2009 年度十大

SQL Server 技巧文章

2009 年度十大 SQL Server 技巧文章

在向 2009 年告别之际，我们来回顾一下过去的一年中最受欢迎的 SQL Server 技巧，包括了 OPENROWSET、FILESTREAM 等函数的用法、密码工具介绍以及 DBA 日常工作建议等内容。 通过对这些精华文章的再次回顾，希望可以帮助您梳理一下这一年以来的工作以及学习心得，对未来一年更进一步打下更坚实的基础。

使用 bcp 工具导入和导出批量数据

微软 SQL Server 中的批量复制程序（Bulk Copy Program，BCP）能让数据库管理员将数据批量导入表中或将数据从表中批量导入文档中。它还支持一些定义数据如何导出、导入到什么地方、加载哪些数据等选项。

- 用 bcp 工具导入和导出批量数据（上）
- 用 bcp 工具导入和导出批量数据（中）
- 用 bcp 工具导入和导出批量数据（下）

使用 SQL Server 的 OPENROWSET 函数

你可能常常会需要运行一个 ad hoc 查询从远程 OLE DB 数据源提取数据，或者批量向 SQL Server 表导入数据。在这种情况下，你可以在 T-SQL（Transact-SQL，微软对 SQL 的扩展）中用 OPENROWSET 函数给数据源传入一个连接串和查询来提取需要的数据。

- 使用 SQL Server 的 OPENROWSET 函数（上）
- 使用 SQL Server 的 OPENROWSET 函数（下）

SQL Server 密码破解工具简介

在对 SQL Server 系统执行入侵测试或者更高级别的安全审计时，有一种测试不应该被忽略，那就是 SQL Server 密码测试。这一点看起来显而易见，但是很多人都会忽略它。

- SQL Server 密码破解工具简介

使用 XML 在 SQL Server 上创建计算列

在 SQL Server 数据库中，当你想使用一个数据，而这个数据不保存在表中，计算列很有用。在 SQL Server 中使用 XML 数据来创建计算列，你的列定义必须包含必要的用来检测向列中插入的是什么数据的表达式。

- 使用 XML 在 SQL Server 上创建计算列（上）
- 使用 XML 在 SQL Server 上创建计算列（下）

SQL Server 中已满事务日志原因的检测

对于 SQL Server 数据库管理员来讲，已满事务日志是一个琐碎的，但又很常见的问题。它能引发事务的提前终止，甚至通过阻止所有事务的引入，从而引起系统的崩溃。对于数据库管理员来说，关键是理解将要发生的情况，以便他们可以追踪引起问题的原因。

- SQL Server 中已满事务日志原因的检测（上）
- SQL Server 中已满事务日志原因的检测（下）

DBA 五大浪费时间的工作

DBA 以常规方式执行的一些任务，不仅对 SQL Server 数据库几乎没有益处，而且实际上可能对他们的生产环境造成不利影响。在本文中，我会阐述几类这样的工作。如果你正在执行其中的一些工作，我希望你能尽快停下来。

- 收缩数据库
- 碎片整理后重建索引
- 恢复完整备份
- 删减事务日志
- 人工通读错误日志

利用动态管理视图提高 SQL Server 索引效率

就如同数据库 DBA 了解的一样，合适的索引能够提高查询性能和应用程序可测量性。但是每个附加的索引，都给系统增加了额外开销，因为随着数据从表和视图中不断增加、修改或清除，SQL Server 需要维护这些索引。

- 利用动态管理视图提高 SQL Server 索引效率（一）
- 利用动态管理视图提高 SQL Server 索引效率（二）
- 利用动态管理视图提高 SQL Server 索引效率（三）

在 SQL Server tempdb 满时检查数据文件

作为一名数据库 DBA，肯定会听说过“tempdb 数据库满了”。通常我们很容易确定造成这一问题的原因。但是更多的时候这一问题主要源于一组请求，涉及到新代码部署或逐渐增加的数据。

- 在 SQL Server tempdb 满时检查数据文件

SQL Server 运作的简短课程

在实际执行任务之前，有一点背景信息可以会起到帮助作用。那么，到底 SQL Server 是如何工作的呢?不管你信不信，理解“黑盒”知识几乎可以在 Microsoft SQL Server 的所有方面起到帮助作用，例如从备份与存储到复制与镜像。

- SQL Server 运作的简短课程

实现 SQL Server 2008 中的文件流功能

SQL Server 2008 中最新的文件流功能使得你可以配制一个数据类型为 varbinary（max）的列，以便将实际数据存储在文件系统中，而非在数据库中。只要愿意，你仍可以作为一个常规的二进制列来查询此列，即使数据自身存储在外部。

- 实现 SQL Server 2008 中的文件流功能（上）
- 实现 SQL Server 2008 中的文件流功能（下）

用 bcp 工具导入和导出批量数据（上）

微软 SQL Server 中的批量复制工具程序(Bulk Copy Program,BCP)能让数据库管理员将数据批量导入表中或将数据从表中批量导入文档中。它还支持一些定义数据如何导出、导入到什么地方、加载哪些数据等选项。

本技巧讨论一些用 bcp 命令批量复制数据迁入或迁出 SQL Server 表的示例。这些示例在 SQL Server 2005 和 SQL Server 2008 上已经测试过。并且我还用了 AdventureWorks 样本数据库。

用 bcp 工具导入数据

一个最简单的操作就是你可以用 bcp 工具将数据从 SQL Server 表 bulk-copy 到文本文件。在 Windows 命令提示符中插入命令，你就可以运行 bcp 命令了。例如以下命令，从 AdventureWorks 数据库里的 Sales.vSalesPerson 视图复制数据到 C:\Data\SalesPerson.txt 文件：

```
bcp AdventureWorks.Sales.vSalesPerson out C:\Data\SalesPerson.txt -c  -T
```

如同你看到的一样，bcp 命令以工具名称开头，后面为完全合格表名 database.schema.table。接下来就是 out 关键字，关键字告诉 bcp 工具数据将会从该表中导出。目标文本文件的路径和文件名称紧跟 out 关键字之后。注意本文中列出的命令例子可能包括很多行，但是所有的例子应该像一个单独的命令一样运行。

除了这些基本参数，bcp 工具还支持控制工具行为的 switch。在以上例子中，无论数据是以何种方式存储在源表中的，-c switch 表示所有的数据都应是字符数据。如果你没有指定-c 开关或其他相关类型的 switch，你就需要在进入 bcp 命令后指定每个列的 switch 类型。

上述例子中另一个 switch 就是-T，它主要是告诉 bcp 工具使用可靠连接来关联 SQL Server 示例。如果你没有指定-T，你就必须提供用户名（-U switch）和密码（-P switch），或者你需要提供相关信息。

因为在先前列举的例子中没有指定实例，bcp 工具就在本地机上使用的默认实例。要指定一个 SQL Server 实例，就要用到-S switch，后面紧跟的是服务器名称，如下所示：

```
bcp AdventureWorks.Sales.vSalesPerson out C:\Data\SalesPerson.txt -c -T -
S Server01
```

Bcp 工具现在和 Server01 上的默认实例连接。如果你想连接到具体实例而不是默认的实例，你就必须指定实例名称和服务器名称，如 Server01\SqlSrv。

通过默认，bcp 工具使用制表符分隔目标文件中里的域。但你也可以用-t switch 调过这一操作，如下：

```
bcp AdventureWorks.Sales.vSalesPerson out C:\Data\SalesPerson.csv -c -
T -t,
```

在这种情况下，-t switch 后有一个逗号，意思就是说数据域现在由逗号分开。这样做可让你讲数据保存到.csv 文件，便于你在 Microsoft Excel 文件中查看这些数据。

以上一些例子只限于将数据从表中导出。但是你还可以用 bcp 命令运行 Transact-SQL 查询、到出查询条件。例如以下 bcp 命令，包括只从 vSalesPerson 视图中检索 SalesPersonID、FirstName 和 LastName 的 SELECT 语句：

```
bcp "SELECT SalesPersonID, FirstName, LastName FROM AdventureWorks.Sales.vSale
sPerson"
queryout C:\Data\SalesPerson.csv -c -T -t,
```

这种情况下引号里的查询通过的是 bcp 命令而不是表名称。此外，queryout 取代了 out 关键字。但是命令其他部分和先前的例子相同。结果，SalesPerson.csv 文件现在只包含三个指定列。你还可以让查询更加精炼：例如你可以包括限定只从源表中返回那些行的 WHERE 子句。

（作者：Robert Sheldon 译者：April 来源：TT 中国）

原文标题：用 bcp 工具导入和导出批量数据（上）

链接：http://www.searchdatabase.com.cn/showcontent_20432.htm

用 bcp 工具导入和导出批量数据（中）

用 bcp 工具导入数据

Bcp 工具使得导入数据和导出一样简单。要在这部分里运行这个示例，首先就要执行下面的 T-SQL 脚本，在 AdventureWorks 数据库里创建 SalesPeople 表：

```
USE AdventureWorks
GO
IF OBJECT_ID (N'SalesPeople', N'U') IS NOT NULL
DROP TABLE dbo.SalesPeople
GO
CREATE TABLE dbo.SalesPeople (
SalesPersonID INT IDENTITY PRIMARY KEY,
FirstName NVARCHAR(50) NOT NULL,
LastName NVARCHAR(50) NOT NULL
)
```

要导出数据，你需要一个源文件从中复制数据。例如下面的命令就用了最近创建的 SalesPerson.csv 文件加载数据到 SalesPeople 表：

```
bcp AdventureWorks.dbo.SalesPeople in C:\Data\SalesPerson.csv -c -T  -t,
```

首先，你必须指定目标表，这种情况下紧跟其后的为代替 out 或 queryout 的 in 关键字。其次，你必须指定源文件的路径和文件名称，后跟任意可用的 switch。

在运行这一命令并查看结果时，要注意源文件如果包括售货员的 ID，这些值就不能插入到 SalesPersonID 列。该列定义为 IDENTITY 列，这样我们就可以忽视源数据。要保留原始值，你就必须在命令里增加-E switch，如下面的例子中所示：

```
bcp AdventureWorks.dbo.SalesPeople in C:\Data\SalesPerson.csv -c -T -t, -E
```

现在表中就包含了你想要得到的数据。

使用格式文件

在导入或导出数据时，你会发现源数据架构和目标数据架构不匹配。例如，文本文件中的这些列可能和目标表中的列的顺序不一致，或者说这些列的多少还不一样。你可以通过创建格式文件映射源文件和目标架构解决这个问题。我们通过以下示例看看它是如何工作的：

假如你使用了以下命令从 vSalesPerson 视图中导出数据到 SalesPeople.txt 文件：

```
bcp "SELECT LastName, FirstName, SalesPersonID FROM AdventureWorks.Sales.vSale
                                    sPerson"
                    queryout C:\Data\SalesPeople.txt -c -T -t,
```

