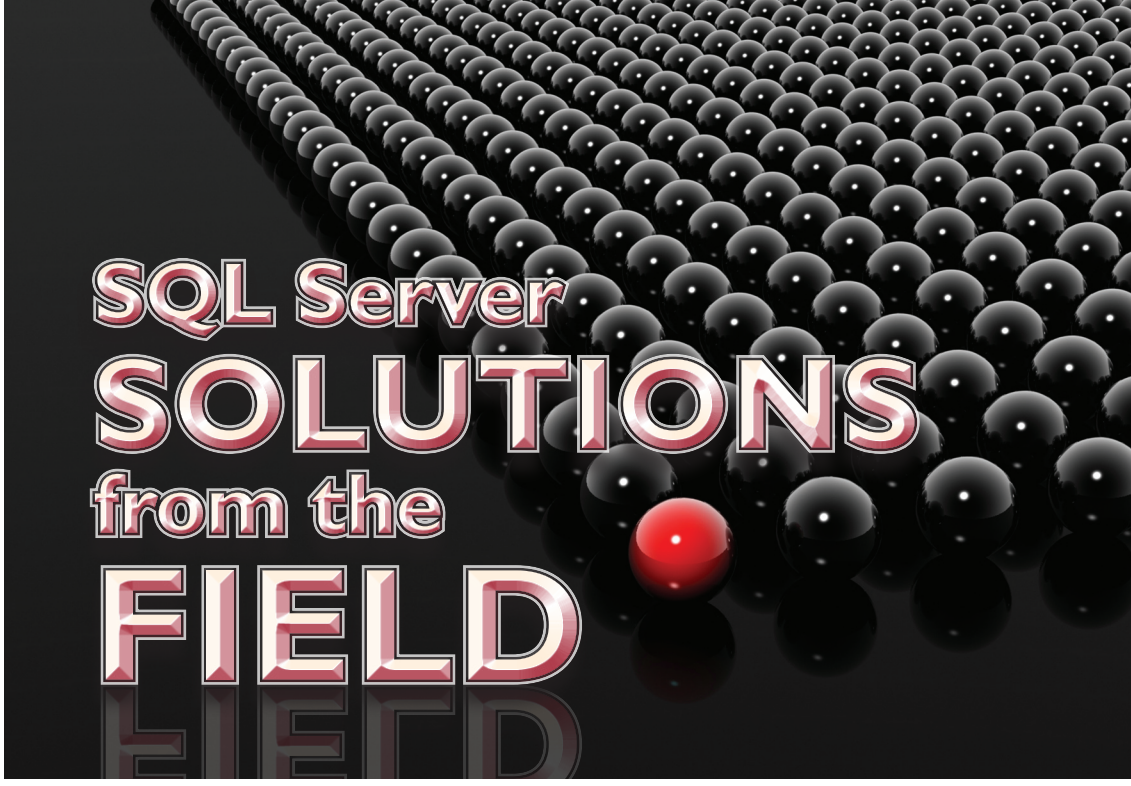

At *SQL Server Magazine*, we believe in bringing you field-tested, practical solutions. Reader to Reader content puts experts like you directly in contact with the solutions other reader-experts have found. We're proud to feature four Reader to Reader solutions this month as November's cover story. We love receiving your practical solutions from the field, and I encourage all of you to send us your favorite solutions to the hard SQL Server problems you've encountered.

—Sheila Molnar,
executive editor,
SQL Server Magazine

THREE-IN-ONE Database Maintenance SOLUTION

Back up databases, check their integrity, and optimize indexes

Ola Hallengren

DBA at a Nordic investment bank, Sweden, ola@hallengren.com, ola.hallengren.com

Maintaining databases is an important but time-consuming job. I wrote a script, MaintenanceSolution.sql, that lets you easily perform three common maintenance tasks: backing up databases, checking the integrity of databases, and optimizing indexes. The script is set up so that you can call its three main stored procedures—DatabaseBackup, IndexOptimize, and DatabaseIntegrityCheck—separately. Thus, you can perform each task independently on the databases you specify.

MaintenanceSolution.sql is not only easy to use but also offers features such as dynamic index optimization. Rebuilding and reorganizing indexes dynamically has a number of advantages. It lets you rebuild or reorganize only the indexes that need to be rebuilt or reorganized, so locking and blocking are minimized and fewer system resources are used. This, in turn, can result in higher availability and reduce the size of differential and transaction log backups.

MaintenanceSolution.sql works on the Standard, Enterprise, Workgroup, Express, and Developer Editions of SQL Server 2008 and SQL Server 2005 SP2 running on X86, X64, or IA64 platforms. The solution is supported on the same OSs that SQL Server supports. You can download MaintenanceSolution.sql by going to www.sqlmag.com, entering 100178 in the InstantDoc ID text box, and clicking the 100178.zip hotlink. Aternatively, you can download the script at http://blog.ola.hallengren.com/blog/_archives/2008/1/1/3440068.html.

