Nico Weinreich

Entwicklung eines alternativen Backup- und Recoveryverfahrens für relationale Datenbanken unter Microsoft SQL Server 2005

eingereicht als

DIPLOMARBEIT

an der

HOCHSCHULE MITTWEIDA

UNIVERSITY OF APPLIED SCIENCES

Fakultät für Mathematik/Naturwissenschaften/Informatik

Mittweida, 2010

Erstprüfer: Herr Prof. Dr.-Ing. Wilfried Schubert
Zweitprüfer: Frau Dipl.-Inf. Ines Feuerstein

Vorgelegte Arbeit wurde verteidigt am:

Bibliographische Beschreibung:

Weinreich, Nico:
Entwicklung eines alternativen Backup- und Recoveryverfahrens für relationale Datenbanken unter Microsoft SQL Server 2005. – 2010. – 146 S.
Mittweida, Hochschule Mittweida, Fakultät für Mathematik/Naturwissenschaften/Informatik, Diplomarbeit, 2010

Referat:

Im Rahmen dieser Diplomarbeit wird ein Programm entwickelt, welches vorhandene Backupstrategien ergänzen soll. Oft ist es erforderlich, im Falle eines Disasters, den entstandenen Datenverlust durch eine Wiederherstellung in kürzest möglicher Zeit zu beheben.

Das Programm soll neben Standardfunktionalitäten, wie Sicherung und Wiederherstellung von Datenbanken, die Möglichkeit bieten, aus diesen angefertigten Sicherungen einzelne Objekte, insbesondere Tabellen, wiederherstellen zu können.

Dabei werden Konsistenzfaktoren, sowie die Zugriffsmöglichkeit auf die Schemadaten und das Transaktionsprotokoll von SQL Server untersucht, und unter Berücksichtigung dieser Erkenntnisse eine Backup- und Restorefunktionalität entworfen.

Abschließend soll die Funktionalität im Vergleich mit anderen Produkten bewertet, sowie eine Einschätzung für den Praxiseinsatz getroffen werden.

Inhaltsverzeichnis

Verzeichnisse

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

1. Einführung

1.1. Gründe für ein alternatives Backup- und Recoveryverfahren

Datenbanken werden bereits seit vielen Jahren dort eingesetzt, wo große Datenmengen effizient gespeichert werden sollen. Hierbei können die Daten nach den Erfordernissen der Anwendung oder des Unternehmens strukturiert und bereitgestellt werden. Heute hat nahezu jeder einmal Kontakt mit einer solchen Datenbank gehabt. Dazu gehören nicht nur der Internetshop, über den man sich seine Bücher kaufen kann – auch bei Suchmaschinen, Telefon- und Kabelnetzbetreibern und unzählig anderen Anbietern sind Datenbanken im Einsatz, nicht zuletzt gehört auch ein Krankenhaus mit den Krankendaten seiner Patienten dazu. Es ist auch nicht selten der Fall, dass sich diese Datenbanken, besonders im Bereich Data Warehousing – hier werden diese in der Regel für Auswertungen und Reportings genutzt – im Bereich mehrerer Gigabyte bewegen. Durch Historisierung[1] der Daten in einem Data Warehouse wachsen diese Datenbanken oft um mehrere hundert Megabyte am Tag.

Microsoft SQL Server 2005 (weiterhin SQL Server) bietet lediglich ein ganzheitliches Konzept zur Sicherung und Wiederherstellung einer Datenbank. Mit den Mitteln, die SQL Server bietet, kann eine Datenbank nur vollständig gesichert werden. „Vollständig“ muss in diesem Zusammenhang im Sinne von „Sicherung aller Objekte der Datenbank“ verstanden werden [Mic083]. Hierbei wird ein komplettes Abbild der hinter der Datenbank liegenden Datei erstellt [Woo07 S. 137]. Sicher mag dies die Handhabbarkeit vereinfachen, da man meist nur mit einem Objekt, der Backupdatei, arbeitet und entsprechende Dialogsysteme für Sicherung und Wiederherstellung demnach recht einfach gestaltet sind. Die sehr umfangreiche SQL Server Dokumentation im Microsoft Developer Network (weiterhin MSDN) zeigt ebenfalls nur die genannte Variante zur Sicherung und Wiederherstellung auf [Mic082]. Darüber hinaus ist ein Export einzelner Tabellen oder Schemaobjekte aus einer Datenbank nur durch erhöhten Aufwand durchführbar. Tabellen lassen sich zwar durch einen Befehl „bcp“ auf der Windows-Kommandozeile exportieren [Mic084] [Woo07 S. 107ff] [Bau06 S. 269ff], Schemaobjekte, wie die Struktur einer Tabelle, können aber nur über die grafische Oberfläche von SQL Server in Form eines Skripts generiert werden, welches dann durch den Anwender als Textdatei gespeichert werden muss [Mic085] [Bau06 S. 83]. Daraus folgt, dass für diese Arbeit meist die grafische Oberfläche zur Anwendung kommen muss, somit eine Automatisierung eines Sicherungsprozesses ausgeschlossen ist und der Aufwand

[1] Hier: die Aufbewahrung älterer Datenstände zu bestimmten Stichtagen

zur Sicherung mehrerer Objekte linear steigt. Zuletzt bleibt auch der Aspekt der Verwaltung der so erzeugten Daten offen, welche ebenfalls durch den Anwender geschehen muss. In der Praxis und aus eigener Erfahrung zeigt sich so, dass Sicherungsprozesse auf Basis der Mittel von SQL Server einen entscheidenden Nachteil haben. Die Wiederherstellung eines einzelnen Objekts aus einer gesamten Sicherung ist mit Standardmitteln von SQL Server nicht möglich.

Eine Wiederherstellung von Datenbanken oder Teilen einer Datenbank kann jederzeit erforderlich werden. Die IT-Grundschutz-Kataloge des Bundesamtes für Sicherheit in der Informationstechnik geben einen Überblick über typische und häufige Ursachen eines Datenverlusts. Auszugsweise werden einige Gründe genannt [Inf08]:

Organisatorische Mängel

- Fehlende oder unzureichende Aktivierung von Datenbank-Sicherheitsmechanismen
- Mangelhafte Konzeption einer Datenbank

Menschliches Versagen

- Fehlerhafte Administration von Zugangs- und Zugriffsrechten
- Fehlerhafte Administration einer Datenbank oder eines Datenbankmanagement-systems
- Unbeabsichtigte Datenmanipulation

Technisches Versagen

- Ausfall einer Datenbank
- Verlust von Daten einer Datenbank
- Verlust der Datenbankintegrität/-konsistenz

Vorsätzliches Handeln

- Unberechtigte IT-Nutzung
- Manipulation an Daten oder Software bei Datenbanksystemen
- SQL-Injection

Nun müsste man für die beiden beispielhaften Fälle:

- Wiederherstellung einer gelöschten Lookup-Tabelle
- Einsicht in einen historisierten Datenstand in einem Data Warehouse

ein vorhandenes Backup vollständig wiederherstellen – gegebenenfalls in einer zusätzlichen Datenbank, um die vorhandenen Daten nicht zu gefährden – und kann sich dann aus dieser Wiederherstellung die gewünschte Tabelle herausnehmen. Wenn es sich, wie eben im Beispiel, um eine kleine Referenztabelle handelt, die in einer produktiven Datenbank gelöscht

wurde und in möglichst kurzer Zeit wieder zur Verfügung gestellt werden soll, kann sich der Prozess der Wiederherstellung bei einem mehrere Gigabyte umfassenden Backup durchaus viele Stunden hinziehen.

Mit einem Verfahren, das eine Wiederherstellung einer einzelnen Tabelle oder allgemein eines einzelnen Objekts direkt aus dem Backup ermöglicht, lässt sich ein zeitlicher Ausfall während der Wiederherstellung und auch – bei entsprechender Automatisierung – der Aufwand für beispielsweise das Einbinden der Tabelle in die produktive Datenbank minimieren.

1.2. Übersicht über verfügbare Produkte

Es sind unterschiedliche Produkte am Markt verfügbar, welche ebenfalls auf eine Sicherung und Wiederherstellung von Datenbanken spezialisiert sind. Allen professionellen Produkten ist gemeinsam, dass es sich um proprietäre Software handelt, die kommerziell angeboten wird und meist grundlegend, wenn auch leicht modifiziert, nur die Funktionen von SQL Server selbst zur Verfügung stellt und somit nicht den gewünschten Anforderungen gerecht wird. Nur wenige Anwendungen können ein sogenanntes Object-Level-Recovery bieten, also die Wiederherstellung einzelner Objekte aus einer Sicherung. Zu den wichtigen Softwareprodukten, welche als Ergänzung oder Ersatz der vorhandenen Sicherungsmechanismen von SQL Server dienen sollen, gehören:

Acronis Recovery für MS SQL Server[2]

> Acronis Recovery bietet eine durch Assistenten geführte Erstellung von Backup- und Disaster-Recovery-Strategien, unterstützt eine Vielzahl von Speichermedien und ermöglicht eine assistentengestützte oder automatische Wiederherstellung einer Datenbank bis zum letzten bekannten stabilen Zustand. Acronis ist ebenfalls in der Lage, Backup und Restore auf Grundlage der CPU- und Netzwerkauslastung ressourcenschonend zu erledigen. Eine selektive Wiederherstellung von Objekten ist nicht möglich, daher wird dieses Produkt nicht weiter untersucht.

[2] http://www.acronis.de/enterprise/products/ARSQL/

Quest LiteSpeed for SQL Server[3]

Schwerpunkte dieses Produkts sind die hohe Komprimierung und Performanz bei Backup und Restore. Weiterhin gehören das Auslesen von Transaktionsprotokollen und die Wiederherstellung von einzelnen Datenbankobjekten, wie Tabellen, Prozeduren und Sichten, zu den Stärken dieses Programms. LiteSpeed zählt zu den bekannteren Softwareprodukten, welche eine Wiederherstellung auf Objektbasis zulassen, und wird daher noch genauer untersucht.

Peer DRS[4]

Peer DRS bietet ein Datenbankbackup auf Byte-Level-Ebene, also das Sichern von binären Unterschieden zur Schonung der Ressourcen, und ist auf die Replikation von Datenbanken optimiert. Diese Software bietet keine Möglichkeit eine gesicherte Datenbank wieder herzustellen. Da weder eine Objektsicherung noch generell eine Wiederherstellung eines Backups möglich ist, bleibt es hier nur bei der Nennung dieses Produkts.

idera SQL safe backup[5]

Hierbei handelt es sich um eine durch eine Oberfläche gestützte Software, welche Wert auf Performanz und Sicherheit legt. Dieses Produkt lässt ebenfalls die Funktion eines Object-Level-Recoverys vermissen.

redgate SQL Backup Pro[6]

Schwerpunkte von SQL Backup Pro sind die hohe Kompression und die Möglichkeit, beispielsweise durch Netzwerkfehler unterbrochene Transaktionen an der Stelle fortzusetzen, an der sie unterbrochen wurden. Neu in SQL Backup Pro ist die Unterstützung von Object-Level-Recovery. Die von redgate angebotenen Softwareprodukte gehören zu den bekanntesten und daher wird SQL Backup Pro auch in weiteren Tests untersucht.

---

[3] http://www.questsoftware.de/litespeed-for-sql-server/

[4] http://de.peersoftware.com/products/peer_drs/peerdrs.aspx

[5] http://www.idera.com/Products/SQL-Server/SQL-safe-backup/

[6] http://www.red-gate.com/products/SQL_Backup/index_v2.htm

DataBK.com SQL Server Backup 7.8.6[7]

Dieses Produkt bietet die Standardfunktionen, die auch in den anderen bereits genannten Anwendungen enthalten sind. Besonderheit dieses Produkts ist die Möglichkeit, die produkteigenen Backups in das SQL Server-eigene Format zu konvertieren. Dieses Produkt lässt eine selektive Wiederherstellung vermissen, was auch nicht die Konvertierung in das SQL Backup-Format aufwiegen kann, daher wird diese Software nicht weiter untersucht.

Quest LiteSpeed for SQL Server und redgate SQL Backup Pro sind die wenigen bekannten Produkte, welche die Wiederherstellung von einzelnen Datenbankobjekten ermöglichen, und werden deshalb später in dieser Arbeit einem genaueren Test unterzogen. Nicht zu dieser Liste zählt Standard-Backupsoftware, welche auf die Sicherung eines kompletten Systems ausgelegt ist, wie beispielsweise CA ARCserve Backup[8].

1.3. Ziel dieser Arbeit und Abgrenzung

Wie in Abschnitt 1.1 erwähnt, ist es für den „fiktiven“ Anwendungsfall (der Wiederherstellung eines einzelnen Objekts in einer sehr großen Datenbank) hilfreich, wenn nicht sogar notwendig (falls es entsprechende zeitliche Anforderungen gibt), Objekte einer Datenbank selektiv wiederherzustellen. Die für diesen Zweck angebotenen Softwareprodukte, die unter Abschnitt 1.2 erwähnt wurden, bieten aber keine Transparenz und lassen sich auch nicht direkt oder nur bedingt in eigenen SQL Programmen verwenden.

In dieser Arbeit soll ein Weg gefunden werden, der es ermöglicht, vollständige sowie partielle Backups von Datenbanken des SQL Server zu erstellen und aus diesen einzelne Objekte wiederherzustellen. Dabei sind folgende Probleme zu lösen:

- Auslesen des Datenbankschemas
- Eindeutige Kennzeichnung eines Datensatzes in einer Tabelle
- Bildung von Inkrementen eines Backups
- Verwaltung eines Backups
- Wiederherstellung eines Datenbankobjekts aus dem Backup
- Wahrung der Datenintegrität

[7] http://www.databk.com/sql-server-backup.htm

[8] http://www.ca.com/de/products/product.aspx?id=4536

Auf Grundlage der gewonnenen Erkenntnisse soll ein Skript geschaffen werden, welches ganzheitlich diese Aufgaben übernimmt. Wichtigstes Kriterium soll sein, dass jegliche Datenbewegungen, die durch das Programm vorgenommen werden, keinerlei Veränderungen an der bestehenden und zu sichernden Datenbank bewirken[9]. Schwerpunkte bei dieser Arbeit sind dabei die Logik für die Bildung von Dateninkrementen und die Wahrung der Datenintegrität.

Die Implementierung erfolgt auf niedrigster Applikationsebene über der Datenbank, welches bei SQL Server die Skriptsprache TSQL ist. Durch Umsetzung in dieser Sprache kann das Programm auch dem Wunsch nach Transparenz und Revisionsfähigkeit gerecht werden, da durch offen liegenden Quellcode jederzeit nachvollziehbar ist, was mit den zu verarbeitenden Daten geschieht. Dies kann beispielsweise bei vertraglichen Vereinbarungen mit externen Dienstleistern oder Vorschriften wie den „Grundsätzen ordnungsgemäßer Datenverarbeitung und DV-Revisionen“ [10] notwendig sein. Leider ist diese Transparenz bei externen Tools, welche auf nativem Code basieren, nicht vorhanden. Damit ist der Übergang von einer „Blackbox“-Backupstrategie, die eine proprietäre Software bietet, zu einer „Whitebox“-Backupstrategie möglich.

Das Ergebnis soll eine funktionale Erweiterung für SQL Server sein, welche die Vorteile externer Software, also die selektive Wiederherstellung auf Objektebene, mit den Funktionen von SQL Server, wie der Integritätswahrung, verbindet. Es soll keine eigenständige Anwendung entstehen, die die Aufgaben einer vollwertigen Backupsoftware vollumfänglich übernimmt – die Erkenntnisse dieser Arbeit sollen dazu dienen, vorhandene Strategien zu optimieren und zu ergänzen. Weiterhin soll im Rahmen dieser Arbeit kein Schwerpunkt auf Performance oder die Verifikation einer angefertigten Sicherung beziehungsweise Wiederherstellung (wie bei kommerziellen Programmen üblich) gelegt werden. Abschließend soll eine Aussage über die Anwendbarkeit in der Praxis getroffen werden.

---

[9] Ausnahme kann hier bei der Wiederherstellung eines einzelnen Objekts die Manipulation dieses Objekts zur Wahrung der Konsistenz sein

[10] Die Grundsätze ordnungsgemäßer Datenverarbeitung und DV-Revisionen (Abk. „GoDV“) sind Standardrichtlinien, welche sich aus geltendem Recht, insbesondere den Vorschriften aus dem Handelsgesetzbuch über ordnungsgemäße Buchführung herleiten; „Grundsätze für eine ordnungsmäßige Datenverarbeitung (GoDV). Handbuch der DV-Revision (Gebundene Ausgabe)“ von Rainer Schuppenhauer, Idw-Verlag GmbH, 5. Auflage, ISBN 3802107527

1.4. Vorgehensweise

Zur Einführung in die Thematik soll auf grundlegende Sicherungsarten zum Thema Backup eingegangen und anschließend die Möglichkeiten von SQL Server in Bezug auf Datensicherung aufgezeigt werden. SQL Server bietet einen reichhaltigen Katalog an Optionen zur Sicherung und Wiederherstellung von Datenbanken mit unterschiedlichsten Szenarien. Dies überschreitet den Rahmen dieser Arbeit und ist zudem meist nur für spezielle Fälle, wie hochverfügbare oder auf mehrere Speichermedien verteilte Datenbanken, geeignet. Daher soll hier nur auf die wichtigsten Eigenschaften der integrierten Backupmöglichkeiten eingegangen werden.

Der Kern dieser Arbeit beginnt mit einer Abgrenzung des, im Rahmen dieser Arbeit zu erstellenden Programms, es sollen dessen Leistungsumfang und Anforderungen festgehalten werden. Ein wichtiger Aspekt bei der Arbeit mit dem fertigen Programm ist in jedem Fall die Wahrung der Datenintegrität, hierzu sollen umfangreiche Überlegungen stattfinden. Zu lösende Probleme sind unter anderem das Auslesen des Datenbankschemas und die eindeutige Kennzeichnung von Zeilen einer Tabelle. Auch hierzu sollen ebenfalls Vorüberlegungen stattfinden.

Anschließend soll schrittweise, unterstützt durch verschiedene Diagramme und Pläne, eine Funktionalität des Programms entworfen und dabei auch der technische Hintergrund beleuchtet werden. Nach dem Entwurf werden einige Funktionstests an einer Test-Datenbank durchgeführt.

Nach der Entwicklung wird das gewonnene Produkt den hauseigenen Mitteln von SQL Server und einigen kommerziellen Produkten gegenübergestellt, um so Effizienz und Handhabung miteinander vergleichen zu können. Abschließend soll eine Aussage über den Einsatz in produktiven Umgebungen getroffen, sowie ein Ausblick auf weitere Entwicklungen gegeben werden.

2. Grundlagen und –begriffe

Unter dem Begriff Backup verstehen viele Anwender, eine Datei von A nach B zu kopieren. Dabei umfasst dieser Begriff weitaus mehr, wenn auch nebenläufige Dinge, die meist durch eine Software für den Nutzer erledigt werden. In diesem Kapitel sollen wichtige Grundbegriffe und Arbeitstechniken in Bezug auf Backups erläutert werden.

2.1. Backup und Restore

Der Begriff „Backup“[11] umfasst allgemein einen Sicherungsvorgang. Es ist hierbei irrelevant, um welche Art von Daten es sich handelt, in welchem Umfang und wohin sie gesichert werden. Üblicherweise werden hierbei aber (unternehmens-) kritische Daten oder auch für einen Endanwender wichtige Dokumente, meist vollumfänglich, auf ein externes Speichermedium, wie eine DVD, oder auf Magnetband gesichert. Die Wahl des Sicherungsmediums sollte an die spezifischen Bedürfnisse angepasst sein. Hier können folgende Kennzahlen wichtig sein [Inf05]:

- Umfang des zu sichernden Datenvolumens
- Zugriffszeiten auf dem Medium
- Aufbewahrungsfristen (zulässige Lagerungszeiten für ein Medium)
- Revisionssicherheit

Ziel eines Backups ist in der Regel für den Fall eines Ausfalls oder Totalverlustes diese wichtigen Daten nochmals zur Verfügung zu haben, um diese gegebenenfalls auch auf einem anderen Rechnersystem wiederherstellen zu können. Eine Sicherung auf demselben Medium, auf dem sich auch die zu sichernden Daten befinden, ist in dem Sinne kein Backup, da ein Backup zum Grundsatz hat, Daten gegen Verlust zu schützen. Dem widerspräche beispielsweise eine Sicherung auf einem Produktivsystem, bei dem die Festplatte ausfällt – alle Daten wären verloren.

Eine Sicherung soll vor Datenverlust bei:

- Hardwaredefekten (mechanische Beschädigung des Rechnersystems, insbesondere der Festplatte)
- Diebstahl der Daten (egal ob lokal am System oder online über das Internet)

[11] http://www.bullhost.de/b/backup.html

- versehentlichem oder absichtlichem Ändern und Löschen der Daten
- Beschädigung oder Verlust durch Viren, Würmer oder Trojaner

schützen [Inf09]. Daher ist es ratsam, eine Sicherung auf einem transportablen Medium anzufertigen und örtlich getrennt vom Produktivsystem, idealerweise auch durch Schließfach oder Tresor gegen Zugriff geschützt aufzubewahren [Inf06] [Inf061].

Restore (auch Recovery genannt) stellt allgemein den Vorgang der Datenwiederherstellung dar. Es existiert keine genaue Definition, die einen Unterschied zwischen Restore und Recovery herausstellt. Im Bereich der Systemadministration wird teilweise von einem Restore gesprochen, wenn es um die Wiederherstellung einzelner Daten oder Dateien geht, Recovery hingegen bezeichnet die Wiederherstellung eines vollständigen PC-Systems [Adm08]. In Bezug auf SQL Server gibt es lediglich eine kleine Unterscheidung: *„You have a backup file from that you can create the database using restore command you can also overwrite an existing database. Recovery is a different process, when you start Sql Server it will recover the database i.e it will roll forward all the committed transactions and roll back all the uncommitted transactions.“* [Mic08] - Restore wird hier für die Wiederherstellung einer Datenbank aus einem Backup verwendet und Recovery für den Vorgang, den SQL Server beim Start auf seinen Datenbanken ausführt, um diese beispielsweise bei einem Absturz wieder in einen konsistenten Zustand zu bringen. Im Rahmen dieser Arbeit wird weiterhin von Restore gesprochen.

2.2. Historisierung und Versionierung

Es handelt sich hierbei um zwei Begriffe, die nah beieinander liegen und oft gleichgesetzt werden. Beide Begriffe werden in dieser Arbeit nur ein nebensächlich von Bedeutung sein, werden aber oft in Zusammenhang mit Backups und Sicherungen erwähnt, daher sollen sie hier trotzdem kurz angesprochen werden. Grundsätzlich lassen sich diese wie folgt definieren:

Versionierung

Grundgedanke der Versionierung ist die Dokumentation von Änderungen an einem Dokument. Eine Änderung an einem Dokument bewirkt die Erzeugung einer neuen Version davon. Hierbei werden meist Zeitstempel und Personendaten des ausführenden Nutzers gespeichert. Ein Versionsverwaltungssystem übernimmt hierbei das Management.

Historisierung

Historisierung bedeutet, inhaltliche Änderungen an Daten zu dokumentieren. Es wird hierbei die zeitliche Entwicklung dieser Daten festgehalten. Bezogen auf Datenbanken, gibt es für die Historisierung verschiedene Möglichkeiten Daten zu historisieren. Zwei wichtige Methoden sind:

- Erweiterung der Tabelle um eine zusätzliche Spalte, welche den neuen Wert enthält
- Einführung eines Zeitstempels, der die Gültigkeit einer Datenzeile bestimmt

Die Definition von Historisierung lässt sich insofern auf diese Arbeit übertragen, als dass ein Backup einen historisierten Datenbestand widerspiegelt. Es werden hier also keine Änderungen in den Daten an sich berücksichtigt, sondern eine Sicherung ist eine historische Sicht auf die Daten zum Zeitpunkt der Sicherung.

2.3. Archivierung

Die Archivierung ist begrifflich von einer Datensicherung zu trennen. Eine Sicherung hat zum Zweck, eine Kopie von Daten physikalisch vom System getrennt und zugriffsgeschützt zu erstellen. Eine Archivierung hingegen ist direkt in die Systeme integriert und im laufenden Betrieb eingebunden. Durch eine Archivierung sollen Daten dauerhaft in geeigneter Form gespeichert werden, um sie jederzeit einfach wiederzufinden und sofort darauf zugreifen zu können [Inf091]. Die Archivierung findet sehr oft in Dokumentenmanagement-Systemen Anwendung. Dabei gibt es meist eine Indexdatenbank, welche die Metadaten, also beispielsweise Beschreibung und Speicherort, verwaltet, und einen physikalischen Speicher der das eigentliche Dokument, beziehungsweise die Daten, enthält.

In dieser Arbeit kann insofern von einem Archivsystem gesprochen werden, als dass eine Metadatenbank alle nötigen Informationen über die erstellten Backups enthält und die physikalisch erstellten Sicherungen logisch verwaltet.

2.4. Vollständige Sicherung

Eine vollständige Sicherung bedeutet, dass bei jedem Sicherungsvorgang eine vollumfängliche Kopie der zu sichernden Daten angefertigt wird [Bau06 S. 281f], unabhängig davon, ob es sich um Dateien, Verzeichnisse, ein komplettes Laufwerk, oder in diesem Fall, den kompletten Datenbestand einer Datenbank handelt. Um einen Datenstand

wiederherzustellen, muss lediglich die letzte Sicherung herangezogen werden. Als Vorteil ist hier zu sehen, dass in jeder vorhandenen Sicherung der komplette (und somit auch konsistente) Datenbestand zu finden ist und bei einem Defekt einer Sicherung im schlimmsten Fall die letzte vorhergehende Sicherung verwendet werden muss. Sicher wird hier schnell der Gedanke aufkommen, dass eine vollständige Sicherung auch den meisten Speicherplatz verbraucht. Insofern es also der Geschäftsprozess und auch die Anforderungen an die Sicherheit und Verfügbarkeit einer Datenbank zulassen, gilt es, nach kostengünstigeren Lösungen zu suchen.

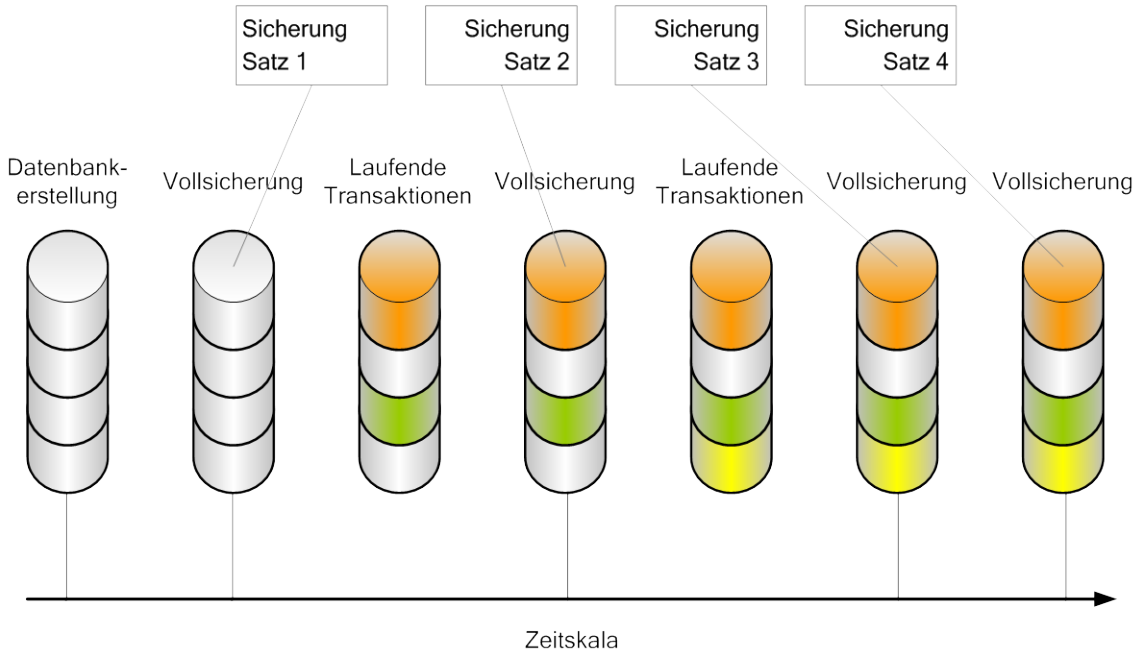

Abbildung 1: Vollständige Sicherung

2.5. Differenzielle Sicherung

Eine differenzielle Sicherung ist der ausgewogenste Ansatz zwischen Sicherheit der erstellten Backups und Speicherplatzverbrauch. Grundlage für diese Sicherungsart ist eine vollständige Sicherung der Datenbank (oder auch Dateien, Verzeichnisse etc.), da im weiteren Verlauf nur Änderungen, die seit der letzten vollständigen Sicherung vorgenommen wurden, dokumentiert werden [Bau06 S. 283]. Somit ist zur Wiederherstellung einer Datenbank ein weiterer Schritt nötig.

Zuerst muss die vorhandene vollständige Sicherung wieder eingespielt werden, welche als Basis für die differenziellen Sicherungen dient und anschließend die gewünschte (letzte) differenzielle Sicherung [Woo07 S. 145]. Es lässt sich erkennen, dass aufgrund der

dokumentierten Änderungen zur letzten Vollsicherung bedeutend weniger Speicherplatz verbraucht wird, als bei jedem Sicherungsvorgang die komplette Datenbank zu sichern. Ein angenehmer Nebeneffekt ist hier, dass der Sicherungsvorgang an sich auch schneller ist. Wenn man bei Dateien beispielsweise das letzte Änderungsdatum zur Sicherung heranzieht und dieses mit dem Sicherungsdatum der letzten Vollsicherung vergleicht, kann man die zu sichernde Datenmenge recht einfach reduzieren. Die Sicherheit der Daten ist hierbei aber nicht mehr in dem Maße wie bei einer Vollsicherung gegeben. Bei einem Defekt der Datei, welche die Vollsicherung enthält, ist dieses Backup nicht mehr verwendbar und man muss auf die davorliegende vollständige Sicherung zurückgreifen, weil durch den Defekt der Vollsicherung alle nachfolgenden differenziellen Sicherungen unbrauchbar geworden sind. Hier ist es also auch wichtig, eine vernünftige Strategie zu finden, die dieses Risiko minimiert.

Bei Datenbanken im Bereich mehrerer Gigabyte und einer geplanten täglichen Sicherung, ist es sicher ungünstig, einmal monatlich eine Vollsicherung und für den Rest des Monats differenzielle Backups durchzuführen. Eine wöchentliche Vollsicherung ist hier geeigneter, zumal eine differenzielle Sicherung immer größer ist als die letzte, da hier ja alle Änderungen im Vergleich zur letzten Vollsicherung dokumentiert und somit auch die Daten der letzten differenziellen Sicherung enthalten sind. Auch aus diesem Grund ist es sinnvoll, in regelmäßigen Abständen neue Vollsicherungen, die als Basis für folgende differenzielle Sicherungen dienen, anzufertigen.

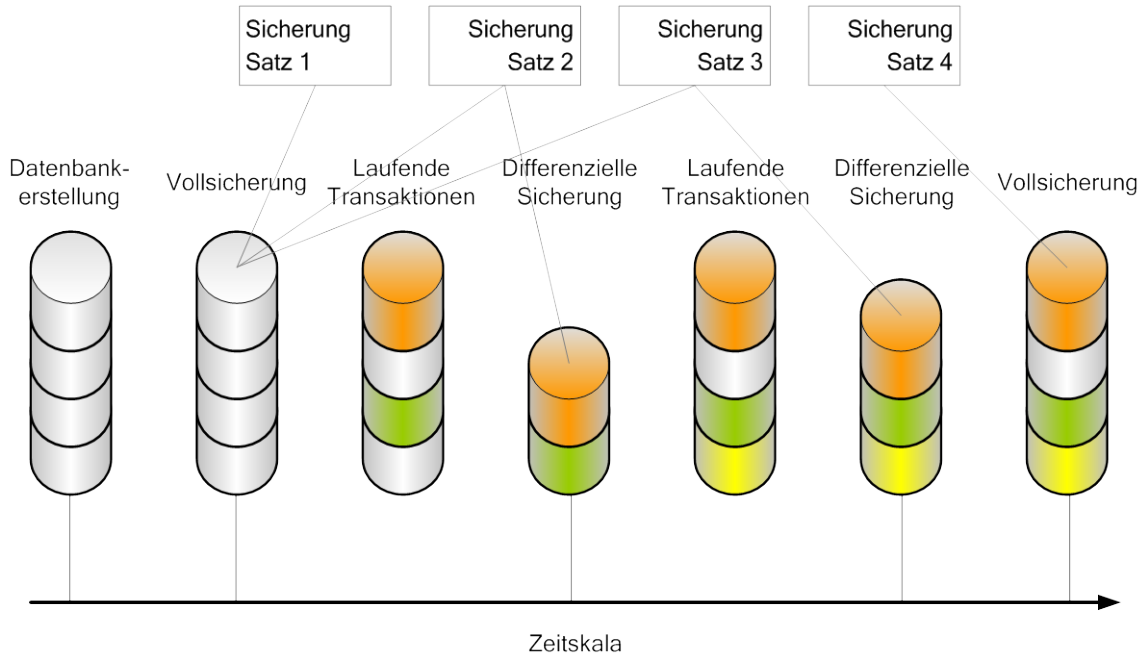

Abbildung 2: Differenzielle Sicherung

2.6. Inkrementelle Sicherung

Die platzsparendste Methode ist die inkrementelle Sicherung. Diese arbeitet prinzipiell wie eine differenzielle Sicherung, dokumentiert aber im Gegensatz zu der differenziellen Sicherung nur die Unterschiede zur letzten inkrementellen Sicherung. Eine Sicherung baut also auf den Datenstand der letzten Sicherung auf. Somit muss ein Wiederherstellungsvorgang wie folgt verlaufen: zuerst wird die letzte vollständige Sicherung eingespielt und darauf aufbauend in chronologischer Reihenfolge alle (und damit sind auch alle gemeint) angefertigten inkrementellen Sicherungen. Ist nur eine dieser Sicherungen defekt oder wird eine davon beim Wiederherstellen vergessen, so ist der Datenstand unbrauchbar. Von SQL Server wird diese Variante einer Sicherung nicht angeboten, soll aber prinzipiell in dieser Arbeit berücksichtigt werden.

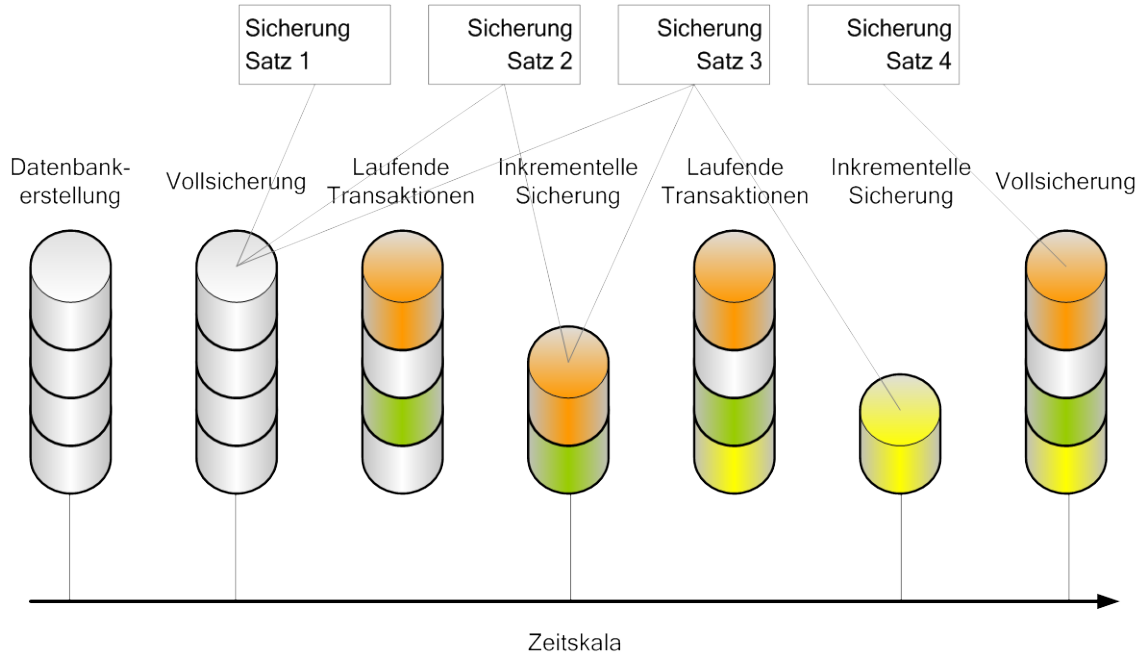

Abbildung 3: Inkrementelle Sicherung

2.7. Vater-Sohn-Prinzip

Das Vater-Sohn-Prinzip stellt keine Sicherungsart, sondern lediglich eine Strategie zur Aufbewahrung der Datensicherungen dar (Archivierung). Häufig wird es auch als Generationenprinzip bezeichnet. Hierbei werden verschiedene Sicherungszyklen festgelegt, wie beispielsweise: täglich, wöchentlich, monatlich – in diesem Fall spricht man von Großvater-Vater-Sohn-Prinzip [Wik10]. Diese Zyklen rotieren dann regelmäßig.

Um es an einem kurzen Beispiel zu verdeutlichen:

- Die tägliche Sicherung von Montag bis Samstag wird als Sohn (**S**) bezeichnet
- Die wöchentliche Sicherung am Sonntag wird als Vater (**V**) bezeichnet
- Und die Großväter (**G**) stellen die Sicherung am letzten Tag im Monat dar

Für die Söhne wird in diesem Fall kein Sonntags-Medium benötigt, weil diese Sicherung mit den Vätern zusammenfällt – da sonntags die wöchentliche Sicherung stattfindet, kann eine zusätzliche tägliche Sicherung entfallen. Zyklisch werden die Sicherungsmedien wie folgt überschrieben: am Montag der folgenden Woche wird das erste Medium der aktuellen Woche überschrieben. Nach dem fifo-Prinzip (first in first out) werden also immer die ältesten Sicherungen mit der aktuellen Sicherung überschrieben.

Dieses Generationenprinzip kann auf alle 3 Sicherungsarten angewendet werden.

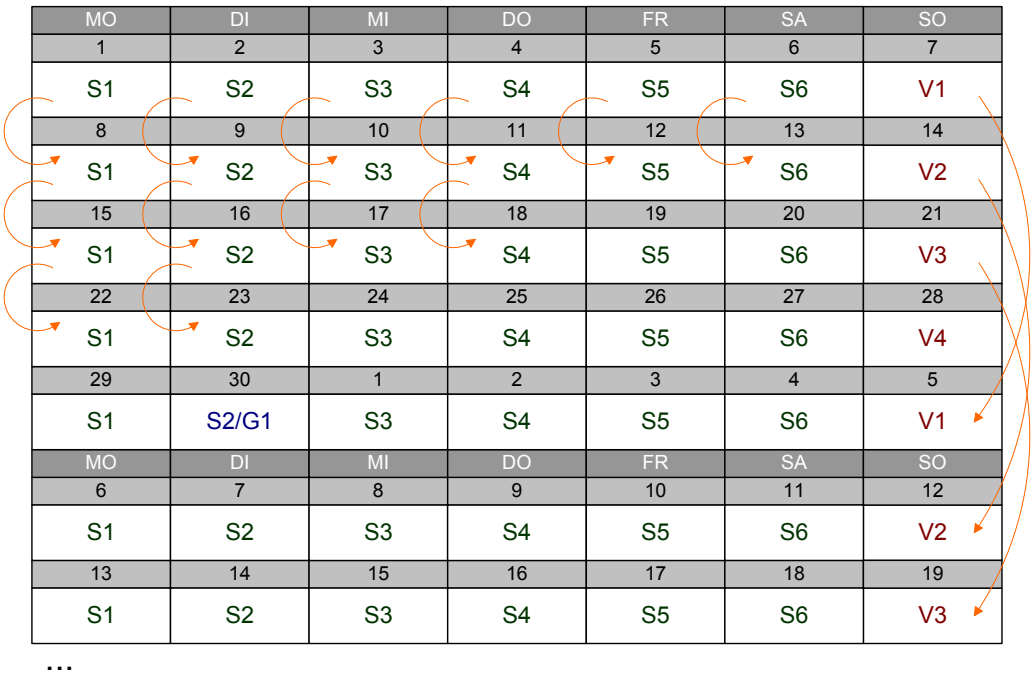

Abbildung 4: Vater-Sohn-Prinzip

3. Standard-Sicherungsmechanismen unter SQL Server

3.1. Wiederherstellungsmodelle

Das Wiederherstellungsmodell bei SQL Server hat wesentlichen Einfluss auf die Sicherungs- und Wiederherstellungsstrategien einer Datenbank. Es muss bereits beim Anlegen einer Datenbank ausgewählt werden, kann aber nachträglich noch geändert werden [Bau06 S. 172]. Das Wiederherstellungsmodell bestimmt die Verwendung der Protokolldatei einer Datenbank und somit, wann eine Transaktion aus dem Transaktionsprotokoll gelöscht wird [Woo07 S. 130f]. Wenn eine Transaktion begonnen wird, werden alle Anweisungen dieser Transaktion fortlaufend in das Protokoll geschrieben und erst bei Abschluss der Transaktion die daraus resultierenden Änderungen in der Datenbank vorgenommen.

