于应用，只要它们是作为 64 位的应用编译的；运行在 Windows on Windows (WOW)的 32 位应用也有着与它们在 32 位平台上运行时的相同的内存限制。

虽然 SQL Server 只是提供几个简单的内存设置，但是对它进行正确的设置是极为重要的。正确的内存设置将使 SQL Server 能长时间平稳地运行。内存设置必须定期检查以确保原先的设置仍然是恰当的。毕竟，去年安装的内存总量可能已经不够用或者不正确了。

(作者：Denny Cherry 译者：陈柳／曾少宁 来源：TT 中国)

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：陈柳／曾少宁 来源：TT 中国）

SQL Server 中日期/时间值到字符类型的数据转换（二）

显式转换数据

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<? xml:namespace prefix = o /><? xml:namespace
                              prefix = o /><? xml:namespace prefix = o />
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：陈柳／曾少宁 来源：TT 中国）