How to Back Up Databases

DatabaseBackup is the stored procedure in MaintenanceSolution.sql that performs database backups. When you call this stored procedure, it creates a backup directory, performs a database backup, verifies the backup file, then deletes any old backup file for each database you specify. You specify the database to back up and other details using an EXECUTE statement such as

```
EXECUTE dbo.DatabaseBackup
@Databases = 'USER_DATABASES',
@Directory = 'C:\Backup',
@BackupType = 'FULL',
```

```
@Verify = 'Y',
@CleanupTime = 24
```

This statement tells DatabaseBackup to perform a full backup of all user databases, verify the backups, then delete any backup files older than 24 hours.

Let's take a closer look at the EXECUTE statement. You use the @Databases parameter to specify the databases you want to back up. You can select all user databases by specifying 'USER_DATABASES' or select all system databases by specifying 'SYSTEM_DATABASES'. You can exclude databases from the user databases using the syntax

```
@Databases =
  'USER_DATABASES, -Database1, -Database2'
```

where *-Database1* and *-Database2* are the names of the databases you want to exclude. With this setup, you can, for example, have a backup strategy for one database and another backup strategy for all other databases.

Alternatively, you can back up an individual database by specifying the database's name, following the syntax

```
@Databases = 'Database1'
```

To back up two or more individual databases, follow the syntax

```
@Databases = 'Database1, Database2'
```

Besides the DatabaseBackup stored procedure, the IndexOptimize and DatabaseIntegrityCheck stored procedures use the @Databases parameter.

The @BackupType parameter identifies the backup type. The stored procedure can perform full backups ('FULL'), differential backups ('DIFF'), or transaction log backups ('LOG') on the databases you specify.

You use the @Directory parameter to specify the backup root directory, under which DatabaseBackup creates a directory structure consisting of the SQL Server instance name, database name, and backup type. So, for example, if your SQL Server instance is Server1$Instance1 and you specify 'AdventureWorks' for the @Databases parameter, 'C:\Backup' for the @Directory parameter, and 'FULL' for the @BackupType parameter, the stored procedure creates the directory structure C:\Backup\Server1$Instance1\AdventureWorks\FULL. Each backup file's name consists of the SQL Server instance name, database name, backup type, and date and time of the backup. In this example, the filename would be Server1$Instance1_AdventureWorks_FULL_20080904_000001.bak.

The last two parameters are @Verify and @CleanupTime. The @Verify parameter controls whether the backup file should be verified ('Y') or not ('N'). The @CleanupTime parameter specifies when to delete old backup files, assuming the backup and the verification (if that was selected) was successful.

How to Optimize Indexes

Although there aren't any set rules when it comes to determining when and how to optimize indexes, there are some basic concepts to guide you:

- Whether an index should be rebuilt, reorganized, or left untouched depends on the index's fragmentation level. Highly fragmented indexes (typically more than 30 percent fragmented) should be rebuilt. Indexes with little fragmentation (typically less than 5 percent fragmented) should be left alone. Moderately fragmented indexes (typically between 5 percent and 30 percent fragmented) should be reorganized. (For more information about this concept, see "Reorganizing and Rebuilding Indexes" in SQL Server 2008 *Books Online—BOL*—at msdn.microsoft.com/en-us/library/ms189858.aspx.)
- Index fragmentation in a very small table has no impact on performance.
- When an index is rebuilt, the statistics are always rebuilt and therefore updated. However, when an index is reorganized, the statistics aren't updated. Therefore, you might want to update the statistics after an index is reorganized.
- An index can be rebuilt online or offline. An offline rebuild is faster than an online rebuild. (Note that only the Enterprise and Developer Editions of SQL Server 2008 and SQL Server 2005 support online rebuilds.)
- When an index contains a large object (LOB) column, an online rebuild can't be done.

MORE on the WEB

Download the code at InstantDoc IDs 100178, 100201, 100213, and 100217

The IndexOptimize stored procedure incorporates these concepts into its logic. This stored procedure categorizes indexes into six groups based on their level of fragmentation (high, medium, or low) and whether LOB columns are present (LOB) or not (NonLOB). The parameters that represent these groups are as follows:

- @FragmentationHigh_LOB represents indexes with a high fragmentation level and LOB columns.
- @FragmentationHigh_NonLOB represents indexes with a high fragmentation level and no LOB columns.
- @FragmentationMedium_LOB represents indexes with a medium fragmentation level and LOB columns.
- @FragmentationMedium_NonLOB represents indexes with a medium fragmentation level and no LOB columns.
- @FragmentationLow_LOB represents indexes with a low fragmentation level and LOB columns.

Whether you should rebuild an index, reorganize it, or leave it untouched depends on the index's fragmentation level.