3.1.1. Einfach

Das einfache Wiederherstellungsmodell hat zur Folge, dass alle Einträge zu einer Transaktion nach Übernahme in die Datenbank aus dem Transaktionsprotokoll gelöscht werden. Während der Transaktion vergrößert sich also das Protokoll und verkleinert sich nach Abschluss automatisch wieder [Woo07 S. 131]. Vorteile dieses Modells sind, dass es schnell und platzsparend ist und den Verwaltungsaufwand minimiert (es muss keine Protokollsicherung vorgenommen werden). Es ist aber ebenso das Unsicherste [Mic086].

Konkret bedeutet dies, dass Daten im Falle eines Crashs oder Ausfalls verloren gehen können. Vorhandene Sicherungen enthalten ja lediglich den Datenstand zum Zeitpunkt der Sicherung. Alle danach vorgenommen Änderungen an der Datenbank sind somit verloren, weil diese an keiner weiteren Stelle gespeichert waren. Das einfache Wiederherstellungsmodell bietet sich daher bei selten verändernden oder schreibgeschützten Datenbanken an. Dieser Nachteil kann zwar durch recht kurze Zeitabstände zwischen den Sicherungen gemindert, aber niemals komplett eliminiert werden [Mic087].

3.1.2. Vollständig

Die Nachteile des einfachen Wiederherstellungsmodells gelten beim vollständigen Modell nicht im gleichen Maße. Ist dieses Modell aktiviert, verbleiben Transaktionen auch nach der Übernahme in die Datenbank im Transaktionsprotokoll. Diese Einträge werden erst aus dem Protokoll entfernt, wenn die Datenbank oder die Protokolldatei gesichert wird [Woo07 S.

132]. Letztlich bedeutet dies also einen erhöhten Wartungsaufwand, weil so das Protokoll relativ schnell anwächst, aber auch einen höheren Schutz.

In einem Sicherungsszenario müssen also neben (vollständigen) Sicherungen der Datenbank auch verhältnismäßig kleine Sicherungen des Protokolls vorgenommen werden, dies kann mehrmals täglich bis stündlich (oder gar noch häufiger) passieren – je nach Anforderung an die Sicherheit. Der Wiederherstellungsvorgang gestaltet sich nun wie folgt: zuerst wird die letzte Vollsicherung eingespielt und anschließend alle nachfolgenden Protokollsicherungen. Somit kann eine Datenbank bis zum Zeitpunkt des Auftretens eines Fehlers wiederhergestellt werden. Durch die Protokollsicherung wird es aber auch möglich, die Datenbank zu einem bestimmten Zeitpunkt innerhalb der Sicherung, also nicht unbedingt bis zur letzten Transaktion, wiederherzustellen [Mic088]. Auch hier kann es natürlich passieren, dass bei einem Ausfall alle Daten seit der letzten Protokollsicherung verloren sind. Hier hilft letztlich nur eine Hochverfügbarkeitslösung, um Datenverlust vollständig auszuschließen.

3.1.3. Massenprotokolliert

Das massenprotokollierte Wiederherstellungsmodell ähnelt dem vollständigen Modell. Es bringt aber einen niedrigeren Schutz bei Ausfällen mit sich. Das heißt, dass beispielsweise das Einfügen von großen Datenmengen über die Befehle `BULK INSERT` oder `INSERT … SELECT * FROM OPENROWSET(BULK …)` oder andere Massenvorgänge wie `SELECT INTO` oder `CREATE INDEX` nur minimal protokolliert werden. Damit steigt natürlich das Risiko von Datenverlust bei einem Fehler.

Das massenprotokollierte Wiederherstellungsmodell sollte nicht allein eingesetzt werden, sondern bei Verwendung des vollständigen Wiederherstellungsmodells kann im Bedarfsfall vor größeren Einfügungsvorgängen auf das massenprotokollierte Modell umgeschaltet, und nach Abschluss der Einfügeoperationen wieder zurück gewechselt werden [Woo07 S. 133f]. Somit wird die Leistung bei Massenvorgängen gesteigert und die Speicherplatzbelegung verringert.

3.2. Sicherungsmedien

Unter SQL Server gibt es zwei Arten von Sicherungsmedien:

- physikalisch
- logisch

Ein physikalisches Sicherungsmedium kann ein Bandlaufwerk, eine USB-Festplatte oder ein sonstiger Datenträger sein, der durch das Betriebssystem bereitgestellt wird [Woo07 S. 140f]. Pfade können hierbei als absolute lokale Pfade, oder im Falle von Netzwerkfreigaben als UNC-Pfade angegeben werden. Prinzipiell sollte für Sicherungen eine andere Festplatte verwendet werden als die, auf denen Datenbank- oder Protokolldateien liegen.

Bei logischen Sicherungsmedien werden logische Namen für physikalisch vorhandene Medien definiert [Bau06 S. 285]. Diese Namen können dann in der grafischen Oberfläche von SQL Server oder bei direkten SQL-Anweisungen benutzt werden und bieten gerade bei Skripten mehr Flexibilität – muss also ein Medium getauscht werden, beispielsweise soll ein Wechsel von lokaler Festplatte auf Netzwerkmedium vorgenommen werden, muss lediglich die Verknüpfung hinter dem logischen Namen geändert werden. Bereits vorhandene Sicherungsaufträge und –programme müssen dann nicht erst noch angepasst werden [Woo07 S. 141].

3.3. Systemdatenbanken

Zu den sicherungsbedürftigen Datenbanken von SQL Server gehören

- `master`
- `model`
- `msdb`

Diese Datenbanken sind Grundvoraussetzung für den Betrieb von SQL Server und werden bei der Installation des SQL Servers automatisch angelegt. Spielt Replikation beim Betrieb des SQL Servers eine Rolle, muss zusätzlich die Datenbank `distribution` gesichert werden [Bau06 S. 278].

Die `master`-Datenbank enthält wichtige Informationen wie Anmeldekonten, Systemeinstellungen und Informationen für den Zugriff auf andere Datenbanken. Die Datenbank `model` dient als Vorlage für das Erstellen neuer Datenbanken, unter anderem auch für die `tempdb`, welche der Arbeitsbereich für SQL Server ist (hier werden temporäre Tabellen und Ergebnisse zwischengespeichert). Als dritte Datenbank im Bunde enthält die `msdb` Daten des SQL Server-Agents sowie zugehöriger Zeitplanungsinformationen zum

Ausführen von Aufgaben und den vollständigen Sicherungs- und Wiederherstellungsverlauf [Bau06 S. 113, 277f]. Eine Sicherung der `msdb`-Datenbank sollte an den jeweiligen Geschäftsprozess angepasst sein, mindestens aber nach jeder Datenbankänderung durchgeführt werden. Alle drei Datenbanken können mit denselben Mitteln wie für normale Datenbanken gesichert werden. Lediglich für die `master` kann nur eine vollständige Datenbanksicherung durchgeführt werden [Bau06 S. 282].

3.4. Backup-Funktionsweise

Ein Backup mit Hilfe der in SQL Server integrierten Funktionen ist auf unterschiedliche Weise möglich:

- durch Drittanbietertools, welche eine API[12] nutzen [Mic081]
- TSQL-Befehl auf der SQL-Befehlszeile
- Sicherung über die grafische Oberfläche von SQL Server, dem SQL Server Management Studio [Kon07 S. 50ff] [Bau06 S. 79ff]
- manuelle Sicherung

Die sicher allgemeinste Variante ist die Sicherung einer normalen, nicht verteilten oder replizierten Datenbank mit nur einer Dateigruppe auf ein Sicherungsmedium, daher soll dieses Szenario hier detaillierter erläutert werden.

Ein Backup lässt sich in TSQL mit `BACKUP DATABASE {dbname} TO <medium>;` auf ein logisches Sicherungsmedium durchführen [Bau06 S. 289]. Dies ist ausreichend, um eine vollständige Sicherung einer Datenbank zu erhalten. Alternativ kann mit `BACKUP DATABASE {dbname} TO DISK='path';` auch eine Sicherung auf einen physikalischen Pfad erreicht werden. Wird das vollständige Wiederherstellungsmodell verwendet, so muss entsprechend das Transaktionsprotokoll gesichert werden. Dies kann mit `BACKUP LOG {dbname} TO <medium>;` erfolgen.

[12] Die API stellt die Funktionen zur Verfügung, wie sie auch in den integrierten Möglichkeiten durch SQL Server selbst angeboten werden

Diese Funktionalität wird ebenfalls durch das SQL Server Management Studio in Form einer grafischen Oberfläche bereit gestellt [Woo07 S. 149ff].

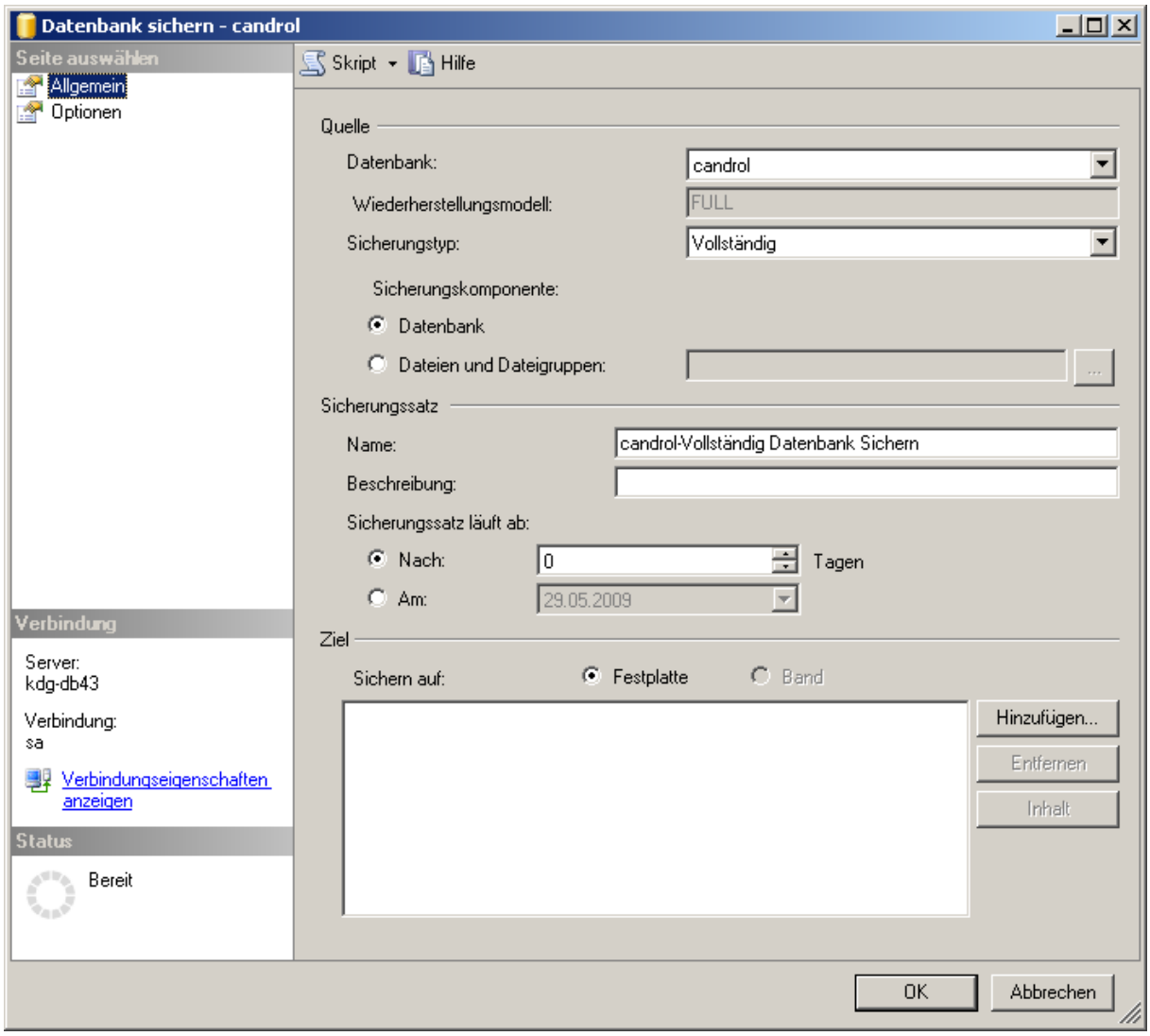

Abbildung 5: Sicherungsdialog von SQL Server

Im Menüpunkt „Optionen“ werden alle Einstellungen erreicht, die ebenso in TSQL in Verbindung mit `WITH` angegeben werden können. Über diesen Dialog werden auch Sicherungen des Transaktionsprotokolls beim vollständigen oder massenprotokollierten Wiederherstellungsmodell vorgenommen.

Letztlich besteht noch die Möglichkeit einer manuellen Sicherung. Dazu kann eine Datenbank vom Server „getrennt“ werden [Woo07 S. 143]. Es werden also alle Transaktionen beendet und die Verbindung geschlossen. Die hinter einer Datenbank liegenden Dateien lassen sich nach dem Trennen wie übliche Dateien handhaben und können so an einen beliebigen Ort

zum Sichern kopiert werden. Je nach Datenbankaktivität sollte ein solches Trennen aber nur unter großer Vorsicht erfolgen, da hiermit die Datenbank für Benutzer während der Sicherung nicht mehr erreichbar ist.

Alle Varianten, bei denen SQL Server ein Backup auf Basis der integrierten Funktionen durchführt, ist zu eigen, dass ein Backup aus einem Schnappschuss erstellt wird, der zu Beginn des Backups von der Datenbank erstellt wird. Damit wird ein konsistenter Bestand der Daten gesichert. Die Datenbank ist aber weiterhin für andere Nutzer erreichbar. Transaktionen, die nun durch andere Benutzer durchgeführt werden nachdem ein Sicherungsvorgang gestartet wurde, beeinflussen das Backup aber nicht, sondern werden unabhängig vom gewählten Wiederherstellungsmodell im Transaktionsprotokoll zwischengespeichert. Ist die Sicherung der eigentlichen Datenbank beendet, wendet SQL Server alle nach Beginn der Sicherung ausgeführten und abgeschlossenen Transaktionen nacheinander auf das erstellte Backup an und bringt somit bei Abschluss das Backup auf den aktuellen Datenstand, wie er auch in der Datenbank vorhanden ist [Woo07 S. 136] .

3.5. Restore-Funktionsweise

Interne Arbeitsweise von SQL Server:

Die Wiederherstellung einer Datenbank unterteilt sich prinzipiell in 3 Phasen [Mic0810]:

1. Kopieren der Daten – diese Phase umfasst das Kopieren aller Daten (auch Protokolldaten) aus den Sicherungsdateien in die Datenbankdateien
2. Rollforwardphase – hierbei werden die abgeschlossenen, protokollierten Transaktionen (beispielsweise bei Wiederherstellung einer Protokollsicherung) auf die soeben kopierten Daten angewendet
3. Rollbackphase – in dieser Phase werden Transaktionen, welche nicht durch ein Commit abgeschlossen wurden, rückwärts wieder aufgehoben

In der ersten Phase werden alle zu einer Sicherung gehörigen Daten in die Datenbank kopiert. Dies umfasst mindestens eine vollständige Sicherung und eventuell erstellte differenzielle Sicherungen. Die Rollforwardphase dient dazu, noch vorhandene Transaktionen in Protokollsicherungen auf die soeben hergestellten Daten anzuwenden. Bei dem vollständigen Wiederherstellungsmodell kann dabei der Zeitpunkt in Form einer Datumsangabe oder einer bestimmten Transaktionsnummer angegeben werden, bis zu dem die Wiederherstellung durchgeführt werden soll. Begonnen wird mit dem ersten vorhandenen Protokoll, welches in

der Regel in der Datensicherung selbst enthalten ist. Diese Transaktionen werden schrittweise und chronologisch, so wie sie auch vom Nutzer ausgeführt wurden, auf den Daten abgearbeitet. Ziel der Wiederherstellung ist, einen konsistenten Datenstand in der Datenbank zum Zeitpunkt der Wiederherstellung zu erhalten. Durch das Rollforward können in der Wiederherstellung aus den Protokollsicherungen Transaktionen enthalten sein, für die noch kein Commit zum Zeitpunkt der Wiederherstellung durchgeführt wurde – die also noch nicht abgeschlossen sind. Um einen konsistenten Datenstand zu erreichen, wird für diese offenen Transaktionen ein Rollback durchgeführt. Dabei werden diese Transaktionen, insofern sie nicht durch andere Transaktionen gesperrt sind, rückgängig gemacht. Sind die Daten konsistent, werden sie online[13] gestellt.

Durchführung einer Wiederherstellung durch den Nutzer:

Die Wiederherstellung eines Backups unter SQL Server gestaltet sich recht einfach. Auch hier stehen die Möglichkeiten der Bedienung über das SQL Server Management Studio oder die skriptbasierte Wiederherstellung in TSQL zur Verfügung [Bau06 S. 298ff].

[13] Hier: für Benutzer zur Verfügung gestellt, in dem Verbindungen zur Datenbank zugelassen werden

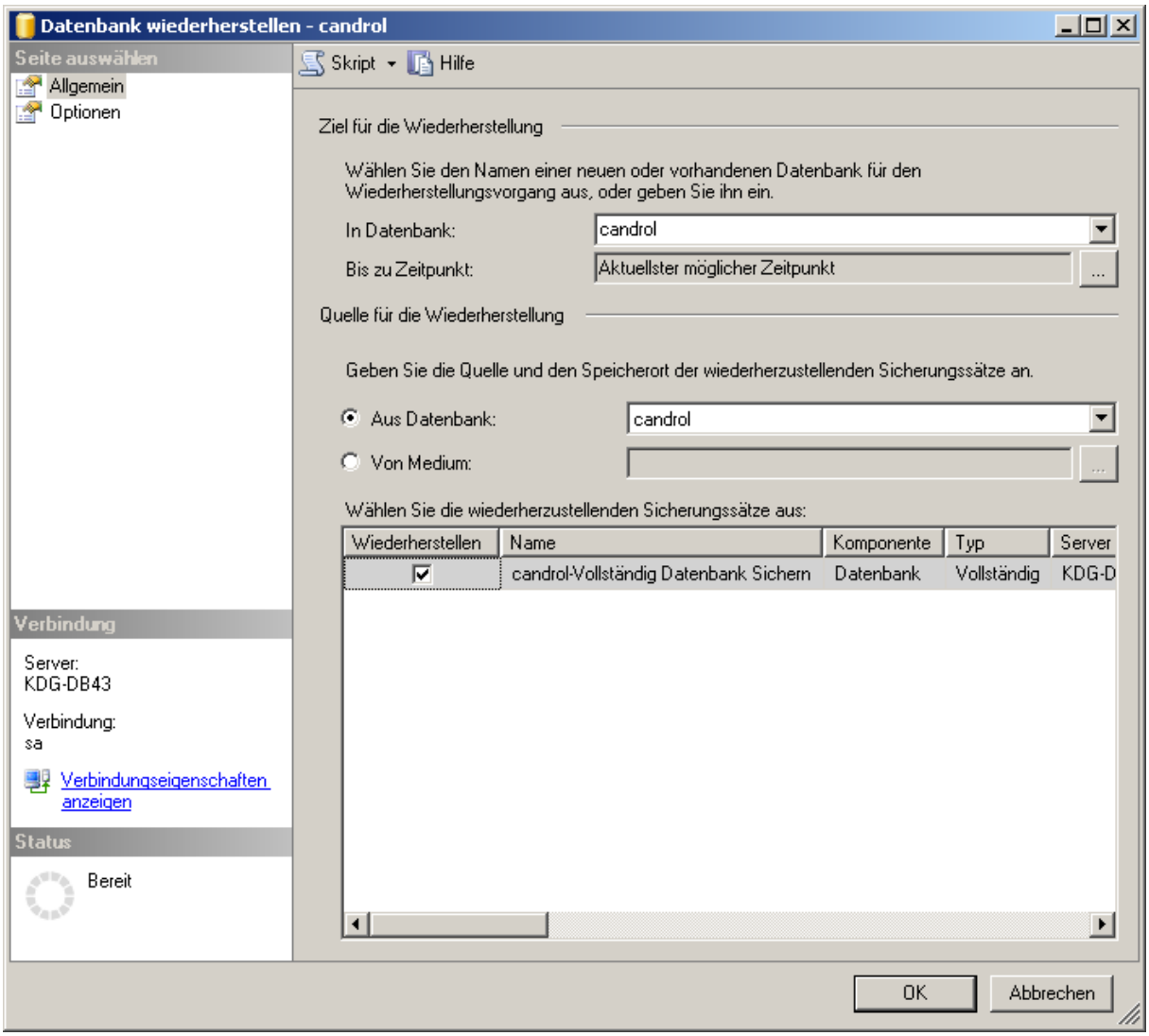

Abbildung 6: Wiederherstellungsdialog von SQL Server

Unter dem Punkt „Optionen“ finden sich wieder all jene Funktionen, welche auch in TSQL mit den `WITH`-Argumenten erreicht werden können. In TSQL wird ein Restore ähnlich einfach wie ein Backup durchgeführt: `RESTORE DATABASE { dbname } FROM <medium>;` Mit Ersetzung von `<medium>` durch `DISK='path'` kann entsprechend ein physischer Pfadname anstatt eines Backupmediums verwendet werden. Soll statt einer Datenbank nur ein Transaktionsprotokoll wiederhergestellt werden, um beispielsweise nach einem differenziellen Restore die verbleibenden Protokolle einzuspielen, kann dies mit `RESTORE LOG { dbname } FROM <medium>;` erfolgen.

4. Untersuchung zu Integritätsfaktoren

4.1. Übersicht

SQL Server selbst garantiert bei der Wiederherstellung eines Backups aus seinem proprietären Format die Datenintegrität durch die Rollforward- und Rollbackphase, welche sich an das Kopieren der eigentlichen Datendateien aus dem Backup in die Datenbank anschließt. Transaktionen, für die noch kein Commit zum Zeitpunkt der Sicherung durchgeführt wurde, werden bei einem Restore so weit durch ein Rollback rückgängig gemacht, bis ein konsistenter Zustand der Daten erreicht wird [Mic0810].

Nachfolgende Kapitel werden fast ausschließlich für die Wiederherstellung einzelner Tabellen aus einer Sicherung von Bedeutung sein, da vollständige Sicherungen und Wiederherstellung in der Regel auf konsistenten Datenbestanden agieren und somit eine Konsistenz der Daten vom System her bereits gewahrt ist.

Es gilt verschiedene Objekte bei einem Object-Level-Recovery zu beachten, die Einfluss auf die Integrität haben:

- Constraints (als Spezialfall hier die CHECK-Einschränkung und Schlüsselprüfung)
- Trigger
- Views
- Referenzielle Integrität (als Sonderfall der Constraints)

4.2. Constraints

Constraints (Einschränkungen) können auf Spaltenebene oder Tabellenebene vergeben sein. Es handelt sich hierbei um einen Oberbegriff für unterschiedliche Regeln. Ein Sonderfall, die referenzielle Integrität, zu der Primär- und Fremdschlüssel gehören, wird in Abschnitt 4.6 behandelt.

Nachfolgende Tabelle gibt einen Überblick, welche Einschränkungen es gibt und welchen Gültigkeitsbereich sie haben [Fae07 S. 211ff]:

Constraint	**Gültigkeitsbereich**
[NOT] NULL	Spalte
UNIQUE	Spalte/Tabelle
CHECK	Spalte/Tabelle
PRIMARY KEY	Spalte/Tabelle
FOREIGN KEY	Spalte/Tabelle

Tabelle 1: Constraint-Typen

NULL-Einschränkungen definieren auf Spaltenebene, dass auch NULL-Werte eingefügt werden dürfen. NOT NULL negiert dies. Weiterhin gehören zu den Constraints Schlüsseleinschränkungen (PRIMARY KEY und FOREIGN KEY), sowie UNIQUE-Schlüssel. Die Schlüsseleinschränkungen definieren allgemein eine oder mehrere Spalten einer Tabelle, die nur eindeutige Werte enthalten [War03 S. 191f]. Eine weitere Einschränkung sind die Gültigkeitsprüfungen (auch CHECK-Einschränkung), welche den Wertebereich einer Spalte definieren. So wird geprüft, ob eine einzufügende Zeile alle Wertebereichsvorgaben einhält, andernfalls wird sie verworfen [War03 S. 204f].

Bei der Wiederherstellung einzelner Tabellen muss auf folgende Constraints **keine** Rücksicht genommen werden:

- [NOT] NULL
- UNIQUE
- CHECK
- PRIMARY KEY

Die Integrität kann hierbei dadurch gewahrt werden, dass die Tabelle gelöscht und mit dem Schema zum Zeitpunkt der Sicherung wieder angelegt wird. Somit müssen alle Datenzeilen den gegebenenfalls vorhandenen Constraints entsprechen.

4.3. CHECK-Einschränkung

Auch wenn CHECK-Einschränkungen prinzipiell nicht beachtet werden müssen, gibt es einen Sonderfall, der behandelt werden muss: Tabellen, die selbst Primärschlüssel oder eindeutige Schlüssel enthalten und von anderen Tabellen per Fremdschlüssel referenziert werden, wobei auf dem Primär-/UNIQUE-Schlüssel eine CHECK-Einschränkung existiert.

Es existiere eine Tabelle `manager` mit dem Primärschlüssel `manager_id` (Typ `INT`) und einer CHECK-Einschränkung, die prüft, ob `manager_id<100` gilt. Weiterhin existiere eine Tabelle `sales` mit einem Fremdschlüssel auf der Spalte `manager_id`, welcher auf den Primärschlüssel in der Tabelle `manager` verweist. Davon ausgehend, dass bei Wiederherstellung der Tabelle `manager` in der Tabelle `sales` Datensätze enthalten sind, die einen Manager enthalten, der noch nicht in der `manager`-Tabelle existiert[14], müsste dieser angelegt werden um die Konsistenz zu wahren. Handelt es sich hierbei um Datensätze in der Tabelle `sales` mit einer `manager_id` größer als 100, würde dies der CHECK-Einschränkung aus der `manager`-Tabelle widersprechen und ein Einfügen zur Konsistenzwahrung wäre nicht möglich.

```
SELECT a.name AS IndexName, c.name AS ColumnName, d.name AS CheckName,
      d.definition AS CheckDefinition
FROM sys.indexes a
INNER JOIN sys.index_columns b ON a.object_id=b.object_id
INNER JOIN sys.syscolumns c ON a.object_id=c.id AND b.column_id=c.colid
INNER JOIN sys.check_constraints d ON a.object_id=d.parent_object_id AND
      b.column_id=d.parent_column_id
WHERE a.is_primary_key=1
ORDER BY a.name, b.index_column_id
```

Listing 1: Abfrage von Schlüsselbeziehungen bei CHECK-Einschränkungen

Vorstehende Abfrage listet alle vorhandenen Primärschlüssel und zugehörigen Spalten auf, die in einer CHECK-Einschränkung verwendet werden und gibt auch die angewandte CHECK-Einschränkung im Klartext aus. Für tabellenbezogene CHECK-Einschränkungen enthält die Spalte `parent_column_id` in der Tabelle `check_constraints` den Wert 0, somit kann auch kein Bezug mehr zu Spalten aus einem Primärschlüssel hergestellt werden. Für diesen Fall muss bei Existenz einer CHECK-Einschränkung auf Tabellenebene immer die Gültigkeit der

[14] Dies kann zutreffen, weil beispielsweise zwischen Backup und aktuellem Datenstand ein großer Zeitraum liegt, und in dieser Zeit viele Änderungen in der Datenbank geschehen sind

Werte geprüft werden, unabhängig davon, ob ein Primär- oder UNIQUE-Schlüssel davon betroffen ist oder nicht.

Zusammenfassend kann gesagt werden, dass CHECK-Einschränkungen nur berücksichtigt werden müssen, falls diese nur einen Bezug zur gesamten Tabelle aufweisen.

4.4. Trigger

Trigger sind eigenständige Datenbankobjekte und dafür verantwortlich, dass bei einer spezifizierten SQL-Anweisung auf eine Tabelle oder den SQL Server eine vorgegebene Aktion ausgeführt wird. Es gibt hierbei drei Arten von Triggern:

- DML[15]-Trigger reagieren auf Änderungen am Datenbestand
- DDL[16]- Trigger werden bei Änderungen am Datenbankschema ausgelöst
- LOGON-Trigger werden beim Beginn einer Benutzersitzung in SQL ausgelöst

Für weitere Betrachtungen sind DDL- und DML-Trigger von Bedeutung. Ein einfaches Beispiel für einen DML-Trigger kann sein, dass bei Einfügen eines Verkaufs in eine `sales`-Tabelle in einer entsprechenden `manager`-Tabelle der kumulierte Umsatz des Verkäufers mitgeschrieben wird. So reagiert der Trigger auf ein `INSERT INTO dbo.sales …` und führt ein `UPDATE dbo.manager SET volume_of_sales = …` aus. Auf diese Weise kann auch Datenkonsistenz gewährleistet werden.

Auf Trigger muss bei einer Wiederherstellung Rücksicht genommen werden, da bei Wiederherstellung einer Tabelle, je nach Typ des Triggers, Aktionen ausgelöst werden können, die ungewollt den Datenbestand verändern. Über die Systemtabellen einer Datenbank lassen sich referenzierte Tabellen und Spalten eines Triggers auslesen, wie auch das Ereignis, auf welches der Trigger reagiert.

---

[15] [War03 S. 152], diese Trigger reagieren auf die SQL Befehle `INSERT`, `DELETE` und `UPDATE` [Kon07 S. 259ff]

[16] [War03 S. 152], sind auf Datenbankebene definiert und reagieren in der Hauptsache auf die SQL Befehle `CREATE`, `DROP` und `ALTER` [Kon07 S. 289ff] [Woo07 S. 73]

Dies kann durch folgenden Codeschnipsel erledigt werden:

```
SELECT DISTINCT CASE WHEN OBJECT_NAME(a.parent_id) IS NULL THEN 'DATABASE'
      ELSE OBJECT_NAME(a.parent_id) END AS TableName, g.name AS ColumnName,
      d.name AS TableSchema, a.name AS TriggerName, f.name AS TriggerSchema
FROM sys.triggers a
INNER JOIN sys.sysdepends b ON a.object_id=b.id
LEFT JOIN sys.objects c ON a.parent_id = c.object_id
LEFT JOIN sys.schemas d ON c.schema_id = d.schema_id
LEFT JOIN sys.objects e ON a.object_id = e.object_id
LEFT JOIN sys.schemas f ON e.schema_id = f.schema_id
LEFT JOIN sys.syscolumns g ON b.depid=g.id AND b.depnumber=g.colid
WHERE a.is_disabled=0
```

Listing 2: Auslesen der Tabellenbeziehungen bei Triggern

Der eigentliche Zweck eines Triggers bleibt jedoch für das Programm unverständlich. Das Skript, welches durch den Trigger ausgeführt wird, kann über Systemprozeduren von SQL Server ausgelesen werden. Um die Funktionsweise eines Triggers maschinell verstehen zu können, müsste dieses Skript aber unter Zuhilfenahme eines Parsers analysiert werden. Dieses Vorhaben wäre zu umfangreich für diese Arbeit und kann daher in diesem Rahmen nicht berücksichtigt werden. So kann für eine Datenbank, welche Trigger enthält, lediglich eine vollständige Wiederherstellung als konsistent angesehen werden, da eine Auswirkung des Triggers nicht voraussehbar ist. Lediglich Tabellenbeziehungen von Triggern können als Grundlage für die Reaktion auf vorhandene Trigger verwendet werden.

Es bleibt zu sagen, dass Trigger vor einer Wiederherstellung deaktiviert und anschließend wieder aktiviert werden sollten, um unerwartete Nebenwirkungen auszuschließen oder minimieren zu können.

4.5. Views

Views, also Sichten auf Tabellen, können anstatt der bekannten `INSTEAD OF`-Trigger (siehe Abschnitt 4.4) auch die Angabe `WITH CHECK OPTION` enthalten. Diese wird dafür verwendet, bei Datenänderungen, die direkt über den View ausgelöst werden, wie eine `INSERT`- oder `UPDATE`-Anweisung die Daten auf Korrektheit zu überprüfen [Fre98 S. 366]. Da dieses Verhalten in der Regel nur bei der direkten Arbeit mit der Datenbank Anwendung findet, und

alle Sicherungs- und Wiederherstellungsvorgänge des Programms bei der Arbeit mit Datenbeständen nur auf Tabellen und nicht auf Sichten[17] arbeiten soll, ist hier keine weitere Behandlung notwendig.

4.6. Referenzielle Integrität

Die referenzielle Integrität gewährleistet die Konsistenz zwischen Tabellen und ist für den Fall der Wiederherstellung einer Tabelle der sicher häufigste Fall, bei dem Inkonsistenzen auftreten können. Um eine referenzielle Integrität zu definieren, müssen zwischen den entsprechenden Tabellen Schlüsselbeziehungen existieren [War03 S. 97f]. Man unterscheidet dabei:

- Primärschlüssel (Primary Key, PK)
- Fremdschlüssel (Foreign Key, FK)

Ein Primärschlüssel in einer Tabelle definiert einen eindeutigen Schlüssel für eine Zeile und kann aus mehreren Spalten bestehen, aber nur ein einziges Mal für jede Tabelle existieren. Werden mehrere Spalten verwendet, muss die Kombination dieser Spalten einen eindeutigen Wert ergeben [Per03 S. 296f].

Ein Fremdschlüssel in einer Tabelle kann nun auf diesen Primärschlüssel oder auch einen UNIQUE-Schlüssel (siehe Abschnitt 4.2) verweisen. Dabei kann der Fremdschlüssel auch aus mehreren Spalten bestehen, muss sich aber auf den Primärschlüssel beziehungsweise UNIQUE-Schlüssel beziehen [Per03 S. 298f]. Mit diesem Schritt ist bereits eine Konsistenz der Daten gewährleistet, denn bei Definition des Fremdschlüssels kann festgelegt werden, was im Falle einer Schlüsselverletzung geschehen soll.

```
CREATE TABLE sales (
      sales_id INT PRIMARY KEY,
      cust_id INT NULL REFERENCES customers (cust_id)
            ON DELETE SET NULL
            ON UPDATE CASCADE,
      price MONEY,
      sales_date DATETIME,
);
```

Listing 3: Fremdschlüsseldefinition bei Erstellung einer Tabelle

[17] Sichten sind reine Schemaobjekte und somit nicht Träger von Daten

Dieses Beispiel zeigt die Syntax zum Erstellen der Schlüssel bei Erstellung einer Tabelle. Damit ist die Spalte `sales_id` als Primärschlüssel definiert und die Spalte `cust_id` bezieht sich auf Spalte `cust_id` aus der Tabelle `customers`. Voraussetzung für eine ordnungsgemäße Funktion der Schlüsselbeziehungen ist natürlich, dass in der Tabelle `customers` die Spalte `cust_id` ebenfalls als Primär- oder UNIQUE-Schlüssel angelegt wurde. Für die `ON DELETE`- und `ON UPDATE`-Anweisungen gibt es 4 unterschiedliche Aktionen, die ausgeführt werden können [Per03 S. 303][18] [War03 S. 198ff]:

- *`NO ACTION`* – von SQL wird ein Fehler ausgelöst und die ausgeführte Aktion wird rückgängig gemacht (Rollback)
- *`CASCADE`* – betroffene Zeilen aus der verweisenden Tabelle werden gelöscht/aktualisiert
- *`SET NULL`* – betroffene Zeilen aus der verweisenden Tabelle werden auf `NULL` gesetzt (Voraussetzung ist, dass die Fremdschlüsselspalte `NULL` als Wert zulässt)
- *`SET DEFAULT`* – betroffene Zeilen aus der verweisenden Tabelle werden auf ihren definierten Standardwert gesetzt

Im vorigen Beispiel bedeutet das nun, wenn ein Kunde in der `customer`-Tabelle gelöscht wird und für den Kunden Umsätze in der `sales`-Tabelle enthalten sind, wird die `cust_id`-Spalte in der `sales`-Tabelle für diese Kunden auf `NULL` gesetzt. Im zweiten Fall, bei einer Aktualisierung der Kundennummer, werden entsprechende Zeilen aus der `sales`-Tabelle ebenfalls aktualisiert und mit der neuen Kundennummer versehen.

Insofern auf der zu wiederherzustellenden Datenbank Schlüsselbeziehungen existieren, können diese für die Wahrung der Integrität verwendet werden. An dieser Stelle muss aber beachtet werden, dass bei Vorhandensein von Schlüsselbeziehungen mit großer Wahrscheinlichkeit Inkonsistenzen oder auch Datenverlust auftreten, sobald eine wiederherzustellende Tabelle einer Schlüsselbeziehung einen älteren oder neueren Datenstand aufweist als die restlichen beteiligten Tabellen. In einem solchen Fall muss in geeigneter Weise durch das Programm in einem ersten Schritt geprüft werden, ob Schlüsselbeziehungen existieren, ob durch eine Wiederherstellung einer Tabelle Inkonsistenzen auftreten können und entsprechend den Nutzer zu einer Reaktion auf die Konsistenzprobleme auffordern.

[18] In der Quelle wird von einer weiteren Reaktion `RESTRICT` gesprochen, welche in SQL Server aber nicht implementiert ist

5. Produktdefinition

5.1. Problemanalyse

Für das Backup und Restore von Datenbanken sind nur wenige Softwareprodukte von Drittanbietern am Markt verfügbar (siehe Abschnitt 1.2). Allen gemeinsam ist, dass sie kommerziell vertrieben werden und dass es sich um proprietäre Software handelt. Nur wenige dieser Produkte bieten laut Spezifikation die Möglichkeit aus einer Datenbanksicherung einzelne Elemente wiederherzustellen. Einem Einsatz von kommerziellen Produkten steht die fehlende Anpassbarkeit und Revisionsfähigkeit gegenüber. Bereits eingehend wurde erläutert, dass auch die Mechanismen von SQL Server nicht in der Lage sind, aus einer bestehenden Datenbanksicherung einzelne Elemente zu extrahieren.

Es soll also eine Möglichkeit gefunden werden, Daten zu sichern und entsprechend so zu verwalten, dass daraus einzelne Schema- oder Datenobjekte wiederhergestellt werden können und den arbeitstechnischen wie auch zeitlichen Aufwand eines Restores zu minimieren. Eine Umsetzung dieser Lösung in TSQL bringt zudem Vorteile mit sich:

- Revisionsfähigkeit und Anpassbarkeit, da TSQL eine Skriptsprache ist und somit der Quellcode offen liegt
- Integration in Stored Procedures und SQL Server Agents

Für einen einfacheren Umgang mit dem Programm und zur Hervorhebung in dieser Arbeit soll weiterhin der Name **TSQL-Backup** für das zu schaffende Werkzeug verwendet werden.

