SQL Server Backup und Recovery ohne teure Tools

SQL Server Backup und Recovery ohne teure Tools

[expand title="mehr lesen…"]

Selbstsicherung

Thorsten Kansy

Das Sichern von Datenbanken beherrscht Microsofts SQL Server mit Bordmitteln. Eine gute Kenntnis der Grundlagen und Fallstricke vorausgesetzt, stellt der Administrator im Ernstfall die Daten schnell und zuverlässig wieder her.

iX-TRACT

Microsofts Datenbank SQL Server bringt Konzepte und Werkzeuge fürs Sichern und Wiederherstellen einer Datenbank bereits mit.

SQL-Server-Backups lassen sich ins Unternehmens-Backup integrieren, der Kauf zusätzlicher Tools ist nicht unbedingt notwendig.

Mehrere Backup-Methoden, Kompression und Verschlüsselung stehen – je nach Version und Edition – zur Auswahl.

Microsofts relationales Datenbankmanagementsystem SQL Server bietet von Haus aus genügend Tools und Fähigkeiten, die ohne zusätzliche kostenpflichtige Software Datenbanken sichern und

wiederherstellen. Backups können bedarfsweise komprimiert und verschlüsselt werden – Automatisierung inklusive. Der Artikel gibt einen Überblick der Backup-Varianten und verrät Tipps für den Produktivbetrieb. Besonderheiten beim Sichern des SQL Server in virtuellen Maschinen sind jedoch nicht Thema.

Generell sollte man für ein Backup einen Zeitpunkt wählen, an dem der Server und die gesamte Infrastruktur (Netz, Storage) über möglichst viele Ressourcen verfügen. Dennoch kann ein Backup jederzeit stattfinden. SQL Server stellt sicher, dass der Inhalt des Backups exakt dem Zustand zum Start der Sicherung entspricht und nur freigegebene Transaktionen enthält. Dazu führt das System eine Log Sequence Number (LSN), mit der es jede Transaktion und jedes Backup versieht.

Zeitpunkt und Datenkonsistenz entscheiden

Während eines Backups durchläuft die Software die interne Struktur einer Datenbank, die auf 8 KByte großen Seiten basiert oder aus Stream-Daten besteht, und sichert nur belegte Bereiche. Zugleich überprüft sie, gewissermaßen als Nebeneffekt, die Konsistenz – ein weiterer Grund für ein Backup.

Der Backup-Dialog von SQL Server: Die physischen Daten liegen in mehreren Dateien, die in Filegroups organisiert sind (Abb. 1).

Auf die Frage, was gesichert werden soll, lässt sich nicht schlicht mit „meine Datenbank“ antworten. Viele Datenbanken bestehen aus Performancegründen oder wegen ihrer schieren Größe aus mehreren Dateien, die in File-Gruppen organisiert sind – Letztere sind der physische Speicherort der Daten. Dazu kommen Dateien für das Transaktionsprotokoll, das alle Änderungen an der Datenbank vor dem Umsetzen speichert.

Daher muss die korrekte Antwort auf die Frage lauten: „Die Inhalte meiner Datenbanken“ – ein wichtiger Unterschied. Das

bedeutet im Detail, dass für ein komplettes Backup alle Datendateien inklusive der FileStream-Daten (beispielsweise aus dem FileTable-Feature) oder das Transaktionsprotokoll gesichert werden müssen. Letzteres lässt sich später jedoch nur wieder einspielen, wenn man auf eine davorliegende Sicherung von Daten zugreifen kann.

Außerdem ist das Sichern der eigenen Datenbank(en) meist zu kurz geplant. So enthält etwa die *master*-Datenbank alle Log-ins, was insbesondere beim Einsatz der SQL-Server-Authentifizierung die Kennwörter als Hash einschließt (das gilt allerdings nicht für Contained Databases). Zudem speichert das System in der *msdb*-Datenbank unter anderem alle Jobs des SQL-Server-Agenten und alle Wartungspläne. Weitere Datenbanken des sogenannten Systemkatalogs lassen sich entweder nicht sichern (*tempdb*) oder werden selten bis gar nicht genutzt (*model*). Deshalb sollte man die Datenbanken *master* und *msdb* neben den eigenen in der Planung der Backups berücksichtigen.

