Workshop SQL-Server Verwaltung

Verwaltung und Fehlersuche

Freundallee 13 a
30173 Hannover
www.netz-weise.de

Inhalt

Transact-SQL Grundlagen

SQL-Server verwendet einen eigenen SQL-Dialekt namens Transact-SQL, der aber zu großen Teilen ANSI-SQL-Konform ist – ANSI-SQL ist der Industriestandard für die SQL-Abfragesprache. Jeder Hersteller hat seinen eigenen SQL-Dialekt, der sich mehr oder weniger stark vom Standard unterscheidet.

SQL (Structured Query Language) ist in 3 Bereiche aufgeteilt, die DML (Data Manipulation Language), die DCL (Data Control Language) und die DDL (Data Definition Language).

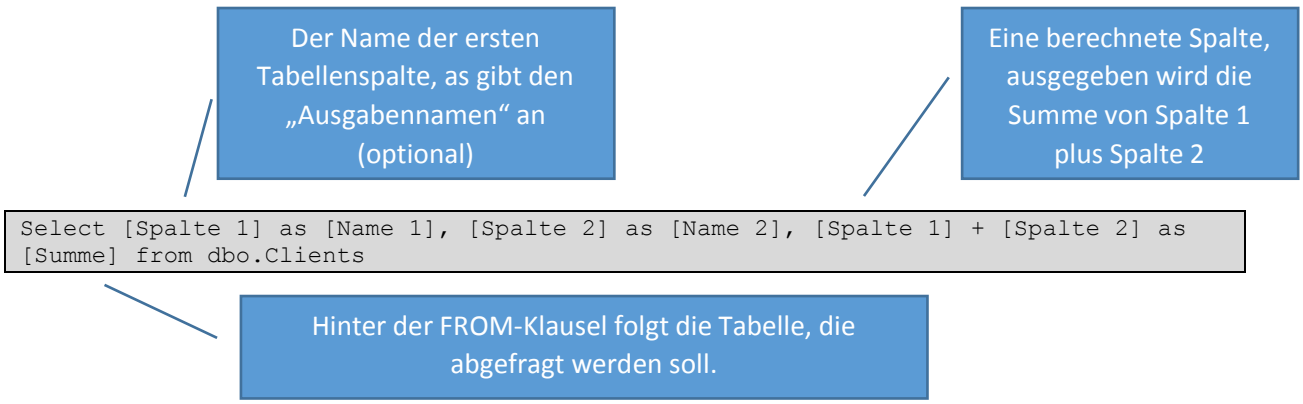
DML – Data Manipulation Language

Mit der DML können Daten vom Datenbankserver abgefragt werden. Die wesentlichen Befehle lauten:

Statement	
Select	Abfragen von Daten
Insert	Einfügen von Datensätzen
Update	Aktualisieren von Datensätzen
Delete	Löschen von Datensätzen
Merge	Eine Kombination von Select und Insert/Update

Select

Mit dem Select-Statement werden Daten aus Tabellen oder Sichten abgerufen. Die Standard-Syntax lautet:



Dem Select folgen die Spalten einer Tabelle, die abgefragt werden sollen. Die Spalten können über das AS hinter dem Spaltennamen in der Ausgabe umbenannt werden. Man kann auch berechnete Spalten erzeugen, indem man mehrere Spalten miteinander verrechnet [Spalte 1] + [Spalte 2]. Hinter dem FROM-Statement folgt die Tabelle, aus der die Daten geliefert werden sollen.
Auffällig ist hier der zweiteilige Name dbo.Rechnungen. Jedes Datenbankobjekt kann über einen **4-teiligen Namen** referenziert werden: Server1.KAV.dbo.Clients. Der erste Abschnitt gibt den Servernamen an, der zweite Teil den Datenbanknamen, der 3. Teil das Schema (quasi ein Ordner für Tabellen), und der 4. Teil gibt den Objektnamen an. Das Standardschema für Objekte, wenn kein Schema angegeben wird, lautet dbo (database owner).
Daten in relationalen Datenbanken sind normalerweise normalisiert, was bedeutet, dass man versucht, Redundanzen in der Datenbank zu vermeiden. Dazu werden Daten, die für sich alleine stehen können, in einzelne Tabellen zusammengefasst. Will man Beispielsweise Daten für Bestellungen verwalten, legt man nicht alle Daten (Wer hat bestellt, was hat er bestellt, wie teuer war die Bestellung, wann war die Bestellung, wer hat geliefert usw) in einer Tabelle ab, sondern versucht die Daten zu identifizieren, die als eigene Einheit (Entitäten) abgebildet werden können. In Beispiel könnte man die Bestellungen-Tabelle also aufteilen in:

- Kunden (1 Tabelle)
- Produkte (1 Tabelle)
- Bestelldetails
- Mitarbeiter

Usw. Dieser Vorgang nennt sich normalisieren der Daten.
Um Daten aus normalisieren Tabellen jetzt wieder miteinander zu verknüpfen, stellt SQL das Schlüsselwort JOIN zur Verfügung.
Join kennt „inner- und outer Joins“, wobei der am häufigsten genutzte Join der „Inner join“ ist. Beim Inner Join werden mind. 2 Tabellen miteinander verknüpft, indem man SQL-Server eine Spalte angibt, die in beiden Tabellen die gleichen Daten verwaltet, z.B. eine Kundennummer (wenn man alle Bestellungen eines Kunden einsehen möchte), eine Bestellnummer (wenn man alle Artikel einer Bestellung sehen möchte) usw. Die JOIN-Klausel wird der FROM hintenangestellt, zusammen mit der Spalte, über die die Tabellen „zusammengeklebt“ werden sollen.

```
USE KAV
SELECT  ho.tmLastInfoUpdate
       ,ho.strWinHostName
       ,ho.nStatus
       ,ag.tmUpdate
FROM    hosts AS ho
        INNER JOIN admgroups AS ag ON ho.ngroup = ag.nID


tmLastInfoUpdate           strWinHostName      nStatus      tmUpdate
2013-04-15 11:38:42.247    SERVER              29           2013-03-30 21:24:20.027
2013-04-15 11:33:32.823    3GNGEL243           0            2013-02-01 10:03:19.020
2013-04-15 11:33:32.823    3GNGEL30            0            2013-02-01 10:03:19.020
2013-04-15 11:33:32.823    3GNGEL385           0            2013-02-01 10:03:19.020
2013-04-15 11:33:32.823    3ANGEL827           0            2013-02-01 10:03:19.020
```

In diesem Beispiel werden die beiden Tabellen admgroups und host zusammengefügt. Dabei vergleicht SQL-Server die erste Tabelle (hosts) zeilenweise mit der zweiten Tabelle. Im Prinzip kann man sich das vorstellen, als würde SQL-Server die 1. Zeile der Hosts nehmen, und mit jeder einzelnen Zeile der admgroups vergleichen und für jede Zeile, für die hosts.ngroug und ad.nid gleich sind, eine Zeile zurückliefern. Dann macht er das Ganze für die 2. Zeile der host usw., bis er alle Zeilen miteinander verknüpft hat, die in admgroups.nID und hosts.ngroup den gleichen Wert stehen haben.

Ein Outer join liefert neben den Werten des inner join auch die Werte zurück, für die in der einen Tabelle ein Wert exisitert, in der Vergleichstabelle jedoch KEIN EINZIGER Vergleichswert. Dies ist wichtig, wenn man z.B. alle hosts finden will, für die keine Admingroup exisitert – der Wert, der in hosts.ngroup steht, exisitert in der admingroup nicht (z.B. weil der Datensatz manuell gelöscht wurde).

```
USE KAV
SELECT  ho.tmLastInfoUpdate
       ,ho.strWinHostName
       ,ho.nStatus
       ,ag.tmUpdate
FROM    hosts AS ho
        LEFT OUTER JOIN admgroups AS ag ON ho.ngroup = ag.nID
WHERE   ag.nid IS NULL
```

Insert

Mit dem Insert-Befehl werden neue Datensätze eingefügt. Insert benötigt mindestens die Tabelle und die Daten, die in die Tabelle eingefügt werden sollen

```
INSERT  INTO dbo.Clients
VALUES  ( 'pc1', 'pc2' )
INSERT  INTO dbo.Clients
VALUES  ( 'pc1', 'pc2' ), ( 'pc3', 'pc4' ), ( 'pc5', 'pc6' )
```

Wenn nicht alle Spalten eingefügt werden sollen, oder die Spalten nicht in der gleichen Reihenfolge eingefügt werden, wie sie in der Tabelle vorkommen, kann man die Spalten hinter Tabelle mit angeben:

```
INSERT  INTO dbo.Clients ( [Name 2] )
VALUES  ( 'pc1' ), ( 'pc3' ), ( 'pc5' )
```

Um eine große Menge von Daten von einer Tabelle in eine andere zu übertragen, kann man einen Select mit einem Insert verbinden:

```
INSERT INTO dbo.Clients ( [NAME 1] )
      SELECT Computername FROM dbo.Hosts
```

http://technet.microsoft.com/de-de/library/ms174335.aspx

Update

Zum Einfügen von Daten wird der Update-Befehl verwendet. Update ist Datensatzbasiert, ändert also grundsätzlich einen Satz von Daten und nicht Einzeldaten. Wird kein Filter (where) verwendet, ändert update also jede Zeile innerhalb einer Tabelle.

```
UPDATE SalesLT.SalesOrderDetail
      SET UnitPrice = UnitPrice * 1.1
```

Um einzelne Datensätze zu ändern, müssen mit der Where-Klausel die zu ändernden Datensätze eingeschränkt werden:

```
UPDATE SalesLT.SalesOrderDetail
      SET UnitPrice = UnitPrice * 1.1
      WHERE OrderQty > 10
```

Diese Where-Klausel filtert erst alle Datensätze, die in der Spalte OrderQty einen Wert größer 10 haben, und ändert dann nur diese Datensätze.

Delete

Delete löscht Datensätze aus einer Tabelle. Auch Delete arbeitet Datensatzbasiert, löscht also ohne Where-Klausel alle Datensätze einer Tabelle:

```
DELETE FROM SalesLT.SalesOrderDetail
```

Sollten nicht alle Datensätze gelöscht werden, muß wieder erst mit einer Where-Klausel gefiltert werden:

```
DELETE FROM SalesLT.SalesOrderDetail
WHERE ModifiedDate < GETDATE()-28
```

http://msdn.microsoft.com/de-de/library/ms189835.aspx

Soll eine komplette Tabelle gelöscht werden, empfiehlt es sich, Truncate Table zu verwenden. Truncate Table löscht eine Tabelle unter Umgehung des Transaktionsprotokolls (es wird nicht jeder Datensatz einzeln gelöscht), was bei einer großen Tabelle einen deutlichen Performance-Vorteil bedeutet:

```
TRUNCATE TABLE SalesLT.SalesOrderDetail
```

Um Änderungen rückgängig zu machen – TSQL löscht und überschreibt Daten ohne Nachfragen – ist es sinnvoll, Änderungen in einer Transaktion zu verpacken:

```
BEGIN TRANSACTION
DELETE FROM SalesLT.SalesOrderDetail
WHERE ModifiedDate < GETDATE()-28
SELECT * FROM SalesLT.SalesOrderDetail
COMMIT
```

Mit Hilfe von ROLLBACK TRANSACTION kann, bevor der Commit durchgeführt wird, die Transkation wieder rückgängig gemacht werden.

DCL – Data Contol Language

Mit der DDL werden die Berechtigungen auf dem Datenbankserver gesetzt. Die wesentlichen Befehle lauten:

Statement	
Grant	Vergeben von Berechtigungen
Deny	Verweigern von Berechtigungen
Revoke	Entfernen von Berechtigungen

Die Berechtigungsvergabe findet am einfachsten über die GUI statt. Weiteres gibt's im Kapitel Die Kontostände der beiden Konten müssen während des gesamten Vorgangs für weitere Änderungen blockiert sein, damit nicht z.B. zwischen Schritt 1 und 2 oder 4 und 5 ein anderer Vorgang eine Änderung des Kontostand vornimmt (eine andere Überweisung beispielsweise), die von unserem Änderungsvorgang nicht bemerkt wird.

Außerdem muß sichergestellt sein, dass im Falle eines Fehlers (Server stürzt ab, Netzwerkleitung bricht zusammen, Flugzeug schlägt ins Rechenzentrum ein) der Ursprungszustand wiederhergestellt werden kann, weil sonst z.B. das Konto des Kunden A belastet wird, aber der Betrag dem Konto B niemals gut geschrieben wird. Dies geschieht, indem eine Transaktion erzeugt wird. Vor Schritt 1 wird in SQL mit dem „Begin Transaction"-Befehl eine Transaktion gestartet, nach Schritt 5 wird die Transaktion beendet. Wird er Überweisungsvorgang z.B. durch einen Serverfehler vorzeitig beendet, wird die Transkation beim Serverneustart wieder bis zum Begin-Transaction rückgängig gemacht. Tritt während der Transaktion ein programmatischer Fehler auf – z.B. stellt das Script fest, dass das Zielkonto nicht existiert - kann mit „Rollback Transaction" die Transaktion auf Scriptebene rückgängig gemacht werden.