该命令使用了先前例子中同样的参数。但是注意这些列从视图中检索的顺序：LastName、FirstName、最后是 SalesPersonID。

现在假设你打算使用该文件导入数据到 SalesPeople 表。SalesPeople 表中列的顺序和文本文件中列顺序不同。要解决这个问题，你可以创建一个格式文件将这些列从源文件映射到目的文件。以下命令说明如何创建一个格式文件：

```
bcp AdventureWorks.dbo.SalesPeople format nul -f C:\Data\SalesPeople.fmt -
                                    c -T -t,
```

（作者：Robert Sheldon 译者：April 来源：TT 中国）

原文标题：用 bcp 工具导入和导出批量数据（中）

链接：http://www.searchdatabase.com.cn/showcontent_20480.htm

用 bcp 工具导入和导出批量数据（下）

先前的例子表明，命令首先指定了目标表。而这一次表明后面为关键字 format nul，表示 bcp 工具应该创建格式文件。-f 参数用于指定格式文件的路径和文件名称，其后为 switch。最后在你运行该命令时，就生成了包括 SalesPeople 表架构的格式文本。

下面的数据显示上述命令生成的 SalesPeople.fmt 格式文本内容：

```
                                10.0
                                  3
          1   SQLCHAR    0    12    ″,″    1    SalesPersonID   ″″
 2   SQLCHAR   0   100    ″,″   2   FirstName   SQL_Latin1_General_CP1_CI_AS
3   SQLCHAR   0   100   ″\r\n″    3   LastName   SQL_Latin1_General_CP1_CI_AS
```

文件(10.0)第一行确定了目前使用的 bcp 版本。第二行(3) 确定了表中的列数，接下来的三行为列的相关信息：

* 第一个域为列在源文件中出现的顺序。

* 第二个域显示每个列的源文件数据类型。因为在生成文件时指定了 - c switch，在从数据文件中提取时，所有的域都用字符型数据类型。插入数据时，SQL Server 会将数据转换到正确的类型。

*第三个域表示域的前缀长度，SQL Server 通常会用它来提供最紧凑的文件存储。在创建格式文件时如果你指定-c switch，那么就会自动用到 0。

* 第四个域代表特殊域数据类型字节长度。

* 第五个域表明如何终止行和列。由于在创建格式文件时用了-t switch，源文件的域值就必须通过逗号终止。

* 第六个域映射这些列在 SQL Server 表中的排列顺序。

*第七个和最后一个域提供了 SQL Server 表中字符列的整理信息。

要用格式文件将数据导入 SalesPeople 表中，我们必须如下修改文件：

```
                                10.0
                                  3
1    SQLCHAR    0    100    ","    3    LastName    SQL_Latin1_General_CP1_CI_AS
    2   SQLCHAR   0   100   ","   2   FirstName   SQL_Latin1_General_CP1_CI_AS
             3   SQLCHAR   0   12   "\r\n"    1   SalesPersonID    ""
```

你可以看到，列的顺序已经进行了修改，这一顺序就是它们在格式文件中的排列顺序。SalesPersonID 列现在排在最后，并且以\r\n 结尾。LastName 列现在排在开头并且以逗号结尾。

修改、保存格式文件后，准备在 bcp 命令中用了。下面的例子说明如何调用格式文件：

```
bcp AdventureWorks.dbo.SalesPeople in C:\Data\SalesPeople.txt -
                f C:\Data\SalesPeople.fmt  -T
```

注意，你从 SalesPeople.txt 文件中导入数据时，还必须用到-f switch 调用格式文件。还要注意到你现在已经不需要包括-t 和-c switch，因为现在在格式文件中已经包括了这些信息。

无论你用的是格式文件还是只运行基本命令，你现在都应该更好地了解到了如何使用 bcp 工具。记住 bcp 工具支持的 switch 比我所列举的要多得多。你可以在《SQL Server 联机从书》上了解更多有关 bcp 工具的信息。同时，本技巧还提供了有关启用 bcp 工具、轻松将数据导入到 SQL Server 表以及导出数据到文本文件的比较充足的信息。

（作者：Robert Sheldon 译者：April 来源：TT 中国）

原文标题：用 bcp 工具导入和导出批量数据（下）

链接：http://www.searchdatabase.com.cn/showcontent_20482.htm

使用 SQL Server 的 OPENROWSET 函数（上）

你可能常常会需要运行一个 ad hoc 查询从远程 OLE DB 数据源提取数据，或者批量向 SQL Server 表导入数据。在这种情况下，你可以在 T-SQL（Transact-SQL，微软对 SQL 的扩展）中用 OPENROWSET 函数给数据源传入一个连接串和查询来提取需要的数据。

你可以使用 OPENROWSET 函数从任何支持注册 OLE DB 的数据源获取数据，比如从 SQL Server 或 Access 的远程实例中提取数据。如果你用 OPENROWSET 从 SQL Server 实例中获取数据，该实例必须配置为允许 ad hoc 分布式查询。

要配置远程 SQL Server 实例支持 ad hoc 查询，需要使用系统存储过程 sp_configure 先设置 advanced options，再启用 Ad Hoc Distributed Queries（ad hoc 分布式查询）。请看下面的 T-SQL 脚本：

```
EXEC sp_configure 'show advanced options', 1;
GO
RECONFIGURE;
GO
EXEC sp_configure 'Ad Hoc Distributed Queries', 1
GO
RECONFIGURE;
GO
```

要注意的是，在运行完存储过程之后，你必须运行“RECONFIGURE”命令。

一旦你配置好了远程 SQL Server 实例，你就可以对它使用 OPENROWSET 函数。这个函数可以在 SELECT 语句的 FROM 从句里使用。下面的例子显示了该函数的基本语法：

```
OPENROWSET('provider', 'connection string', target)
```

可以看到，这个函数有三个参数：

• Provider —— 某特定数据源支持的 OLE DB 提供者的人机友好名称（ProgID）。Provider 的名字必须用单引号括起来。

• Connection string —— 连接串。它是与具体提供者 provider 相关的字符串，包括连接到给字符串中指定的数据源所需要的细节信息。根据 provider 的不同，连接串信息需要用一对或多对单引号括起来。

• Target —— target 参数可以使一个数据库对象或者一个查询。

o Object —— 数据库对象的名字，比如表或者视图的名称。对象的完整名字必须提供，它们不需要用单引号括起来。

o Query —— query 是从远程数据源提取数据的 Select 语句。Query 必须用单引号括起来。

下面的例子展示了 OPENROWSET 函数的用法：

```
SELECT Employees.*
  FROM OPENROWSET(
    'SQLNCLI',
'Server=SqlSrv1;Trusted_Connection=yes',
'SELECT EmployeeID, FirstName, LastName, JobTitle
  FROM AdventureWorks.HumanResources.vEmployee
    ORDER BY LastName, FirstName'
    ) AS Employees
```

注意该 Select 语句的 FROM 从句中使用了 OPENROWSET 函数和 3 个参数。第一个参数 SQLNCLI 是 SQL Server OLE DB 提供者的名称。

（作者：Robert Sheldon 译者：冯昀晖 来源：TT 中国）

原文标题：使用 SQL Server 的 OPENROWSET 函数（上）

链接：http://www.searchdatabase.com.cn/showcontent_21142.htm

使用 SQL Server 的 OPENROWSET 函数（下）

第二个参数是连接串。对于 SQL Server 提供者，整个连接串应该被单引号括起来，连接串内的每一组信息用分号分割。在上面的例子中，第一组信息指定了目标服务器 SqlSrv1，第二组信息指定了该连接可信任连接。在指定目标 Server 时，如果实例不是该 Server 的默认实例，则一定要在连接串中指定实例名。（注意：SQLNCLI 提供者还支持其他参数。）

OPENROWSET 函数的最后一个参数是实际执行的 Select 语句。注意 SQL 语句中使用了完整对象名来访问视图。

这样我们就可以使用 OPENROWSET 函数了。函数返回一个结果集（我把它用 AS 命名为“Employees”），From 使用该结果集的方式与使用其他普通查询的方式一样。

我们在上面提到，你也可以从 SQL Server 以外的数据源提取数据。例如：下面的 Select 语句查询微软 Access 数据库的 Employees 表。

```
SELECT Employees.*
  FROM OPENROWSET(
 'Microsoft.Jet.OLEDB.4.0',
 'C:\Data\Employees.mdb';'admin';' ',
 'SELECT EmployeeID, FirstName, LastName, JobTitle
  FROM Employees
 ORDER BY LastName, FirstName'
 ) AS Employees
```

你可能注意到了，这次的 provider 不同于我们在访问 SQL Server 时使用的 Provider。在本例中，Provider 是 Microsoft.Jet.OLEDB.4.0（注意：对于 Access 2007，有新的 Provider 可用）。

连接串与前面例子中的写法也不一样。整个连接串从头到尾分成了三部分，每一部分都被单引号单独括起来，各部分之间用分号分割。

第一部分指定了 Access 数据库文件的路径和文件名，后面紧跟着是用户账号 admin（Access 数据库内部的管理员账号）。第三部分是一个空字符串，是 Access 数据库的密码。因为 admin 账号没有设定密码，所以使用空字符串。如果该账号设置了密码，应该把密码写在第三部分。

整个连接串与后面用来从 Access 数据库查询数据的 Select 语句用逗号“，”隔开。（我在 Access 中使用的 Employees 表是从 SQL Server 的 vEmployee 视图导入的）

这就是从 Access 数据库查询数据要做的全部事情。你的查询会返回一个结果集，该结果集与访问本地 SQL Server 数据库时得到的结果集类似。

你也可以使用 OPENROWSET 函数从多个数据源中查询数据。例如：下面的例子我使用 inner join（内连接）从远程 SQL Server 实例和 Access 数据库查询数据。

```
SELECT e1.EmployeeID, e2.FirstName, e2.LastName, e1.JobTitle
                    FROM OPENROWSET(
                        'SQLNCLI',
                'Server=SqlSrv1;Trusted_Connection=yes;',
        'SELECT EmployeeID, FirstName, LastName, JobTitle
          FROM AdventureWorks.HumanResources.vEmployee'
                    ) AS e1
                  INNER JOIN OPENROWSET(
                'Microsoft.Jet.OLEDB.4.0',
            'C:\Data\Employees.mdb'; 'admin';'',
        'SELECT EmployeeID, FirstName, LastName, JobTitle
                    FROM Employees'
                    ) AS e2
                  ON e1.EmployeeID = e2.EmployeeID
                ORDER BY e2.LastName, e2.FirstName
```

注意：外层的 Select 语句从两个表返回数据——从 SQL Server 返回员工 ID 和工作头衔，从 Access 数据库返回姓和名。由于你可以得到可靠的连接查询，尽管你是从本地 SQL Server 实例连接表中查询的数据，你可以处理这些数据。