LISTING 1: Sample Execute Statement for the IndexOptimize Stored Procedure

```
EXECUTE dbo.IndexOptimize
@Databases = 'USER_DATABASES',
@FragmentationHigh_LOB = 'INDEX_REBUILD_OFFLINE',
@FragmentationHigh_NonLOB = 'INDEX_REBUILD_ONLINE',
@FragmentationMedium_LOB = 'INDEX_REORGANIZE_STATISTICS_UPDATE',
@FragmentationMedium_NonLOB = 'INDEX_REORGANIZE_STATISTICS_UPDATE',
@FragmentationLow_LOB = 'NOTHING',
@FragmentationLow_NonLOB = 'NOTHING',
@FragmentationLevel1 = 5,
@FragmentationLevel2 = 30,
@PageCountLevel = 1000
```

- @FragmentationLow_NonLOB represents indexes with a low fragmentation level and no LOB columns.

IndexOptimize uses the avg_fragmentation_in_percent column in the sys.dm_db_index_physical_stats dynamic management view (DMV) to obtain the percentage of fragmentation in each index. Using the threshold limits you set in the @FragmentationLevel1 (lower threshold) and @FragmentationLevel2 (upper threshold) parameters, it places each index in the appropriate group. Indexes with fragmentation levels higher than the upper threshold go into one of the high fragmentation groups. Indexes with fragmentation levels at or between the two thresholds go into one of the medium fragmentation groups. Indexes with fragmentation levels under the lower threshold go into one of the low fragmentation groups.

```
DateTime: 2008-09-04 11:38:10
Command: ALTER INDEX [IX_Customer_TerritoryID] ON
[AdventureWorks].[Sales].[Customer] REORGANIZE
Comment: IndexType: NonClustered, LOB: No,
PageCount: 1001, Fragmentation: 6.88235
Outcome: Succeeded
DateTime: 2008-09-04 11:38:21
```

Figure 1

Sample log entry for the IndexOptimize stored procedure's ALTER INDEX command

IndexOptimize can also place indexes in one of the low fragmentation groups based on their page count. Indexes under the size specified in the @PageCountLevel parameter go into one of the low fragmentation groups.

For each group, you can select one of the following actions:

- 'INDEX_REBUILD_ONLINE'—Tells the stored procedure to rebuild the indexes online. (You need to be running the Enterprise or Developer Edition of SQL Server 2008 or SQL Server 2005 to use this option.)
- 'INDEX_REBUILD_OFFLINE'—Tells the stored procedure to rebuild the indexes offline.
- 'INDEX_REORGANIZE'—Tells the stored procedure to reorganize the indexes.
- 'INDEX_REORGANIZE_STATISTICS_UPDATE'—Tells the stored procedure to reorganize the indexes and update the statistics.
- 'STATISTICS_UPDATE'—Tells the stored procedure to update the statistics.
- 'NOTHING'—Tells the stored procedure to do nothing to the indexes.

So, for example, the EXECUTE statement in Listing 1 tells IndexOptimize to rebuild indexes that are more than 30 percent fragmented, online if possible (no LOBs). If these highly fragmented indexes have LOBs, an offline rebuild is to be done. Indexes with a fragmentation level between 5 percent and 30 percent are to be reorganized and have their statistics updated. Indexes that are less than 5 percent fragmented or have fewer than 1,000 pages aren't to be touched. IndexOptimize uses T-SQL's ALTER INDEX command to rebuild and reorganize indexes.

How to Check Databases' Integrity

The DatabaseIntegrityCheck stored procedure uses T-SQL's DBCC CHECKDB command to perform integrity checks. Using this stored procedure instead of the Database Maintenance Plan Wizard's Check Database Integrity Task might mean that you don't have to install the hotfix for bug 50001012. (The Check Database Integrity Task can lose database context under certain circumstances in SQL Server 2005 builds 3042 through 3158—see support.microsoft.com/kb/934458.)

The EXECUTE statement you use to run DatabaseIntegrityCheck is simple. You just need to specify the databases you want to check. For example, the statement

```
EXECUTE dbo.DatabaseIntegrityCheck
@Databases = 'USER_DATABASES'
```

checks the integrity of all user databases.

All Types of Information Are Readily Available

The DatabaseBackup, IndexOptimize, and DatabaseIntegrityCheck stored procedures have thorough logging and error handling. The start and end time, command text, and output are logged for each command in the stored procedures. Additional information is logged for the IndexOptimize's ALTER INDEX command, as Figure 1 shows. All command information is immediately written to a log file. You can find information about how the stored procedures handle errors in MaintenanceSolution.sql's documentation at blog.ola.hallengren.com/_attachments/3440068/Documentation.html. The documentation also includes information about how to use each stored procedure as well as answers to frequently asked questions.

InstantDoc ID 100178