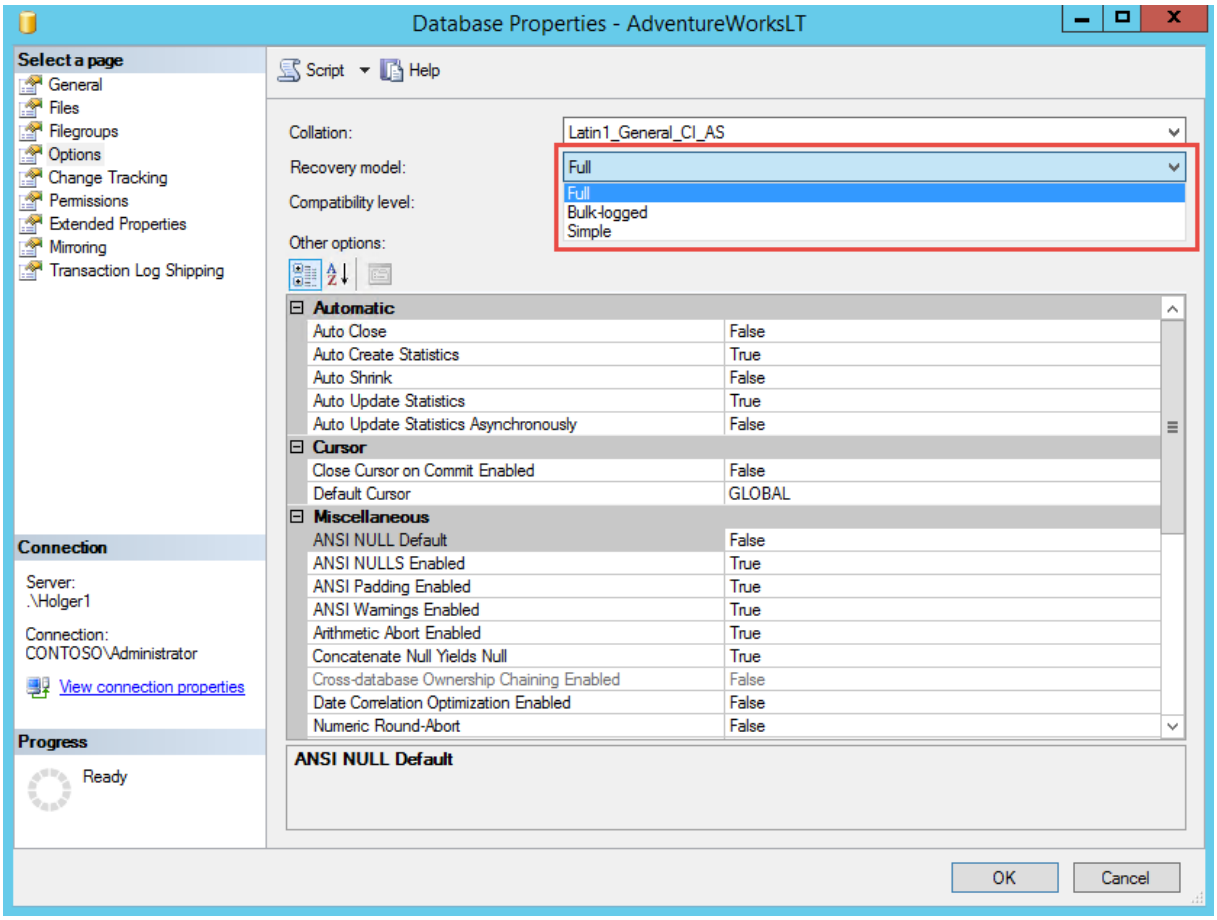
Damit der Ursprungszustand aber wiederhergestellt werden kann, müssen die Änderungen irgendwo aufgezeichnet werden. Und dies geschieht im Transaktionsprotokoll, dass jede Änderung akribisch mit einer Versionsnummer (LSN = Log Sequence Number) und einem Zeitstempel protokolliert. Wenn es jetzt zu einem Abbruch kommt, kann der Datenbankserver einfach alle Änderungen wieder rückwärts ablaufen lassen bis zum Begin Transaction.

Transaktionsprotokolle sind grundsätzlich eine feine Sache, die allerdings auch einige Nachteile mit sich bringen:

- Alle Änderungen müssen doppelt geschrieben werden (in die DB und ins Protokoll)

- Die Protokollierung ist zeitaufwändig
- Die Protokolle werden im Laufe der Zeit immer größer

Deshalb gibt es die Möglichkeit SQL Server anzuweisen, nicht alles zu protokollieren. Dafür gibt es das sogenannte Wiederherstellungsmodell (Recovery-Model) der Datenbank. Das Wiederherstellungsmodell wird in den Datenbankoptionen gesetzt.



Wiederherstellungsmodel	Protokolliert wird…
Vollständig (Voll)	Alle Transaktionen werden vollständig gelogged.
Massenprotokolliert (Bulk Logged)	Masseneinfügevorgänge werden nur minimal gelogged – der SQL-Server speichert nur ein Binärabbild der Datenseiten vor und nach dem Änderungsvorgang http://msdn.microsoft.com/en-us/library/ms190925.aspx#MinimallyLogged
Einfach (Simple)	Alle Transaktionen werden minimal gelogged. Außerdem werden alle vollständigen Transaktionen sofort überschrieben

Setzen des Recovery-Models mit TSQL:

```
ALTER DATABASE AdventureworksLT SET RECOVERY FULL
```

Mehr zum Thema SQL-Server Transaktionsprotokolle:
ebook sql server transaction log management:
http://www.red-gate.com/community/books/sql-server-transaction-log-management
Understanding Logging and Recovery in SQL Server (gut erklärt: Virtuelle Log Files)

http://technet.microsoft.com/en-us/magazine/2009.02.logging.aspx

Berechtigungsvergabe in SQL-Server.

DDL – Data Definition Language

Mit der DDL werden neue Strukturen wie Tabellen aber auch Datenbanken angelegt. Die wesentlichen Befehle lauten:

Statement	
Create	Anlegen eines Datenbankobjekts
Drop	Löschen eines Datenbankobjekts

Um neue Objekte wie Tabellen, Sichten, gespeicherte Prozeduren usw anzulegen, werden die Befehle der DDL verwendet. Drop löscht Objekte (Drop Table, Drop Database, Drop View), Create lege neue Objekte an (Create Table, Create Trigger, Create Procedure).

Transaktionen – eine Einführung

Transaktionen stellen die kleinste Einheit her, in der SQL-Server Daten in der Datenbank ändert. Transaktionen sind z.B. Einfüge-, Änderungs- und Lösch-Vorgänge (Insert-, Update- und Delete). Transaktionen sind allerdings keine grundsätzlich von der Natur gegebene Einheit. Von Haus aus befindet sich der SQL-Server zwar im sogenannten Auto-Commit Modus, es wird also für jeden Schreibvorgang eine Trankaktion ausgeführt, aber mit den Befehlen kann man auch mehrere Befehle zu einer Transaktion zusammenfassen. Damit wird also aus mehreren Schreibvorgängen, die im Autocommit-Modus jeder für sich eine einzelne Transaktion wären, eine Transkaktion gemacht.

Transaktionen werden benötigt, um mehrere zusammenhängende Änderungen innerhalb einer Datenbank immer gemeinsam auszuführe. Das klassische Beispiel dafür ist die Überweisung eines Geldbetrags von einem Bankkonto auf ein anderes.

Wenn Kunde A von der „Detschen Bank“ Geld auf das Konto von Kunde B, ebenfalls bei der „Detschen Bank“, überweisen möchte, muss die Datenbanklogik folgende Vorgänge vornehmen:

1. Überprüfen, ob Kunde A über eine ausreichende Kontodeckung verfügt (Select)
2. Den Überweisungsbetrag vom Konto des Kunden A subtrahieren (Update Kontostand, set Kontostand = „Alter Kontostand“ - Überweisungsbetrag)
3. Überprüfen, ob das Zielkonto des Kunden B existiert
4. Den Kontostand des Kunden B auslesen (Select)
5. Den Kontostand des Kunden B aktualisieren (Update Kontostand, set Kontostand = „Alter Kontostand“ + Überweisungsbetrag

Die Kontostände der beiden Konten müssen während des gesamten Vorgangs für weitere Änderungen blockiert sein, damit nicht z.B. zwischen Schritt 1 und 2 oder 4 und 5 ein anderer Vorgang eine Änderung des Kontostand vornimmt (eine andere Überweisung beispielsweise), die von unserem Änderungsvorgang nicht bemerkt wird.

Außerdem muß sichergestellt sein, dass im Falle eines Fehlers (Server stürzt ab, Netzwerkleitung bricht zusammen, Flugzeug schlägt ins Rechenzentrum ein) der Ursprungszustand wiederhergestellt werden kann, weil sonst z.B. das Konto des Kunden A belastet wird, aber der Betrag dem Konto B niemals gut geschrieben wird. Dies geschieht, indem eine Transaktion erzeugt wird. Vor Schritt 1 wird in SQL mit dem „Begin Transaction“-Befehl eine Transaktion gestartet, nach Schritt 5 wird die Transaktion beendet. Wird er Überweisungsvorgang z.B. durch einen Serverfehler vorzeitig beendet, wird die Transkation beim Serverneustart wieder bis zum Begin-Transaction rückgängig gemacht. Tritt während der Transaktion ein programmatischer Fehler auf – z.B. stellt das Script fest, dass das Zielkonto nicht existiert - kann mit „Rollback Transaction“ die Transaktion auf Scriptebene rückgängig gemacht werden.

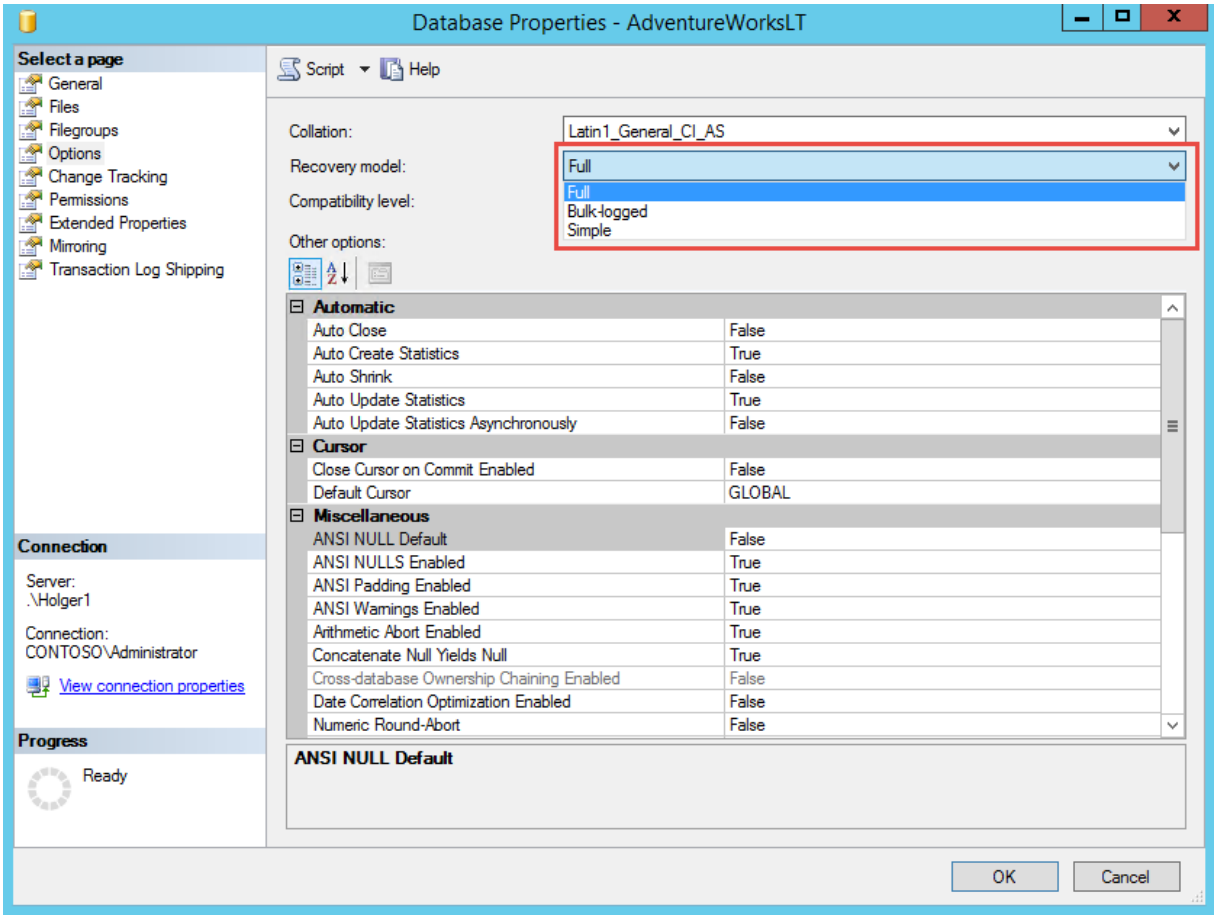
Damit der Ursprungszustand aber wiederhergestellt werden kann, müssen die Änderungen irgendwo aufgezeichnet werden. Und dies geschieht im Transaktionsprotokoll, dass jede Änderung akribisch mit einer Versionsnummer (LSN = Log Sequence Number) und einem Zeitstempel protokolliert. Wenn es jetzt zu einem Abbruch kommt, kann der Datenbankserver einfach alle Änderungen wieder rückwärts ablaufen lassen bis zum Begin Transaction.