现在我们来看看 OPENROWSET 函数的另一个重要功能——批量导入。为了举例需要，我在 AdventureWorks 数据库中用下面的脚本创建了表 Employees 并导入数据。

```
USE AdventureWorks
GO
IF OBJECT_ID (N'Employees', N'U') IS NOT NULL
DROP TABLE dbo.Employees
GO
SELECT EmployeeID, FirstName, LastName, JobTitle
INTO Employees
FROM HumanResources.vEmployee
GO
ALTER TABLE Employees
ADD ResumeFile VARBINARY(MAX) NULL
GO
```

注意：我没有把 ResumeFile 列的数据导入，它的数据类型是 VARBINARY(MAX)。我会用下面的 Update 语句把 Employee1.docx 文件作为二进制数据批量导入到该列。

```
USE AdventureWorks
GO
UPDATE Employees
SET ResumeFile = (
SELECT *
FROM OPENROWSET(BULK 'C:\Data\Employee1.docx', SINGLE_BLOB)
AS ResumeContent)
WHERE EmployeeID = 1
```

可以看到，OPENROWSET 函数提供了 BULK 选项，你可以用它来导入数据。要使用 BULK 选项，需要指定你想要导入的文件，并指定导入方式。既然我想把文件以二进制形式导入，我在上面的例子中使用了 SINGLE_BLOB 选项。当然，如果该列支持字符型数据，我也可以用 SINGLE_CLOB 或者 SINGLE_NCLOB 选项指定数据存储为字符类型格式。此外，在使用 OPENROWSET 函数批量导入数据功能时，你也可以使用格式化的文件，不过关于格式化文件的用法超出了本文讨论的范围。

不管你是否使用 OPENROWSET 函数批量导入数据或者连接 OLE DB 数据源，你都会发现用它获取数据非常方便。关于这个函数的更多细节，请在 SQL Server 联机图书查看标题为“OPENROWSET （Transact-SQL）”的文章。在那里你可以看到关于 SQLNCLI 提供者（或者 SQL Server 2008 的 SQLNCL10 提供者）的更多细节。

（作者：Robert Sheldon 译者：冯昀晖 来源：TT 中国）

原文标题：使用 SQL Server 的 OPENROWSET 函数（下）

链接：http://www.searchdatabase.com.cn/showcontent_21144.htm

SQL Server 密码破解工具简介

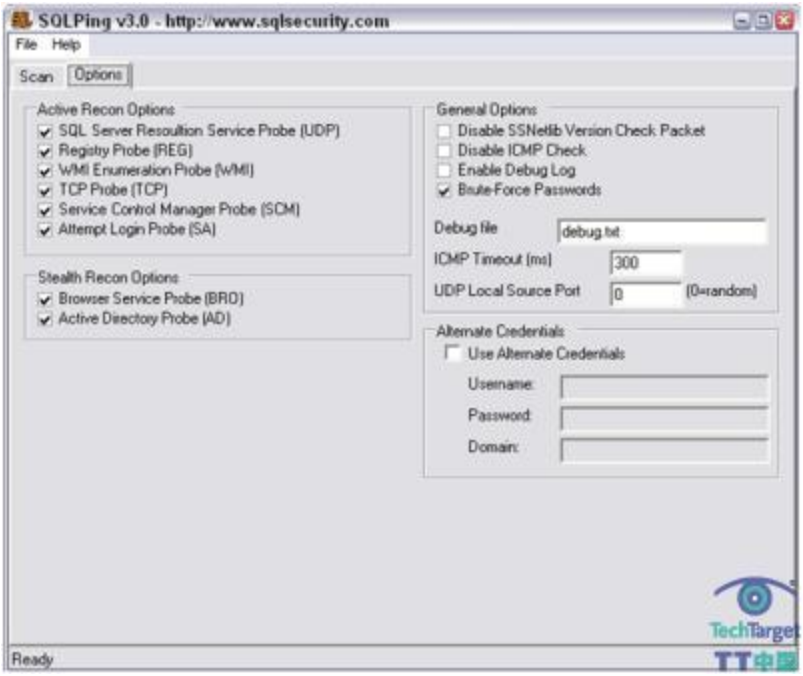
在对 SQL Server 系统执行入侵测试或者更高级别的安全审计时，有一种测试不应该被忽略，那就是 SQL Server 密码测试。这一点看起来显而易见，但是很多人都会忽略它。

密码测试可以帮助检查恶意入侵者或者外部攻击者，测试他们要强行进入数据库有多容易，而且还可以确保 SQL Server 用户对他们的账号负责。此外，测试密码的漏洞在 SQL Server 混合模式认证的情况下尤其重要，这种模式比其他 Windows 认证模式安全性要差一些。

密码测试的第一步是确定要测试的系统。虽然你对你的环境可能了如指掌，但是找出那些可能被遗忘的服务器，或者有未经你知晓有人就连接到网络的服务器的情况是没有坏处的。

SQLPing3 是一个免费的 SQL Server 查找和密码破解工具，可以帮助你开始测试。该工具有多个选项可以供你搜索活动状态的 SQL Server 系统，如图 1 所示：

图 1：用 SQLPing3 搜索活动 SQL Server 系统的选项界面(点击放大)。

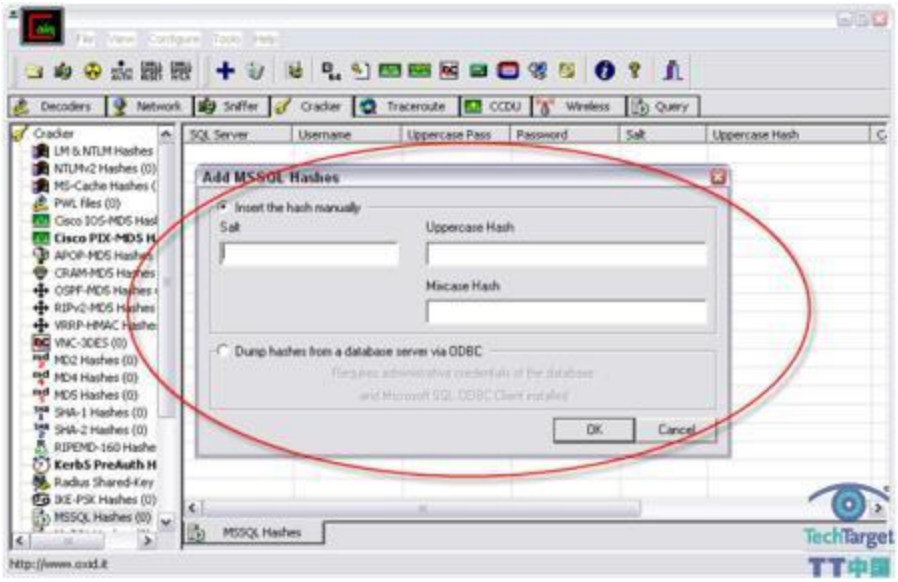


此外，SQLPing3 可以扫描到那些通过约定俗成的端口扫描可能扫描不到的 SQL Server 数据库实例，而且它可以找到那些“sa”密码为空的系统。SQLPing3 还可以针对 SQL Server 数据库运行字典攻击，这种做法就像加载你自己的用户账号和密码列表一样简单。

因为这是最基础层面的 SQL Server 搜索和密码破解，所以我们从这里开始非常合适。

另一个免费工具是 Cain&Abel，它支持你转存并攻击 SQL Server 数据库密码哈希，如图 2 所示：

图 2：使用 Cain&Abel 转存并破解哈希(点击放大)。

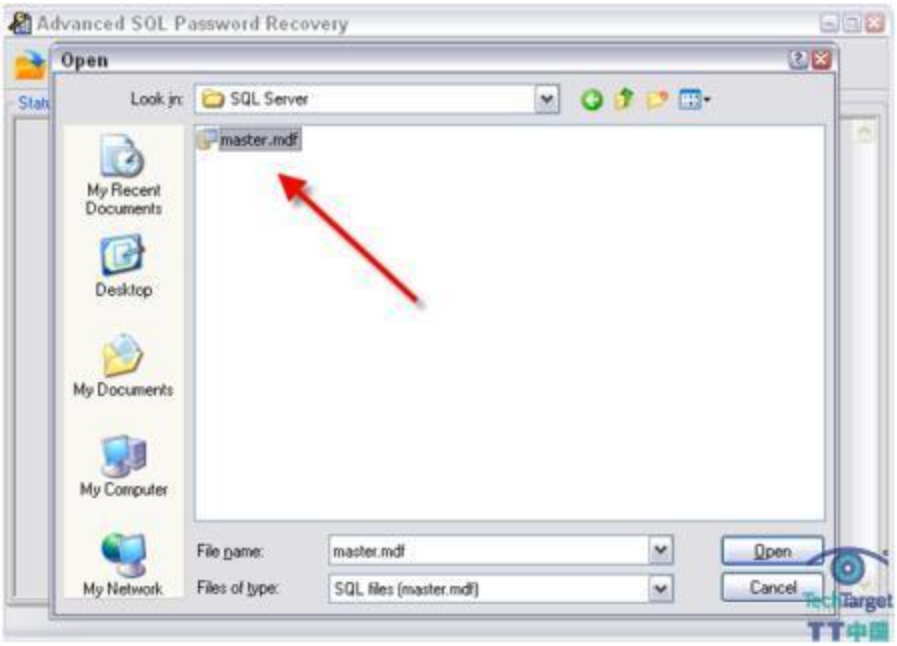


使用 Cain&Abel，你可以插入你自己的哈希或者通过 ODBC 连接到数据库并把它们一股脑转存下来，以便后续破解使用。

在商业软件里面，NGSSQLCrack 和 AppDetective Pro 都是很好的工具，他们可以执行字典破解和暴力密码破解。

我最喜爱的新的商业 SQL Server 密码破解工具是 Elcomsoft 公司开发的 Advanced SQL 密码恢复工具。使用 Advanced SQL Password Recovery，你可以立即从“master.mdf”文件中恢复密码，如图 3 所示：

图 3：使用 Advanced SQL Password Recovery，从“master.dbf”指向并直接点击密码破解(点击放大)。



这看起来似乎是不可思议的或者是完全不可能，因为 SQL Server 系统被认为在网络范围内是锁定的。然而，我经常会碰上管理员级别的密码或者发现丢失的补丁，一试之下，发现可以很容易地以全部权限访问数据库服务器。从这一点上看，系统中的一切就都是暴露着的游戏而已。

一定要记住的，SQL Server 密码破解不应该被忽视。要把它当成正式的安全评估，得到管理方面的支持，周密地规划。因为你不想碰到麻烦。

尽管如此，密码破解也存在一些缺点要记住：

- 密码破解会消耗宝贵的系统资源，包括 CPU 时间，内存和网络带宽，积累到一定量就会给系统造成拒绝式服务攻击。
- 字典型攻击和暴力攻击会花大量时间，有时候你可能得不到结果(时间太长)，尤其是如果你只能在特定的时间窗口内测试系统。
- 字典攻击的效果取决于你使用的字典，所以确保你操作时拿到的字典是可靠的。我发现 BlackKnight List 是最全面的字典。