5.2. Spezifikation des Produkts

5.2.1. Zielbestimmungen

Muss-Kriterien:

- Sicherung und Wiederherstellung von Datenbanken
- Wiederherstellung von Datenobjekten aus einer Sicherung
- Logbuch-Funktion zur Speicherung der Textausgabe an der SQL Befehlszeile
- Prüfung und Ausgabe der Syntax bei fehlerhaftem Aufruf

Soll-Kriterien:

- Sicherung einer Datenbank auf vollständiger, differenzieller oder inkrementeller Basis

- Auslesen und Wiederherstellen des Datenbankschemas
- automatische Korrektur von Dateninkonsistenzen bei einem Wiederherstellung-vorgang
- Löschen von angefertigten Backups
- Wiederherstellen von Datenbanken aus differenziellen und inkrementellen Sicherungen (dabei sollen alle abhängigen Sicherungen automatisch wiederhergestellt werden)
- Zugriff auf Metadaten (Einsicht in alle gesicherten Elemente eines Backups)
- Komprimierung erstellter Backups

Kann-Kriterien:

- Voreinstellungen sollen die Arbeit mit **TSQL-Backup** erleichtern
- Verifikation zwischen zu sichernder Datenbank und zu erstellendem Backup, sowie wiederhergestellter Datenbank und zugehöriger Sicherung (Prüfung auf Richtigkeit der ausgeführten Vorgänge)

Abgrenzungskriterien:

- keine Sicherung in oder Wiederherstellung aus Microsoft-eigenem Backup-Format
- keine grafische Oberfläche
- es können keine Systemdatenbanken mit **TSQL-Backup** gesichert werden

5.2.2. Produkteinsatz

Anwendungsbereich:

TSQL-Backup stellt Sicherungs- und Wiederherstellungfunktionen für SQL Server Datenbanken bereit, dabei soll ein völlig neuer Ansatz verwendet werden. Basisfunktionen sollen die vollständige und partielle Sicherung und die Wiederherstellung von SQL Server Datenbanken sein. Alleinstellungsmerkmal ist aber die Wiederherstellung einzelner Objekte aus angefertigten Sicherungen. Durch diese Funktionalität des Object-Level-Recovery soll Datenbankadministratoren ein Programm zur Verfügung gestellt werden, welches Lücken in vorhandenen Sicherungsstrategien ergänzen und so den Prozess des Disaster Recovery optimieren soll. Nebensächlich wird es möglich sein, auf Basis von Schemadaten aus Sicherungen neue, leere Datenbanken zu erzeugen.

Zielgruppe:

Die Arbeiten mit dem Programm geschehen direkt auf Datenbankebene und dienen der Sicherung einer Datenbank. Damit umfasst die Zielgruppe die Datenbankadministratoren. Da es somit nur eine Anwendergruppe gibt, werden kein Rollenkonzept oder verschiedene Berechtigungsstufen benötigt.

Es wird ein Grundverständnis im Umgang mit SQL Server, der SQL Befehlsreferenz und dem Aufbau von Datenbanken vorausgesetzt. Ein Verständnis für Datenbankmodelle und Konsistenzbeziehungen innerhalb einer Datenbank sind ebenso Voraussetzung, da ein Eingreifen des Nutzers bei Konsistenzproblemen notwendig ist.

Betriebsbedingungen:

Der vom Programm selbst beanspruchte Speicherplatz wird zu vernachlässigen sein und sich im Bereich weniger hundert Kilobyte bewegen. Durch die Arbeit mit **TSQL-Backup** entstehen aber Datenmengen in nicht unerheblicher Größe. Es muss also ausreichend Speicherkapazität für die Backupdateien zur Verfügung stehen. Dies kann jeder in Windows als Laufwerk einzubindender Datenträger sein, auch die Einbindung eines Netzlaufwerks ist möglich. Für die Arbeit benötigt das Programm eine eigene Metadatenbank, welche auf dem Server liegen muss und auf dem sich auch die zu sichernden Datenbanken befinden. Der zu erwartende Speicherverbrauch durch Backups und die Metadatenbank wird abschließend im Abschnitt 8.3.2 ermittelt.

5.2.3. Produktübersicht

Prinzipiell ist zu sagen, dass durch die Umsetzung in TSQL keine eigenständige Anwendung entsteht. Es wird sich bei **TSQL-Backup** um eine Erweiterung der Funktionalität von SQL Server handeln, genauer: eine Prozedur innerhalb von SQL Server. Hierbei wird auf eine bestehende Client-Server-Architektur von SQL Server aufgesetzt, **TSQL-Backup** wird dabei durch einen Client – dies wird in der Regel das SQL Server Management Studio sein – gesteuert. Nachfolgend soll daher eine Übersicht über die Strukturen um **TSQL-Backup** wiedergegeben werden:

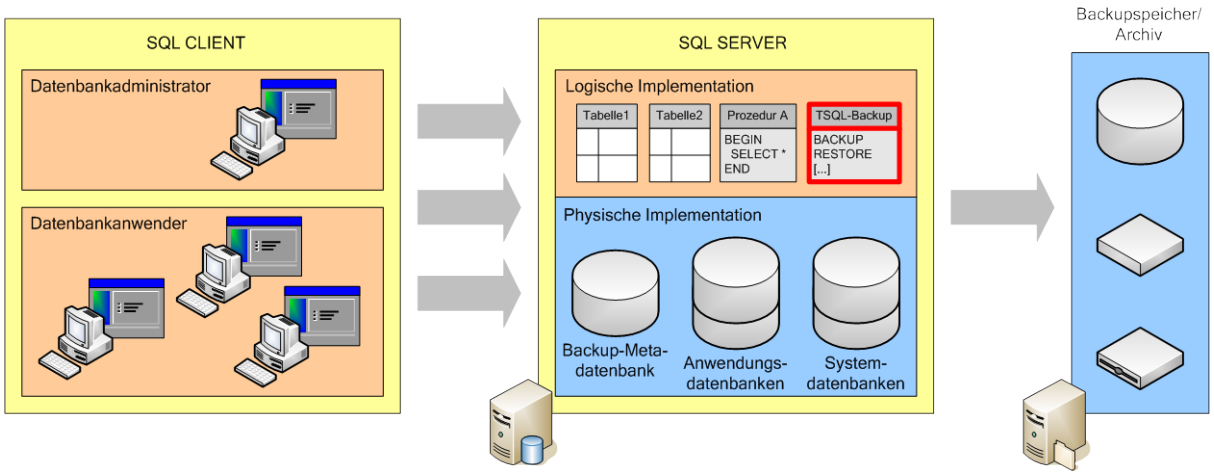

Abbildung 7: Produktübersicht

TSQL-Backup wird also in der bestehenden 2-Schichten-Architektur von SQL Server in die Server-Schicht integriert.

5.2.4. Produktfunktionen

Systemfunktionen

/PF010/ *Syntaxausgabe*: Ohne übergebene Parameter an **TSQL-Backup** soll eine Ausgabe der Syntax erfolgen, die optisch an den Stil der SQL Dokumentation in MSDN angelehnt ist

/PF020/ *Syntaxüberprüfung*: Alle an **TSQL-Backup** übergebenen Parameter sollen vor Ausführung einer Funktion auf syntaktische und semantische Korrektheit geprüft werden

/PF030/ *Anzeige der Voreinstellungen*: Bestimmte Parameter sollen mit Standardwerten initialisierbar sein, um den Aufruf von **TSQL-Backup** zu vereinfachen

/PF040/ *Ändern und Setzen der Voreinstellungen*: Voreinstellungen sollen durch den Nutzer geändert oder hinzugefügt werden können; folgende Voreinstellungen sollen berücksichtigt werden:

- Verzeichnisvorgaben
- Komprimierungseinstellungen

Meta-Funktionen

/PF110/ *Anzeige der vorhandenen Backups*: Alle angefertigten Sicherungen durch **TSQL-Backup** sollen tabellarisch angezeigt werden; darzustellen ist dabei:

- ID des Backups
- Name der gesicherten Datenbank
- Typ der Sicherung
- Datum des Sicherungsvorgangs
- Erfolgsstatus des Vorgangs
- Kurzbeschreibung

/PF120/ *Anzeige der gesicherten Elemente*: Für ausgeführte Sicherungen sollen dem Nutzer die einbezogenen Elemente angezeigt werden, welche in der Sicherung enthalten sind; folgende Informationen sollen ausgegeben werden:

- ID des gesicherten Elements
- Typ des Elements
- Besitzer
- Name des Elements

/PF130/ *Führen eines Logbuchs*: Für alle ausgeführten Vorgänge in **TSQL-Backup** soll ein Logbuch geführt werden, welches, mit Zeitstempel versehen, alle Ausgaben von **TSQL-Backup** mitschreibt

/PF140/ *Anzeige des Logbuchs*: Für angefertigte Sicherungen soll der Nutzer das zugehörige Logbuch abrufen können, dabei sollen listenartig, chronologisch aufsteigend alle, beziehungsweise zu einem Sicherungsvorgang gehörende, Einträge ausgegeben werden

/PF150/ *Löschen von Sicherungen*: Dem Nutzer soll es möglich sein, angefertigte Sicherungen zu Wartungszwecken und für Bereinigungsarbeiten durch **TSQL-Backup** wieder löschen zu lassen, dabei sollen physikalisch vorhandene Backupdateien und zugehörige Metadaten gelöscht werden; folgende Löschvorgänge sollen umgesetzt werden:

- Löschen aller vorhandenen Daten
- Löschen eines bestimmten Backups unter Angabe der Backup-ID
- Löschen aller Backups einer bestimmten Datenbank unter Angabe des Datenbanknamens

- Löschen von Backups unter Beibehaltung einer frei definierbaren Anzahl von zuletzt angefertigten Backups
- Löschen von Backups unter Beibehaltung von gesonderten Schemasicherungen

Sicherungsfunktionen

/PF210/ *Schemasicherung*: Der Nutzer soll getrennt vom Datenbestand einer Datenbank nur das Schema dieser Datenbank sichern können

/PF220/ *Vollständige Sicherung*: Eine Datenbank soll vollumfänglich gesichert werden, dabei sind Schemaobjekte und der Datenbestand zu erfassen

/PF230/ *Differenzielle Sicherung*: Es sollen nur Unterschiede zur letzten vollständigen Sicherung dokumentiert und gesichert werden; das Datenbankschema ist ebenfalls zu sichern

/PF240/ *Inkrementelle Sicherung*: Es sollen nur Unterschiede zur letzten inkrementellen beziehungsweise vollständigen Sicherung dokumentiert und gesichert werden; das Datenbankschema ist ebenfalls zu sichern

/PF250/ *Komprimierung von Sicherungen*: Für angefertigte Sicherungen sollen physikalisch erzeugte Backupdaten komprimiert werden können

Wiederherstellungsfunktionen

/PF310/ *Schemawiederherstellung*: Aus allen Sicherungsarten soll es möglich sein, auf Basis des gesichertes Schemas eine neue, leere Datenbank zu erstellen; vorhandene Datenbanken sollen überschrieben werden können

/PF320/ *Wiederherstellung aus vollständiger Sicherung*: Eine Datenbank soll aus einer vollständigen Sicherung mit allen gesicherten Daten wiederhergestellt werden; vorhandene Datenbanken sollen überschrieben werden können

/PF330/ *Wiederherstellung aus Teilsicherung*: differenzielle und inkrementelle Sicherungen sollen wiederherstellbar sein, dazu sollen automatisch alle zugehörigen, vorhergehenden Sicherungen eingespielt werden (bei differenziellen Sicherungen betrifft dies die vorhergehende vollständige

Sicherungen und bei inkrementellen Sicherungen die vorhergehende vollständige und alle dazwischen angefertigten inkrementellen Sicherungen); vorhandene Datenbanken sollen überschrieben werden können

/PF340/ *Wiederherstellung eines Objektes aus einer Sicherung*: Aus vollständigen, differenziellen und inkrementellen Sicherungen sollen sich folgende Objekte einzeln wiederherstellen lassen:

- Tabellen
- Sichten
- Funktionen
- Prozeduren

vorhandene Objekte sollen dabei überschrieben werden können; bei der Wiederherstellung aus einer Teilsicherung sollen gemäß /PF330/ alle zugehörigen, vorhergehenden Sicherungen ebenfalls zur Wiederherstellung herangezogen werden

/PF350/ *Konsistenzprüfung*: Bei der Wiederherstellung sollen, insofern notwendig, Konsistenzprüfungen stattfinden, um festzustellen, ob der Datenbestand nach der Wiederherstellung konsistent ist

/PF360/ *Vorgehen bei Konsistenzfehlern*: Wenn die Konsistenzprüfung nach /PF350/ negativ ausfällt, dann soll der Nutzer die Möglichkeit haben eine Wiederherstellung mit folgenden Optionen zu erzwingen (andernfalls wird eine Wiederherstellung nicht ausgeführt):

- Versuch der Konsistenzwiederherstellung durch **TSQL-Backup**, gegebenenfalls durch Löschen von Daten
- Versuch, alle Beziehungen, welche den Konsistenzfehler hervorrufen würden, aufzulösen, um bei der Datenwiederherstellung keine Daten zu verlieren

5.2.5. Produktdaten

/PD010/ *Voreinstellungen (max. 10)*

Parameter – Text

Wert – Text

/PD020/ *Parameter (max. 50)*

ID – Zahl, eindeutig

Parameter – Text

Wert – Text, optional

Typ – {INT/STRING/ARRAY}

Elternelement – Zahl, optional (bildet eine Hierarchie ab)

/PD030/ *Backupdaten (unbegrenzt)*

ID – Zahl, eindeutig

Typ – Text

Datenbankname – Text

Datum – Zeitstempel

Kurzbeschreibung – Text, optional

Zustand – {finished/not_finished}

Hashs vorhanden – {ja/nein}

Elternbackup – Zahl, optional (ID des vorgehenden Backups bei Teilsicherungen)

Logbuch-ID – Zahl

/PD040/ *Objektdaten (unbegrenzt)*

ID – Zahl, eindeutig

Backup-ID – Zahl (Zuordnung des Objekts zum Backup)

Typ – Text

Name – Text

Besitzer – Text, optional

Skript – Text, optional (SQL Befehl zur Erzeugung des Objekts)

komprimiert – {ja/nein} (zeigt bei Tabellen an, ob diese komprimiert wurden)

/PD050/ *Hashwerte (unbegrenzt)*

Backup-ID – Zahl (Zuordnung zu einem Backup)

Objekt-ID – Zahl (Zuordnung zu einem Objekt)

Hash – Binärdaten (Hashwert über eine Datenzeile)

gelöscht – {ja/nein} (zeigt an, ob es sich um einen Hashwert handelt, der bei

einer Teilsicherung nicht mehr existierte, das heißt, es handelt sich um eine gelöschte Datenzeile)

/PD060/ *Logbuch (unbegrenzt)*

ID – Zahl, eindeutig

Log-ID – Zahl (ID eines Vorgangs)

Datum – Zeitstempel

Text – Text, optional

5.2.6. Produktleistungen

/PL010/ *Identität*

In der Metadatenbank erzeugte Einträge zu Sicherungsvorgängen im Logbuch und Daten zu gesicherten Objekten müssen aus Konsistenzgründen eindeutig identifizierbar sein

/PL020/ *Fehler*

Durch das Programm oder den Nutzer verursachte Fehler sollen akkumuliert ausgegeben werden

/PL030/ *Performanz Metadaten*

Eine Anfrage an die Metadaten muss, insofern keine Sicherungsvorgänge parallel laufen, innerhalb von 3 Sekunden beantwortet werden

/PL040/ *Performanz Ausführung*

Während der Arbeit mit **TSQL-Backup** sollen parallele Nutzerprozesse nicht behindert (blockiert) werden

/PL050/ *Bedienung*

Der Aufruf von **TSQL-Backup** soll für den Nutzer intuitiv und knapp möglich sein

5.2.7. Qualitätsanforderungen nach ISO/IEC 9126

Produktqualität	sehr gut	gut	normal	irrelevant
Funktionalität				
Angemessenheit	X			
Richtigkeit	X			
Interoperabilität			X	
Sicherheit	X			
Ordnungsmäßigkeit		X		
Zuverlässigkeit				
Reife	X			
Fehlertoleranz		X		
Robustheit	X			
Wiederherstellbarkeit			X	
Benutzbarkeit				
Verständlichkeit		X		
Erlernbarkeit			X	
Bedienbarkeit		X		
Effizienz				
Zeitverhalten		X		
Verbrauchsverhalten			X	
Änderbarkeit				
Analysierbarkeit			X	
Modifizierbarkeit			X	
Stabilität			X	
Prüfbarkeit			X	
Übertragbarkeit				
Anpassbarkeit				X
Installierbarkeit		X		
Konformität				X
Austauschbarkeit				X

Tabelle 2: Qualitätsanforderungen an TSQL-Backup

Funktionalität

TSQL-Backup soll in jedem Fall Sicherungs- und Wiederherstellungsvorgänge korrekt durchführen, die enthaltenen beziehungsweise wiederhergestellten Daten müssen also „richtig“ sein, daher werden Funktionalität und Zuverlässigkeit hoch bewertet. Wichtigstes

Kriterium ist, die Quelldaten bei der Arbeit durch **TSQL-Backup** in keinster Weise zu verändern.

Zuverlässigkeit

Es ist ebensfalls wichtig einen unbefugten Zugriff zu verhindern, wie auch Fehleingaben des Nutzers und Fehler, die während der Arbeit mit **TSQL-Backup** entstehen, korrekt zur behandeln. Unvollständige oder fehlende Angaben in der Parameterliste des Programms sollen dem Anwender möglichst deskriptiv mitgeteilt werden. Diese Fehler werden auf der TSQL-Befehlszeile zurückgegeben und können somit durch den Anwender, beispielsweise durch Anpassung der Parameterliste, korrigiert werden. Insofern **TSQL-Backup** Hinweise oder Fehler während der Arbeit ausgibt, kann dies zwei Ursachen haben:

1. Tritt während der Arbeit ein Fehler auf, der durch **TSQL-Backup** festgestellt wurde, soll der Nutzer darauf hingewiesen werden und ihm die möglichen Optionen angeboten werden, mit denen er **TSQL-Backup** erneut aufrufen kann. Bei der Entwicklung muss also beispielsweise Rücksicht auf alle denkbaren Situationen genommen werden, bei denen durch ein Restore eine Inkonsistenz auftreten kann. Für diese Fälle soll dem Nutzer, insofern die Inkonsistenz durch das Programm überhaupt selbständig behoben werden kann, eine Option zur Verfügung stehen, die die Wiederherstellung erzwingt.
2. Handelt es sich hierbei aber um einen Fehler, der direkt durch SQL Server erzeugt wird, erhält der Nutzer zwar die Fehlerausgabe von SQL Server, muss aber an dieser Stelle den Fehler selbst behandeln, da **TSQL-Backup** auf diese Fehler nicht reagieren und diese auch nicht auswerten kann. Die Ursache hierfür liegt in dem Schweregrad des Fehlers, den SQL Server ausgibt. Ab einem bestimmten Schweregrad werden SQL Skripte sofort abgebrochen[19], somit ist auch keine Reaktion durch **TSQL-Backup** auf diese Fehler mehr möglich. Meist wird es sich hierbei um Konflikte in Schemaobjekten handeln.

[19] Schweregrade zwischen 19 und 25 bewirken eine Beendigung der Verbindung zwischen Server und Client [Bau06 S. 376]

Benutzbarkeit

Da die Zielgruppe für **TSQL-Backup** Datenbankadministratoren sind, wird ein gewisses Grundverständnis für SQL und die Funktionsweise von TSQL-basierten Befehlen vorausgesetzt, die Benutzbarkeit wird daher nicht so sehr gewichtet, obgleich die Bedienung von **TSQL-Backup** einfach sein soll.

Effizienz

Die Effizienz spielt an dem Punkt eine Rolle, wo Objekte aus Sicherungen wiederhergestellt werden sollen. Die Arbeit auf Basis von TSQL verdeutlicht, dass das Backup nur so schnell sein kann, wie der SQL Server selbst ist. Ein nativer Zugriff auf die Basisdateien ist an dieser Stelle sicher performanter. Nun ist auch zu bedenken, dass über jede einzelne Zeile eine Prüfsumme gebildet werden könnte, welche dann gegebenenfalls mit älteren Backups verglichen werden muss, um ein differenzielles oder inkrementelles Abbild zu erzielen. Dies ist ein sehr rechenintensiver Prozess. Ein abschließender Performanz-Test im Kapitel 8.3 soll schließlich Aufschluss über die Effizienz geben und zeigen, in welchen Größenordnungen hinsichtlich Speicherplatzverbrauch und Anzahl Datenzeilen das Programm am effizientesten arbeitet. Da ein Schwerpunkt von **TSQL-Backup** aber die Wiederherstellung einer einzelnen Tabelle aus einer Sicherung ist, sollte die Laufzeit des Wiederherstellungsvorgangs entstehende Laufzeitschwächen des Backups kompensieren. Der Verbrauch von Speicherplatz ist weniger relevant, da mit Komprimierungsverfahren gearbeitet werden kann.

Änderbarkeit und Übertragbarkeit

Änderbarkeit und Übertragbarkeit wurden sehr niedrig bewertet, da **TSQL-Backup** sehr nah am SQL-Standard von Microsoft arbeiten wird und daher der Einsatz auf anderen Betriebssystemen oder Datenbanksystemen ausgeschlossen wird. Für das Programm wird ein Initialisierungsskript erstellt, welches die Arbeitsumgebung einrichtet, Funktionen und Prozeduren einrichtet und Standardwerte einstellt. Diese Standardwerte und für **TSQL-Backup** notwendige Prozeduren werden in einer Datenbank gespeichert.

5.2.8. Benutzeroberfläche

TSQL-Backup wird aus Sicht des Nutzers vollständig auf TSQL-Ebene arbeiten, es erfolgt also eine klassische Interaktion zwischen Programm und Nutzer auf der SQL-Befehlskonsole, daher ist auch kein dialogbasiertes Arbeiten möglich. Das heißt, bei jedem Schritt im Programm, der ein Eingreifen des Nutzers erfordert, muss die Prozedur mit Erweiterung der Befehlszeile um die notwendigen Eingaben erneut aufgerufen werden. Wichtig hierbei ist es, dem Nutzer bei Aufruf der Prozedur ohne Parameter oder mit unvollständigen Parametern eine Hilfestellung zur Syntax zu ermöglichen. Standardwerte sollen dem Nutzer ermöglichen, die am häufigsten verwendeten Funktionen ohne aufwändige und komplexe Parameterübergabe zu nutzen.

Eine Benutzeroberfläche wird daher nicht von **TSQL-Backup** selbst, sondern durch den SQL Client zur Verfügung gestellt.

5.2.9. Nichtfunktionale Anforderungen

Eine Revisionsfähigkeit ist durch den offen liegenden Quellcode gegeben. Insofern Daten gesichert werden, die für eine Buchführung relevant sind, sollen Bestimmungen des Handelsgesetzbuches bezüglich Aufbewahrung und Zugänglichmachung dieser Daten erfüllt werden[20]. Weitere Gesetze oder Normen, wie beispielsweise Datenschutzrichtlinien, sind nicht vorgegeben.

Der Zugriffsschutz auf TSQL-Backup soll dadurch gegeben sein, dass eine Nutzung nur durch (Datenbank-) Administratoren möglich ist. Ein Zugriffsschutz auf die gesicherten Daten soll gewährleistet werden, dies kann durch einen Passwortschutz der erzeugten Daten und/oder einen physikalischen Zugriffsschutz (durch räumlich gesicherte) Lagerung der Daten erfolgen.

Das zu erwartende Datenaufkommen ist nicht schätzbar, es sollten aber Speichermedien eingesetzt werden, die auf die Ablage großer Dateien optimiert sind. Es sollte auch keine Begrenzung einer Dateianzahl pro Verzeichnis geben. Eine redundante Auslegung der Hardware, beispielsweise durch gespiegelte Festplattensysteme, wird aus Sicherheitsgründen empfohlen. Die Speichermedien sollten zudem mit ausreichend groß dimensionierter Bandbreite angebunden werden, um keinen Engpass bei der Arbeit mit Sicherungen zu bilden.

---

[20] § 257 HGB regelt unter anderem, dass elektronisch aufbewahrte Dokumente jederzeit lesbar gemacht werden müssen; http://www.gesetze-im-internet.de/hgb/__257.html

5.2.10. Umgebungsbedingungen

Die Entwicklung und der Testbetrieb finden in SQL Server 2005 Standard auf Basis eines Microsoft Windows Server 2003 Standard R2 statt. Durch Einsatz von SQL Server Standard können alle sonstigen Editionen von SQL Server bedenkenlos eingesetzt werden, da alle verwendeten SQL Funktionen in jeder Edition nutzbar sind. Einzige Ausnahme bildet die Express Edition von SQL Server, welche nur minimalistische Funktionen zum Betrieb von SQL Datenbanken bietet und an dieser Stelle ausgeschlossen wird. Durch den Einsatz von Microsoft SQL Server 2005 Standard bleiben auch Funktionen unberücksichtigt, welche nur in bestimmten Ausgaben des SQL Server[21] verfügbar sind. Dazu gehören beispielsweise partitionierte Tabellen und partitionierte Indizes, welche nur in der Enterprise, Developer und Evaluation Edition angeboten werden [Bau06 S. 25f]. Einem Einsatz in SQL Server 2008 steht nichts entgegen, auf SQL Server 2000 kann ein Betrieb nicht garantiert werden, da es Probleme in der Abwärtskompatibilität von SQL Befehlen geben kann.

Die Programmierung geschieht in SQL Server Management Studio, welches als Client-Applikation die Verbindung zu SQL Server herstellt und in Form des Query Analyzers die SQL Befehlskonsole am Arbeitsplatz zur Verfügung stellt. Da SQL Server Management Studio keine Möglichkeiten zur Dokumentation bietet, wird die Dokumentation direkt im Quellcode vorgenommen.

[21] Ausgenommen Microsoft SQL Server 2005 Express

6. Analyse

6.1. Anwendungsfälle

In Abschnitt 5.2.4 wurde durch die Produktfunktionen die funktionelle Sicht von **TSQL-Backup** dargestellt. Folgend sollen nun die Anwendungsfälle erarbeitet werden. Akteur wird in jedem Fall ein Datenbankadministrator sein. Vorgänge, die das Datenbankschema betreffen, sind jeweils im entsprechenden Anwendungsfall des Backups oder Restores enthalten, weil diese Funktionen einen Teil einer vollständigen Sicherung oder Wiederherstellung darstellen.

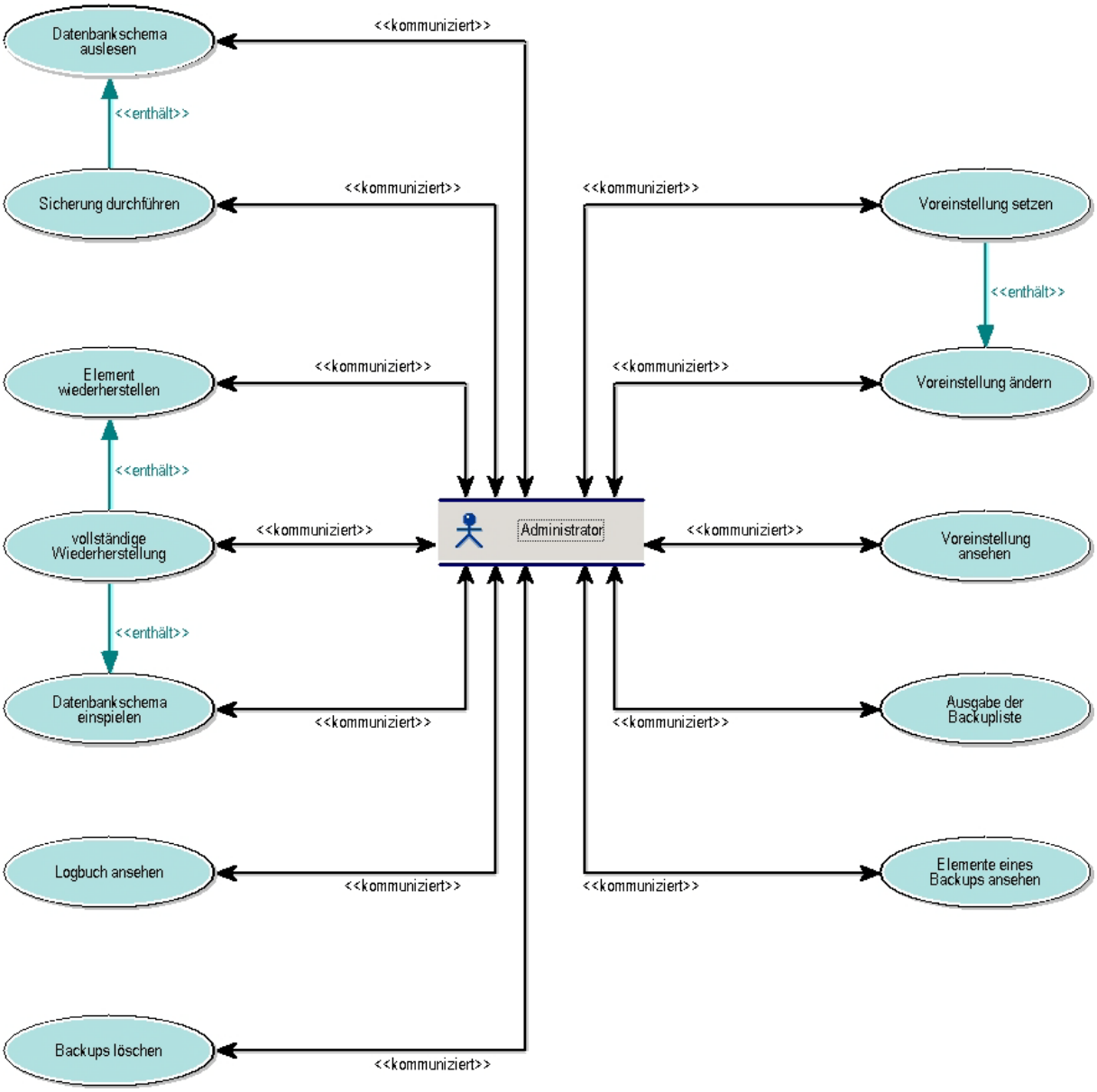

Abbildung 8: Anwendungsfalldiagramm

Nachfolgend soll exemplarisch auf einige wichtige Anwendungsfälle nach dem Schema von Cockburn [Sch07] näher eingegangen werden.

Use-Case	Datenbankschema auslesen
Ziel	Es sollen alle Schemaobjekte einer Datenbank extrahiert und in ein SQL Skript überführt werden
Vorbedingung	Die auszulesende Datenbank existiert im System
Nachbedingung Erfolg	In der Metadatenbank sind SQL Skripte abgelegt, mit denen die Schemaobjekte der Datenbank neu angelegt werden können
Nachbedingung Fehlschlag	Die Metadatenbank enthält keine Informationen über das Datenbankschema
Akteure	Datenbankadministrator
Auslösendes Ereignis	Der Benutzer übergibt als Aktionsparameter „Schema auslesen“ an **TSQL-Backup**
Eingabedaten	• Name der auszulesenden Datenbank
Beschreibung	1. Nutzer übergibt den Namen der auszulesenden Datenbank 2. **TSQL-Backup** liest das Datenbankschema aus 3. Datenbankschema wird in Form eines SQL Skripts in der Metadatenbank abgelegt 4. **TSQL-Backup** gibt Informationen über die ausgelesenen Schemaobjekte aus
Erweiterungen	-
Alternativen	-

Tabelle 3: Anwendungsfall "Datenbankschema auslesen"

Use-Case	Sicherung durchführen
Ziel	Eine Datenbank soll gesichert werden, dabei soll das Datenbankschema und der Datenbestand erfasst und gesichert werden
Vorbedingung	Die auszulesende Datenbank existiert im System
Nachbedingung Erfolg	In der Metadatenbank sind SQL Skripte zu den Schemaobjekten angelegt, sowie Informationen über das ausgeführte Backup und im physischen Backup-Speicher sind die exportierten Daten der Datenbank archiviert
Nachbedingung Fehlschlag	Die Metadatenbank enthält keine Informationen über das Datenbankschema und es wurden keine Daten exportiert
Akteure	Datenbankadministrator
Auslösendes Ereignis	Der Benutzer übergibt als Aktionsparameter „Sicherung durchführen“ an **TSQL-Backup**
Eingabedaten	• Name der zu sichernden Datenbank • Art der Sicherung (vollständig/differenziell/inkrementell) • Option zur Aktivierung der Komprimierung
Beschreibung	1. Nutzer übergibt den Namen der auszulesenden Datenbank 2. Anwendungsfall „Datenbankschema auslesen“ wird gestartet 3. Zu exportierende Daten werden gesammelt

	4. Datenbestand wird aus Datenbank exportiert 5. Metainformationen über exportierte Daten werden in der Metadatenbank abgelegt 6. **TSQL-Backup** gibt Informationen über die gesicherten Objekte aus
Erweiterungen	3a. Bei differenziellen oder inkrementellen Sicherungen müssen nur die geänderten Daten zum Export markiert werden 4a. Daten können komprimiert werden, wenn vom Nutzer gewünscht
Alternativen	-

Tabelle 4: Anwendungsfall "Sicherung durchführen"

Use-Case	Element wiederherstellen
Ziel	Aus einer von **TSQL-Backup** angefertigten Sicherung soll ein einzelnes Datenbankobjekt wiederhergestellt werden
Vorbedingung	Die Sicherung, aus der ein Objekt wiederhergestellt und die Datenbank in der das Objekt angelegt werden soll, müssen existieren
Nachbedingung Erfolg	Datenbankobjekt wurde aus der Sicherung in der Zieldatenbank angelegt und mit Daten befüllt (wenn es sich um ein Datenobjekt handelt)
Nachbedingung Fehlschlag	Zieldatenbank ist unverändert und es wurde kein Objekt angelegt
Akteure	Datenbankadministrator
Auslösendes Ereignis	Der Benutzer übergibt als Aktionsparameter „vollständige Wiederherstellung“ an **TSQL-Backup** und übergibt weitere Parameter zur Wiederherstellung eines einzelnen Elements
Eingabedaten	• ID der Sicherung welche verwendet werden soll • ID des Elements innerhalb der Sicherung welches wiederhergestellt werden soll • Name der Zieldatenbank • Optionen zur Konsistenzwahrung • Option zum Überschreiben eines vorhandenen Objekts
Beschreibung	1. Nutzer übergibt ID der Sicherung und des wiederherzustellenden Elements 2. Prüfung, ob Objekt noch nicht existiert, muss erfolgreich ablaufen 3. Sicherung wird ausgelesen und Element wird extrahiert 4. Konsistenzprüfung für Wiederherstellung muss erfolgreich ablaufen 5. Schema des wiederherzustellenden Objekts wird in der Zieldatenbank angelegt 6. Im Fall eines Datenobjekts werden ebenfalls die gesicherten Daten eingespielt 7. **TSQL-Backup** gibt Informationen über das Restore aus
Erweiterungen	6a. Handelt es sich bei der Sicherung um eine Teilsicherung, werden automatisch alle vorhergehenden Backups verwendet, um ebenfalls enthaltene Datenbestände einzuspielen
Alternativen	2a. Ist Option zum Überschreiben eines vorhandenen Objekts aktiviert, kann ein vorhandenes Element automatisch durch das

	System gelöscht werden 4a. Ist entsprechende Option zur Konsistenzwahrung aktiviert, kann durch das System automatisch eine Korrektur von Inkonsistenzen vorgenommen werden

Tabelle 5: Anwendungsfall "Element wiederherstellen"

Use-Case	vollständige Wiederherstellung
Ziel	Eine Datenbank soll aus einer Sicherung vollständig mit Schemaobjekten und Datenbestand wiederhergestellt werden
Vorbedingung	Sicherung, welche wiederhergestellt werden soll, muss existieren
Nachbedingung Erfolg	Datenbank wurde aus der Sicherung mit allen enthaltenen Informationen wiederhergestellt
Nachbedingung Fehlschlag	Keine Veränderung am betreffenden Server, keine Datenbank wurde angelegt beziehungsweise wiederhergestellt
Akteure	Datenbankadministrator
Auslösendes Ereignis	Der Benutzer übergibt als Aktionsparameter „vollständige Wiederherstellung“ an **TSQL-Backup**
Eingabedaten	• ID der Sicherung welche verwendet werden soll • Name der Zieldatenbank • Option zum Überschreiben eines vorhandenen Objekts
Beschreibung	1. Nutzer übergibt ID der Sicherung 2. Prüfung, ob Datenbank noch nicht existiert, muss erfolgreich ablaufen 3. Datenbank wird angelegt 4. Sicherung wird ausgelesen und für jedes enthaltene Element der Anwendungsfall „Element wiederherstellen“ gestartet 5. **TSQL-Backup** gibt Informationen über das Restore aus
Erweiterungen	-
Alternativen	2a. Ist Option zum Überschreiben eines vorhandenen Objekts aktiviert, kann eine vorhandene Datenbank automatisch durch das System gelöscht werden

Tabelle 6: Anwendungsfall "vollständige Wiederherstellung"

Der komplexeste Anwendungsfall der vollständigen Wiederherstellung soll anschließend noch in einem Aktivitätsdiagramm dargestellt werden, um die internen Abläufe besser zu verdeutlichen. Erkennbar ist, und dieses Verhalten zieht sich durch jeden Anwendungsfall, dass es nur einen Interaktionspunkt mit dem Nutzer gibt. Wie bereits erwähnt, ist es auf Basis der Entwicklung in TSQL für den Nutzer nur beim Aufruf von **TSQL-Backup** möglich mit dem Programm zu interagieren. Ein weiteres Eingreifen des Nutzers ist nicht möglich.

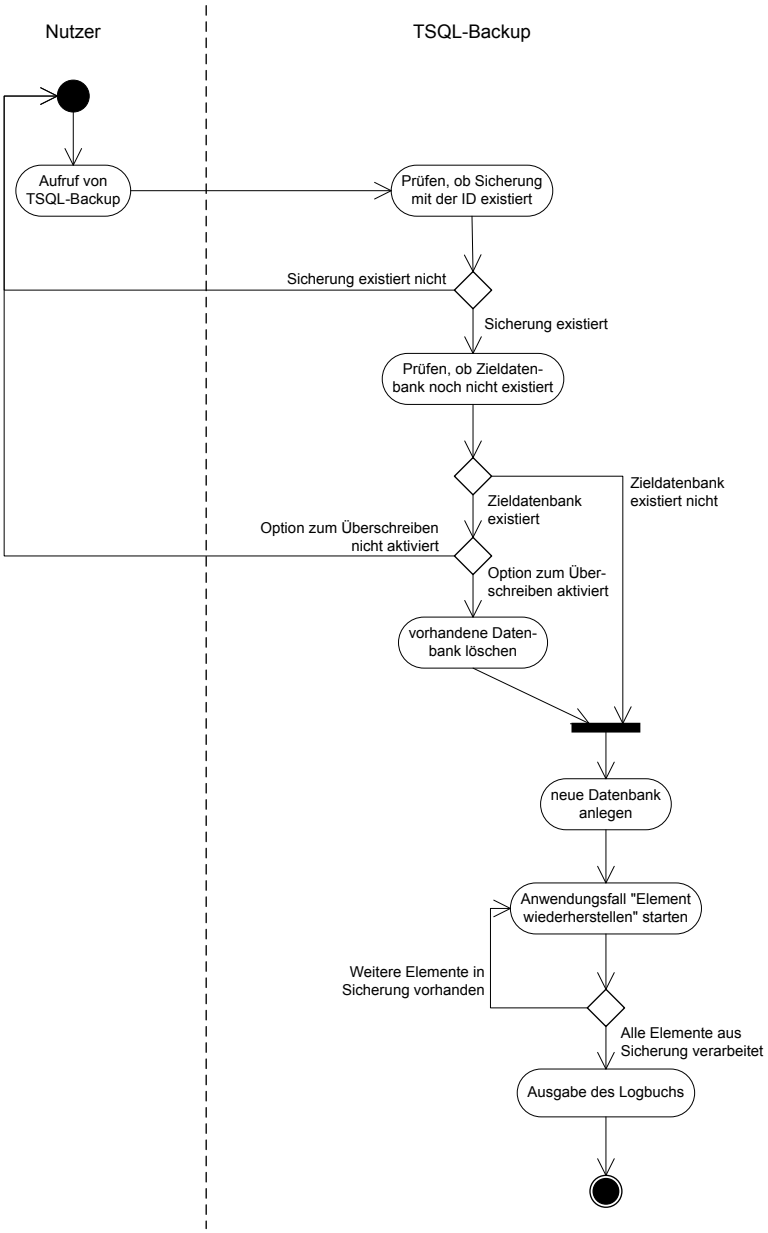

Abbildung 9: Aktivitätsdiagramm "vollständige Wiederherstellung"

6.2. Untersuchung zum SQL Server Transaktionsprotokoll

Für die folgenden Kapitel ist zu sagen, dass die verwendete Literatur meist nur unzureichend über bestimmte Interna von SQL Server informiert. Viele Fachbücher zu SQL Server enthalten einen hohen Anteil SQL Syntax, welche auch problemlos im Internet einsehbar ist, aber nur wenig Details über den internen Aufbau, was bei einem kommerziellen Produkt

verständlich erscheint. Hilfe konnte hier meist in speziellen SQL-Fachforen im Internet gefunden werden.