Transaktionsprotokolle sind grundsätzlich eine feine Sache, die allerdings auch einige Nachteile mit sich bringen:

- Alle Änderungen müssen doppelt geschrieben werden (in die DB und ins Protokoll)
- Die Protokollierung ist zeitaufwändig

- Die Protokolle werden im Laufe der Zeit immer größer

Deshalb gibt es die Möglichkeit SQL Server anzuweisen, nicht alles zu protokollieren. Dafür gibt es das sogenannte Wiederherstellungsmodell (Recovery-Model) der Datenbank. Das Wiederherstellungsmodell wird in den Datenbankoptionen gesetzt.



Wiederherstellungsmodel	Protokolliert wird…
Vollständig (Voll)	Alle Transaktionen werden vollständig gelogged.
Massenprotokolliert (Bulk Logged)	Masseneinfügevorgänge werden nur minimal gelogged – der SQL-Server speichert nur ein Binärabbild der Datenseiten vor und nach dem Änderungsvorgang http://msdn.microsoft.com/en-us/library/ms190925.aspx#MinimallyLogged
Einfach (Simple)	Alle Transaktionen werden minimal gelogged. Außerdem werden alle vollständigen Transaktionen sofort überschrieben

Setzen des Recovery-Models mit TSQL:

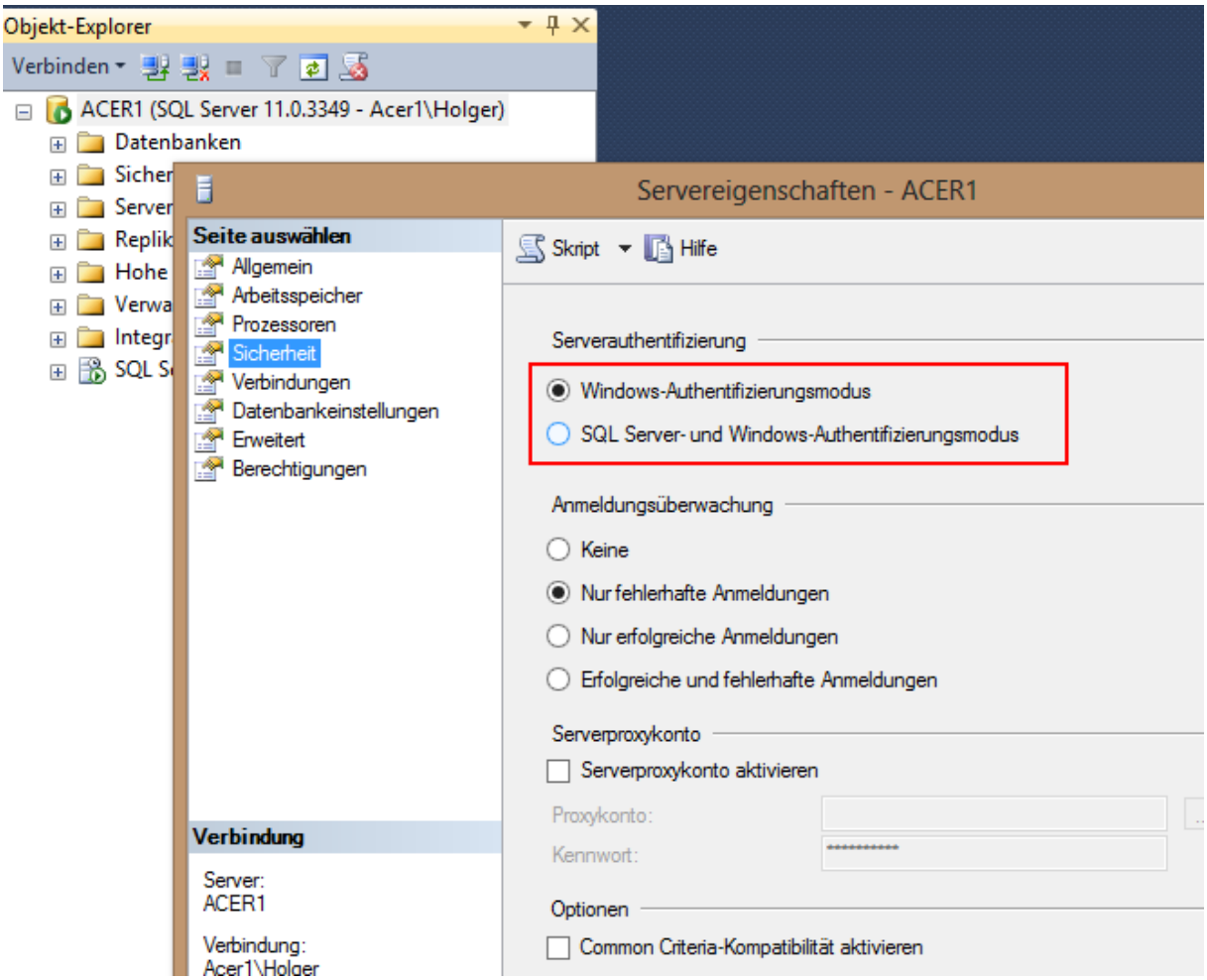
```
ALTER DATABASE AdventureworksLT SET RECOVERY FULL
```

Mehr zum Thema SQL-Server Transaktionsprotokolle:
ebook sql server transaction log management:
http://www.red-gate.com/community/books/sql-server-transaction-log-management
Understanding Logging and Recovery in SQL Server (gut erklärt: Virtuelle Log Files)
http://technet.microsoft.com/en-us/magazine/2009.02.logging.aspx

Berechtigungsvergabe in SQL-Server

SQL-Server ist vom Konzept her „Secure by Design“, d.h. von Haus aus so eingestellt, dass er hohe Sicherheitsanforderungen schon mit der Standardinstallation erfüllt.

Ein wesentlicher Bestandteil des Sicherheitskonzepts ist dabei die Möglichkeit, auf jedes Objekt Berechtigungen zu vergeben, die festlegen, wer darauf zugreifen darf. Dabei unterscheidet SQL-Server 2 verschiedene Typen von Benutzern: Windows-Benutzer (werden vom Windows-Server bzw. im Active Directory verwaltet) und SQL-Benutzer, die im SQL-Server angelegt und verwaltet werden. In der Standardinstallation sind SQL-Benutzer deaktiviert, und nur die Authentifizierung über Windows-Benutzer ist möglich. Dies bietet eine erhöhte Sicherheit, hat allerdings den Nachteil, dass nicht-Windows-Nutzer (z.B. Apple oder Linux) bzw. Clients, die nicht in einer AD-Umgebung eingebunden sind (Web-User) oder deren Authentifizierung über eine Firewall geblockt wird, nicht auf den SQL-Server zugreifen können. Die Authentifizierung kann über die Eigenschaften des Servers im SQL-Server Managementstudio umgestellt werden:



Die Windows-Authentifizierung benutzt Kerberos und Domänen-Anmeldungen, um einen Benutzer zu Authentifizieren. Der SQL-Server vertraut damit der Domäne bei der Anmeldung des Benutzers.

Benutzer und Rollen

Damit ein Benutzer Zugriff auf den SQL-Server bekommen kann, benötigt er einen sogenannten SQL-Server Login. Ein Login ist ein Benutzerkonto, das auf dem SQL-Server hinterlegt ist und es dem SQL-

Server erlaubt, den Benutzer zu Authentifizieren. Zusätzlich werden in jeder auf dem SQL-Server angelegten Datenbank Datenbankbenutzer verwaltet. Ein Datenbankbenutzer ist ein Benutzerobjekt, das nur innerhalb der Datenbank existiert und über das ein Benutzer Zugriff auf die Datenbank selber bekommt. Ein Login wird immer mit einem Datenbankbenutzer verknüpft – ein Login hat also selber keine Berechtigungen in der Datenbank, sondern „erbt“ diese vom Datenbankbenutzer.

Bis SQL-Server 2008 R2 war ein Login zwingend notwendig, um Zugriff auf eine Datenbank zu bekommen. Seit SQL-Server 2012 gibt es auch noch die Möglichkeit, sogenannte „Contained Databases“ zu verwenden, in denen die Authentifizierung nur in der Datenbank durchgeführt wird. Der Vorteil dieses Verfahrens ist, dass beim Verschieben der Datenbank auf eine andere SQL-Server Instanz (z.B. bei Spiegelung oder Always on) die Logins nicht auf dem zweiten Server neu angelegt werden müssen.

Neben Datenbankbenutzern und Logins gibt es auf dem SQL-Server noch die Möglichkeit, Benutzer zu Gruppen zusammenzufassen. Gruppen haben auf dem SQL-Server die Bezeichnung „Rollen“. Es gibt 3 bzw. seit SQL-Server 2012 4 verschiedene Gruppen:

Rollentyp	Beschreibung
Feste Datenbankrollen	Sind immer vorhanden und bereits mit Berechtigungen versehen – vergleichbar lokalen Gruppen auf einem PC
Freie Datenbankrollen	Können selbst definiert werden, beinhalten Datenbankuser
Feste Serverrollen	Sind immer vorhanden und bereits mit Berechtigungen versehen – vergleichbar globalen Gruppen in der Domäne. Sie vergeben administrative Rechte auf dem Server
Freie Serverrollen (ab SQL-Server 2012)	Können selbst definiert werden, um selber administrative Gruppen erstellen zu können

Feste Datenbankrollen

Rolle	Berechtigung
db_owner	Administrative Rechte innerhalb der Datenbank
db_accessadmin	Können für Logins Datenbankbenutzer erstellen
db_securityadmin	Rollenmitgliedschaften und Berechtigungen verwalten
db_ddladmin	Neue Objekte in der DB anlegen (Data Definition Language)
db_backupoperator	Backup der Datenbank ausführen – auch von Daten, auf die die Rolle selbst keine Rechte hat
db_datareader	Alle Daten der DB lesen (Select-Rechte)
db_datawriter	In alle Tabellen schreiben (Select, Update, Delete-Rechte)
db_denydatareader	Lesen in der DB verweigern
db_denydatawriter	Schreiben in der DB verweigern

Feste Serverrollen

Rolle	Berechtigung
Public	Automatische Rolle. jeder Login ist automatisch Mitglied
Sysadmin	Administratoren des Servers
Securityadmin	Logins hinzufügen und Rollenmitgliedschaften ändern (zu behandeln wie sysadmin)

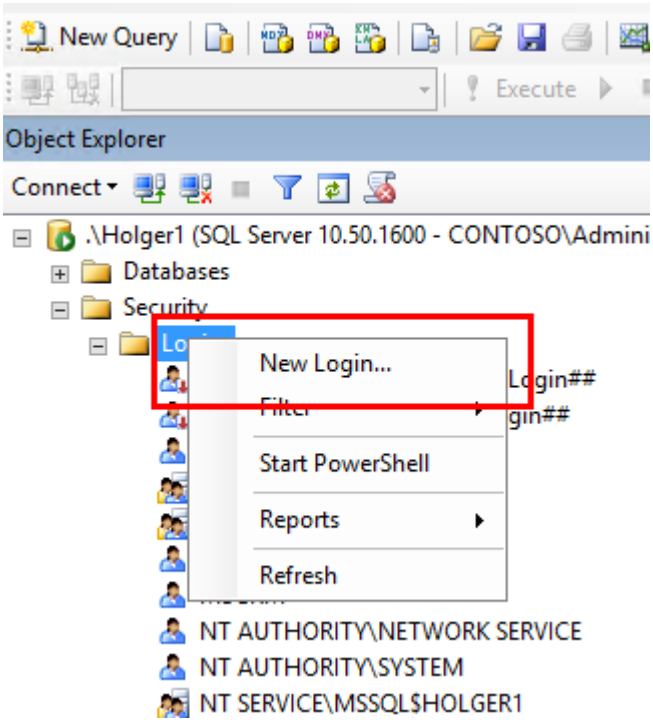
Serveradmin	Servereinstellungen ändern (bezieht sich auf den Server, nicht auf Einstellungen wie Logins)
Setupadmin	Linked Server hinzufügen
Processadmin	Beliebige Benutzerprozesse mit Hilfe von Kill beenden
Diskadmin	Verwaltet Datenträger (wie DBCC Shrinkfile,…)
Dbcreator	Neue Datenbanken anlegen. Der anlegende Login wird automatisch db_owner der neuen Datenbank
bulkadmin	Kann Masseneinfügevorgänge vornehmen