最后，可能也是最重要的，一定要跟进你的发现。这可能意味着与管理层和你的 IT 部门同事分享你的发现，调整你的密码策略并传播安全意识，来说明安全对企业来说是多么重要的一个问题。

（作者：Robert Sheldon 译者：冯昀晖 来源：TT 中国）

原文标题：使用 SQL Server 的 OPENROWSET 函数（下）

链接：http://www.searchdatabase.com.cn/showcontent_21144.htm

使用 XML 在 SQL Server 上创建计算列（上）

在 SQL Server 数据库中，当你想使用一个数据，而这个数据不保存在表中，计算列很有用。例如，你有一张表，它包括列 dollar amounts, wholesale prices 和 retail prices。你肯定不想在每次查询表时来计算那两列之间的差值，你希望将其值保存在第三列中，让其自动计算前两列之间的差值。而此列就是计算列。

在 SQL Server 中使用 XML 数据来创建计算列，你的列定义必须包含必要的用来检测向列中插入的是什么数据的表达式。例如，在上面的例子中，你的表达式应该从 retail 列中的值减去 wholesale 列中的值。当你添加或更新表中的数据行时，差值将自动插入至计算列中。

你可以很容易地在两个或更多的包含字符串或数字类型值的列的基础上创建计算列。(更多关于如何创建此类型的计算列的详细信息，请参考 Microsoft SQL Server Books Online)。然而，如果你想要基于指定的 XML 列中元素值创建一个计算列，该过程相对更加复杂一些。因为你必须使用 Xquery 表达式来从 XML 列中获取指定元素数据,且 SQL Server 不支持在计算列的定义中使用 Xquery 表达式。

要解决此问题，可以创建一个函数来接收你想包含在计算列中的 XML 数据，并在计算列定义中调用此函数。更好的示范这是如何工作的，我们在这给出一个例子。我在 SQL Server 2005 的示例数据库 AdventureWorks 中创建以下的架构和表:

```
USE AdventureWorks;
GO
CREATE SCHEMA hr
GO
SELECT TOP 10 JobCandidateID AS CandidateID,
[Resume] AS JobResume
INTO hr.CandidateNames
FROM HumanResources.JobCandidate
GO
```

正如名称所示，HumanResources.JobCandidate 表中的 Resume 列是一个 XML 列，它包含侯选人的履历信息。我从这张表中提取数据来创建 hr 架构中的 CandidateNames 表。(我创建了一个单独的表，因为我希望可以修改表定义，从而可以增加计算列)

在建立好测试环境后，你可以创建函数。函数应该包括在从指定的 XML 列中获取数据时所需的 XQuery 表达式。例如，以下函数接收工作候选人的姓名，并保存在 JobResume 列中：

```
CREATE FUNCTION hr.FullName (@name XML)
RETURNS NVARCHAR(60) AS
BEGIN
RETURN @name.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)')
END
```

正如你所看到的，函数 FullName 带一个输入参数，该参数被定义成 XML 类型。这个做法是当调用此函数时，可以把包含所需提取的数据的 XML 列名称作为输入值来使用。

(作者：Robert Sheldon 译者：张峰 来源：TT 中国)

原文标题：使用 XML 在 SQL Server 上创建计算列（上）

链接：http://www.searchdatabase.com.cn/showcontent_27730.htm

使用 XML 在 SQL Server 上创建计算列（下）

Value()方法带两个参数。第一个参数定义了目标 XML 列使用的名称空间，第二个参数包含接收实际数据的 Xquery 表达式。在这个例子中，表达式使用 concat()方法来连接姓与名，就像它们在 XML 文件中。要想了解更多的关于如何使用 value()方法，以及如何创建 Xquery 表达式，请查看我的文章《Retrieve XML data values with XQuery》。

一旦创建了函数，你可以通过从 hr.CandidateNames 表的 JobResume 列中接收数据测试:

```
SELECT CandidateID, hr.FullName(JobResume) AS FullName

FROM hr.CandidateNames
```

正如你所看到的，我已经传入了 XML 列名称，将其做为函数 FullName 的一个参数。SELECT 语句应该返回以下结果：

CandidateID	FullName
1	Shai Bassli
2	Max Benson
3	Krishna Sunkammurali
4	Stephen Jiang
5	Thierry D'Hers
6	Christian Kleinerman
7	Lionel Penuchot
8	Peng Wu
9	Shengda Yang
10	Tai Yee

(影响 10 行)

注意，以上结果包含姓与名，正如它们在 XML 列中显示的一样。如果回到函数定义，可发现在 value()方法中使用的 Xquery 表达式指定了此表达式返回值为 NVARCHAR(60)类型，以适应 Unicode 字符，如查询结果集的最后三行中的那些字符。

一旦函数经过测试，你就可以开始创建计算列：以下 ALTER TABLE 语句添加了 FullName 列到 CandidateNames 表中来:

```
ALTER TABLE hr.CandidateNames

ADD FullName AS hr.FullName(JobResume)
```

我已经在计算列表达式中使用 FullName 函数，并将列 JobResume 作为参数传入函数。在运行 ALTER TABLE 语句后，你可以用以下 SELECT 语句测试数据是否已经被插入到计算列中：

```
SELECT CandidateID, FullName FROM hr.CandidateNames
```

运行以上语句后，应该返回与上文中相同的结果集。

这就是在 SQL Server 中基于 XML data 数据创建的一个计算列。关键是创建一个函数来运行 Xquery 表达式，且稍后在计算列中使用此函数定义。要了解更多关于计算列、XML 列、Xquery 表达式的详细信息，请查看 Microsoft SQL Server 在线书籍。

(作者：Robert Sheldon 译者：张峰 来源：TT 中国)

原文标题：使用 XML 在 SQL Server 上创建计算列（下）

链接：http://www.searchdatabase.com.cn/showcontent_27731.htm

SQL Server 中已满事务日志原因的检测（上）

对于 SQL Server 数据库管理员来讲，已满事务日志是一个琐碎的，但又很常见的问题。它能引发事务的提前终止，甚至通过阻止所有事务的引入，从而引起系统的崩溃。对于数据库管理员来说，关键是理解将要发生的情况，以便他们可以追踪引起问题的原因。

事务日志填充方式

以下是一些可能引起事务日志填满的原因：

填满的，细节的，或者没有在已满恢复模式下进行的日志备份，都会引起日志逐渐地填充。

进程中有活动的备份（备份被作为事务一样来处理），它会填充部分日志，而事务将填充剩余部分。

长时间运行的活动事务，例如从来都不会产生提交的 SPID，以及暂停或高速运行数据库镜像都会引发延迟。前者会引起事务不发送，如果在发送事务到镜像服务器之后的较长时间内，才进行高速运行，则后者才会发生。

对于事务复制，如果复制延迟或失败，事务日志将不会被清除，因为除非事务与日志都提交并发送至分布式数据库，否则事务都不能被清除。

如果进程中有一个数据库快照，当它创建时，所有的事务都堆积在它后边。

简单的响应方法

需要解决已满事务日志的问题时，你可以从以下几个选择入手：

1、你可以执行备份来消减日志。事务日志备份是最快的，但也可能是最慢的，这取决于系统性能以及日志的大小。通常不推荐填满的或细节的备份，这取决于在大小合适的系统中完成这些备份所需的时间。

2、你可以向数据库中添加额外的事务日志文件并执行以上备份方法中的一种。当你在进行必要的备份时， 额外的事务日志文件的添加可以为你赢得额外的时间。当然，它也可在稍后被删除。

3、你可以将数据库的模式切换到简单恢复模式，它将自动清除日志。但要记住，你将会丢失自最近一次已满的/细节的事务日志备份之前的事务历史。

4、正在填充日志的活动事务可以连同一些系统进程被终止，以至不会被重新执行并填充日志。当问题源被追踪到，它将提供一些缓解，但它不应该被考虑为一种解决方案。

5、在查找系统缓慢的原因时，数据库镜像/复制可以关闭。

根本原因的检测

当微软的人员在讨论减少已满事务日志的问题时，他们经常从问题的返回信息来解决问题。微软通常不会帮助你学习如何找出并解决代码问题，虽然这些代码问题是潜在的根源。

我们假定在同一数据库中运行两个事务…m1 和 n2：

```
                        Transaction 1
                        begin tran m1
                          update tbl
                       set f1 = f1 + f1
                          update tbl
                       set f1 = f1 + f1
                          update tbl
                       set f1 = f1 + f1
                      -- rollback tran m1
                        Transaction 2
                        begin tran n2
                         update tblm
set txtval = Convert（varchar（4000）， txtval + txtval）
                         update tblm
set txtval = Convert（varchar（4000）， txtval + txtval）
                         update tblm
 set txtval = Convertvarchar（4000）， txtval + txtval）
                      --rollback tran n2
```

首先，我想让大家注意一下以上事务的名字（m1 和 n2）。你会发现，如果你为事务命名，在系统中追踪，以及在查找问题原因时会变得更容易。其可读性也会更好。注意，我没有提交或回滚事务——你在应用程序中控制事务时也会领会这个作用。

执行以下 SQL 脚本，它会返回当前活动事务的列表：

```
select * from sys.dm_tran_active_transactions
```

结果集

我们当前运行的两个事务连同它们相应的名称一起出现在列表中，这使得它们很容易就被识别出来。当事务开始时，可以很容易地从屏幕上将它们区别出来，所以你可以了解到它是否过期以及应用程序，调度等在没有提交或回滚时是否被允许挂起。在事务类型列中，1 表示：“读/写”，2 表示“只读”，3 表示“系统”，4 表示“分布式”。通常，填充日志的事务是 1，但那取决于日志来源的填充。如果日志由用户进程填充，它们将是类型 1。

sys.dm_tran_session_transactions—将活动事务列表中的事务 ID 与活动的 SPID 联系起来。

sys.dm_tran_database_transactions—如果你只想查看一个数据库，它会只把与事务相关的那个数据库列出来。它也会把声明/状态列出来。数据库事务状态：1、未初始化，3、已初始化，但没有日志记录，4、产生了日志记录，5、事务准备，10、事务提交，11、事务回滚，12、事务处于即将提交的进程中。

由于已满事务日志归因于消耗空间的活动日志实体，因此只有状态 4 或状态 12 会消耗日志空间。

列 database_transaction_log_record_count 显示将要读取的日志记录，也会显示正在等待被复制的日志记录。Database_transaction_log_bytes_used 显示当前使用了多少空间，而 Database_transaction_log_bytes_reserved 则显示使用提交的事务预留空间（在这种情况下，预留的空间大小比实际使用的更重要）。