Recovery Model passend auswählen

Listing 1: Festlegen des Recovery Model für eine Datenbank

```
USE [master];

GO

-- Recovery Model Simple

ALTER DATABASE: [dotnetconsulting_Backups] SET RECOVERY SIMPLE WITH NO_WAIT;

GO

-- Recovery Model: Bulk-logged
```

```
ALTER DATABASE [dotnetconsulting_Backups] SET RECOVERY BULK_LOGGED WITH NO_WAIT;

GO

-- Recovery Model: Full

ALTER DATABASE [dotnetconsulting_Backups] SET RECOVERY FULL WITH NO_WAIT;
```

Eine Option bestimmt maßgeblich die Art des Backups: Das Recovery Model (Wiederherstellungsmodell) steuert die Verwendung des Transaktionsprotokolls für eine Datenbank und ist für eine Sicherung so wichtig, dass sein aktueller Wert im Backup-Dialog (siehe Abbildung 1) zu Informationszwecken angezeigt wird. Drei Konfigurationswerte sind fürs Recovery Model erlaubt: Full, Bulk-logged und Simple (siehe Tabelle). Man ändert sie über die Eigenschaften der Datenbank (Karteireiter „Options“) oder per Transact-SQL (T-SQL) durch eine der drei Varianten, die Listing 1 zeigt.

Die in der Tabelle getroffene Aussage zur erwarteten Größe des Transaktionsprotokolls hängt von der Menge der Änderungen in der Datenbank (Daten und Struktur) und davon ab, ob Bulk-Operationen vorkommen. Außerdem enthält die Tabelle die Angabe, ob ein Just-in-Time (JIT) Recovery möglich ist, ob man also die Daten zu einem bestimmten Zeitpunkt wiederherstellen kann.

SQL Server kennt drei Arten von Sicherungen: vollständig, differenziell und Log-Backup. Ein vollständiges Backup umfasst alle Daten einer Datenbank, einer Filegroup oder einer Datei. Für ein Restore benötigt man lediglich diese Sicherung. Die Größe hängt von der Datenmenge ab.

Ein differenzielles Backup sichert alle Veränderungen seit dem letzten vollständigen Backup. Für ein Restore wird Letzteres sowie das Differential Backup benötigt – es sind demnach immer zwei Backups beteiligt. In diesem Zusammenhang ist die

Einstellung „Copy-Only“ wichtig (Abbildung 1). Hat man sie beim Erstellen des vollständigen Backups gesetzt, berücksichtigt ein Differential Backup dieses nicht, es ist gewissermaßen unsichtbar. Damit erstellt man aus der Reihe fallende Backups, ohne dass sie später beim Restore fehlen. Ein Differential Backup wird über die Zeit immer größer, je länger das letzte Backup zurückliegt.

Datenbanken und Logs sichern

Ein Log-Backup schließlich sichert das Transaktionsprotokoll. Dabei werden alle Informationen aus dem Protokoll gesichert, egal in welcher Datei sie sich befinden – welche Dateien zu sichern sind, lässt sich nicht angeben. Für ein Restore benötigt man ein vollständiges Backup (wahlweise mit anschließendem Differential Backup), an das sich ein oder mehrere Log-Backups in der richtigen Reihenfolge anschließen können. Auch hier ist die Copy-Only-Einstellung wichtig: Ist sie gesetzt, wird das Transaktionsprotokoll nicht abgeschnitten, sondern weitergeführt.

Der Zeitpunkt des letzten Backups ist im Eigenschaften-Dialog einer Datenbank ersichtlich (Abb. 2).
Ein Log-Backup kann starten, wenn zuvor mindestens ein vollständiges oder differenzielles Backup lief. Wann eine Datenbank zuletzt gesichert wurde, ist aus ihrem Eigenschaften-Dialog ersichtlich (siehe Abbildung 2). Ein Backup startet der Administrator per Dialog oder mit der T-SQL-Anweisung *BACKUP DATABASE* respektive *DATABASE LOG*.