Weiterführende Informationen:

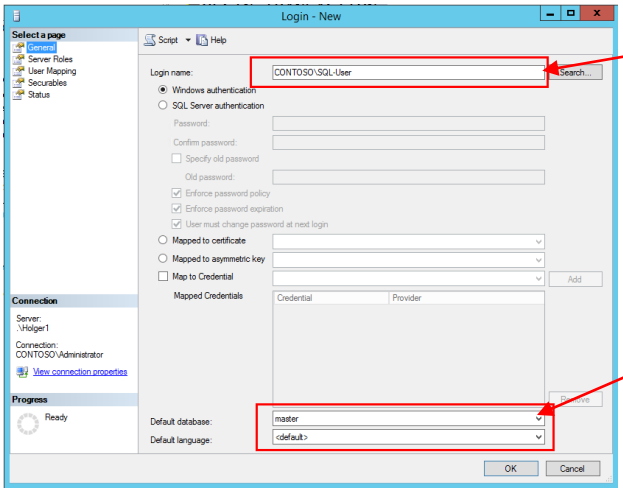
http://msdn.microsoft.com/de-de/library/ms181127%28v=sql.105%29.aspx

Anlegen eines neuen Logins und Datenbankbenutzers

Neue Logins können mit Hilfe des SQL-Server Managementstudios hinzugefügt, oder indem Sie mit Create Login einen Login erzeugen. Der Eintrag New Login findet sich im Management-Studio unter dem Eintrag Security -> Logins.



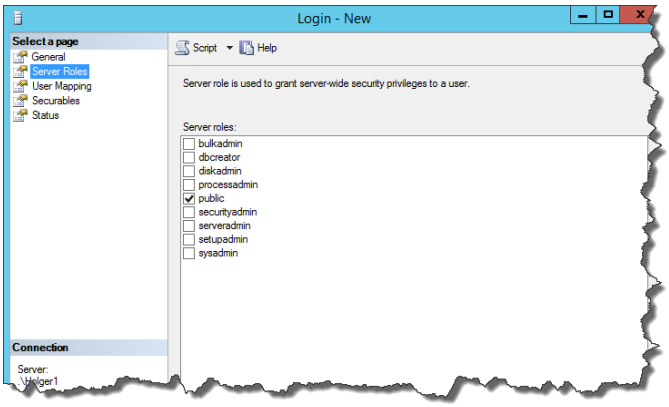
Zuerst müssen Sie festlegen, ob Sie einen SQL-Login oder einen Windows-Login anlegen wollen. Ein Windows-Login kann ein Active-Directory-Benutzer, ein lokaler Benutzer oder aber auch eine Gruppe sein. Wenn Sie viele Logins verwalten müssen, empfiehlt es sich, eine Active-Directory Gruppe als Login hinzuzufügen. Alle Mitglieder der Gruppe können sich dann auch auf dem SQL-Server anmelden und müssen nicht mehr einzeln mit einem Login versehen werden.



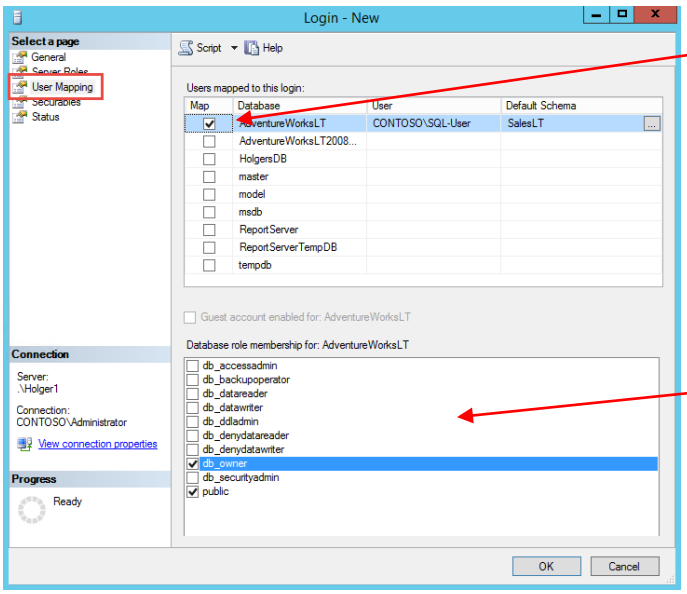
Der Benutzername – Wenn es sich um einen Windows-Account handelt, inkl. Domäne bzw. Computernamen

Die Datenbank, die verwendet wird, wenn keine Datenbank bei der Verbindung angegeben wird. Wird hier eine Datenbank angegeben, auf die der Benutzer keine Rechte hat, oder die DB wird gelöscht, kann der User sich nicht anmelden!

Als nächste legen Sie fest, welchen Serverrollen der Benutzer hinzugefügt werden soll, indem Sie auf Serverrollen wechseln:



Jeder Benutzer ist Mitglied der Serverrolle Public. Diese Rolle kann nicht entfernt werden. Alle Rollen, die Sie an dieser Stelle auswählen, verschaffen dem Benutzer administrative Berechtigungen (s. Serverrollen weiter oben). Daher werden Sie hier normalweise keine weiteren Berechtigungen vergeben. Wechseln Sie zu User-Mappings – hier können Sie verlinkte Datenbankbenutzer für den Login erstellen.



Hier legen Sie die Datenbanken fest, auf die der Login Zugriff erhalten soll. Es wird automatisch ein verlinkter User erstellt. Das Default Schema legt das Schema fest, das verwendet wird, wenn beim Select kein Schema angewendet wird (sonst Standardmässig dbo).

Die Datenbankrollen, in die der Benutzer aufgenommen werden soll (s. oben unter Datenbankrollen)

Wenn Sie nun OK wählen, wird der Login angelegt sowie für jede Datenbank, die Sie unter User-Mappings ausgewählt haben, ein verknüpfter Datenbankuser erzeugt.

Wenn Sie beim Erstellen eines Logins einen SQL-Login auswählen, werden die Benutzerinformationen inklusive des Kennworts auf dem SQL-Server in der Masterdatenbank gesichert. Wenn Sie die Erzwingung von Kennwortrichtlinien gewählt haben, greift der SQL-Server auf die Einstellungen des Windows-Servers zurück, in einer Domänenumgebung gelten damit die Domänen-Kennwortrichtlinien.

SQL-Server Backups

SQL-Server stellt eine Backup-Schnittstelle zur Verfügung, die sich komplett über TSQL steuern lässt. Mit den Befehlen Backup Database und Backup Log kann man jede Datenbank des SQL-Servers sichern, sofern man über db_backupoperator-Berechtigungen verfügt.

SQL-Server kennt drei Formen von Backups:

- Vollbackup: Die komplette Datenbank wird gesichert. Die Datenbanksicherung hat den Stand der Datenbank nach Beendigung der Sicherung.
- Differenzbackup: Es werden mit jedem Diff-Backup alle Änderungen seit der letzten Vollsicherung gesichert.
- Transaktionsprotokollbackup: Das Transaktionsprotokollbackup entspricht einer inkrementellen Sicherung. Es werden alle Daten seit dem letzten Transaktionsprotokollbackup gesichert.

Transaktionsprotokollsicherungen setzen voraus, dass überhaupt eine vollständige Protokollierung aller Änderungen in der Datenbank vorgenommen wird. Daher muss das Wiederherstellungsmodell der Datenbank auf „Vollständig“ oder „Massenprotokolliert“ stehen (s. auch Transaktionen, eine Einführung, S. 9).

Eine Vollsicherung sichert die Datenseiten der Datenbank, in denen die Daten gespeichert sind, nacheinander in eine Backup-Datei. Dabei beginnt das Backup mit der ersten Datenseite und sichert die Datenseiten dann nacheinander weg. Um sicher zu stellen, dass das Backup hinterher auch konsistent ist (da ja nicht alle Datenseiten zur gleichen Zeit gesichert werden und die Daten während des Sicherungsvorgangs geändert werden können), sichert der SQL-Server zusätzlich auch den Abschnitt des Transaktionsprotolls mit, der während des Backups geschrieben wurde. Dadurch hat das Backup den Stand, den die Datenbank zum Abschluß des Backups hatte.

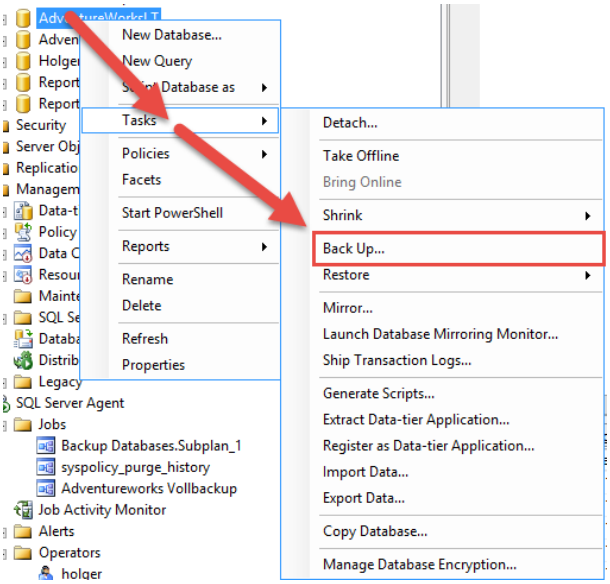
Eine Differenzsicherung sichert nur noch die Datenseiten (bzw. die Extents – Blöcke von jeweils 8 Datenseiten), die sich seit dem letzten Backup geändert haben.

Eine Transaktionsprotokollsicherung sichert den aktiven Teil des Transaktionsprotokolls und gibt vollständig abgeschlossene Transkationen wieder zum Überschreiben frei. Das Transaktionsprotokoll wird dabei nicht verkleinert, es wird nur Platz innerhalb der Datei zum Überschreiben frei gegeben. Um ein Transaktionsprotokoll zu verkleinern, muss es explizit geshrinkt (geschrumpft) werden.

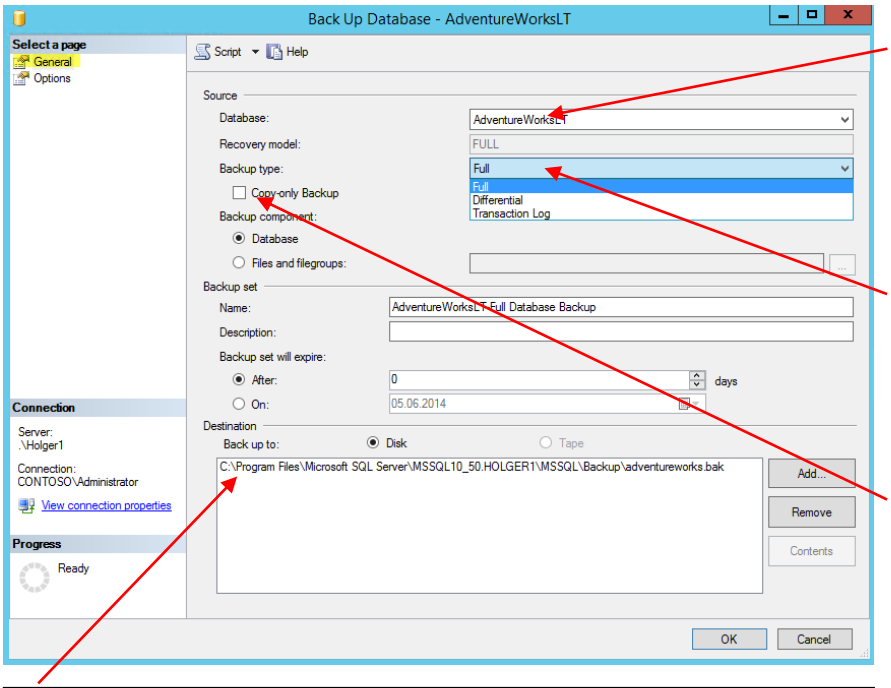
Eine Sicherungsstrategie umfasst auf jeden Fall eine regelmäßige Vollsicherung der Datenbank. Eine gute Sicherungsstrategie umfasst auch eine regelmäßige Sicherung des Transaktionsprotokolls. Das Transaktionsprotokoll versetzt Sie in die Lage, die Datenbank zu jedem beliebigen Zeitpunkt wiederherzustellen (s. auch Point in Time Recovery). Nach dem Einspielen des Vollbackups können die Transkationen aus der Transaktionsprotokollsicherung wiederhergestellt werden. Da das Transaktionsprotokoll alle Änderungen der Datenbank in chronologischer Reihenfolge enthält, kann der Restore zu jedem beliebigen Zeitpunkt abgebrochen werden und man erhält den Stand der Datenbank zu exakt dem Zeitpunkt der letzten Änderung, die eingespielt wurde.
Differenzsicherungen werden im Normalfall nur dann genutzt, wenn der Platz zum Speichern von Backups knapp ist, der verfügbare Zeitrahmen zum Wiederherstellen der Backups nicht für eine vollständige Wiederherstellung aller Transaktionsprotokolle ausreicht oder das Zeitfenster zum Sichern der Datenbanken für ein Vollbackup nicht groß genug ist.