事务诊断 T-SQL 脚本—事务不必保持活动状态，只要打开它就行了。

以下 T-SQL 脚本可以在你所关注的数据库中执行。它将向你提供以下内容：

占据数据库空间的事务。

初始触发事务的 T-SQL 脚本。

事务及事务中系统使用的大小（兆字节和字节）。

事务的当前声明或状态。

连同日志编号的日志记录条数。

（作者：Matthew Schroeder 译者：张峰 来源：TT 中国）

原文标题：SQL Server 中已满事务日志原因的检测（上）

链接：http://www.searchdatabase.com.cn/showcontent_21968.htm

SQL Server 中已满事务日志原因的检测（下）

即使请求还没有被执行，此脚本仍可在事务打开后的任意时刻执行。

```
CREATE TABLE #Tmp_Transaction (
ID int identity (1, 1) ,
[TransactionName] [nvarchar] (32)  NOT NULL,
[transaction_id] [bigint] NOT NULL,
[transaction_begin_time] [datetime] NOT NULL,
[transaction_type] [int] NOT NULL,
[transaction_state] [int] NOT NULL,
[session_id] [int] NOT NULL,
[TranLog_MB_Used] [bigint] NULL,
[TranLog_MB_Reserved] [bigint] NULL,
[TranLogSys_MB_Used] [int] NULL,
[TranLogSys_MB_Reserved] [int] NULL,
[database_transaction_type] [int] NOT NULL,
[database_transaction_state] [int] NOT NULL,
[database_transaction_status] [int] NOT NULL,
[database_transaction_status2] [int] NOT NULL,
[database_transaction_log_record_count] [bigint] NOT NULL,
[database_transaction_replicate_record_count] [int] NOT NULL,
[database_transaction_log_bytes_used] [bigint] NOT NULL,
[database_transaction_log_bytes_reserved] [bigint] NOT NULL,
[database_transaction_log_bytes_used_system] [int] NOT NULL,
[database_transaction_log_bytes_reserved_system] [int] NOT NULL,
[database_transaction_begin_lsn] [numeric] (25,  0)  NULL,
[database_transaction_last_lsn] [numeric] (25,  0)  NULL,
[database_transaction_most_recent_savepoint_lsn] [numeric]
(25,  0)  NULL,
[database_transaction_commit_lsn] [numeric] (25,  0)  NULL,
[database_transaction_last_rollback_lsn] [numeric] (25,  0)  NULL,
[database_transaction_next_undo_lsn] [numeric] (25,  0)  NULL,
EventInfo nvarchar ( Max)
CREATE TABLE #inputb  (EventType nvarchar
( Max) ,  Parameters int,  EventInfo nvarchar Max) )  -- hold buffer
declare @iRwCnt int
declare @i int
```

```
declare @iSPID int
declare @vSPID varchar (4)
set @i = 1
insert into #Tmp_Transaction (TransactionName, transaction_id,

transaction_begin_time, transaction_type, transaction_state, session_id,

TranLog_MB_Used, TranLog_MB_Reserved, TranLogSys_MB_Used, TranLogSys_MB_Res
erved,

database_transaction_type, database_transaction_state, database_transaction_
status,
database_transaction_status2, database_transaction_log_record_count,

database_transaction_replicate_record_count, database_transaction_log_bytes_u
sed,
database_transaction_log_bytes_reserved,
database_transaction_log_bytes_used_system,

database_transaction_log_bytes_reserved_system, database_transaction_begin_ls
n,

database_transaction_last_lsn, database_transaction_most_recent_savepoint_lsn
,
database_transaction_commit_lsn, database_transaction_last_rollback_lsn,
database_transaction_next_undo_lsn)

select at.name [TransactionName], at.transaction_id, at.transaction_begin_ti
me,
at.transaction_type, at.transaction_state, st.session_id,
(dt.database_transaction_log_bytes_used/1048576) [TranLog_MB_Used],

(dt.database_transaction_log_bytes_reserved/1048576) [TranLog_MB_Reserved],
(dt.database_transaction_log_bytes_used_system/1048576) [TranLogSys_MB_Used
],
(dt.database_transaction_log_bytes_reserved_system/1048576)
[TranLogSys_MB_Reserved],
dt.[database_transaction_type], dt.[database_transaction_state],
dt.[database_transaction_status], dt.[database_transaction_status2],
dt.[database_transaction_log_record_count],
dt.[database_transaction_replicate_record_count],
```

```
                dt.[database_transaction_log_bytes_used],
              dt.[database_transaction_log_bytes_reserved],
            dt.[database_transaction_log_bytes_used_system],
          dt.[database_transaction_log_bytes_reserved_system],
                dt.[database_transaction_begin_lsn],
                dt.[database_transaction_last_lsn],
          dt.[database_transaction_most_recent_savepoint_lsn],

dt.[database_transaction_commit_lsn],  dt.[database_transaction_last_rollback_
                                              lsn],
                dt.[database_transaction_next_undo_lsn]
                from sys.dm_tran_active_transactions at
      inner join sys.dm_tran_session_transactions st on at.transaction_id =
                                        st.transaction_id
     inner join sys.dm_tran_database_transactions dt on at.transaction_id =
                                        dt.transaction_id
                             where dt.database_id = DB_ID
           ()  and dt.database_transaction_state in  (4, 12)  and
                             st.is_user_transaction = 1
                              set @iRwCnt = @@ROWCOUNT
                                 while @i <= @iRwCnt
                                       begin
       select @iSPID = t.session_id from #Tmp_Transaction t where t.ID = @i
                        set @vSPID = Convert (varchar, @iSPID)
                                 truncate table #inputb

INSERT #inputb EXEC  ( 'DBCC INPUTBUFFER  (' + @vSPID + ')  WITH NO_INFOMSGS'
                                              )
                                      update t
            set t.EventInfo = select top 1 EventInfo from #inputb)
                                 from #Tmp_Transaction t
                                     where t.ID = @i
                                      set @i = @i+1
                                           end

select TransactionName,  transaction_id,  transaction_begin_time,  transaction
                                          _type,
    transaction_state,  session_id,  TranLog_MB_Used,  TranLog_MB_Reserved,

TranLogSys_MB_Used,  TranLogSys_MB_Reserved,  EventInfo,  database_transaction
                                          _type,
```

```
            database_transaction_state,  database_transaction_status,
      database_transaction_status2,  database_transaction_log_record_count,

database_transaction_replicate_record_count,  database_transaction_log_bytes_u
                                    sed,
                  database_transaction_log_bytes_reserved,
                database_transaction_log_bytes_used_system,

database_transaction_log_bytes_reserved_system,  database_transaction_begin_ls
                                    n,

database_transaction_last_lsn,  database_transaction_most_recent_savepoint_lsn
                                    ,
    database_transaction_commit_lsn,  database_transaction_last_rollback_lsn,
                      database_transaction_next_undo_lsn
                          from #Tmp_Transaction
                        drop table #Tmp_Transaction
                          drop table #inputb
```

事务 T-SQL 诊断—事务必须激活执行

下一部分代码是基于以上 T-SQ 脚本来编写的，目的是提供一个完整的图像。基本上讲，它将为你提供关于动态执行事务的相关信息：

1、初始化 T-SQL 调用的相关信息。

2、当前正在执行的初始化 T-SQL 调用的潜在信息。

3、当前状态/开始时间，完成百分比。

```
    SELECT st.Session_id,  req.Blocking_Session_ID [Blocker],  req.Wait_Type,

req.Wait_Time [WaitTimeMS],  req.Wait_Resource,  req.open_transaction_count,

req.percent_complete,  dt.transaction_id,  dt.database_transaction_begin_time
                                    ,  case

when dt.database_transaction_type = 1 then 'RW' when dt.database_transaction_t
                                    ype =
```

```
2 then 'R' whendt.database_transaction_type = 3 then 'Sys' else 'Unknown' end
                                    [TranType],
      case when dt.database_transaction_state = 1 then 'Not Initialized' when
      dt.database_transaction_state = 3 then 'Initialized, but no logs' when
            dt.database_transaction_state = 4 then 'Generated logs' when
               dt.database_transaction_state = 5 then 'Prepared' when
              dt.database_transaction_state = 10 then 'Committed' when
              dt.database_transaction_state = 11 then 'Rolled Back' when

dt.database_transaction_state = 12 then 'In process of committing' else 'Unkno
                                    wn'
                                 end [TranState],
             req. Status, req.Command, stxt.objectid [ExecObjID],
         (SUBSTRING (stxt. text, req.statement_start_offset/2, ( CASE WHEN
             req.statement_end_offset = -1 then LEN (CONVERT (nvarchar
                        (max), stxt. text) ) * 2 ELSE
             req.statement_end_offset end -req.statement_start_offset)
                        /2) ) [SubText], stxt. text,
                            req.statement_start_offset
                FROM sys.dm_tran_database_transactions dt nolock)

inner join sys.dm_tran_session_transactions st nolock) on dt.transaction_id =
                                    st.transaction_id
       inner join sys.dm_exec_requests req (nolock) on st.transaction_id =
                                    req.transaction_id
             CROSS APPLY sys.dm_exec_sql_text (req. sql_handle) [stxt]
          where dt.database_id = db_id () and st.is_user_transaction = 1
```

以上为你提供了快速解决事务日志问题的相关知识与工具。既然我们已经讨论了如何检测事务日志填充的来源，我希望你可以更有效地利用事务日志来帮助其他人。

(作者：Matthew Schroeder 译者：张峰 来源：TT 中国)

原文标题：SQL Server 中已满事务日志原因的检测（下）

链接：http://www.searchdatabase.com.cn/showcontent_21971.htm

DBA 五大浪费时间的工作：收缩数据库

DBA 以常规方式执行的一些任务，不仅对 SQL Server 数据库几乎没有益处，而且实际上可能对他们的生产环境造成不利影响。在本文中，我会阐述几类这样的工作。如果你正在执行其中的一些工作，我希望你能尽快停下来。

1、收缩数据库

每天执行收缩（Shrink）数据库是一种不好的做法，有如下几个原因。从技术角度考虑，你看到的最大影响会是，每次数据库收缩之后会产生大量的索引碎片。另外，收缩数据库文件既增加了磁盘子系统的物理文件碎片，也增加了服务器的 I/O 负载，在运行收缩操作期间，会降低其他功能的性能。

现在，收缩数据库并不是一定会引起碎片。但是，因为文件本身是持续增长的，而你又一直在对它进行收缩，那么随着数据库自身的增长，数据库碎片将会变得越来越多。

如果你收缩日志文件，也有必然会再增长的坏影响，所有数据库操作在事务日志增长时都会处于暂停状态。在非常繁忙的系统中，这会花上一两秒的时间，会引起所有类型的锁定和阻塞，因为进程在等待事务日志增长。

另一个缺点是当数据库维护开始再次运行时，文件需要增长，这会占用 CPU 和磁盘资源才能完成。那么，这就会使得数据库维护时间花的更长，在 SQL Server 2000 和更早的版本中，或者在没有启用即时文件初始化设置的 SQL Server 2005 系统中尤其如此。