Mehrere Ziele für Backups stehen zur Verfügung: Band, Datei und Cloud (vorgesehen ist dafür Windows Azure Storage). Ein Backup etwa auf DLT (Digital Linear Tape) ist zwar auch mit SQL-Server-eigenen Mitteln möglich, jedoch recht rudimentär und daher eher selten anzutreffen. Häufiger wird in eine Datei gesichert, die das Backup-System des Unternehmens anschließend (zusammen mit anderen Dateien) auf ein anderes Medium

überträgt. Somit lässt sich der SQL Server einfach in die Backup-Planung für die gesamten Firmendaten einbinden.

Damit Sicherungen nicht zu umfangreich geraten, komprimiert der SQL Server Backups optional. Dabei erreicht er eine Kompressionsrate von etwa 50 Prozent. Zwar ist das verglichen mit den bis zu 90 Prozent von nachträglich angewandten Werkzeugen wie WinZip oder 7Zip recht wenig, hat jedoch den Vorteil der Integration in den Backup-Prozess. Die Kompression spart Festplattenplatz und reduziert die Last fürs Speichersystem, beansprucht allerdings CPU-Leistung, die jedoch bei zeitgemäßer Hardware nicht übermäßig ins Gewicht fällt.

Listing 2: Verschlüsselung eines Backups

```
USE master;

GO

-- Database Master Key erzeugen

CREATE MASTER KEY ENCRYPTION BY PASSWORD = '123456';

GO

-- Zertifikat für Backup erstellen

CREATE CERTIFICATE BackupCertificate WITH SUBJECT = 'Zertifikat für Backupverschlüsselung';

GO

-- Backup mit AES256 durchführen

BACKUP DATABASE [dotnetconsulting_Backups]

TO DISK = N'{Pfad}'

WITH
```

```
  COMPRESSION,

   ENCRYPTION (ALGORITHM = AES_256, SERVER CERTIFICATE = BackupCertificate);
```

Außerdem bietet SQL Server das Verschlüsseln von Backups an, was bei schützenswerten Daten oder beim Sichern in die Cloud zur Gewährleistung der Datensicherheit beiträgt. Dazu setzt die Software ein Zertifikat mit einem symmetrischen Verschlüsselungsalgorithmus wie dem Advanced Encryption Standard (AES) ein. Der gewünschte Algorithmus und das Zertifikat müssen beim Erstellen des Backups angegeben werden. Wie das in T-SQL aussehen kann, zeigt Listing 2.

Wahlweise Kompression und Verschlüsselung

Die deutsche Version des SQL Server beweist beim Erstellen eines Backups (wohl unbeabsichtigt) Sinn für Humor mit folgender Meldung: „Das zum Verschlüsseln des Verschlüsselungsschlüssels für die Datenbank verwendete Zertifikat wurde nicht gesichert.“ Trotzdem verweist die Meldung auf den wichtigen Umstand, dass das verwendete Zertifikat (BackupCertificate) für ein Restore zur Verfügung stehen muss und daher seinerseits gesichert und sicher aufbewahrt werden sollte (mit der Anweisung *BACKUP CERTIFICATE*).

Listing 3: Restore einer Datenbank

```
USE [master];

RESTORE DATABASE [dotnetconsulting_Backups] FROM  DISK = {Pfad} WITH FILE = 1;
```

Will man eine Sicherung wiederherstellen, darf es zu diesem Zeitpunkt keine Client-Verbindungen zur betreffenden Datenbank geben, andernfalls scheitert das Restore mit einem Timeout. Ein schlichtes Restore mit T-SQL zeigt Listing 3. Mit dem Zusatz *MOVE* bestimmt der Administrator, wie die

Datenbankdateien wiederhergestellt werden sollen – etwa wenn der Zielserver eine unterschiedliche Verzeichnisstruktur enthält oder wenn die Wiederherstellung eine Kopie einer Datenbank erstellen soll. Übrigens akzeptiert die *RESTORE*-Anweisung bei Bedarf einen alternativen Name für die Datenbank.