Sichern der Datenbanken

Zum Sichern einer Datenbank rufen Sie das Kontextmenü der Datenbank auf und wählen Tasks –> Restore -> Backup. Alternativ können Sie auch das Kontextmenü des übergeordneten Ordners „Datenbanken“ auswählen.



Im Backupmenü wählen Sie den Backuptyp, den Namen des Backupsatzes (das Backup kann mehrere Backupsätze in eine Sicherungsdatei bzw. auf ein Band speichern), und das Backupmedium aus.



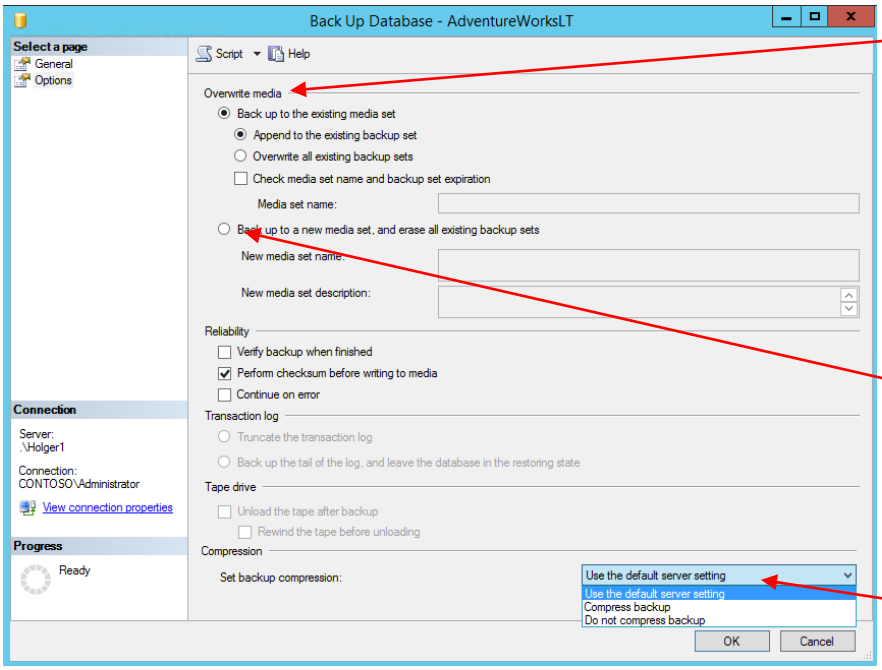
Recoverymodel – hier sehen Sie das Wiederherstellungsmodell, dass die Datenbank nutzt. Steht hier Simple oder Einfach, können Sie keine Transaktionssicherung ausführen.

Backup Type – Full (Vollständig), Differentiell (Differntial) oder Transaktionsprotokoll (Transaction Log)

Copy Only Backup – Das Transaktionsprotokoll wird nicht gekürzt, und die gesicherten Datenbankseiten werden nicht als gesichert markiert. Dadurch wird der aktuelle Sicherungsverlauf nicht gestört

Destination – Legen Sie hier fest, wohin das Backup gespeichert werden soll. Möglich sind Banlaufwerk und Disk. Es können mehrere Medien angegeben werden – SQL-Server sichert dann parallel auf alle Medien, um die Sicherung zu beschleunigen. Alternativ kann die Sicherung per TSQL auch gespiegelt werden (Mirror) (s. Sicherung parallel auf mehrere Sicherungsmedien spiegeln)

In den Optionen des Backups können Sie weitere Einstellungen vornehmen:



Overwrite Media – Wenn Sie alte Backups auf ein Backupmedium überschreiben wollen, wählen Sie „overwrite all existing Backup sets“.

Wenn Sie eine Sicherung auf mehrere Medien sichern, wird ein Backupsatz mit gemeinsamem Verwaltungs-Header erstellt. **Back up to a new media set, and erase all existing backup sets** verwendet Medien aus einem Mediensatz wieder, indem der Header neu geschrieben wird. Der Mediensatz ist damit zerstört.

Compression – Komprimiert das Backup. Verfügbar ab SQL Server 2008 Enterprise, seit 2008 R2 auch Standard-Edition. Default-Settings werden in den Server-Einstellungen vorgenommen

Eine Vollsicherung ist die Basis jedes Sicherungsmodells. Nur Vollsicherungen sind nur dann empfehlenswert, wenn ein Datenverlust tolerierbar ist, z.B. bei Session-Datenbanken für Web-Server oder Daten, die jederzeit aus dem Live-System wiederhergestellt werden können. Dann ist es auch sinnvoll, die Datenbanken im einfachen Wiederherstellungsmodus zu betreiben. Ansonsten sollte auf jeden Fall eine regelmäßige Transaktionsprotokollsicherung mit eingeplant werden. Transaktionsprotokollsicherungen haben kaum Einfluss auf die Performance des Systems, stellen aber sicher, dass bei einem Datenverlust ein Großteil der Daten wiederhergestellt werden können. Bei sehr wichtigen Daten kann auch alle 5 Minuten eine Transaktionsprotokollsicherung vorgenommen werden. Dies wirkt sich jedoch eventuell nachteilig auf die Restore-Zeiten aus, da jede Transaktionsprotokollsicherungsdatei geöffnet werden muss. Wenn Datenverlust nicht tolerierbar ist, lohnt es sich, über Hochverfügbarkeitslösungen wie Datenbankspiegelung, Failover-Cluster oder ab SQL Server 2012 Always on nachzudenken.

Sicherung parallel auf mehrere Sicherungsmedien spiegeln

Um Sicherungen auf mehrere Medien zu spiegeln, müssen Sie TSQL verwenden. Verwenden Sie dafür folgenden Befehl:

Für die Sicherung in eine Datei

Schlüsselwort für die Spiegelung

```
BACKUP DATABASE AdventureWorksLT TO
DISK = N'C:\Backup\adventureworksLT_m1.bak'
MIRROR TO DISK = N'C:\Backup\adventureworksLT_m2.bak'
WITH FORMAT, STATS;
GO
```

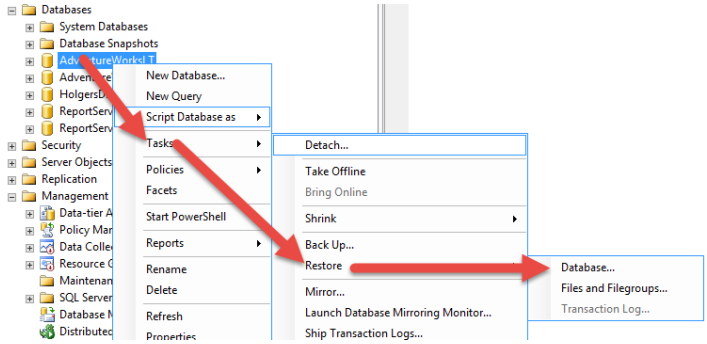
Mehr Informationen zu gespiegelten Backups:
http://www.sqlskills.com/blogs/paul/search-engine-qa-15-mirrored-backups/

Wiederherstellen von Datenbanken

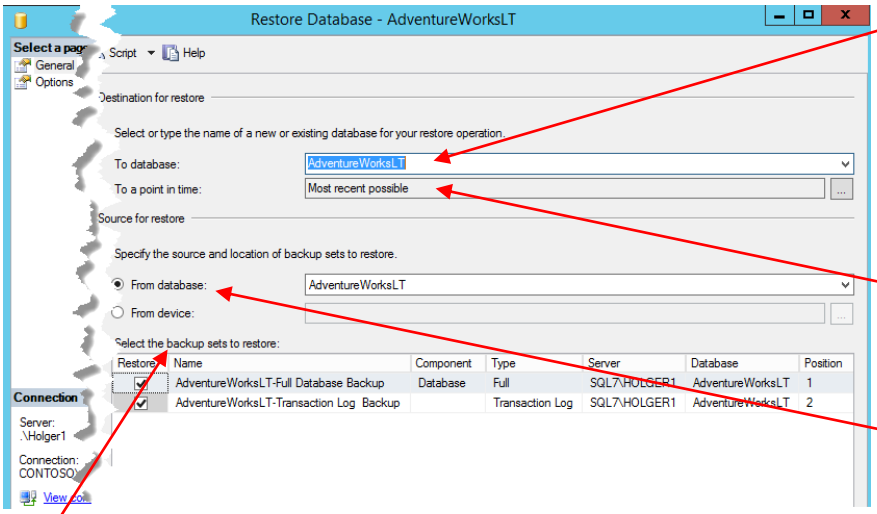
Bei der Wiederherstellung von Datenbanken muss man zwischen einem Komplettverlust oder einer einfachen Wiederherstellung unterscheiden. Soll eine Datenbank einfach nur auf einem anderen System oder unter einem anderen Namen wiederhergestellt werden (zum Anlegen einer Kopie, beispielsweise einer Entwicklungs-Umgebung), ist die Wiederherstellung einfach. Im Desaster-Fall gilt es, noch ein paar Vorbereitungen zu treffen, bevor man die Datenbank wiederherstellt.

Einfaches Wiederherstellen

Rufen Sie das Kontextmenü der Datenbank auf, und wählen Sie Tasks -> Restore -> Database.



Im Wiederherstellungs-Menü wählen Sie aus, in welche Datenbank Sie wiederherstellen werden. Außerdem können Sie hier einen Zeitpunkt festlegen, bis zu dem Sie Wiederherstellen wollen.



To database - die Datenbank, in die wiederhergestellt wird. Existiert die Datenbank, muss unter Optionen Überschreiben ausgewählt werden und es dürfen keine Benutzer mehr mit der Datenbank verbunden sein.

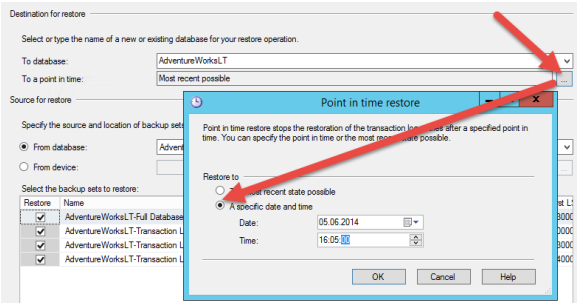
To a point in time – Die Uhrzeit, auf die die Datenbank „zurückgesetzt“ werden soll (s. unten)

From database / from device – in beiden Fällen wird das Backup von einem Sicherungsmedium geholt, aber die Sicherungshistorie wird bei „from database“ aus der msdb-Datenbank gelesen, anstatt aus der Sicherungsdatei

Backup sets to restore – Die Sicherungen, die wiederhergestellt werden sollen. Im weiteren rechten Verlauf des Fensters sieht man u.a. die LSN (Log Sequence Numbers), die im Backup gespeichert sind.

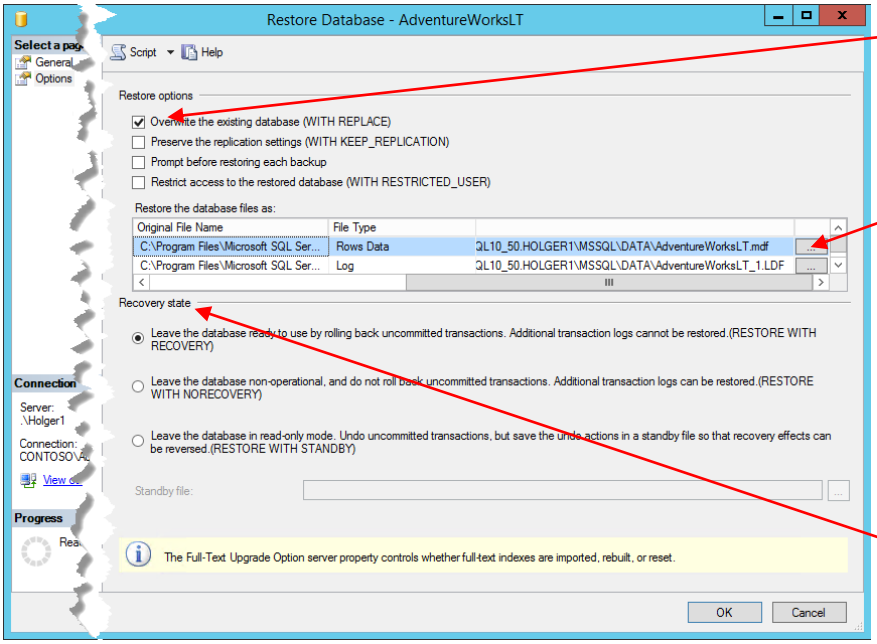
Wenn Sie die Datenbank nur bis zu einem bestimmten Zeitpunkt zurücksetzen möchten, wählen Sie Point in Time Restore und legen Sie die Uhrzeit fest, bis zu der die Datenbank wiederhergestellt werden soll. Der Point in Time Restore ist allerdings nur möglich, wenn das Wiederherstellungsmodell Vollständig oder Massenprotokolliert ist, und bei Massenprotokolliert auch nur so lange, wie kein minimal geloggter Vorgang wie BULK COPY oder SELECT INTO ausgeführt wurde. Mit dem minimal geloggten Einfügen wird ein Point in Time Restore nach dem

Masseneinfügevorgang unmöglich. Daher ist ein gängiges Vorgehen, Datenbanken nach dem Datenimport mit einem Vollbackup zu sichern, um die Point-in-time Fähigkeit wiederherzustellen.