从管理的角度来看，这可能会给你造成一种安全错觉，因为你不知道你的数据库实际上需要多少占用空间。换句话说，如果每次你运行数据库维护进程时，你的数据库从一百 GB 增长到了一百三十 GB，然后你把它再收缩到一百 GB，你就不知道数据库实际上需要多少空间了。它需要一百 GB 还是一百三十 GB?答案是它需要一百三十 GB 空间，以便它可以执行需要的数据库维护。如果你做了收缩，然后在磁盘上放了其他数据，你不可能有足够的空间执行你的数据库维护，这项任务就会失败的。

（作者：Denny Cherry 译者：冯昀晖 来源：TT 中国）

原文标题：DBA 五大浪费时间的工作：收缩数据库

链接：http://www.searchdatabase.com.cn/showcontent_29846.htm

DBA 五大浪费时间的工作：碎片整理后重建索引

碎片整理，然后重建索引

正如你所知道的（希望如此），有两种方式清理你的数据库索引。你可以使用“REORG” 对索引进行碎片整理，或者你可以完全重建索引。实际上，在 SQL Server 2005 有新的数据库维护计划，做这两种工作都变的很容易了。

虽然这么做不会对数据库造成特别损坏，但主要是浪费时间（不是数据库维护，而是对相同的索引执行这两种操作）。这是因为两种操作执行的最终结果都是相同的，都会得到一个没有碎片的，对所有数据库页都有合适的填充因子的索引。

如果你频繁地在索引重建以后执行索引重组，那么你为了重组花费的 CPU 处理能力和磁盘 I/O 就浪费了，因为在执行完索引重建命令以后索引会被完全重建。你应该做其中一项操作，或者另一项，但是不能都做。如果你不确定该选择哪一种，有大量可以自动处理这些工作的产品可以供你购买（例如：Quest 的容量管理器，或者 Idera 的 SQL 碎片整理管理器），或者你可以在网上找到一些免费脚本。

（作者：Denny Cherry 译者：冯昀晖 来源：TT 中国）

原文标题：DBA 五大浪费时间的工作：碎片整理后重建索引

链接：http://www.searchdatabase.com.cn/showcontent_29872.htm

DBA 五大浪费时间的工作：恢复完整备份

恢复完整备份到日志传送目标

这件事希望你不是每天都在做。日志传送被某个没有完全理解事务日志怎样工作的人设置的第一征兆就是，日志传送配置被设置为每日或者每周恢复完整备份到日志传送目标服务器。

当你备份事务日志时，自上次日志备份以来的所有数据库操作都包括在内了。这包括了新增字段和表，索引重建等待。通过恢复完整备份来更新在此期间缺少的操作，相当于你只是简单地把目标数据库删掉，然后把它恢复到相同的状态，然后向前应用在全备份恢复时备份的所有日志。所做的这些都增加了日志备份遗失的概率。

（作者：Denny Cherry 译者：冯昀晖 来源：TT 中国）

原文标题：DBA 五大浪费时间的工作：恢复完整备份

链接：http://www.searchdatabase.com.cn/showcontent_29848.htm

DBA 五大浪费时间的工作：删减事务日志

删减事务日志

我在联机环境中见过的最普遍的设置之一就是下面的数据库维护计划表：

日志备份

索引重建

全备份

删减日志

每三十分钟做日志备份

在这里实际完成的是索引重建，而且执行了全备份。到目前为止，一切还算正常，不是吗?实际上，日志被删减会打断日志链，这会使所有日志备份变得无用，直到下次执行全备份。这是因为日志序列号（Log sequence Number，简称 LSN）链被删减日志操作中断了。

无论什么时候事务出现时，事务日志中就会写进去一个日志序列号。在执行备份时，第一个日志序列号和最后一个日志序列号都包含在备份中，被写到了日志备份的头位置。在日志被恢复时，日志备份中的日志序列号必须是连续的。如果他们不连续，那么 SQL Server 就认为日志记录丢失了，日志备份不能恢复了。

在这种情况下，全备份可以恢复到数据库。不幸的是，所做的日志备份没用了。这是因为包括在事务日志备份中的最后一个日志序列号与在日志删减后的第一份事务日志备份中的日志序列号不相同，因为删减日志命令改变了日志的序列号。

我所见到的另一种十分常见的场景是删减日志，然后执行全备份。这中做法会好一点，但是也强不到哪里去。如果全备份被破坏的话，任何在删减（truncate）语句和下一次全备份之间的事务都不能被恢复。这是为什么呢?因为既然删减日志步骤会重置日志序列号，你就不能从两天前恢复全备份，然后向前滚动所有事务日志。是的，把日志转换为简单恢复模式做的实际也是一样的事。

如果你打算删减你的事务日志，以便你可以进行收缩，那么请将屏幕向上滚动，把上面的内容再读一遍。

现在，如果你不需要完整的事务日志记录，而是让数据库完全恢复，那么你应该把数据库改为简单恢复模式。这种方式的事务日志不会增长，因为日志条目将被覆盖，而不是保存到下一次日志备份。

（作者：Denny Cherry 译者：冯昀晖 来源：TT 中国）

原文标题：DBA 五大浪费时间的工作：删减事务日志

链接：http://www.searchdatabase.com.cn/showcontent_29847.htm

DBA 五大浪费时间的工作：人工通读错误日志

人工通读错误日志

在一些小企业的许多 DBA 们会每天花时间通读错误日志来寻找问题。如果你只有一台或者两台服务器要管理的话，这中做法不会花太多时间。然而，如果你添加了越来越多的 SQL Server 服务器，人工浏览这些日志文件可能会花上非常长的时间。

你最好想办法以自动的方式来阅读这些日志文件并寻找错误日志。这会节约你大量时间，尤其是在日志文件增长的情况下，这样可以令你腾出时间来为能增加更多公司基线的项目工作。

如果你已经有监控解决方案了，它可能有一种读应用程序日志的方式。任何错误日志文件中的关键错误都会写到 Windows 应用日志中。如果你没有任何类型的监控程序，或者如果它不支持读错误日志，你可以加载错误日志文件和（或者）应用程序日志到一个表中，然后来查找错误。

要记住，有大量日常工作可以提升你所在组织的价值，也有另外一些工作不仅不给企业和（或者）SQL Server 增加价值，相反实际上还可能破坏基线。好的做法是，常回头看看，看看所有这些任务，每一个任务实际上都在做什么，也要看看这些任务是确实提供了成本效益（例如，备份），还是没有（人工通读日志）。

（作者：Denny Cherry 译者：冯昀晖 来源：TT 中国）

原文标题：DBA 五大浪费时间的工作：人工通读错误日志

链接：http://www.searchdatabase.com.cn/showcontent_29873.htm

利用动态管理视图提高 SQL Server 索引效率（一）

之前，我介绍了一下动态管理视图（DMV）。它是一种很有用的监控和解决 SQL Server 故障的工具。本文是它的续篇，我将继续和大家一起探讨其他的一些数据库管理员用来能够测定现存索引效率的动态管理视图（DMV）和分片级别。此外，我还提供了应该用来检索指定的 SQL Server 程序 ID 号（SPID）执行的最后语句。

DMV 提高索引效率

就如同数据库 DBA 了解的一样，合适的索引能够提高查询性能和应用程序可测量性。但是每个附加的索引，都给系统增加了额外开销，因为随着数据从表和视图中不断增加、修改或清除，SQL Server 需要维护这些索引。在安装新的索引之前，你需要检测数据库活动，保证你只有一些能提高平常执行的查询的索引。注意 SQL Server 并不能阻止你在相同的列上建立多个索引。它也不能提醒你你即将建立的查询并不能优化查询。

复制索引对系统并没有好处。同样地，SQL Server 查询优化程序不能用解决查询问题的索引也不能对系统带来什么好处。因此，在这里我们至关重要的事情就是了解索引的利用效率和它们对查询性能的影响。幸运的是，SQL Server 2005 和 2008 包括了 sys.dm_db_index_usage_stats 动态管理视图，我们可以用它来测量索引的效率。和所有其他的动态管理视图一样，contents of sys.dm_db_index_usage_stats 的内容在你重启 SQL Server 实例时就被丢弃了。所以如果你想收集索引使用统计数据，你就应该对自定义表定期复制 DMV。

每次用索引进行扫描时，DMV 就增加了在 SQL Server 中搜索或查找列。例如一下的查询就在 AdventureWorksDW 示例数据库中检索用户表和相应视图使用统计：

```
SELECT
object_name(a.object_id) AS table_name,
COALESCE(name, 'object with no clustered index') AS index_name,
type_desc AS index_type,
user_seeks,
user_scans,
user_lookups,
user_updates
```

```
FROM sys.dm_db_index_usage_stats a INNER JOIN sys.indexes b
        ON a.index_id = b.index_id
        AND a.object_id = b.object_id
WHERE database_id = DB_ID('AdventureWorksDW')
        AND a.object_id > 1000
```

有用的索引在 user_seeks 列中的总数最大。要注意 user_updates 这个列，这个列表示指定索引需要的维护级别。如果你注意到了一些用户搜索、扫描或查询很少用到但是还是会经常更新的索引，维护它们的成本就要比持有它们的成本要高。

（作者：Baya Dewald 译者：April 来源：TT 中国）

原文标题：利用动态管理视图提高 SQL Server 索引效率（一）

链接：http://www.searchdatabase.com.cn/showcontent_17875.htm

利用动态管理视图提高 SQL Server 索引效率（二）

动态管理函数（DMF）和分片索引（fragmented indexes）

数据更改会造成索引分段，高级别的分片还会减少索引的效率。结果，SQL Server 就不得不去扫描更多的索引页，甚至在用到索引时查询会变得越来越慢。为避免分片的负面影响，DBA 可以重建或对索引消除碎化。在 SQL Server 之前的版本中，你不得不用到 DBCC SHOWCONTIG 语句获取索引分片级别。这个语句还有 WITH TABLERESULTS 这个选项，它返回的结果表格形式、有序结果。

你可以想象，在一个有成千上万个表的数据库中检测每个索引肯定是一件很枯燥的工作。更不用说手动执行这项任务就等于是浪费数据库管理员的时间了。相反，许多 DBA 都实施了一个自动解决方案，这个解决方案上带有临时表、并且得到的结果为 DBCC SHOWCONTIG。然后，你就可以根据索引的分片级别对索引进行重建或消除碎化。

然是，这种方法已经过时了。虽然 DBCC SHOWCONTIG 仍然在 SQL Server 2005 或 2008 里还存在，你还是应该用 sys.dm_db_index_physical_stats 动态管理函数（DMF）。DBCC SHOWCONTIG 不支持最新版本中的新索引特征，可能不久就会被清除掉。

有了 sys.dm_db_index_physical_stats，你就不在需要创建临时表存储结果了。相反你可以在指定的时间内在定义列中用到最新的分片级别，DMF 的句法如下：

```
Sys.dm_db_index_physical_stats (
{database_id | NULL | DEFAULT | 0},
{object_id | NULL | DEFAULT | 0},
{index_id  | NULL | 0 | -1 | DEFAULT},
{partition_number | NULL | 0 | DEFAULT},
{mode | DEFAULT | NULL}  )
```