Es soll untersucht werden, ob es möglich ist, dass SQL Transaktionsprotokoll auszulesen, um damit einerseits Transaktionsprotokollsicherungen und –rücksicherungen wie in SQL Server durchführen zu können, andererseits aber auch, um mit Hilfe des Protokolls durch Rollback- und Rollforwardvorgänge die Datenkonsistenz bei Wiederherstellungen zu wahren (vergleiche Abschnitt **Fehler! Verweisquelle konnte nicht gefunden werden.**).

Das Transaktionsprotokoll von SQL Server liegt in einem binären, proprietären Format vor, welches nicht lesbar ist. Mit `DBCC LOG(<dbname>,-1)` lässt sich das Transaktionsprotokoll für eine Datenbank im höchsten Detailgrad ausgeben [Bau06 S. 133f], aber es ist nicht ohne Weiteres möglich, hieraus einen lesbaren und ausführbaren TSQL-Code zu erzeugen, der die angezeigten Transaktionen reproduzieren kann. Mit vorgenanntem `DBCC`-Befehl erhält man bei höchstem Detailgrad für jede Protokollfolgenummer [22] den Protokolleintrag in hexadezimaler Form. Um zu verdeutlichen, dass ein Lesen des Logs im Klartext nicht möglich ist, soll eine beliebiger solcher Protokolleintrag umgewandelt werden. Ein Beispiel für einen solchen hexadezimalen Eintrag lautet:

```
0x00003E000B0200001504000078000 2005E16000000000305AD52000001000D00590000000
B020000150400007700000100000A000000000101000000460002003401140048002601000 0
ED2C000000000300494620 4E4F5420455849535453202853454C454354202A2046524F4D207
379732E736368656D6173205748455245206E616D65203D204E2748756D616E5265736F7572
63657327290A45584543207379732E73705F6578656375746573716C204E274352454154452
0534348454D41205B48756D616E5265736F75726365735D20415554484F52495A4154494F4E
205B64626F5D270A474F0A4946204E4F5420455849535453202853454C454354202A2046524
F4D203A3A666E5F6C697374657874656E64656470726F706572747928 4E274D535F44657363
72697074696F6E27202C204E27534348454D41272C4E2748756D616E5265736F75726365732
72C204E554C4C2C4E554C4C2C204E554C4C2C4E554C4C29290A0000000000000000DE130000
00000101000C000059ED607F00000102000402030004
```

Für die Umwandlung soll das Freeware-Tool „XVI32“ [23] behilflich sein. Dazu wird der Eintrag ohne die führenden Zeichen `0x`, welche den Beginn einer hexadezimalen Zeichenkette kennzeichnen, in das Programm übertragen.

---

[22] Log Sequence Number, LSN: eine Protokollfolgenummer identifiziert eine Transaktion im Protokoll eindeutig, dabei können mehrere Protokolleinträge zu einer Transaktion gehören und sind durch Zeiger verknüpft [Mic0812]

[23] http://www.chmaas.handshake.de/delphi/freeware/xvi32/xvi32.htm

Das Programm erzeugt daraus folgende Ausgabe (übermäßige Leerzeichen und Zeilenumbrüche wurden dabei entfernt):

```
>
x ^ ————————————————————————-R  Y
w
F 4H &  í,    ———————————————————— IF NOT EXISTS (SELECT * FROM
sys.schemas WHERE name = N'HumanResources')
EXEC sys.sp_executesql N'CREATE SCHEMA [HumanResources] AUTHORIZATION
[dbo]'
GO
IF NOT EXISTS (SELECT * FROM ::fn_listextendedproperty(N'MS_Description' ,
N'SCHEMA',N'HumanResources', NULL,NULL, NULL,NULL))
   Þ  Yí`□
```

Deutlich erkennbar sind TSQL-Befehle, aber auch binäre Steuerzeichen.

Schon allein, dass der Befehl zum Auslesen des Transaktionsprotokolls in der Hilfe von SQL Server undokumentiert ist, weist darauf hin, dass ein direktes Eingreifen des Nutzers in das Protokoll durch Microsoft unerwünscht zu sein scheint. Auch in Fachbüchern ist dieser Befehl nur selten zu finden, wie beispielsweise in [Bau06 S. 133f].

Es existieren einige wenige Produkte, die das Transaktionslog auslesen können. Eines der bekanntesten Programme ist „ApexSQL Log Tool“[24], welches kommerziell vertrieben wird. Trotz aktiviertem vollständigen Wiederherstellungsmodell lässt das Tool aber bestimmte SQL-Anweisungen vermissen. So werden beispielsweise DML-Anweisungen auf Spalten vom Typ `TEXT` nicht dokumentiert. Dieser Spaltentyp kann nur mit speziellen Anweisungen von SQL Server gelesen und geschrieben werden. Allein dieser Nachteil und der nicht unerhebliche Preis für dieses Produkt lassen das Tool aber für den Einsatz in dieser Arbeit ausscheiden. „SQL Log Rescue“ [25] von redgate ist zwar frei herunterzuladen, arbeitet hingegen aber nur auf der Version SQL Server 2000.

[24] http://www.apexsql.com/sql_tools_log.asp

[25] http://www.red-gate.com/products/SQL_Log_Rescue/index.htm

Eine Art Transaktionsprotokollsicherung und -rücksicherung, wie sie unter SQL Server möglich ist, kann somit nicht realisiert werden. Auch die Konsistenzwiederherstellung in der Datenbank auf Grundlage des Transaktionsprotokolls ist im Rahmen von **TSQL-Backup** nicht möglich.

6.3. Untersuchung zum Auslesen des Datenbankschemas

6.3.1. Möglichkeiten zum Auslesen

Zweck von **TSQL-Backup** ist es, eine Datenbank (vollständig) zu sichern und wiederherstellen zu können. Neben den zu sichernden Daten gehört dazu auch das Schema der Datenbank, welches vorzugsweise auch einzeln ohne Daten gesichert werden soll. Angenehmer Nebeneffekt dieser Schemasicherung ist, dieses einzeln, ohne Wiederherstellung der Daten in einer neuen Datenbank, wieder einspielen zu können, um so eine neue, leere Datenbank nach dem Muster einer bereits vorhanden zu erhalten.

Nicht beachtet werden müssen Systemobjekte, wie Systemtabellen, Systemsichten oder Systembenutzer, da diese bei einer `CREATE DATABASE`-Anweisung standardmäßig nach Vorgabe der `model`-Datenbank angelegt werden [Woo07 S. 66].

Für das Auslesen des Schemas einer Datenbank kann es verschiedene Methoden geben. Grob lassen sich diese wie folgt klassifizieren:

- Skripterzeugung über grafische Oberflächen (wie SQL Server Management Studio)
- Skripterzeugung durch direktes Auslesen von Systemsichten und -tabellen und Zusammensetzen zu einem Skript
- Skripterzeugung durch Kommandozeilenbefehle/-programme

Da ein automatisierter Prozess geschaffen werden soll, kommen nur die beiden letzten Möglichkeiten in Betracht.

6.3.2. Integrierte Funktionen in SQL Server

Eine Erzeugung des Skripts für ein Schemaobjekt durch Auslesen von Systeminformationen ist die flexibelste Variante, da hier die durchaus komplexe TSQL-Syntax berücksichtigt werden kann und alle Parameter in die Erzeugung eines Skripts einfließen können. Für

bestimmte Schemaobjekte leistet SQL Server bereits Unterstützung, indem Systemprozeduren wichtige Informationen über Objekte liefern. Dazu gehören unter anderem:

- `sp_helpconstraint` liefert alle Einschränkungstypen für eine bestimmte Tabelle
- `sp_helpindex` gibt Informationen zu Indizes auf einer Tabelle zurück
- `sp_helptrigger` liefert Informationen zu Triggern
- `sp_helptext` zeigt die Definition einer nicht verschlüsselten Prozedur, einer Sicht, einer CHECK-Einschränkung, eines Triggers und anderer Objekte an

Es existieren weitere, wichtige Objekte, die auf diese Weise nicht ausgelesen werden können: Tabellen. Weiterer Nachteil an den Systemprozeduren ist, dass diese oft unterschiedliche, aber wichtige Information gleichzeitig in mehreren Resultsets gleichzeitig zurückgeben und deshalb ein automatisiertes Auslesen dieser Ergebnisse nicht möglich ist. Werden die Skripte durch Auslesen von Systeminformationen zusammengestellt, müssen hierbei aber auch alle zulässigen Parameter beachtet werden, es muss also die SQL-Syntax semantisch und syntaktisch korrekt nachgebildet werden. Allein die Syntax zum Erstellen einer Tabelle ist derart umfangreich [Fre98 S. 346ff], dass im Rahmen dieser Arbeit von dieser Vorgehensweise Abstand genommen werden muss, da eine Prozedur zur Erzeugung von Skripten für alle möglichen Objekte den zeitlichen Rahmen bei Weitem überschreitet.

6.3.3. Auslesen durch Hilfstools

DPW

Daher soll die dritte Variante, das Auslesen des Schemas über ein Kommandozeilenprogramm, genauer untersucht werden. Häufig im Internet zu finden ist der Microsoft Database Publishing Wizard [26] (weiterhin DPW). Dies ist ein Microsoft-Tool, welches über eine grafische Oberfläche, als auch per Kommandozeile zu bedienen ist. Voraussetzung zum Einsatz dieses Tools ist die Komponente „Verwaltungsobjektauflistung“ aus dem Microsoft FeaturePack für SQL Server 2005[27]. Sind beide Komponenten installiert, kann bereits ein Schema ausgelesen werden. Ein gravierender Unterschied zwischen der grafischen Oberfläche von DPW und der Kommandozeilenversion besteht darin, dass über die

[26] http://www.microsoft.com/downloads/details.aspx?displaylang=de&FamilyID=56e5b1c5-bf17-42e0-a410-371a838e570a

[27] http://www.microsoft.com/downloads/details.aspx?displaylang=de&FamilyID=df0ba5aa-b4bd-4705-aa0a-b477ba72a9cb

GUI einzelne Objekte aus der Datenbank markiert werden können, für welche letztlich ein Skript erzeugt werden soll. In der Hilfe von DPW taucht `-include` als Option für die Kommandozeilenvariante auf, diese Option findet sich aber an keiner weiteren Stelle in der Hilfe wieder und erzeugt auch bei Verwendung an der Kommandozeile einen Fehler mit dem Hinweis, dass dies eine unbekannte Option sei. Somit besteht über die Kommandozeile nur die Möglichkeit, alle Schemaobjekt der Datenbank auf einmal auszulesen. Dies ist nicht schön und erhöht sicherlich die Laufzeit des Prozesses geringfügig, ist aber für einen weiteren Einsatz nicht unbedingt hinderlich. Ein Aufruf des Tools auf der Kommandozeile zum Auslesen des Schemas einer Datenbank `AdventureWorks` über eine vertrauenswürdige Verbindung und Abspeichern in einer Datei sieht wie folgt aus:

```
"C:\Programme\Microsoft SQL Server\90\Tools\Publishing\SqlPubWiz.exe"
      script -f -q -d AdventureWorks -schemaonly -nodropexisting
      C:\AdventureWorks.sql
```

Bei weiteren Experimenten mit diesem Tool stößt man an eine Grenze, sobald sich in der Datenbank ein verschlüsseltes Objekt befindet. Verschlüsselte Objekte können sein:

- Views
- Trigger
- Prozeduren

Diese sind zwar innerhalb von SQL Server ausführbar, sollen aber verhindern, dass Dritte Einblick in deren Aufbau erhalten. Mit dieser Tatsache kann auch DPW nicht umgehen, wirft einen entsprechenden Fehler auf der Kommandozeile und bricht die Ausführung ab. Ungewollter Nebeneffekt dieses Verhaltens ist, dass DPW in diesem Fall vollständig abbricht und keinerlei Skripte zurückgibt. Es existiert auch keine weitere Option für dieses Tool, um zu veranlassen, dass verschlüsselte Objekte ignoriert werden sollen. Wenn DPW also weitere Anwendung in dieser Arbeit findet, dann bedingt ein Abbruch aufgrund vorhandener verschlüsselter Objekte auch einen kompletten Abbruch von **TSQL-Backup**. Ein Beispiel, der einen Abbruch bedingen würde, zeigt folgende Ausgabe von DPW:

```
Microsoft (R) SQL Server Datenbankveröffentlichungs-Assistent
Version (1.1.1.0)
Copyright (c) Microsoft Corporation. Alle Rechte vorbehalten.

Skript für die AdventureWorks-Datenbank wird erstellt
- Nur Metadatenskript generieren
- Skript für SQL Server '2005' generieren
Fehler: 'sp_encrypted_procedure' ist eine verschlüsselte gespeicherte
```

```
Prozedur. Die Skripterstellung für verschlüsselte gespeicherte Prozeduren
wird nicht unterstützt.
```

Im Internet finden sich Hinweise, dass diese verschlüsselten Objekte nicht tatsächlich verschlüsselt sind, sondern aufgrund einer Nachlässigkeit von Microsoft nur „verfremdet" werden[28,29] und wieder zu entschlüsseln sind[30]. Dabei wird eine Prozedur über verschiedene XOR-Befehle unlesbar gemacht. Weiterhin kursieren wenige kommerzielle Produkte, wie „DecryptSQL"[31] von Devlib Inc. und „SQL Decryptor"[32] von DMT Software Inc., sowie freie SQL Skripte[33], welche ein vorhandenes verschlüsseltes Objekt entschlüsseln sollen. Meist sind diese freien Skripte aber an SQL Server 2000 gebunden und funktionieren oft nur bei Prozeduren. Nur kommerzielle Produkte, wie die genannten, werben mit der Fähigkeit, auch Objekte für SQL Server 2005 und 2008 entschlüsseln zu können.

Da es keine verlässliche Freeware gibt und die verfügbaren freien SQL Skripte nicht in jedem Fall fehlerfrei funktionieren oder Einschränkungen aufweisen (beispielsweise bei der Größe von Datentypen haben[33]), muss der Aspekt des Abbruchs bei verschlüsselten Objekten als de facto angenommen werden.

SMO

Eine weitere Möglichkeit, um das Schema befehlszeilenorientiert auszulesen, besteht über SQL Server Management Objects (weiterhin SMO). SMO ist eine Bibliothek des Microsoft .NET-Frameworks[34], mit der es möglich ist, SQL Server zu verwalten und eine Verbindung mit SQL Server aufzubauen, um somit DML- wie auch DDL-Befehle ausführen zu können. Um SMO im Rahmen dieser Arbeit verwenden zu können, wird „Windows PowerShell 1.0"[35] (weiterhin PowerShell) als Skriptsprache herangezogen. SMO lässt sich aus sehr vielen

[28] http://blog.sqlauthority.com/2007/07/01/sql-server-explanation-of-with-encryption-clause-for-stored-procedure-and-user-defined-functions/#comment-38868

[29] http://www.derkeiler.com/Newsgroups/microsoft.public.sqlserver.security/2006-08/msg00287.html

[30] http://www.mssqltips.com/tip.asp?tip=1046

[31] http://www.devlib.net/decryptsql.htm

[32] http://www.sql-tools.net/sqldecryptor/

[33] http://www.planet-source-code.com/vb/scripts/ShowCode.asp?txtCodeId=505&lngWId=5

[34] http://www.microsoft.com/downloads/details.aspx?familyid=10CC340B-F857-4A14-83F5-25634C3BF043&displaylang=de

[35] http://www.microsoft.com/downloads/details.aspx?familyid=C61FB27B-E71C-4ECF-9D2C-9B299B149490&displaylang=de

Interpreter- und Programmiersprachen verwenden. Für diesen Zweck reicht die PowerShell aber aus.

Fazit:

Für diese Arbeit wird SMO nicht zum Einsatz kommen, da hierfür eine neue Skriptsprache angewandt werden muss, bei der der Aufwand zur Einarbeitung dem Nutzen überwiegt. Somit wird DPW als Möglichkeit zum Auslesen des Datenbankschemas genutzt.

Um bei der weiteren Entwicklung eventuell auftretende Probleme schneller lösen zu können, seien hier die wichtigsten bekannten Probleme und Eigenarten von DPW aufgeführt [Mic07]:

- Datenbanklogins werden in Rollen und ggf. Schemas umgewandelt
- Rollen, welche anderen Rollen zugeordnet sind, können nicht verarbeitet werden
- Es können Probleme auftreten, wenn eine durch DPW festgelegte Kollation einer Textspalte nicht mit der Kollation der Zieldatenbank übereinstimmt

Datenbanknutzer in SQL Server an zwei Stellen im Server gespeichert, bei einer Wiederherstellung durch die Funktionalitäten von SQL Server selbst wird die interne Verbindung dieser zwei gespeicherten Nutzerdaten getrennt, so dass an dieser Stelle eine Korrektur erfolgen muss, um die Verbindung wiederherzustellen [Woo07 S. 144]. Aufgrund der Konvertierung von Datenbanklogins in Rollen ist zu abzusehen, dass bei der Wiederherstellung einer Datenbank mit umfassendem Nutzerkonzept eine nachträgliche Korrektur durch einen Administrator notwendig sein kann.

6.4. Untersuchung zur eindeutigen Kennzeichnung von Daten

Da differenzielle und inkrementelle Backups vorgesehen sind, muss ein Weg gefunden werden, eine Datenzeile eindeutig zu kennzeichnen. Tabellen sind die Träger von Daten, daher müssen als Schemaobjekte für den Prozess der Datenextraktion aus der Datenbank auch nur Tabellen in Betracht gezogen werden.

Eine Überführung von Relationen in Normalformen bedingt die Einführung von Schlüsseln, somit kann allgemein gesagt werden, dass Relationen in mindestens der dritten Normalform eindeutige Datenzeilen enthalten. Davon ausgehend, dass aber nicht immer Schlüsselbeziehungen oder eindeutige Indizes in einer Datenbank existieren, ein klassischer

Fall hierfür ist häufig ein Data Warehouse, oder auch das Vorliegen der dritten Normalform nicht immer gewünscht ist, muss ein anderer Weg gefunden werden.

SQL Server bietet die Systemfunktion `CHECKSUM` an. Diese wurde entwickelt, um damit Hashwerte für Indizes zu erstellen. Schon bei den ersten Versuchen, wird klar, dass diese Funktion ungeeignet ist, da sie für unterschiedliche Zeichenketten die gleiche Prüfsumme liefert. Beispielsweise liefern folgende SQL-Befehle stets eine Prüfsumme von 142:

```
SELECT CHECKSUM('a')
SELECT CHECKSUM('A')
SELECT CHECKSUM('a ')
SELECT CHECKSUM('A ')
```

Damit ist ersichtlich, dass bereits Änderungen an der Groß- und Kleinschreibung in der Datenzeile verloren gehen würden. Selbst Microsoft erläutert in seiner Dokumentation, dass für die Zwecke einer eindeutigen Kennzeichnung die Funktion `HashBytes` geeigneter ist [Mic0813]. Diese Funktion erwartet als ersten Parameter einen Hashalgorithmus und als zweiten Parameter eine Zeichenkette vom Typ `VARBINARY` mit einer Länge von maximal 8000 Zeichen. Als Hashalgorithmen werden MD2, MD4, MD5, SHA und SHA-1 verstanden. SHA-1 erzeugt einen 160-Bit-Hashwert und gilt als sicherste der genannten Methoden. Auch wenn MD5 bereits aufgrund gefundener Kollisionen als unsicher gilt[36], sollte dessen Anwendung dennoch ausreichen. Letztlich bedeuten sicherere Verschlüsselungsalgorithmen auch mehr verbrauchte Rechenzeit. Da es sich hierbei aber nur um einen Parameter einer Funktion handelt, der leicht austauschbar ist, sollte sich zu jeder Zeit der Algorithmus ändern lassen. Zu beachten wäre dann aber, dass bereits erstellte differenzielle und inkrementelle Backups nach einer Umstellung des Hashalgorithmus nicht mehr einsetzbar sind.

Tabellen enthalten meist mehrere Spalten. Um einen eindeutigen Hashwert zu erhalten, sollte man über jede einzelne Spalte der Tabelle einen Hashwert bilden, in der Reihenfolge, in der die Spalten in der Tabelle auch angeordnet sind und diese einzelnen Hashwerte dann auch in dieser Reihenfolge zu einer langen Zeichenkette verbinden. Durch Beachtung der Spaltenreihenfolge und das Nacheinanderverketten der Hashwerte kann eine Kollision von Hashwerten für unterschiedliche Datenzeilen nahezu ausgeschlossen werden. Hierbei tritt aber ein Nebeneffekt auf. Ohne Schlüssel in Tabellen, können Datenzeilen in einer Tabelle

[36] MD5 wurde 1991 als sicherer Nachfolger von MD4 entwickelt. Bereits 1996 fand Hans Dobbertin zwei unterschiedliche Nachrichten, die bei MD5-Verschlüsselung den gleichen Hashwert ergaben. Technischer Hintergrund ist, dass der Schlüsselwert eines Textes T gleich dem Schlüsselwert eines anderen Textes T‘ entspricht, man sucht also zwei Texte T und T‘ welche diese Bedingung erfüllen. MD5 gilt somit als theoretisch geknackt.

identisch mehrfach vorkommen. Die gefundene Verfahrensweise würde diese Datenzeilen nur ein einziges Mal erfassen, da sie identisch sind. Inwiefern sich hieraus ein Nachteil ergibt, wird sich in der weiteren Arbeit zeigen.

6.5. Datenkonsistenz während eines Sicherungsvorgangs

Für **TSQL-Backup** ist es bei Sicherungsvorgängen wichtig, stets einen konsistenten und sich nicht mehr ändernden Datenbestand vorzufinden, mit dem gearbeitet werden kann. Bereits in SQL Server 2005 gibt es die Möglichkeit echter Snapshots von Datenbanken, diese Möglichkeit ist aber nur der Enterprise Edition vorbehalten und scheidet daher hier aus [Kon07 S. 409]. Bei einem echten Snapshot wird von der betroffenen Datenbank ein Abbild erstellt, auf welchem gearbeitet werden kann.

Daher muss auf Sperren und Isolationsstufen zurückgegriffen werden. Prinzipiell zu unterscheiden ist dabei, dass SQL Server in der Regel selbstständig Sperren auf Objekten vergibt, durch den Nutzer aber explizit Sperren auf Tabellenebene aber auch die Aktivierung von verschiedenen Isolationsstufen für Transaktionen vorgenommen werden können. Die internen Sperrtypen von SQL Server können sich auf verschiedenen Ebenen erstrecken, zu den Wichtigen gehören [Bau06 S. 388f]:

- Zeilensperre („RID“)
- Seitensperre („PAGE“)
- Tabellensperre („TABLE“)
- Dateisperren („FILE“)
- Datenbanksperre („DATABASE“)

Diese Sperren werden unter anderem je nach Anforderung, Anzahl zeitgleicher Transaktionen auf ein Objekt, vorhandener Ressourcen und Transaktionsisolationsstufe durch SQL Server automatisch gesetzt. Dabei gibt es grundlegend 4 Arten von Sperren [Bau06 S. 389f]:

- gemeinsame Sperre – ermöglicht mehreren Transaktionen ein zeitgleiches Lesen von Daten
- exklusive Sperre – wird bei `INSERT`-, `UPDATE`- oder `DELETE`-Vorgängen aktiviert
- beabsichtigte Sperre – wird intern für das Sperrmanagement durch SQL Server verwendet
- Aktualisierungssperre – ist eine Mischung aus gemeinsamer und exklusiver Sperre

Dabei werden Sperren aktiviert, sobald eine Transaktion gestartet wird. Dabei gilt das „Alles-oder-nichts-Prinzip“. Das heißt, dass eine Transaktion, egal wie viele Befehle darin ausgeführt werden, erfolgreich abgeschlossen, oder bei einem Fehler vollständig durch ein Rollback rückabgewickelt wird [Lei03 S. 322f]. Der technische Hintergrund dabei ist, dass SQL Server alle Änderungen einer Transaktion erst im Speicher ausführt und nur bei Erfolg sowie einem `COMMIT` diese Änderungen auch auf der Festplatte schreibt [Hen04 S. 501f].

Von Interesse hingegen sind die Isolationsstufen für Transaktionen. Mit ihnen kann man steuern, wie isoliert voneinander verschiedene Transaktionen ablaufen dürfen. Folgende Stufen (aufsteigend nach Restriktion) sind in SQL Server implementiert [Bau06 S. 391ff] [Hen04 S. 508ff]:

Isolationsstufe	Dirty Reads	Nichtwieder-holbare Lese-vorgänge	Phantom-werte	Merkmale
`READ UNCOMMITTED`	ja	ja	ja	keinerlei Benutzung von Sperren
`READ COMMITTED`	nein	ja	ja	Standardeinstellung von SQL Server; gemeinsame Sperren beim Lesen; verhindert Dirty Reads
`REPEATABLE READ`	nein	nein	ja	gemeinsame Sperren beim Lesen; verhindert nichtwiederholbare Lesevorgänge
`SNAPSHOT`	nein	nein	nein	Schnappschuss auf Sicht der Daten, beim Beginn der Transaktion; verhindert Phantomwerte
`SERIALIZABLE`	nein	nein	nein	sehr restriktiv, blockiert alle anderen Aktualisierungs- oder Einfügevorgänge

Tabelle 7: Isolationsstufen in SQL Server

Aus der Tabelle ist ersichtlich, welche unangenehmen Nebeneffekte bei der Verwendung der Isolationsstufen auftreten können. Es ist nun nicht ratsam, immer die Isolationsstufe mit der höchsten Restriktion einzusetzen, da hier immer abgewägt werden muss, inwiefern parallele Transaktionen eingeschränkt beziehungsweise behindert werden.

Die Snapshot-Isolationsstufe ist neu in SQL Server 2005 [Bau06 S. 23]. Hierbei werden geänderte Daten durch andere Transaktionen in der `tempdb`-Datenbank vorgehalten, bis die Transaktion per `COMMIT` oder `ROLLBACK` beendet wird und auch die Snapshot-Transaktion beendet ist [Bau06 S. 392]. Die Stufe ist dabei so flexibel, dass eine konsistente Datensicht gewährt wird, aber auch in gewissem Maße Änderungen an Daten durch andere

Transaktionen möglich sind. Sie ist daher nicht so restriktiv wie `SERIALIZABLE` und soll im Rahmen dieser Arbeit Verwendung finden.

Zur Benutzung der Snapshot-Isolationsstufe muss in SQL Server eine administrative Option gesetzt werden:

```
ALTER DATABASE {name} SET ALLOW_SNAPSHOT_ISOLATION ON
```

Mit dieser gesetzten Option ist es möglich, eine Snapshot-Transaktion zu starten. Wichtig hierbei zu wissen: in allen Datenbanken, die in einer Snapshot-Transaktion beteiligt sind, muss diese Option aktiviert sein – also nicht nur in der zu sichernden Datenbank, sondern auch in der Metadatenbank[37].

Eine Snapshot-Transaktion wird in TSQL dann wie folgt umgesetzt:

```
SET TRANSACTION ISOLATION LEVEL SNAPSHOT
BEGIN TRANSACTION
      […]
COMMIT TRANSACTION
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
```

Listing 4: Arbeiten mit einer Transaktions-Isolationsstufe

Mit der abschließenden Anweisung wird der Transaktionslevel wieder auf den Standardwert zurückgesetzt (dieser kann natürlich abweichen, insofern durch den Nutzer eine andere Einstellung in SQL Server vorgenommen wurde).

6.6. Datenkonsistenz bei der Wiederherstellung

Abschnitt 6.2 hat gezeigt, dass das Auslesen des Transaktionsprotokolls kompliziert ist und im Rahmen dieser Arbeit nicht für die Konsistenzwahrung verwendet werden kann. Daher muss **TSQL-Backup** eigenständig die Konsistenz sicherstellen. Auf Grundlage der Integritätsfaktoren aus dem Kapitel 4 können folgende Ansätze formuliert werden:

[37] Ein Versuch, Daten zwischen zwei Datenbanken zu verknüpfen, bei denen nur für eine der Snapshot-Isolationsmodus aktiviert war, scheiterte (der Befehl lief endlos), da entsprechende Ressourcen nicht gesperrt werden konnten [SQL10]

CHECK-Einschränkungen:

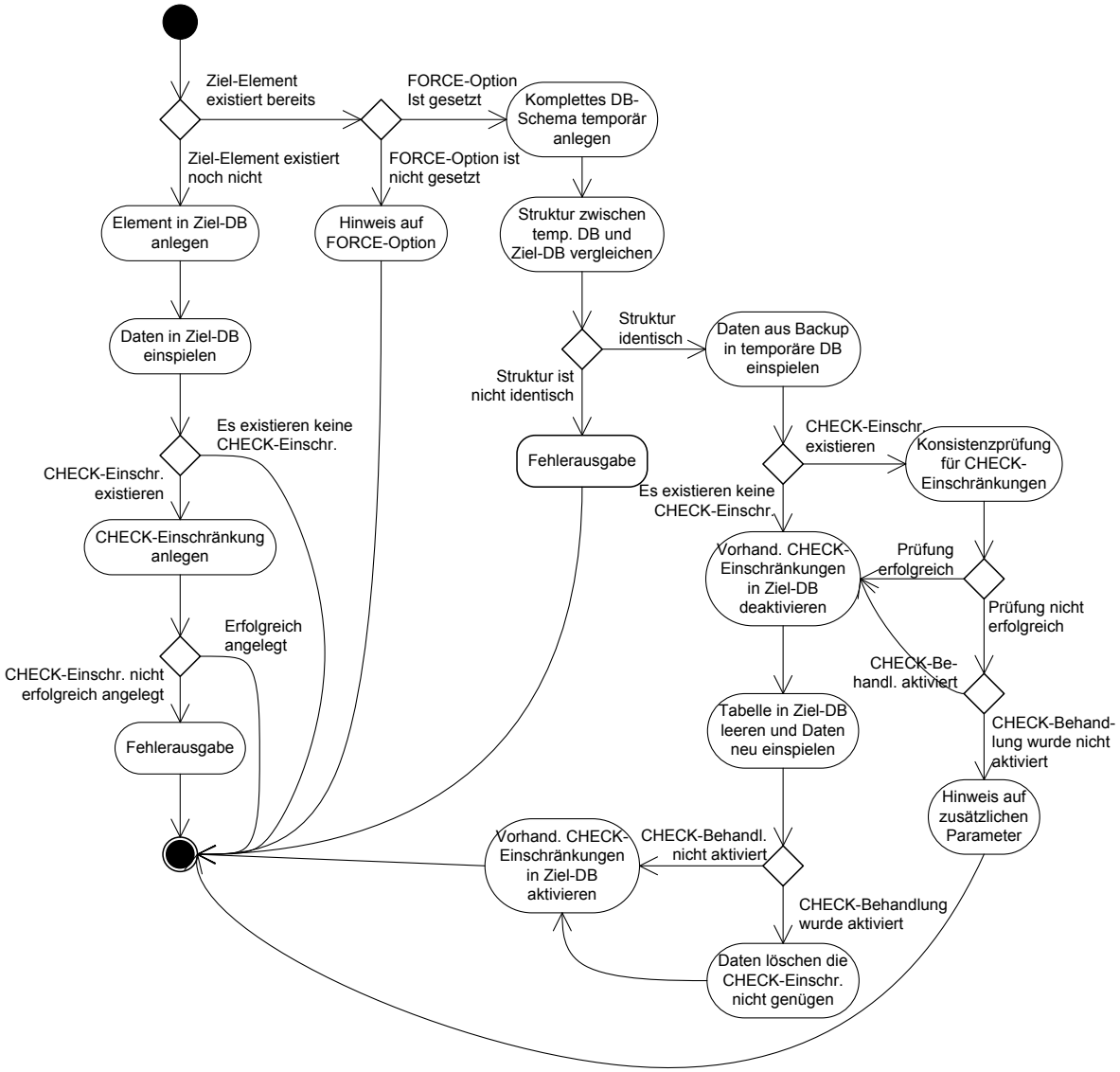

Abbildung 10: CHECK-Einschränkung bei der Wiederherstellung

Der einfachste Anwendungsfall besteht in der Wiederherstellung einer noch nicht vorhandenen Tabelle. Für den Fall einer existierenden Tabelle müssen Strukturvergleiche vorgenommen werden. Der Strukturvergleich zwischen vorhandener Tabelle und der gesicherten Tabelle im Backup kann nur über eine temporär wiederhergestellte Kopie aus dem Backup geschehen. Dabei müssen die Anzahl der Spalten, die Reihenfolge der Spalten und

die Spaltentypen übereinstimmen. Weiterhin müssen beide Tabellen auf inhaltliche Übereinstimmung der existierenden CHECK-Bedingungen geprüft werden.

Eine Konsistenzprüfung bei Vorhandensein von CHECK-Einschränkungen kann in diesem Fall auf einfache Weise erfolgen. Es reicht die Gesamtanzahl Zeilen im Backup mit der Anzahl Zeilen zu vergleichen, die der CHECK-Einschränkung genügen. Hierzu muss diese Einschränkung in die `WHERE`-Klausel der Abfrage aufgenommen werden, welche die Anzahl zutreffender Datensätze zählt. Weichen beide Werte voneinander ab, so gehen Daten bei der Wiederherstellung unter Umständen verloren. Der Nutzer soll hier die Möglichkeit haben, die CHECK-Einschränkungen zu deaktivieren, um wirklich alle Daten aus dem Backup wiederherstellen zu können und als Alternative sollen die Daten bei aktivierter Einschränkung mit dem Risiko des Datenverlustes eingespielt werden.

Trigger:

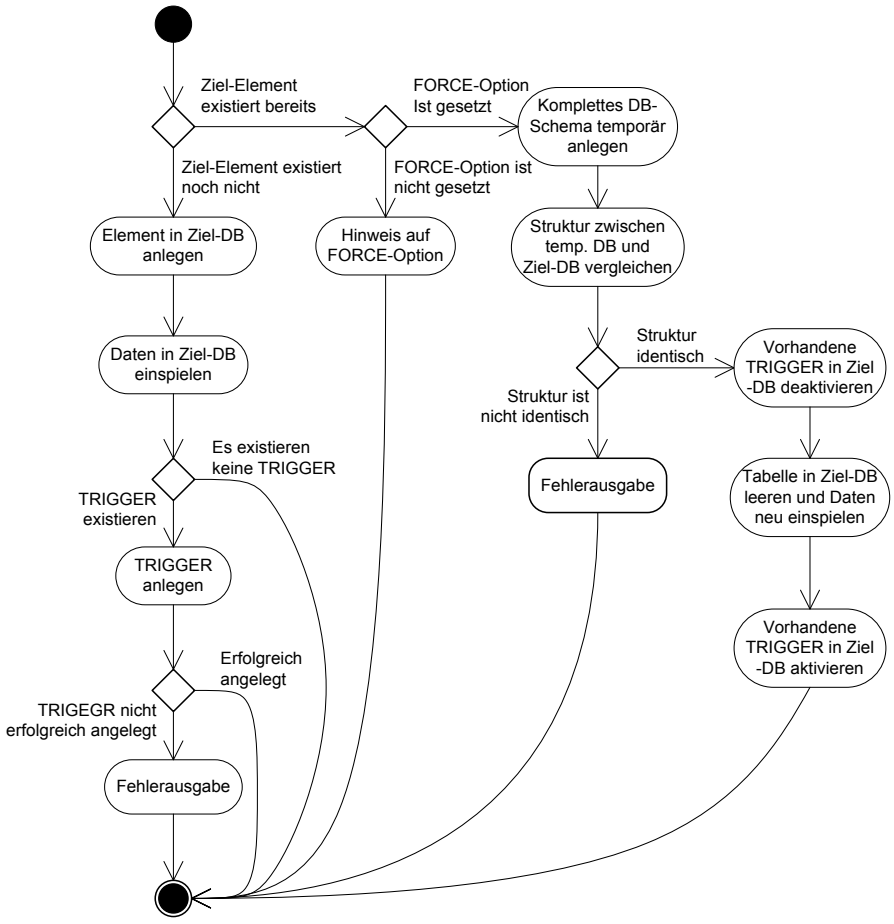

Abbildung 11: Wiederherstellung mit vorhandenen Triggern

Auch hier ist ersichtlich, dass für den Fall einer nicht existierenden Tabelle keine besonderen Prüfungen stattfinden müssen. Andernfalls muss wieder ein Strukturvergleich zwischen bestehender Tabelle und existierender Tabelle durchgeführt werden. Wie in Abschnitt 4.4 erläutert, ist es im Rahmen dieser Arbeit nicht möglich, die Funktionsweise eines Triggers für das Programm verständlich zu machen, daher kann bei vorhandenen Triggern nur derart reagiert werden, dass diese Trigger vor dem Einspielen von Daten deaktiviert und anschließend wieder aktiviert werden.

Fremdschlüssel:

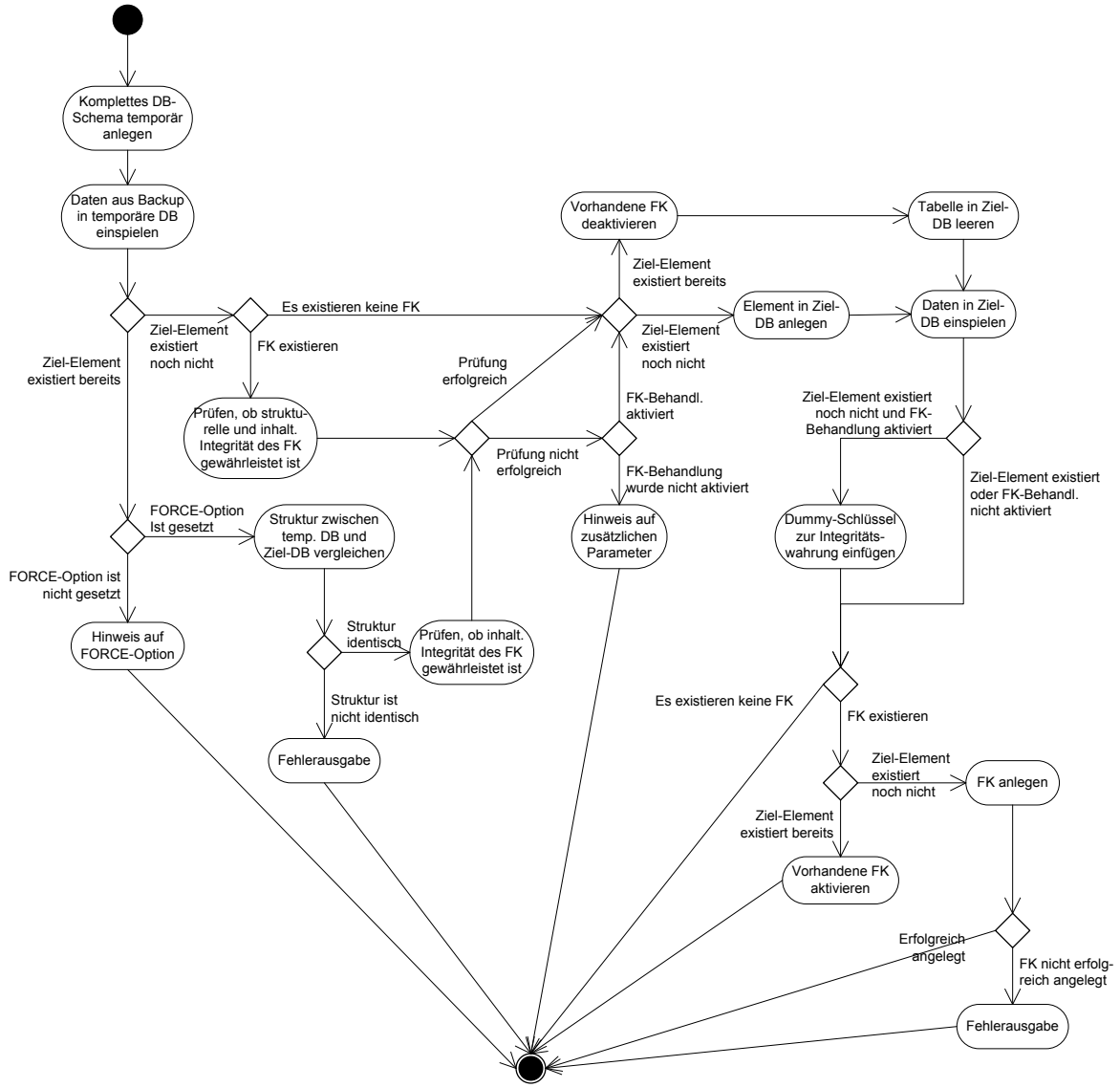

Abbildung 12: Wiederherstellung bei vorhandenen Schlüsselbeziehungen

Bei Vorhandensein von Schlüsselbeziehungen muss zusätzlich zur Strukturprüfung der Tabelle auch eine strukturelle und inhaltliche Prüfung der beteiligten Schlüssel stattfinden. Ist die Zieltabelle bereits vorhanden, muss geprüft werden, ob der Aufbau des Schlüssels im Backup, bezüglich der verwendeten Spalten und Spaltentypen, dem Aufbau des Schlüssels in der Zieltabelle entspricht. Existiert die Zieltabelle nicht, muss geprüft werden, ob die verwendeten Spalten im Schlüssel in der referenzierten Tabelle existieren und vom Datentyp übereinstimmen. Diese Strukturprüfung muss ebenso für Fremdschlüssel stattfinden, die auf anderen Tabellen liegen, aber auf die wiederherzustellende Tabelle verweisen, da hier zwischen Sicherung und Wiederherstellung Änderungen am Schema geschehen sein könnten.