Der Vorteil des Wiederherstellungsmodells Massenprotokolliert ist im Übrigen, dass große Datenimport deutlich schneller durchgeführt werden, wenn nicht jeder einzelne Einfügevorgang eine Schreibaktion im Transaktionsprotokoll auslöst.

In den erweiterten Optionen des Restore-Fensters legen Sie Pfadänderungen fest:



Overwrite the existing database – Standardmäßig sind Datenbanken vor dem Überschreiben geschützt. Aktivieren Sie Überschreiben hier

Restore the database files as – Hier kann der Pfad und der Dateiname geändert werden. SQL-Server erzeugt den Dateinamen aus dem Datenbanknamen – haben Sie also einen neuen Namen für Ihre Datenbank angelegt, muss nichts angepasst werden.

Recovery state – legt fest, ob die Datenbank nach dem Restore online genommen wird (s. unten)

Die Informationen der Backup-Datei

t the backup sets to restore:

Name	Component	Type	Server	Database	Position	First LSN	Last LSN	Checkpoint LSN	Full LSN	Start Date	Finish Date	Size	User Name
AdventureWorksLT-Full ...	Database	Full	SQL7...	AdventureWo...	1	33000000044100151	33000000050300001	33000000044100151	33000000044100151	05.06.2014 ...	05.06.2014 17:1...	5521408	CONTOS...
AdventureWorksLT-Tran...		Transaction Log	SQL7...	AdventureWo...	2	33000000002400001	33000000051000001	33000000044100151	33000000044100151	05.06.2014 ...	05.06.2014 17:1...	1064960	CONTOS...

Option	Bedeutung
Position	Position im Backup-Medium – relevant vor allem für Bänder
First LSN	Die Log Sequence Number (ID) der ersten Transaktion im Backup
Last LSN	Die Log Sequence Number (ID) der ersten Transaktion im Backup
Checkpoint LSN	Das Backup beginnt mit einem Checkpoint (dem synchronisieren der Daten im Hauptspeicher mit denen auf der Festplatte). Der Checkpoint wird ebenfalls im Log festgehalten. Dies ist der aktuellste Checkpoint im Backup
Full LSN	Laut Dokumentation die Checkpoint-LSN des letzten Full Backups. Tatsächlich anscheinend die LSN des letzten Checkpoints
Start Date	Uhrzeit, zu der das Backup begonnen hat - entspricht dem Zeitpunkt, zu dem die Checkpoint-LSN geschrieben wurde
Finish Date	Uhrzeit, an der das Backup beendet wurde
Size	Größe des Backups in Kilobyte

Weiterführende Informationen

http://www.red-gate.com/community/books/sql-server-transaction-log-management

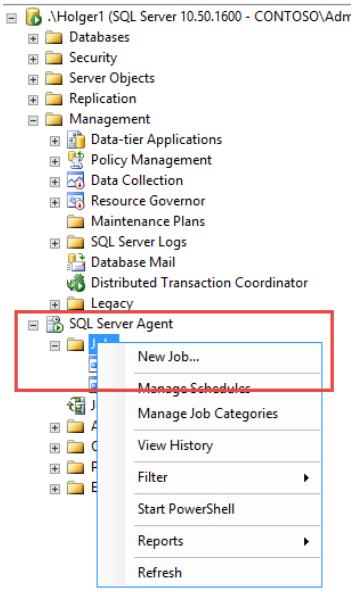
Wiederherstellung nach Totalverlust

Nach einem Totalverlust des Servers ist es wichtig, dass Sie eine Sicherung der master- und msdb-Datenbank haben. Installieren Sie Ihren SQL-Server neu, und stellen Sie die master- und msdb-Datenbank aus dem Backup wieder her. Nutzen Sie hierfür „Restore from device (Medium)“. Geben Sie das Medium an, auf dem sich Ihre Master-und msdb-Datenbanken befindet. Nach der Wiederherstellung steht Ihnen der Sicherungsverlauf wieder zur Verfügung und Sie können Ihre Datenbank ganz normal wiederherstellen.

SQL-Server Agent – Aufgaben automatisieren

Mit dem SQL-Server Agent können Sie Aufgaben automatisieren. Der Agent ist ein zusätzlicher Dienst auf dem Windows-Server, der ähnlich wie der Windows Task Scheduler (geplante Aufgaben) automatisiert zu bestimmten Zeiten Jobs auf dem SQL-Server startet. Das kann ein Datenbank-Backup sein, aber auch beliebige andere Aufgaben. Der SQL-Server Agent kann auch auf das Betriebssystem zugreifen. Der Agent ist erst mit der Standard-Version des SQL-Servers verfügbar und ist ohne den SQL-Server nicht lauffähig, da seine komplette Konfiguration im SQL-Server (in der Systemdatenbank msdb) abgelegt ist.

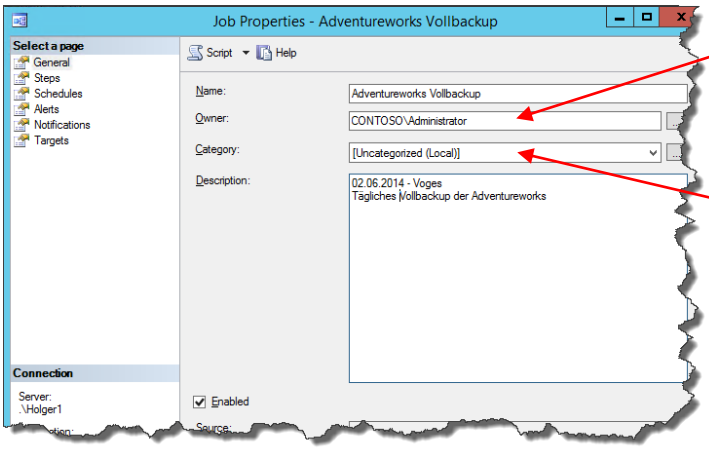
Um mit dem SQL-Server Agent eine Aufgabe zu automatisieren, müssen Sie einen Job und einen Zeitplan festlegen. Ein Job wiederum besteht aus Jobsteps, die Sie einzeln konfigurieren können. Im Folgenden werden wir beispielhaft die Automatisierung eines Vollbackups zeigen.



Wechseln Sie zuerst im Management Studio in den SQL-Server Agent Knoten. Achten Sie darauf, dass der SQL-Server Agent läuft. Ein kleiner grüner Pfeil auf dem Agent zeigt Ihnen an, dass der Agent-Dienst gestartet ist.

Gehen Sie jetzt in den Unterknoten Jobs, öffnen Sie das Kontextmenü der Jobs und wählen Sie „New Job“ aus. Hier legen Sie einen neuen Job an.

Im folgenden Fenster legen Sie einen Jobnamen fest, sowie den Owner des Jobs. Alles Transact-SQL-Befehle des Jobs werden standardmäßig im Benutzerkontext (mit den Berechtigungen) des hier eingetragenen Owners durchgeführt. Nutzen Sie außerdem die Möglichkeit, eine Beschreibung zu hinterlegen, am besten mit Datum und Kommentar. Die Kategorie können Sie frei lassen. Sie wird für Standard-Jobs wie Replikation usw. verwendet.



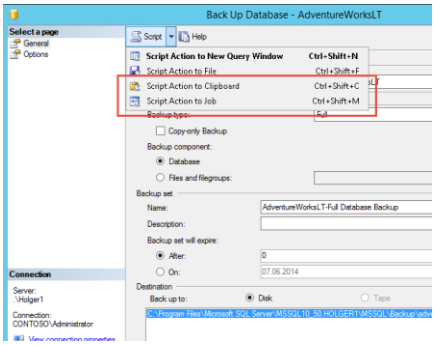
Owner - Besitzer des Jobs. Ist der Job ein SQL-Script, bestimmt er den Benutzer, unter dem der Job ausgeführt wird.

Category – Wird vom System verwendet, um z.B. Replikationsjobs festzulegen. Hier ist keine Eingabe notwendig.

Als nächstes legen Sie die Jobsteps fest. Dazu wechseln Sie in das Menü Steps und legen einen neuen Jobstep an. Klicken Sie hier auf Neu oder New. Darauf öffnet sich ein Fenster, in dem Sie den neuen Job anlegen können.

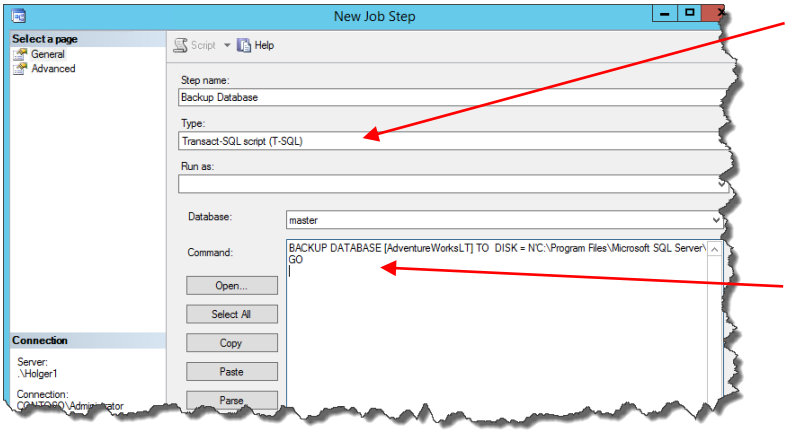
Im New Job Step Fenster legen Sie einen neuen Job vom Typ Transact-SQL an. Wählen Sie dafür den entsprechenden Typ aus. Das Kommando zum Sichern der Datenbank erzeugen wir über das

Management-Studio. Wechseln Sie hierzu im Management Studio auf Ihre Datenbank, und wählen Sie aus dem Kontextmenü Tasks -> Backup aus. Im Backup-Fenster wählen Sie Ihre Backup-Optionen und wählen dann aus dem Backup Fenster oben links die Option „Script“.



Die Option Script finden Sie fast überall im Management-Studio. Sie erzeugt aus den von Ihnen im Fenster eingegebenen Optionen ein SQL-Script, dass Sie in ein neues Abfrage-Fenster speichern, in eine Datei kopieren, in die Zwischenablage speichern oder direkt als neuen Job verwenden können. Wir kopieren das Script im Folgenden einfach in das Kommando-Fenster, daher wählen Sie als „Script Action to Clipboard“.

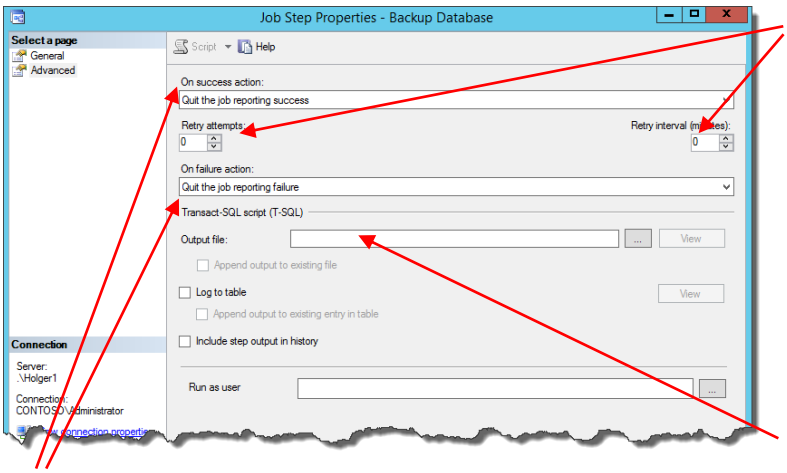
Kopieren Sie im Job-Step das erzeugte Script jetzt in das Kommando-Fenster. Das Kommando BACKUP DATABASE sichert die Datenbank in eine Datei auf der Festplatte. Das N vor dem Dateinamen bedeutet, dass SQL-Server den Dateipfad als Unicode interpretiert – es können beliebige Sonderzeichen verwendet werden.