你能够在 SQL Server 联机丛书上找到每个字段的详细说明。注意在指定 DMF 的字段时，你可以用 db_id()和 object_id()这两个系统函数。

以下查询返回所有数据库所有索引的分片信息：

```
SELECT * FROM sys.dm_db_index_physical_stats(NULL, NULL, NULL, NULL, NULL)
```

第二个语句返回特定对象所有索引的索引分片级别：

```
SELECT * FROM sys.dm_db_index_physical_stats(6, 469576711, NULL, NULL, NUL
                                             L)
```

你会得到如下结果：

```
database_id  object_id  index_id  partition_number
6  469576711  1  1
6  469576711  1  1
index_type_desc  alloc_unit_type_desc  index_depth
CLUSTERED INDEX  IN_ROW_DATA  3
CLUSTERED INDEX  LOB_DATA  1
index_level  avg_fragmentation_in_percent  fragment_count
0  0.592592593  87
0  0  NULL
avg_fragment_size_in_pages  page_count
7.75862069  675
NULL  8396
avg_page_space_used_in_percent  record_count
NULL  NULL
NULL  NULL
ghost_record_count  version_ghost_record_count
NULL  NULL
NULL  NULL
min_record_size_in_bytes  max_record_size_in_bytes
NULL  NULL
NULL  NULL
avg_record_size_in_bytes  forwarded_record_count
NULL  NULL
NULL  NULL
```

尽管得到的结果很庞大，该 DMF 还是只允许你检索你感兴趣的这些列。这是 DBCC SHOWCONTIG 的另一种更新，它不允许你检索这些列的子集。

（作者：Baya Dewald 译者：April 来源：TT 中国）

原文标题：利用动态管理视图提高 SQL Server 索引效率（二）

链接：http://www.searchdatabase.com.cn/showcontent_17876.htm

利用动态管理视图提高 SQL Server 索引效率（三）

检索目前执行的 SQL 语句

许多 DBA 已经用过 SQL Server 之前版本的 DBCC INPUTBUFFER 命令来获取已给出的链接执行的最后那个 SQL 语句。但是这个语句之返回了该语句最后的 255 个字符，可能不是整个句子。SQL Server 2005 和 2008 提供了检索该信息的几个选项。sys.dm_exec_sql_text 和 sys.dm_exec_requests DMV 应该通力合作，并且 fn_get_sql() 也应该和同一个 sys.dm_exec_requests DMV 进行关联，这是另一个获取最后语句的选项。

以下是示例：

```
To find a statement executed by a specific session (53):
                        SELECT
  SUBSTRING(b.text,  (a.statement_start_offset/2) + 1,
                ((CASE statement_end_offset
                 WHEN -1 THEN DATALENGTH(b.text)
                ELSE a.statement_end_offset END
          - a.statement_start_offset)/2) + 1) AS statement_text
                FROM sys.dm_exec_requests a
           CROSS APPLY fn_get_sql (a.sql_handle) b
                WHERE a.session_id = 53
To get SQL statements submitted by all running or suspended sessions:
                        SELECT
                    a.session_id,
                      a.status,
                    a.start_time,
                      a.command,
  SUBSTRING(b.text,  (a.statement_start_offset/2) + 1,
                ((CASE statement_end_offset
                 WHEN -1 THEN DATALENGTH(b.text)
                ELSE a.statement_end_offset END
          - a.statement_start_offset)/2) + 1) AS statement_text
                FROM sys.dm_exec_requests a
        CROSS APPLY sys.dm_exec_sql_text(a.sql_handle) b
          WHERE a.status IN ('running',  'suspended')
```

你可以在 SQL Server 联机丛书上找出 fn_get_sql， sys.dm_exec_requests 和 sys.dm_exec_sql_text 的详细信息。以上查询用 statement_start_offset 和 statement_end_offset 列只检索目前执行的 SQL 语句，即使是存储程序或自定义函数中已经付带了。如果我们已经用 DBCC INPUTBUFFER，那我们应该只能获取存储程序或自定义函数名称和执行参数。

（作者：Baya Dewald 译者：April 来源：TT 中国）

原文标题：利用动态管理视图提高 SQL Server 索引效率（三）

链接：http://www.searchdatabase.com.cn/showcontent_17877.htm

在 SQL Server tempdb 满时检查数据文件

作为一名数据库 DBA，肯定会听说过“tempdb 数据库满了”。通常我们很容易确定造成这一问题的原因。但是更多的时候这一问题主要源于一组请求，涉及到新代码部署或逐渐增加的数据。

“Tempdb 满了”意味着什么？

当 SQL Server tempdb 满了时，上层管理常常需要决策、一些开发人员可能会推卸责任，就连高级 DBA 也害怕碰到这种情况。

和我告诉管理员的一样，首先经验的做法就是：保持冷静。不要让还没有公布的情况给其他方面造成压力，那样可能酿成更大的错误。

既然情况已经出现了，那我们就来解决问题。Tempdb 数据库由两部分组成：一是原始文件组里的数据文件，二是 tempdb 日志文件。这两者都可能出错，但错误信息会告诉你哪一部分满了。首先我们一起看看数据文件部分。在以后的文章部分中再讲解日志文件。

我们怎么压缩源文件？

首先我们要了解一下确定是什么占用大部分空间的方法，哪一个服务器有我们处理的 ID 号（SPID）、请求是从哪一台主机上发出的。以下查询将返回数据库里占空间的前 1000 个 SPID。记住这些返回的值为页码数。为此，我算了一下存储值（单位为 MB）。同样，我们还要注意计数器是随着 SPID 的使用时间而逐渐积累的：

```
SELECT top 1000
    s.host_name, su.[session_id], d.name [DBName], su.[database_id],
su.[user_objects_alloc_page_count] [Usr_Pg_Alloc], su.[user_objects_deallo
c_page_count] [Usr_Pg_DeAlloc],
su.[internal_objects_alloc_page_count] [Int_Pg_Alloc], su.[internal_object
s_dealloc_page_count] [Int_Pg_DeAlloc],
(su.[user_objects_alloc_page_count]*1.0/128) [Usr_Alloc_MB], (su.[user_obj
ects_dealloc_page_count]*1.0/128)
[Usr_DeAlloc_MB],
(su.[internal_objects_alloc_page_count]*1.0/128) [Int_Alloc_MB], (su.[inte
```

```
rnal_objects_dealloc_page_count]*1.0/128)
                [Int_DeAlloc_MB]
        FROM [sys].[dm_db_session_space_usage] su
    inner join sys.databases d on su.database_id = d.database_id
inner join sys.dm_exec_sessions s on su.session_id = s.session_id
        where (su.user_objects_alloc_page_count > 0 or
            su.internal_objects_alloc_page_count > 0)
order by case when su.user_objects_alloc_page_count > su.internal_objects_
                alloc_page_count then
su.user_objects_alloc_page_count else su.internal_objects_alloc_page_count end
                desc
```

第二个查询也非常类似，它返回的是 SPID 给分配空间的前 1000 条。该查询能跟踪可以循环、创建项目或运行时创建、删除多个临时对象的程序。

```
SELECT top 1000 s.host_name, su.[session_id], d.name [DBName], su.[database_i
                d],
    su.[user_objects_alloc_page_count] [Usr_Pg_Alloc], su.[user_objects_deallo
                c_page_count] [Usr_Pg_DeAlloc],
    su.[internal_objects_alloc_page_count] [Int_Pg_Alloc], su.[internal_object
                s_dealloc_page_count] [Int_Pg_DeAlloc],
    (su.[user_objects_alloc_page_count]*1.0/128) [Usr_Alloc_MB], (su.[user_obj
                ects_dealloc_page_count]*1.0/128)
                [Usr_DeAlloc_MB],
    (su.[internal_objects_alloc_page_count]*1.0/128) [Int_Alloc_MB], (su.[inte
                rnal_objects_dealloc_page_count]*1.0/128)
                [Int_DeAlloc_MB]
        FROM [sys].[dm_db_session_space_usage] su
    inner join sys.databases d on su.database_id = d.database_id
inner join sys.dm_exec_sessions s on su.session_id = s.session_id
        where (su.user_objects_dealloc_page_count > 0 or
            su.internal_objects_dealloc_page_count > 0)
order by case when su.user_objects_dealloc_page_count > su.internal_object
                s_dealloc_page_count then
    su.user_objects_dealloc_page_count else su.internal_objects_dealloc_page_c
                ount end desc
```

由于 tempdb 在压缩后没有报告它的大小，以下查询可以提供 tempdb 里的有用空间。

```
SELECT sum(unallocated_extent_page_count) [Free_Pages],
(sum(unallocated_extent_page_count)*1.0/128) [Free_Space_MB]
FROM sys.dm_db_file_space_usage
```

如果你已经决定了 SPID，你就可以决定用 dbcc 缓冲器（SPID）运行什么样的 T-SQL。

假设你清楚运行的 T-SQL 代码，但是你还需要知道会牵涉到的临时表。你可以执行以下程序：

```
select * from tempdb.sys.objects where type = 'u'
```

临时表源于 T-SQL 里那些应该有#YourDefinedTblName____UniqueID 格式的用户。它能帮你识别涉及到的代码。你还可以用 sys.dm_exec_requests 命令联结 SPID、用 sys.dm_exec_sql_text(SQL_Handle)获取当时运行的命令，但要求脚本在实际运行时用“polling loop”监控。

小结

在现有的系统表和视图的基础上，我们很难在没有预先准备的基础上解决问题。充满的 tempdb 有时可以像单个 SPID 那么简单，有时像一组会话一样复杂，但是上面我所概述的这些步骤帮你将问题化小。

（作者：Matthew Schroeder 译者：April 来源：TT 中国）

原文标题：在 SQL Server tempdb 满时检查数据文件

链接：http://www.searchdatabase.com.cn/showcontent_20287.htm

SQL Server 运作的简短课程

面对现实吧，虽然你从来没有打算成为一名 SQL Server 专家，但是随着数据库引擎种类和版本的增加，这就要求一些人来专门从事并关注这方面的内容。作为“微软人”（或者称为 Gal）,无论你是不是愿意，你都被选中了。这一系列的文章全都是关于帮助作为管理员而非程序员的你在使用 SQL Server 时更加高效。

在实际执行任务之前，有一点背景信息可以会起到帮助作用。那么，到底 SQL Server 是如何工作的呢?不管你信不信，理解“黑盒”知识几乎可以在 Microsoft SQL Server 的所有方面起到帮助作用，例如从备份与存储到复制与镜像。

SQL Server 将数据存储在磁盘中 8KB 大小的块中，称为页。在内存中,SQLServer 操作的也是那些 8KB 大小的块，这意味着 SQL Server 中处理的数据最小单元也是 8KB。