In jedem Fall muss nachfolgend untersucht werden, ob es inhaltliche Differenzen gibt. Das heißt, handelt es sich um einen Fremdschlüssel, der auf der wiederherzustellenden Tabelle angelegt wurde und auf eine andere Tabelle verweist, dürfen die Schlüsselspalten der wiederherzustellenden Tabelle nur Werte annehmen, die auch in der referenzierten Tabelle enthalten sind. Insofern diese Inkonsistenzen durch **TSQL-Backup** korrigiert werden sollen, kann nur derart reagiert werden, dass die Datenzeilen, welche die Inkonsistenz hervorrufen, in der wiederherzustellenden Tabelle gelöscht werden.

Weiterhin muss beachtet werden, dass auf anderen Tabellen Fremdschlüssel existieren können, welche auf die wiederherzustellende Tabelle verweisen. Es kann der Fall eintreten, dass in den referenzierenden Tabellen Daten enthalten sind, welche in der wiederherzustellenden Tabelle nicht mehr (oder noch nicht) existieren. An dieser Stelle könnte eine Korrektur durch **TSQL-Backup** so geschehen, dass Dummy-Werte in der wiederherzustellenden Tabelle angelegt werden, um die Konsistenz zu gewähren. Dieses Vorgehen ist aber sehr kompliziert, da Schlüssel auch aus mehreren Spalten bestehen (es müssen also die exakten Datentypen dieser Spalten berücksichtigt werden) und auf diesen Spalten weitere CHECK-Einschränkungen existieren können.

Eine Aktivierung vorhandener CHECK-Einschränkungen, Trigger oder Schlüsselbeziehungen nach dem Einspielen von Daten auf der wiederhergestellten Tabelle könnte selbst dann durch **TSQL-Backup** automatisch erfolgen, wenn der Nutzer die Konsistenzbehandlung durch das Programm nicht aktiviert hat. Hierbei kann lediglich der Fall eintreten, dass eine Reaktivierung dieser Integritätsfaktoren aufgrund einer Verletzung der Integritätsbedingung nicht möglich ist.

6.7. Einbindung von externen Hilfstools

TSQL bietet nur einen begrenzten Sprach- und Funktionsumfang. Um insbesondere die Arbeit mit Dateien zu ermöglichen, muss auf die Windows-Kommandozeile ausgewichen werden. Dieser Weg kann erforderlich sein für:

- Auflisten von Dateien in Verzeichnissen
- Komprimierung/Dekomprimierung von Dateien („gzip“)
- Arbeit mit DPW
- Export/Import von Dateien aus/in SQL Server

In TSQL steht hierzu die Systemfunktion `xp_cmdshell` zur Verfügung [Mic0814]. Diese Funktion arbeitet synchron, wartet also auf die Beendigung der Kommandozeile und kann die Ausgabe an der Kommandozeile an TSQL zurückgeben.

Standardmäßig ist diese Funktion aus Sicherheitsgründen deaktiviert und muss per administrativem Eingriff aktiviert werden. Dazu ist folgendes Vorgehen notwendig [Bau06 S. 67f]:

```
EXEC sp_configure 'show advanced options',1
GO
RECONFIGURE
GO
EXEC sp_configure 'xp_cmdshell',1
GO
RECONFIGURE
GO
EXEC sp_configure 'show advanced options',0
```

Listing 5: Aktivieren von xp_cmdshell

6.8. Datenmodell

Zur Verwaltung von Einstellungen, sowie der Informationen über Sicherungen und Wiederherstellungen und zum Vorhalten der Logbücher wird eine Datenbank benötigt.

Nachfolgend sollen, auf Grundlage von Abschnitt 5.2.5, die Tabellen definiert werden, welche Metadaten enthalten:

Tabelle: **logs** (enthält das Logbuch)

Spalte	Typ-Definition	Beschreibung
log_id	INT NOT NULL	ID des erzeugten Logs zum Programmaufruf
entry_id	INT IDENTITY(1,1) NOT NULL	eindeutige ID einer Logbuch-Zeile
date	DATETIME NOT NULL	Datum des Logbucheintrags
log_text	VARCHAR(MAX) NULL	Log-Inhalt

Tabelle: **hashs_XYZ** (enthält berechnete Hashwerte für gesicherte Tabellen)

Spalte	Typ-Definition	Beschreibung
hash	VARBINARY(MAX) NOT NULL	Hashwert der Datenzeile
delflag	BIT NOT NULL DEFAULT (0)	Flag, das anzeigt, ob Datenzeile gelöscht wurde (bei diff./inkl. Backups wichtig)

Tabelle: **objects** (enthält Grundinformationen gesicherter Objekte)

Spalte	Typ-Definition	Beschreibung
object_id	INT IDENTITY(1,1) NOT NULL	eindeutige ID eines gesicherten Objektes
type	VARCHAR(255) NOT NULL	Typ des Objektes (aus Tabelle schema_lookup)
name	VARCHAR(255) NOT NULL	Name des Objektes in der Datenbank

Tabelle: **objects_detail** (enthält Detailinformationen zu den Objekten)

Spalte	Typ-Definition	Beschreibung
object_id	INT NOT NULL	Objekt-ID
owner	VARCHAR(255) NOT NULL	Besitzer des Objektes in der Datenbank
create_script	TEXT NOT NULL	SQL-Skript zur Wiedererzeugung des Objektes
compressed	BIT NOT NULL DEFAULT (0)	Flag, das anzeigt, ob die exportierten Daten komprimiert wurden (nur bei Tabellen)

Tabelle: **backups** (enthält Informationen über Sicherungsvorgänge)

Spalte	Typ-Definition	Beschreibung
backup_id	INT IDENTITY(1,1) NOT NULL	eindeutige ID des ausgeführten Backups
type	VARCHAR(255) NOT NULL	Typ des Backups (vollständig/diff./inkr./Schemasicherung)
db_name	VARCHAR(255) NOT NULL	Name der gesicherten Datenbank
date	DATETIME NOT NULL	Datum der Sicherung
description	VARCHAR(8000) NULL	Beschreibung des Backups
state	VARCHAR(20) NOT NULL DEFAULT ('not_finished')	Status des Backups: finished = ordnungsgemäß abgeschlossen not_finished = „kaputtes“ Backup
hashs_exist	BIT NOT NULL DEFAULT (0)	Flag, das anzeigt, ob zu diesem Backup Hashwerte berechnet wurden
parent_id	INT NOT NULL DEFAULT (0)	ID eines übergeordneten Backups (bei inkr. Backups das letzte inkr., bei diff. Backups das letzte vollständige Backup)
master_id	INT NOT NULL DEFAULT (0)	ID des Hauptbackups (bei diff./inkr. Backups das letzte vollständige Backup)
log_id	INT NOT NULL DEFAULT (0)	ID des zugehörigen Logbuchs

Tabelle: **backup_objects** (Zuordnungstabelle von Objekten zu Backups)

Spalte	Typ-Definition	Beschreibung
backup_id	INT NOT NULL	ID des Backups
object_id	INT NOT NULL	IDs der Objekte, die durch das Backup erfasst wurden

Tabelle: **schema_lookup** (Referenztabelle für Schemaobjekte)

Spalte	Typ-Definition	Beschreibung
type	VARCHAR(255) NOT NULL	Kürzel für Objekttyp
type_desc	VARCHAR(255) NOT NULL	Objekttyp, wie er aus DPW erzeugt wird
user_desc	VARCHAR(255) NOT NULL	Alternativtext für die Ausgabe

Tabelle: **settings** (enthält Voreinstellungen)

Spalte	Typ-Definition	Beschreibung
setting_name	VARCHAR(255) NOT NULL	Parameter der Voreinstellung
setting_value	VARCHAR(255) NOT NULL	Wert zu diesem Parameter

Tabelle: **allowed_arguments** (enthält die zulässige Syntax und bildet eine Hierarchie ab)

Spalte	Typ-Definition	Beschreibung
argument_id	INT IDENTITY(1,1) NOT NULL	eindeutige ID einer Zeile in dieser Tabelle
param	VARCHAR(255) NULL	erlaubter Parameter für Programmaufruf
value	VARCHAR(255) NULL	erlaubter Wert für Parameter
type	VARCHAR(25) NOT NULL	zulässige Wertebereiche für übergebene Werte ("string", "int", "{a, b ,c}")
depends_on	INT NULL	ID des übergeordneten Elementes (nötig für zusätzliche Optionen für eine andere Option)
needs_child	BIT NOT NULL	Flag, das kennzeichnet, ob dieser Parameter weitere Optionen erwartet
must_be_given	BIT NOT NULL	Flag, das kennzeichnet, ob dieser Parameter beim Aufruf des Programms angegeben werden muss

Aus diesen Tabellen ergibt sich folgendes Datenbankschema:

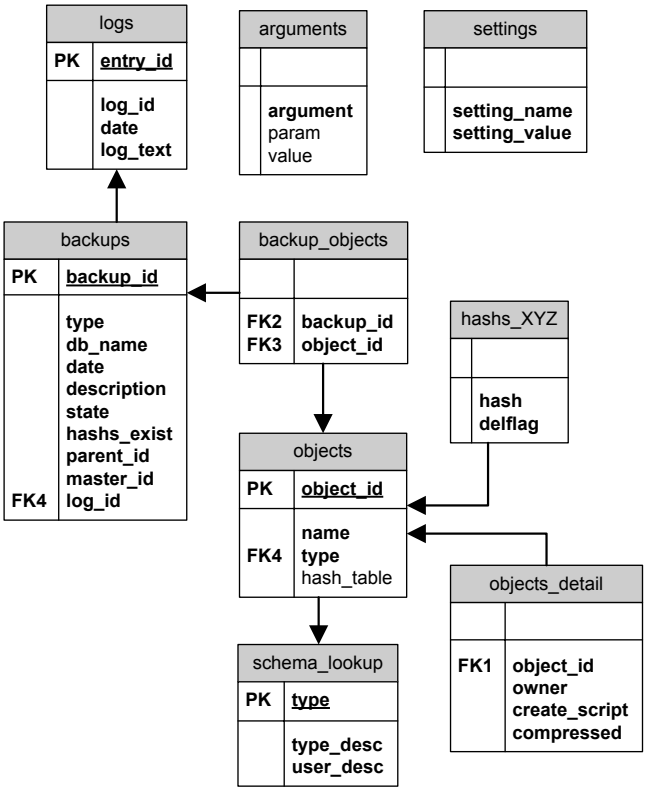

Abbildung 13: Entity-Relationship-Model der Metadatenbank

Zu jeder Datentabelle, die durch **TSQL-Backup** erfasst wird, wird eine Hashtabelle erstellt. Der Tabellenname dieser Hashtabelle enthält dabei die durch das Programm erstellte eindeutige `object_id` der erfassten Tabelle. Da bei großen, zu sichernden Datenbanktabellen oder häufiger Nutzung des Backupprogramms schnell große Datenbestände in einer einzigen Hashtabelle auflaufen würden, sollte eine Trennung (auch „Partitionierung“) durchgeführt werden. Durch die Trennung wird ein Performanzgewinn erreicht.

6.9. Einsatz von Prozeduren oder Funktionen

TSQL ist eine prozedurale Skriptsprache. Eine objektorientierte Entwicklung ist somit leider nicht möglich. Zur besseren Wiederverwendbarkeit und zur Übersichtlichkeit lassen sich aber bestimmte Funktionalitäten als Prozedur verfassen. TSQL unterstützt sowohl Funktionen als auch Prozeduren.

Nachfolgend eine Übersicht der Unterschiede beider Varianten:

Eigenschaft	Funktion	Prozedur
Parameterrückgabe	Rückgabe direkt an Tabelle oder `SELECT`-Befehl	Rückgabe an das aufrufende Programm
DML-Aktionen	nein	ja
DDL-Aktionen	nein	ja
Änderung von Umgebungsvariablen	nein	ja
Verwendung als Datenquelle in `SELECT`-Befehl (`SELECT * FROM function/procedure`)	ja	ja
Fehlerbehandlung	Abbruch bei Auftreten eines Fehlers	Fehler können bis zu einem bestimmten Schweregrad abgefangen werden
Aufruf	Aufruf meist nur in Verbindung mit einem `SELECT`-Befehl	Aufruf durch `EXEC`-Befehl, somit eigenständiges Unterprogramm möglich

Tabelle 8: Unterschiede zwischen Funktionen und Prozeduren in TSQL

Aufgrund der flexiblen Einsatzmöglichkeiten werden Unterprogramme und ausgelagerte Funktionalitäten fast ausschließlich als Prozeduren umgesetzt. Ausschlaggebend für diese Entscheidung ist die fehlende Unterstützung von DML- und DDL-Aktionen in Funktionen. Für die Berechnung von Hashwerten wird eine Funktion zum Einsatz kommen, da die Rückgabe dieser Funktion direkt in einer `SELECT`-Anweisung weiter verarbeitet wird und somit der Einsatz einer Prozedur für diesen Zweck ausscheidet.

Parameterübergaben lassen sich in TSQL auch über temporären Tabellen lösen. Dabei können zur Laufzeit Tabellen durch das SQL Skript erstellt werden, die bei Beendigung des Skripts automatisch wieder gelöscht werden. Solche Tabellen können entweder lokal für das aktuell ausgeführte Skript oder global[38], also auch übergreifend für andere Prozeduren innerhalb einer SQL-Sitzung, definiert werden. Hierbei geht aber die Übersichtlichkeit verloren, daher sollen vorrangig echte Parameterübergaben bei Prozeduren stattfinden.

[38] Hierbei kann es unerwartet zu Namenskollisionen kommen, wenn bereits in anderen Prozeduren oder Prozessen globale temporäre Tabellen verwendet werden, daher sollte deren Einsatz nur bei Notwendigkeit erfolgen [SQL101]

6.10. Abhängigkeitsdiagramm für Prozeduren

An dieser Stelle soll auf Basis der bisherigen Erkenntnisse eine Übersicht gegeben werden, wie **TSQL-Backup** aufgebaut ist. Bestimmte Funktionen innerhalb des Programms lassen sich in Prozeduren oder Funktionen kapseln, da sie sich als eigenständiges Unterprogramm darstellen lassen, so die Übersichtlichkeit erhöhen und eine Wiederverwendbarkeit erreichen. Die verwendeten Prozeduren lehnen sich an die geforderten Produktfunktionen aus Abschnitt 5.2.4 an. Nachfolgend sollen diese Prozeduren und deren Verknüpfung durch eine UML-Notation verdeutlicht werden. Parametern, die an diese Prozedur übergeben werden müssen, ist ein „+“ (Plus) vorangestellt. Rückgabeparameter der Prozedur sind an einem „#“ (Doppelkreuz oder Raute) zu erkennen. Insofern intern Variablen verwendet werden, die einer eindeutigen Verwendung zuzuordnen sind, werden diese durch ein „-“ (Minus) ausgedrückt. Die verwendeten Datentypen sollen eine Orientierung sein und werden während der weiteren Entwicklung noch genauer spezifiziert.

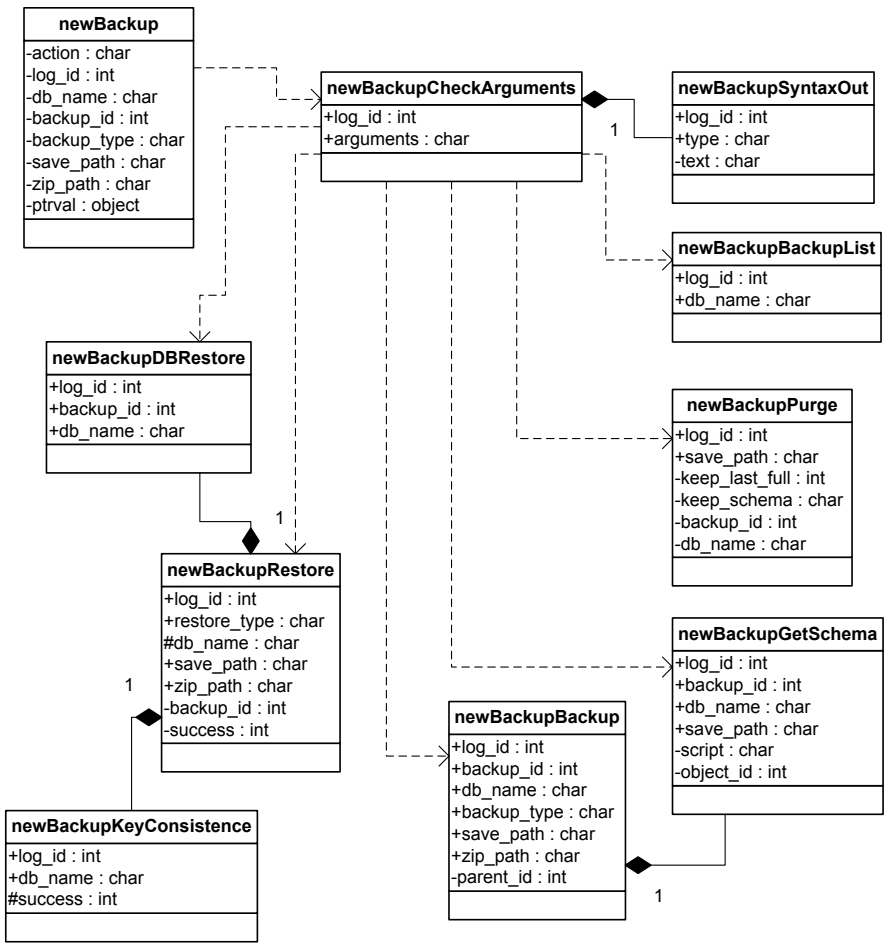

Abbildung 14: Grobe Übersicht über den Aufbau von TSQL-Backup

Um dem Benutzer nur eine Schnittstelle zur Verfügung zu stellen, wird eine Prozedur erstellt, über deren Aufruf alle Funktionen ausgelöst werden können. Als Konvention soll festgelegt sein, dass Prozeduren mit `newBackup` beginnen.

Die in UML-Notation ausgedrückte Komposition stellt dabei eine Art "verwendet von" oder auch Abhängigkeit dar. So wird die Prozedur `newBackupGetSchema` von der Prozedur `newBackupBackup` verwendet. Ein logischer Programmfluss wird, insofern möglich, durch die Pfeile dargestellt. Ausgangspunkt ist der Aufruf der Prozedur `newBackup`. Erkennbar ist, dass im Programmablauf zuerst der Aufruf der Prozedur `newBackupCheckArguments` erfolgt, welche in jedem Fall die Korrektheit der übergebenen Argumente sicherstellen soll. Von da ausgehend erfolgt je nach übergebenem Aktionsparameter der Aufruf der zugehörigen Prozedur. Da eine prozedurale Programmierung von **TSQL-Backup** vorrangig auch der Übersichtlichkeit dient, ist es denkbar, dass kleinere Programmabschnitte, dies könnten beispielsweise die SQL-Anweisungen für die Voreinstellungen sein, nicht ausgelagert werden, sondern in der Prozedur `newBackup` erstellt werden.

Dabei können die verwendeten Prozeduren aus Abbildung 14 wie folgt den Produktfunktionen aus Abschnitt 5.2.4 zugeordnet werden:

`newBackupBackup`	/PF220/, /PF230/, /PF240/, /PF250/ übernimmt die eigentliche Backup-Funktion
`newBackupCheckArguments`	/PF020/ prüft übergebene Parameter auf Korrektheit
`newBackupDBRestore`	/PF310/ stellt das Datenbankschema wieder her
`newBackupGetSchema`	/PF210/ liest das Datenbankschema aus
`newBackupKeyConsistence`	/PF350/, /PF360/ prüft Datenkonsistenz und behebt Konsistenzprobleme
`newBackupList`	/PF110/ gibt alle angefertigten Sicherungen aus
`newBackupPurge`	/PF150/ löscht vorhandene Sicherungen

`newBackupRestore`	/PF320/, /PF330/, /PF340/ übernimmt die Restore-Funktion
`newBackupSyntaxOut`	/PF010/ gibt die Syntax von **TSQL-Backup** aus

7. Entwurf

7.1. Einflussfaktoren

7.1.1. Einsatzbedingungen

Bei **TSQL-Backup** handelt es sich um ein sequentielles Programm, welches durch eine Interaktion des Nutzers ausgelöst, und schrittweise, aufeinanderfolgend abgearbeitet wird. Eine Parallelisierung des eigentlichen Programms ist nicht möglich, dies wird durch SQL Server nicht unterstützt. Eine Mehr-Benutzer-Fähigkeit ist nicht erforderlich, da es nur eine Nutzerrolle – den Datenbankadministrator gibt – und davon auszugehen ist, dass keine zeitgleichen Sicherungsvorgänge mehrerer Nutzer an einem Server ausgeführt werden.

7.1.2. Umgebungs- und Randbedingungen

TSQL-Backup wird auf jeder Plattform laufen, welche von SQL Server 2005 unterstützt wird. Darüberhinaus sind damit auch die Bedingungen für den Einsatz von DPW gegeben. Die Mindestanforderungen für ein 32-Bit-System ergeben sich damit wie folgt:

Serverseitig

- Prozessor: mind. 600 MHz
- Betriebssystem: Microsoft Windows 2000 Server SP4 oder höher
- Datenbank: Microsoft SQL Server 2005 Workgroup oder höher
- Arbeitsspeicher: mind. 512 MB
- Festplatte: 350 MB freier Speicherplatz zuzüglich ausreichend Kapazität für erstellte Sicherungen
- Laufwerk: CD-ROM oder DVD-Laufwerk
- Netzwerkanbindung
- Sonstige Software:
    - Microsoft Internet Explorer 6.0 SP1 oder höher
    - Für die serverseitige Komprimierung der Backupdateien wird die freie Software „gzip“[39] zum Einsatz kommen

[39] http://gnuwin32.sourceforge.net/packages/gzip.htm

Clientseitig

- Betriebssystem: Windows 95 oder höher, MAC/UNIX[40]
- Client-Software für SQL Server
- Netzwerkanbindung

Bei Anwendung auf einem 64-Bit-System gelten entsprechend höhere Anforderungen durch SQL Server.

Datenbankanforderungen

Unter SQL Server können Datenbanken für sehr spezielle Umgebungen und Anforderungen, wie verteilte, replizierte, sowie hochverfügbare Datenbanken, eingerichtet werden. Für diese Szenarien muss ein Einsatz von **TSQL-Backup** ausgeschlossen werden. Auch wenn das entstehende Programm die Möglichkeit bieten soll, das Datenbankschema auszulesen und auf dieser Grundlage eine neue Datenbank anzulegen, ist allein eine Konfiguration von Replikationen zur Spiegelung einer Datenbank derart aufwendig [Woo07 S. 348ff], sodass bestimmte Einschränkungen an die zu sichernde Datenbank getroffen werden müssen.

Der Befehl `CREATE DATABASE { dbname }` erstellt in SQL Server eine Datenbank mit Standardoptionen, also nach dem Standardschema der `model`-Datenbank [Woo07 S. 66] [Kon07 S. 75]. Auf eine solche Datenbank wird diese Arbeit ausgelegt sein und anhand einer solchen auch entwickelt.

TSQL-Backup benötigt administrative Rechte für Eingriffe und auszuführende Aufgaben am SQL Server, dazu gehören:

- Löschen von Tabellen und Datenbanken
- Anlegen von Tabellen und Datenbanken
- Ausführen der Windows Kommandozeile
- Ändern von Grundeinstellungen einer Datenbank
- datenbankübergreifende Lese- und Schreibzugriffe

Daher soll vereinbart werden, dass nur ein angemeldeter Datenbankadministrator das Programm nutzen kann – dies ist in der Regel der Benutzer „sa“ oder ein Windows-Nutzer, der die Rolle „sysadmin“ bei Anmeldung am SQL Server übernimmt und somit über alle

[40] Gesonderte Anforderungen zum Einsatz unter MAC oder UNIX sind den Systemanforderungen von Microsoft SQL Server Management Studio zu entnehmen und werden hier nicht berücksichtigt

Rechte verfügt [Bau06 S. 328]. Die Metadatenbank von **TSQL-Backup** wird ebenfalls unter den Rechten des Administrators erzeugt.

Eine Einschränkung auf Microsoft SQL Server ist in diesem Fall notwendig, da vom üblichen SQL-Standardbefehlssatz[41] abgewichen wird. Durch den offenliegenden Quellcode kann, insofern identische Funktionen zur Verfügung stehen, eine Portierung auf ein anderes Datenbanksystem erfolgen. Selbst dann reicht es aber meist nicht, nur die benötigten SQL-spezifischen Funktionen auszutauschen, weil der prinzipielle Aufbau einer Prozedur bei einem anderen Datenbanksystem abweichen wird [Lei03 S. 209].

7.1.3. Nichtfunktionale Produkt- und Qualitätsanforderungen

Da ein Einsatz in internationalem Gebiet nicht auszuschließen ist, wird die Anwendung in englischer Sprache mit dem Nutzer kommunizieren, eine Mehrsprachigkeit wird nicht angestrebt. Da es sich bei **TSQL-Backup** um ein Programm innerhalb von SQL Server handelt, welches seine Ausgaben lediglich durch ein SQL Client-Tool anzeigen lässt, ist ein plattformübergreifender Einsatz bereits dadurch gegeben, dass ein entsprechendes Client-Programm zur Anbindung an SQL Server auf dem verwendeten PC eingesetzt wird. Zahlen und Zeitinformationen sollen deutschen Standards entsprechen.

7.2. Grundsatzentscheidungen

7.2.1. Datenhaltung

Wie bereits erwähnt, müssen durch **TSQL-Backup** Metadaten über die angefertigten Backups verwaltet werden. Hier ist naheliegend, diese Daten in SQL Server abzulegen. Alle nötigen Tabellen, die Metadatenbank und auch alle Prozeduren und Funktionen werden durch ein gemeinsames Installationsskript eingerichtet. In diesem Skript wird der Name der zukünftigen Metadatenbank als Variable gespeichert, so dass eine freie Namenswahl ermöglicht wird. Standardmäßig soll diese Datenbank `newBackupMeta` heißen.

Die exportierten Daten aus der zu sichernden Datenbank werden auf einem durch den Nutzer angegebenen Medium gesichert, dies wird in der Regel eine Festplatte oder Netzlaufwerk

[41] Der Standardbefehlssatz für SQL ist mittlerweile SQL3 (auch SQL 2003) und definiert Datentypen, Funktionen und Anweisungen, welche grundsätzlich auch in jeder herstellerspezifischen SQL-Implementierung zur Verfügung stehen sollten [Fae07 S. 193f]

sein. Zur Ablage kommen dort lediglich die Dateninhalte aus Tabellen. Hierbei soll für jede Datenbank, auf die **TSQL-Backup** angewandt wird, ein eigener Unterordner erstellt werden, der wiederum für jedes erzeugte Backup ein einzelnes Unterverzeichnis enthält. Die eigentlichen Daten liegen dann je in einer einzelnen Textdatei, welches im Namen ebenso wie die zugehörige Hashtabelle die Objektnummer trägt, welche während des Backupprozesses vergeben wurde.

7.2.2. Verteilung im Netz

TSQL-Backup stellt, als Gesamtprodukt gesehen, lediglich eine Prozedur innerhalb von SQL Server dar. Es existiert somit keine Client-Server-Architektur. Eine Bedienung, beziehungsweise Nutzung, von **TSQL-Backup** erfolgt hingegen über die bestehende Client-Server-Architektur von SQL Server.

7.3. Programmaufruf in TSQL

Wie in Abschnitt 6.10 festgelegt, soll der Einstiegspunkt in **TSQL-Backup** für den Nutzer die Prozedur `newBackup` sein. In TSQL erfolgt der Aufruf einer gespeicherten Prozedur für gewöhnlich mit folgender Syntax:

```
EXEC { procedure_name } { value } [ ,…n ]
```

Es können also ein oder mehrere Parameter an die Prozedur übergeben werden, hierbei wird noch zwischen Eingabe- und Ausgabevariablen unterschieden. **TSQL-Backup** wird viele Funktionalitäten unter einer Prozedur zur Verfügung stellen, daher wird auch die Zahl der möglichen und erlaubten Argumente variieren. Um dies zu lösen, werden alle zu übergebenden Argumente in einer Zeichenkette zusammengefasst. Das heißt, sollen zwei Parameter mit Werten übergeben werden, würde der Aufruf wie folgt aussehen:

```
EXEC newBackup 'parameter1=value1, parameter2=value2'
```

Damit können beliebig viele Argumente übergeben werden, es muss nur ein Trennzeichen festgelegt werden, anhand dessen die Argumente getrennt werden, dies soll das Komma sein. Bedingung für alle Argumente ist also, dass nie ein Komma als Wert an einen Parameter übergeben werden darf. Sollen einfache Hochkommata übergeben werden, müssen diese maskiert werden, dies geschieht in TSQL einfach durch Verdoppelung der Zeichen. Um also ein Hochkomma in der Argumentliste zu verwenden, müssen stattdessen zwei Hochkommas

eingegeben werden, andernfalls würde für TSQL die Zeichenkette schon eher enden und eine ungültige Syntax entstehen.

Prinzipiell wird mindestens ein Parameter vom Nutzer erwartet, damit das Programm eine Funktion auslöst. Die möglichen Funktionen werden im folgenden Kapitel erläutert, und sollen als Wert dem Parameter `action` angehängt werden. Je nach Funktion werden gegebenenfalls weitere Argumente erwartet. Das Parsen der übergebenen Parameterzeichenkette geschieht unter Zuhilfenahme der Tabelle `allowed_arguments`. Diese Tabelle bildet in einer Hierarchie alle zulässigen Aufrufparameter mit zugehörigen Wertebereichen ab. Werden unzulässige Parameter oder ungültige Werte übergeben, bricht **TSQL-Backup** mit entsprechender Fehlerausgabe ab.

Es bietet sich natürlich an, bestimmte wiederkehrende Prozesse, oder in sich abgeschlossene Teilprozesse, in einzelne Unterprozeduren auszulagern. Damit gibt es einen Schritt, der jeder Funktion des Programms vorgelagert sein wird – die Prüfung auf Korrektheit der übergebenen Argumente.

7.3.1. Funktionen des Programms

In Kapitel 5.2 und Abschnitt 6.1 wurde der Funktionsumfang des Programms bereits definiert. An dieser Stelle soll genauer spezifiziert werden, wie diese Funktionen durch den Anwender angesprochen werden können:

- *`get_settings`* – gibt alle gespeicherten Voreinstellungen aus
- *`set_setting`* – setzt eine oder mehrere Voreinstellungen auf einen benutzerdefinierten Wert
- *`schema_from_db`* – liest aus einer gegebenen Datenbank das Schema aus
- *`schema_to_db`* – erzeugt aus einem ausgelesenen Schema eine neue Datenbank oder stellt daraus ein einzelnes Element wieder her
- *`list`* – zeigt alle erstellten Backups an, bei Angabe eines Datenbanknamens werden nur die Backups der zugehörigen Datenbank angezeigt
- *`list_elements`* – gibt alle durch ein Backup erfassten Elemente aus
- *`show_low`* – gibt die Logbucheinträge zu einem bestimmten Programmaufruf aus
- *`purge`* – löscht ein gegebenes Backup aus dem Archiv; bei Übergabe eines Datenbanknamens werden alle Backups zu dieser Datenbank gelöscht; ohne Angabe eines Datenbanknamens oder bestimmten Backups werden alle vorhandenen Backups

gelöscht; es soll möglich seinn eine bestimmte Anzahl X Backups vom Löschvorgang auszuschließen (dabei werden die letzten X erstellten Backups erhalten); Backups, die nur das Schema einer Datenbank enthalten, sollen auf Wunsch des Nutzers vollständig erhalten werden

- *backup* – stellt die Funktion zur Erstellung eines vollständigen, differenziellen oder inkrementellen Backups einer Datenbank zur Verfügung; Backups sollen auf Wunsch komprimiert werden können
- *restore* – ermöglicht die Wiederherstellung einer kompletten Datenbank oder eines ausgewählten Elements aus einem Backup; der Datenbankname, auf den die Wiederherstellung angewandt werden soll, soll übergeben werden

Um also alle bereits erstellten Backups aufzulisten, soll der Prozeduraufruf wie folgt aussehen:

```
EXEC newBackup 'action=list'
```

Wird die Prozedur ohne jegliche Argumente aufgerufen, wird dem Nutzer die vollständige Syntax des Programms ausgegeben.

7.3.2. Vollständige Syntax des Programmaufrufs

Abhängig von der gewünschten Funktion sind weitere Argumente erforderlich oder optional anzugeben. Ausgehend von Abschnitt 7.3.1 baut sich die vollständige Syntax wie folgt auf:

```
1.    EXEC newBackup 'action=<action_values>'

2.    <action_values>:={
3.        get_settings
4.      | set_setting,<set_setting_options>
5.      | schema_from_db,db=(STRING),<schema_from_db_options>
6.      | backup,db=(STRING),<backup_options>
7.      | list,<list_options>
8.      | show_log,id=(INT)
9.      | purge,<purge_options>
10.     | list_elements,id=(INT)
11.     | schema_to_db,id=(INT),db=(STRING),<schema_to_db_options>
12.     | restore,id=(INT),db=(STRING),<restore_options>
13.   }

14.   <set_setting_options>:={
15.       save_path=(STRING)
16.     | schema_login=(user,win)
17.     | schema_script=(STRING)
18.     | compress=(yes,no)
19.     | zip_path=(STRING)
20.   }
```

```
21.   <schema_from_db_options>:={
22.       schema_pw=(STRING)
23.     | schema_user=(STRING)
24.   }

25.   <backup_options>:={
26.       type=(full,incr,diff)
27.     | quick=(yes,no)
28.     | compress=(yes,no)
29.     | schema_pw=(STRING)
30.     | schema_user=(STRING)
31.   }

32.   <list_options>:={
33.       db=(STRING)
34.   }

35.   <purge_options>:={
36.       force=(yes,no)
37.     | db=(STRING)
38.     | keep_last_full=(INT)
39.     | id=(INT)
40.     | keep_schema=(yes,no)
41.   }

42.   <schema_to_db_options>:={
43.       force=(yes,no)
44.   }

45.   <restore_options>:={
46.       force=(yes,no)
47.     | element_id=(INT),<consistence_options>
48.   }

49.   <consistence_options>:={
50.       consistence=(handle,ignore)
51.   }
```

Listing 6: Vollständige Syntax von TSQL-Backup

Der Einstieg in die Syntax erfolgt über das Argument `action`, welches bei jedem Aufruf erwartet wird. Die weitere Syntax ist wie folgt zu verstehen:

- mögliche Werte zu einem Parameter sind durch | (Pipe) getrennt, wobei nur einer der Werte gewählt werden kann
- erfordert ein Wert weitere Argumente, schließen sich diese direkt durch Komma getrennt auf gleicher Zeile an; sind diese weiteren Argumente hingegen optional, wird durch spitze Klammern auf eine Untersyntax verwiesen
- werden Parameter erwartet, die eine benutzerdefinierte Eingabe des Nutzers erwarten, wird der erlaubte Datentyp des einzugebenden Werts in runden Klammern aufgeführt,

wobei `INT` (Ganzzahl) oder `STRING` (Zeichenkette) zur Verfügung stehen, weiterhin können Aufzählungen angegeben werden (durch Komma getrennte Werte in runden Klammern), woraus ein Wert ausgewählt werden muss

Aus dieser Syntax ergeben sich Abhängigkeiten und damit die Befüllungslogik für die Tabelle `allowed_arguments`, welche bereits bei Einrichtung des Programms entsprechend initialisiert werden soll. Eine Erklärung aller Optionen der Syntax kann dem Anhang entnommen werden.

Die Optionen für die Konsistenzwahrung (Parameter `consistence`) wirken sich nur bei der Wiederherstellung eines einzelnen Elements aus und haben folgende Bedeutung:

- *`handle`* – **TSQL-Backup** versucht Inkonsistenzen selbst zu korrigieren, dabei können überflüssige Datenzeilen im wiederhergestellten Element gelöscht, aber auch fehlende Schlüssel durch einen Dummy-Eintrag im wiederhergestellten Element angelegt werden. Ziel ist, die wiederhergestellten Daten in eine Form zu bringen, die allen vorhandenen und aktiven Schlüsselbeziehungen und Einschränkungen entspricht.
- *`ignore`* – **TSQL-Backup** stellt in jedem Fall alle Daten aus dem Backup wieder her. Sollten hierbei Verletzungen von Schlüsselbeziehungen oder Einschränkungen auftreten, müssen diese deaktiviert werden. Ziel ist, keine Daten durch den Versuch einer Konsistenzwiederherstellung ungewollt zu verlieren.

7.3.3. Verwendete Prozeduren in TSQL-Backup

Das bereits in Abschnitt 6.10 vorgestellte Abhängigkeitsdiagramm soll um zusätzliche Funktionalitäten erweitert werden. Somit können weitere Prozeduren definiert werden:

- `newBackupTextOut` – übernimmt die Logbuchfunktion und wird zur Textausgabe verwendet
- `newBackupSyntaxFromDB` – liest die möglichen Parameter zu den Programmfunktionen von **TSQL-Backup** aus der Metadatenbank aus und bildet eine an Grammatik nach, welche an die Standardnotation von SQL angelehnt ist
- `newBackupPrePathInit` – liest Verzeichnispfade aus den Voreinstellungen und legt den physikalischen Speicherpfad an
- `newBackupIsolation` – aktiviert die Snapshot-Isolationsstufe
- `newBackupPreDBInit` – erzeugt eine eindeutige Backup-ID

- `newBackupCreateDB` – erzeugt ein `CREATE DATABASE`-Skript (DPW kann nur die in einer Datenbank enthaltenen Schemaobjekte auslesen, aber nicht das Skript, womit die Datenbank selbst erzeugt wird)
- `newBackupColToBin` – konvertiert übergebene Spaltentypen in einen für die Hashfunktion verständlichen Datentyp
- `newBackupHashData` – berechnet aus einer übergebenen Zeichenkette einen Hashwert
- `newBackupKeyTrigger` – übernimmt das Aktivieren/Deaktivieren von Constraints

Damit ergibt sich ein neues Abhängigkeitsdiagramm für die Prozeduren:

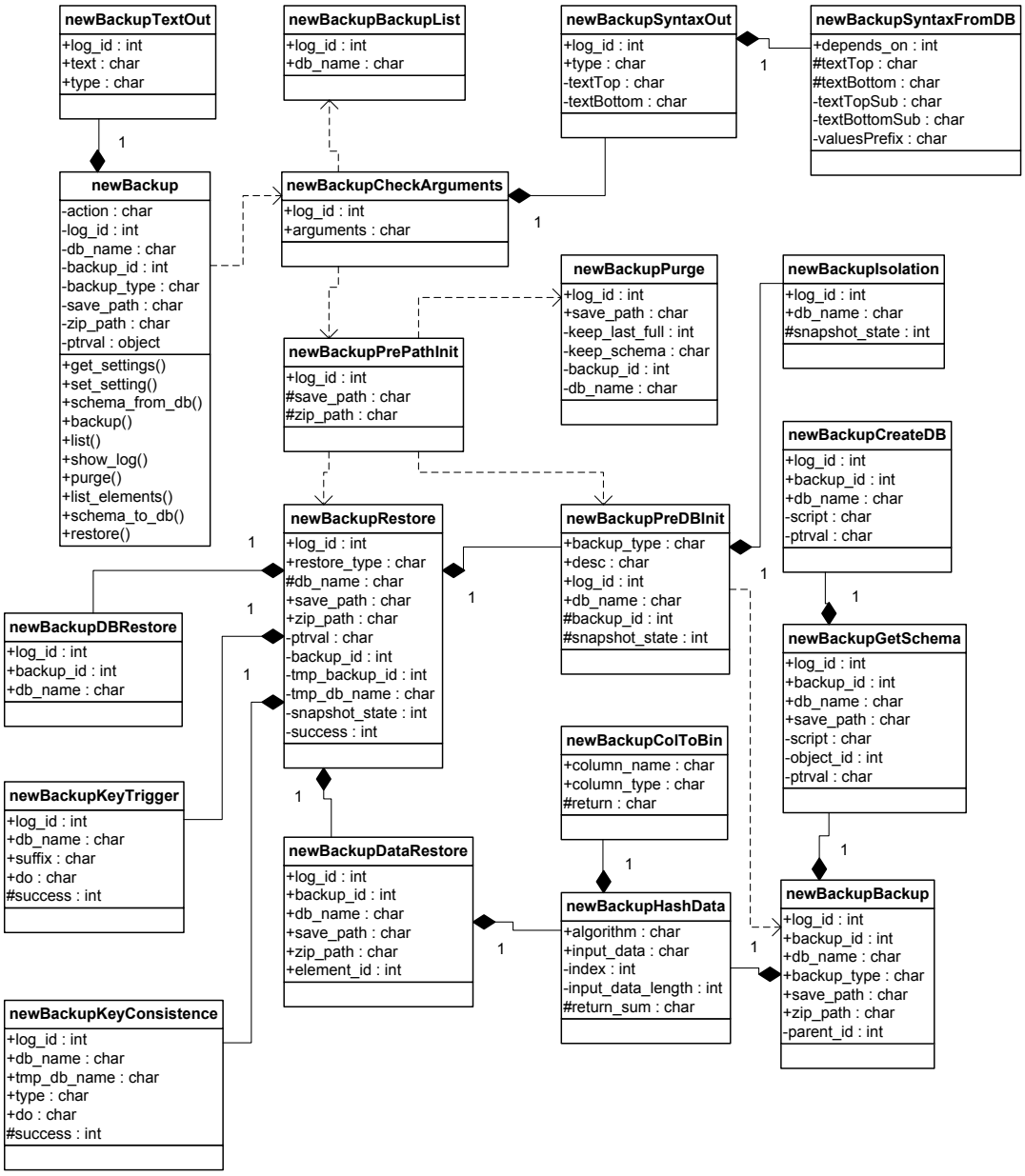

Abbildung 15: Abhängigkeiten der Prozeduren von TSQL-Backup

7.3.4. Schnittstellen- und Prozedurbeschreibung

In diesem Abschnitt soll eine Übersicht über die Schnittstellen, konkret die Prozedurköpfe, und eine zusammenfassende Beschreibung aller Prozeduren in **TSQL-Backup** gegeben werden. Es werden hierbei Ein- und Ausgabeparameter der Prozeduren, sowie auch Variablen, die innerhalb einer Prozedur einer eindeutigen Verwendung zuordenbar sind, berücksichtigt. Die Schnittstellenbeschreibung ist dem Anhang zu entnehmen.