Der Typ des Jobs bestimmt, welche Optionen Ihnen als Kommandos zur Verfügung stehen. Mit Transact-SQL Script können Sie beliebigen SQL-Code ausführen.

Das Kommando, das ausgeführt werden soll. SQL-Statements können im Management-Studio über den Button Scripts erzeugt werden.

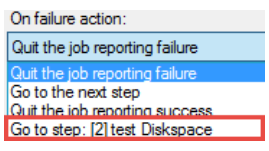
Unter Advanced können Sie jetzt noch weitere Optionen für den Ablauf des Job-Steps festlegen:



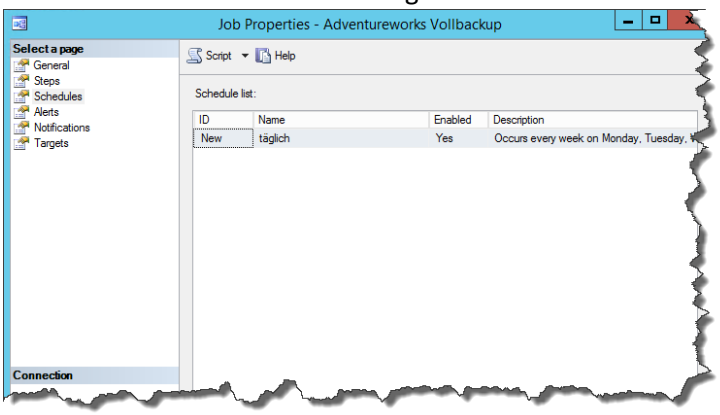
Retry attempts: Retry Interval – Wenn der Job fehlschlägt, kann hier definiert werden, ob und in welchen Intervallen ein neuer Versuch durchgeführt werden soll. Sinnvoll beispielsweise, wenn der Job von einem andere Job abhängt, und die Ergebnisse noch nicht vorliegen (Import-Dateien usw…)

Output-File – Für Jobs, die keine direkte Ausgabe an den SQL-Server liefern, wie Kommandozeilen-Jobs, kann die Ausgabe in eine Datei umgeleitet werden.

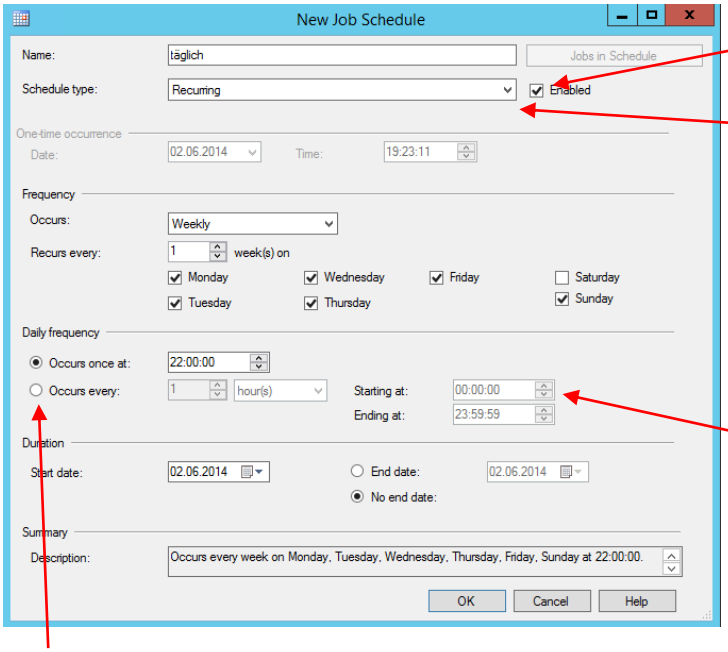
On Success action : On Failure action – Hier kann definiert werden, wie ein Fehler des Jobs behandelt wird. Hier kann festgelegt werden, dass bei einem Fehler ein anderer Job-Step angesprungen wird – Z.B. zur Fehlerbehandlung. Dadurch sind komplexe Abläufe realisierbar.



Nun legen Sie unter „New Job Schedule“ fest, zu welchen Zeiten der Job gestartet werden soll. Hier haben beliebige Möglichkeiten – Täglich, wöchentlich, 2-wöchentlich, beim Start des SQL-Server Agents, mit Enddatum oder ohne. Und Sie können auch mehrere Pläne kombinieren. In den meisten Fällen sind die Standardeinstellungen aber ausreichen.



Schedule List - Legen Sie hier mehrere Pläne an, wenn Sie mehrere abweichende Konfigurationen haben



Enabled – hier können Sie den Job deaktivieren

Schedule Type – folgende Optionen:

- Einmalig
- Wiederholend
- Beim SQL-Server Agent Start
- Wenn Server im Leerlauf ist

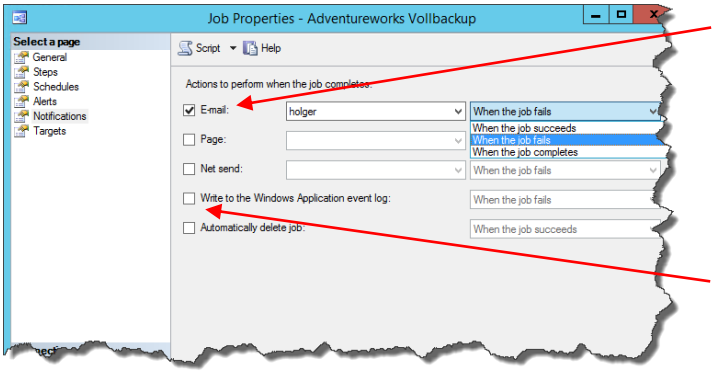
Der Bedingung, wann der Server im Leerlauf ist, wird in den Agent-Optionen bestimmt.

Starting at : Ending at – Definieren Sie, zu welchen Tageszeiten der Job starten soll.

Daily Frequence – Legt fest, ob ein Job mehrmals täglich laufen soll – beispielsweise Transaction Log Backups

Als letztes legen Sie fest, ob und wie der SQL-Server nach Beendigung des Jobs benachrichtigen soll. Dabei ist nur eine Benachrichtigung über den gesamten Job möglich. Einzelne Job-Step-Benachrichtigungen müssen manuell implementiert werden. Legen Sie dafür fest, wie Sie benachrichtigt werden wollen und bei welchem Ereignis – Fehler, Erfolg oder immer. Zusätzlich können Sie den Job automatisch löschen lassen (sinnvoll bei einmaligen Jobs) und einen Eintrag ins Windows Anwendungs-Protokoll schreiben lassen.

Für email-Benachrichtigungen müssen Sie vorab 2 Dinge konfigurieren – richten Sie SQL-Server Datenbankmail ein (ein im SQL-Server eingebauter mail-Server, der emails per SMTP-Mail-Relay weiterleiten kann), und hinterlegen Sie die Emailadressen, die benachrichtigt werden sollen, als Operator.

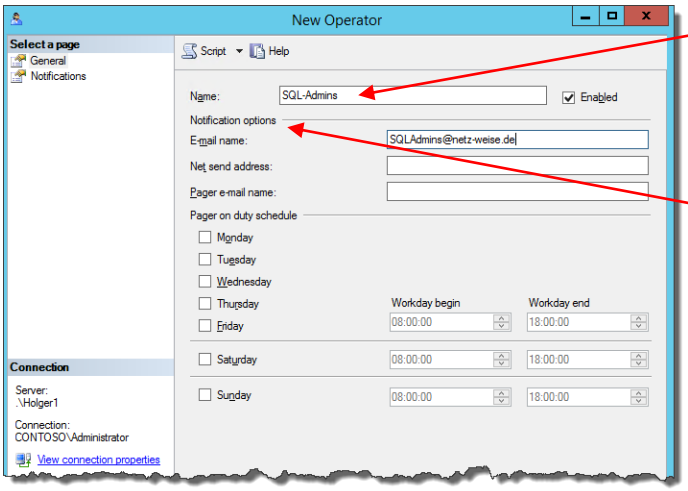


E-mail – hier wird der Operator hinterlegt, der benachrichtigt werden soll. (s. unten). Außerdem definieren Sie, wann eine Benachrichtigung geschickt werden soll (Erfolg, Fehler, immer)

Write to the Application event log – Erzeugt nach jeder Ausführung einen Eintrag im Anwendungsprotokoll

Anlegen eines Operators

Ein Operator ist einfach eine auf dem Server hinterlegte E-Mail-Adresse, die für Benachrichtigungen verwendet werden kann. Wechseln Sie hierzu im Agent auf Operators (Operatoren) und wählen Sie im Kontextmenü „New Operator“.



Name – Der Name, der zur Benachrichtigung verwendet wird. SQL-Server zeigt nicht die mail-Adresse an, sondern den Operatornamen

Notification Options – hier legen Sie unter E-Mail name die Mail-Adresse fest. Es kann nur eine Adresse angegeben werden. Nutzen Sie, wenn Sie mehrere Nutzer benachrichtigen wollen, E-Mail-Verteilerlisten.

Alle weiteren Optionen sind heutzutage nicht mehr relevant – es sei denn, sie nutzen noch Pager. ;-)

SQL Server Performanceoptimierung

Indizes

Indexe werden zur schnellen Datensuche verwendet. Man benötigt Sie, wenn in einer Tabelle große Datenmengen stehen, aber nur nach einzelnen Datensätzen gesucht wird, z.B. einem ganz bestimmten Namen, einer Kundennummer oder allen Datensätzen innerhalb eines bestimmten Zeitraums.

Wenn kein Index auf einer Tabelle existiert, werden die Daten in Form eines sogenannten Heaps (Haufen) gespeichert - ein Datensatz wird einfach an den nächsten gehängt. Da die Daten im Normalfall nicht sortiert erstellt werden, sind die Daten innerhalb der Tabelle auch komplett unsortiert. Wird nach einem bestimmten Datensatz gesucht (der Kunde mit dem Namen „Alfreds Futterkiste“), muß der SQL-Server die komplette Tabelle einlesen und alle Datensätze überprüfen, ein sogenannte Tabellen-Scan. Je größer die Tabelle ist, desto aufwendiger ist dieser Vorgang.

Um Daten in einer Tabelle zu sortieren, kann man einen sogenannten gruppierten (engl. clustered) Index anlegen. Dabei wird die Tabelle selber in eine Sortierreihenfolge gebracht. Ein gruppierter Index ist normalerweise auch eindeutig (unique), d.h. er erzwingt, dass jeder Datensatz (z.B. die Kundennummer oder eine Bestellnummer) nur genau 1 Mal vorkommen darf. Der gruppierte Index sortiert die Tabelle selber. Zusätzlich zur Sortierung wird noch ein sogenannter B-Baum angelegt, der die Spalte, über die der Index angelegt wird, schneller durchsucht werden kann. Den B-Baum (balanced tree) kann man sich vorstellen wie einen Baum, dessen Wurzel ungefähr mit dem Datensatz genau in der Mitte der sortieren Daten steht. Von da aus werden die Daten im Baum nach rechts und links sortiert, bis man am eigentlichen Datensatz ankommt. Durch diese Vorsortierung muß nicht jeder einzelne Datensatz angefasste werden, sondern man kommt, wenn man einen einzelnen Datensatz sucht, viel schneller zum Zieldatensatz.

Dummerweise kann man eine Tabelle nur einmal sortieren. Wenn Sie aber mehr als 1 Spalte enthält, was der Normalfall sein sollte, liegt nur eine Spalte sortiert vor, alle anderen Spalten sind wieder unsortiert. Um auch andere Spalten schneller durchsuchen zu können, gibt es den nicht-gruppierten Index, von dem es bis zu 253 Stück pro Tabelle geben kann.

Ein Nicht-Gruppierter Index verhält sich wie ein gruppierter Index, sortiert als eine oder mehrere Spalten nach einem bestimmten Kriterium in Form eines B-Baums. Der Unterschied zum gruppierten Index ist, dass ein nicht-gruppierter Index eben nicht die Daten selber sortiert. Im nicht-gruppierten Index werden Kopien aller Spalten, die im Index enhalten sein sollen, wieder in Form eine B-Baums angelegt. Die Datensätze im nicht-gruppierten Index enhalten dann einen Zeiger auf den eigentlichen Datensatz, so dass nicht nur die Spalten, die im Index angegeben sind, angezeigt werden können, sondern auch die Daten, die in der Tabelle sind.

Wann benötigt man Indizes?