当数据写入磁盘时，每一整行数据必须符合 8KB 大小的页。SQL Server 允许多行数据共享一个页,但是不允许一行数据跨多个页。因此，如果一个客户表包含列：Name,Address,City,State,以及 Phone,那么，所有的数据组合必须小于 8KB。对于某种特定的数据类型来说则有一个例外，这个时候，实际的页只包含对真实数据的指针，如二进制数据（图片或文本大字段），其真实数据可以存储在多个页上，或者存储在一个文件中（那是特殊的 FILESTREAM 类型）。SQL Server 将这所有的这些 8KB 大小的页收集在一起放入磁盘的一个简单文件中，这种文件通常有一个.MDF 或.NDF 的文件扩展名。

当 SQL Server 被告知要做什么时，它是通过由结构化查询语言（SQL）语法写的查询来实现的。以下是最先发生的：SQL Server 的内部查询优化器监视着此查询并构造一个处理计划来执行它（例如：指出从磁盘中取出数据所要遵循的步骤）。这实际上相当复杂，因为 SQL Server 有大量可使用的技术，而且部分技术在某特定条件下比其它的要好。

一旦 SQL Server 构造成功此计划，它将执行此计划并从磁盘取出需要的数据。如果接收到查询，数据将通过网络流动到正在请求的客户端。如果更改了查询，SQL Server 则会修改内存中页的数据，但不会将修改写回磁盘。那有点愚蠢，因为可能在页上还有其它的随之发生的修改，并且系统加载时可能不提供一个很好的向磁盘写数据的机会。然而，SQL Server 所做的是生成被修改查询的一个副本，并将其保存在一个指定的事务日志文件中。这个文件有一个.LDF 的扩展文件名，保存着 SQL Server 执行的每一个事务的记录。

最后---也许几秒钟以后---SQL Server 决定把修改后的页写到磁盘中。当它这样处理时，它查找事务日志并取消产生修改的事务。从本质上讲，也就是“OK，我做了修改并且此修改已经写入磁盘。”这样一来，SQL Server 知道这些修改在磁盘中是安全的。

在 SQL Server 垮掉的情况下,它有自一个动恢复模式,可在开始备份时进行切换。它直接打开事务日志并查找未提交的或者未取消的事务。众所周知，当服务器死机时,未取消的事务在磁盘中是安全的，其它任何数据没有被写到磁盘中，而是仍然存留在内存中。因此,SQL Server 从那些事务日志文件中读取日志,重新执行它们,并迅速将受影响的页写入磁盘。这个过程允许 SQL Server 捕获进程中的所有操作，并确保你不会丢失任何数据——提供给你的磁盘文件是完好的，当然，现在想一想这个重要事实—SQL Server 中发生的每一事件只通过事务日志产生，并且 SQL Server 可以重读日志来重现所发生的事务。这个过程使得 SQL Server 几乎可以实现每一件事情。

当然，这仅仅是默认情况，你可以修改它。个人数据库可被从完全恢复模式（我已经在前边描述过）切换到简单恢复模式，简单恢复模式不使用事务日志（好吧，它使用，但取消的事务会被自动移除从而保持日志文件比较小）。简单恢复仅适用于那些没有被修改的只读数据库。没有被修改，在死机中就不会丢失数据。

那就是数据如何从磁盘向内存中移动的过程。这整个过程绝对是 SQL Server 的大多数功能实际工作的本质,比如如何管理它。在我的下一篇文章中，我将关注 SQL Server 中的灾难恢复是如何处理的，以及如何才能为你的数据库实现一个合理的，安全的灾难恢复计划。

（作者：Don Jones 译者：张峰 来源：TT 中国）

原文标题：SQL Server 运作的简短课程

链接：http://www.searchdatabase.com.cn/showcontent_22301.htm

实现 SQL Server 2008 中的文件流功能（上）

SQL Server 2008 中最新的文件流功能使得你可以配制一个数据类型为 varbinary（max）的列，以便将实际数据存储在文件系统中，而非在数据库中。只要愿意，你仍可以作为一个常规的二进制列来查询此列，即使数据自身存储在外部。

文件流特性通过将二进制大字段数据存储在本地文件系统中,从而将 Windows 新技术文件系统（NTFS）和 SQL Server 数据库引擎集成在一起。你可以使用 Transact-SQL 语句来查询、插入或更新数据，或者使用 Win32 文件系统界面来直接访问数据。

微软建议你仅在以下几种情况下使用文件流:（1）你的二进制大字段数据文件平均大于 1M,（2）你需要快速读取此数据,（3）你的应用程序使用中间列来处理应用逻辑。否则,你应该使用常规的 varbinary（max）列。

要激活 SQL Server 2008 服务中的文件流支持，需要遵照以下几步：

1、配制数据库来支持文件流存储。

2、定义支持文件流存储的列。

3、启动 SQL Server 服务中的文件流支持

要激活 SQL Server 2008 中指定实例的文件流支持,你必须首先配制此实例的 SQL Server 服务。在 SQL Server 配制管理器中，打开服务属性并选择“文件流”页签，如图所示。

你至少需要选上“启动 Transact-SQL 文件流访问”勾选框。因为插入及更新数据最有效的方法是通过 Win32 界面，然而，你也需要激活服务来支持文件流（如果有必要，也可以启动允许远程客户端访问文件流数据）。

在你激活了 SQL Server 服务中的文件流支持后，必须设置文件流访问级别，你可以在 SQL Server 管理器中设置。要设置访问级别，需执行以下 T-SQL 语句：

```
EXEC sp_configure filestream_access_level, 2
GO
RECONFIGURE
GO
```

在这里，我使用系统存储过程 sp_configure 将访问级别设为 2，这个级别可同时支持 T-SQL 和 Win32 流访问。如果我想只是支持 T-SQL 访问，则需要将访问级别设为 1。如果设置为 0，将会禁用 SQL Server 实例的文件流支持。在你运行存储过程后，需要运行 RECONFIGURE 命令来应用新的选项设置。

配制数据库来支持文件流存储

支持文件流存储的下一步操作是向数据库定义中添加一个文件流文件组。文件流文件组是一个特殊的文件组类型，它包含文件系统目录（数据容器）。例如，在下边的数据库定义中，我创建了一个名为 FileStreamGrp 的文件流文件组。

```
USE master
GO
IF EXISTS
SELECT name FROM sys.databases
WHERE name = 'HumanResources')
DROP DATABASE HumanResources
GO
CREATE DATABASE HumanResources
ON
PRIMARY (
NAME = HumanRscsDat,
FILENAME = 'C:\Data\HR\HumanRscsDat.mdf'),
FILEGROUP FileStreamGrp CONTAINS FILESTREAM (
NAME = HumanRscsFs,
FILENAME = 'C:\Data\HR\FileStream')
LOG ON (
NAME = HumanRscsLog,
FILENAME = 'C:\Data\HR\HumanRscsLof.ldf')
```

（作者：Robert Sheldon 译者：张峰 来源：TT 中国）

原文标题：实现 SQL Server 2008 中的文件流功能（上）

链接：http://www.searchdatabase.com.cn/showcontent_21861.htm

实现 SQL Server 2008 中的文件流功能（下）

注意,文件流文件组定义包括“文件流关键字”，后边跟着逻辑名与文件名。在这里，此文件名仅仅是一个目录路径，没有一个真实的名称。当你指定了路径，每个对象（除了最深的一个）必须存在，且最深的一个不存在。举个例子，目录 C:\DATA\HR 必须在你运行此语句之前存在,但是 C:\Data\HR\FileStream 不能存在。

当你向数据库定义中增加一个文件流文件组，SQL Server 将自动创建必要的文件夹及 filestream.hdr 文件（这个文件是文件流容器的头文件）和$FSLOG 文件夹（支持文件流日志）。

定义支持文件流存储的列

设置文件流存储的下一步是配制文件流列。要使一张表包含一个文件流列，它必须也要包含一个 ROWGUIDCOL 关键字，且此关键字需要配制为非空和唯一约束。这个文件流列对于支持 Win32 文件流访问来说是必须的。

此文件流列本身必须配制为 varbinary（max）类型，并包含 FILESTREAM 关键字，如下边的建表语句:

```
USE HumanResources
GO
IF OBJECT_ID （ N'Candidate', N'U'） IS NOT NULL
DROP TABLE dbo.Candidate
GO
CREATE TABLE Candidate（
CandidateId INT IDENTITY PRIMARY KEY,
CandidateGuid UNIQUEIDENTIFIER ROWGUIDCOL
NOT NULL UNIQUE DEFAULT NEWID（）,
CandidateResume VARBINARY（MAX） FILESTREAM NULL
```

正如你看到的，列 CandidateResume 包含 FILESTREAM 关键字，它在数据类型名称之后。

查询文件流列

一旦你建立了 SQL Server 来支持文件流存储，就可以使用 T-SQL 语句来查询并修改数据。例如，以下的插入语向 CandidateResume 列中添加二进制数据。

```
INSERT INTO Candidate（CandidateResume）
VALUES （CAST（
’Resume test data’ AS VARBINARY（MAX）））
```

然后你可以获取 CandidateResume 列返回的数据，就好像你从任何其它列获取数据一样。

```
SELECT CandidateResume
FROM Candidate
WHERE CandidateId = 1
```

此 SELECT 语句返回以下二进制数据集：0x526573756D6520746573742064617461

你也可以轻松地通过替换值来更新数据：

```
UPDATE Candidate
SET CandidateResume =
CAST（ ’New resume test data’ AS VARBINARY（MAX））
WHERE CandidateId = 1
```

注意，我传递进去的是一个二进制值。如果你重新执行上边的 SELECT 语句，你现在会获以下结果：

0x4E657720726573756D6520746573742064617461

正如你在前边的例子中看到的，使用 T-SQL 语句来访问列 CandidateResume 的数据是一个便捷的过程。然而，很显然，我插入并更新的测试数据要比你在一般情况下存储在 FILESTREAM 列中的数据要小很多。实际上，你通常会希望使用 Win32 文件流来访问数据。

为了支持 Win32 文件流，SQL Server 2008 提供函数和 API，可使得从你的应用程序访问。尽管关于 Win32 文件流不是本篇文章讨论的范围之内，但了解 SQL Server2008 中文件流功能轻松地支持从应用程序高效数据访问是很重要的。

获得更多关于文件流数据的 Win32 流细节信息，请参考 SQL Server 2008 在线图书中的主题：通过使用 Win32 来管理文件流数据。你通常也可以在主题文件流存储的设计与实现中获得更多关于文件流存储的细节信息。同时，你目前应该已经有了当你在 SQL Server 实例中建立文件流存储并定义支持文件流存储的列时所需要的相关细节信息。

（作者：Robert Sheldon 译者：张峰 来源：TT 中国）

原文标题：实现 SQL Server 2008 中的文件流功能（下）

链接：http://www.searchdatabase.com.cn/showcontent_21865.htm