7.4. Entwurf der Backup-Funktionalität von TSQL-Backup

An dieser Stelle soll exemplarisch die Backup-Funktionalität im Rahmen der schriftlichen Ausarbeitung erläutert werden, da diese umfangreicher ist als die weiteren geplanten Funktionen und einen guten Einblick in den Aufbau und Ablauf von **TSQL-Backup** geben wird.

7.4.1. Aktivitätsdiagramm

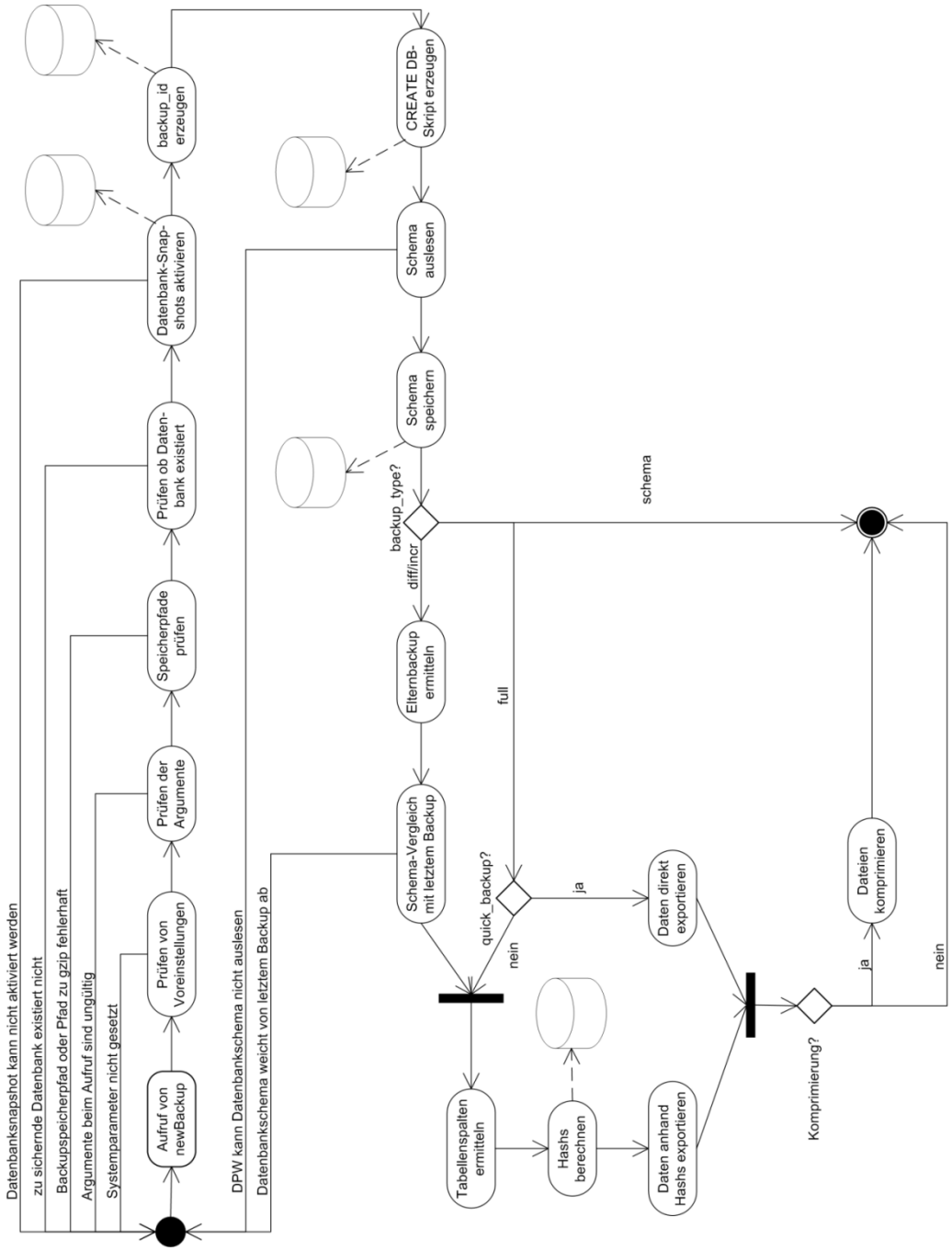

Abbildung 16: Aktivitätsdiagramm für die Backup-Funktionalität

Hierbei handelt es sich nur um den Ausschnitt von **TSQL-Backup**, der die Sicherungsfunktion ausführt. Deutlich erkennbar ist auch hier, dass es, aufgrund der SQL-Befehlskonsole, nur einen Interaktionspunkt des Nutzers mit dem Programm gibt - den Aufruf von `newBackup`. Jeder Fehler führt, unter Ausgabe einer Fehlermeldung, zum Abbruch.

Bereits direkt nach Programmausführung werden durch **TSQL-Backup** alle wichtigen Prüfprozesse ausgeführt, die eine korrekte Arbeitsweise garantieren sollen. Bis zum Auslesen des Datenbankschemas ist die Funktionsweise für ein vollständiges, differenzielles, inkrementelles oder ein Schema-Backup gleich. Danach werden je nach Art des Backups verschiedene Aktionen ausgeführt, lediglich bei einem Schema-Backup erfolgt sofort eine Beendigung des Programms. Ein Backup ohne Hashberechnung (vergleiche Programmaufruf aus Abschnitt 7.3.2), welches zweifelsohne schneller ablaufen wird als ein Backup mit aufwendiger Hashberechnung, steht nur bei vollständigem Backup zur Verfügung. Die Komprimierung der exportieren Daten aus der Datenbank kann wieder für die drei Backup-Arten (abgesehen vom Schema-Backup) ausgeführt werden.

Zu erkennen sind auch diejenigen Aktionen, welche die Metadatenbank um Informationen über das aktuell ausgeführte Backup anreichern. Dazu gehören die Generierung einer `backup_id`, sowie das Speichern des Datenbankschema und der Hashwerte. Lesende Zugriffe auf die Datenbanken sollen hier nicht dargestellt werden, da prinzipiell jede Aktion in diesem Diagramm einen lesenden Zugriff ausführen wird. Nur die Aktivierung der Datenbanksnapshots ist noch zu erkennen, weil diese Modifikationen an der Datenbank vornimmt und somit als schreibende Aktivität zu bewerten ist.

7.4.2. Aufruffolge der Prozeduren

Nach Funktionalität der Prozeduren aus den Abschnitten 0 und 7.3.4 sowie der Aktivität aus Abschnitt 7.4.1 ergibt sich folgende Aufrufreihenfolge, welche mittels einem UML-Sequenzdiagramm dargestellt werden soll:

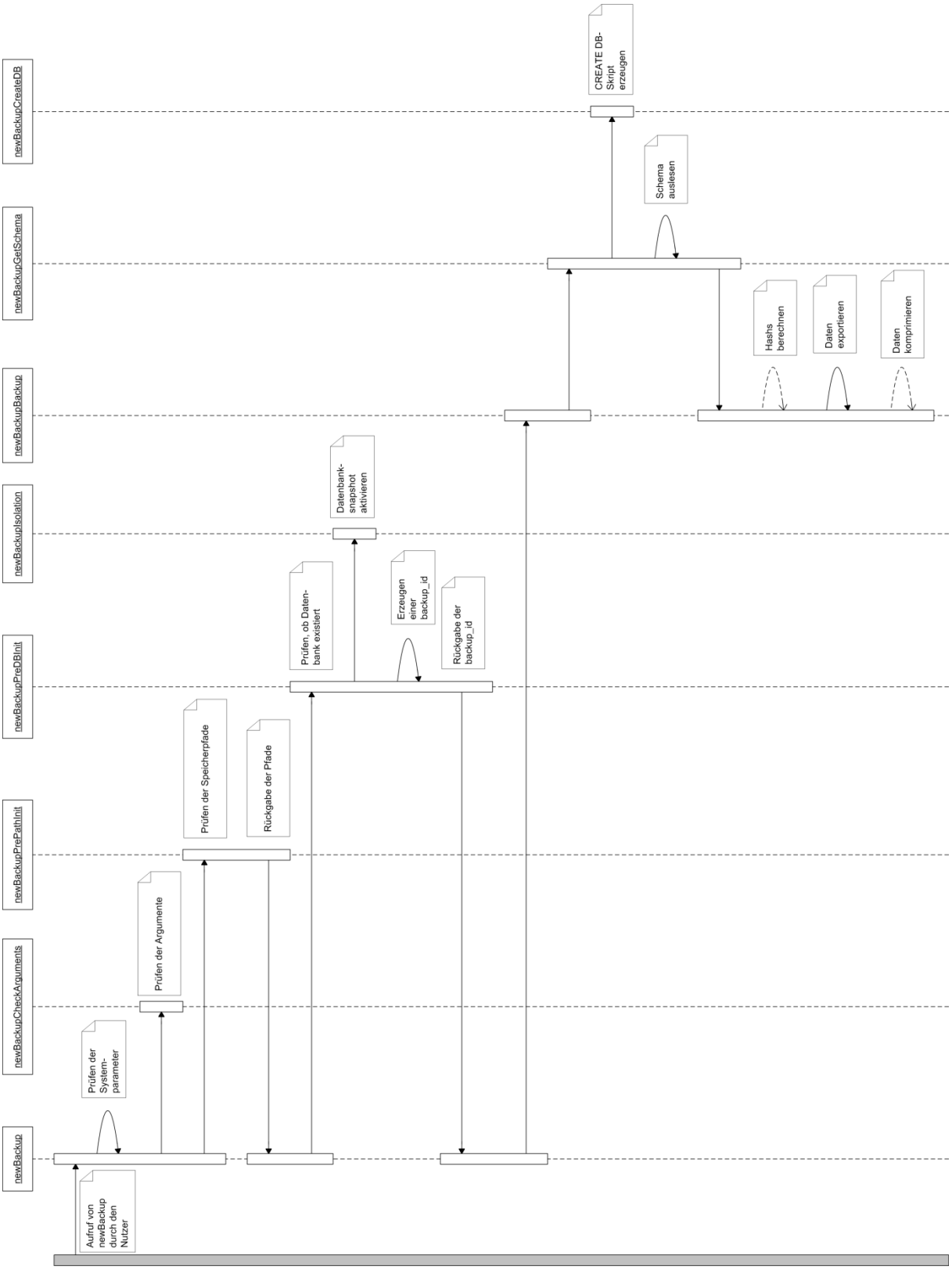

Abbildung 17: Sequenzdiagramm für die Backup-Funktionalität

Die Berechnung der Hashs und die Komprimierung der Daten sind optional und in der Abbildung durch entsprechende Pfeile dargestellt. Der Aufruf einiger Prozeduren folgt nicht dem logischen Programmfluss aus Abbildung 15. Auf diese Weise kann aber die Wiederverwendbarkeit der Prozeduren erhöht werden, indem nach Abschluss der Prozedur die Kontrolle wieder an die höhere Instanz (hier also in der Regel `newBackup`) zurückgegeben wird.

Mit Hilfe dieses Sequenzdiagrammes kann nun ein erster Ansatz für die benutzten Prozeduren in Pseudocode erfolgen:

```
1. PROCEDURE newBackup() {
2.      SystemparameterPrüfen
3.      newBackupCheckArguments()
4.      newBackupPrePathInit()
5.      newBackupPreDBInit()
6.      newBackupBackup()
7. }

1. PROCEDURE newBackupCheckArguments() {
2.      ArgumentePrüfen
3. }

1. PROCEDURE newBackupPrePathInit() {
2.      SpeicherpfadePrüfen
3. }

1. PROCEDURE newBackupPreDBInit() {
2.      DatenbankAufExistenzPrüfen
3.      newBackupIsolation()
4.      BackupIdErzeugen
5. }

1. PROCEDURE newBackupIsolation() {
2.      SnapshotsAktivieren
3. }

1. PROCEDURE newBackupBackup() {
2.      newBackupGetSchema()
3.      WENN BerechnungVonHashsAktiv DANN BerechneHashs
4.      DatenExportieren
5.      WENN KomprimierungVonDatenAktiv DANN DatenKomprimieren
6. }

1. PROCEDURE newBackupGetSchema() {
2.      newBackupCreateDB()
3.      SchemaAuslesen
4. }

1. PROCEDURE newBackupCreateDB() {
2.      ErzeugeCreateDbSkript
3. }
```

Listing 7: Pseudocode

Exemplarisch soll noch genauer auf die wichtigen Prozeduren `newBackupGetSchema` und `newBackupBackup` eingegangen werden.

7.4.3. Prozeduren newBackupGetSchema und newBackupBackup

Um nun detailliert auf eine Prozedur einzugehen und deren Arbeitsweise zu verdeutlichen, bietet sich für den Entwurf ein Nassi-Shneiderman-Diagramm an (bekannt auch als Struktogramm), da hiermit insbesondere auch Schleifen und Verzweigungen gut dargestellt werden können. Es würden sich auch Programmablaufpläne eignen, jedoch ist bei Struktogrammen eine größere Übersichtlichkeit gewährleistet. Zudem sollen wichtige Codefragmente gelistet werden. Die zugehörigen Struktogramme können dem Anhang entnommen werden.

Der Abschnitt vom Auslesen der Datenbank mit DPW bis zum Einlesen in eine temporäre Tabelle kann wie folgt aussehen:

```
--Hilfsprogramm zum Auslesen der Datenbank starten und Ausgabe des
Programms in eine Hilfstabelle schreiben
SET @helperStr=CHAR(34)+@script+CHAR(34)+' script -f -q -d '+@db_name+'
      -schemaonly -nodropexisting '+@save_path+'\'+@db_name+'.sql'
INSERT INTO #nB_schematmp EXEC master.dbo.xp_cmdshell @helperStr
```

Listing 8: Auslesen des Datenbankschemas

Die Variable `@script` enthält hierbei den Pfad von DPW zur Ausführung auf der Kommandozeile. Die temporäre Tabelle enthält nun die Kommandozeilenrückgabe von DPW und kann auf Fehler untersucht werden.

Das Einlesen der erzeugten Datei kann so aussehen, wobei die temporäre Tabelle für weitere Prüfungen verwendet werden kann :

```
--Parsen der ausgegebenen Datei
TRUNCATE TABLE #nB_schematmp
SET @helperStr='dir /B '+CHAR(34)+@save_path+'\'+@db_name+'.sql'+CHAR(34)
INSERT INTO #nB_schematmp EXEC master.dbo.xp_cmdshell @helperStr
IF EXISTS (SELECT * FROM #nB_schematmp WHERE text='Datei nicht gefunden')
BEGIN
      SET @helperStr ='database '''+@db_name+''' contains no objects –
            aborted'
      EXEC dbo.newBackupTextOut @log_id,@helperStr,'fail'
      GOTO jumpFail
END
--SQL-Skript in Hilfstabelle schreiben
TRUNCATE TABLE #nB_schematmp
SET @helperStr='BULK INSERT #nB_schematmp FROM
      '+CHAR(34)+@save_path+'\'+@db_name+'.sql'+CHAR(34)+' WITH
      (DATAFILETYPE=''widechar'')'
EXEC(@helperStr)
```

Listing 9: Einlesen der Ausgabe von DPW

Wenn DPW bei seiner Arbeit keine Datei erzeugt, bedeutet dies, dass die Datenbank keine weiteren Schemaobjekte enthält, die durch Nutzer angelegt wurde. Es zeigt sich bereits hier, dass, aufgrund der begrenzten Syntax und Funktionen von TSQL, für die Arbeit im Dateisystem nur die Windowskommandozeile verwendet werden kann.

Nachdem durch `newBackupPreDBInit` der Modus zur Snapshot-Isolationsstufe aktiviert wurde, laufen die nachfolgenden Schritte innerhalb der Prozedur in einer einzelnen Transaktion ab (siehe Abschnitt 6.5). In dieser Transaktion laufen dann gekapselt alle Schritte zur Erstellung einer Sicherung ab, um so einen konsistenten Datenstand zu gewährleisten.

Über ein etwas umfangreicheres SQL-Skript lässt sich vergleichen, ob die ausgelesenen Schemaobjekte des aktuellen Backups mit dem des letzten Backups übereinstimmen:

```
SELECT x.name,y.owner,z.type_desc FROM dbo.backup_objects w
INNER JOIN dbo.objects x ON w.object_id=x.object_id
INNER JOIN dbo.objects_detail y ON x.object_id=y.object_id
INNER JOIN dbo.schema_lookup z ON x.type=z.type
WHERE w.backup_id=@backup_id AND EXISTS (
      SELECT * FROM dbo.backup_objects c
      INNER JOIN dbo.objects a ON c.object_id=a.object_id
      INNER JOIN dbo.objects_detail b ON a.object_id=b.object_id
      WHERE c.backup_id=@parent_id AND a.name=x.name AND b.owner=y.owner
            AND a.type=x.type AND
            dbo.newBackupHashData('MD5',CAST(CAST(b.create_script AS
            NVARCHAR(MAX)) AS VARBINARY(MAX))) <>
            dbo.newBackupHashData('MD5',CAST(CAST(y.create_script AS
            NVARCHAR(MAX)) AS VARBINARY(MAX)))
)
```

Listing 10: Vergleich von Schemaobjekten in der Metadatenbank

Für die Erzeugung der Hashwerte müssen alle Spalten einer Tabelle herangezogen werden, die SQL-interne ID der Tabelle ist hierbei in der Variablen `@helperInt` enthalten:

```
--alle Spalten zu aktueller Tabelle aus Datenbank auslesen
SET @helperStr='SELECT x.name,y.name FROM '+@db_name+'.sys.syscolumns x
INNER JOIN '+@db_name+'.sys.systypes y ON x.xusertype=y.xusertype
WHERE x.id='+CAST(@helperInt AS VARCHAR(10))+' ORDER BY x.colorder'
TRUNCATE TABLE #nB_cols
INSERT INTO #nB_cols EXEC(@helperStr)
SET @helperStr=''
--SQL-String zur Erzeugung des Hashs aus allen Spalten der Tabelle
      zusammenbauen
DECLARE columncur CURSOR FOR
      SELECT colname,coltype FROM #nB_cols
OPEN columncur
FETCH NEXT FROM columncur INTO @helperStr1, @helperStr2
WHILE @@FETCH_STATUS=0
BEGIN
      --aus allen Spalten einer Tabelle wird ein Hashwert gebildet und
            dieser zu einer langen Zeichenkette verbunden, damit entsteht
            ein eindeutiger Hashwert für jede Zeile einer Tabelle (unter
```

```
            der Annahme, dass MD5 keine Kollisionen verursacht)
      SET @helperStr=@helperStr+'+'+db_name()+
            '.dbo.newBackupHashData(''MD5'','+
            dbo.newBackupColToBin('x.['+@helperStr1+']',@helperStr2)+')'
      FETCH NEXT FROM columncur INTO @helperStr1, @helperStr2
END
CLOSE columncur
DEALLOCATE columncur
```

Listing 11: Alle Spalten zur Hashwertbildung verwenden

Die erzeugte Zeichenkette verknüpft dann die Hashwerte aller Spalten zu einem Gesamt-Hashwert, welcher dann für Vergleiche (mit den erzeugten Hashwerten aus dem letzten Backup) genutzt werden kann, um diese in die zugehörige Hashtabelle einzuspielen:

```
SET @helperStr='INSERT INTO dbo.hashs_'+CAST(@helperInt AS VARCHAR(10))+'
      SELECT CASE WHEN y.hash IS NULL THEN x.hash ELSE y.hash END AS
            hash,CASE WHEN y.hash IS NULL THEN 0 ELSE 1 END AS delflag
      FROM (
            SELECT '+@helperStr+' AS hash FROM '+@table+' x
      ) x FULL OUTER JOIN hashs_'+CAST(@lastBackup AS VARCHAR(10))+' y ON
            x.hash=y.hash
      WHERE y.hash IS NULL OR x.hash IS NULL'
```

Listing 12: Hashwerte einer Tabelle berechnen

`@table` ist hier der kanonische Name der Tabelle, zu der die Hashs erzeugt werden sollen und `@lastBackup` die ID der zugehörigen Hashtabelle aus dem letzten Backup.

Auf Grundlage der erzeugten Hashtabelle können die Daten nun aus der Tabelle exportiert werden:

```
SET @helperStr='bcp '+CHAR(34)+'SELECT * FROM '+@table+' x WHERE EXISTS
      (SELECT * FROM '+db_name()+'.dbo.hashs_'+CAST(@helperInt AS
      VARCHAR(10))+' y WHERE y.hash='+@helperStr+')'+CHAR(34)+' queryout
      '+CHAR(34)+@save_path+'\'+@db_name+'\backup_'+CAST(@backup_id AS
      VARCHAR(10))+'\table_'+CAST(@helperInt AS VARCHAR(10))+CHAR(34)+' -k
      -T -N -m 0'
EXEC master.dbo.xp_cmdshell @helperStr, no_output
```

Listing 13: Datenexport anhand der berechneten Hashs

Zum Verständnis der Arbeitsweise von **TSQL-Backup** soll nochmals verdeutlicht werden, dass bei aktivierter Hashberechnung (welche für die Erzeugung von differenziellen und inkrementellen Sicherungen notwendig ist), zuerst über alle Datenzeilen einer Tabelle Hashwerte gebildet und in der Metadatenbank gespeichert werden. Anhand dieser Hashwerte wird dann in einem weiteren Schritt der Datenbestand durch „bcp“ in das Backup exportiert. Hierzu werden die berechneten Hashwerte nochmals mit der zu sichernden Tabelle verknüpft, um alle Datenzeilen zu markieren, die exportiert werden sollen.

7.5. Sicherheitsaspekte

Für gewöhnlich werden die zu sichernden Datenbanken sensible Daten (meist Kundendaten oder Firmendaten) enthalten, die es zu schützen gilt. Hierbei können drei Arten von Schutzmechanismen unterschieden werden:

Physischer Schutz

Die von **TSQL-Backup** erstellten Backupdateien sollten getrennt vom eigentlichen Datenbankserver auf einem gesonderten Laufwerk gespeichert werden. Meist steht in größeren Firmen ein eigener Serverraum zur Verfügung, der zusätzlich gegen physische Zugriffe von unberechtigten Personen gesichert ist und somit die Gefahr des Diebstahls durch Einbruch vermindert wird. Zusätzlich ist damit der Aspekt der Datensicherheit bei Ausfall des Datenbankserver gewährleistet.

Unberechtigter Aufruf

Weiterhin muss sichergestellt sein, dass ein Aufruf des Programms und damit das Löschen von Backups oder das Überschreiben einer Datenbank aus einer Sicherung durch unberechtigte Personen verhindert wird. Dies wird dadurch erreicht, dass eine Nutzung von **TSQL-Backup** nur durch ein Mitglied der Administratoren-Rolle möglich ist.

Kennwortschutz

Um die Daten vor Manipulation, und im Falle eines Diebstahls durch einen kompromittierten Rechner, zu schützen, sollten diese verschlüsselt werden. Eine Möglichkeit ist die Verschlüsselung direkt durch das Komprimierungsprogramm, insofern unterstützt. „gzip“ bietet diese Möglichkeit leider nicht. Prinzipiell lässt sich aber unter Anpassung des Quellcodes von **TSQL-Backup** jedes Komprimierungsprogramm für Windows einsetzen, welches eine

Kommandozeilenunterstützung anbietet. Ein geeignetes Programm, welches einen Passwortschutz bietet, ist "WinRAR"[42] (kommerziell). Durch den offenen Quellcode ist es aber auch möglich, ein zusätzliches Tool einzubinden, welches entsprechend nach dem Export oder vor dem Import von Daten die zugehörige physische Datei verschlüsselt. Hierzu muss der Quellcode lediglich um den entsprechenden `EXEC`-Befehl erweitert werden. Ein geeignetes, frei verfügbares Programm auf Kommandozeilenebene ist "GNU Privacy Guard"[43].

[42] http://www.winrar.de/

[43] http://www.gnupg.org/

8. Implementierung und Praxistest

8.1. Installation auf dem Datenbankserver

Die erstellten Prozeduren und das Skript zur Initialisierung der Metadatenbank wurden zu einem einzigen SQL-Skript zusammengefügt, um dem Nutzer eine unkomplizierte Installation zu ermöglichen. Für den Nutzer besteht die Möglichkeit der freien Namenswahl für die Metadatenbank, ausgenommen hiervon sind die Namen der Systemdatenbanken. Dazu muss im Installationsskript die Zeile 7 geändert werden:

```
SET @metaDb='newBackupMeta'
```

Zur Installation wird im SQL Server Management Studio die Installationsdatei `TSQLinstall.sql` und eine Verbindung zum Server, auf dem **TSQL-Backup** installiert werden soll, geöffnet. Schließlich muss das Skript nur noch mit dem Tastendruck F5 ausgeführt werden. Bei Vorhandensein einer Datenbank mit gleichem Namen wird ein Fehler ausgegeben:

```
meta database already exists
```

Durch Ändern der Zeile 8 im Installationsskript nach:

```
SET @dropOld='true'
```

kann die vorhandene Datenbank überschrieben werden (Achtung: vorhandene Daten gehen unwiderruflich verloren). Ein erfolgreiches Anlegen wird mit folgender Ausgabe bestätigt:

```
meta database created
scripts created
```

Unter der Annahme, dass die Metadatenbank noch nicht existiert, kann so die Installation von TSQL-Backup einfach durch Ausführung des erstellten Skripts durchgeführt werden. Für einen fehlerfreien Start müssen neben der Installation von DPW (siehe Abschnitt 6.3.3) und „gzip“ auf dem entsprechenden Server noch die zugehörigen Pfade für **TSQL-Backup** gesetzt werden.

Standardmäßig ist der Pfad zu DPW für ein x64-System schon gesetzt und kann wie folgt abgerufen werden:

Programmaufruf

```
newBackup 'action=get_settings'
```

Ausgabe

```
predefined settings:
log_level     | all
schema_login  | win
schema_script | C:\Program Files (x86)\Microsoft SQL
Server\90\Tools\Publishing\SqlPubWiz.exe
```

Der Speicherort für die Backups und „gzip“ muss noch angegeben werden, beispielsweise mit:

Programmaufruf

```
newBackup 'action=set_setting,
zip_path=C:\Programme\GnuWin32\bin\gzip.exe, save_path=D:\backup'
```

Ausgabe

```
settings saved
```

8.2. Anwendungstest

8.2.1. Prüfung auf Fehlverhalten

Nachfolgende Tests von **TSQL-Backup** sollen für den Nutzer zum einen als Einstieg in die Funktionsweise dienen, andererseits aber auch Fehlfunktionen bei der Arbeit aufdecken. Für den Fall der Syntaxausgabe durch **TSQL-Backup** wird die Syntax zur kürzeren Darstellung nachfolgend mit […] abgeschnitten. Die Testszenarien können dem Anhang entnommen werden.

Die durchgeführten Tests (mit denen alle umgesetzten Funktionen, ausgenommen Sicherung und Wiederherstellung, getestet wurden) verliefen fehlerfrei, beziehungsweise zeigten das erwartete Verhalten (wie beispielweise die Prüfung auf Nutzung des Programms als Administrator).

8.2.2. Durchführung eines Backups

In diesem Abschnitt sollen folgende Funktionalitäten gezeigt werden:

- Auslesen des Datenbankschemas
- vollständige Sicherung
- differenzielle Sicherung
- inkrementelle Sicherung

Diese Vorgänge werden an der `AdventureWorks` [44] -Datenbank ausgeführt. Bei Sicherungsvorgängen werden unter anderem auch die gesicherten Objekte ausgegeben. Da hierbei sehr lange Listen entstehen können, werden hier nur Auszüge der Ausgabe wiedergegeben. Die durchgeführten Sicherungsvorgänge können dem Anhang entnommen werden.

Alle durchgeführten Sicherungsvorgänge wurden vollständig und korrekt durchgeführt. Zu sehen ist, dass auch bei differenzieller und inkrementeller Sicherung die Hashwerte für alle Tabellen berechnet werden, um auf Grundlage dieser Hashwerte ein Delta zum letzten gesicherten Datenstand zu ermitteln. Wie erwartet, bricht die Ausführung einer Sicherung bei Vorhandensein einer verschlüsselten Prozedur ab (vergleiche Abschnitt 6.3.3).

8.2.3. Durchführung einer Wiederherstellung

In diesem Kapitel sollen folgende Funktionalitäten gezeigt werden:

- Wiederherstellung eines Datenbankschemas
- Wiederherstellung einer Datenbank
- Wiederherstellung einer Tabelle aus einer Sicherung

Diese Vorgänge werden mit den im vorigen Kapitel angefertigten Sicherungen ausgeführt. Zur Wiederherstellung muss die ID des zu verwendenden Backups angegeben werden. Die durchgeführten Wiederherstellungsvorgänge können dem Anhang entnommen werden.

Die Wiederherstellungsvorgänge wurden erfolgreich abgeschlossen, auch wurde die erzeugte Schlüsselverletzung korrekt erkannt und eine Wiederherstellung verhindert, da an dieser Stelle ein Eingreifen des Nutzers notwendig war. Ein erneuter Aufruf von **TSQL-Backup** mit

[44] Es handelt sich hierbei um eine Beispieldatenbank von Microsoft, welche bereits die verbreitetsten Schemaobjekte beinhaltet, siehe http://msdn.microsoft.com/de-de/library/ms124659%28SQL.90%29.aspx

aktivierter Korrektur von Schlüsselverletzungen führte schließlich zur erfolgreichen Wiederherstellung der Datenbank.

8.3. Vergleich mit vorhandenen Tools

In diesem Abschnitt soll das entworfene Produkt den integrierten Möglichkeiten von SQL Server und einigen kommerziellen Produkten gegenübergestellt werden. Der Vergleich mit SQL Server selbst soll vollzogen werden, da **TSQL-Backup** nicht ausschließlich eine Funktionslücke in SQL Server schließen soll, sondern dazu dient, bestehende Lücken in Sicherungsstrategien zu ergänzen. Dazu werden zuerst die Funktionen der Programme verglichen und schließlich in einem Test Aussagen über Performanz und Ressourcenverbrauch gemacht.

8.3.1. Funktionsumfang

Funktion	Quest LiteSpeed for SQL Server	redgate SQL Backup Pro	Microsoft SQL Server	TSQL-Backup
Bezugsart	kommerziell	kommerziell	-	frei
vollständige Sicherung	ja	ja	ja	ja
differenzielle Sicherung	ja	ja	ja	ja
inkrementelle Sicherung	nein	nein	nein	ja
Schemasicherung	nein	nein	nein	ja
Schemawiederherstellung	bedingt	nein	nein	ja
Transaktionsprotokollsicherung	ja	ja	ja	nein
Dateisicherung	ja	ja	ja	nein
Object-Level-Recovery aus vollständiger Sicherung	ja	ja	nein	ja
Object-Level-Recovery aus Teilsicherung (diff./inkr.)	nein	nein	nein	ja
Object-Level-Recovery bei vorhandener Schlüsselverletzung	bedingt[45]	-[46]	nein	ja
Sichern in Standard-SQL-Format	ja	nein	ja	nein
Wiederherstellung aus Standard SQL-Format	ja	nein	ja	nein
Wiederherstellung aus Teilsicherung (diff.)	nein	ja	nein	ja
Passwortschutz/ Verschlüsselung	ja	ja	nein	nein
Komprimierung	ja	ja	nein	ja
GUI	ja	ja	ja	nein
Integration in TSQL	ja	ja	ja	ja
Benötigte Berechtigungsstufe	sysadmin	sysadmin	serveradmin/ sysadmin	sysadmin
Löschen von Backups	nein	nein	nein	ja
Installation von Systemdiensten / SQL Server Komponenten	ja	ja	nein	ja

Tabelle 9: Funktionsvergleich mit anderen Tools

[45] Siehe Auswertung in Kapitel 8.3.3

[46] Kann aufgrund der unausgereiften Funktion nicht beurteilt werden

Bereits aus vorstehender Tabelle ist erkennbar, dass selbst die kommerziellen Produkte vom Funktionsumfang nicht das halten, was man von ihnen erwartet. Insbesondere das Object-Level-Recovery ist meist nur aus einer vollständigen Sicherung möglich und kann nur unter bestimmten Umständen ausgeführt werden (so kann bei redgate kein Restore bei einer Schlüsselverletzung erfolgen). Ansonsten ähneln diese Produkte stark der Funktionalität von SQL Server selbst. Deutlich ist auch erkennbar, dass **TSQL-Backup** versucht, bestimmte Lücken in diesem Funktionsumfang zu füllen.

8.3.2. Performanz

Für die Performanztests wird die `AdventureWorks`-Datenbank verwendet, welche eine Dateigröße von 178,75 MB und einen freien Platz[47] von 16,02 MB aufweist. Um die Datenbank zu vergrößern (für verlässlichere Ergebnisse), werden folgende SQL Befehle ausgeführt:

```
CREATE TABLE table_dummy (
   text VARCHAR(MAX)
)
GO
DECLARE @i INT
SET @i = 1
WHILE @i <= 10000000
BEGIN
   INSERT INTO table_dummy SELECT HashBytes('MD5',CAST(@i AS VARCHAR(10)))
   SET @i = @i + 1
END
```

Listing 14: Erzeugen einer Dummy-Tabelle

[47] Bei der Arbeit mit Datenbanken wächst in der Regel die Datenbankdatei (durch das Einfügen von Daten und das Anwachsen den Transaktionsprotokolls). Wird Speicher wieder freigegeben (durch Löschen von Objekten und Transaktionsprotokolleinträgen), wird die Datenbankdatei physikalisch nicht wieder verkleinert – der freie Platz wird für die folgenden Transaktionen verwendet. SQL Server bietet aber die Möglichkeit, insofern gewünscht, die Datenbankdateien manuell zu verkleinern („shrink“)

Die `AdventureWorks`-Datenbank hat nach der Vergrößerung eine Dateigröße von 466,75 MB. Es wird folgende Strategie für jedes zu testende Produkt angewandt (die vollständige Sicherung wird hierbei einmal mit maximaler und einmal ohne Komprimierung ausgeführt):

- Löschen einer vorhandenen `AdventureWorks`-Datenbank
- Einhängen einer neuen `AdventureWorks`-Datenbank in SQL Server
- Ausführen einer vollständigen Sicherung
- Einfügen einer neuen Tabelle in die `AdventureWorks`-Datenbank:

```
CREATE TABLE table_diff (
   text VARCHAR(MAX)
)
GO
DECLARE @i INT
SET @i = 1
WHILE @i <= 1000000
BEGIN
   INSERT INTO table_diff SELECT HashBytes('MD5',CAST(@i AS
VARCHAR(10)))
   SET @i = @i + 1
END
```

- Ausführen einer differenziellen Sicherung
- Löschen der `AdventureWorks`-Datenbank
- Restore der vollständigen Sicherung
- Restore der differenziellen Sicherung (wobei hierbei die vollständige Sicherung automatisch erkannt und eingespielt werden soll)
- Object-Level-Recovery der Tabelle `table_dummy` (dazu wird diese gelöscht)
- Object-Level-Recovery der Tabelle `table_diff` aus einer differenziellen Sicherung (dazu wird diese Tabelle gelöscht)

Dabei werden Laufzeiten, Prozessorbelastung und Speicherplatzverbrauch untersucht. Die Ergebnisse sind Näherungswerte über den gesamten Prozess, da es prinzipiell zu zeigen gilt, welche grundsätzlichen Performanzunterschiede es gibt. Getestet wird in einer virtuellen Maschine auf 32-Bit-Basis. Physikalisch handelt es sich um einen Intel Xeon Quad-Core mit 4x 2666 MHz Taktfrequenz. Der virtuellen Maschine werden hierbei 2 CPU-Kerne und 4 GB RAM dediziert zugeteilt, wodurch sich realistische Leistungen messen lassen. Softwarebasis wird Windows Server 2003 Standard und SQL Server 2005 Standard sein.

	Quest LiteSpeed for SQL Server	redgate SQL Backup Pro	Microsoft SQL Server	TSQL-Backup mit Hashs	TSQL-Backup ohne Hashs
vollständige Sicherung ohne Komprimierung					
Laufzeit	0:08 Min.	0:15 Min.	0:11 Min.	32:10 Min.	1:17 Min.
CPU-Auslastung	5%	15%	3%	~45%	~20%
Dateigröße[48]	460 MB	460 MB	455 MB	456 MB + 701 MB[49]	456 MB + 20 MB[49]
vollständige Sicherung mit maximaler Komprimierung					
Laufzeit	1:06 Min.	0:49 Min.	/	37:11 Min.	5:34 Min.
CPU-Auslastung	91%	65%	/	~45%	~20%
Dateigröße[48]	201 MB	188 MB	/	231 MB + 701 MB[49]	231 MB + 20 MB[49]
differenzielle Sicherung ohne Komprimierung					
Laufzeit	0:03 Min.	0:03 Min.	0:03 Min.	18:38 Min	/
CPU-Auslastung	4%	15%	8%	~46%	/
Dateigröße[48]	35 MB	34 MB	31 MB	38 MB + 56 MB[49]	/
vollständige Wiederherstellung					
Laufzeit	0:09 Min.	0:14 Min.	0:12 Min.	4:39 Min	4:37 Min
CPU-Auslastung	7%	16%	8%	~30%	~30%
Wiederherstellung aus differenzieller Sicherung (automatisch inkl. vollständiger Sicherung)					
Laufzeit	/[50]	0:15 Min.	/[50]	5:56 Min.	/[50]
CPU-Auslastung	/	18%	/	~26%	/
Object-Level-Recovery der Tabelle `table_dummy`					
Laufzeit	1:49 Min.	/[51]	/[50]	2:31 Min.	2:27 Min.
CPU-Auslastung	35%	/	/	~40%	~40%
Object-Level-Recovery der Tabelle `table_diff`					
Laufzeit	/[50]	/[50]	/[50]	0:49 Min.	/[50]
CPU-Auslastung	/	/	/	~35%	/

Tabelle 10: Performanzvergleich mit anderen Tools

[48] Dateigrößen basieren auf einem Faktor von 1024 (also 1 MB = 1024 kB)

[49] Der Speicherplatzverbrauch bei Sicherungen mit Hashberechnung umfasst zusätzlich die Hashwerte und das Logbuch in der Metadatenbank

[50] Funktion nicht verfügbar

[51] Aufgrund Speicherplatzmangel auf der Festplatte ist dieses Restore abgebrochen, siehe nachfolgende Bemerkungen

8.3.3. Auswertung

Während der Tests sind einige Besonderheiten aufgetreten, die nachfolgend dokumentiert werden sollen.