Indizes werden benötigt, um einzelne Datensätze in großen Datenmengen schnell zu lokalisieren. Daher gilt, dass Indizes grundsätzlich nur beim Lesen von Daten hilfreich sind. Beim Schreiben ist genau das Gegenteil der Fall – hier ist ein Index grundsätzlich von Nachteil. Neben den eigentlichen Daten muß auch der oder die Indizes geschrieben werden. Außerdem kann in einem Heap ein neuer Datensatz einfach „auf den Haufen geworfen“ werden, er wird also einfach hinten angehängt. Im Prinzip ist das wie in meiner Wohnung: Wenn es schnell gehen soll, werfe ich alles auf einen Haufen, aber wehe, ich muß etwas wiederfinden. Will man aber die Daten also gar nicht oft anfassen, aber viel schreiben, ist ein Index für die Performance eher nachteilig. Sucht man aber oft nach Daten, z.B.

mit einer Where-Klausel, oder werden Daten oft gejoined, ist ein Index immer ein Boost für die Performace – zumindest, wenn die Daten sich ausreichend unterscheiden und nicht eh alle fast identisch sind.

Grundsätzlich sollte das Anlegen von Indizes eine Sache des Programmierers der Datenbank sein – er kennt seine Datenbank am besten und sollte wissen, auf welche Daten häufig zugegriffen wird. Leider ist das aber nicht immer der Fall. Da Indizes auf die Datenstruktur keine Auswirkungen haben, kann auch ein Administrator fehlende Indizies nachpflegen oder überflüssige Indizes löschen, um Schreibvorgänge zu beschleunigen oder die Datenbank zu verkleinern.

Index-Pflege und Optimierung

Indizes benötigen grundsätzlich Wartung. Das liegt daran, dass Indizes fragmentieren – das tun sie naturgemäß.

Man unterscheidet dabei innere und äußere Fragmentierung. Beide basieren auf dem gleichen Problem – Page Splits. Jedes Mal, wenn neue Daten geschrieben werden, müssen diese einer Datenseite zugeordnet werden. Da ein Index sortiert ist, muss er die neuen Daten an den Ort schreiben, an den Sie logisch im Index gehört. Dafür sucht der SQL-Server die Datenseite, die an den Datensatz anschließt. Ist meine Tabelle z.B. nach Namen sortiert, und ich füge einen neuen Datensatz „Schmidt“ ein, dann muss der SQL-Server die Datenseite finden, auf der die vorhergehenden und nachfolgenden Namen sitzen und den neuen Datensatz auf diese Seite schreiben. Ist diese Datenseite jedoch voll, macht SQL-Server einen Page-Split: Es wird eine neue Seite angelegt, die Daten der alten Datenseite werden 50-50 zwischen den beiden Datenseiten aufgeteilt (natürlich unter Beibehaltung der Reihenfolge), und der neue Datensatz wird geschrieben.

Dummerweise wird die neue Datenseite physikalisch auf der Festplatte nicht unbedingt direkt an die alte Datenseite anschließen. Dadurch geraten die Daten durcheinander. Außerdem sind die Datenseiten nun nicht mehr ideal gefüllt. Nach vielen Page-Splits wird man auf der Festplatte deutlich mehr halbgefüllte Datenseiten finden, und die SQL-Server muss zum Lesen der gleichen Nutzdaten deutlich mehr Leerdaten lesen. Den ersten Fall – die Datenseiten befinden sich nicht mehr in der richtigen Reihenfolge – bezeichnet man als äußere Fragmentierung, den zweiten Fall als innere Fragmentierung.

Die Fragmentierung können Sie sich mit

```
SELECT * FROM
sys.dm_db_index_physical_stats(db_id('AdventureWorksLT'),NULL,NULL,NULL,'Sampled')
```

anzeigen lassen. In der Ausgabe finden Sie den Tabellen- bzw. Index-Typ, die Fragmentierung in Prozent, die Anzahl der fragmentierten Seiten (alles unter 9 kann ignoriert werden), und den in den Datenseiten durchschnittlich verwendeten Speicherplatz.

Ein Index mit einer Fragmentierung bis zu 30% sollte neu organisiert werden (rebuild), ein Index über 30% neu erstellt (rebuild). Glücklicherweise müssen Sie sich um all das nicht kümmern, wenn Sie die fertigen Skripte von Ola Hallengreen nutzen. Die Index-Wartungsscripte kümmern sich um alles selbständig und entscheiden, ob ein Index überhaupt defragmentiert werden muss und wenn ja, wie. Installieren Sie einfach das Script MaintenanceSolution.sql. Es legt eine Reihe von SQL-Jobs an. Sie brauchen nur noch einen Zeitplan festzulegen.

SQL Server Maintenance Solution
http://ola.hallengren.com/

Die Scripte sind tausendfach im Einsatz und sind Klartext-SQL-Scripte. Insofern sind sie überprüft und frei von Schadcode.

Fehlende Indizes bestimmen

SQL-Server führt intern Listen, in denen protokolliert wird, wie oft Indizes benötigt worden wären, aber nicht vorhanden waren. Diese Daten kann man nutzen, um fehlende Indizes zu finden. Die entsprechenden Views und Funktionen lauten:

- sys.dm_db_missing_index_details
- sys.dm_db_missing_index_group_stats(Index)
- sys.dm_db_missing_index_group_handle
- sys.dm_db_missing_index_columns(handle)

Zusammengefasst ergibt sich daraus folgende Abfrage, die fehlende Indizes anzeigt:

```
SELECT mig.*, statement AS table_name,
column_id, column_name, column_usage
FROM sys.dm_db_missing_index_details AS mid
CROSS APPLY sys.dm_db_missing_index_columns (mid.index_handle)
INNER JOIN sys.dm_db_missing_index_groups AS mig ON mig.index_handle =
mid.index_handle
ORDER BY mig.index_group_handle, mig.index_handle, column_id;
```

Nicht genutzte Indizes finden

SQL-Server zeigt auch das Nutzungsverhalten von Indizes an. Daraus kann man die Indizes bestimmen, die nicht genutzt werden.

```
sys.dm_db_index_operational_stats:
SELECT OBJECT_NAME(A.[OBJECT_ID]) AS [OBJECT NAME],
       I.[NAME] AS [INDEX NAME],
       A.LEAF_INSERT_COUNT,
       A.LEAF_UPDATE_COUNT,
       A.LEAF_DELETE_COUNT
FROM   SYS.DM_DB_INDEX_OPERATIONAL_STATS (db_id(),NULL,NULL,NULL ) A
       INNER JOIN SYS.INDEXES AS I
         ON I.[OBJECT_ID] = A.[OBJECT_ID]
            AND I.INDEX_ID = A.INDEX_ID
WHERE  OBJECTPROPERTY(A.[OBJECT_ID],'IsUserTable') = 1

sys.dm_db_index_usage_stats
SELECT OBJECT_NAME(S.[OBJECT_ID]) AS [OBJECT NAME],
       I.[NAME] AS [INDEX NAME],
       USER_SEEKS,
       USER_SCANS,
       USER_LOOKUPS,
       USER_UPDATES
FROM   SYS.DM_DB_INDEX_USAGE_STATS AS S
       INNER JOIN SYS.INDEXES AS I ON I.[OBJECT_ID] = S.[OBJECT_ID] AND I.INDEX_ID
= S.INDEX_ID
WHERE  OBJECTPROPERTY(S.[OBJECT_ID],'IsUserTable') = 1
       AND S.database_id = DB_ID()
```

Statistiken

Der SQL-Server weiß grundsätzlich nicht exakt, welche Daten in welchen Tabellen wie stehen. Wenn ein Select-Statement ausgeführt wird, tritt der sogenannte Query-Analyzer in Aktion, der versucht, die beste Strategie zum Auslesen der Daten zu ermitteln. Dafür schaut er sich z.B. an, ob Indexe existieren, ob Spalten eindeutig sind, ob eine Where-Klausel im Statement vorkommt usw. Dann erstellt er einen Plan, um die Daten zu lesen. Um die bestmögliche Strategie zum Auslesen zu

entwerfen, muß der SQL-Server dazu ungefähr wissen, was für Daten in den einzelnen Spalten stehen, und ob diese z.B. eindeutig (unique) sind, also jeder Datenwert einmalig ist, oder ob einzelne Datenwerte sehr häufig redundant sind, wie z.B. Postleitzahlen.

Diese Informationen erhält der Query Analyzer über sogenannte Statistiken. Der SQL-Server führt zu jeder Spalte und jedem Index eine Statistik, die diese Information (Eindeutigkeit, Anzahl der Datensätze innerhalb der Tabelle usw) enthält. Diese Statistiken werden im Normallfall automatisch gepflegt, und zwar immer dann, wenn sich 20% der Datensätze innerhalb einer Tabelle oder eines Index geändert haben.

Oftmals ist die automatische Aktualisierung des SQL-Servers nicht ausreichend, speziell bei großen Tabellen, bei denen 20% eine sehr große Menge an neuen Daten bedeutet. Glücklicherweise übernehmen die Scripte von Ola Hallengreen auch die Aktualisierung der Statistiken. (s.o.) Mit SQL-Server 2012 gibt es außerdem die Möglichkeit, den Schwellwert dynamisch anzupassen. Dazu muß das Traceflag 2371 gesetzt werden.

```
dbcc traceon (2371)
```

http://support.microsoft.com/kb/2754171

Um alle Statistiken einer Datenbank zu aktualisieren, stellt SQL-Server eine stored procedure zur Verfügung:

```
exec sp_updatestats
```

Waitstats

Um Probleme im SQL-Server zu identifizieren, wie z.B. lange Wartezeiten auf die Festplatte, hohe CPU-Last, zu wenig Arbeitsspeicher usw., kann man beim SQL-Server sogenannte Waitstats (Wait-Statistiken) abfragen. Waitstats zeigen an, worauf der SQL-Server während seines letzten Neustarts am längsten warten musste. Wenn der SQL-Server also langsam läuft, ist ein erster Schritt, die Fehlerursache zu finden, in die Waitstats zu gucken. Die Wait-Statistiken lassen sich mit Hilfe der DMV sys.dm_os_wait_stats abfragen:

```
select * from sys.dm_os_wait_stats
```

Eine Auflistung der Bedeutung der einzelnen Wait-Typen und Ihrer Bedeutung findet man hier:
http://blogs.msdn.com/b/psssql/archive/2009/11/03/the-sql-server-wait-type-repository.aspx

Eine Menge weitere Informationen finden sich bei Brent Ozar:
http://www.brentozar.com/sql/wait-stats/

Best Practices

SQL-Server umbenennen
http://msdn.microsoft.com/de-de/library/ms143799.aspx

Copy User Script
http://support.microsoft.com/kb/918992/de

Tools

Profiler

Der SQL-Server Profiler ist ein Werkzeug ähnlich einem Netzwerkmonitor. Mit Hilfe des Profilers ist es möglich, zu protokollieren, was auf dem SQL-Server passiert. Sie können alle Abfragen sehen, die Benutzer, die Sie ausführen, die Clients von denen die Abfragen kommen, und jede Menge weitere Informationen zu Blockings, Deadlocks usw. Außerdem ist es möglich, die Aktionen, die auf dem SQL-Server laufen, zu speichern und erneut abzuspielen (Replay). Das ist sinnvoll, um Fehlersituationen nachzustellen, oder Performance-Optimierungen zu überprüfen. Ein Handbuch über den SQL-Server Profiler kann man bei Redgate als pdf kostenlos herunter laden.

Mastering SQL Server Profiler
http://www.red-gate.com/community/books/mastering-sql-server-profiler

Management Studio

Das SQL-Server Management-Studio ist seit SQL-Server 2005 das zentrale Verwaltungstool für die relationale Engine. Es wird bei der Installation des SQL-Servers installiert, kann aber aber nachträglich oder alleine auf den Administrations-Clients installiert werden. Außerdem gibt es vom SQL-Server Management-Studio noch eine kostenlose Version, die für SQL-Server Express ausgeliefert wird. Mit Ihr ist es aber nicht möglich, erweitere Funktionen wie den SQL-Server Agent zu konfigurieren.

SQL-Server Express Edition (Management-Studio kann einzeln ausgewählt werden)
http://www.microsoft.com/de-de/server-cloud/products/sql-server-editions/sql-server-express.aspx#fbid=0dFkXAwxP9n

WhoIsActive

Um eine Übersicht über auf dem Server laufende Prozesse zu erhalten, gibt es den SQL-Server Aktivitätsmonitor. Er zeigt die laufenden Prozesse auf dem Server an. Alternativ können Sie auch über die gespeicherte Prozedur sp_who2 gehen. Wenn Sie richtig viele Informationen benötigen, bieten sich aber die „who is Active“-Scripte von Adam Machanic an. Er pflegt einen Satz von Scripten zu verschiedenen SQL-Server Versionen und aktualisiert diese ständig. Sie geben ein Füllhorn von Informationen über Ihre SQL-Server Instanzen aus.

http://sqlblog.com/blogs/adam_machanic/archive/tags/who+is+active/default.aspx

Reporting Services

Hochverfügbare SQL-Server