Besteht ein Backup aus mehreren Teilen, etwa weil ein differenzielles Backup durchgeführt wurde, erlaubt der Zusatz *NORECOVERY* das Einspielen weiterer Dateien als Teil des Restore:

```
RESTORE DATABASE [dotnetconsulting_Backups]
 FROM  DISK = {Pfad} WITH FILE = 1, NORECOVERY;
```

Eine Datenbank im Object Explorer während des Restore (Abb. 3)
Bei einem Restore markiert das SQL Server Management Studio den Zustand der Datenbank mit einem entsprechenden Symbol und dem Wort „Restoring“ (Abbildung 3). Erst nach dem Überspielen der letzten Datei – ohne den Zusatz *NORECOVERY* beziehungsweise *RECOVERY* – lässt sich die Datenbank wieder regulär benutzen.

Grenzen der Bordmittel in großen Umgebungen

Die Tools fürs Backup des SQL Server stoßen jedoch an Grenzen. Mit zunehmender Zahl von SQL-Server-Installationen muss der Administrator immer mehr Datenbanken in unterschiedlichen Versionen verwalten, sichern und zudem erfolgreiche und fehlgeschlagene Sicherungsjobs überblicken. Zwar verteilt der SQL Server Aufgaben (die SQL Server Agent Jobs) dezentral, jedoch mangelt es dabei an Komfort und Transparenz. Zumal in einem Unternehmen das Sichern der SQL-Server-Datenbank nur ein Aspekt des Backups ist: Dateiserver, Exchange-Server und viele mehr kommen dazu. Außerdem verraten die Bordmittel nicht, wie lange die freien Ressourcen und Backup-Zeitfenster für einen reibungslosen Betrieb reichen, denn Datenbanken wachsen und

eine Prognose ist schwierig. Diese Aufgaben können nur umfassende Backup-Tools übernehmen.

Zwischen den Versionen und Editionen des SQL Server bestehen kleinere Unterschiede hinsichtlich der Sicherungstools. So hielt etwa die Kompression erst in SQL Server 2008 Einzug und stand damals nur der Enterprise Edition zur Verfügung. Ab SQL Server 2012 kommt jedoch auch die Standard Edition in den Genuss dieses Features. Generell bedeuten die Ressourcen einer SQL-Server-Installation auch für ein Backup eine Limitierung. Nutzt etwa eine Edition nur eine begrenzte Zahl an Prozessoren (Kernen) und eine begrenzte Menge Speicher, kann das Backup länger dauern. Andererseits sind in diesen Fällen die Datenbanken meist nicht allzu groß. Jedoch sei zur Beruhigung gesagt, dass jede Version oder Edition selbstverständlich in der Lage ist, ihre jeweilige maximale Datenbankgröße zu sichern (und wiederherzustellen).

Zum Schluss noch einige Tipps für einen einfacheren Alltag mit Backups und Restores:

– Backups sollten automatisiert laufen. Das erledigt entweder der SQL-Server-Agent, oder ein Backup ist Teil eines Wartungsplans.

– Bei der Planung sollte man Backups möglichst einfach halten. Wenn Zeit und Größe es erlauben, sind vollständige Backups zu bevorzugen.

– Der Administrator sollte regelmäßig Restores durchführen und die dafür benötigte Zeit messen. Das stellt sicher, dass eine Wiederherstellung auch im Ernstfall gelingt.

– Eine Sicherung ist nur so gut wie die dazugehörende Wiederherstellung. Wenn Letztere aus irgendwelchen Gründen nicht klappt, ist Erstere wertlos.

Fazit

SQL Server bietet alles Notwendige, um sowohl Backups als auch bei Bedarf einen Restore effizient und flexibel durchzuführen. Microsofts Bordmittel für seine Datenbankplattform decken also auch diese Phase des Betriebs ab. (tiw) Thorsten Kansy arbeitet als Berater, Softwarearchitekt und Coach mit den Schwerpunkten Datenbanken und Business Intelligence.

[/expand]