Quest LiteSpeed for SQL Server:

- Ein Object-Level-Recovery einer Tabelle mit vorhandenen Schlüsseln wird von Quest derart umgesetzt, dass die Tabelle ohne jegliche Schlüssel wiederhergestellt wird.
- Das Programm benötigt den SQL Befehl `xp_cmdshell` (und entsprechende Berechtigungen) darauf zum Betrieb.

redgate SQL Backup Pro:

- Ein Object-Level-Recovery der Tabelle `table_dummy` brach auf dem Testserver nach 7:26 Minuten mit dem Hinweis ab, dass der Festplattenspeicher nicht mehr ausreicht (Hintergrund: das Programm legt temporäre Daten unter C:\Temp ab, die ein Mehrfaches der Größe der `AdventureWorks`-Datenbank eingenommen haben; hierbei wurden bis zur Speicherplatzerschöpfung 2,91 GB verbraucht).
- Ein weiterer Versuch, die Herstellung der Tabelle `HumanResources.EmployeeAddress` zur Simulation von Schlüsselverletzungen, brach mit dem Fehler ab, dass eine Spalte des Typs `CHAR` nicht in einen `DATETIME`-Wert konvertiert werden kann.
- Prinzipiell sind Object-Level-Recoverys bei diesem Programm nur aus Datenbanksicherungen möglich, die mit SQL Server selbst angefertigt wurden (programmeigene Sicherungen können dafür nicht verwendet werden).

Allgemeine Hinweise:

- Ein Schema-Restore bei den kommerziellen Tools ermöglicht meist nur die Wiederherstellung einer Tabelle, Sicht, Funktion oder Prozedur, nicht aber einer ganzen Datenbank.
- Alle kommerziellen Tools stellen innerhalb von SQL Server Prozeduren zur Verfügung, mit denen man eine Sicherung und Wiederherstellung auch in eigenen SQL Skripten durchführen kann. Diesen Prozeduren ist aber gemeinsam, dass es sich um sogenannte erweiterte gespeicherte Prozeduren handelt, welche letztlich nur eine DLL-Datei des kommerziellen Tools ansprechen – es handelt sich also nicht um eine klassische Prozedur in SQL.

Es ist erkennbar, dass **TSQL-Backup**, insbesondere bei Vorgängen, bei denen die Hashberechnung eine Rolle spielt, deutlich länger brauchte als die kommerziellen Tools. Es gibt einige Faktoren, die dieses Verhalten erklären:

- Kommerzielle Tools arbeiten nativ auf Dateiebene und sichern die Datenbank meist als ganze Einheit – **TSQL-Backup** hingegen exportiert aufwendig jede Tabelle einzeln mittels dem Befehl „bcp“ direkt aus dem SQL Server.
- Kommerzielle Tools können bei entsprechender Softwareentwicklung von Multithreading und Parallelisierung profitieren und damit mehrere Kerne des Prozessors nutzen – eine Transaktion innerhalb von SQL Server wird aber auf nur einem Kern ausgeführt.
- Die eingesetzten Hilfsprogramme „bcp“ und „gzip“ nutzten ebenfalls nur 1 Prozessorkern. „bcp“ erreichte damit beispielsweise nur die Hälfte des Datendurchsatzes der kommerziellen Produkte.

Es ist also erkennbar, dass es sich bei den Performanzunterschieden hauptsächlich um Designprobleme der zugrundeliegenden Anwendungen handelt. SQL Server nutzt Threading für parallele Transaktionen, aber nicht zwingend innerhalb einer Transaktion.

Eine Möglichkeit zur Optimierung von **TSQL-Backup** besteht darin, ein Komprimierungsprogramm einzusetzen, welches Multithreading unterstützt. Hier bietet sich das Programm „7zip“[52] an, welches alle verfügbaren Prozessorkerne nutzen kann. Hierzu musste das Skript von **TSQL-Backup** geringfügig an die Syntax von „7zip“ angepasst werden. Um zu verdeutlichen, welchen Unterschied der Einsatz eines anderen Komprimierungsprogramms ausmacht, wurde zusätzlich zur Komprimierung mit „gzip“ ein vollständiger Sicherungsvorgang ohne Hashberechnung mit „7zip“ auf der `AdventureWorks`-Datenbank durchgeführt:

- „gzip“ erreichte mit maximaler Kompression eine Laufzeit von 05:34 Minuten und eine Backupgröße von 230 MB
- „7zip“ hingegen erreichte mit kleinstmöglicher Kompressionsrate eine Laufzeit von 02:33 Minuten und eine Backupgröße von 201 MB

Also bereits bei minimaler Kompression übertraf „7zip“ den Konkurrenten „gzip“ deutlich in Laufzeit und Speicherplatzverbrauch. Für „bcp“ existiert leider kein adäquater Ersatz.

[52] http://www.7-zip.org/

Letztlich spielt bei **TSQL-Backup** die Hashberechnung eine große Rolle, wenn es um die Laufzeit geht. Für jede zu exportierende Tabelle wird eine eigene Hashtabelle in der Metadatenbank angelegt. Allein für die angelegte Tabelle `table_dummy`, welche der Vergrößerung der Datenbank diente, sind für die 10.000.000 eingefügten Datenzeilen auch 10.000.000 Hashwerte zu erwarten, mit denen SQL umgehen muss – eine nicht unerhebliche Menge, die insbesondere dann kritisch wird, wenn anhand dieser Menge die zu exportierende Tabelle mit der Hashtabelle verknüpft wird (um die zu sichernden Zeilen zu markieren). Hier ist also nicht die eigentliche Datenbankgröße ausschlaggebend[53], sondern die Anzahl der Datenzeilen in den Tabellen. Als handhabbare Größe sollten hier, je nach Leistung des Servers, 100.000 bis 1.000.000 Datenzeilen angesehen werden.

Es muss natürlich auch berücksichtigt werden, dass **TSQL-Backup** bei einem Object-Level-Recovery erst das vollständige Datenbankschema temporär anlegt, um damit Strukturvergleiche mit der Datenbank durchzuführen, in welcher das Objekt wiederhergestellt werden soll. Dies ist ein Prozess, der durch kommerzielle Produkte nicht durchgeführt wird und auch unter Anpassung des Quellcodes von **TSQL-Backup** ausgelassen werden kann. Zur Wahrung der Datenkonsistenz ist dieser Prozess aber unabdingbar. Eine Deaktivierung kann einen Zeitgewinn von 20 bis 40 Sekunden bewirken, dies bringt **TSQL-Backup** insbesondere bei der Wiederherstellung kleinerer Tabellen nach vorn. Den größten Zeitanteil (wie auch bei Sicherungen) nimmt hier aber wieder die Arbeit mit dem Tool „bcp“ ein.

8.4. Anwendungsempfehlung

Es hat sich im vorhergehenden Kapitel gezeigt, dass **TSQL-Backup** aufgrund der Architektur von SQL Server, aber auch der eigenen Arbeitsweise (Hashtabellen werden mit den zu exportierenden Daten verknüpft), von der Performanz deutlich hinter kommerziellen Tools zurückliegt. Der Einsatz eines alternativen Komprimierungsprogramms bringt deutliche Vorteile, insofern die Backupdaten komprimiert werden sollen.

[53] Wie aus Tabelle 10 ersichtlich, liegt die Laufzeit von **TSQL-Backup** ohne Hashberechnung deutlich näher an denen der kommerziellen Produkte

Unter folgenden Voraussetzungen kann ein Einsatz von **TSQL-Backup** empfohlen werden:

bei der Notwendigkeit von Teilsicherungen (differenziell/inkrementell)

- Der Server sollte über ausreichend physikalischen Arbeitsspeicher verfügen, der SQL Server auch zugeteilt werden kann. Hierbei sollten mindestens 4 GB RAM zum Einsatz kommen.
- Tabellen in der zu sichernden Datenbank sollten weniger als 1.000.000 Zeilen enthalten.

wenn Teilsicherungen nicht zum Einsatz kommen sollen (Hashberechnung wird deaktiviert)

- Hier sind bis auf die Datenbankanforderungen aus Abschnitt 7.1.2 keine Einschränkungen vorzunehmen.

Die Leistung des Object-Level-Recovery kann sich, wenn man die Schemawiederherstellung außer Acht lässt, mit Quest LiteSpeed for SQL Server messen. Insbesondere, da durch **TSQL-Backup** auch Konsistenzaspekte beachtet werden, ist ein Einsatz hier empfehlenswert. Weiterhin ist **TSQL-Backup** das einzige Produkt, welches auch ein Object-Level-Recovery aus einer differenziellen Sicherung umsetzt. Das ideale Einsatzszenario ergibt sich somit in der vollständigen Sicherung von Datenbanken ohne Hashberechnung, mit dem Einsatzhintergrund einer selektiven Wiederherstellung von Datenobjekten oder eines Datenbankschemas.

8.5. Einsatz unter SQL Server 2008

TSQL-Backup wurde unter SQL Server 2005 entwickelt, verschiedene Tests unter SQL Server 2008 verliefen aber ebenso erfolgreich. Grundlage für die Tests war ein SQL Server 2008 Enterprise auf Microsoft Windows 2003 Server. Folgende Vorgänge konnten ohne Fehler an der `AdventureWorks`-Datenbank durchgeführt werden:

- Schema aus der Datenbank auslesen
- Vollständige Sicherung mit und ohne Hashberechnung
- Vollständige Sicherung mit Komprimierung
- Wiederherstellung der Datenbank
- Wiederherstellung der Tabelle `HumanResources.EmployeeAddress`

9. Abschließende Betrachtungen

Das Ziel dieser Arbeit war die Erstellung des Programms **TSQL-Backup**. Kapitel 8 hat gezeigt, dass ein funktionsfähiges Programm entstanden ist. Dennoch muss gesagt werden, dass der zeitliche Rahmen dieser Arbeit für den geplanten Funktionsumfang und die damit verbundene Realisierung nicht ausgereicht hat. Nachfolgende Tabelle gibt einen Überblick über die umgesetzten Funktionen (vergleiche Abschnitt 5.2.4):

Funktion	Muss-Kriterium	Soll-Kriterium	Kann-Kriterium	Vollständig umgesetzt	Teilweise umgesetzt	Nicht umgesetzt
/PF010/	X			X		
/PF020/	X			X		
/PF030/			X	X		
/PF040/			X		X	
/PF110/		X		X		
/PF120/		X		X		
/PF130/	X			X		
/PF140/		X		X		
/PF150/		X		X		
/PF210/		X		X		
/PF220/	X			X		
/PF230/		X		X		
/PF240/		X		X		
/PF250/		X		X		
/PF310/		X		X		
/PF320/	X			X		
/PF330/		X		X		
/PF340/	X			X		
/PF350/		X			X	
/PF360/		X			X	

Tabelle 11: Übersicht über umgesetzten Funktionen

Bei der teilweise umgesetzten Funktion /PF040/ handelt es sich um die Voreinstellungen für **TSQL-Backup**, wobei der Komprimierungsgrad noch nicht eingestellt werden kann. Die Funktionen /PF350/ und /PF360/ spezifizierten die Konsistenzprüfung und –korrektur.

Durch **TSQL-Backup** werden bereits folgende Integritätsfaktoren geprüft und korrigiert:

- CHECK-Einschränkungen
- Trigger
- Fremdschlüssel (die auf der wiederherzustellenden Tabelle angelegt wurden)

Offen geblieben ist die Behandlung von Fremdschlüsseln, welche auf anderen Tabellen existieren, und auf die wiederherzustellende Tabelle verweisen. Dieses Thema ist derart komplex [War03 S. 34ff, 92ff] [Fae07 S. 145ff], sodass hierzu erst ausführliche Untersuchungen stattfinden sollten, um auch das Zusammenwirken verschiedener Integritätsfaktoren (siehe Kapitel 4) zu berücksichtigen.

Leider bleibt auch die Performanz hinter den Erwartungen zurück, die Gründe hierfür wurden im Abschnitt 8.3.3 eruiert. Ist der Einsatz der Hashberechnung aber nicht erforderlich, oder wird **TSQL-Backup** auf Datenbanken mit verhältnismäßig kleinen Tabellen eingesetzt (vergleiche Abschnitt 8.4), kann das Programm seine Stärken, insbesondere beim Object-Level-Recovery und der damit verbundenen Konsistenzprüfung und –korrektur, ausspielen.

Unter den Gesichtspunkten einer softwaretechnischen Entwicklung [Sch07] wurde in verschiedenen Phasen die Entwicklung eines Softwareprodukts nachvollzogen. So war es wichtig, den Funktionsumfang während der Produktdefinition abzugrenzen und zu spezifizieren. Schwierigkeit hierbei war, dass keine klassische objektorientierte Entwicklung möglich war und eine Oberflächenentwicklung nicht stattfinden konnte. Die prozedurale Sprache TSQL hat die Entwicklung erschwert. Zudem mussten erst grundlegende Untersuchungen zu bestimmten Funktionalitäten, wie dem Auslesen des Datenbankschemas, den Transaktions-Isolationsstufen und der Hashwert-Berechnung angestellt werden, welche dann erst in die eigentliche Entwicklung einfließen konnten. So haben insbesondere die Kapitel 5 und 6 den Grundstein und die Rahmenbedingungen für die weitere Arbeit gelegt. Ein besonders wichtiger Aspekt war ebenso die umfangreiche Testreihe. Erlernte Methoden des Studiums konnten so angewandt und gefestigt werden.

Folgende Erkenntnisse über **TSQL-Backup** sind zusätzlich während der Entwicklung entstanden:

- Die Verwendung von Hashwerten zum Export von Tabellen hat den Nachteil, dass zwischen Hashberechnung und Export geänderte Datenzeilen nicht erfasst werden können und sich erst beim nächsten Sicherungsvorgang im Backup wiederfinden. Der Grund ist, dass die Snapshot-Isolation zwischen Hasherzeugung und Export unterbrochen werden muss, weil sonst auf die Hashwerte nicht zugegriffen werden kann.

- Ein paralleles Ausführen von Sicherungsvorgängen mit Hashberechnung ist nicht möglich, da es zu Überschneidungen der Snapshot-Isolationsstufe dieser Instanzen kommt, damit innerhalb eines Vorgangs zu Inkonsistenzen der Metadatenbank, und somit N-1 Prozesse (alle bis auf einen) durch SQL Server abgebrochen werden.
- Bei Sicherungen ohne Hashberechnung tritt dieses Problem nicht auf, hierbei wird aber das Auslesen des Datenbankschemas durch mehrere Prozesse jeweils nacheinander ausgeführt, weil durch die Snapshot-Isolation immer nur ein Prozess in der Metadatenbank schreiben darf. Im Anschluss, beim Export der Datenbestände, werden aber alle Prozesse wieder parallel ausgeführt, so dass hier effektiv eine bessere Nutzung von mehreren CPU-Kernen möglich ist und entgegen Abschnitt 8.3.3 sogar ein Performanz-Gewinn erreicht werden kann. Bei Einsatz einer 2-Kern-CPU können so zwei parallele Sicherungsvorgängen um den Faktor zwei beschleunigt werden. Es kann zwar so kein einzelner Sicherungsvorgang beschleunigt werden, wohl aber der Gesamtprozess bei gleichzeitiger Durchführung mehrerer Datenbank-sicherungen.
- Zur Hashberechnung wird über alle Spalten einer Tabelle eine Zeichenkette gebildet, welche dann in einem `EXEC`-Befehl ausgeführt wird. Hierbei kann bei vielen Spalten eine sehr lange Zeichenkette entstehen. SQL Server 2005 bietet mit dem Datentyp `VARCHAR(MAX)` die Möglichkeit mit diesem Befehl Zeichenketten mit einer Größe von fast 2 GB auszuführen. Sobald aber in dieser Zeichenkette Variablen eingebunden werden, die nicht vom Typ `VARCHAR(MAX)` sind, wird die ausführbare Länge der Zeichenkette auf 8000 Zeichen begrenzt – dies ist ein Relikt aus älteren Versionen von SQL Server. Um also die maximale Zeichenlänge in `EXEC` verwenden zu können, müssen alle verwendeten Variablen innerhalb dieser Zeichenkette vom Typ `VARCHAR(MAX)` sein.

TSQL-Backup hat einen Grad an Reife erreicht, der einen produktiven Einsatz ermöglicht. Damit ist auch ein Anreiz gegeben, das Programm um noch fehlende, wünschenswerte oder aufgrund Zeitmangels nicht mehr umsetzbare Funktionen zu erweitern:

- Eine mehrfache Ausführung von **TSQL-Backup** zur Sicherungen mit Hashberechnung sollte verhindert werden.
- Es sind weitere Prüfungen und Tests bei Ausführungsbeginn von **TSQL-Backup** durchzuführen. So müssen unter anderem Schreibrechte im Backup-Speicherort und die physikalische Existenz von DPW und „gzip“ geprüft werden.
- Eine Reaktion auf Fehler von der Kommandozeile, beispielsweise bei der Ausführung von „gzip“, wäre wünschenswert.

- Über einen optionalen Aufrufparameter und die Voreinstellungen sollte der Komprimierungsgrad für „gzip“ einstellbar sein.
- In den Voreinstellungen war bereits die Möglichkeit vorgesehen, DPW nicht nur über eine vertrauenswürdige Verbindung auf SQL Server zugreifen zu lassen (diese Art der Verbindung ist nicht immer möglich), sondern ein Zugriff sollte auch explizit über Nutzername und Passwort erfolgen – hierzu sind Anpassungen erforderlich.
- Eine wünschenswerte Funktion ist auch, bei der Wiederherstellung einer Tabelle einen weiteren Parameter (beispielsweise `restore_depencies`) einzuführen, der abhängige Datenbankobjekte vom wiederherzustellenden Element auf Basis von Trigger und Schlüsselbeziehungen erkennt und deren Wiederherstellung automatisch durchführt.
- Ein Vergleich von ausgeführten Sicherungen mit der Datenbank, beziehungsweise zwischen wiederhergestellter Datenbank und Backup, sollte umgesetzt werden.
- Es sollte untersucht werden, ob ein schnelleres und platzsparenderes Hashverfahren eingesetzt werden kann, um die Metadatenbank klein zu halten.

In Abschnitt 7.1.2 wurde bereits angesprochen, dass **TSQL-Backup** in der Umgebung von Microsoft SQL Server entwickelt wurde. Hintergrund dabei ist unter anderem, dass sehr nah am Betriebssystem gearbeitet wird. So kann im Rahmen dieser Arbeit der Einsatz von Hilfsprogrammen nur durch einen Aufruf der Windows-Kommandozeile durch die erweiterte Prozedur `xp_cmdshell` innerhalb von TSQL erfolgen. Weiterhin werden sehr spezifische Funktionen von TSQL verwendet, wie die zur Hashberechnung oder auch die Snapshot-Isolationsstufe. Eine Portierung auf andere Datenbankmanagementsysteme – zu den bekannten Produkten gehören Oracle Database und MySQL – sowie andere Betriebssysteme ist nicht möglich. In **TSQL-Backup** kommen spezifische TSQL-Funktionen zum Einsatz, für die möglicherweise ein Pendant im anderen Datenbankmanagementsystem gefunden werden kann, beispielsweise für die Hashberechnung und den Einsatz der Windows Kommandozeile durch `xp_cmdshell`. Aber selbst die SQL-Syntax differiert zwischen den Systemen. Für den Abruf der ersten N Zeilen aus einer Tabelle sieht der SQL-Befehl wie folgt aus:

Microsoft SQL Server

```
SELECT TOP N * FROM <table>
```

Oracle

```
SELECT * FROM <table> WHERE rownum<=N
```

MySQL

```
SELECT * FROM <table> LIMIT N
```

Weiterhin unterscheidet sich auch grundsätzlich die Verwaltung der Datenbankschemata von System zu System [Fae07 S. 279ff], so dass hier grundlegende Untersuchungen zum Auslesen des Schemas erfolgen müssen. Somit ist zu erwarten, dass ein Wechsel des Datenbankmanagementsystems eine vollständige und aufwändige Überarbeitung des Quellcodes erfordert. Eine Portierung auf ein anderes Betriebssystem erfordert neben der Anpassung des Quellcodes bezüglich des Aufbaus von Verzeichnispfaden auch einen Ersatz für alle Programme, die an der Windows Kommandozeile ausgeführt werden. Dies betrifft hauptsächlich DPW, da dieses nur unter Microsoft Windows lauffähig ist.

TSQL-Backup kann unter Berücksichtigung der Anforderungen aus Abschnitt 7.1.2 und unter Ausschluss der Express-Edition von SQL Server 2008 (vergleiche Abschnitt 5.2.10) auch unter Microsoft SQL Server 2008 eingesetzt werden (vergleiche Abschnitt 8.5). Ein Betrieb in zukünftigen Versionen von SQL Server ist möglich, insofern DPW eingesetzt werden kann und die Strukturen von Systemtabellen gleich bleiben. Dies ist notwendig, da Systemtabellen für die Konsistenzprüfung (siehe auch Listings aus Kapitel 4) abgefragt werden. Weitere Einschränkungen sind nicht zu erwarten.

TSQL-Backup hat gezeigt, dass es fehlende Funktionen in existierenden Backup-Programmen ersetzen kann. Der offene Quellcode ermöglicht darüber hinaus weitere Anpassungen und die Implementation zusätzlicher Funktionen, insbesondere zur Konsistenzwahrung. Inhalte für weitere Projekt- oder Abschlussarbeiten können sein:

- Untersuchung von Faktoren zur Wahrung der referenziellen Integrität bei Object-Level-Recovery und Implementation einer Korrekturroutine in TSQL
- Erarbeitung und Umsetzung eines Prüfalgorithmus zur Verifikation von Sicherungen und Wiederherstellungen auf Basis von **TSQL-Backup**

Das Programm eignet sich somit zur Erweiterung bestehender Sicherungsstrategien. Insbesondere die Anforderung des Restores der eingangs beispielhaft erwähnten Lookup-Tabellen in einem Data Warehouse kann erfüllt werden. Weiterhin wird der wichtige Aspekt der Beachtung und Korrektur von Schlüsselverletzungen bei einem Wiederherstellungsvorgang gewährleistet – dies konnte bisher kein kommerzieller Anbieter umsetzen (siehe Abschnitt 8.3.1).

Anhang – Syntaxerklärung für TSQL-Backup

Zeile	Parameter	Wert	Bedeutung
3	action	get_settings	Ausgabe der Voreinstellungen
4	action	set_setting	Verändern der Voreinstellungen
5	action	schema_from_db	Datenbankschema auslesen
5	db	(STRING)	Name der auszulesenden Datenbank
6	action	backup	Sicherung einer Datenbank
6	db	(STRING)	Name der zu sichernden Datenbank
7	action	list	Ausgabe aller vorhandenen Backups
8	action	show_log	Log zu einem Vorgang betrachten
8	id	(INT)	ID des Vorgangs (erhältlich durch action=list)
9	action	purge	Löschen von Backups
10	action	list_elements	Elements in einem Backup auflisten
10	id	(INT)	ID des zu verwendenden Backups
11	action	schema_to_db	Schema aus einem Backup wiederherstellen
11	id	(INT)	ID des zu verwendenden Backups
11	db	(STRING)	Name der Datenbank, in die das Schema wiederhergestellt werden soll
12	action	restore	Wiederherstellung einer Datenbank oder eines Elements aus einer Datenbank
12	id	(INT)	ID des zu verwendenden Backups
12	db	(STRING)	Name der Datenbank, in der das Backup oder Element wiederhergestellt werden soll
15	save_path	(STRING)	Physischer Speicherort für Backups
16	schema_login	(user,win)	Verbindungsmethode für DPW (Login mit Zugangsdaten oder vertrauenswürdige Verbindung)
17	schema_script	(STRING)	Pfad zu DPW
18	compress	(yes,no)	Voreinstellung zur Komprimierung von Backups

19	zip_path	(STRING)	Pfad zu gzip
22	schema_pw	(STRING)	Passwort für Login mit Zugangsdaten für DPW
23	schema_user	(STRING)	Benutzername für Login mit Zugangsdaten für DPW
26	type	(full,incr,diff)	Art des Backups (vollständig, inkrementell oder differenziell)
27	quick	(yes,no)	Schnelles Backup (es werden keine Hashs erzeugt)
28	compress	(yes,no)	Komprimieren des Backups (überschreibt Voreinstellung)
29	schema_pw	(STRING)	Passwort für Login mit Zugangsdaten für DPW
30	schema_user	(STRING)	Benutzername für Login mit Zugangsdaten für DPW
33	db	(STRING)	Datenbankname, zu dem alle Backups ausgegeben werden sollen
36	force	(yes,no)	Löschen von Backups erzwingen
37	db	(STRING)	Löschen von Backups zu einer bestimmten Datenbank
38	keep_last_full	(INT)	Erhält beim Löschen die letzten X vollständigen Backups
39	id	(INT)	Löschen eines bestimmten Backups (ID des Backups)
40	keep_schema	(yes,no)	Erhält beim Löschen von Backups reine Schema-Backups
43	force	(yes,no)	Erzwingen der Wiederherstellung eines Datenbankschemas
46	force	(yes,no)	Erzwingen der Wiederherstellung einer Datenbank oder eines Elements
47	element_id	(INT)	ID des Elements, welches wiederhergestellt werden soll
50	consistence	(handle,ignore)	Behandlung von auftretenden Konsistenzproblemen bei der Wiederherstellung

Anhang – Schnittstellenbeschreibung

newBackup

Diese Prozedur stellt das Hauptprogramm von **TSQL-Backup** dar und wird in Abhängigkeit der übergebenen Parameter die passenden Unterprozeduren aufrufen. Für jeden Aufruf werden hier bereits die Prozeduren zur Argumentprüfung und die Prüfung von Systemparametern durchgeführt.

Parameter	Art	Datentyp	Beschreibung
action	intern	VARCHAR(MAX)	an das Hauptprogramm übergebene, auszuführende Funktion von **TSQL-Backup**
log_id	intern	INT	ID des aktuellen Logs
backup_id	intern	INT	ID des aktuellen/zu verwendenden Backups
backup_type	intern	VARCHAR(255)	Typ des Backups ("`schema`", "`full`", "`incr`", "`diff`")
save_path	intern	VARCHAR(MAX)	Speicherort der Backupdateien
zip_path	intern	VARCHAR(MAX)	Speicherort von gzip
ptrval	intern	BINARY(16)	Zeiger für die Arbeit mit `TEXT`-Feldern

newBackupBackup

Das Kernstück von **TSQL-Backup** bildet die eigentliche Backupfunktionalität, welche in dieser Prozedur bereitgestellt wird. Dies umfasst die vollständige, differenzielle und inkrementelle Sicherung der Dateninhalte.

Parameter	Art	Datentyp	Beschreibung
log_id	Eingabe	INT	ID des aktuellen Logs
backup_id	Eingabe	INT	ID des aktuellen Backups
db_name	Eingabe	VARCHAR(255)	Name der Datenbank, die gesichert werden soll
backup_type	Eingabe	VARCHAR(255)	Typ des Backups ("`schema`", "`full`", "`incr`", "`diff`")
save_path	Eingabe	VARCHAR(MAX)	Speicherort der Backupdateien
zip_path	Eingabe	VARCHAR(MAX)	Speicherort von gzip

parent_id	intern	INT	ID des übergeordneten Backups (für diff./inkr. Backups)

newBackupBackupList

Hiermit können alle von **TSQL-Backup** erzeugten Backups ausgegeben werden.

Parameter	**Art**	**Datentyp**	**Beschreibung**
log_id	Eingabe	INT	ID des aktuellen Logs
db_name	Eingabe	VARCHAR(255)	Name der Datenbank, zu welcher Backups ausgegeben werden sollen

newBackupCheckArguments

Zur Gewährleistung der korrekten Funktionsweise von **TSQL-Backup** überprüft diese Prozedur alle übergebenen Argumente auf syntaktische Korrektheit – es handelt sich also um einen Parser für die Argumente-Zeichenkette – und veranlasst im Fehlerfall die Ausgabe der Syntax.

Parameter	**Art**	**Datentyp**	**Beschreibung**
log_id	Eingabe	INT	ID des aktuellen Logs
arguments	Eingabe	VARCHAR(MAX)	an das Hauptprogramm übergebene Parameterkette

newBackupColToBin

Die Funktion zur Erzeugung von Hashwerten erwartet die Eingabe der Daten in einem bestimmten Datentyp. Sie gibt, wenn nötig, eine Zeichenkette zur Umwandlung des übergebenen Spaltentyps in das notwendige Format `VARBINARY` zurück.

Parameter	**Art**	**Datentyp**	**Beschreibung**
column_name	Eingabe	NVARCHAR(128)	Name der Spalte, welche umgewandelt werden soll
column_type	Eingabe	NVARCHAR(128)	aktueller Typ der Spalte
return	Ausgabe	NVARCHAR(255)	Rückgabe einer SQL-Zeichenkette zur weiteren Verarbeitung in einem SQL-Befehl, welcher die Spalte entsprechend umwandelt

newBackupCreateDB

DPW kann lediglich die Schemaelemente einer Datenbank auslesen, aber nicht das Skript zur Erzeugung der Datenbank selbst. Mit dieser Prozedur wird ein solches `CREATE DATABASE`-Skript erzeugt.

Parameter	Art	Datentyp	Beschreibung
log_id	Eingabe	INT	ID des aktuellen Logs
backup_id	Eingabe	INT	ID des aktuellen Backups
db_name	Eingabe	VARCHAR(255)	Name der Datenbank, für welche das Skript erzeugt werden soll
script	intern	VARCHAR(MAX)	Hilfsvariable, die letztlich das vollständige `CREATE`-Skript enthält
ptrval	intern	BINARY(16)	Zeiger für die Arbeit mit `TEXT`-Feldern

newBackupDataRestore

Diese Prozedur spielt alle Tabelleninhalte (also auch zugehörige differenzielle und inkrementelle Backups) in eine angegebene Datenbank ein und bildet damit einen Teil der Restore-Funktion. Dabei ist die Wiederherstellung aller Daten einer Datenbank möglich, aber auch die Wiederherstellung eines einzelnen Elementes.

Parameter	Art	Datentyp	Beschreibung
log_id	Eingabe	INT	ID des aktuellen Logs
backup_id	Eingabe	INT	ID des zu verwendenden Backups
db_name	Eingabe	VARCHAR(255)	Name der Datenbank, in welcher die Daten wiederhergestellt werden sollen
save_path	Eingabe	VARCHAR(MAX)	Speicherort der Backupdateien
zip_path	Eingabe	VARCHAR(MAX)	Speicherort von gzip
element_id	intern	INT	Hilfsvariable, die ungleich 0 ist, falls ein bestimmtes Element (Tabelle) angegeben wurde, welches wiederhergestellt werden soll

newBackupDBRestore

Mit dieser Prozedur kann ein Datenbankschema vollständig wieder eingespielt werden. Sie stellt somit auch ein Teil der Restore-Funktion dar.

Parameter	Art	Datentyp	Beschreibung
log_id	Eingabe	INT	ID des aktuellen Logs
backup_id	Eingabe	INT	ID des zu verwendenden Backups
db_name	Eingabe	VARCHAR(255)	Name der Datenbank, in welcher das Schema wiederhergestellt werden soll

newBackupGetSchema

Mit dieser Prozedur wird das Datenbankschema ausgelesen. Damit können alle Schemaelemente einer Datenbank als SQL-Skript erzeugt werden. Sie stellt einen Teil der Backup-Funktion dar.

Parameter	Art	Datentyp	Beschreibung
log_id	Eingabe	INT	ID des aktuellen Logs
backup_id	Eingabe	INT	ID des aktuellen Backups
db_name	Eingabe	VARCHAR(255)	Name der Datenbank, für welche das Schema ausgelesen werden soll
save_path	Eingabe	VARCHAR(255)	Speicherort der Backupdateien
script	intern	VARCHAR(MAX)	Speicherort von DPW
object_id	intern	INT	eindeutige ID des Schema-Objektes
ptrval	intern	BINARY(16)	Zeiger für die Arbeit mit `TEXT`-Feldern

newBackupHashData

Dies soll eine echte Funktion sein und den Hashwert zu einer übergebenen Zeichenkette erzeugen.

Parameter	Art	Datentyp	Beschreibung
algorithm	Eingabe	NVARCHAR(4)	Algorithmus, der für die Hashfunktion verwendet werden soll
input_data	Eingabe	VARBINARY(MAX)	Zeichenkette, welche umgewandelt werden soll
index	intern	INT	Hilfsvariable zur Arbeit mit Zeichenketten, die länger als 8000 Zeichen sind

input_data_length	intern	INT	Hilfsvariable zur Arbeit mit Zeichenketten, die länger als 8000 Zeichen sind
return_sum	Ausgabe	VARBINARY(MAX)	berechneter Hashwert für die Zeichenkette

newBackupIsolation

Hiermit kann ein Snapshot für eine Datenbank aktiviert werden. Dies ist notwendig, um die Konsistenz beim Auslesen des Datenbankschemas oder der Erstellung eines Backups zu gewährleisten.

Parameter	**Art**	**Datentyp**	**Beschreibung**
log_id	Eingabe	INT	ID des aktuellen Logs
db_name	Eingabe	VARCHAR(255)	Name der Datenbank, für welche ein Snapshot aktiviert werden soll
snapshot_state	Ausgabe	INT	Flag über den Zustand des Snapshot-Modus

newBackupKeyConsistence

Zur Wahrung der Konsistenz von Schlüsselbeziehungen und Triggern bei einem Restore soll diese Prozedur bestimmte Prüfungen und Korrekturen durchführen. Dazu gehört die Prüfung, ob bei einem Fremdschlüssel in der Zieltabelle alle Spalten enthalten sind, oder auch das Einfügen von Primärschlüsseln zur Wahrung der Integrität. Diese Prozedur ist Teil der Restore-Funktion.

Parameter	**Art**	**Datentyp**	**Beschreibung**
log_id	Eingabe	INT	ID des aktuellen Logs
db_name	Eingabe	VARCHAR(255)	Name der Zieldatenbank, in der die Konsistenzprüfungen laufen sollen
tmp_db_name	Eingabe	VARCHAR(255)	Name der temporären Schema-Datenbank, welche als Referenz für Schema-Prüfungen verwendet wird
type	Eingabe	VARCHAR(255)	Typ des zu prüfenden Elements ("`all`", "`fk`", "`trigger`", "`check`")
do	Eingabe	VARCHAR(255)	Flag, welches angibt, ob die Konsistenz nur geprüft werden soll, oder ob auch Korrekturen vorgenommen werden sollen
success	Ausgabe	INT	Flag, welches den Erfolg der Aktion zurückgibt

newBackupKeyTrigger

Diese Prozedur hilft, alle vorhandenen Schlüsselbeziehungen und Trigger in einer Datenbank oder für ein Element temporär zu aktivieren oder zu deaktivieren, und ist ebenfalls Teil der Restore-Funktion.

Parameter	**Art**	**Datentyp**	**Beschreibung**
log_id	Eingabe	INT	ID des aktuellen Logs
db_name	Eingabe	VARCHAR(255)	Name der Datenbank, in der Schlüssel und Trigger de-/aktiviert werden sollen
suffix	Eingabe	VARCHAR(255)	wird genutzt, um die Art der Schlüssel und Trigger anzugeben, welche de-/aktiviert werden sollen (beispielsweise "`_ref`" für verweisende Schlüssel auf ein Objekt)
do	Eingabe	VARCHAR(255)	gibt an, ob Schlüssel /Trigger aktiviert oder deaktiviert werden sollen
success	Ausgabe	INT	Flag, welches den Erfolg der Aktion zurückgibt

newBackupPreDBInit

Wenn ein Backup erstellt oder ein Datenbankschema ausgelesen werden soll, dient diese Prozedur dazu, alle notwendigen Vorbereitungen zu treffen. Dazu gehört, in der Metadatenbank einen entsprechenden Eintrag für das Backup zu erzeugen und somit eine `backup_id` zu erhalten.

Parameter	**Art**	**Datentyp**	**Beschreibung**
backup_type	Eingabe	VARCHAR(255)	Art des Backups ("`schema`", "`full`", "`diff`", "`incr`")
desc	Eingabe	VARCHAR(MAX)	Beschreibung für den aktuellen Backupvorgang
log_id	Eingabe	INT	ID des aktuellen Logs
db_name	Eingabe	VARCHAR(255)	Name der zu sichernden Datenbank
backup_id	Ausgabe	INT	eindeutige ID für das aktuell ausgeführte Backup
snapshot_state	Ausgabe	INT	Flag über den Zustand des Snapshot-Modus

newBackupPrePathInit

Für viele Funktionen in **TSQL-Backup** ist es notwendig, dass bestimmte Speicherpfade korrekt sind. Dazu gehört die Übergabe des physischen Pfads von DPW und von gzip an das Hauptprogramm, aber auch die Erzeugung des physischen Pfads zur Speicherung der Daten, wenn ein Backup erstellt wird.

Parameter	Art	Datentyp	Beschreibung
log_id	Eingabe	INT	ID des aktuellen Logs
save_path	Ausgabe	VARCHAR(MAX)	aus Voreinstellungen ausgelesener Speicherort der Backupdateien
zip_path	Ausgabe	VARCHAR(MAX)	aus Voreinstellungen ausgelesener Speicherort von gzip

newBackupPurge

Zur Löschung von alten Backups aus **TSQL-Backup** stellt diese Prozedur alle Funktionen bereit.

Parameter	Art	Datentyp	Beschreibung
log_id	Eingabe	INT	ID des aktuellen Logs
save_path	Eingabe	VARCHAR(MAX)	Speicherort der Backupdateien
keep_last_full	intern	INT	enthält den Wert der letzten X zu erhaltenden vollständigen Backups (in der Regel 0)
keep_schema	intern	VARCHAR(25)	Flag, ob Schema-Backups erhalten bleiben sollen (in der Regel 0)
backup_id	intern	INT	Hilfsvariable, die ungleich 0 ist, falls als Parameter beim Programmaufruf ein bestimmtes Backup angegeben wurde, welches gelöscht werden soll
db_name	intern	VARCHAR(255)	Hilfsvariable, die nicht leer ist, falls als Parameter beim Programmaufruf eine bestimmte Datenbank angegeben wurde, zu welcher alle Backups gelöscht werden sollen

newBackupRestore

Diese Prozedur stellt die Funktionalität zur Wiederherstellung eines einzelnen Datenbankschemas, einer ganzen Datenbank oder eines bestimmten Elementes aus einer

Datenbank zur Verfügung. Ein Grundgedanke ist, dass bei einem Restore das ausgelesene Schema eines Backups hilfsweise in eine temporäre Datenbank eingespielt wird, um dann mithilfe dieser Datenbank das Schema von Objekten zu vergleichen und auch Schlüsselbeziehungen und Trigger zu finden.

Parameter	**Art**	**Datentyp**	**Beschreibung**
log_id	Eingabe	INT	ID des aktuellen Logs
restore_type	Eingabe	VARCHAR(255)	Art der Wiederherstellung (vollständiges Restore oder Wiederherstellung eines Datenbankschemas)
db_name	Ausgabe	VARCHAR(255)	Ausgabe des Namens der Datenbank, in welcher die Wiederherstellung stattgefunden hat
save_path	Eingabe	VARCHAR(MAX)	Speicherort der Backupdateien
zip_path	Eingabe	VARCHAR(MAX)	Speicherort zu gzip
ptrval	intern	BINARY(16)	Zeiger für die Arbeit mit `TEXT`-Feldern
backup_id	intern	INT	verwendetes Backup zur Wiederherstellung
tmp_backup_id	intern	INT	Hilfsvariable für den Vergleich von Schemaobjekten
tmp_db_name	intern	VARCHAR(255)	Name der Datenbank, in welcher das Schema des Backups hilfsweise eingespielt wurde
snapshot_state	intern	INT	Flag über den Zustand des Snapshot-Modus
success	intern	INT	Flag, welches den Erfolg von `newBackupKeyTrigger` anzeigt

`newBackupSyntaxFromDB`

Die Syntax von **TSQL-Backup** wird rekursiv in der Datenbank gespeichert (siehe Abschnitt 6.8). Diese Prozedur erzeugt in Abhängigkeit vom übergebenen Elternknoten die zugehörige Syntax.

Parameter	**Art**	**Datentyp**	**Beschreibung**
depends_on	Eingabe	INT	Elternknoten der auszugebenden Elemente; Einstieg für gewöhnlich über die Wurzel (Element 0)
textTop	Ausgabe	VARCHAR(MAX)	Kopf der Syntax (entsteht durch rekursiven Aufruf)

textBottom	Ausgabe	VARCHAR(MAX)	Rest der Syntax (entsteht durch rekursiven Aufruf)
textTopSub	intern	VARCHAR(MAX)	Hilfsvariable für rekursiven Aufruf
textBottomSub	intern	VARCHAR(MAX)	Hilfsvariable für rekursiven Aufruf
valuesPrefix	intern	CHAR(2)	jeder Parameter erscheint in Anlehnung an TSQL auf einer eigenen Zeile, diese Variable steuert den vor dem Parameter auszugebenden Text

newBackupSyntaxOut

Die Ausgabe der Syntax von **TSQL-Backup** soll syntaktisch an TSQL angelehnt sein. Dazu dient diese Prozedur. Die benötigten Daten werden von `newBackupSyntaxFromDB` bereit gestellt.

Parameter	**Art**	**Datentyp**	**Beschreibung**
log_id	Eingabe	INT	ID des aktuellen Logs
type	Eingabe	VARCHAR(25)	Information welche Syntax ausgegeben werden soll; prinzipiell wird die vollständige Syntax ausgegeben; über diesen Parameter liese sich steuern, ob nur ein bestimmter Ausschnitt aus der Syntax angezeigt werden soll
textTop	intern	VARCHAR(MAX)	enthält den Kopf der Syntax
textBottom	intern	VARCHAR(MAX)	enthält den Rest der Syntax

newBackupTextOut

Diese Prozedur dient der Textausgabe innerhalb der SQL Befehlszeile und einen zur Textausgabe identischen Eintrag im Log. Es wird somit kein `PRINT`-Befehl in SQL ausgeführt, sondern an jeder Stelle, an der Text ausgegeben werden soll, die Prozedur aufgerufen.

Parameter	**Art**	**Datentyp**	**Beschreibung**
log_id	Eingabe	INT	ID des aktuellen Logs
text	Eingabe	VARCHAR(MAX)	auszugebender Text
type	Eingabe	VARCHAR(25)	Logstufe (Klassifizierung in beispielsweise "`info`", "`warning`", "`error`")

Anhang – Struktogramm für newBackupGetSchema

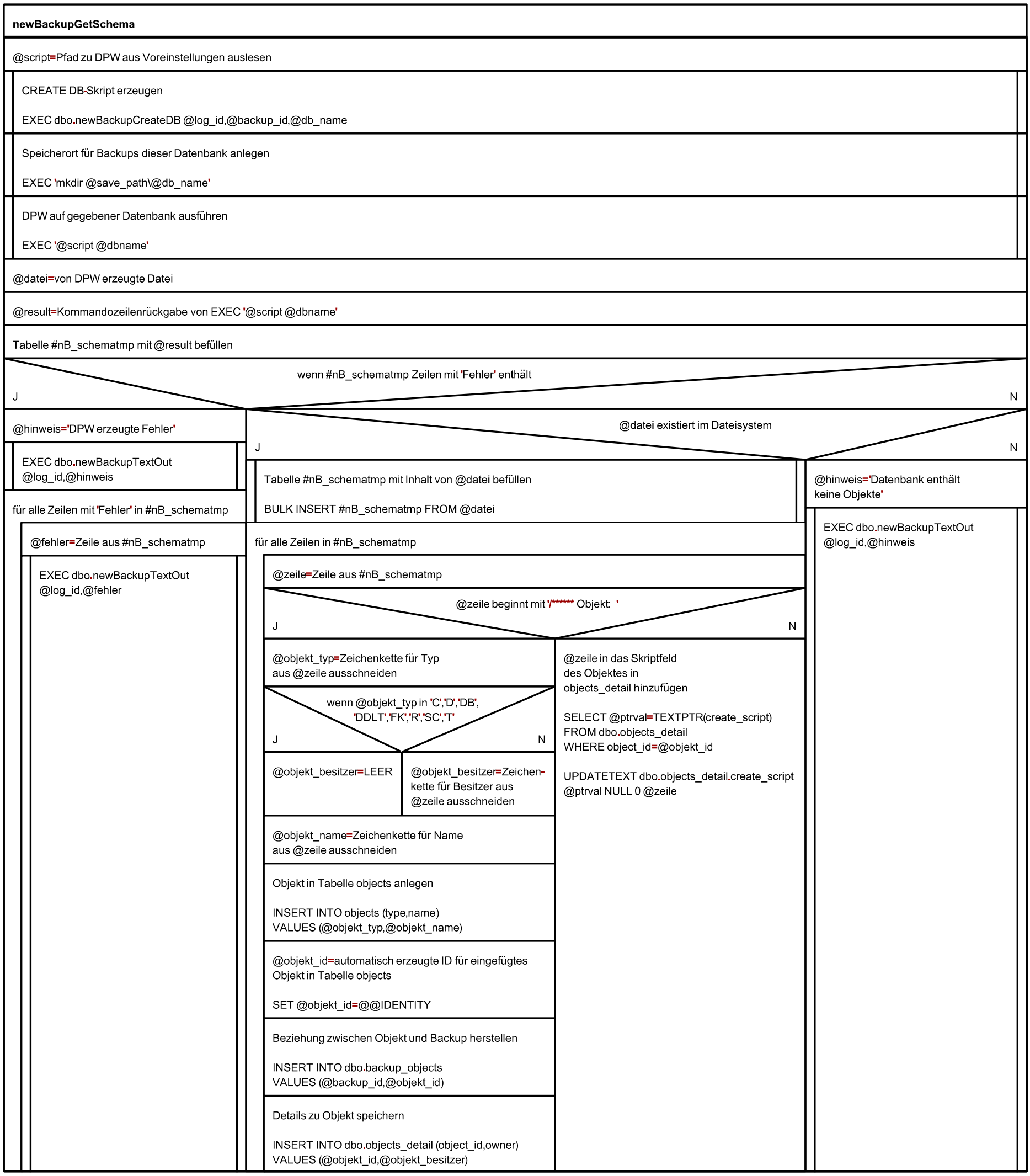

Abbildung 18: Struktogramm für newBackupGetSchema

Anhang – Struktogramm für newBackupBackup

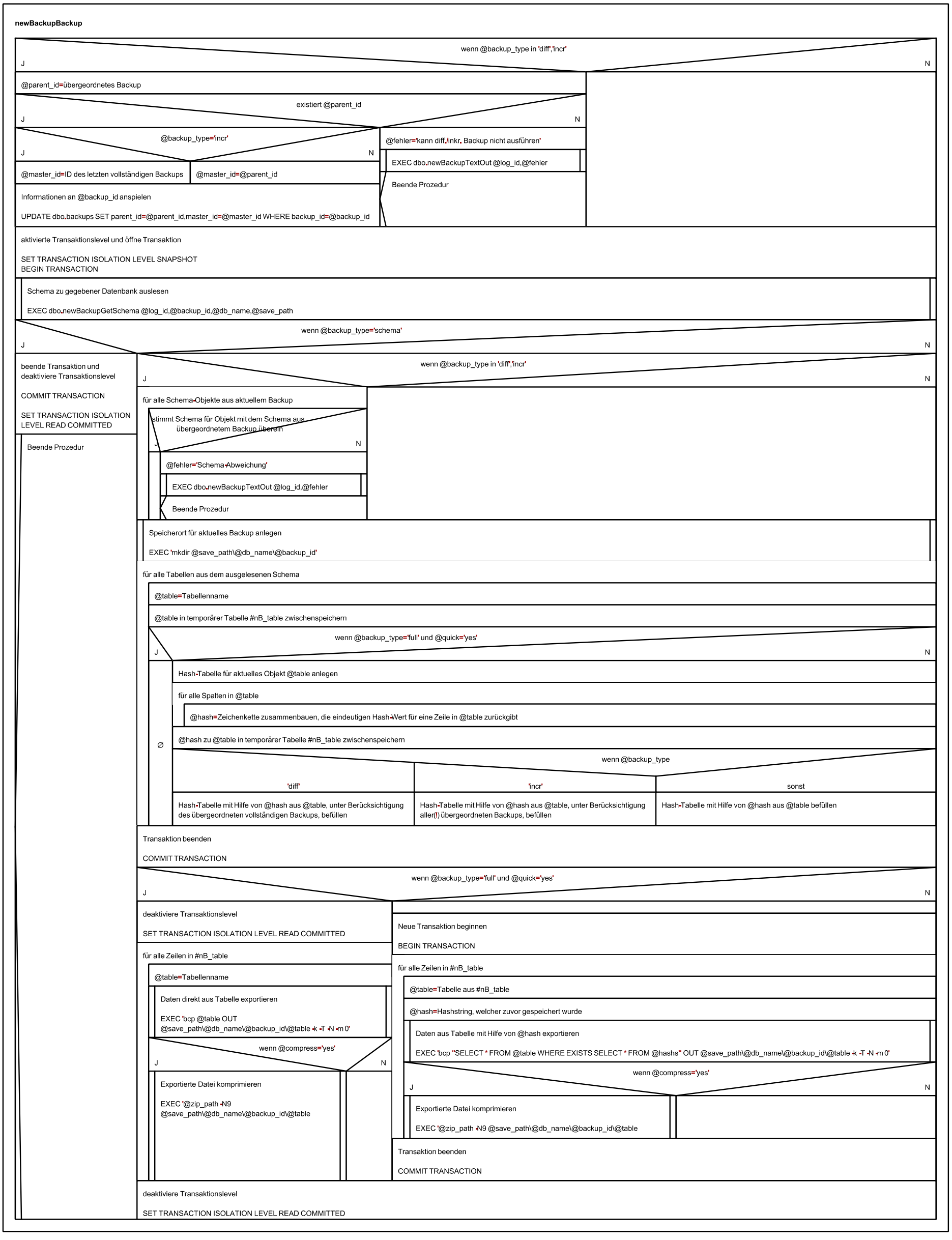

Abbildung 19: Struktogramm für newBackupBackup

Anhang – Testszenarien in TSQL-Backup

Szenario	korrektes Verhalten
Aufruf ohne Parameter	ja

Programmaufruf

```
newBackup
```

Ausgabe

```
no arguments given
syntax: […]
```

Szenario	korrektes Verhalten
Aufruf mit fehlender Berechtigung	ja

Programmaufruf

```
newBackup 'action=list'
```

Ausgabe

```
Execute only as 'sa'
```

Szenario	korrektes Verhalten
Aufruf mit fehlenden Systemeinstellungen (I)	ja

Programmaufruf

```
newBackup 'action=list'
```

Ausgabe

```
Please set option 'show advanced options' to 1 via sp_configure:
EXEC master.dbo.sp_configure 'show advanced options', 1
RECONFIGURE
```

Szenario	korrektes Verhalten
Aufruf mit fehlenden Systemeinstellungen (II)	ja

Programmaufruf

```
newBackup 'action=list'
```

Ausgabe

```
Please set option 'xp_cmdshell' to 1 via sp_configure:
EXEC master.dbo.sp_configure 'xp_cmdshell', 1
RECONFIGURE
```

Szenario	korrektes Verhalten
Aufruf mit fehlendem Parameter	ja

Programmaufruf

```
newBackup 'action=backup'
```

Ausgabe

```
missing expected arguments for given parameters:
   param: action | value: backup
syntax: […]
```

Szenario	korrektes Verhalten
Aufruf mit mehrfachen Parametern	ja

Programmaufruf

```
newBackup 'action=backup,db=AdventureWorks,db=OtherDb'
```

Ausgabe

```
parameters given multiply:
   param: db
```

Szenario	korrektes Verhalten
Aufruf mit ungültigem Parameter	ja

Programmaufruf

```
newBackup 'action=backup,some=parameter'
```

Ausgabe

```
disallowed parameter settings:
   param: some | disallowed value: parameter
syntax: […]
```

Szenario	korrektes Verhalten
Aufruf mit falschem Werttyp	ja

Programmaufruf

```
newBackup 'action=set_setting,compress=1'
```

Ausgabe

```
wrong data type for parameter values:
   param: compress has to be {yes|no}
syntax: […]
```

Szenario	korrektes Verhalten
Aufruf mit fehlendem Pfad zu DPW	ja

Programmaufruf

```
newBackup 'action=backup,db=AdventureWorks'
```

Ausgabe

```
no save path for backup files given, please set one
syntax: […]
```

Szenario	korrektes Verhalten
Aufruf mit fehlendem Pfad zu gzip	ja

Programmaufruf

```
newBackup 'action=backup,db=AdventureWorks'
```

Ausgabe

```
no path to app gzip given, please set one
syntax: […]
```

Szenario	korrektes Verhalten
Aufruf mit nicht existierender Datenbank	ja

Programmaufruf

```
newBackup 'action=backup,db=NonExistent'
```

Ausgabe

```
database 'NonExistent' does not exist
```

Szenario	korrektes Verhalten
Aufruf mit nicht existierender Backup-ID	ja

Programmaufruf

```
newBackup 'action=list_elements,id=-1'
```

Ausgabe

```
backup for given ID -1 does not exist
```

Anhang – Durchgeführte Sicherungsvorgänge

Szenario

Sicherung des Datenbankschemas

Programmaufruf

```
newBackup 'action=schema_from_db,db=AdventureWorks'
```

Ausgabe

```
using backup id 1
generate script for database 'AdventureWorks'
script for database 'AdventureWorks' (id 1) saved
getting database schema for 'AdventureWorks'
schema for object 'AdventureWorks' (id 2) saved
schema for object 'HumanResources' (id 3) saved
schema for object 'Person' (id 4) saved
[…]
```

Szenario

Vollständige Sicherung ohne Hash-Erzeugung

Programmaufruf

```
newBackup 'action=backup,db=AdventureWorks,quick=yes'
```

Ausgabe

```
using backup id 2
generate script for database 'AdventureWorks'
script for database 'AdventureWorks' (id 533) saved
getting database schema for 'AdventureWorks'
schema for object 'AdventureWorks' (id 534) saved
schema for object 'HumanResources' (id 535) saved
schema for object 'Person' (id 536) saved
[…]
export data for tables in 'AdventureWorks'
data for table 'AdventureWorks.dbo.AWBuildVersion' (id 554) exported
data for table 'AdventureWorks.Production.TransactionHistoryArchive' (id
555) exported
data for table 'AdventureWorks.Sales.CreditCard' (id 558) exported
[…]
```

Szenario

Vollständige Sicherung mit Komprimierung

Programmaufruf

```
newBackup 'action=backup,db=AdventureWorks,compress=yes'
```

Ausgabe

```
using backup id 3
generate script for database 'AdventureWorks'
script for database 'AdventureWorks' (id 1065) saved
getting database schema for 'AdventureWorks'
schema for object 'AdventureWorks' (id 1066) saved
schema for object 'HumanResources' (id 1067) saved
schema for object 'Person' (id 1068) saved
[…]
computing hashs and export data for tables in 'AdventureWorks'
hashs for table 'AdventureWorks.dbo.AWBuildVersion' (id 1086) computed
hashs for table 'AdventureWorks.Production.TransactionHistoryArchive'
(id 1087) computed
hashs for table 'AdventureWorks.Sales.CreditCard' (id 1090) computed
[…]
compress exported file for table 'AdventureWorks.dbo.AWBuildVersion' (id
1086)
data for table 'AdventureWorks.dbo.AWBuildVersion' (id 1086) exported
compress exported file for table
'AdventureWorks.Production.TransactionHistoryArchive' (id 1087)
data for table 'AdventureWorks.Production.TransactionHistoryArchive' (id
1087) exported
compress exported file for table 'AdventureWorks.Sales.CreditCard' (id
1090)
data for table 'AdventureWorks.Sales.CreditCard' (id 1090) exported
[…]
```

Szenario

Differenzielles Backup

Programmaufruf

```
newBackup 'action=backup,db=AdventureWorks,type=diff'
```

Ausgabe

```
using backup id 4
using backup with ID 3 as last master backup
generate script for database 'AdventureWorks'
script for database 'AdventureWorks' (id 1597) saved
getting database schema for 'AdventureWorks'
schema for object 'AdventureWorks' (id 1598) saved
schema for object 'HumanResources' (id 1599) saved
schema for object 'Person' (id 1600) saved
[…]
computing hashs and export data for tables in 'AdventureWorks'
hashs for table 'AdventureWorks.dbo.AWBuildVersion' (id 1618) computed
hashs for table 'AdventureWorks.Production.TransactionHistoryArchive'
```

```
(id 1619) computed
hashs for table 'AdventureWorks.Sales.CreditCard' (id 1622) computed
[…]
data for table 'AdventureWorks.dbo.AWBuildVersion' (id 1618) exported
data for table 'AdventureWorks.Production.TransactionHistoryArchive' (id
1619) exported
data for table 'AdventureWorks.Sales.CreditCard' (id 1622) exported[…]
```

Szenario

Inkrementelles Backup

Programmaufruf

```
newBackup 'action=backup,db=AdventureWorks,type=incr'
```

Ausgabe

```
using backup id 5
using backup with ID 3 as last master backup
generate script for database 'AdventureWorks'
script for database 'AdventureWorks' (id 2129) saved
getting database schema for 'AdventureWorks'
schema for object 'AdventureWorks' (id 2130) saved
schema for object 'HumanResources' (id 2131) saved
schema for object 'Person' (id 2132) saved
[…]
computing hashs and export data for tables in 'AdventureWorks'
hashs for table 'AdventureWorks.dbo.AWBuildVersion' (id 2150) computed
hashs for table 'AdventureWorks.Production.TransactionHistoryArchive'
(id 2151) computed
hashs for table 'AdventureWorks.Sales.CreditCard' (id 2154) computed
[…]
data for table 'AdventureWorks.dbo.AWBuildVersion' (id 2150) exported
data for table 'AdventureWorks.Production.TransactionHistoryArchive' (id
2151) exported
data for table 'AdventureWorks.Sales.CreditCard' (id 2154) exported
[…]
```

Szenario

Vollständige Sicherung bei vorhandener verschlüsselter Prozedur

Programmaufruf

```
newBackup 'action=backup,db=AdventureWorks'
```

Ausgabe

```
using backup id 6
generate script for database 'AdventureWorks'
script for database 'AdventureWorks' (id 2661) saved
getting database schema for 'AdventureWorks' failed
Fehler: 'sp_encrypted' ist eine verschlüsselte gespeicherte Prozedur.
Die Skripterstellung für verschlüsselte gespeicherte Prozeduren wird
nicht unterstützt.
```

Anhang – Durchgeführte Wiederherstellungsvorgänge

Szenario

Ausgabe der angefertigten Backups

Programmaufruf

```
newBackup 'action=list'
```

Ausgabe

```
BACKUP LIST FOR DB AdventureWorks

ID      DB             TYP    DEPENDS DATE                STATE
DESCRIPTION
-------------------------------------------------------------------------
----------------------
6       AdventureWorks full   6       08.04.2010 10:43:17 failed
full backup for database
5       AdventureWorks incr   3       08.04.2010 09:22:15 finished
incr backup for database
4       AdventureWorks diff   3       08.04.2010 09:11:00 finished
diff backup for database
3       AdventureWorks full   3       07.04.2010 14:06:19 finished
full backup for database
2       AdventureWorks full   2       07.04.2010 14:01:11 finished
full backup for database
1       AdventureWorks schema 1       07.04.2010 13:56:07 finished
schema for database
REMEMBER: backups with state 'not_finished' or 'failed' are not available
for restore or following diff/incr backups
```

Für alle folgenden Wiederherstellungsvorgänge wird das Backup mit der ID 3 verwendet.

Szenario

Wiederherstellung eines Datenbankschemas bei vorhandener Datenbank

Programmaufruf

```
newBackup 'action=schema_to_db,db=AdventureWorks,id=3'
```

Ausgabe

```
database 'AdventureWorks' exists, use force option to overwrite database
(CAUTION: all data will be lost)
```

Szenario

Wiederherstellung eines Datenbankschemas mit Überschreiben einer vorhandenen Datenbank

Programmaufruf

```
newBackup 'action=schema_to_db,db=AdventureWorks,id=3,force=yes'
```

Ausgabe

```
dropping existing database 'AdventureWorks'
getting database schema from backup ID 3
executing database schema script on database AdventureWorks
database AdventureWorks created
element 'AdventureWorks' (type: database, id 1066) created
element 'HumanResources' (type: schema, id 1067) created
element 'Person' (type: schema, id 1068) created
[…]
schema elements restored
```

Szenario

Wiederherstellung einer Datenbank

Programmaufruf

```
newBackup 'action=restore,db=AdventureWorks,id=3,force=yes'
```

Ausgabe

```
dropping existing database 'AdventureWorks'
getting database schema from backup ID 3
executing database schema script on database AdventureWorks
database AdventureWorks created
element 'AdventureWorks' (type: database, id 1066) created
element 'HumanResources' (type: schema, id 1067) created
element 'Person' (type: schema, id 1068) created
[…]
schema elements restored
disable all triggers and constraints temporary
handle fk FK_PurchaseOrderHeader_Vendor_VendorID
handle fk FK_VendorAddress_Address_AddressID
handle fk FK_VendorAddress_AddressType_AddressTypeID
[…]
compressed file for table 'dbo.AWBuildVersion' (id 1086) decompressed
data for table 'dbo.AWBuildVersion' (id 1086) in database
'AdventureWorks' imported
compressed file for table 'Production.TransactionHistoryArchive' (id
1087) decompressed
data for table 'Production.TransactionHistoryArchive' (id 1087) in
database 'AdventureWorks' imported
compressed file for table 'Sales.CreditCard' (id 1090) decompressed
data for table 'Sales.CreditCard' (id 1090) in database 'AdventureWorks'
imported
[…]
re-enable all disabled triggers and constraints
handle fk FK_PurchaseOrderHeader_Vendor_VendorID
handle fk FK_VendorAddress_Address_AddressID
handle fk FK_VendorAddress_AddressType_AddressTypeID
[…]
database 'AdventureWorks' restored
```

Für die Simulation einer Schlüsselverletzung müssen ein paar Eingriffe an der `AdventureWorks`-Datenbank vorgenommen werden. Dazu werden folgende Befehle ausgeführt:

```
USE AdventureWorks
GO
DROP TABLE HumanResources.EmployeeAddress
DELETE FROM Sales.CustomerAddress WHERE AddressID<=10
DELETE FROM Sales.SalesOrderHeader WHERE ShipToAddressID<=10
DELETE FROM Person.Address WHERE AddressID<=10
```

Damit wird eine Relationstabelle gelöscht und nachträglich werden Adressen gelöscht, welche in der gelöschten Relation verwendet wurden. Um die ID des wiederherzustellenden Elementes (hier: der Tabelle `HumanResources.EmployeeAddress`) zu erfahren, kann eine Elementliste abgerufen werden.

Szenario

Ausgabe der Elemente einer Sicherung

Programmaufruf

```
newBackup 'action=list_elements,id=3'
```

Ausgabe

```
ID      TYP                 OWNER           NAME
[…]
1194    trigger                             dVendor
1109    usertable           HumanResources  Employee
1110    usertable           HumanResources  EmployeeAddress
[…]
```

Szenario

Wiederherstellen einer Tabelle mit bestehenden Schlüsselbeziehungen

Programmaufruf

```
newBackup 'action=restore,db=AdventureWorks,id=3,element_id=1110'
```

Ausgabe

```
getting database schema from backup ID 3
executing database schema script on database
newBackupMeta_20100409135301513
database newBackupMeta_20100409135301513 created
element 'AdventureWorks' (type: database, id 1066) created
element 'HumanResources' (type: schema, id 1067) created
element 'Person' (type: schema, id 1068) created
```

```
[…]
schema elements restored
handle fk (tmp) FK_EmployeeAddress_Address_AddressID
handle fk (tmp) FK_EmployeeAddress_Employee_EmployeeID
handle trigger (tmp) uEmployeeAddress
compressed file for table 'HumanResources.EmployeeAddress' (id 1110)
decompressed
data for table 'HumanResources.EmployeeAddress' (id 1110) in database
'newBackupMeta_20100409135301513' imported
CAUTION: fk (tmp) FK_EmployeeAddress_Address_AddressID could not be
created, because rows in element violate constraint, use consistence
option to handle or ignore constraints
CAUTION: trigger (tmp) uEmployeeAddress could not be created, because
referred object 'HumanResources.EmployeeAddress' does not exist, use
consistence option to ignore constraints
```

Szenario

Wiederherstellen einer Tabelle mit bestehenden Schlüsselbeziehungen und aktivierter Behandlung

Programmaufruf

```
newBackup 'action=restore,db=AdventureWorks,id=3,element_id=1110,
consistence=handle'
```

Ausgabe

```
getting database schema from backup ID 3
executing database schema script on database
newBackupMeta_20100409140011960
database newBackupMeta_20100409140011960 created
element 'AdventureWorks' (type: database, id 1066) created
element 'HumanResources' (type: schema, id 1067) created
element 'Person' (type: schema, id 1068) created
[…]
schema elements restored
handle fk (tmp) FK_EmployeeAddress_Address_AddressID
handle fk (tmp) FK_EmployeeAddress_Employee_EmployeeID
handle trigger (tmp) uEmployeeAddress
compressed file for table 'HumanResources.EmployeeAddress' (id 1110)
decompressed
data for table 'HumanResources.EmployeeAddress' (id 1110) in database
'newBackupMeta_20100409140011960' imported
element 'EmployeeAddress' (type: usertable, id 1110) in database
'AdventureWorks' created
compressed file for table 'HumanResources.EmployeeAddress' (id 1110)
decompressed
data for table 'HumanResources.EmployeeAddress' (id 1110) in database
'AdventureWorks' imported
data will be deleted to create fk_tmp
FK_EmployeeAddress_Address_AddressID
creating triggers and constraints
element successfully restored
```

Glossar

API
Programmierschnittstelle („application programming interface“), welche Funktionen des Programms für andere Anwendungen zur Anbindung an das eigene System zur Verfügung stellt

Backup
Begriff für den Vorgang der Sicherung von Daten oder Dateien auf einem externen Medium zum Schutz vor Datenverlust

Commit
Vorgang, welcher eine offene Transaktion in SQL abschließt und temporär vorgenomme Änderungen und vorgehaltene Daten permanent festschreibt, insofern keine Inkonsistenzen auftreten

Disaster Recovery

Umfassender Begriff für alle Maßnahmen, die im Fall des Datenverlusts durchgeführt werden. Dies umfasst die Datenwiederherstellung, wie auch das Ersetzen defekter Hardware und Infrastruktur

DDL
Datenbeschreibungssprache („data definition language“), die das Anlegen, Modifizieren oder Löschen von Datenbankobjekten, also dem Schema, ermöglicht

Differenzielles Backup
Sicherungsvorgang, der nur die Unterschiede zur letzten vollständigen Sicherung dokumentiert

DML
Datenmanipulationssprache („data manipulation language“), die die möglichen Operationen in Bezug auf die Datenbasis (wie Einfügen, Modifizieren, Löschen, Abfragen) festlegt

DPW
„Database Publishing Wizard“ – Tool von Microsoft zum Auslesen des Datenbankschemas und der Dateninhalte auf einer grafischen Oberfläche oder der Kommandozeile

Flatfile
Datei, die im Klartext vorliegt und mit einem gängigen Texteditor bearbeitet werden kann; in

Verbindung mit einem Trennzeichen zur Separierung der Daten in der Datei entsteht eine CSV-Datei

Inkrementelles Backup
Sicherungsvorgang, der nur die Unterschiede zur letzten inkrementellen Sicherung beziehungsweise, falls es sich um die erste inkrementelle Sicherung handelt, der letzten vollständigen Sicherung dokumentiert

LSN
Protokollsequenznummer („log sequence number“), welche einen Eintrag im Transaktionsprotokoll von SQL Server eindeutig identifiziert

MSDN
Grundsätzlich kostenfreie Informationsplattform von Microsoft („Microsoft Developer Network“), auf Datenträger oder im Internet verfügbar; gibt Hilfestellung zu den Entwicklerprodukten von Microsoft

Normalformen
Die Normalformenlehre umfasst Regeln, die zur Optimierung einer Datenbank (allgemein eines Relationenmodells) dienen, um beispielsweise die Performanz zu erhöhen, Redundanzen und Datenanomalien zu verhindern und die Datenintegrität zu wahren

Object-Level-Recovery
Wiederherstellung eines einzelnen Objektes (einer Datenbank) aus seiner Vollsicherung

Prüfsumme
Berechnung eines (eindeutigen) Wertes – in der Regel eine Summe - über eine Zeichenkette, eine Datei oder einen Datenträger zur Gewährleistung der Datenintegrität

Recovery
Wiederherstellung von Daten, einer Datei oder eines Systems aus einer zuvor angefertigten Sicherung

Resultset
Ergebnis einer SQL-Abfrage in tabellenartiger Form

Rollback
Rückabwicklung einer SQL Transaktion; erfolgt im Fehlerfall oder explizit ausgelöst durch den Nutzer, dabei werden alle ausgeführten Aktionen seit Beginn einer Transaktion rückgängig gemacht

Rollforward
Anwenden von abgeschlossenen Transaktionen aus einer Protokollsicherung auf eine wiederhergestellte Datenbank

SMO
Bei „Server Management Object“ handelt es sich um eine .NET-Bibliothek von Microsoft zur Interaktion mit SQL Server aus anderen Programmiersprachen heraus

Snapshot
Abbild eines (Datenbank-) Systems in Form einer Kopie, welche einen lesenden Zugriff auf die konsistenten Daten ermöglicht und dabei im Hintergrund für andere Anwendungen einen unveränderten Zugriff gewährleistet

SQL Server-Agent
Systemdienst, welcher zeitgesteuert Prozesse in SQL Server ausführt

SQL-Injection
Ausnutzen einer Sicherheitslücke in einer Anwendung, um durch Manipulation der übergebenen Argumente an die SQL-Anweisung Zugriff auf die Datenbank zu erhalten

Stored Procedure
In SQL Server gespeicherte Abfolge von SQL-Anweisungen (Prozedur)

Transaktion
Ein Vorgang, der eine Menge von SQL-Anweisungen umfasst, die in einem „Zug“ abgearbeitet werden sollen

TSQL
„Transact SQL“ ist die Erweiterung des SQL Standards durch Microsoft und stellt die Sprache dar, in der alle Anweisungen in SQL Server abgearbeitet werden

TSQL-Backup
Name der in dieser Arbeit zu schaffenden Anwendung

UNC-Pfad
Standard zur Bezeichnung von (Speicher-) Ressourcen innerhalb eines Netzwerkes, um auf lokale oder entfernte Medien zugreifen zu können

Vertrauenswürdige Verbindung
Art der Verbindungsherstellung zu SQL Server über ein in Windows integriertes Benutzerkonto mit ausreichender Berechtigung

Literaturverzeichnis

[Adm08]

Administrator Technology GmbH, administrator.de. Netzwerke und Protokolle. *Grundlagen.* [Online] 2008. [Zitat vom: 21. 09 2009.] http://www3.administrator.de/index.php?content=85305.

[Bau06]

Bauder, Irene. *SQL Server 2005 für Administratoren.* München : Carl Hanser Verlag, 2006.

[Fae07]

Faeskorn-Woyke, Heide, et al. *Datenbanksysteme.* s.l. : Pearson Studium, 2007.

[Fre98]

Freeze, Wayne. *Die SQL-Referenz.* [Übers.] Gerhard Franken. 1. Auflage. Bonn : ITP, 1998.

[Hen04]

Henderson, Ken. *The Guru's Guide to SQL Server Architecture and Internals.* sixth printing. s.l. : Addison-Wesley, 2004.

[Inf05]

Informationstechnik, Bundesamt für Sicherheit in der. IT-Grundschutz-Kataloge. *M 4.169 Verwendung geeigneter Archivmedien.* [Online] 2005. [Zitat vom: 08. 01 2010.] https://www.bsi.bund.de/cln_183/ContentBSI/grundschutz/kataloge/m/m04/m04169.html.

[Inf06]

Informationstechnik, Bundesamt für Sicherheit in der. IT-Grundschutz-Kataloge. *B 2.5 Datenträgerarchiv.* [Online] 2006. [Zitat vom: 08. 01 2010.] https://www.bsi.bund.de/cln_183/ContentBSI/grundschutz/kataloge/baust/b02/b02005.html.

[Inf061]

Informationstechnik, Bundesamt für Sicherheit in der. IT-Grundschutz-Kataloge. *M 6.20 Geeignete Aufbewahrung der Backup-Datenträger.* [Online] 2006.

[Zitat vom: 08. 01 2010.]
https://www.bsi.bund.de/cln_183/ContentBSI/grundschutz/kataloge/m/m06/m06020.html.

[Inf08]

Informationstechnik, Bundesamt für Sicherheit in der. IT-Grundschutz-Kataloge. *B 5.7 Datenbanken.* [Online] 2008. [Zitat vom: 08. 01 2010.]
https://www.bsi.bund.de/cln_183/ContentBSI/grundschutz/kataloge/baust/b05/b05007.html.

[Inf09]

Informationstechnik, Bundesamt für Sicherheit in der. IT-Grundschutz-Kataloge. *G 4.13 Verlust gespeicherter Daten.* [Online] 2009. [Zitat vom: 24. 03 2010.]
https://www.bsi.bund.de/cln_165/ContentBSI/grundschutz/kataloge/g/g04/g04013.html.

[Inf091]

Informationstechnik, Bundesamt für Sicherheit in der. IT-Grundschutz-Kataloge. *B 1.12 Archivierung.* [Online] 2009. [Zitat vom: 24. 03 2010.]
https://www.bsi.bund.de/ContentBSI/grundschutz/kataloge/baust/b01/b01012.html.

[Kon07]

Konopasek, Klemens und Tiemeyer, Ernst. *SQL Server 2005 Der schnelle Einstieg.* 2., aktualisierte Auflage. s.l. : Addison-Wesley, 2007.

[Lei03]

Leitenbauer, Günter. *Datenbank Modellierung.* s.l. : Franzis', 2003.

[Mic07]

Corporation, Microsoft. SQL Server Hosting Toolkit. *Known Issues with Database Publishing Wizard 1.0.* [Online] 2007. [Zitat vom: 27. 01 2010.]
http://sqlhost.codeplex.com/wikipage?title=DPW%20Known%20Issues.

[Mic08]

Corporation, Microsoft. SQL Server Development Center. *Getting started with SQL Server.* [Online] 2008. [Zitat vom: 21. 09 2009.]
http://social.msdn.microsoft.com/Forums/en-US/sqlgetstarted/thread/49c25238-3cc6-44a6-a3cf-63acf4543a11.

[Mic081]

Corporation, Microsoft. SQL Server 2005-Onlinedokumentation. *Sicherungs- und Wiederherstellungs-APIs für unabhängige Softwareanbieter.* [Online] 2008. [Zitat vom: 21. 09 2009.] http://msdn.microsoft.com/de-de/library/ms189096(SQL.90).aspx.

[Mic0810]

Corporation, Microsoft. SQL Server 2005-Onlinedokumentation. *Grundlegendes zur Funktionsweise der Wiederherstellung von Sicherungen in SQL Server.* [Online] 2008. [Zitat vom: 20. 01 2010.] http://msdn.microsoft.com/de-de/library/ms191455%28SQL.90%29.aspx.

[Mic0811]

Corporation, Microsoft. SQL Server 2005-Onlinedokumentation. *RESTORE (Transact-SQL).* [Online] 2008. [Zitat vom: 20. 01 2010.] http://msdn.microsoft.com/de-de/library/ms186858%28SQL.90%29.aspx.

[Mic0812]

Corporation, Microsoft. SQL Server 2005-Onlinedokumentation. *Logische Architektur des Transaktionsprotokolls.* [Online] 2008. [Zitat vom: 21. 01 2010.] http://msdn.microsoft.com/de-de/library/ms180892%28SQL.90%29.aspx.

[Mic0813]

Corporation, Microsoft. SQL Server 2005-Onlinedokumentation. *CHECKSUM (Transact-SQL).* [Online] 2008. [Zitat vom: 27. 01 2010.] http://msdn.microsoft.com/de-de/library/ms189788%28SQL.90%29.aspx.

[Mic0814]

Corporation, Microsoft. SQL Server 2005-Onlinedokumentation. *xp_cmdshell (Transact-SQL).* [Online] 2008. [Zitat vom: 09. 04 2010.] http://msdn.microsoft.com/de-de/library/ms175046%28SQL.90%29.aspx.

[Mic082]

Corporation, Microsoft. SQL Server 2005-Onlinedokumentation. *Sichern und Wiederherstellen von Datenbanken in SQL Server.* [Online] 2008. [Zitat vom: 25. 08 2009.] http://msdn.microsoft.com/de-de/library/ms187048(SQL.90).aspx.

[Mic083]

Corporation, Microsoft. SQL Server 2005-Onlinedokumentation. *Vollständige Datenbanksicherungen.* [Online] 2008. [Zitat vom: 17. 09 2009.] http://msdn.microsoft.com/de-de/library/ms186289%28SQL.90%29.aspx.

[Mic084]

Corporation, Microsoft. SQL Server 2005-Onlinedokumentation. *bcp (Dienstprogramm).* [Online] 2008. [Zitat vom: 17. 09 2009.] http://msdn.microsoft.com/de-de/library/ms162802%28SQL.90%29.aspx.

[Mic085]

Corporation, Microsoft. SQL Server 2005-Onlinedokumentation. *Vorgehensweise: Erstellen eines Skripts (SQL Server Management Studio).* [Online] 2008. [Zitat vom: 17. 09 2009.] http://msdn.microsoft.com/de-de/library/ms178078%28SQL.90%29.aspx.

[Mic086]

Corporation, Microsoft. SQL Server 2005-Onlinedokumentation. *Übersicht über die Wiederherstellungsmodelle.* [Online] 2008. [Zitat vom: 25. 09 2009.] http://msdn.microsoft.com/de-de/library/ms189275%28SQL.90%29.aspx.

[Mic087]

Corporation, Microsoft. SQL Server 2005-Onlinedokumentation. *Sicherungen mit dem einfachen Wiederherstellungsmodell.* [Online] 2008. [Zitat vom: 25. 09 2009.] http://msdn.microsoft.com/de-de/library/ms191164%28SQL.90%29.aspx.

[Mic088]

Corporation, Microsoft. SQL Server 2005-Onlinedokumentation. *Sichern beim vollständigen Wiederherstellungsmodell.* [Online] 2008. [Zitat vom: 25. 09 2009.] http://msdn.microsoft.com/de-de/library/ms190217%28SQL.90%29.aspx.

[Mic089]

Corporation, Microsoft. SQL Server 2005-Onlinedokumentation. *BACKUP (Transact-SQL).* [Online] 2008. [Zitat vom: 20. 01 2010.] http://msdn.microsoft.com/de-de/library/ms186865%28SQL.90%29.aspx.

[Per03]

Pernul, Günther und Unland, Rainer. *Datenbanken im Unternehmen.* 2., korrigierte Auflage. Essen : Oldenbourg Verlag, 2003.

[Sch07]

Schubert, Prof. Dr.-Ing. Wilfried. *Vorlesung Softwaretechnik Grundlagen.* Mittweida : Hochschule Mittweida (FH), 2007.

[SQL10]

SQLServerCentral. xp_cmdshell and bcp problems. [Online] [Zitat vom: 22. 04 2010.] http://www.sqlservercentral.com/Forums/Topic795656-338-1.aspx.

[SQL101]

SQLServerCentral. Problem with global temporary tables in EXEC statement. [Online] [Zitat vom: 22. 04 2010.] http://www.sqlservercentral.com/Forums/Topic867831-338-1.aspx.

[War03]

Warner, Daniel. *SQL Das Praxishandbuch.* Oerlinghausen : Franzis', 2003.

[Wik09]

Wikipedia. Versionsverwaltung. *Hauptaufgaben eines Versionsverwaltungssystems.* [Online] 2009. [Zitat vom: 08. 01 2010.] http://de.wikipedia.org/wiki/Versionsverwaltung#Hauptaufgaben_eines_Versionsverwaltungssystems.

[Wik10]

Wikipedia. Generationenprinzip. [Online] 2010. [Zitat vom: 20. 01 2010.] http://de.wikipedia.org/wiki/Generationenprinzip.

[Woo07]

Woody, Buck. *SQL Server 2005 - Das Handbuch für Administratoren.* [Übers.] G&U Technische Dokumentation GmbH. s.l. : Addison-Wesley, 2007.

Selbständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

_____ _____

Ort, Datum Unterschrift